INTRODUCTION TO ANDROID

COURSE MATERIAL

Introduction to Android

The Open University of Sri Lanka
Department of Electrical and Computer Engineering

Copyright

This course has been developed as part of the collaborative advanced ICT course development project of the Commonwealth of Learning (COL). COL is an intergovernmental organisation created by Commonwealth Heads of Government to promote the development and sharing of open learning and distance education knowledge, resources and technologies.

The Open University of Sri Lanka (OUSL) is the premier Open and Distance learning institution in the country where students can pursue their studies through Open and Distance Learning (ODL) methodologies. Degrees awarded by OUSL are treated as equivalent to the degrees awarded by other national universities in Sri Lanka by the University Grants Commission of Sri Lanka.

For the avoidance of doubt, by applying this license the Commonwealth of Learning does not waive any privileges or immunities from claims that it may be entitled to assert, nor does the Commonwealth of Learning submit itself to the jurisdiction, courts, legal processes or laws of any jurisdiction. The ideas and opinions expressed in this publication are those of the author/s; they are not necessarily those of *Commonwealth of Learning* and do not commit the organisation.

The Open University of Sri Lanka
P. O. Box 10250,
Nawala,
Nugegoda,
Sri Lanka
Phone: +94 112881481
Fax: +94 112821285
Email: hdelect@ou.ac.lk
Website: www.ou.ac.lk

Commonwealth of Learning
4710 Kingsway, Suite 2500, Burnaby
V5H 4M2,
British Columbia,
Canada
Phone: +1 604 775 8200
Fax: +1 604 775 8210
Email: info@col.org
Website: www.col.org

Acknowledgements

Department of Electrical and Computer Engineering (ECE), The Open University of Sri Lanka (OUSL) wishes to thank those below for their contribution to this course material and accompanying Videos and Screencasts:

Chairperson of the Course team:

H.U.W. Ratnayake (Senior Lecturer, Dept. of ECE, OUSL)

Authors:

W.A.S.N. Perera (Lecturer, Dept. of ECE, OUSL)	Units 1 and 2
J. Nananyakkara (Lecturer, Dept. of ECE, OUSL)	Units 3 and 4
B.K. Werapitiya (Lecturer, Dept. of ECE, OUSL)	Units 5 and 6
S. Rajasingham (Lecturer, Dept. of ECE, OUSL)	Units 7 and 8
U.S. Premaratne (Lecturer, Dept. of ECE, OUSL)	Units 9 and 10
W.U. Erandaka (Lecturer, Dept. of Textile & Apparel Tech, OUSL)	Units 11 and 12
W.W.A.I.D. Wickramasinghe (Demonstrator, Dept. of ECE, OUSL)	Units 13 and 14
H.U.W. Ratnayake (Senior Lecturer, Dept. of ECE, OUSL)	Units 15 and 16

Content Editors:

C.W.S. Goonatilleke (Senior Software Engineer, WSO2 Inc)

G.S.N. Meedin (Lecturer, Dept. of ECE, OUSL)

Language Editor:

G.S.N. Meedin (Lecturer, Dept. of ECE, OUSL)

Reviewer:

D.G. U. Kulasekara (Senior Lecturer/Centre for Educational Technology & Media, OUSL)

Video Presenters:

U.S. Premartane (Lecturer, Dept. of ECE, OUSL)

S. Rajasingham (Lecturer, Dept. of ECE, OUSL)

Screen casters:

C.W.S. Goonatilleke (Senior Software Engineer, WSO2 Inc)

Contents

About this course material

This book 'Introduction to Android' has been produced by The Open University of Sri Lanka.

How this course material is structured

The course overview

The course overview gives you a general introduction to the course. Information contained in the course overview will help you determine:

- if the course is suitable for you
- what you will already need to know
- what you can expect from the course
- how much time you will need to invest to complete the course

The overview also provides guidance on:

- study skills
- where to get help
- course assignments and assessments
- activity icons
- units

We strongly recommend that you read the overview *carefully* before starting your study.

The course content

The course consists of many units. Each unit comprises:

- an introduction to the unit content.
- unit outcomes.
- new terminology.
- core content of the unit with a variety of learning activities.
- a unit summary.
- assignments and/or assessments, as applicable

Resources

For those interested in learning more on this subject, we provide you with a list of additional resources at the end of each unit; these may be books, articles or web sites.

Your comments

After completing Introduction to Android we would appreciate if you would take a few moments to give us your feedback on any aspect of this course. Your feedback might include comments on:

- course content and structure.
- course reading materials and resources.
- course assignments.
- course assessments.
- course duration.
- course support (assigned tutors, technical help, etc.)

Your constructive feedback will help us to improve and enhance this course.

Course overview

Welcome to Introduction to Android

This course will enable you with basic computing skills to develop mobile applications with Android operating system. Fundamental concepts of Android, theoretical and practical knowledge to develop an app incorporating multimedia and security, and performance issues of Android are discussed here. At the end of this course, you should be able to design and develop a mobile app to solve a real world problem using Android.

Video V-0: Introduction to Android

A series of videos have been developed as supporting materials for the lessons given in this book. This video, 'Course Overview', gives a general introduction to this series of videos produced. You can view this at URL: **https://tinyurl.com/ycsrmtho**

Introduction to Android —is this course for you?

This course is intended for people who aspire to become mobile application developers using Android operating system..

You should have basic ICT skills to use a computer and knowledge of a programming language such as Java to become a competent programmer in Android..

Course objectives

Objectives

The objectives of this course are:

- to introduce learners to basic concepts in Android
- to enable learners design mobile applications using fundametal concepts in Android
- to enable learner to acquire skills in programming with Android
- to introduce learners to performance and security issues that arise when developing mobile applications

Course outcomes

Outcomes

Upon completion of Introduction to Android you will be able to:

- *explain* the functionality of components in Android operating system and how the states of an Android activity change when running an Android mobile application
- *identify* the components and structures of Android development environment and explain how and when to apply these components to develop a working application
- *design* Android mobile applications using an Android development environment with existing mobile device features
- *develop* Android mobile applications using an Android development environment with existing mobile device features and deploy in Android market
- *analyse* the limitations of a mobile application for a given range of mobile devices
- *use* different testing tools and techniques to inspect and debug an Android mobile application

Timeframe

How long?

This is one-academic year course of 150 total learning hours

Face to face delivery would include 16 lectures of two hour duration, 11 laboratory exercises of two hours each, 14 videos of approximately 4 minutes.

Self study time is 5 hours per week for a 8-month academic year which include exams as well.

Study skills

As an adult learner your approach to learning will be different to that from your school days: you will choose what you want to study, you will have professional and/or personal motivation for doing so and you will most likely be fitting your study activities around other professional or domestic **responsibilities.**

Essentially you will be taking control of your learning environment. As a consequence, you will need to consider performance issues related to time management, goal setting, stress management, etc. Perhaps you will also need to reacquaint yourself in areas such as essay planning, coping with exams and using the web as a learning resource.

Your most significant considerations will be *time* and *space* i.e. the time you dedicate to your learning and the environment in which you engage in that learning.

We recommend that you take time now—before starting your self-study—to familiarize yourself with these issues. There are a number of excellent resources on the web. A few suggested links are:

- http://www.how-to-study.com/

  The "How to study" web site is dedicated to study skills resources.

- http://www.howtostudy.org/resources.php

  Another "How to study" web site with useful links to time management, efficient reading, questioning/listening/observing skills, getting the most out of doing ("hands-on" learning), memory building, tips for staying motivated, developing a learning plan.

The above links are our suggestions to start you on your way. At the time of writing these web links were active. If you want to look for more go to www.google.com and type "self-study basics", "self-study tips", "self-study skills" or similar.

Need help?

Help

This course is offered by the Department of Electrical and Computer Engineering of The Open University of Sri Lanka for registered students. If you need help regarding this course and if you are a registered student at OUSL, please contact:

Coordinator/ Mobile Application Development,
Department of Electrical and Computer Engineering,
Faculty of Engineering Technology,
The Open University of Sri Lanka

The coordinator can be contacted by email dist@ou.ac.lk

Lecturer's and Laboratory instructor's names and email addresses will be given with the activity schedule for the particular year.

Assignments

Assignments

There will be two assignments for this course which will be changed every year and will be given in the Learning Management System (LMS)

Assignments should be submitted to LMS on Android

Assignment submission deadline will be given together with the Assignments. Depending on the start and end of the academic year, dates will vary.

Assessments

Assessment

There will be:

- 2 Assignments,
- 2 online-quizzes,
- 3 lab assessments and
- one mini-project in this course which will be assessed as Continues Assessment components

All continous assesment components will take place before the Final Exam

Getting around this course material

Margin icons

While working through this course material you will notice the frequent use of margin icons. These icons serve to “signpost” a particular piece of text, a new task or change in activity; they have been included to help you to find your way around this course material.

A complete icon set is shown below. We suggest that you familiarize yourself with the icons and their meaning before starting your study.

Activity	Assessment	Assignment	Case study
Discussion	Objectives	Help	Note it!
Outcomes	Reading	Reflection	Study skills
Summary	Terminology	Time	Tip
Computer-Based Learning	Answers to Assessments	Video	Feedback

Unit 1

Introduction to Android

Introduction

The purpose of this study unit is to give you the first glimpse of the popular Android operating system for mobile devices and tablets. It is designed to empower mobile software developers to write innovative mobile applications. First part of this unit looks at the version history of the Android mobile operating system. Hence, you will be able to compare different mobile operating systems with unique features of Android.

The next part of the unit will help you to identify the devices that run Android as the Operating System with its open and customizable nature. Furthermore, at the end of this unit, you would be able to have an idea about the available categories of applications in Google Play.

One video material will be provided with this unit and you are expected to watch this and complete the relevant activities.

Upon completion of this unit you should be able to:

Outcomes

- describe how versioning and naming of Android Operating Systems has evolved.
- explain the unique features of Android OS comparing to existing mobile Operating Systems.
- identify the devices that run Android as the Operating System.
- identify the types of applications run on top of Android devices.

Terminology

IDE:	Integrated Development Enviornment to write, compile and and execute programs
app:	mobile application software
API:	Application Program Interface is a set of protocols and tools for building software
Kernel	essential core of an operating system

1.1 Android as a popular mobile platform

Android is an open-source operating system for mobile devices such as smart phones, smart watches, tablets, and other Android enabled platforms including Android TV and Android Auto. Android Auto is a smartphone projection standard developed by Google to allow mobile devices running the Android operating system (version 5.0 "Lollipop" and later) to be operated in automobiles through the dashboard's head unit. Android is a Linux based operating system.

> *"Android is the first truly open and comprehensive platform for mobile devices. It includes an operating system, user-interface and applications - all of the software to run a mobile phone, but without the proprietary obstacles that have hindered mobile innovation."*

-By Andy Rubin (Founder of Android Inc.)

In other words, Android can be defined as a system that includes an open source operating system, an open source development platform and devices that run the operating system and applications created for it.

In the next section, we will be discussing about the popularity of Android OS among people.

Rapid innovation

Android is continuously pushing the boundaries of hardware and software forward, to bring new capabilities to users and developers. For developers like you, the rapid evolution of Android technology lets you stay in front with powerful, differentiated applications.

Android gives you access to the latest technologies and innovations across a multitude of device form factors, chipset architectures, and price points. If you are not familiar with the terms, a form factor is the size, configuration, or physical arrangement of the device and a price point is a point on a scale of possible prices at which something might be marketed. The range of features further includes multicore processing and high-performance graphics, state of the art sensors, vibrant touch-screens, and emerging mobile technologies.

Powerful development framework

Android facilitate you with everything you need to build best-in-class app experiences. It gives you a single application model that lets you deploy your apps broadly to hundreds of millions of users across a wide range of devices from phones to tablets and beyond.

Android also gives you tools for creating apps while taking advantage of the hardware capabilities available on each device. It automatically adapts your User Interface (UI) to look its best on each device, and gives you control over UI on different device types. This is further discussed later in this material.

For example, you can create a single app binary that's optimized for both phone and tablet form factors. What is a single app binary and how a single app binary can be deployed is discussed in a later unit of this material. Android allows you to declare your UI in lightweight sets of Extensible Markup Language (XML) resources, one set for parts of the UI that are common to all form factors and other sets for optimizations specific to phones or tablets. How UI designing is done is further explained in a later unit of this material. At runtime, Android applies the correct resource sets based on its screen size, density, locale, and so on.

To help you develop efficiently, the Android Developer Tools offer a full Java Integrated development environment (IDE) with advanced features for developing, debugging, and packaging Android apps. Using the IDE, you can develop on any available Android device or create virtual devices that emulate any hardware configuration.

Video V-1: Course Overview

You may watch the video on "course overview" before moving further and answer the questions in Activity 1.1.

URL: https://tinyurl.com/ydhf7mf6

Activity

Activity 1.1

Check the Android version, i.e. Kernel version, in an Android phone.

What are the other devices you have seen having Android operating system?

1.2 History of Android

In October, 2003, four computer experts, Andy Rubin, Nick Sears, Rich Miner and Chris White founded a software development organization Android Inc. in Palo Alto, California, USA. They wanted to make a Linux based operating system that can work on digital cameras which can connect with computers. However, this plan was not as successful as they thought, so they focused on smart phones.

In August, 2005, Google purchased the Android Inc. and became the proprietor of the company. In November, 2007, Google disclosed a consortium of different mobile technology providers named Open Handset Alliance (OHA) that includes mobile hardware manufacturers (HTC, Motorola etc.), chipset manufacturers (Qualcomm, Texas Instruments etc.), and telecommunication service providers (T-Mobile etc.). There were 34 different companies in OHA consortium that agreed to provide a mobile device which does not belongs to a single company as iPhone from Apple. But for couple of years Google could not bring any mobile under the OHA consortium. In October, 2008, HTC brought first smart phone “HTC Dream” in the market which was commercially available. At the time when the first version of the Android was unveiled, only 35 Android apps were accessible. But today, millions of Android applications are available in the market.

Android has been released in many versions since its inception. Before commercialization, many internal alpha versions were released on the name of fictional robots (like Astro Boy, Bender, R2-D2 etc.). On November 5, 2007 Google released first beta version of Android whose Software Development Kit (SDK) was released on November 12, 2007. Since then, November 5th is considered as Android’s Birthday.

Version History

All versions of Android are released under a confectionary or sweet theme; i.e. names of the Android versions are the name of confectionary product in alphabetic order. It started with Android 1.5 "Cupcake"; versions 1.0 and 1.1 (API version 1 and 2) and they were not released under explicit code names.

API level is mainly the Android version used as an alternative to the Android version name (e.g. 3.0, 4.0, 4.4, etc.) where integer numbers are applied. This number keeps on increasing with each version, for e.g. Android 1.5 is API Level 3; Android 1.6 is API Level 4, and so on. Figure 1.1 shows the details of evolution of Android with product name, version name, release date and API level. The video *V-2 Evolution of Android* will also take you through the different phases of Android in an interesting manner.

Video V-2: Evolution of Android

Let us watch the vide on “Evolution of Android” before moving further and answer the questions in Activity 1.2.

URL: **https://tinyurl.com/ydhf7mf6**

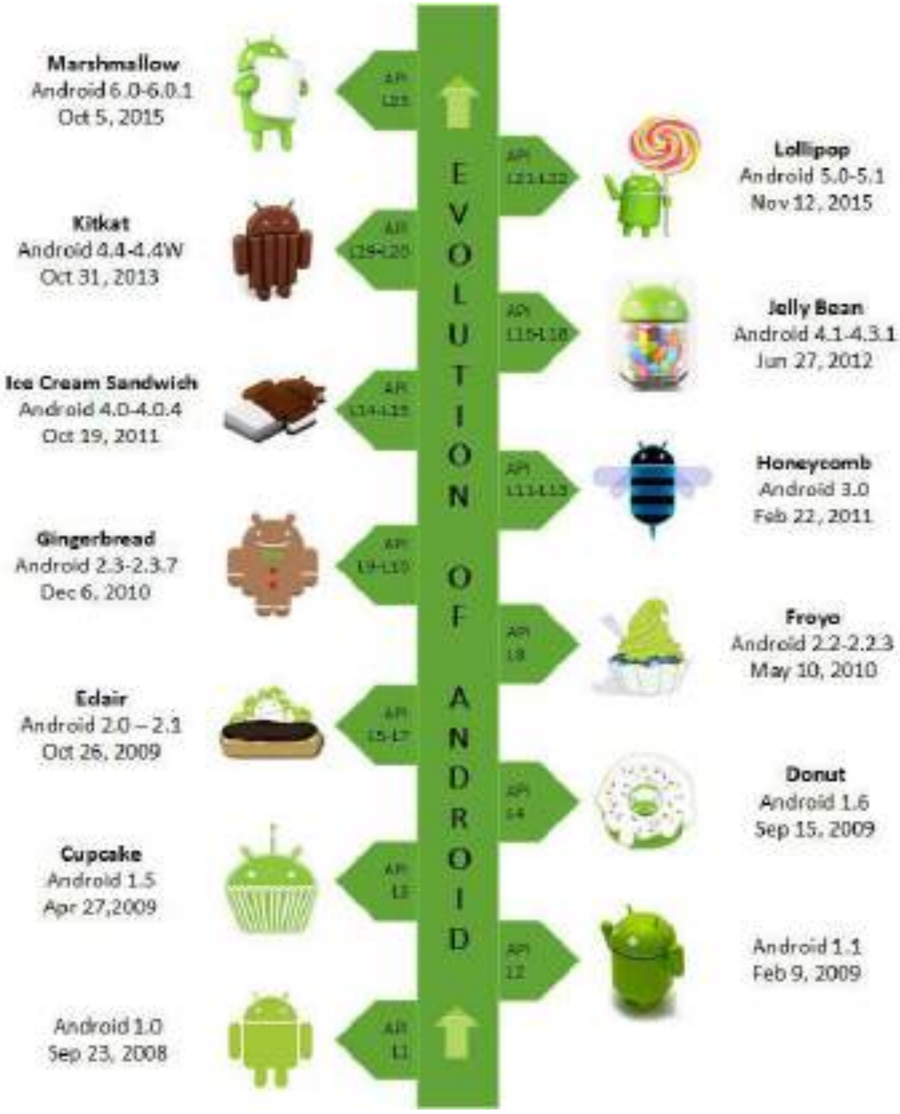


Figure 1.1: Android Version Evolution

(Source: Graphic Era Hill University, Dehradun, India. 2016, CC BY 4.0)

With their main features, different versions of Android are summarized in the following Table 1.1. You need not to worry if you are not familiar with the terms given under features. You can refer this table while reading the remaining units of this material as when required.

Table 1.1: Version history of Android

	Version Name	**Features**
1.	**API** **Level 1**	This was the first commercial version of Android implemented on the mobile device HTC Dream. There were many features such as Android Market, Web Browser, Digital Camera, Gmail, Google Maps, Google Search, Google Talk, Voice Dialer, Google Contacts, Google Calendar, Media Player, Wi-Fi and Bluetooth support.
	API **Level 2**	This version was internally known as “Petite Four”. This version resolved many bugs of the previous version and added additional features like saving attachment in messages, show and hide dial pad etc.
2.	**Cupcake**	This was the first version whose code name was on the name of a bakery product. Cupcake was based on Linux Kernel 2.6.27. It had the features like third party virtual keyboard, screen widgets, copy and paste in the browser, autorotation, upload facility on YouTube and Picasa, auto pairing for Bluetooth, video recording and playback in 3GP and MPEG-4 formats.
3.	**Donut**	This version was based on Linux kernel 2.6.29. Donut was having the capability of speech and gesture support, selecting the multiple photos for deletion, support for WVGA screen resolution, etc.
4.	**Éclair**	First version of Eclair (API Level 5) was having features like Microsoft Exchange email support, Bluetooth 2.1, HTML5, Google Map 3.1.2, Live wallpapers, optimized hardware speed, support for more resolutions, double tap zoom, camera features like flash support, digital zoom, white balance, colour effect, etc.
5.	**Froyo**	It was based on Linux kernel 2.6.32. and was having the features like JIT compilation, Android Cloud to Device Messaging (C2DM), push notification, USB tethering and Wi-Fi hotspot, support for alphanumeric passwords, installing apps in external memory, Adobe Flash support. It is also known as “Frozen Yogurt”.

6.	**Ginger-Bread**	It was based on Linux kernel 2.6.35. First version of Gingerbread (API Level 9) was having updated user interface, support for WXGA resolution, NFC and native code development; new download manager, concurrent garbage collection, native support for new sensors like Gyroscope and Barometer, etc.
7.	**Honeycomb**	It was the first tablet oriented Android update based on Linux kernel 2.6.36. “Motorola Xoom” tablet was the first device to run this update. This version was having the features like holographic interface, System Bar, Action Bar, soft navigation button at the bottom of the screen, two pane contact and email UI, support for multi-core processors, encryption of all user data, etc. Second and third versions of Honeycomb were Android 3.1 (API Level 12) and Android 3.2 (API Level 13) with some bug fixes and enhancements.
8.	**Ice Cream Sandwich**	It was based on Linux kernel 3.0.1. It was the last version that was supporting Adobe Flash Player. First version of Ice Cream Sandwich (API Level 14) was having the features like accessing app from lock screen, real time speech to text dictation, face unlock, built-in photo editor, Wi-Fi Direct, shut down app by swipe from recent menu, integrated screenshot capture, etc.
9.	**Jelly Bean**	First version of Jelly Bean (API Level 16) was based on Linux kernel 3.0.31. It was having “Buttery Smooth” UI and other advancements. Second version of Jelly Bean was Android 4.2 – 4.2.2 (API Level 17). It was based on Linux kernel 3.4.0 and with some features like Group Messaging etc. Third version of Jelly Bean was Android 4.3 – 4.3.1(API Level 18) with some features like 4K resolution support, native emoji support, Dial pad auto complete, etc.

10.	**Kitkat**	First version of Kitkat was Android 4.4 - 4.4.4(API Level 19). This was based on Linux kernel 3.10. This was optimized for larger range of devices than previous versions. Recommended RAM for Kitkat is 512 MB but it can run on minimum 340 MB of RAM. It was having different advanced features such as public API for developing text messaging clients; disable access to battery by third party, very elegant UI and much more. Second version of Kitkat was Android 4.4W (API Level 20) which was designed for wearable extensions like smart watch.
11.	**Lollipop**	First version of Lollipop was Android 5.0-5.0.2 (API Level 21). It was based on Linux kernel 3.16.1 and built around material design under project Volta to improve the battery life. It supports 64 bit CPU, trace based Just in Time(JIT) compilation, refreshed lock screen and notification tray; third party apps can modify the external storage; recently used apps remembered after restarting the device; audio I/O through USB, smart lock features and HD voice calls. Second version of Lollipop was Android 5.1 (API Level 22) with official support for multiple SIM cards, high definition voice calls, replicate the silent mode which was removed in API Level 21, native Wi-Fi calling etc.
12.	**Marshmallow**	Marshmallow is based on Linux kernel 3.18.10. It is released under the code name Android M. This is the latest updated version of Android which is having the features like native figure print reader, App standby feature, Doze mode, Now on Tap feature, USB Type-C support, MIDI support, 184 new emoji etc
13.	**Nougat**	The latest version of Android with system behaviors to save battery and memory. It brings new features for performance, productivity and security. There are several advantages such as multi-window UI, direct reply notifications etc.

Activity

Activity 1.2

Write the Android version name corresponding to following distinct features of different versions.

1. *First version whose name was on the name of a bakery product.*
2. *First version having the capability of speech and gesture support.*
3. *First version that is having USB tethering and Wi-Fi hotspot.*
4. *This version can work on 340 MB RAM.*
5. *This version is built around the API level 5.*
6. *This version enables multi-window UI.*

In the next section we will be discussing the features of Android.

1.3 Features of Android

Android is a powerful operating system with many supporting features for mobile application developers. Few of them are listed below in Table 1.2.

Table 1.2: Features of Android

Feature	**Description**
UI	Android provides a variety of pre-built UI components such as structured layout objects and UI controls to build the graphical user interface for your app. Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus with its own unique effects and animations.
Connectivity	Android supports connectivity technologies including for Wide Area Networks (WAN) like GSM, 3G, 4G and CDMA. Also it is supporting for Wi-Fi and Ethernet as Local Area Network (LAN) technologies. Apart from that Android has support for Bluetooth as a Personal Area Network (PAN). Also newer versions support for Near Field Communication (NFC).

Storage	Android provides several options for you to save persistent application data. The solution you choose depends on your specific needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires. Your data storage options are the following: Shared Preferences, Internal Storage, External Storage, SQLite Databases, Network Connection, Cloud storage
Media support	Media format support built into the Android platform. Audio - MP3, MIDI Images – JPEG, GIF, PNG, BMP Video - MPEG-4 SP (More media format support built into the Android platform will be discussed in future units)
Feature	**Description**
Messaging	Short Message Service (SMS) and Multimedia Messaging Service (MMS). Google Cloud Messaging (GCM) is also a part of Android Push Messaging services.
Web browser	The web browser available in Android is based on the open-source Blink (previously WebKit) layout engine, coupled with Chrome's V8 JavaScript engine. It supports both Hyper Text Markup Language (HTML5) and Cascading Style Sheets (CSS3).
Multi-touch	A multi-touch gesture is when multiple pointers (fingers) touch the screen at the same time. Android has native support for multi-touch.
Multi-tasking	Multitasking — running multiple tasks simultaneously. When an activity has been launched, the user can go to Home and launch a second activity without destroying the first activity. User can jump from one task to another and same time various applications can run

	simultaneously.
Resizable widgets	App Widgets are miniature application views that can be embedded in other applications (such as the Home screen) and receive periodic updates. These views are referred to as Widgets in the user interface, and you can publish one with an App Widget provider. Resizing allows users to adjust the height and/or the width of a widget within the constraints of the home panel placement grid. You can decide if your widget is freely resizable or if it is constrained to horizontal or vertical size changes.
Multi-Language	It is always a good idea to make the app localized and Android supports multiple languages.
Android Beam	Android Beam is a device-to-device data transfer tool that uses NFC and Bluetooth to send photos, videos, contact information, links to webpages, navigate directions and more from one device to another just by bumping them together. Android framework APIs supports these features.

Next, we will be comparing Android with other existing mobile operating systems.

1.4 Comparison of mobile Operating systems

Google's Android, Apple's iOS, Microsoft's Windows and BlackBerry Ltd's Blackberry are operating systems used primarily in mobile devices, such as smartphones and tablets. The popular mobile operating systems are very similar in some ways. Every OS supports some kind of mobile device management, but the way each OS supports is different and one of the unique features of Android is its source model. Android has its unique features when comparing with proprietary mobile operating systems. Table 1.3 shows a comparison of Android with one of the proprietary operating system, iOS.

Table 1.3: Comparison of mobile operating systems

	Android	iOS
Developer	Open Handset Alliance	Apple Inc.
Initial release	September 23, 2008	July 29, 2007
Source model	Open source	Closed, with open source components.
Available on	Many phones and tablets, LG, HTC, Samsung, Sony, Motorola, Nexus, Google Glasses	iPod Touch, iPhone, iPad, Apple TV
Messaging	Google Hangouts	iMessage
App store	Google Play	Apple app store
Video chat	Google Hangouts	Facetime
OS family	Linux	Unix-like, based on Darwin
Programmed in	C, C++, Java	C, C++, Objective-C, Swift
Internet browsing	Google Chrome	Mobile Safari
Voice commands	Yes	Siri
Latest stable release	Android 6.0.1 (Marshmallow), (October 2015)	9.3 (March 21, 2016)

	Android	iOS
Device manufacturer	Google, LG, Samsung, HTC, Sony, ASUS, Motorola, and many more	Apple Inc

1.5 Devices that run Android as the Operating System

Android occupies a large section of the global mobile market.

Here is a list of devices, which already run Android.

- Watches
- Smart glasses
- Home Appliances – E.g.: Refrigerators, washing machines, oven
- Cars
- Cameras
- Smart TVs
- Game consoles
- Home automation systems
- Robots

Activity

Activity 1.3

Give reasons for Android operating system becoming very popular.

1.6 Categories of Android applications

Google Play is the premier marketplace for selling and distributing Android apps. When you publish an app on Google Play, you reach the huge installed base of Android.

Using the Google Play Developer Console, you can choose a category for your apps. Users can browse for apps by category using a computer (play.google.com) and the Play Store app.

There are many Android applications in the market. Some of the top categories are:

- Music
- Multimedia
- News
- Sports

- Travel
- Weather
- Books
- Finance
- Social Media

Unit summary

The objective of this unit is to introduce Android and its capabilities. In the first part of this unit, we discussed about the history of Android with the names of the various versions of the Android Operating System (OS). Then we discussed the features provided by Android to create mobile applications and compared different operating systems available for mobile devices. Android is a powerful Operating System that supports many applications in Smart Phones.

Reference

https://developer.android.com/about/android.html (CC-BY)

Unit 2

Android Architecture

Introduction

This unit gives you an overview of the basic internal architecture of Android. The first part of this unit focuses on the Android architecture and the application framework, which developers can leverage in developing Android applications. Later it discusses different types of mobile applications with their pros and cons. This unit has one video that you need to watch and three activities to complete.

Upon completion of this unit you should be able to:

Outcomes

- illustrate the components of Android Operating System.
- differentiate mobile application development.
- identify the components provided by the application framework.

Terminology

Linux:	Name of the operating system that Android is built on
Dalvik:	Runtime and Virtual Machine used by the Android system for running Android applications
Native app:	apps that are fully programmed in the development environment specific to each operating system
ART:	(Android Run Time) is the successor of Dalvik and used in latest versions of Android

2.1 Android Architecture

The architecture of an Android system is a collection of different layers. Each layer has a specific role and set of functionality. Each layer provides the functionality to the layer above it.

As you can see in the Figure 2.1 that Android Architecture (also called

Software Stack) has the following layers:

- Linux Kernel
- Hardware Abstraction Layer (HAL)
- Native Libraries
- Android Run Time
- Android Application Framework
- Application Layer

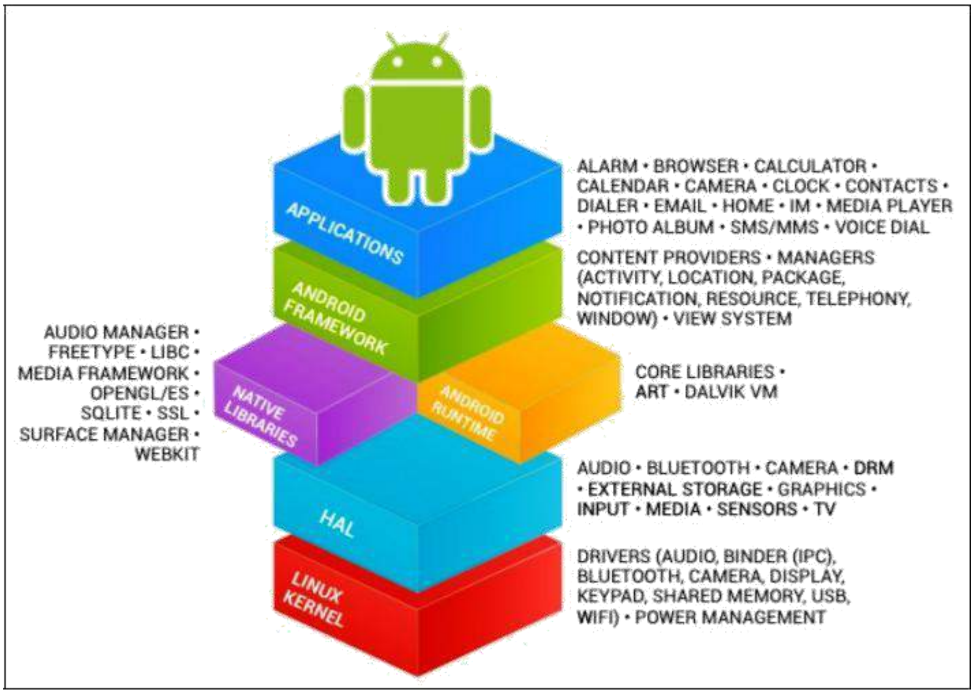

APPLICATIONS
ALARM • BROWSER • CALCULATOR • CALENDAR • CAMERA • CLOCK • CONTACTS • DIALER • EMAIL • HOME • IM • MEDIA PLAYER • PHOTO ALBUM • SMS/MMS • VOICE DIAL
ANDROID FRAMEWORK
CONTENT PROVIDERS • MANAGERS (ACTIVITY, LOCATION, PACKAGE, NOTIFICATION, RESOURCE, TELEPHONY, WINDOW) • VIEW SYSTEM
NATIVE LIBRARIES
AUDIO MANAGER • FREETYPE • LIBC • MEDIA FRAMEWORK • OPENGL/ES • SQLITE • SSL • SURFACE MANAGER • WEBKIT
ANDROID RUNTIME
CORE LIBRARIES • ART • DALVIK VM
HAL
AUDIO • BLUETOOTH • CAMERA • DRM • EXTERNAL STORAGE • GRAPHICS • INPUT • MEDIA • SENSORS • TV
LINUX KERNEL
DRIVERS (AUDIO, BINDER (IPC), BLUETOOTH, CAMERA, DISPLAY, KEYPAD, SHARED MEMORY, USB, WIFI) • POWER MANAGEMENT

Figure 2.1: Android System Architecture

(Source: https://source.android.com/source/index.html)

Let us discuss each layer one by one.

Linux Kernel

Android is designed on the top of Linux Kernel which is an open source operating system as explained in the previous unit.

Basic services like process management, memory management, security management, power management and providing hardware driver for different devices (like Bluetooth, WI-FI, Camera etc.) are managed by Linux Kernel as in Figure 2.2.

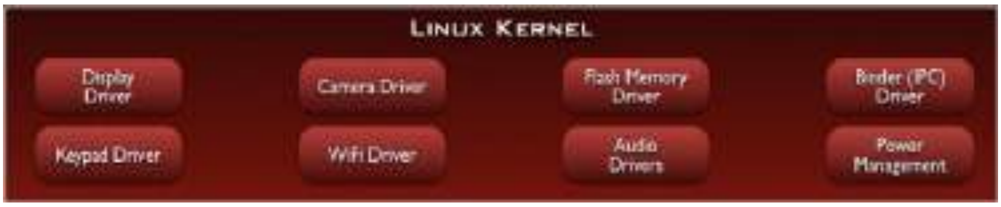


Figure 2.2: Linux kernel

(Source: http://androidfulcrum.blogspot.com/2014/02/what-is-android-architecture.html)

Linux Kernel never really interacts with the users and developers, but is at the heart of the whole system. Its importance stems from the fact that it provides the following functions in the Android system:

- Hardware abstraction
- Memory management programs
- Security settings
- Power management software
- Other hardware drivers (Drivers are programs that control hardware devices.)
- Support for shared libraries
- Network stack

In this section we have learned how Linux is involved with Android. Now you need to complete the given activity before reading further in this unit.

Activity

Activity 2.1

Explain the role of Linux Kernel in Android

Hardware Abstraction Layer (HAL)

The Hardware Abstraction Layer (HAL) provides an interface for hardware vendors to define and implement the drivers for specific hardware without affecting lower level features. HAL implementations are packaged into modules, you will find these as files with .so extension and loaded by the Android system at the appropriate time. Figure 2.3 shows the components of Hardware abstraction layer (HAL).

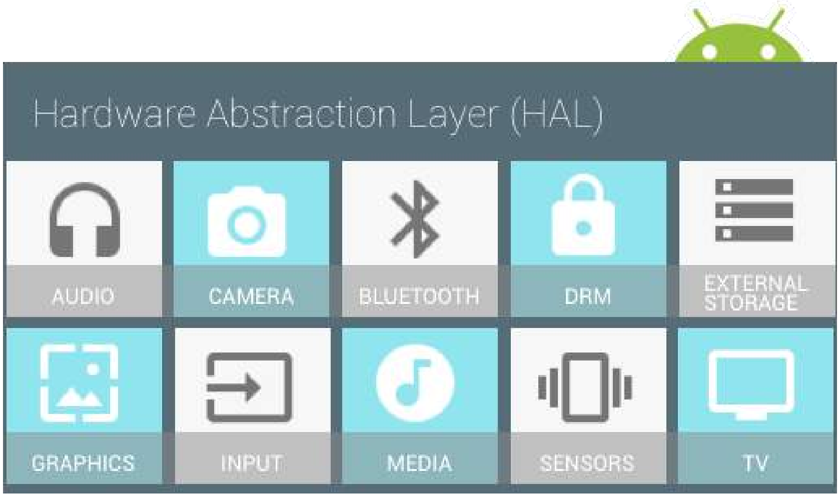


Figure 2.3: Hardware abstraction layer (HAL) components

(Source: https://source.android.com/source/index.html)

Native Libraries

Native libraries run over the HAL and it consist various C / C++ library like libc. It also includes following standard libraries:

- **Secure Sockets Layer (SSL):** It is responsible for Internet security.
- **Graphics Library:** OpenGL and SGL used to create 2D and 3D graphics.
- **WebKit:** It is open source web browser engine that gives the functionality to render the web content.
- **SQLite:** This is an open source relational database management system which is designed to be embedded in Android devices.
- **Media Library:** These libraries are used to play the audio/video media etc.

The following Figure 2.4 shows the Android library layer and its components.

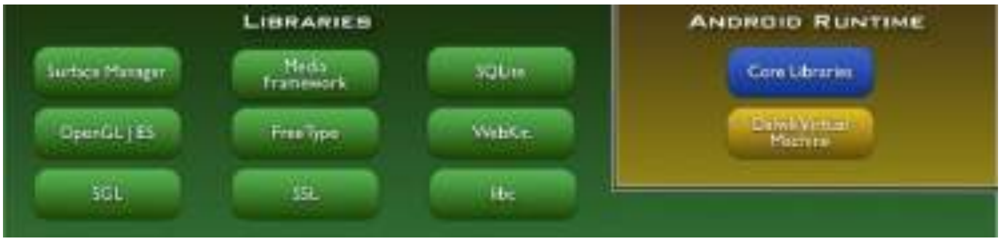


Figure 2.4: Android Libraries Layer

(Source: http://androidfulcrum.blogspot.com/2014/02/what-is-android-architecture.html)

Libraries carry a set of instructions to guide the device in handling different types of data. For instance, the playback and recording of various audio and video formats is guided by the Media Framework Library. This layer enables the device to handle different types of data with the use of these libraries.

Dalvik Virtual Machine (DVM)

It is a modified Java virtual machine (JVM) which is introduced for low end devices to run application objects efficiently. It is a register based virtual machine that is optimized to run multiple objects efficiently. It depends on the Linux kernel for efficiently execute the instances because memory management and thread management is part of the Linux kernel. DVM executes the Dalvik Executable Code (.dex), which is optimized to take least memory and processing resources.

Android Run Time

Android Runt time (ART) includes Dalvik Virtual Machine (DVM) and Core Libraries which help your apps to run on an Android mobile device.

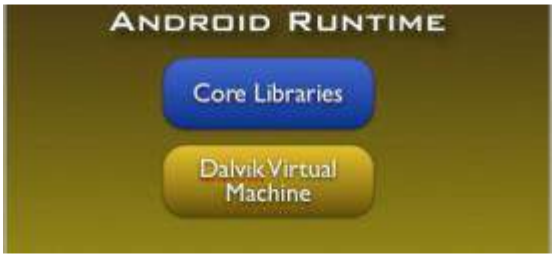


Figure 2.5: Android Runtime

(Source:http://androidfulcrum.blogspot.com/2014/02/what-is-android-architecture.html)

ART is the successor of Dalvik Virtual machine. ART is the managed runtime system that helps to run applications and system services. ART and its predecessor Dalvik were originally created specifically for the Android project. ART is compatible with DVM, so it helps to run Dalvik Executable codes. For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the Android Runtime (ART)ART has the following features:

- Ahead-of-time (AOT) compilation
- Improved garbage collection
- Improvements in development and debugging

Core Libraries

Core libraries are the collection of Android specific core java libraries. You will be using these libraries when writing your Android applications in later units.

Video V3: Android Architecture

What you have read upto now regarding Android Architecture is presented in this video. After watching the video you may attempt bActivity 2.2.

URL: https://tinyurl.com/ya7t24et

Activity

Activity 2.2

Describe the use of Dalvik Virtual Machine and Android Runtime in Android operating systems.

Android Application Framework

Application framework contains the classes used to create an Android application. This behaves as an abstraction layer for hardware access. It also manages application resources and user interface. Content provider, activity manager, fragment manager, telephony manager, location manager, package manager, notification manager and view system are the parts of Android Application Framework.

Application Layer

Figure 2.6 shows the application layer which is the top layer of Android architecture. Every application (e.g. Contacts, Browsers, etc.), whether it is a native application or a third party application, runs in Application layer. Preinstalled applications provided by the vendors are called native apps and applications developed by another developer are called third party applications. In application layer, third party apps can replace the native apps. This is the beauty of Android.

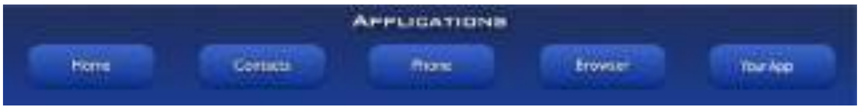


Figure 2.6: Android application layer

(Source: http://androidfulcrum.blogspot.com/2014/02/what-is-android-architecture.html)

2.2 Types of mobile applications

It is obvious that mobile technologies have been evolving rapidly in recent years, with a huge amount of creativity on mobile application development. As a result, there are more prominent needs than at any other time for a mobile application client platform which can not only deliver mobile-optimized user experience, but also support the increasingly complex business logic that the application must support.

It is important to consider various factors related to the purpose of the developing the app when deciding between which types of app you should build. Therefore, it is important to check your current priorities and where you want to be in the future before selecting the app. The quality of the user experience you need your app to have, the complexity of the features you need for your app to work and the budget, helps you choose which approach is best.

Today, we often speak of three types of mobile applications according to how they are developed: native, web and hybrid. Further explanations on these types are given below.

Native apps

Those that are fully programmed in the development environment specific to each operating system.

Native applications can leverage the full array of features and functions available through the mobile device’s core operating system. Generally, they are faster and offer a significantly usable interface than the others.

Advantages:

- Smooth performance
- Good user experience
- App icon available on the device
- Can receive push notifications
- Runs inside the operating system
- Can use the platform APIs

Disadvantages:

- Developers need to know each of the platforms languages
- Source code only works on the targeted platform
- Slower to market due to multiple source codes

Web apps

Fully developed in HTML 5, Mobile Web apps offer an attractive option for companies that do not want to invest in building native applications across four different mobile platforms. Whether getting a new application up and running or maintaining or updating an existing mobile solution with Mobile Web Apps is simple and inexpensive. Better yet, HTML5-driven mobile Web apps are cross-platform compatible and, more secure than native applications (given that very little data is stored locally on the native device.)

Advantages:

- Cross platform
- Single code base
- Fast to production
- Lower development cost

Disadvantages:

- Sluggish performance
- Require loading
- Network connection required
- Not available in the app stores
- Extremely limited API access Lives in the browser

Hybrid apps

Apps developed partly with the native development environment and partly in WEB language (HTML 5).

Today, technology changes so rapidly that most businesses require immense flexibility and scalability to adapt content, design and even application architecture. By deploying applications that rely on a robust combination of HTML5 Web technologies and native operating system features, you preserve a large degree of control over the content and design of the solutions we build for mobile platforms.

Many companies find that this hybrid development process empowers them to perform fast, easy, on-demand updates, without losing the inherent advantages that come from hosting a solution in the iTunes Apps Store or the Android Marketplace.

Advantages:

- Single source code
- Access to all platforms

- Less time for deployment
- Available in app store
- Has application icon on the device

Disadvantages:

- Dependent on such as phone gap
- Middleware may be slow to update
- More bug prone
- Some bug fixes need middleware updates
- Some bug fixes are outside of your control
- Slower performance
- More issues from device fragmentation

Each has its positives and negatives that can and should influence the decision when making a choice for development. Which is most appropriate will vary based on your particular requirements. The Table 2.1 shows a summary of the features discussed in each type.

Table 2.1: Hybrid vs. Native vs. Mobile Web

Feature	**Native**	**HTMAL5**	**Hybrid**
Graphics	Native APIs	HTML, Canvas, SVG	HTML, Canvas, SVG
App performance	Fast	Moderate	Moderate
Distribution	App Store/Market	Web	App Store/Market
Native look and feel	Native	Emulated	Emulated
Camera	Yes	No	Yes
Push Notifications	Yes	No	Yes
File upload	Yes	Yes	Yes
Contacts, calendar	Yes	No	Yes
Connectivity	Online and offline	Mostly online	Online and offline
Development skills needed	XML, Java	HTML5, CSS, Javascript	HTML5, CSS, Javascript
Geolocation	Yes	Yes	Yes

Activity

Activity 2.3

Differentiate native, web, and hybrid mobile apps by stating the advantages, disadvantages and special features.

2.3 Application Fundamentals

Android apps are written in the Java programming language. The Android SDK tools compile your code along with any data and resource files into an APK: an Android package, which is an archive file with an .apk suffix. One APK file contains all the contents of an Android app and this is the file that Android-powered devices use to install the app.

- Once installed on a device, each Android app lives in its own security sandbox
- The Android operating system is a multi-user Linux system in which each app is a different user.
- By default, the system assigns each app a unique Linux user ID (the ID is used only by the system and is unknown to the app). The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them.
- Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
- By default, every app runs in its own Linux process. Android starts the process when any of the app's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other apps.

In this way, the Android system implements the principle of least privilege. That is, each app, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an app cannot access parts of the system for which it is not given permission.

However, there are ways for an app to share data with other apps and for an app to access system services:

- It's possible to arrange for two apps to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, apps with the same user ID can also arrange to run in the same Linux process and share the same VM (the apps must also be signed with the same certificate).
- An app can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more. The user has to explicitly grant these permissions.

That covers the basics regarding how an Android app exists within the system.

The rest of this unit introduces you to:

- The core framework components that define your app.
- The manifest file in which you declare components and required device features for your app.
- Resources that are separate from the app code and allow your app to gracefully optimize its behaviour for a variety of device configurations.

App components are the essential building blocks of an Android app. Each component is a different point through which the system can enter your app.

Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your app's overall behaviour.

There are four different types of app components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed. Here are the four types of application components. These four components are further explained in unit 5.

Activities

An activity represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities (if the email app allows it). For example, a camera app can start the activity in the email app that composes new mail, in order for the user to share a picture.

An activity is implemented as a subclass of Activity and you can learn more about activity lifecycle in the next unit.

Services

A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

A service is implemented as a subclass of Service and you can learn more about it in the Services developer guide.

Content providers

A content provider manages a shared set of app data. You can store the data in the file system, SQLite database, on the web, or any other persistent storage location your app can access. Through the content provider, other apps can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any app with the proper permissions can query part of the content provider (such as contact data) to read and write information about a particular person.

Content providers are also useful for reading and writing data that is private to your app and not shared. For example, the Note Pad sample app uses a content provider to save notes.

A content provider is implemented as a subclass of ContentProvider and must implement a standard set of APIs that enable other apps to perform transactions. For more information, see the Content Providers developer guide.

Broadcast receivers

A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

Unit summary

The objective of this study unit was to introduce the architecture of Android. In the first part of this study unit, we discuss about the different types of layers of Android operating system and the role of each layer with functionalities. Further, it discussed about the three types of mobile applications: native, web and hybrid. The rest of the unit discussed about the fundamental components of an Android application.

References

https://source.android.com/devices/ (CC-BY)
http://androidfulcrum.blogspot.com/2014/02/what-is-android-architecture.html (CC-BY)
https://source.android.com/source/index.html (CC-BY)

Unit 3

Activity lifecycle

Introduction

In this unit you will be exploring the Activity Lifecycle of an Android application. After an introduction to *Activity*, *Activity life cycle* and *life cycle methods* are discussed. By watching the screen cast developed for this particular unit, you will get a better understanding of how activity lifecycle works. The demonstration on how to determine the inter-process dependencies at runtime will also be further explained in the screencast.

Upon completion of this unit you will be able to:

Outcomes

- Sketch the activity life cycle diagram and identify the components.
- List the status of the activity life cycle and describe each status related to the working mobile application.
- Explain the process of the activity life cycle, foreground, visible, background and empty processes.

Terminology

activity:	Activity is an application component that provides a screen with which users can interact
life cycle:	journey of an Activity passing through different stages of life

3.1 What is an Activity in Android?

An Activity in Android is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map as explained in the previous unit. Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows.

When you open an application, the very first screen that appears in front of you, is called a default Activity (or Main Activity). There can be more than one loosely coupled Activities in your application. Generally, as the complexity of an application increases, the need of number of Activities increases proportionally.

Activity

Activity 3.1

Give an example of an Activity of an Android application

Now you have an idea about what an Activity is. Let us look into more details about how Activities are interrelated. An Activity can start another Activity to perform some actions. If it does so then system stops the current Activity and preserves it in a stack called activity stack which will be discussed in the next section.

Activity Stack

Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time. Each activity can then start another activity in order to perform different actions. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack called activity stack or "back stack". It is illustrated in the Figure 3.1.

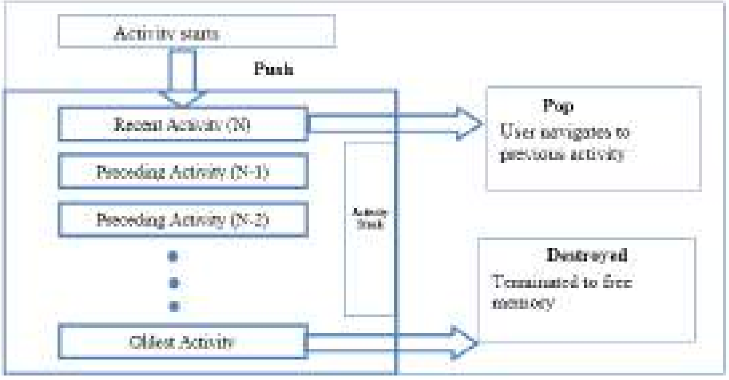


Figure 3.1: Activity Stack

When a new activity starts, it is pushed into the back stack and takes user focus. The back stack abides to the basic "last in, first out" stack mechanism, so that when the user is done with the current activity and presses the Back button, it is popped from the stack and the previous activity resumes. The old activities are destroyed by the OS to free up memory. But activity stack keeps activity reference and re-launch an activity when needed.

Activity

Activity 3.2

Draw and briefly explain, how each new activity in a task adds an item to the back stack by referring to the following link.

https://developer.android.com/guide/components/tasks-and-back-stack.html

Hint: You can consider three activities at a time and progress between them

Once an activity is launched, it goes through a lifecycle called Activity Lifecycle; a term that refers to the steps the activity progresses through as the user (and operating system) interacts with it.

3.2 What is an Activity Lifecycle?

From its creation to its conclusion, an Activity goes through many stages of its life such as start, pause, resume, stop, etc. This journey of Activity passing through different stages of life called lifecycle of the Activity. Every stage of the Activity lifecycle has a specific method associated to it, called call-back method. When an Activity stops and another starts, it means a call-back method is called for each Activity. The lifecycle of an Activity is managed by the Android run time system.

Generally, an Activity can remain in following four states:

- If an activity is in the foreground of the screen (at the top of the stack), it is *active* or *running*.

If an activity has lost focus but is still visible (that is, a new non-full-sized or transparent activity has focus on top of your activity), it is paused. A paused activity is completely alive (it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations.

- If an activity is completely obscured by another activity, it is *stopped*. It still retains all state and member information, however, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere.

- If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.
- Figures 3.2 depicts the state diagrams to represent the different stages of the Activity lifecycle with different call-back methods.

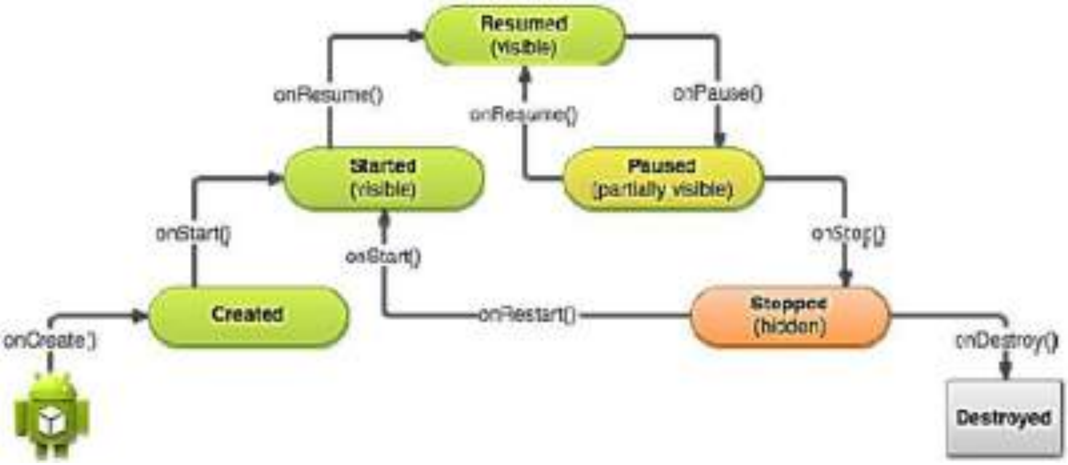


Figure 3.2: A simplified illustration of Activity Lifecycle

(Source: http://developer.android.com/training/basics/activity-lifecycle/starting.html)

Video-V4: Android Application Fundamentals

Let us watch this video and understand the android activity life cycle that moves to each state and respond to different call-back methods. This video will give you a detailed explanation of an Activity, Activity lifecycle, use of back-stack and the different call-back methods of Activity Lifecycle.

URL: **https://tinyurl.com/y7czgm5f**

Figure 3.3 illustrates the loops and the paths of an activity that might take place between states.

The rectangles represent the call-back methods you can implement to perform operations when the activity transitions between states. You can see that when a process is killed and the user navigates back to onCreate() method.

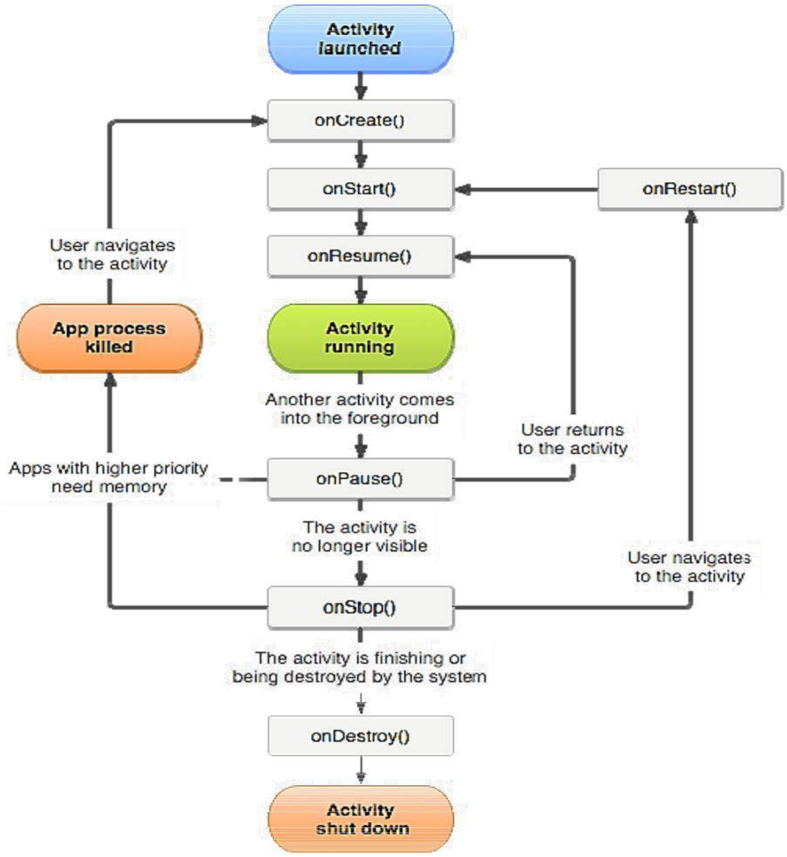

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 3.3: Activity Lifecycle

(Source:http://developer.android.com/guide/components/activities.html)

Now you have learnt the Activity Lifecycle and its states. Next, we are going to discuss how these states will be implemented as methods.

Activity Lifecycle Methods

The Android system defines a lifecycle for activities via predefined lifecycle methods. The following Table 3.1 is showing each lifecycle call-back method with details such as name of the method, description of the method, what method is called after the specified method, method is killable or not, etc.

Table 3.1: A summary of the activity lifecycle's call-back methods

Methods	**Description**	**Killable after?**	**Next**
onCreate()	Called when the activity is first created. This is where you should do all of your normal static set up such as create views, bind data to lists, and so on.	No	onStart()
onRestart()	Called after the activity has been stopped, just prior to it being started again.	No	onStart()
onStart()	Called just before the activity becomes visible to the user.	No	onResume() or onStop()
onResume()	Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack.	No	onPause()
onPause()	Called when the system is about to start resuming another activity.	Yes	onResume() or onStop()
onStop()	Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it.	Yes	onStart() or onDestroy()
onDestroy()	Called before the activity is destroyed. This is the final call that the activity will receive	Yes	Nothing

(Source:http://developer.android.com/guide/components/activities.html)

Within the lifecycle call-back methods, you can declare how your activity behaves when the user leaves and re-enters the activity. For example, if you're building a streaming video player, you might pause the video and terminate the network connection when the user switches to another app. When the user returns, you can reconnect to the network and allow the user to resume the video from the same spot.

To get the detailed information of working of an Activity, pursue the following link:

http://developer.android.com/guide/components/activities.html

With the understating of the Activities, Activity Lifecycle and call-back methods, we can discuss how these Activities are managed in the Android application.

Managing Activities in the application

All Android applications started by the user are remained in memory, which makes restarting applications faster. But in reality the available memory on an Android device is limited. To manage these limited resources the Android system is allowed to terminate running processes or recycling Android components.

As stated earlier Android applications run within instances of the Dalvik virtual machine with each virtual machine being viewed by the operating system as a separate process. If the system identifies that resources on the device are reaching capacity it will take steps to terminate processes to free up its memory.

If the Android system needs to free up resources it follows logic. Every process gets a priority. If the Android system needs to terminate processes it follows the priority system. You will learn the Android process states and priority system in next section.

3.3 What are the Android process states?

When deciding as to which process to terminate in order to free up memory, the system consider both the priority and state of all currently running processes. It is considered by Google as an important hierarchy. Processes are then terminated starting with the lowest priority and working up the hierarchy until sufficient resources have been liberated for the system to function. As outlined in Figure 3.4, a process can be in one of the following five states at any given time of priority.

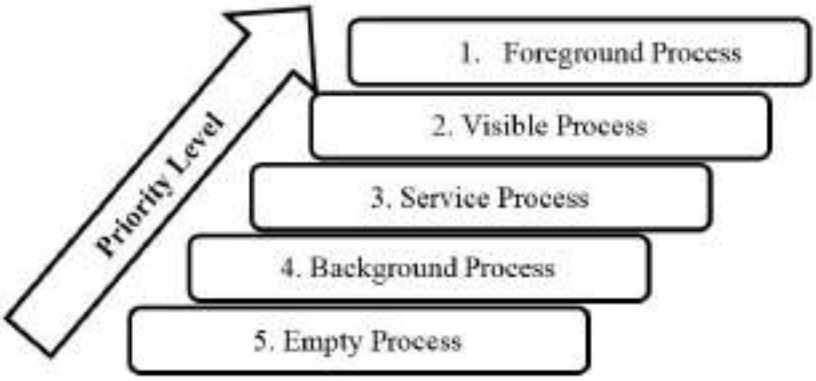


Figure: 3.4: Android process states and priority levels

Let us explain the priority levels more in details below.

Foreground Process

These processes are assigned the highest level of priority. It is one that is required for what the user is currently doing. Various application components can cause its containing process to be considered foreground in different ways. A process is considered to be in the foreground if any of the following conditions hold:

- Hosts an activity with which the user is currently interacting.
- Hosts a Service connected to the activity with which the user is interacting.
- Hosts a Service that has indicated, via a call to startForeground(), that termination would be disruptive to the user experience.
- Hosts a Service executing either its onCreate(), onResume() or onStart() callbacks.
- Hosts a Broadcast Receiver that is currently executing its onReceive() method.

Visible Process

It is a process containing an activity that is visible to the user but is not the activity with which the user is interacting. Such a process is considered extremely important and will not be killed unless doing so is required to keep all foreground processes running. This may occur, for example, if the foreground Activity is displayed as a dialog that allows the previous Activity to be seen behind it.

Service Process

A process that contain a Service that has already been started and is currently executing is called a service process. However, these processes are not directly visible to the user. These processes are generally doing things that the user cares about such as background mp3 playback or background network data upload or download. The system will always keep such processes running unless there is not enough memory to retain all foreground and visible processes.

Background Process

It is a process that contains one or more activities that are not currently visible to the user, and does not host a Service that qualifies for Service Process status. These processes have no direct impact on the user experience. Provided they implement their Activity life-cycle correctly, the system can kill such processes at any time to reclaim memory for one of the three previous processes types.

Empty Process

Empty processes no longer contain any active application component. They are held in memory ready to serve as hosts for newly launched applications. Such processes are, obviously, considered to be the lowest priority and are the first to be killed to free up resources.

Activity

Activity 3.3

Select True/False statements
You can select A) True or B) False for the following statements

1. There is no guarantee that an activity will be stopped prior to being destroyed.
2. During an activity lifecycle, onStart() is the first callback method invoked by the system.
3. Finish() method is used to close an activity.
4. Once the onStop() method is called, the activity is no longer visible.
5. When onPause() method is called in an activity, another activity gets into the foreground state.

Hint check your answers with Answer guide at the end of this unit.

Now you have learnt how activities are managed. The situation of deciding the highest priority process is sometimes complex than outlined in the previous section since that processes can often be inter-dependent. Inter-Process Dependencies are explained further in the following section.

Inter-Process Dependencies

Android system will also take into consideration that whether the process is in some way serving another process of higher priority. Likewise, when deciding as to the priority of a process the system consider inter-dependencies.

For instance, consider a service process acting as the content provider for a foreground process. The Android documentation states that a process can never be ranked lower than another process if it is currently serving a foreground process. Thus, the systems manage the activities and memory facilitating fast running applications.

(Source: https://developer.android.com/training/articles/memory.html)

Unit summary

Mobile applications are common in day to day life. For instance, we can recall some of them as Calculator app, Date and activity schedule app, Map finding etc. All these application components provides a screen for user to interact and perform something with the mobile device. Those components are made of multiple activities providing various user interfaces. In this unit we described what an activity is and the activity lifecycle in a mobile application.

Activity lifecycle has few states and those states depend on the process states of it changes based on pre-defined priority levels. Moreover, activity lifecycle has methods which can be called upon wherever necessary. The rest of unit discussed the basics of how to build and use an activity.

Unit 4

Android Development Environment

Introduction

In this unit you will be able to get familiar with available Android development platforms. You need to watch the video and install the required tools to start developing your first Android application.

Upon completion of this unit you should be able to:

Outcomes

- explain development platforms and distinguish each against their features and capabilities.
- describe the background, and platform versions, system features, Android tools for the development environment.
- configure the Android development environment in a computer.

Terminology

command line:	commands entered as inputs without IDE
SDK:	Software Development Kit
JDK:	Java Development Kit
Android Studio:	Official Android platform to develop Android apps

4.1 Reasons for Android Development

Today, mobile telephones have fundamentally changed the way of people interact. It is evident that mobile applications will be the future of handheld devices, Television and Automobile. Moreover, developers have started embedding Android in home appliances and other devices.

There are many reasons for the popularity of Android apps, such as:

- Google provides one window solution, as Play Store, to upload and download the application either free or with minimal charges. For uploading and distributing the app, developers have no need of any approval of someone.

- Developer is the owner of his / her app and has the total control on product. However, Google has rights to unpublish any Android application in play store, if it is not complying with Google’s licenses. For instance, if application contains malicious code or violate license, Google has right to unpublish the application.
- Android has open source operating system, open source software development kit (SDK) and good documentation.
- Android applications are not limited to mobile devices (Phones & Tabs), but can be run on TVs, wearable devices, vehicles and even refrigerators.
- (Source: First Thrust Towards Android, Android Programming, Course Material for Open Distance Learning, Commonwealth of Learning 2016)

4.2 Android Development Platforms, Features and Tools

In unit 2, you have learned Android architecture and major components of the Android platform. Let’s look at the Android platform and the features they are providing.

Android Studio is the official IDE for Android development, and with a single download it includes everything you need to begin developing Android apps as you can see below

- IntelliJ IDE + Android Studio plugin
- Android SDK Tools
- Android Platform-tools
- A version of the Android platform
- Android Emulator with an Android system image including Google Play Services

Android Studio provides tools for building apps on every type of Android device. Code editing, debugging, performance tooling, a flexible build system, and an instant build or deploy system are included in Android studio. Let's see what are the systems requirements to install Android studio in different operating systems.

System Requirements

System requirements for Windows, Mac OS and Linux are given below.

Windows - Microsoft® Windows® 7/8/10 (32- or 64-bit)

- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- 2 GB of available disk space minimum,

- 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution
- For accelerated emulator: 64-bit operating system and Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality

Mac - Mac® OS X® 10.10 (Yosemite) or higher, up to 10.12 (macOS Sierra)

- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- 2 GB of available disk space minimum,
- 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

Linux - GNOME or KDE desktop

- Tested on Ubuntu® 12.04, Precise Pangolin (64-bit distribution capable of running 32-bit applications)
- 64-bit distribution capable of running 32-bit applications
- GNU C Library (glibc) 2.19 or later
- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- 2 GB of available disk space minimum,
- 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

For accelerated emulator: Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality, or AMD processor with support for AMD Virtualization™ (AMD-V™).

(Source: https://source.android.com/source/requirements.html, CC:BY: 2.5)

Command Line Tools

The Android SDK tools available from the SDK Manager provide additional command-line tools to help you during your Android development. The tools are classified into two groups: SDK tools and platform tools. SDK tools are platform independent and are required no matter which Android platform you are developing on. Platform tools are customized to support the features of the latest Android platform.

Additional Command Line Tools

The Android SDK tools available from the SDK Manager provide additional command-line tools to help you during your Android development. The tools are classified into two groups: SDK tools and platform tools. SDK tools are platform independent and are required no matter which Android platform you are developing on. Platform tools are customized to support the features of the latest Android platform.

SDK Tools

The SDK tools are installed with the SDK starter package and are periodically updated. The SDK tools are required if you are developing Android applications. The most important SDK tools include the Android SDK Manager (Android sdk), the AVD Manager (Android AVD) the emulator (emulator), and the Dalvik Debug Monitor Server (DDMS). A short summary of some frequently-used SDK tools is provided below.

Virtual Device Tools

a) Android Virtual Device Manager

The AVD Manager provides a graphical user interface in which you can create and manage Android Virtual Devices (AVDs) that run in the Android Emulator.

b) Android Emulator (emulator)

A QEMU(short for quick emulator) based device emulation tool that you can use to debug and test your applications in an actual Android run-time environment.

c) mksdcard

Helps you create a disk image that you can use with the emulator, to simulate the presence of an external storage card (such as an SD card).

Development Tools

Hierarchy Viewer (hierarchyviewer) - Provides a visual representation of the layout's View hierarchy with performance information for each node in the layout, and a magnified view of the display to closely examine the pixels in your layout.

SDK Manager

SDK Manager lets you manage SDK packages, such as installed platforms and system images.

sqlite3 - Lets you access the SQLite data files created and used by Android applications.

Debugging Tools

The debugging tools are further explained in the later units of this material.

a) Android Monitor

Android Monitor is integrated into Android Studio and provides logcat, memory, CPU, GPU, and network monitors for app debugging and analysis.

b) adb

Android Debug Bridge (adb) is a versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device. It also provides access to the device shell.

c) Dalvik Debug Monitor Server (DDMS)

DDMS lets you debug Android apps.

d) Device Monitor

Android Device Monitor is a stand-alone tool that provides a graphical user interface for several Android application debugging and analysis tools.

e) Systrace

This tool lets you analyze the execution of your application in the context of system processes, to help diagnose display and performance issues.

f) traceview

Provides a graphical viewer for execution logs saved by your application.

g) Tracer for OpenGL ES

Allows you to capture OpenGL ES(the standard for Embedded Accelerated 3D Graphics) commands and frame-by-frame images to help you understand how your app is executing graphics commands.

Build Tools

a) apksigner

Signs APKs and checks whether APK signatures will be verified successfully on all platform versions that a given APK supports.

b) JOBB

Allows you to build encrypted and unencrypted APK expansion files in Opaque Binary Blob (OBB) format.

c) ProGuard

Shrinks, optimizes, and obfuscates your code by removing unused code and renaming classes, fields, and methods with semantically obscure names.

d) zipalign

Optimizes APK files by ensuring that all uncompressed data starts with a particular alignment relative to the start of the file.

Image Tools

a) Draw 9-patch

Allows you to easily create a NinePatch (class permits drawing a bitmap in nine or more sections) graphic using a WYSIWYG (What You See Is What You Get) editor. It also previews stretched versions of the image, and highlights the area in which content is allowed.

b) Etc1tool

A command line utility that lets you encode PNG images to the ETC1 compression standard and decode ETC1 compressed images back to PNG.

Platform Tools

The platform tools are typically updated every time you install a new SDK platform. Each update of the platform tools is backward compatible with older platforms. Usually, you directly use only one of the platform tools—the Android Debug Bridge (adb). Android Debug Bridge is a versatile tool that lets you manage the state of an emulator instance or Android-powered device. You can also use it to install an Android application (APK) file on a device.

The other platform tools, such as aidl, aapt, dexdump, and dx, are typically called by the Android build tools, so you rarely need to invoke these tools directly. As a general rule, you should rely on the build tools to call them as needed.

Note: The Android SDK provides additional shell tools that can be accessed through adb, such as bmgr and logcat.

a) bmgr

A shell tool you can use to interact with the Backup Manager on Android devices supporting API Level 8 or greater.

b) logcat

Provides a mechanism for collecting and viewing system debug output.

(Source: https://developer.android.com/studio/command-line/index.html CC:BY: 2.5)

Now, you have and learnt the systems requirements (hardware/software features) to set up the Android development platform and the Android command line tools. It is vital to determine the specific features of each Android version and how it has been developed to performing better. In unit 1 we learnt the history of Android and how each version of Android evolved. Next, we will summarize the Android platform versions with their unique features.

Android platform versions and specific features

There are rapid developments and new versions for the Android and Table 4.1 summarize the specific features of different Android platform versions up to now.

Table 4.1 summary the specific features of different Android platform versions

Android Platform version	**Specific Features**
Android 1.6 - Donut	Quick search box Screen size diversity Android market
Android 2.1 - Eclair	Google maps navigation Home screen customization Speech-to- Text
Android 2.2- Froyo	Voice actions Portable Hotspot Performance
Android 2.3- Gingerbread	Gaming APIs Near Field Communication (NFC) Battery Management
Android 3.0- Honeycomb	Tablet-friendly design System bar Quick settings

Android 4.0- Ice cream Sandwich	Custom home screen Data usage control Android Beam
Android 4.1- Jelly Bean	Google now Actionable Notifications Account switching
Android 4.4- Kitkat	Voice: OK Google Immersive Design Smart Dialer
Android 5.0- Lollipop	Material Design Multiscreen Notifications
Android 6.0- Marshmallow	Now on tap Permissions Battery works smarter
Android 7.0 - Nougat	Multi Locale language settings Multi-window view Quick switch between apps Data Saver Notification Controls Display Size

You can read more information online about the relative numbers of devices that are running different versions in the following link.

https://developer.android.com/about/dashboards/index.html

Thus, creating apps in Android for various mobile devices are increasing day by day. Since developing native apps is expensive, the demand for cross platform app development tools is also increasing. It is essential to know cross platforms and tools for mobile application development to develop apps for enhancing the market capacity.

You can watch the online presentation given in the link below to get familiar with the cross platforms for mobile application development.

bit.ly/XPlatformMobileDev

Activity

Activity 4.1

Explore most popular cross platforms and name three major cross platforms.

Briefly describe one of the major cross platform with the features and uses of them.

Hint: Check your answers with Answer guide at the end of this unit.

Now you are aware of the native and cross platform for mobile application development. We will now explore how the development environment is set up and configured.

4.3 Configuring Android Development Environment

To develop an Android application, first you have to setup the Android development environment.

First download and install the Android Studio, Java and then download and install every individual tool (like Java, SDK Manger, DDMS tool, AVD Manager, etc.)

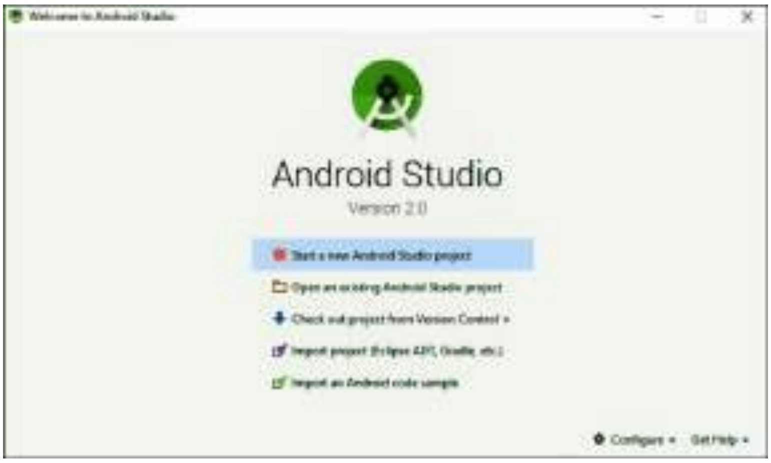
After installing Android tooling, it is possible to integrate development with various IDEs like, IntelliJ IDEA, Eclipse variants etc. Otherwise you can use any other java IDE or editor tool like notepad to compose the code.

Video - V5: Setting Up Android Development Environment

URL: https://tinyurl.com/y9kcehov

Let us now watch this video of setting up Android development environment set-by-step approach. This will help you at set up your Android development environment before you start programming with Android.

Next, we will be using Android Studio to develop the application. Android Studio is a native Android IDE that is fully dedicated to Android development. ADT Plugin needs to be installed to make it ready for Android development. Eclipse is an open source Java IDE that is compatible for several platforms. In early days Eclipse was the mostly used IDE by developers for Android applications.


Welcome to Android Studio
Android Studio
Version 2.0
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Import project (Eclipse ADT, Gradle, etc.)
Import an Android code sample
Configure
Get Help

Download and install Java Development Kit (JDK)

Most of the programming of Android is done using the Java programming language. To download latest Java Development Kit, follow the link given below.

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Now, extract the downloaded zip file and double click on the .exe and follow the instructions.

Download and install Android studio

You can download the latest Android Studio from the following link:

http://developer.android.com/sdk/index.html

Download the executable file from the above mentioned link, double click on the file and follow the installation instructions. To install the latest “Android Studio” you need to install compatible Java SE Development Kit first.

Online reading: You can pursue the following link to grab the installation instructions for Android Studio.

http://developer.android.com/sdk/installing/index.html

After installing and when you will start the Android Studio first time, the very first window will look like Figure 4.1.

Figure 4.1: First time opening window of Android Studio

When you click on the first option to start a new Android Studio Project, multiple windows will pop up one after another to setup a new project.

Online reading: Android Studio has numerous features. In this course, you are going to explore and use many features of Android Studio. For the prior reading to understand the Android studio, follow the following web link:

http://developer.android.com/tools/studio/index.html

Once you install Android Studio, it is easy to keep the Android Studio IDE and Android SDK tools up to date with automatic updates and the Android SDK Manager.

Following web page provides update instructions of IDE and Android SDK tools.

https://developer.android.com/studio/intro/update.html#channels

Activity

Activity 4.2

State the most common tools used in Android application development

Video-V6: Install Android for Windows 10

URL:https://storage.googleapis.com/androiddevelopers/videos/studio-install-windows.mp4

This video will show you how to install and configure the Android development environment using Android Studio in Windows 10.

Furthermore, the following link has videos that shows each step of the recommended setup procedure for Mac and Linux.

https://developer.android.com/studio/install.html

Lab Exercise:

Download and install JDK and the latest Android Studio version to configure the development environment.

Now, you are ready to go for making an Android App.

Unit summary

The first step to start on Android based application development is to set up the development environment. Therefore, it is essential to know how to configure development environment and installing tools for Android application development.

In this unit, you have learned Android development platforms, tools and cross platforms. At the last section of the unit, you learnt about setting up the Android development environment by installing the latest JDK and Android Studio. It is also important to keep the Android studio IDE and Android SDK tools up to date at the development environment.

Unit 5

Android application fundamentals

Introduction

In this unit we describe of the basic app components, additional components, resources and the manifest file using Android Studio, which is the Open Source platform provided for application developers. This unit also provides you an overview of the interaction and the association between the components and the application.

Upon completion of this unit you will be able to:

Outcomes

- Explain the activity Components
- Outline the manifest file
- Explain the introduction to AVD
- Create an Android Virtual Device to simulate a device and to display on the development computer

Terminology

Manifest Filbe:	file containing metadata for a group files that are part of a coherent unit
Bound Services:	Server in a client-server interface
RPC:	Remote Procedure Call
API:	Application Programming Interface

5.1 Basic App Components

As you already learnt in Unit 2, there are four types of app components and they are:

- Activities
- Services
- Content Providers
- Broadcast Receivers

Activities

An activity is the entry point for interacting with the user. As explained in unit 2,it represents a single screen with a user interface. An activity facilitates the following key interactions between system and app:

- Keeping track of what the user currently cares about (what is on screen) to ensure that the system keeps running the process that is hosting the activity.
- Knowing that previously used processes contain things the user may return to (stopped activities), and thus more highly prioritize keeping those processes around.
- Helping the app handle having its process is killed so the user can return to activities with their previous state restored.
- Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. (The most classic example here being share.)

An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with `setContentView`(View). While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with windowIsFloating set) or embedded inside of another activity (using ActivityGroup).

There are two methods almost all subclasses of Activity will implement:

- `onCreate`(Bundle) is where you initialize your activity. Most importantly, here you will usually call setContentView(int) with a layout resource defining your UI, and using findViewById(int) to retrieve the widgets in that UI that ryou need to interact with programmatically.
- `onPause`() is where you deal with the user leaving your activity. Most importantly, any changes made by the user should at this point be committed (usually to the ContentProvider holding the data).

To be of use with `Context.startActivity()`, all activity classes must have a corresponding <activity> declaration in their package's `AndroidManifest.xml` as shown in Program 5.1. An activity must be implemented as a subclass of the Activity class.

```
public class MainActivity extends Activity {
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
/** Called when the activity is about to be visible.*/
   @Override
   protected void onStart() {
      super.onStart();
   }
   /** Called when the activity has become visible. */
   @Override
   protected void onResume() {
      super.onResume();
   }
  /** Called when another activity is taking focus. */
   @Override
   protected void onPause() {
      super.onPause();      }
 /** Called when the activity is no longer visible. */
   @Override
   protected void onStop() {
      super.onStop();
   }
 /** Called just before the activity is destroyed. */
   @Override
   public void onDestroy() {
      super.onDestroy();   }
}
```

Program 5.1 Methods in Activity Class

Source (http://www.androdevelopment.com/android-activities/)

Services

A service is a general-purpose entry point for keeping an app running in the background for all kinds of reasons. As stated in unit 2, it is a component that runs in the background to perform long running operations or to perform work for remote processes. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it. There are two very distinct semantics services that tell the system about how to manage an app: Started services tell the system to keep them running until their work is completed. This could be to sync some data in the background or play music even after the user leaves the app. Syncing data in the background or playing music also represents two different types of started services that modify how the system handles them:

- Music playback is something the user is directly aware of, so the app tells the system this by saying it wants to be foreground with a notification to tell the user about it; in this case the system knows that it should try really hard to keep that service's process running, because the user will be unhappy if it goes away.
- A regular background service is not something the user is directly aware as running, so the system has more freedom in managing its process. It may allow it to be killed (and then restarting the service sometime later) if it needs RAM for things that are of more immediate concern to the user.

Bound services run because some other app (or the system) has said that it wants to make use of the service. This is basically the service providing an API to another process. The system thus knows there is a dependency between these processes, so if process A is bound to a service in process B, it knows that it needs to keep process B (and its service) running for A. Further, if process A is something the user cares about, then it also knows to treat process B as something the user also cares about. Because of their flexibility (for better or worse), services have turned out to be a really useful building block for all kinds of higher-level system concepts. Live wallpapers, notification listeners, screen savers, input methods, accessibility services, and many other core system features are all built as services that applications implement and the system binds to when they should be running.

To create a service, you must create a subclass of Service or use one of its existing subclasses. In your implementation, you must override some callback methods that handle key aspects of the service lifecycle and provide a mechanism that allows the components to bind to the service, if appropriate. You need to have prior knowledge of applying object oriented concepts to do this. These are the most important callback methods that you should override:

onStartCommand()

The system invokes this method by calling startService() when another component (such as an activity) requests that the service be started. When this method executes, the service is started and can run in the background indefinitely. If you implement this, it is your responsibility to stop the service when its work is complete by calling stopSelf() or stopService(). If you only want to provide binding, you don't need to implement this method.

onBind()

The system invokes this method by calling bindService() when another component wants to bind with the service (such as to perform RPC). In your implementation of this method, you must provide an interface that clients use to communicate with the service by returning an IBinder. You must always implement this method; however, if you don't want to allow binding, you should return null.

onCreate()

The system invokes this method to perform one-time setup procedures when the service is initially created (before it calls either onStartCommand() or onBind()). If the service is already running, this method is not called.

onDestroy()

The system invokes this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, or receivers. This is the last call that the service receives.

If a component starts the service by calling startService() (which results in a call to onStartCommand()), the service continues to run until it stops itself with stopSelf() or another component stops it by calling stopService().

If a component calls bindService() to create the service and onStartCommand() is not called, the service runs only as long as the component is bound to it. After the service is unbound from all of its clients, the system destroys it.

Traditionally, there are two classes you can extend to create a started service named Service and IntentService.

The Service is the base class for all services (shown in Program 5.2). When you extend this class, it is important to create a new thread in which the service can complete all of its work; the service uses your application's main thread by default, which can slow the performance of any activity that your application is running.

```
public class HelloService extends Service {
  private Looper mServiceLooper;
  private ServiceHandler mServiceHandler;

  // Handler that receives messages from the thread
  private final class ServiceHandler extends Handler {
      public ServiceHandler(Looper looper) {
          super(looper);
      }
      @Override
      public void handleMessage(Message msg) {

 // Normally we would do some work here, like download a
//file. For our sample, we just sleep for 5 seconds.
          try {
              Thread.sleep(5000);
          } catch (InterruptedException e) {
              // Restore interrupt status.
              Thread.currentThread().interrupt();
          }
// Stop the service using the startId, so that we don't stop
//the service in the middle of handling another job
          stopSelf(msg.arg1);
      }
  }
  @Override
  public void onCreate() {
// Start up the thread running the service.  Note that we
//create a separate thread because the service normally runs
//in the process's main thread, which we don't want to
//block.  We also make it background priority so CPU-
//intensive work will not disrupt our UI.

    HandlerThread thread = new
HandlerThread("ServiceStartArguments",
            Process.THREAD_PRIORITY_BACKGROUND);
    thread.start();

// Get the HandlerThread's Looper and use it for our
//Handler
    mServiceLooper = thread.getLooper();
    mServiceHandler = new
ServiceHandler(mServiceLooper);
  }
  @Override
  public int onStartCommand(Intent intent, int flags,
int startId) {
      Toast.makeText(this, "service starting",
Toast.LENGTH_SHORT).show();

// For each start request, send a message to start a
// job and deliver the start ID so we know which
// request we're stopping when we finish the job
      Message msg = mServiceHandler.obtainMessage();
      msg.arg1 = startId;
      mServiceHandler.sendMessage(msg);

// If we get killed, after returning from here, restart
      return START_STICKY;
  }
  @Override
  public IBinder onBind(Intent intent) {
      // We don't provide binding, so return null
      return null;
  }
  @Override
  public void onDestroy() {
    Toast.makeText(this, "service done",
Toast.LENGTH_SHORT).show();
  }
}
```

Program 5.2 Service class declaration

Source(https://developer.android.com/guide/components/services.html)

The IntentService is a subclass of Service that uses a worker thread to handle all of the start requests, one at a time (Shown in Program 5.3). This is the best option if you don't require that your service handle multiple requests simultaneously. Implement onHandleIntent(), which receives the intent for each start request so that you can complete the background work.

```
public class HelloIntentService extends IntentService
{
/** A constructor is required, and must call the super
 * IntentService(String)constructor with a name for
the worker thread.*/
  public HelloIntentService() {
      super("HelloIntentService");
  }
/* The IntentService calls this method from the
default worker thread with the intent that started the
service. When this method returns, IntentService stops
the service, as appropriate. */
  @Override
  protected void onHandleIntent(Intent intent) {
 // Normally we would do some work here, likedownload a
//file. For our sample, we just sleep for 5 seconds.
      try {
          Thread.sleep(5000);
      } catch (InterruptedException e) {
          // Restore interrupt status.
          Thread.currentThread().interrupt();
      }
  }
}
```

Program 5.3 Intent Service subclass declaration
Source (https://developer.android.com/guide/components/services.html)

Broadcast receivers

As already stated in unit 2, broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements. Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that aren't currently running. So, for example, an app can schedule an alarm to post a notification to tell the user about an upcoming event and by delivering that alarm to a BroadcastReceiver of the app, there is no need for the app to remain running until the alarm goes off. Many broadcasts originate from the system for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.

A broadcast receiver is implemented as a subclass (shown in program 5.4) of BroadcastReceiver and each broadcast is delivered as an Intent object.

```
public class MyReceiver extends BroadcastReceiver{
   @Override
   public void onReceive(Context context, Intent
intent) {
      // Implement action for received broadcast.
   }
}
```

Program 5.4 Broadcast receiver subclass

Source(https://developer.android.com/samples/AppShortcuts/src/com.example.android.appshortcuts/MyReceiver.html?hl=pt-br)

Content providers

As already stated in unit 2, content provider manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access. There are a few particular things this allows the system to do in managing an app:

- Assigning a Uniform Resource Identifier (URI) does not require that the app remain running, so URIs can persist after their owning apps have exited. The system only needs to make sure that an owning app is still running when it has to retrieve the app's data from the corresponding URI.

- These URIs also provide an important fine-grained security model. For example, an app can place the URI for an image it has on the clipboard, but leave its content provider locked up so that other apps cannot freely access it. When a second app attempts to access that URI on the clipboard, the system can allow that app to access the data via a temporary URI permission grant so that it is allowed to access the data only behind that URI, but nothing else in the second app.

- A unique aspect of the Android system design is that any app can start another app's component. For example, if you want the user to capture a photo with the device camera, there's probably another app that does that and your app can use it instead of developing an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera app. Instead, you can simply start the activity in the camera app that captures a photo. When complete, the photo is even returned to your app so you can use it. To the user, it seems as if the camera is actually a part of your app.

- When the system starts a component, it starts the process for that app if it's not already running and instantiates the classes needed for the component. For example, if your app starts the activity in the camera app that captures a photo, that activity runs in the process that belongs to the camera app, not in your app's process. Therefore, unlike apps on most other systems, Android apps don't have a single entry point (there's no main() function).

- Because the system runs each app in a separate process with file permissions that restrict access to other apps, your app cannot directly activate a component from another app. However, the Android system can. To activate a component in another app, deliver a message to the system that specifies your *intent* to start a particular component. The system then activates the component for you.

Video -V6: Android Development

URL: https://tinyurl.com/y9xuh62v

In this video, we will be introducing the components of the Android Application Fundamentals. You may watch the video while reading this unit in order to understand the content better. After watching the video answer the question in Activity 5.1.

Activity

Activity 5.1

Identify the element that is not part of the basic application component of an Android application.

○ Activities

○ Services

○ Content Providers

○ Screencast Receivers

o Broadcast Receivers

5.2 Additional Components

Other than basic four app components there are few additional components as well. In this section we will discuss these additional components.

Application Class

The Application class in Android is Base class for maintaining global application state which contains all other components such as activities and services. The Application class, or any subclass of the Application class, is instantiated before any other class when the process for your application/package is created.

Defining Your Application Class

If we do want a custom application class, we start by creating a new class which extends Android.app.Application as the following Program 5.5:

```
import android.app.Application;

public class MyCustomApplication extends Application {
// Called when the application is starting, before any
// other application objects have been created.
        // Overriding this method is totally optional!
        @Override
        public void onCreate() {
            super.onCreate();
            // Required initialization logic here!
        }
// Called by the system when the device configuration
changes //while your component is running.
        // Overriding this method is totally optional!
        @Override
        public void
onConfigurationChanged(Configuration newConfig) {
            super.onConfigurationChanged(newConfig);
        }
// This is called when the overall system is running
//low on memory, and would like actively running
//processes to tighten their belts.
// Overriding this method is totally optional!
        @Override
        public void onLowMemory() {
            super.onLowMemory();
                }
}
```

Program 5.5 Application class

Source (https://guides.codepath.com/android/Understanding-the-Android-Application-Class)

And specify the `android:name` property in the `<application>` node in `AndroidManifest.xml` as in code snippet given below.

```
<application
   android:name=".MyCustomApplication"
   android:icon="@drawable/icon"
   android:label="@string/app_name"
   ...>
```

Source(https://guides.codepath.com/android/Understanding-the-Android-Application-Class)

That is all what you need to get started with your custom application.

Fragment

A Fragment is a piece of an application's user interface or behavior that can be placed in an Activity. Interaction with fragments is done through FragmentManager, which can be obtained via Activity.getFragmentManager() andFragment.getFragmentManager().

The Fragment class can be used many ways to achieve a wide variety of results. In its core, it represents a particular operation or interface that is running within a larger Activity. A Fragment is closely tied to the Activity it is in, and cannot be used apart from one. Though Fragment defines its own lifecycle, that lifecycle is dependent on its activity: if the activity is stopped, no fragments inside of it can be started; when the activity is destroyed, all fragments will be destroyed.

All subclasses of Fragment must include a public no-argument constructor. The framework will often re-instantiate a fragment class when needed, in particular during state restore, and needs to be able to find this constructor to instantiate it. If the no-argument constructor is not available, a runtime exception will occur in some cases during state restore.

Fragment Lifecycle

Though a Fragment's lifecycle is tied to its owning activity, it has its own wrinkle on the standard activity lifecycle. It includes basic activity lifecycle methods such as onResume(), but also important are methods related to interactions with the activity and UI generation.

The core series of lifecycle methods that are called to bring a fragment up to resumed state (interacting with the user) are:

1. **onAttach(Activity)** called once the fragment is associated with its activity.
2. **onCreate(Bundle)** called to do initial creation of the fragment.
3. **onCreateView(LayoutInflater, ViewGroup, Bundle)** creates and returns the view hierarchy associated with the fragment.
4. **onActivityCreated(Bundle)** tells the fragment that its activity has completed its own Activity.onCreate().
5. **onViewStateRestored(Bundle)** tells the fragment that all of the saved state of its view hierarchy has been restored.
6. **onStart()** makes the fragment visible to the user (based on its containing activity being started).
7. **onResume()** makes the fragment begin interacting with the user (based on its containing activity being resumed).

As a fragment is no longer being used, it goes through a reverse series of callbacks:

1. **onPause()** fragment is no longer interacting with the user either because its activity is being paused or a fragment operation is modifying it in the activity.
2. **onStop()** fragment is no longer visible to the user either because its activity is being stopped or a fragment operation is modifying it in the activity.
3. **onDestroyView()** allows the fragment to clean up resources associated with its View.
4. **onDestroy()** called to do final cleanup of the fragment's state.
5. **onDetach()** called immediately prior to the fragment no longer being associated with its activity.

Fragment Layout

Fragments can be used as part of your application's layout, allowing you to better modularize your code and more easily adjust your user interface to the screen it is running on. As an example, we can look at a simple program consisting of a list of items, and display of the details of each item.

An activity's layout XML can include <fragment> tags to embed fragment instances inside of the layout. For example, here is a simple layout that embeds one fragment shown in code snippet below:

```
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/andr
oid"
    android:layout_width="match_parent"
android:layout_height="match_parent">
    <fragment
class="com.example.android.apis.app.FragmentLayout$Tit
lesFragment"
            android:id="@+id/titles"
            android:layout_width="match_parent"
android:layout_height="match_parent" />
</FrameLayout>
```

Source :(https://developer.android.com/reference/android/app/Fragment.html)

The layout is installed in the activity in the normal way as shown in code snippet below:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.fragment_layout);
}
```

Source: (https://developer.android.com/reference/android/app/Fragment.html

View

This class represents the basic building block for user interface components. A View occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for *widgets*, which are used to create interactive UI components (buttons, text fields, etc.). The ViewGroup subclass is the base class for *layouts*, which are invisible containers that hold other Views (or other ViewGroups) and define their layout properties.

Using Views

All of the views in a window are arranged in a single tree. You can add views either from code or by specifying a tree of views in one or more XML layout files. There are many specialized subclasses of views that act as controls or are capable of displaying text, images, or other content.

Set properties: for example, setting the text of a TextView. The available properties and the methods that set them will vary among the different subclasses of views. Note that properties that are known at build time can be set in the XML layout files.

Set focus: The framework will handle moving focus in response to user input. To force focus to a specific view, call requestFocus().

Set up listeners: Views allow clients to set listeners that will be notified when something interesting happens to the view. For example, all views will let you set a listener to be notified when the view gains or loses focus. You can register such a listener using setOnFocusChangeListener(android.view.View.OnFocusChangeListener). Other view subclasses offer more specialized listeners. For example, a Button exposes a listener to notify clients when the button is clicked.

Set visibility: You can hide or show views using setVisibility(int).

Note: The Android framework is responsible for measuring, laying out and drawing views. You should not call methods that perform these actions on views yourself unless you are actually implementing a *ViewGroup*.

Intents and Intent Filters

An Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use-cases:

- **To start an activity:** An Activity represents a single screen in an app. You can start a new instance of an Activity by passing an Intent to startActivity(). The Intent describes the activity to start and carries any necessary data. If you want to receive a result from the activity when it finishes, call startActivityForResult(). Your activity receives the result as a separate Intent object in your activity's onActivityResult() callback. For more information, see the Activities guide.

- **To start a service:** A Service is a component that performs operations in the background without a user interface. You can start a service to perform a one-time operation (such as download a file) by passing an Intent tostartService(). The Intent describes the service to start and carries any necessary data. If the service is designed with a client-server interface, you can bind to the service from another component by passing an Intent to bindService(). For more information, see the Services guide

- **To deliver a broadcast:** A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an Intent to sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().

Intents Types

There are two types of intents:

- **Explicit intents** specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, start a new activity in response to a user action or start a service to download a file in the background.
- **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

When you create an explicit intent to start an activity or service, the system immediately starts the app component specified in the Intent object.

When you create an implicit intent, the Android system finds the appropriate component to start by comparing the contents of the intent to the *intent filters* declared in the manifest file of other apps on the device. If the intent matches an intent filter, the system starts that component and delivers it the Intent object. If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.

5.3 Resources

This sits on top of the asset manager of the application (accessible through getAssets()) and provides a high-level API for getting typed data from the assets.

The Android resource system keeps track of all non-code assets associated with an application. You can use this class to access your application's resources. You can generally acquire the Resources instance associated with your application with getResources().

The Android SDK tools compile your application's resources into the application binary at build time. To use a resource, you must install it correctly in the source tree (inside your project's `res/` directory) and build your application. As part of the build process, the SDK tools generate symbols for each resource, which you can use in your application code to access the resources.

Using application resources makes it easy to update various characteristics of your application without modifying code, and—by providing sets of alternative resources—enables you to optimize your application for a variety of device configurations (such as for different languages and screen sizes). This is an important aspect of developing Android applications that are compatible on different types of devices.

You should always externalize resources such as images and strings from your application code, so that you can maintain them independently. Externalizing your resources also allows you to provide alternative resources that support specific device configurations such as different languages or screen sizes, which becomes increasingly important as more Android-powered devices become available with different configurations. In order to provide compatibility with different configurations, you must organize resources in your project's `res/` directory, using various sub-directories that group resources by type and configuration.

For any type of resource, you can specify *default* and multiple *alternative* resources for your application:

- Default resources are those that should be used regardless of the device configuration or when there are no alternative resources that match the current configuration.
- Alternative resources are those that you've designed for use with a specific configuration. To specify that a group of resources are for a specific configuration, append an appropriate configuration qualifier to the directory name.

For example, while your default UI layout is saved in the res/layout/ directory, you might specify a different layout to be used when the screen is in landscape orientation, by saving it in the res/layout-land/ directory. Android automatically applies the appropriate resources by matching the device's current configuration to your resource directory names.

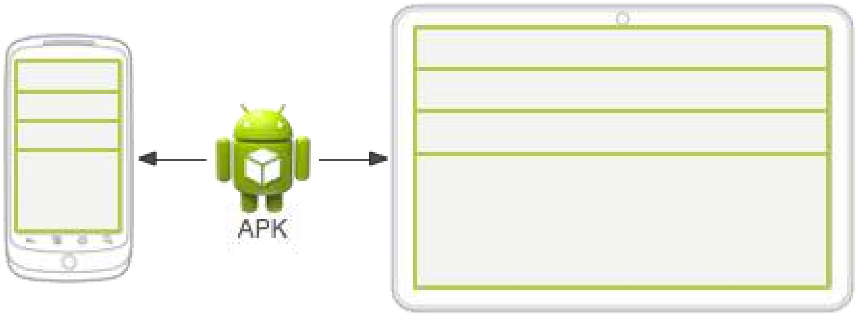


Figure 5.1 Two different devices, each using the default layout (the app provides no alternative layouts).

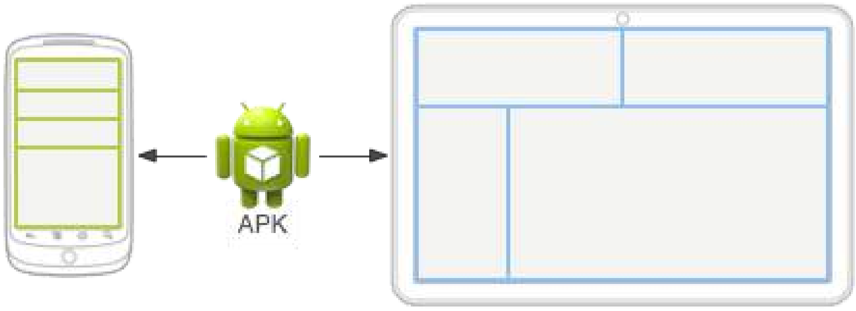


Figure 5.2 Two different devices, each using a different layout provided for different screen sizes.

Figure 5.1: illustrates how the system applies the same layout for two different devices when there are no alternative resources available. Figure 2 shows the same application when it adds an alternative layout resource for larger screens.

The following section provide a complete guide to how you can organize your application resources, specify alternative resources, access them in your application, and more:

Providing Resources

What kinds of resources you can provide in your app, where to save them, and how to create alternative resources for specific device configurations.

Accessing Resources

How to use the resources you've provided, either by referencing them from your application code or from other XML resources.

Handling Runtime Changes

How to manage configuration changes that occur while your Activity is running.

Localization

A bottom-up guide to localizing your application using alternative resources. While this is just one specific use of alternative resources, it is very important in order to reach more users.

Complex XML Resources

An XML format for building complex resources like animated vector drawables in a single XML file.

Resource Types

A reference of various resource types you can provide, describing their

XML elements, attributes, and syntax. For example, this reference shows you how to create a resource for application menus, drawables, animations, and more.

Activity

Activity 5.2

Explain how you can organize your application resources.

5.4 Android Manifest

Every application must have an `AndroidManifest.xml` file (with precisely that name) in its root directory. The manifest file provides essential information about your app to the Android system, which the system must have before it can run any of the app's code.

Among other things, the manifest file does the following:

- It names the Java package for the application. The package name serves as a unique identifier for the application.
- It describes the components of the application, which include the activities, services, broadcast receivers, and content providers that compose the application. It also names the classes that implement each of the components and publishes their capabilities, such as the Intent messages that they can handle. These declarations inform the Android system of the components and the conditions in which they can be launched.
- It determines the processes that host the application components.
- It declares the permissions that the application must have in order to access protected parts of the API and interact with other applications. It also declares the permissions that others are required to have in order to interact with the application's components.
- It lists the Instrumentation classes that provide profiling and other information as the application runs. These declarations are present in the manifest only while the application is being developed and are removed before the application is published.
- It declares the minimum level of the Android API that the application requires.
- It lists the libraries that the application must be linked against.

Manifest file structure

The code snippet below shows the general structure of the manifest file and every element that it can contain. Each element, along with all of its attributes, is fully documented in a separate file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />

    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

 <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>
            <grant-uri-permission />
            <meta-data />
            <path-permission />
        </provider>
        <uses-library />
    </application>
</manifest>
```

Source: (https://developer.android.com/guide/topics/manifest/manifest-intro.html)

5.5 File conventions

This section describes the conventions and rules that apply generally to all of the elements and attributes in the manifest file.

Elements

Only the <manifest> and <application> elements are required. They each must be present and can occur only once. Most of the other elements can occur many times or not at all. However, at least some of them must be present before the manifest file becomes useful.

If an element contains anything at all, it contains other elements. All of the values are set through attributes, not as character data within an element.

Elements at the same level are generally not ordered. For example, the <activity>, <provider>, and <service> elements can be intermixed in any sequence. There are two key exceptions to this rule:

- An <activity-alias> element must follow the <activity> for which it is an alias.
- The <application> element must be the last element inside the <manifest> element. In other words, the `</application>` closing tag must appear immediately before the `</manifest>` closing tag.

Attributes

In a formal sense, all attributes are optional. However, there are some attributes that must be specified so that an element can accomplish its purpose. Use the documentation as a guide. For truly optional attributes, it mentions a default value or states what happens in the absence of a specification.

Except for some attributes of the root <manifest> element, all attribute names begin with an `android:` prefix. For example, `android:alwaysRetainTaskState`. Because the prefix is universal, the documentation generally omits it when referring to attributes by name.

Declaring class names

Many elements correspond to Java objects, including elements for the application itself (the <application> element) and its principal components: activities (<activity>), services (<service>), broadcast receivers (<receiver>), and content providers (<provider>).

If you define a subclass, as you almost always would for the component classes (Activity, Service, BroadcastReceiver, and ContentProvider), the subclass is declared through a `name` attribute. The name must include the full package designation. For example, a Service subclass might be declared as follows:

```
<manifest . . . >
    <application . . . >
        <service
android:name="com.example.project.SecretService" . . .
>
            . . .
        </service>
        . . .
    </application>
</manifest>
```

Source (https://developer.android.com/guide/topics/manifest/manifest-intro.html)

However, if the first character of the string is a period, the application's package name (as specified by the
<manifest> element's package attribute) is appended to the string. The following assignment is the same as that shown above:

```
<manifest package="com.example.project" . . . >
    <application . . . >
        <service android:name=".SecretService" . . . >
            . . .
        </service>
        . . .
    </application>
</manifest>
```

Source *(https://developer.android.com/guide/topics/manifest/manifest-intro.html)*

When starting a component, the Android system creates an instance of the named subclass. If a subclass isn't specified, it creates an instance of the base class.

Multiple values

If more than one value can be specified, the element is almost always repeated, rather than multiple values being listed within a single element.

For example, an intent filter can list several actions:

```
<intent-filter . . . >
    <action android:name="android.intent.action.EDIT"
/>
    <action
android:name="android.intent.action.INSERT" />
    <action
android:name="android.intent.action.DELETE" />
    . . .
</intent-filter>
```

Source:*(https://developer.android.com/reference/android/content/Intent.html)*

Resource values

Some attributes have values that can be displayed to users, such as a label and an icon for an activity. The values of these attributes should be localized and set from a resource or theme. Resource values are expressed in the following format:

@[<i>package</i>:]<i>type</i>/<i>name</i>

You can ommit the *package* name if the resource is in the same package as the application. The *type* is a type of resource, such as *string* or *drawable*, and the *name* is the name that identifies the specific resource. Here is an example:

```
<activity android:icon="@drawable/smallPic" . . . >
```

Source (https://developer.android.com/guide/topics/manifest/manifest-intro.html)

The values from a theme are expressed similarly, but with an initial ? instead of @:

?[<i>package</i>:]<i>type</i>/<i>name</i>

String values

Where an attribute value is a string, you must use double backslashes (\\) to escape characters, such as \\n for a newline or \\uxxxx for a Unicode character.

Activity

Activity 5.3

Explain the role of AndroidManifest.xml file in an Android application.

Unit summary

First part of this unit gave you an in-depth knowledge about the four main types of mobile application components. This unit also introduced you to the methods that was facilitated by Android and the elements used in any Android application development such as fragments and intents. You will need to refer to this unit when you are writing your application to understand the process beneath each method while executing.

Unit 6

Android Development

Introduction

Now you are aware that Android applications can be developed using Android Studio. You have also learnt fundamentals of Android App design. How to build a simple user interface and handle user input will be described further in this unit.

Particularly, you will apply Java features in the context of core Android components (such as Activities and basic UI elements) by applying common tools (such as Android Studio) needed to develop Java programs and useful Android apps.

Upon completion of this unit you will be able to:

Outcomes

- Develop and run an Android application.
- Run the developed application in both on the actual device and in the emulator.

Terminology

AVD:	Android Virtual Emulator
UI:	User Interface
XML:	eXtensible Markup Language

6.1 Creating Your First Program

This unit shows you how to create a new Android project with Android Studio and describes some of the files in the project.

In Android Studio, create a new project

- In the New Project screen, enter the following values:

Application Name: "My First App"
Company Domain: "example.com"

Android Studio fills in the package name and project location for you, but you can edit these if you'd like.

- Click Next.

- In the Target Android Devices screen, keep the default values and click Next.

The Minimum Required SDK is the earliest version of Android that your app supports, which is indicated by the API level. To support as many devices as possible, you should set this to the lowest version available that allows your app to provide its core feature set. If any feature of your app is possible only on newer versions of Android and it's not critical to the core feature set, enable that feature only when running on the versions that support it.

- In the Add an Activity to Mobile screen, select Empty Activity and click Next.

- In the Customize the Activity screen, keep the default values and click Finish.

After some processing, Android Studio opens and displays a "My First App" app with default files. You will add functionality to some of these files in the following lessons.

Now take a moment to review the most important files. First, be sure that the Project window is open (select View > Tool Windows > Project) and the Android view is selected from the drop-down list at the top. You can then see the following files:

app > java > com.example.myfirstapp > MainActivity.java

This file appears in Android Studio after the New Project wizard finishes. It contains the class definition for the activity you created earlier. When you build and run the app, the Activity starts and loads the layout file that says "Hello World!"

app > res > layout > activity_main.xml

This XML file defines the layout of the activity. It contains a TextView element with the text "Hello world!

app > manifests > AndroidManifest.xml

The manifest file describes the fundamental characteristics of the app and defines each of its components. You'll revisit this file as you follow these lessons and add more components to your app.

Gradle Scripts > build.gradle

Android Studio uses Gradle to compile and build your app. There is a build.gradle file for each module of your project, as well as a build.gradle file for the entire project. Usually, you're only interested in the build.gradle file for the module.

Activity

Activity 6.1

- Create a simple “Hello World Program”

6.2 Building and running the application

By default, Android Studio sets up new projects to deploy to the Emulator or a physical device with just a few clicks. With Instant Run, you can push changes to methods and existing app resources to a running app without building a new APK, so code changes are visible almost instantly.

To build and run your app, click Run ▶. Android Studio builds your app with Gradle, asks you to select a deployment target (an emulator or a connected device), and then deploys your app to it. You can customize some of this default behaviour, such as selecting an automatic deployment target, by changing the run configuration.

If you want to use the Android Emulator to run your app, you need to have an Android Virtual Device (AVD) ready. If you haven't already created one, then after you click Run, click Create New Emulator in the Select Deployment Target dialog. Follow the Virtual Device Configuration wizard to define the type of device you want to emulate. For more information, see Create and Manage Virtual Devices.

Select and build a different module

If your project has multiple modules beyond the default app module, you can build a specific module as follows:

- Select the module in the Project panel, and then click Build > Make Module *module-name*.

Android Studio builds the module using Gradle. Once the module is built, you can run and debug it if you've built a module for a new app or new device, or use it as a dependency if you've built a library or Google Cloud module.

To run a built app module:

- Click Run > Run and select the module from the Run dialog.

Change the run/debug configuration

The run/debug configuration specifies the module to run, package to deploy, activity to start, target device, emulator settings, logcat options, and more. The default run/debug configuration launches the default project activity and uses the Select Deployment Target dialog for target device selection. If the default settings don't suit your project or module, you can customize the run/debug configuration, or even create a new one, at the project, default, and module levels. To edit a run/debug configuration, select **Run** > **Edit Configurations**.

Change the build variant

By default, Android Studio builds the debug version of your app, which is intended only for testing, when you click Run. You need to build the release version to prepare your app for public release.

To change the build variant Android Studio uses, select **Build** > **Select Build Variant** in the menu bar (or click Build Variants in the windows bar), and then select a build variant from the drop-down menu. By default, new projects are set up with a debug and release build variant.

Using *product flavours*, you can create additional build variants for different versions of your app, each having different features or device requirements.

Generate APKs

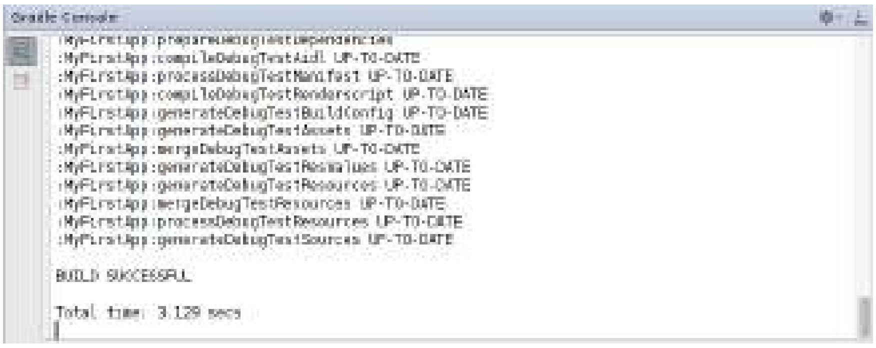
To create an APK for your app, follow these steps:

- Select the build variant you want to build from **the Build Variants** window.
- Click **Build > Build APK** in the menu bar.
- To instead build the APK and immediately run it on a device, click **Run** in the toolbar.

All built APKs are saved in ***project-name*/*module-name*/build/outputs/apk/**. You can also locate the generated APKs by clicking the link in the pop-up dialog that appears once the build is complete, as shown in figure 2

Monitor the build process

You can view details about the build process by clicking **View > Tool Windows > Gradle Console** (or by clicking **Gradle Console** in the tool window bar). The console displays each task that Gradle executes in order to build your app, as shown in Figure 6.1.



Gradle Console

```
:MyFirstApp:prepareDebugTestDependencies
:MyFirstApp:compileDebugTestAidl UP-TO-DATE
:MyFirstApp:processDebugTestManifest UP-TO-DATE
:MyFirstApp:compileDebugTestRenderscript UP-TO-DATE
:MyFirstApp:generateDebugTestBuildConfig UP-TO-DATE
:MyFirstApp:generateDebugTestAssets UP-TO-DATE
:MyFirstApp:mergeDebugTestAssets UP-TO-DATE
:MyFirstApp:generateDebugTestResValues UP-TO-DATE
:MyFirstApp:generateDebugTestResources UP-TO-DATE
:MyFirstApp:mergeDebugTestResources UP-TO-DATE
:MyFirstApp:processDebugTestResources UP-TO-DATE
:MyFirstApp:generateDebugTestSources UP-TO-DATE

BUILD SUCCESSFUL

Total time: 3.129 secs
```

Figure 6.1 The Gradle Console in Android Studio.

If your build variants use product flavours, Gradle also invokes tasks to build those product flavours. To view the list of all available build tasks, click **View** > **Tool Windows** > **Gradle** (or click **Gradle** ◉ in the tool window bar).

If an error occurs during the build process, the *Messages* window appears to describe the issue. Gradle may recommend some command-line options to help you resolve the issue, such as `--stacktrace` or `--debug`. To use command-line options with your build process:

- Open the **Settings** or **Preferences** dialog:
- Navigate to **Build, Execution, Deployment** > **Compiler**.
- In the text field, next to *Command-line Options*, enter your command-line options.
- Click **OK** to save and exit.

Gradle will apply these command-line options the next time you try building your app.

Running on the Emulator

Before you run your app on an emulator, you need to create an Android Virtual Device (AVD) definition. An AVD definition defines the characteristics of an Android phone, tablet, Android Wear, or Android TV device that you want to simulate in the Android Emulator.

Create an AVD Definition as follows:

- Launch the Android Virtual Device Manager by selecting **Tools > Android > AVD Manager**, or by clicking the AVD Manager icon in the toolbar.
- In the Your Virtual Devices screen, click Create Virtual Device.

- In the Select Hardware screen, select a phone device, such as Nexus 6, and then click Next.
- In the System Image screen, choose the desired system image for the AVD and click Next. (if you don't have a particular system image installed, you can get it by clicking the download link.)

Verify the configuration settings (for your first AVD, leave all the settings as they are), and then click **Finish**.

Create and Manage Virtual Devices

An Android Virtual Device (AVD) definition lets you define the characteristics of an Android phone, tablet, Android Wear, or Android TV device that you want to simulate in the Android Emulator. The AVD Manager helps you easily create and manage AVDs

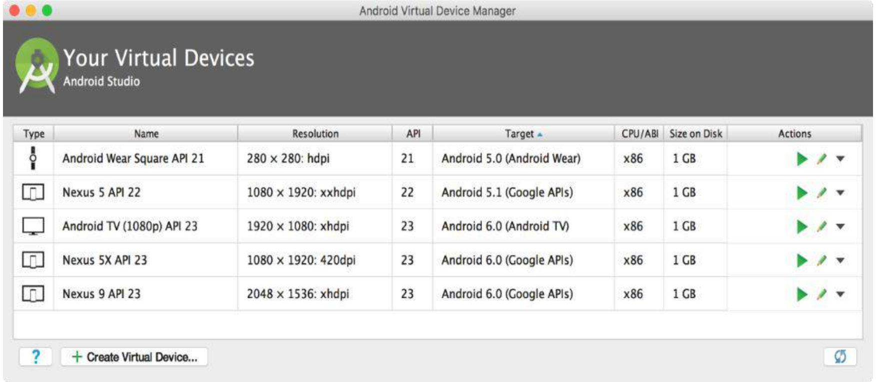
VIEWING AND MANAGING YOUR AVDS

The AVD Manager lets you manage your AVDs all in one place.

To run the AVD Manager, do one of the following:

- In Android Studio, select **Tools** > **Android** > **AVD Manager**.
- Click **AVD Manager** in the toolbar.

The AVD Manager appears as shown in Figure 6.2.



Type	Name	Resolution	API	Target ▲	CPU/ABI	Size on Disk	Actions
	Android Wear Square API 21	280 × 280: hdpi	21	Android 5.0 (Android Wear)	x86	1 GB	▶ ✎ ▼
	Nexus 5 API 22	1080 × 1920: xxhdpi	22	Android 5.1 (Google APIs)	x86	1 GB	▶ ✎ ▼
	Android TV (1080p) API 23	1920 × 1080: xhdpi	23	Android 6.0 (Android TV)	x86	1 GB	▶ ✎ ▼
	Nexus 5X API 23	1080 × 1920: 420dpi	23	Android 6.0 (Google APIs)	x86	1 GB	▶ ✎ ▼
	Nexus 9 API 23	2048 × 1536: xhdpi	23	Android 6.0 (Google APIs)	x86	1 GB	▶ ✎ ▼

Figure 6.2 AVD manager

It displays any AVDs you have already defined. When you first install Android Studio, it creates one AVD. If you defined AVDs for Android Emulator 24.0.*x* or lower, you need to recreate them.

From this page, you can:

- Define a new AVD or hardware profile.
- Edit an existing AVD or hardware profile.
- Delete an AVD or hardware profile.
- Import or export hardware profile definitions.
- Run an AVD to start the emulator.
- Stop an emulator.
- Clear data and start fresh, from the same state as when you first ran the emulator.
- Show the associated AVD `.ini` and `.img` files on disk.
- View AVD configuration details that you can include in any bug reports to the Android Studio team.

Creating an AVD

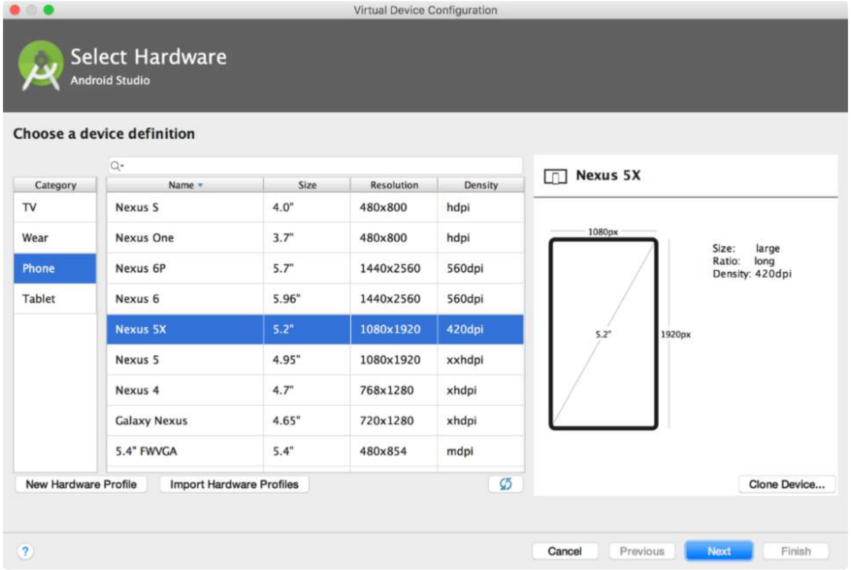
You can create a new AVD from the beginning, or duplicate an AVD and change some properties.

To create a new AVD:

- From the Your Virtual Devices page of the AVD Manager, click Create Virtual Device.

- Alternatively, run your app from within Android Studio. In the Select Deployment Target dialog, click Create New Emulator.

Then Select Hardware page appears as shown in Figure 6.3



Virtual Device Configuration

Select Hardware

Android Studio

Choose a device definition

Category	Name	Size	Resolution	Density
TV	Nexus S	4.0"	480x800	hdpi
Wear	Nexus One	3.7"	480x800	hdpi
Phone	Nexus 6P	5.7"	1440x2560	560dpi
Tablet	Nexus 6	5.96"	1440x2560	560dpi
	Nexus 5X	5.2"	1080x1920	420dpi
	Nexus 5	4.95"	1080x1920	xxhdpi
	Nexus 4	4.7"	768x1280	xhdpi
	Galaxy Nexus	4.65"	720x1280	xhdpi
	5.4" FWVGA	5.4"	480x854	mdpi

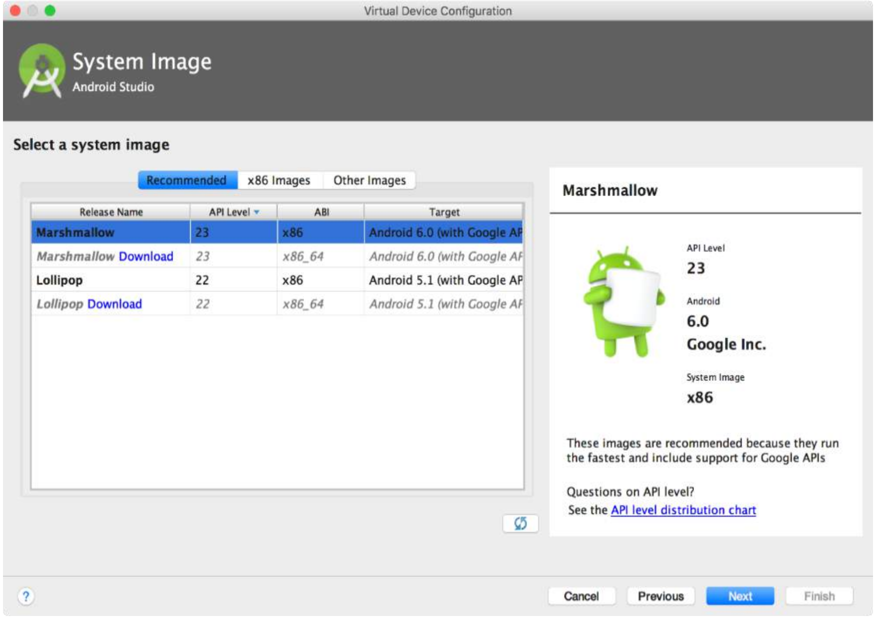
Nexus 5X

Size: large
Ratio: long
Density: 420dpi

Figure 6.3: Virtual Device Configuration(Select Hardware)

- Select a hardware profile, and then click ‘Next’.

If you don't see the hardware profile you want, you can create or import a hardware profile. Then System Image page appears as shown in Figure 6.4.



Select a system image

Release Name	API Level	ABI	Target
Marshmallow	23	x86	Android 6.0 (with Google AP
Marshmallow Download	*23*	*x86_64*	*Android 6.0 (with Google AP*
Lollipop	22	x86	Android 5.1 (with Google AP
Lollipop Download	*22*	*x86_64*	*Android 5.1 (with Google AP*

Marshmallow

API Level
23

Android
6.0
Google Inc.

System Image
x86

These images are recommended because they run the fastest and include support for Google APIs

Questions on API level?
See the API level distribution chart

Figure 6.4: Virtual Device Configuration (System Image)

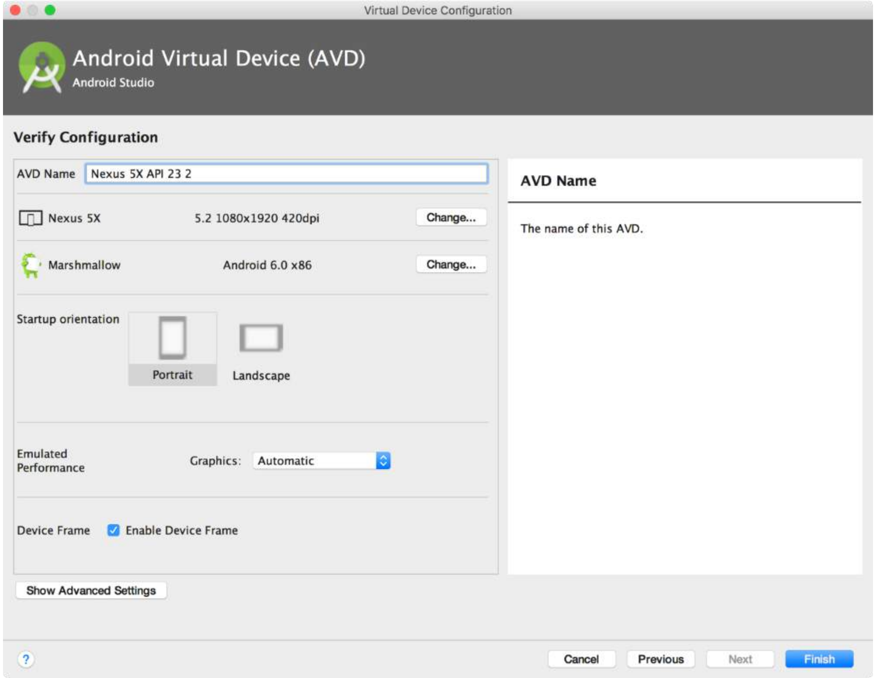
- Select the system image for a particular API level, and then click ‘Next’.

The Recommended tab lists recommended system images. The other tabs include a more complete list. The right pane describes the selected system image. x86 images run the fastest in the emulator. If you see Download next to the system image, you need to click it to download the system image. You must be connected to the internet to download it.

The API level of the target device is important, because your app won't be able to run on a system image with an API level that's less than that required by your app, as specified in the minSdkVersion attribute of the app manifest file. For more information about the relationship between system API level and minSdkVersion, see Versioning Your Apps.

If your app declares a <uses-library> element in the manifest file, the app requires a system image in which that external library is present. If you want to run your app on an emulator, create an AVD that includes the required library. To do so, you might need to use an add-on component for the AVD platform; for example, the Google APIs add-on contains the Google Maps library.

The Verify Configuration page appears as shown in Figure 6.5.



Virtual Device Configuration

Android Virtual Device (AVD)

Android Studio

Verify Configuration

AVD Name: Nexus 5X API 23 2

Nexus 5X — 5.2 1080x1920 420dpi

Marshmallow — Android 6.0 x86

Startup orientation: Portrait / Landscape

Emulated Performance — Graphics: Automatic

Device Frame: ☑ Enable Device Frame

AVD Name

The name of this AVD.

Figure 6.5: Virtual Device Configuration(AVD)

- Change AVD properties as needed, and then click Finish.
- Click Show Advanced Settings to show more settings, such as the skin.

The new AVD appears in the Your Virtual Devices page or the Select Deployment Target dialog.

To create an AVD starting with a copy:

- From the Your Virtual Devices page of the AVD Manager, right-click an AVD and select Duplicate.

Or click Menu ▼ and select Duplicate.

The Verify Configuration page appears (See previous image).

Click Change or Previous if you need to make changes on the System Image and Select Hardware pages.

Make your changes, and then click Finish.

The AVD appears in the Your Virtual Devices page.

Running and Stopping an Emulator, and Clearing Data

From the *Your Virtual Devices* page, you can perform the following operations on an emulator:

- To run an emulator that uses an AVD, double-click the AVD. Or click Launch ► .
- To stop a running emulator, right-click an AVD and select **Stop**. Or click Menu ▼ and select **Stop**.
- To clear the data for an emulator, and return it to the same state as when it was first defined, right-click an AVD and select **Wipe Data**. Or click Menu ▼ and select **Wipe Data**.

Running on the actual device

When building an Android app, it's important that you always test your application on a real device before releasing it to users. This page describes how to set up your development environment and Android-powered device for testing and debugging on the device.

You can use any Android-powered device as an environment for running, debugging, and testing your applications. The tools included in the SDK make it easy to install and run your application on the device each time you compile. You can install your application on the device directly from Android Studio or from the command line with ADB. If you don't yet have a device, check with the service providers in your area to determine which Android-powered devices are available.

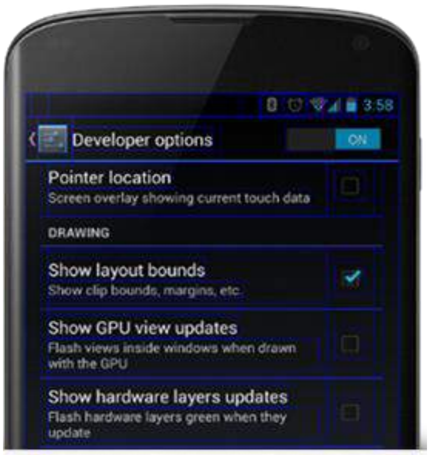
When building an Android app, it's important that you always test your application on a real device before releasing it to users. This page describes how to set up your development environment and Android-powered device for testing and debugging on the device.

Note: When developing on a device, keep in mind that you should still use the Android emulator to test your application on configurations that are not equivalent to those of your real device. Although the emulator does not allow you to test every device feature, it does allow you to verify that your application functions properly on different versions of the Android platform, in different screen sizes and orientations, and more.

Enable Developer mode

- Android-powered devices have a host of developer options that you can access on the phone, which let you:
- Enable debugging over USB.

- Quickly capture bug reports onto the device.
- Show CPU usage on screen.
- Draw debugging information on screen such as layout bounds, updates on GPU views and hardware layers, and other information.
- Plus many more options to simulate app stresses or enable debugging options.



3:58

Developer options ON

Pointer location ☐
Screen overlay showing current touch data

DRAWING

Show layout bounds ☑
Show clip bounds, margins, etc.

Show GPU view updates ☐
Flash views inside windows when drawn with the GPU

Show hardware layers updates ☐
Flash hardware layers green when they update

Figure 6.6: Enabling Developer Mode in Mobile

To access these settings, open the Developer options in the system Settings as shown in Figure 6.6. On Android 4.2 and higher, the Developer options screen is hidden by default. To make it visible, go to Settings > About phone and tap Build number seven times. Return to the previous screen to find Developer options at the bottom.

Running app on the actual device

- In Android Studio, select your project and click **Run** ▶ from the toolbar.
- In the **Select Deployment Target** window, select your device, and click **OK**.

Android Studio installs the app on your connected device and starts it.

Activity

Activity 6.3

- Build a simple app and run the application on
  1. Emulator
  2. Actual Device

Unit summary

In this unit, you learnt how to develop maintainable mobile apps that include the core Android components discussed in the previous unit. You need to watch the provided video on how to create an app from scratch using Android Studio. We also discussed about creating the Android Virtual Devices to run, testing and debugging the application.

Configuring and saving the launch configuration and associating an AVD with your project as discussed here will make your debugging and testing task easier. You also learnt about different project files that are created during the development process of an Android application as well as different development tools, Android application components and adding permissions to an application.

Unit 7

Device Compatibility

Introduction

Android is designed to run on many different types of devices, from phones to tablets and televisions. The range of devices provides a huge potential audience for the Android applications. In order to be successful on all these devices, it provides a flexible user interface that adapts to different screen configurations.

Android provides a dynamic app framework that can provide configuration-specific app resources in static files such as different XML layouts for different screen sizes. Android loads the appropriate resources based on the current device configuration. With some additional app resources, developer can publish a single application package (APK) that provides an optimized user experience on a variety of devices.

This unit will be focused on device compatibility. There are two types of compatibility *device compatibility* and *app compatibility*.

Hardware manufacturer can build a device that runs the Android operating system. Yet, a device is **"Android compatible"** only if it can correctly run apps written for the *Android execution environment* and each device must pass the Compatibility Test Suite (CTS) in order to be considered compatible.

Though Android runs on a wide range of device configurations, some features are not available on all devices. For example, some devices may not include a compass sensor. If your app's core functionality requires the use of a compass sensor, then your app is compatible only with devices that include a compass sensor.

Upon completion of this unit you should be able to:

Outcomes

- identify compatibility of an application with different devices.
- discuss screen configuration for various device sizes and resolutions
- evaluate device compatibility of an application

Terminology

Widget:	an application, or a component of an interface, that enables a user to perform a function or access a service
platform:	where any piece of software is executed

screen density: quantity of pixels within a physical area of the screen

resolution: The total number of physical pixels on a screen

7.1 Application availability to devices

Android supports a variety of features your app can control through platform APIs. Some features are hardware-based such as a compass sensor, some are software-based such as app widgets, and some features dependent on the platform version. You have to control application availability to the devices based on the features of your application because not every device supports every feature.

To achieve the largest user-base possible for an app, developer should strive to support as many device configurations as possible using a single APK. In most situations, it can do by disabling optional features at runtime and providing app resources with alternatives for different configurations.

Device characteristics are Device features, Platform version and Screen configuration.

Activity

Activity 7.1

State the importance of device configuration to maintain device compatibility

7.2 Device Features

In order to manage your app’s availability based on device features, Android defines *feature IDs* for any hardware or software feature that may not be available on all devices.

For instance, you can prevent users from installing your app when their devices do not provide a given feature by declaring it with a <uses-feature> element in your app's manifest file.

Example: Declare the compass sensor

If your app does not make sense on a device that lacks a compass sensor, you can declare the compass sensor as required with the following manifest tag as shown in the code snippet below.

```
<manifest ... >
    <uses-
featureandroid:name="android.hardware.sensor.com
pass"
                  android:required="true"/>
    ...
</manifest>
```

Google Play Store compares the features that your app requires to the features available on each user's device to determine whether your app is compatible with that device. If the device does not provide all the features your app requires, the user cannot install your app.

However, if your app's primary functionality does not require a device feature, you should set the required attribute to "false" and check for the device feature at runtime. If the app feature is not available on the current device, gracefully degrade the corresponding app feature. For example, you can query whether a feature is available by calling hasSystemFeature() like this:

```
PackageManager pm = getPackageManager();
if(!pm.hasSystemFeature(PackageManager.FEATURE_SENSOR_
COMPASS)){
// This device does not have a compass, turn off the
//compass feature
    disableCompassFeature();
}
```

7.3 Platform Version

Different devices may run different versions of the Android platform, such as Android 4.0 or Android 6.0. Each successive platform version often adds new APIs not available in the previous version. To indicate which set of APIs are available, each platform version specifies an API level.

For instance, Android 1.0 is API level 1 and Android 6.0 is API level 23

The API level allows you to declare the minimum version with which your app is compatible, using the <uses-sdk>manifest tag and its minSdkVersion attribute.

For example:

The Calendar Provider APIs were added in Android 4.0 (API level 14). If your app cannot function without these APIs, you should declare API level 14 as your app's minimum supported version like this:

```
<manifest ... >
    <uses-
sdkandroid:minSdkVersion="14"android:targetSdkVe
rsion="19"/>
    ...
</manifest>
```

The minSdkVersion attribute declares the minimum version with which your app is compatible and the targetSdkVersion attribute declares the highest version on which you have optimized your app. Each successive version of Android provides compatibility for apps that were built using the APIs from previous platform versions, so your app should always be compatible with future versions of Android while using the documented Android APIs.

However, if your app uses APIs added in a more recent platform version, but does not require them for its primary functionality, you should check the API level at runtime and gracefully degrade the corresponding features when the API level is too low.

In this case, set the minSdkVersion to the lowest value possible for your app's primary functionality, then compare the current system's version, SDK_INT, to one the codename constants in

Build.VERSION_CODES that corresponds to the API level you want to check.

For example:

```
if(Build.VERSION.SDK_INT
<Build.VERSION_CODES.HONEYCOMB){
    // Running on something older than API level
11, so disable
    // the drag/drop features that use
ClipboardManager APIs
    disableDragAndDrop();
}
```

Activity

Activity 7.2

Discuss different devices and different versions of the Android platform

Now we will see how Android runs on devices of various sizes.

7.4 Screen Configuration

Android runs on devices of various sizes, from phones to tablets and TVs. In order to categorize devices by their screen type, Android defines characteristics for each device:

- **Screen size -**The physical size of the screen. Actual physical size, measured as the screen's diagonal.
- For simplicity, Android groups all actual screen sizes into four generalized sizes: small, normal, large, and

extra-large.

- **Screen density** -. The quantity of pixels within a physical area of the screen; usually referred to as dpi (dots per inch). For example, a "low" density screen has fewer pixels within a given physical area, compared to a "normal" or "high" density screen.
- For simplicity, Android groups all actual screen densities into six generalized densities: low, medium, high, extra-high, extra-extra-high, and extra-extra-extra-high.
- **Resolution**- The total number of physical pixels on a screen. When adding support for multiple screens, applications do not work directly with resolution; applications should be concerned only with screen size and density, as specified by the generalized size and density groups.

A set of six generalized densities

- ldpi (low) ~120dpi
- mdpi (medium) ~160dpi
- hdpi (high) ~240dpi
- xhdpi (extra-high) ~320dpi
- xxhdpi (extra-extra-high) ~480dpi

Density-independent pixel (dp)

A virtual pixel unit that you should use when defining UI layout, to express layout dimensions or position in a density-independent way.

The density-independent pixel is equivalent to one physical pixel on a 160 dpi screen, which is the baseline density assumed by the system for a "medium" density screen.

At runtime, the system transparently handles any scaling of the dp units, as necessary, based on the actual density of the screen in use. The conversion of dp units to screen pixels is simple: px = dp * (dpi / 160).

For example, on a 240 dpi screen, 1 dp equals 1.5 physical pixels. You should always use dp units when defining your application's UI, to ensure proper display of your UI on screens with different densities.

By default, your app is compatible with all screen sizes and densities, because the system makes the appropriate adjustments to your UI layout and image resources as necessary for each screen. However, you should optimize the user experience for each screen configuration by adding specialized layouts for different screen sizes and optimized bitmap images for common screen densities.

Use wrap_content and match_parent

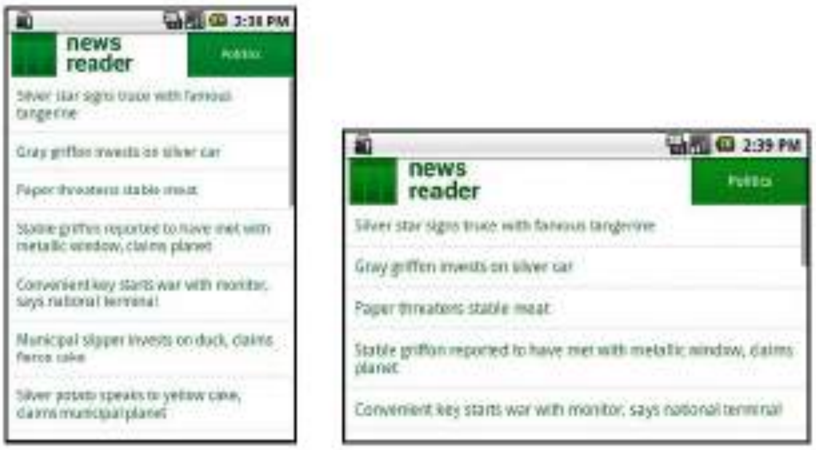
To ensure that your layout is flexible and adapts to different screen sizes, you should use "wrap_content" and "match_parent" for the width and height of some view components. If you use "wrap_content", the width or height of the view is set to the minimum size necessary to fit the content within that view, while "match_parent" makes the component expand to match the size of its parent view. By using
the "wrap_content" and "match_parent" size values instead of hard-coded sizes, your views either use only the space required for that view or expand to fill the available space, respectively.

For example:

```
<LinearLayoutxmlns:android="http://schemas.android.com
/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayoutandroid:layout_width="match_parent"
                  android:id="@+id/linearLayout1"
                  android:gravity="center"
                  android:layout_height="50dp">
        <ImageViewandroid:id="@+id/imageView1"
                   android:layout_height="wrap_content
"
                   android:layout_width="wrap_content"
                   android:src="@drawable/logo"
                   android:paddingRight="30dp"
                   android:layout_gravity="left"
                   android:layout_weight="0"/>
        <Viewandroid:layout_height="wrap_content"
              android:id="@+id/view1"
              android:layout_width="wrap_content"
              android:layout_weight="1"/>
        <Buttonandroid:id="@+id/categorybutton"
                android:background="@drawable/button_b
g"
                android:layout_height="match_parent"
                android:layout_weight="0"
                android:layout_width="120dp"
                style="@style/CategoryButtonStyle"/>
    </LinearLayout>

    <fragmentandroid:id="@+id/headlines"
              android:layout_height="fill_parent"
              android:name="com.example.android.newsre
ader.HeadlinesFragment"
              android:layout_width="match_parent"/>
</LinearLayout>
```

Notice how the sample uses "wrap_content" and "match_parent" for component sizes rather than specific dimensions. This allows the layout to adapt correctly to different screen sizes and orientations.

For example, figure 7.1 shows what this layout looks like in portrait and landscape mode. Notice that the sizes of the components adapt automatically to the width and height:



2:38 PM

news reader

Silver star signs truce with famous tangerine

Gray griffon invests on silver car

Paper threatens stable meat

Stable griffon reported to have met with metallic window, claims planet

Convenient key starts war with monitor, says national terminal

Municipal slipper invests on duck, claims fierce cake

Silver potato speaks to yellow cake, claims municipal planet

2:39 PM

news reader

Silver star signs truce with famous tangerine

Gray griffon invests on silver car

Paper threatens stable meat

Stable griffon reported to have met with metallic window, claims planet

Convenient key starts war with monitor, says national terminal

Figure 7.1: News Reader sample app in portrait (left) and landscape (right)
(Source:https://developer.android.com/training/multiscreen/screensizes.html)

Video- V7: Device Compatibility

Let us watch the video on device compatibility and do the activity 7.3.

URL: **https://tinyurl.com/ydcl9trq**

Activity

Activity 7.3

Calculate resolution values for your mobile device using density independent pixels.

Unit summary

Summary

In this unit, we discussed device compatibility and application availability to devices based on the device characteristics. These characteristics include device features, Platform version and Screen configuration. Furthermore, Android defines characteristics for each device such as screen size, density and resolution.

Unit 8

User Interface Design

Introduction

This unit will focus on theory of User Interface (UI) Design and at the end of this unit you will be able to build a user interface using Android layouts for different types of devices. Hence, this unit helps you to create an application that is smooth and responsive by using best practices for graphical user interface (GUI) design.

Upon completion of this unit you should be able to:

Outcomes

- design Graphical User Interface (GUI) using Extended Markup Language(XML)
- use best practices for GUI design
- apply layouts to improve application performance

Terminology

attributes:	a piece of information which describes the properties of a field
layout:	arrangemnet or the plan
adapter:	converts one type of software to another type
view:	object that user can interact.

8.1 UI Overview

All user interface elements in an Android app are built using *View* and *ViewGroup* objects.

Android provides a collection of both View and ViewGroup subclasses that offer you common input controls (such as buttons and text fields) and various layout models (such as a linear or relative layout).

- **View** - is an object that draws something on the screen that the user can interact. UI widgets such as buttons or text fields.

- **ViewGroup -** is an object that holds other View objects in order to define the layout of the interface. Invisible view containers that define how the child views are laid out, such as in a grid or a vertical list.

8.2 User Interface Layout

The user interface for each component of your app is defined using a hierarchy of View and ViewGroup objects, as shown in figure 8.1.

Each view group is an invisible container that organizes child views, while the child views may be input controls or other widgets that draw some part of the UI. This hierarchy tree can be as simple or complex as you need it to be (but simplicity is best for performance).

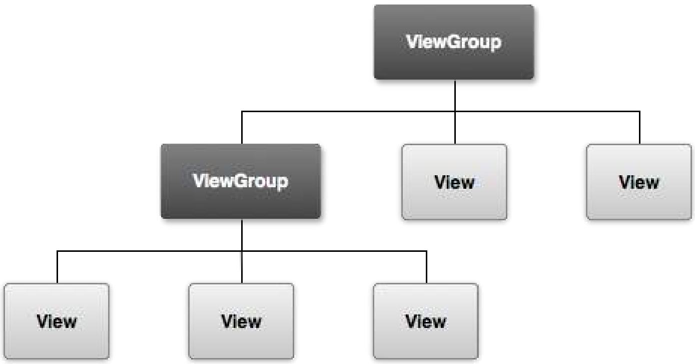


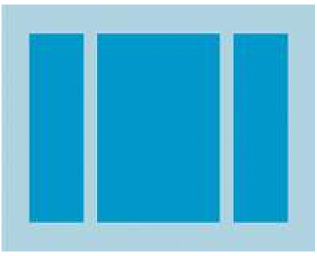
Figure 8.1.ViewGroup objects form branches in the layout withViewobjects

(Source: https://developer.android.com/guide/topics/ui/overview.html)

Common Layouts

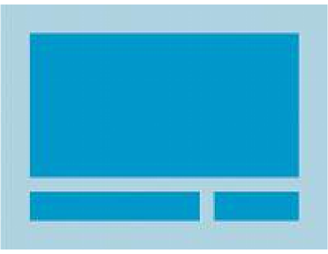
Each subclass of the ViewGroup class provides a unique way to display the views you nest within it. Some of the common layout types that are built into the Android platform are given below in figure 8.2. You can nest one or more layouts within another layout to achieve your UI design.

- Linear Layout



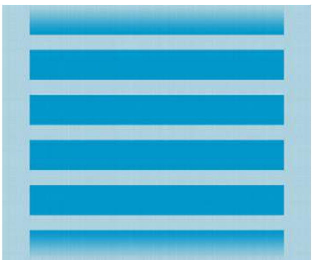
`LinearLayout` is a view group that aligns all its children in a single direction, vertically or horizontally. You can specify the layout direction with the android:orientation attribute. A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

- Relative Layout



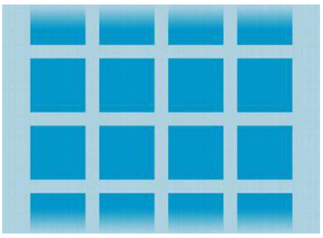
`RelativeLayout` is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent `RelativeLayout` area (such as aligned to the bottom, left or center).

- List View



`ListView` is a view group that displays a list of scrollable items. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database query and converts each item result into a view that is placed into the list.

- Grid View



`GridView` is a View Group that displays items in a two-dimensional, scrollable grid. The grid items are automatically inserted to the layout using a `ListAdapter.`

Figure 8.2: common layouts

(Source:https://developer.android.com/guide/topics/ui/layout/linear.html)

Video – V8: Creating GUI for Android Application

You may watch this screencast to see how to build a simple user interface while studying this unit.

URL: https://tinyurl.com/y9hkqpjb

Building Layouts with an Adapter

When the content for your layout is dynamic or not pre-determined, you can use a layout that subclasses AdapterView to populate the layout with views at runtime. A subclass of the AdapterView class uses an Adapter to bind data to its layout. The Adapter behaves as an intermediary between the data source and the AdapterView layout the Adapter retrieves the data (from a source such as an array or a database query) and converts each entry into a view that can be added into the AdapterView layout.

Common layouts backed by an adapter include:

- List View - Displays a scrolling single column list
- Grid View - Displays a scrolling grid of columns and rows.

A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. You can declare a layout in two ways:

- Declare UI elements in XML - Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- Instantiate layout elements at runtime - Your application can create View and ViewGroup objects programmatically.

Now we will see how to design the UI using XML.

8.3 Input Controls

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, checkboxes, zoom buttons, and toggle buttons. Adding an input control to your UI is as simple as adding an XML element to your XML layout.

Example: layout with a text field and button

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayoutxmlns:android="http://schemas.android.com
/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="horizontal">
<EditTextandroid:id="@+id/edit_message"
android:layout_weight="1"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:hint="@string/edit_message"/>
<Buttonandroid:id="@+id/button_send"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/button_send"
```

```
android:onClick="sendMessage"/>

</LinearLayout>
```

Common controls:

Table 8.1 gives a list of some common controls that you can use in your app.

Table 8.1: common controls

Control Type	Description	Related Classes
Button	A push-button that can be pressed, or clicked, by the user to perform an action.	Button
Text field	An editable text field. You can use the AutoCompleteTextView widget to create a text entry widget that provides auto-complete suggestions	EditText,AutoCompleteTextView
Checkbox	An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive.	CheckBox
Radio button	Similar to checkboxes, except that only one option can be selected in the group.	RadioGroup RadioButton
Toggle button	An on/off button with a light indicator.	ToggleButton
Spinner	A drop-down list that allows users to select one value from a set.	Spinner

8.4 Fundamentals of designing user interfaces using XML

To declare your layout, you can instantiate View objects in code and start building a tree, but the easiest and most effective way to define layout is with an XML file. XML offers a human-readable structure for the layout, similar to HTML.

Write the XML

Using Android's XML vocabulary, you can quickly design UI layouts and the screen elements they contain, in the same way you create web pages in HTML with a series of nested elements.

Each layout file must contain exactly one root element, which must be a View or ViewGroup object. Once you have defined the root element, you can add additional layout objects or widgets as child elements to build a View hierarchy that defines your layout.

Example: XML layout that uses a vertical LinearLayout to hold a TextView and a Button

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com
/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical">
    <TextViewandroid:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="I am a TextView"/>
    <Buttonandroid:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="I am a Button"/>
</LinearLayout>
```

After you have declared your layout in XML, save the file with the .xml extension, in your Android project's res/layout/ directory, so it will properly compile.

Load the XML Resource

When you compile your application, each XML layout file is compiled into a View resource. You should load the layout resource from your application code, in your Activity.onCreate() callback implementation. Do so by calling setContentView(), passing it the reference to your layout resource in the form of: R.layout.*layout_file_name*.

Example:XML layout saved as main_layout.xml

You need to load it for your Activity.

```
publicvoid onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

The onCreate() callback method in your Activity is called by the Android framework when your Activity is launched.

Attributes

Every View and ViewGroup object supports their own variety of XML attributes. Some attributes are specific to a View object (for example, TextView supports the textSize attribute), but these attributes are also inherited by any View objects that may extend this class. Few of these attributes are given below.

- ID attribute

Any View object may have an integer ID associated with it, to uniquely identify the View within the tree. When the application is compiled, this ID is referenced as an integer, but the ID is typically assigned in the layout XML file as a string, in the id attribute. This is an XML attribute common to all View objects (defined by the View class) and you will use it very often.

The syntax for an ID, inside an XML tag is:

```
android:id="@+id/my_button"
```

- Layout Parameters

XML layout attributes named layout_*something* define layout parameters for the View that are appropriate for the ViewGroup in which it resides. Every ViewGroup class implements a nested class that extends ViewGroup.LayoutParams. This subclass contains property types that define the size and position for each child view, as appropriate for the view group.

Figure 8.3 shows the hierarchy of layouts associated with each view.

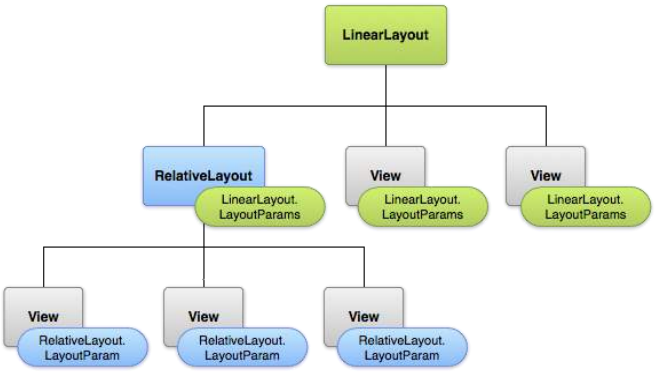


Figure 8.3. Hierarchy with layout parameters associated with each view. (Source:https://developer.android.com/guide/topics/ui/declaring-layout.html#CommonLayouts)

- Layout Position

The geometry of a view is that of a rectangle. A view has a location, expressed as a pair of *left* and*top* coordinates, and two dimensions, expressed as a width and a height. The unit for location and dimensions is the pixel. It is possible to retrieve the location of a view by invoking the methods getLeft() and getTop(). In addition, several convenience methods are offered to avoid unnecessary computations, namely getRight() and getBottom().

- Size, Padding and Margins

The size of a view is expressed with a width and a height. A view possesses two pairs of width and height values. The first pair is known as *measured width* and *measured height*. These dimensions define how big a view wants to be within its parent. The measured dimensions can be obtained by calling getMeasuredWidth() and getMeasuredHeight().

The next section will focus on common layout for Android application.

Activity

Activity 8.1

Create a Linear Layout following the given steps

Step 1: In Android Studio, from the res/layout directory, open the content_main.xml file.

The BlankActivity template you chose when you created this project includes the content_my.xml file with a RelativeLayout root view and aTextView child view.

Step 2: In the Preview pane, click the Hide icon ⇥ to close the Preview pane.

In Android Studio, when you open a layout file, you are first shown the Preview pane. Clicking elements in this pane opens the WYSIWYG tools in the Design pane. For this lesson, you are going to work directly with the XML.

Step 3: Delete the <TextView> element.

Step 4: Change the <RelativeLayout> element to <LinearLayout>.

Step 5: Add the android:orientation attribute and set it to "horizontal".

Step 6: Remove the android:padding attributes and the tools:context attribute.

The result looks like this:

```
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:showIn="@layout/activity_main">
```

As with every View object, you must define certain XML attributes to specify the EditText object's properties.

Activity

Activity 8.2

Add a Text Field to created layout following the given steps.

Step 1: In the content_my.xml file, within the <LinearLayout> element, define an <EditText> element with the id attribute set to @+id/edit_message.

Step 2: Define the layout_width and layout_height attributes as wrap_content.

Step 3: Define a hint attribute as a string object named edit_message.

The <EditText> element should read as follows:

```
<EditTextandroid:id="@+id/edit_message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit_message" />
```

Activity

Activity 8.3

Add a Button to created layout following the given steps.

Step 1: In Android Studio, from the res/layout directory, edit the content_my.xml file.

Step 2: Within the <LinearLayout> element, define a <Button> element immediately following the <EditText> element.

Step 3: Set the button's width and height attributes to "wrap_content" so the button is only as big as necessary to fit the button's text label.

Step 4: Define the button's text label with the android:text attribute; set its value to the button_send string resource you defined in the previous section.

Your <LinearLayout> should look like this:

```
<LinearLayoutxmlns:android="http://schemas.android.com
/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-
auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view
_behavior"
    tools:showIn="@layout/activity_my">
        <EditTextandroid:id="@+id/edit_message"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:hint="@string/edit_message"/>
        <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/button_send"/>
</LinearLayout>
```

The layout is currently designed so that both the EditText and Button widgets are only as big as necessary to fit their content, as given in figure 8.3.

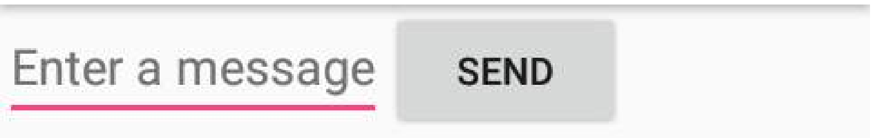


Figure 8.4. The EditText and Button widgets.

(Source: https://developer.android.com/training/basics/firstapp/building-ui.html)

Activity

Activity 8.4

What is the importance of XML-based layouts?

Explain LinearLayout in Android.

Now we will see how to design the same UI with the Layout Editor.

8.5 Design a UI with Layout Editor

Android Studio offers an advanced layout editor that allows you to drag and-drop widgets into your layout and preview your layout while editing the XML.

Within the layout editor, you can switch between the Text view, where you edit the XML file as text, and the Design view. Just click the appropriate tab at the bottom of the window to display the desired editor.

Editing in the Text View

While editing in the Text view, you can preview the layout on devices by opening the Preview pane and you can modify the preview by changing various options at the pane, including the preview device, layout theme, platform version and more. figure 8.5 gives how your application preview.

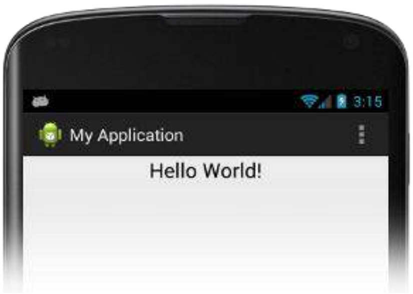


Figure 8.5 Previewing your app

(Source:https://developer.android.com/studio/write/layout-editor.html)

Next, we will focus on how you can switch to graphical editor and edit user interfaces in a design view.

Editing in the Design View

You can switch to the graphical editor by clicking Design at the bottom of the window. While editing in the Design view, you can show and hide the widgets available to drag-and-drop by clicking Palette on the window. Clicking Designer reveals a panel with a layout hierarchy and a list of properties for each view in the layout.

When you drag a widget into the graphical layout for your app, the display changes to help you place the widget. What you see depends on the type of layout.

Example:Dragging a widget into a FrameLayout

It displays a grid to help you place the widget, as shown in figure 8.6.

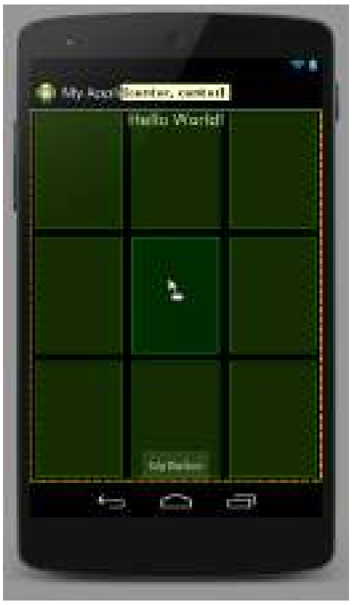


Figure 8.6 Grid layout to place a widget.

(Source: https://developer.android.com/studio/write/layout-editor.html#design-view)

In the next section, we will see how to manage touch events in an android application.

8.6 Managing Touch Events in a ViewGroup

Handling touch events in a ViewGroup takes special care, because it is common for a ViewGroup to have children that are targets for different touch events than the ViewGroup itself. To make sure that each view correctly receives the touch events intended for it, override the onInterceptTouchEvent() method.

Intercept Touch Events in a ViewGroup

The onInterceptTouchEvent() method is called whenever a touch event is detected on the surface of a ViewGroup, including on the surface of its children. If onInterceptTouchEvent() returns true, the MotionEvent is intercepted, meaning it will be not be passed on to the child, but rather to theonTouchEvent() method of the parent.

The onInterceptTouchEvent() method gives a parent the chance to see any touch event before its children do.

In the following snippet, the class MyViewGroup extends ViewGroup. MyViewGroup contains multiple child views. If you drag your finger across a child view horizontally, the child view should no longer get touch events, and MyViewGroup should handle touch events by scrolling its contents.

However, if you press buttons in the child view, or scroll the child view

vertically, the parent shouldn't intercept those touch events, because the child is the intended target. In those
cases, onInterceptTouchEvent() should return false,
and MyViewGroup'sonTouchEvent() won't be called.

```
publicclassMyViewGroupextendsViewGroup{

    privateint mTouchSlop;
    ...
    ViewConfiguration vc
=ViewConfiguration.get(view.getContext());
    mTouchSlop = vc.getScaledTouchSlop();

    ...

    @Override
    publicboolean onInterceptTouchEvent(MotionEvent
ev){


/* This method JUST determines whether we want to intercept the
motion.
* If we return true, onTouchEvent will be called and we do the
actual scrolling there.       */

        finalint action =
MotionEventCompat.getActionMasked(ev);

// Always handle the case of the touch gesture being //complete.
        if(action ==MotionEvent.ACTION_CANCEL ||
action ==MotionEvent.ACTION_UP){
            // Release the scroll.
            mIsScrolling =false;
            returnfalse;

// Do not intercept touch event, let the child handle it
        }
        switch(action){
            caseMotionEvent.ACTION_MOVE:{
                if(mIsScrolling){
// We're currently scrolling, so yes, intercept the
                    // touch event!
                    returntrue;
                }

// If the user has dragged her finger horizontally more
// than the touch slop, start the scroll
// left as an exercise for the reader



                finalint xDiff =
calculateDistanceX(ev);

// Touch slop should be calculated using ViewConfiguration
                // constants.
                if(xDiff > mTouchSlop){
                    // Start scrolling!
                    mIsScrolling =true;
```

```
                         returntrue;
                    }
                    break;}
               ...}
// In general, we don't want to intercept touch events.
//They should be handled by the child view.


          return false;
}
    @Override
     publicboolean onTouchEvent(MotionEvent ev){

// Here we actually handle the touch event (e.g. if the
// action is ACTION_MOVE, scroll this container).
// This method will only be called if the touch event was //
intercepted in onInterceptTouchEvent
          ...}}
```

In this section, we discussed managing touch Events in a ViewGroup. Next we will see what are the best practices of UI.

8.7 Best Practices for User Interface

Android provides a flexible framework for UI design that allows your app to display different layouts for different devices, create custom UI widgets, and even control aspects of the system UI outside your app's window.

- Designing for Multiple Screens - A user interface that's flexible enough to fit perfectly on any screen and create different interaction patterns that are optimized for different screen sizes.
- Adding the App Bar - Use the support library's toolbar widget to implement an app bar that displays properly on a wide range of devices.
- Showing Pop-Up Messages - Use the support library's Snackbar widget to display a brief pop-up message.
- Creating Custom View - Build custom UI widgets that are interactive and smooth.
- Creating Backward-Compatible UIs - Use UI components and other APIs from the more recent versions of Android while remaining compatible with older versions of the platform.
- Implementing Accessibility - Make apps accessible to users with vision impairment or other physical disabilities.
- Managing the System UI - Hide and show status and navigation bars across different versions of Android, while managing the display of other screen components.

- Creating Apps with Material Design - Implement material design on Android.

Unit summary

In this unit, we discussed the fundamentals of user interface design. In addition, we discussed how to build a user interface using Android layouts and the best practices for user interface design. Android provides a flexible framework for UI design. So in this unit we discussed how to display an application in different layouts for different devices and how to create custom UI widgets.

Unit 9

Testing and Debugging

Introduction

This unit emphasizes the importance of the application testing and debugging. It provides you an opportunity to identify errors and faults in a developed application. It also introduces how to use testing tools and techniques to test Android Application and how to apply measures to rectify identified errors and faults.

Upon completion of this unit you should be able to:

Outcomes

- differentiate testing and debugging an application
- set up the testing environment to develop Android applications
- write unit tests to test your Android programme
- perform debugging referring to log messages in Logcat

Terminology

error:	mistake in the program
fault:	usually a hardware problem happening at run time
emulator:	hardware or software that enables one computer system to behave like another

9.1 What is Testing?

Testing is one of the phases in software development life cycle that requires substantial amount of time before releasing software to users. To find the software bugs we use the process of executing a program or application. We have to test the software with test data to verify that a given set of input to a given function produces the expected result. There are two testing approaches; static and dynamic testing. An introduction to static and dynamic testing is given in the next section

Static and Dynamic testing

Static testing is done basically to test the requirement specifications, test plan, user manual etc. They are not executed, but tested with the set of some tools and processes. Reviews, walkthroughs and inspections are some example processes. These processes are not discussed in this material.

Dynamic Testing is when execution is done on the software code as a technique to detect defects and to determine quality attributes of the code.

With dynamic testing methods software is executed using a set of inputs and its output and then compared with the expected results. There are various levels of dynamic testing techniques. Some of them are unit testing, integration testing, system testing and acceptance testing. Here we will be only focusing on how different dynamic testing techniques can be used in an Android application.

Now you know that techniques can be used in testing software. Let's learn how to test an Android application using above techniques.

9.2 How to test Android application?

Testing an application on multiple physical devices at one place is not practical. Testing an application to run on different devices is referred to as device compatibility which is discussed in detail in a different unit. In order to avoid this practical barrier, Android SDK provides an emulator to test your application against all versions of Android on different devices. An emulator provides a virtual environment to test your application. It eliminates the requirement of a real physical device. This emulator uses an Android Virtual Device (AVD) to run an application. In order to do that it is required to create an AVD. Creating an AVD was discussed under an earlier section.

In addition, Amazon and various other companies maintain device farms to test applications with automation as well as manual testing.

An Android application should be tested for its functionality, user interfaces, performance etc. based on the generated test cases. It is application developers' responsibility to perform the unit tests and a separate testing team will be responsible of performing certain other types of testing such as functional testing, integration testing, acceptance testing etc.

9.3 Unit Testing

Unit tests are the fundamental tests in your app testing strategy. By creating and running unit tests against your code, you can easily verify that the logic of individual units is correct. Running unit tests after every build helps you to quickly catch and fix software regressions introduced by code changes to your app.

A unit test generally exercises the functionality of the smallest possible unit of code (which could be a method, class, or component) in a repeatable way. You should build unit tests when you need to verify the logic of specific code in your app. For example, if you are unit testing a class, your test might check that the class is in the right state. Typically, the unit of code is tested in. After completing this section, you will be able to perform unit testing for your application.

Android unit tests are based on JUnit and to test Android apps, you typically create these types of automated unit tests:

- Local tests: Unit tests that run on your local machine only. These tests are compiled to run locally on the Java Virtual Machine (JVM) to minimize execution time. Use this approach to run unit tests that have no dependencies on the Android framework or have dependencies that can be filled by using mock objects.
- Instrumented tests: Unit tests that run on an Android device or emulator. These tests have access to instrumentation information, such as theContext for the app under test. Use this approach to run unit tests that have Android dependencies which cannot be easily filled by using mock objects.

You should write your unit or integration test class as a JUnit 4test class in JUnit framework. This framework offers a convenient way to perform common setup, teardown, and assertion operations in your test.

Activity

Activity 9.1

Differentiate the use of local test from instrumented test when performing a unit test of an Android application.

9.4 How to set up your Testing Environment?

In your Android Studio project, you must store the source files for instrumented tests at *module-name*/src/androidTests/java/. This directory already exists when you create a new project.

Before you begin, you should download the Android Testing Support Library Setup, which provides APIs that allow you to quickly build and run instrumented test code for your apps. The Testing Support Library includes a JUnit 4 test runner (AndroidJUnitRunner) and APIs for functional UI tests (Espressoand UI Automator).

You also need to configure the Android testing dependencies for your project to use the test runner and the rules APIs provided by the Testing Support Library. To simplify your test development, you should also include the Hamcrest library, which lets you create more flexible assertions using the Hamcrest matcher APIs.

In your app's top-level build.gradle file, you need to specify these libraries as dependencies:

```
dependencies {
    androidTestCompile 'com.android.support:support-
annotations:24.0.0'
    androidTestCompile
'com.android.support.test:runner:0.5'
    androidTestCompile
'com.android.support.test:rules:0.5'
    // Optional -- Hamcrest library
    androidTestCompile 'org.hamcrest:hamcrest-
library:1.3'
    // Optional -- UI testing with Espresso
    androidTestCompile
'com.android.support.test.espresso:espresso-
core:2.2.2'
    // Optional -- UI testing with UI Automator
    androidTestCompile
'com.android.support.test.uiautomator:uiautomator-
v18:2.1.2'
}
```

To use JUnit 4 test classes, make sure to specify AndroidJUnitRunner as the default test instrumentation runner in your project by including the following setting in your app's module-level build.gradle file:

```
android {
    defaultConfig {
        testInstrumentationRunner
"android.support.test.runner.AndroidJUnitRunner"
    }
}
```

A basic JUnit 4 test class is a Java class that contains one or more test methods. A test method begins with the @Test annotation and contains the code to exercise and verify a single functionality (that is, a logical unit) in the component that you want to test.

The code snippet below shows an example JUnit 4 integration test that uses the Espresso APIs to perform a click action on a UI element, and check whether an expected string is displayed.

```
@RunWith(AndroidJUnit4.class)
@LargeTest
publicclassMainActivityInstrumentationTest{

    @Rule
    publicActivityTestRule mActivityRule
=newActivityTestRule<>(MainActivity.class);

    @Test
    publicvoid sayHello(){
       onView(withText("Sayhello!")).perform(click());

       onView(withId(R.id.textView)).check(matches(wit
hText("Hello, World!")));
    }
}
```

In your JUnit 4 test class, you can call out sections in your test code for special processing by using the following annotations:

@Before: Use this annotation to specify a block of code that contains test setup operations. The test class invokes this code block before each test. You can have multiple @Before methods but the order in which the test class calls these methods is not guaranteed.

@After: This annotation specifies a block of code that contains test tear-down operations. The test class calls this code block after every test method. You can define multiple @After operations in your test code. Use this annotation to release any resources from memory.

@Test: Use this annotation to mark a test method. A single test class can contain multiple test methods, each prefixed with this annotation.

@Rule: Rules allow you to flexibly add or redefine the behavior of each test method in a reusable way. In Android testing, use this annotation together with one of the test rule classes that the Android Testing Support Library provides, such as ActivityTestRule or ServiceTestRule.

@BeforeClass: Use this annotation to specify static methods for each test class to invoke only once. This testing step is useful for expensive operations such as connecting to a database.

@AfterClass: Use this annotation to specify static methods for the test class to invoke only after all tests in the class have run. This testing step is useful for releasing any resources allocated in the @BeforeClass block.

@Test(timeout=): Some annotations support the ability to pass in elements for which you can set values. For example, you can specify a timeout period for the test. If the test starts but does not complete within the given timeout period, it automatically fails. You must specify the timeout period in milliseconds, for example: @Test(timeout=5000).

Instrumented unit tests

Unit tests that run on an Android device or emulator can take advantage of the Android framework APIs and supporting APIs, such as the Android Testing Support Library. You should create instrumented unit tests if your tests need access to instrumentation information (such as the target app's Context) or if they require the real implementation of an Android framework component (such as a Parcelable or SharedPreferences object). These tests have access to Instrumentation information, such as the Context of the app you are testing. Use these tests when your tests have Android dependencies that mock objects cannot satisfy.

Because instrumented tests are built into a stand-alone APK, they must have an AndroidManifest.xml file. However, Gradle automatically generates this file during the build so it is not visible in your project source set. You can add your own manifest file if necessary, such as to specify a different value for `minSdkVersion` or register run listeners just for your tests. When building your app, Gradle merges multiple manifest files into one manifest.

Create an Instrumented Unit Test Class

Your instrumented unit test class should be written as a JUnit 4 test class. To learn more about creating JUnit 4 test classes and using JUnit 4 assertions and annotations, see Create a Local Unit Test Class.

To create an instrumented JUnit 4 test class, add the @RunWith(AndroidJUnit4.class) annotation at the beginning of your test class definition. You also need to specify the AndroidJUnitRunner class provided in the Android Testing Support Library as your default test runner. This step is described in more detail in Getting Started with Testing.

The following example shows how you might write an instrumented unit test to test that the Parcelable interface is implemented correctly for the LogHistory class:

```
import android.os.Parcel;
import android.support.test.runner.AndroidJUnit4;
import android.util.Pair;
import org.junit.Test;
import org.junit.runner.RunWith;
import java.util.List;
importstatic org.hamcrest.Matchers.is;
importstatic org.junit.Assert.assertThat;

@RunWith(AndroidJUnit4.class)
@SmallTest
publicclassLogHistoryAndroidUnitTest{

    publicstaticfinalString TEST_STRING ="This is a
string";
    publicstaticfinallong TEST_LONG =12345678L;
    privateLogHistory mLogHistory;

    @Before
    publicvoid createLogHistory(){
        mLogHistory =newLogHistory();
    }

    @Test
    publicvoid logHistory_ParcelableWriteRead(){
        // Set up the Parcelable object to send and
receive.
        mLogHistory.addEntry(TEST_STRING, TEST_LONG);

        // Write the data.
        Parcel parcel =Parcel.obtain();
        mLogHistory.writeToParcel(parcel,
mLogHistory.describeContents());

        // After you're done with writing, you need to
reset the parcel for reading.
        parcel.setDataPosition(0);

        // Read the data.
        LogHistory createdFromParcel
=LogHistory.CREATOR.createFromParcel(parcel);
        List<Pair<String,Long>> createdFromParcelData
= createdFromParcel.getData();

        // Verify that the received data is correct.
        assertThat(createdFromParcelData.size(),is(1))
;
        assertThat(createdFromParcelData.get(0).first,
is(TEST_STRING));
        assertThat(createdFromParcelData.get(0).second
,is(TEST_LONG));
    }
}
```

Create a test suite

To organize the execution of your instrumented unit tests, you can group a collection of test classes in a *test suite* class and run these tests together. Test suites can be nested. That is your test suite can group other test suites and run all their component test classes together.

A test suite is contained in a test package, similar to the main application package. By convention, the test suite package name usually ends with the suite suffix (e.g. com. example.android.testing.mysample.suite).

To create a test suite for your unit tests, import the JUnit RunWith and Suite classes. In your test suite, add the @RunWith(Suite.class) and the @Suite.SuitClasses() annotations. In the @Suite.SuiteClasses()

annotation, list the individual test classes or test suites as arguments.

The following example shows how you might implement a test suite called UnitTestSuite that groups and runs the CalculatorInstrumentation-Test and CalculatorAddParameterizedTest test classes together.

```
import
com.example.android.testing.mysample.CalculatorAddPara
meterizedTest;
import
com.example.android.testing.mysample.CalculatorInstrum
entationTest;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

// Runs all unit tests.
@RunWith(Suite.class)
@Suite.SuiteClasses({CalculatorInstrumentationTest.cla
ss,
        CalculatorAddParameterizedTest.class})
publicclassUnitTestSuite{}
```

Run Instrumented Unit Tests

To run your instrumented tests, follow these steps:

1. Be sure your project is synchronized with Gradle by clicking **Sync Project** in the toolbar.
2. Run your test in one of the following ways:
   - To run a single test, open the **Project** window, and then right-click a test and click **Run** ▶.
   - To test all methods in a class, right-click a class or method in the test file and click **Run** ▶.
   - To run all tests in a directory, right-click on the directory and select **Run tests** ▶.

The Android Plugin for Gradle compiles the instrumented test code located in the default directory (src/androidTest/java/), builds a test APK and production APK, installs both APKs on the connected device or emulator, and runs the tests. Android Studio then displays the results of the instrumented test execution in the *Run* window.

Note: While running or debugging instrumented tests, Android Studio does not inject the additional methods required for Instant Run and turns the feature off.

By default, Android Studio sets up new projects to deploy to the Emulator or a physical device with just a few clicks. With Instant Run, you can push changes to methods and existing app resources to a running app without building a new APK, so code changes are visible almost instantly.

Instant Run

Introduced in Android Studio 2.0, Instant Run is a behavior for the Run and

Debug commands that significantly reduces the time between updates to your app. Although your first build may take longer to complete, Instant Run pushes subsequent updates to your app without building a new APK, so changes are visible much more quickly.

Instant Run is supported only when you deploy the debug build variant, use Android Plugin for Gradle version 2.0.0 or higher, and set minSdkVersion to 15 or higher in your app's module-level build.gradle file. For the best performance, set minSdkVersion to 21 or higher.

After deploying an app, a small, yellow thunderbolt icon appears within theRun button (or Debug button), indicating that Instant Run is ready to push updates the next time you click the button. Instead of building a new APK, it pushes just those new changes and, in some cases, the app doesn't even need to restart but immediately shows the effect of those code changes.

Instant Run pushes updated code and resources to your connected device or emulator by performing a hot swap, warm swap, or cold swap. It automatically determines the type of swap to perform based on the type of change you made.

Video – V9: Android Unit Testing

In this video you will be shown how to setup testing environment and how to write a unit test. You may watch this video and do activity 9.2.

URL: https://tinyurl.com/yaatacny

Activity

Activity 9.2

Create an Android application “`MyApp`” with a class “`ConversionUtil`” to perform the given two functionalities.

- To convert centimeters into inches [write a method `ConvertCmtoInch`()]
- To convert inches into centimeters [write a method `ConvertInchtoCm`()]

Then write local unit tests to check whether the written functionalities provide the expected output. Use the values given as inputs and expected output to test the method.

Functionality to test	**Input**	**Output**
Convert centimeters into inches	10 centimeters	3.93701 inches
Convert inches into centimeters	10 inches	25.4 centimeters

9.5 What is Debugging?

It is the procedure of finding defects in a source code and removing them. Android Studio includes a debugger that allows you to debug apps running on the Android Emulator or a connected Android device. With the Android Studio debugger, you can:

- Select a device to debug your app on.
- Set breakpoints in your code.
- Examine variables and evaluate expressions at runtime.
- Capture screenshots and videos of your app.

To start debugging, click **Debug** 🐞 in the toolbar. Android Studio builds an APK, signs it with a debug key, installs it on your selected device, then runs it and opens the **Debug** window.

If no devices appear in the **Select Deployment Target** window after you click **Debug**, then you need to either connect a device or click **Create New Emulator** to setup the Android Emulator.

Activity

Activity 9.3

How to enable USB debugging in your device?

9.6 What is Logcat?

Logcat is a command-line tool that dumps a log of system messages, including stack traces when the device throws an error and messages that you have written from your app with the Log class.

This page is about the command-line logcat tool, but you can also view log messages from the Logcat window in Android Studio. For information about viewing and filtering logs from Android Studio

You can run logcat as an adb command or directly in a shell prompt of your emulator or connected device. To view log output using adb, navigate to your SDK platform-tools/ directory and execute:

```
$ adb logcat
```

You can create a shell connection to a device and execute:

```
$ adb shell
# logcat
```

How to write Log Messages?

The Log class allows you to create log messages that appear in the logcat window. Generally, you should use the following log methods, listed in order from the highest to lowest priority (or, least to most verbose):

- Log.e→ (error)
- Log.w→(warning)
- Log.i→(information)
- Log.d →(debug)
- Log.v → (verbose)

You should never compile versbose logs into your app, except during development. Debug logs are compiled in but stripped at runtime, while error, warning and info logs are always kept.

For each log method, the first parameter should be a unique tag and the second parameter is the message. The tag of a system log message is a short string indicating the system component from which the message originates (for example, ActivityManager). Your tag can be any string that you find helpful, such as the name of the current class.

A good convention is to declare a TAG constant in your class to use in the first parameter. For example, you might create an information log message as follows:

```
private static final String TAG = "MyActivity";

...

Log.i(TAG, "MyClass.getView() – get item number " +
position);
```

Note: Tag names greater than 23 characters are truncated in the logcat output.

Unit summary

Android Studio is designed to make testing simple. This unit explained how to set up a JUnit test that runs on the local JVM or an instrumented test that runs on a device.

Unit 10

Integrating Multimedia

Introduction

This unit offers you with knowledge on how to integrate multimedia to Android applications. Further the unit discusses how to utilize multimedia to enhance the performance by selecting appropriate media formats for audio, video and images. You need to watch the provided video to get an insight of how different multimedia are being used in selected applications.

Upon completion of this unit you should be able to:

Outcomes

- write a code to play audio and video depending on the functional requirements.
- implement camera functions to capture photos.
- select appropriate media codecs to maximize the compatibility and application performance.

Terminology

multimedia:	use of graphics, animations, video clippings, audio etc taken together
state diagram:	diagram depicting various states of an app
codecs:	a device or program that compresses data to enable faster transmission and decompresses received data
compatibility:	state of being two or more things are able to exist or work together in combination
streaming media:	Streaming media is multimedia that is constantly received by and presented to an end-user while being delivered by a provider

10.1 Introduction to Multimedia

Multimedia is an effective tool of communication. Let us spend some time to think in which forms we access the information. The simplest and the most common of these is the printed text. Examples include newspapers, web pages etc. In order to deliver information in a more attractive way, text materials are supported with graphics, still pictures, animations, video clippings, audio commentaries and so on. Use of different attractive formats to convey information in a meaningful manner is termed as multimedia. Television is a very good example of a multimedia broadcasting system. Information is distributed to the community using audio and video signals.

Android applications are developed to distribute different information to the community. Unlike in a broadcasting media, such as the television, in Android mobile applications the users directly interact with it. Multimedia is necessary to improve the user interactions in a mobile application. Integration of different media formats can significantly improve the user experience when interacting with the mobile application.

10.2 Audio and Video Integration into Android Application Development

MediaPlayer and AudioManager are two classes available to play sound and video in the Android framework. MediaPlayer class is the primary API for playing sound and audio while AudioManager class is used to manage audio sources and audio output on a device. MediaCodec and MediaExtractor classes are provided for building custom media players. The open source project, ExoPlayer, is a solution between these two options, providing a pre-built player that you can extend.

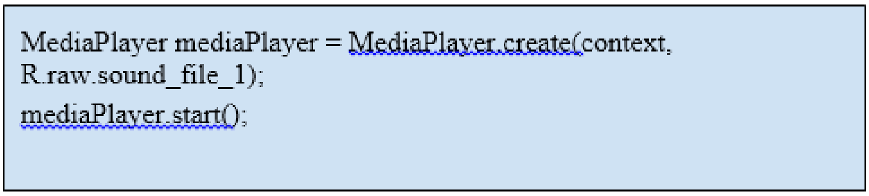
Video-V10: Multimedia for Android Interactive Application Development

By watching this video you will get an understanding how to incorporate multimedia to an Android application.

URL: https://tinyurl.com/y7bj7mys

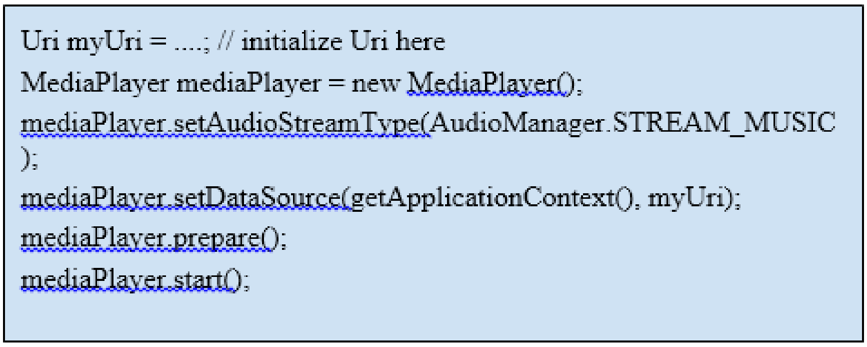
Media Player

An object of this class can fetch, decode, and play both audio and video with minimal setup. It supports several different media sources including local resources, Internet URIs, external URIs. Here is an example of how to play audio that's available as a local raw resource saved in your application's res/raw/ directory:



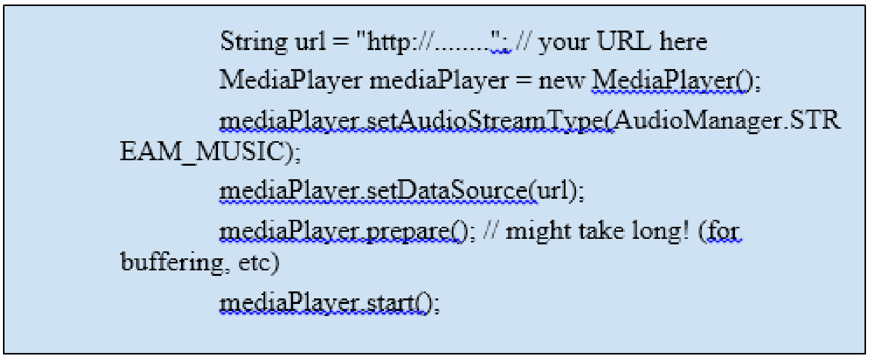
```
MediaPlayer mediaPlayer = MediaPlayer.create(context,
R.raw.sound_file_1);
mediaPlayer.start();
```

Here is an example on how you might play from a URI available locally in the system (that you obtained through a Content Resolver, for instance):



```
Uri myUri = ....; // initialize Uri here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC
);
mediaPlayer.setDataSource(getApplicationContext(), myUri);
mediaPlayer.prepare();
mediaPlayer.start();
```

Playing from a remote URL via HTTP streaming looks like this:

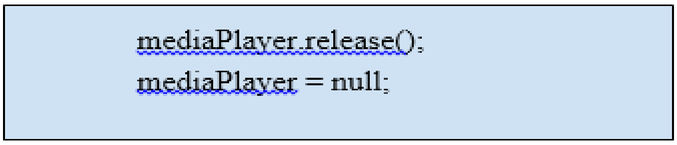


```
        String url = "http://........"; // your URL here
        MediaPlayer mediaPlayer = new MediaPlayer();
        mediaPlayer.setAudioStreamType(AudioManager.STR
EAM_MUSIC);
        mediaPlayer.setDataSource(url);
        mediaPlayer.prepare(); // might take long! (for
buffering, etc)
        mediaPlayer.start();
```

Before using the MediaPlayer, it is necessary to make the appropriate declarations. Example is to request permissions to access Internet for streaming applications. MediaPlayer will not work as expected in certain scenarios. Why does this happen? The possible reasons can be explained by understanding the state based representation of the MediaPlayer class. MediaPlayer has specific “states” (Figure10.1) in which certain operations are only valid. If you perform an operation while in the wrong state, the system may throw an exception or cause other undesirable behaviors. Schematic of the state based representation is shown below.

The state based representation of the MediaPlayer is shown in Figure10.1. The state diagram clarifies which methods move the MediaPlayer from one state to another. For example, when you create a new MediaPlayer, it is in the *Idle* state. Then, you should initialize it by calling setDataSource(), bringing it to the *Initialized* state. Next, you have to prepare it using either the prepare() or prepareAsync() method. When the MediaPlayer is done preparing, it will then enter the *Prepared* state, which indicates that start() can be called to make it play the media. Then, you can move between the *Started*, *Paused* and *PlaybackCompleted* states by calling such methods as start(), pause() and seekTo(), amongst others. Once you call stop(), MediaPlayer cannot call start() again until you prepare it again.

MediaPlayer consumes valuable system resources. Therefore, you should always take extra precautions to make sure you are not hanging on to a MediaPlayer instance longer than necessary. When you are done with it, you should always call release() to make sure any system resources allocated to it are properly released.



```
mediaPlayer.release();
mediaPlayer = null;
```

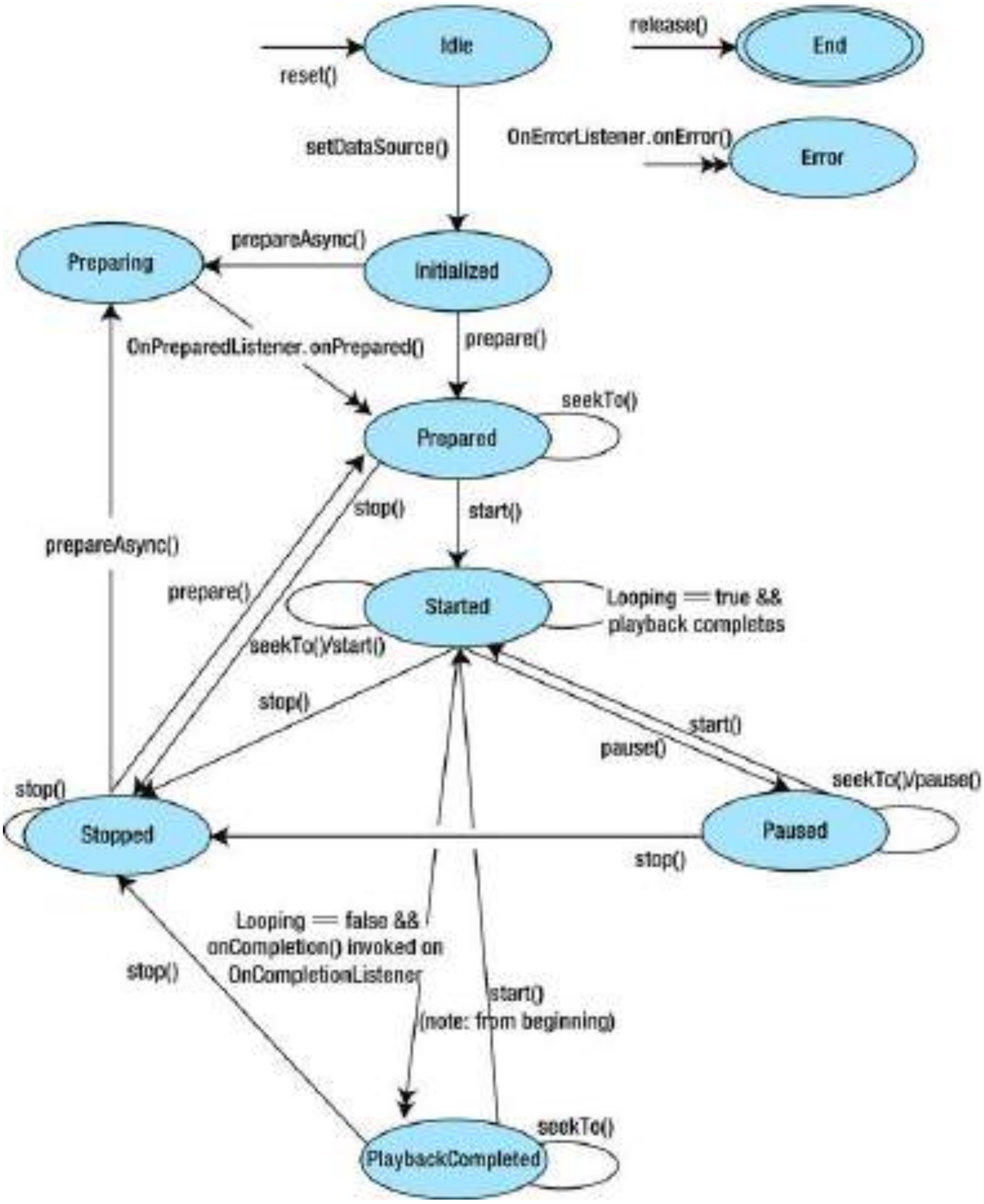


Figure10.1: State Diagram of MediaPlayer.
(Source: https://developer.android.com/index.html)

Activity

Activity 10.1

List five valid or invalid state transitions from the MediaPlayer by studying the state diagram shown in Figure10.1.

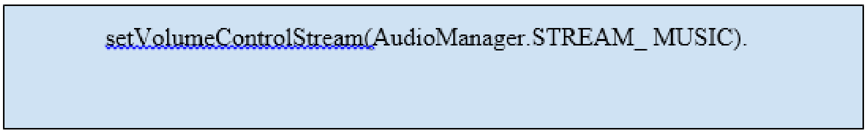
Volume and Playback Control

Depending on the user preference the loudness of the audio output should be manageable. Volume control should be available for the user by using the hardware or software volume controls of their device, bluetooth headset, or headphones. Apart from volume control, the user should also be able to have control on the playback videos. The control functions such as, the play, stop, pause, skip, and previous media playback keys should perform their respective actions on the audio stream used by the application you develop.

Audio streams - In order to control the audio output, the Android applications use different “streams”. An audio stream enables an application to independently and distinctly apply controls depending on the user preferences. The first step to creating a predictable audio experience is understanding which audio stream your app will use. For example, Android maintains a separate audio stream called STREAM_MUSIC for playing music, alarms, notifications, the incoming call ringer, system sounds, in-call volume, and DTMF tones. Most of the Android audio streams are restricted to system events.

Hardware Volume and Playback Controls

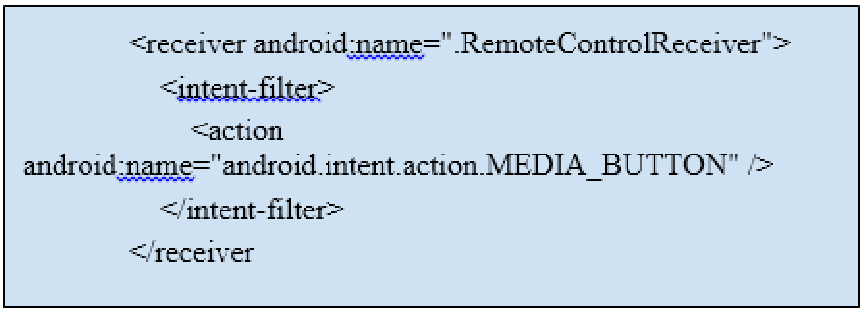
The hardware volume control button, generally adjust the ringer volume of the mobile phone. For example, you do not want the ringer volume to be adjusted by pressing hardware volume control key while playing a game, but the user only wants to increase the volume of the sounds played by the game. For this type of preferences, it is necessary to identify which audio stream to control. Android provides the setVolumeControlStream() method to direct volume key presses to the audio stream that is specified. Having identified the audio stream used in the application, this should be set as the volume stream target. This setting should be coded early in your application’s lifecycle as it is only needed to be called once, typically within the onCreate() method (of the Activity or Fragment that controls your media). This ensures that whenever your app is visible, the volume controls function as the user expects. The code snippet looks like this:



```
setVolumeControlStream(AudioManager.STREAM_ MUSIC).
```

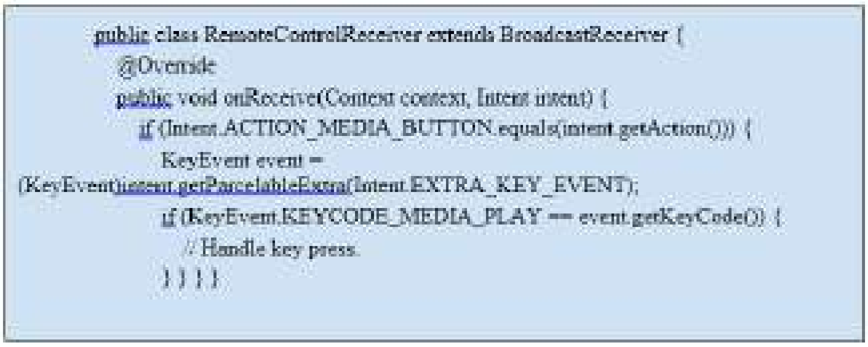
Once this is coded, when the user press the hardware volume keys on the device, the control affect the audio stream you specify whenever the target activity or fragment is visible.

On certain mobile devices, the hardware playback control buttons are available or even externally connected through wireless handsets. When the user presses one of these buttons, the Android application notifies it as a ACTION_MEDIA_BUTTON action. In order to respond to these actions, as the receiving end, a BroadcastReceiver should be registered (or declared). The code snippet would look like this:



```
        <receiver android:name=".RemoteControlReceiver">
           <intent-filter>
              <action
android:name="android.intent.action.MEDIA_BUTTON" />
           </intent-filter>
        </receiver
```

Once the receiver is defined, the appropriate response can only be generated if the receiver knows which playback button was pressed. To identify the button, Intent is used. Then, in order to provide the appropriate response action, KeyEvent class is used. An example is shown below.



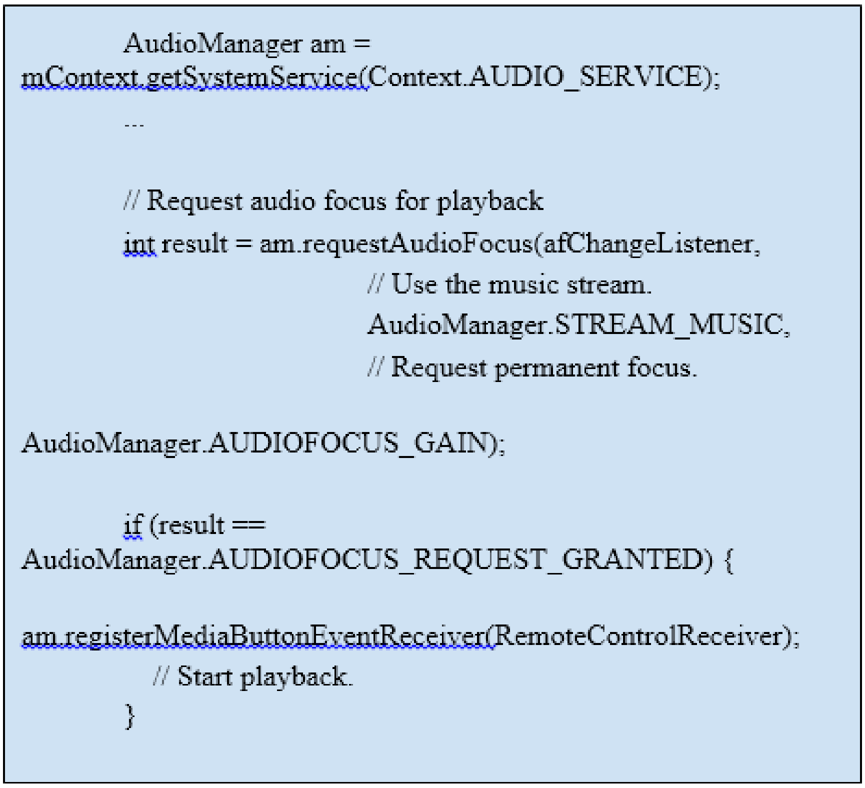
```
        public class RemoteControlReceiver extends BroadcastReceiver {
          @Override
          public void onReceive(Context context, Intent intent) {
            if (Intent.ACTION_MEDIA_BUTTON.equals(intent.getAction())) {
              KeyEvent event =
(KeyEvent)intent.getParcelableExtra(Intent.EXTRA_KEY_EVENT);
              if (KeyEvent.KEYCODE_MEDIA_PLAY == event.getKeyCode()) {
                // Handle key press.
              } } } }
```

Managing Audio Focus (*italics smaller font CHECK HEADINGS)

To avoid every music app playing at the same time, Android uses audio focus to allow only apps that hold the audio focus to play audio. Before your app starts playing any audio, it should hold the audio focus for the stream it will be using. This is done with a call to requestAudioFocus(). AUDIOFOCUS_REQUEST_GRA NTED is returned if the request is successful.

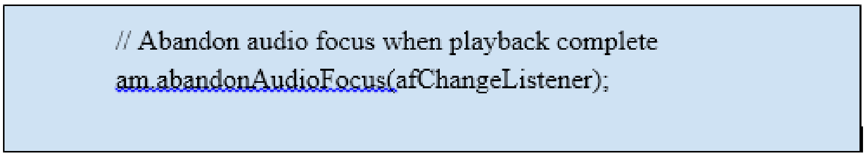
You must specify which stream you are using and whether you expect to require transient or permanent audio focus. Request transient focus when you expect to play audio for only a short time (for example when playing navigation instructions). Request permanent audio focus when you plan to play audio for the foreseeable future (for example, when playing music).

The following snippet requests permanent audio focus on the music audio stream. You should request the audio focus immediately before you begin playback, such as when the user presses play or the background music for the next game level begins.



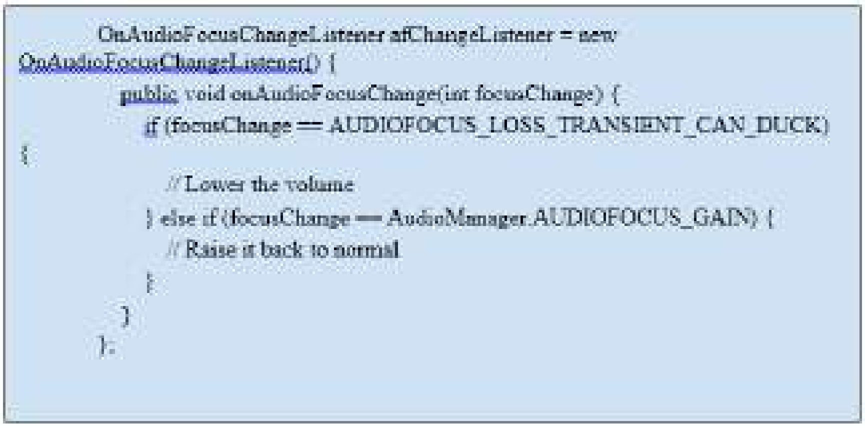
```
            AudioManager am =
mContext.getSystemService(Context.AUDIO_SERVICE);
            ...

            // Request audio focus for playback
            int result = am.requestAudioFocus(afChangeListener,
                                          // Use the music stream.
                                          AudioManager.STREAM_MUSIC,
                                          // Request permanent focus.

AudioManager.AUDIOFOCUS_GAIN);

            if (result ==
AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {

am.registerMediaButtonEventReceiver(RemoteControlReceiver);
               // Start playback.
            }
```

Once you have finished playback call abandonAudioFocus(). To notify the system that it is no longer require focus and unregisters the associated.OnAudioFocusChangeListener. In the case of abandoning transient focus, this allows any interrupted app to continue playback.



```
            // Abandon audio focus when playback complete
            am.abandonAudioFocus(afChangeListener);
```

Ducking is the process of lowering your audio stream output volume to make transient audio from another app easier to hear without totally disrupting the audio from your own application. In the following code snippet lowers the volume on our media player object when we temporarily lose focus, then returns it to its previous level when we regain focus.



```
        OnAudioFocusChangeListener afChangeListener = new
OnAudioFocusChangeListener() {
            public void onAudioFocusChange(int focusChange) {
                if (focusChange == AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK)
{
                    // Lower the volume
                } else if (focusChange == AudioManager.AUDIOFOCUS_GAIN) {
                    // Raise it back to normal
                }
            }
        };
```

A loss of audio focus is the most important broadcast to react to. A temporary loss of audio focus should result in your app silencing its audio stream, but otherwise maintaining the same state. You should continue to monitor changes in audio focus and be prepared to resume playback where it was paused once you have regained the focus. If the audio focus loss is permanent, it is assumed that another application is now being used to listen to audio and your app should effectively end itself. In practical terms, that means stopping playback, removing media button listeners—allowing the new audio player to exclusively handle those events—and abandoning your audio focus. At that point, you would expect a user action (pressing play in your app) to be required before you resume playing audio.

When the playback functions need to be implemented, it is important for the application developers to know the existing protocols supported in the Android framework. The following are the network protocols are supported for audio and video playback in Android framework:

- RTSP (RTP, SDP)
- HTTP/HTTPS progressive streaming
- HTTP/HTTPS live streaming draft protocol:
  - MPEG-2 TS media files only
  - Protocol version 3 (Android 4.0 and above)
  - Protocol version 2 (Android 3.x)
  - Not supported before Android 3.0

Activity

Activity 10.2

Identify the audio and video multimedia functions used in one of the following Android apps: XfinityTV, Google Music, Ustream, Netflix.

Consider the technical aspects we have covered in earlier sections.

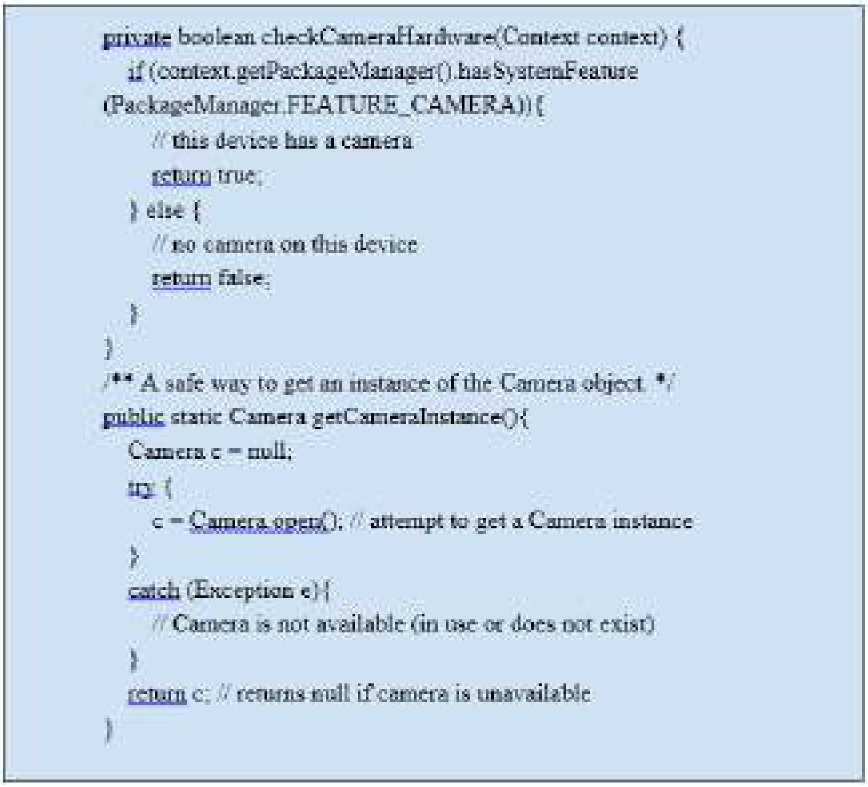
Next, we will discuss the camera function usage in Android application development.

10.3 Camera functions in Android Application Development

The Android framework includes support for various cameras and camera features available on mobile devices to capture pictures and videos in your applications.

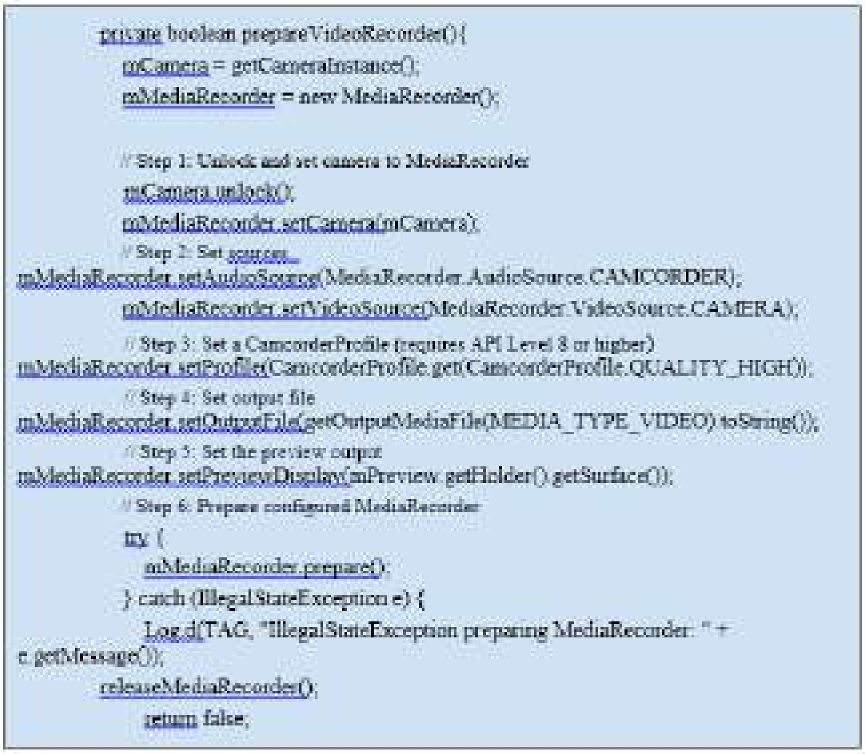
Android class Android.hardware.camera2 API primarily supports capturing images and videos. In addition, Camera, Intent, SurfaceView, MediaRecorder, classes are available. When developing an Android application it is important to declare the necessary permissions and features. A major advantage in declaring the features is to make sure that Google Play will only allow to install a specific application only on the mobile devices with those features of the camera. In addition to storage permission, depending on whether it is an external or internal memory should be specified during the application development. Also, if the application has a location tagging feature, in order to support this appropriate permissions to get the location information should be declared with associated permissions.

In an application, whether a camera is available on the device can be verified at runtime. Then, the camera is accessed through an instance of it. An example code is shown below.



```
private boolean checkCameraHardware(Context context) {
  if (context.getPackageManager().hasSystemFeature
(PackageManager.FEATURE_CAMERA)){
    // this device has a camera
    return true;
  } else {
    // no camera on this device
    return false;
  }
}
/** A safe way to get an instance of the Camera object. */
public static Camera getCameraInstance(){
  Camera c = null;
  try {
    c = Camera.open(); // attempt to get a Camera instance
  }
  catch (Exception e){
    // Camera is not available (in use or does not exist)
  }
  return c; // returns null if camera is unavailable
}
```

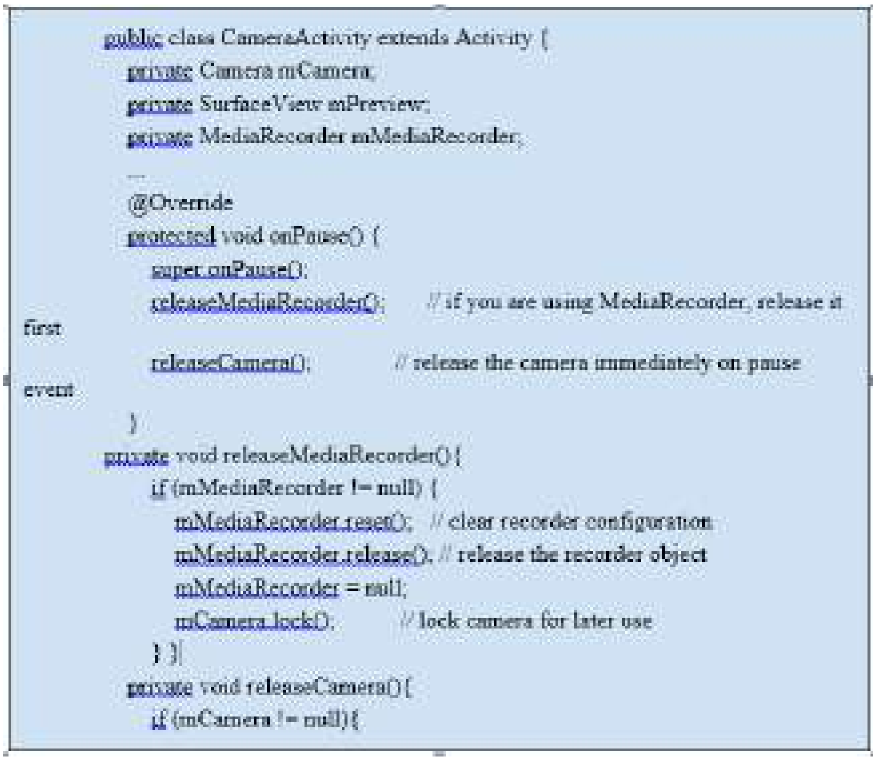
Next, the camera features are loaded. Once the camera is ready to capture the picture or the video SurfaceView is used to obtain preview of the live image. Then, a layout is specified as a container for the preview through FrameLayout. Next, Camera.takePicture() method is used to capture a picture and MediaRecorder class is used to capture a video. Configuration of the MediaRecorder is vital. Example code with the steps for configuration is shown below. Also, note that MediaRecorder can be used to create videos from captured photos. Time lapse video allows users to create video clips that combine pictures taken a few seconds or minutes apart. This feature uses MediaRecorder to record the images for a time lapse sequence.



```
        private boolean prepareVideoRecorder(){
            mCamera = getCameraInstance();
            mMediaRecorder = new MediaRecorder();

            // Step 1: Unlock and set camera to MediaRecorder
            mCamera.unlock();
            mMediaRecorder.setCamera(mCamera);
            // Step 2: Set sources
mMediaRecorder.setAudioSource(MediaRecorder.AudioSource.CAMCORDER);
            mMediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
            // Step 3: Set a CamcorderProfile (requires API Level 8 or higher)
mMediaRecorder.setProfile(CamcorderProfile.get(CamcorderProfile.QUALITY_HIGH));
            // Step 4: Set output file
mMediaRecorder.setOutputFile(getOutputMediaFile(MEDIA_TYPE_VIDEO).toString());
            // Step 5: Set the preview output
mMediaRecorder.setPreviewDisplay(mPreview.getHolder().getSurface());
            // Step 6: Prepare configured MediaRecorder
            try {
              mMediaRecorder.prepare();
            } catch (IllegalStateException e) {
              Log.d(TAG, "IllegalStateException preparing MediaRecorder: " +
e.getMessage());
        releaseMediaRecorder();
              return false;
```

Camera is a resource that is shared by many applications on a device. Your application can make use of the camera after getting an instance of Camera, and you must be particularly careful to release the camera object when your application stops using it, and as soon as your application is paused (Activity.onPause()). If your application does not properly release the camera, all subsequent attempts to access the camera, including those by your own application, will fail and may cause your or other applications to shut down.

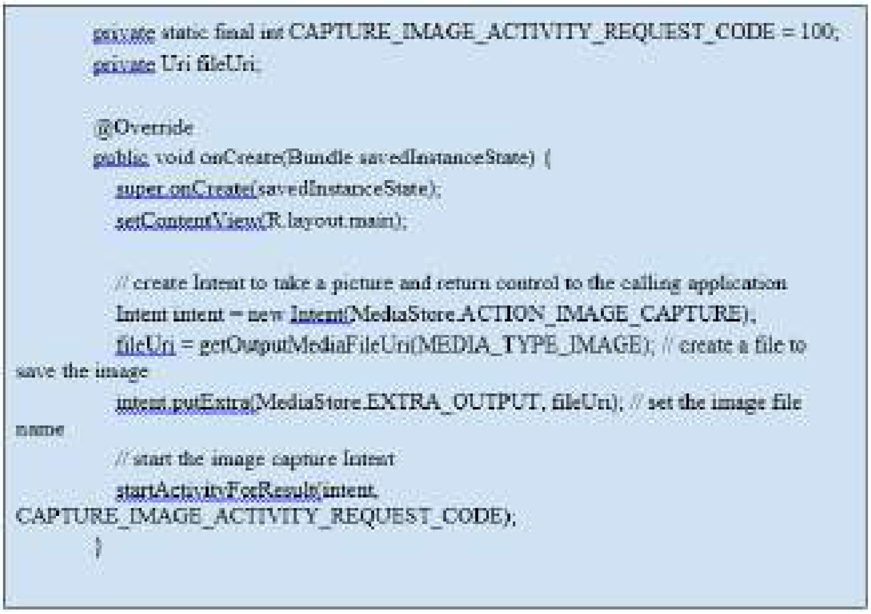
To release an instance of the Camera object, use the Camera.release() method, as shown in the example code below.



```
        public class CameraActivity extends Activity {
           private Camera mCamera;
           private SurfaceView mPreview;
           private MediaRecorder mMediaRecorder;
           ...
           @Override
           protected void onPause() {
              super.onPause();
              releaseMediaRecorder();       // if you are using MediaRecorder, release it
first
              releaseCamera();              // release the camera immediately on pause
event
           }
        private void releaseMediaRecorder(){
              if (mMediaRecorder != null) {
                 mMediaRecorder.reset();   // clear recorder configuration
                 mMediaRecorder.release(); // release the recorder object
                 mMediaRecorder = null;
                 mCamera.lock();           // lock camera for later use
              } }
           private void releaseCamera(){
              if (mCamera != null){
```

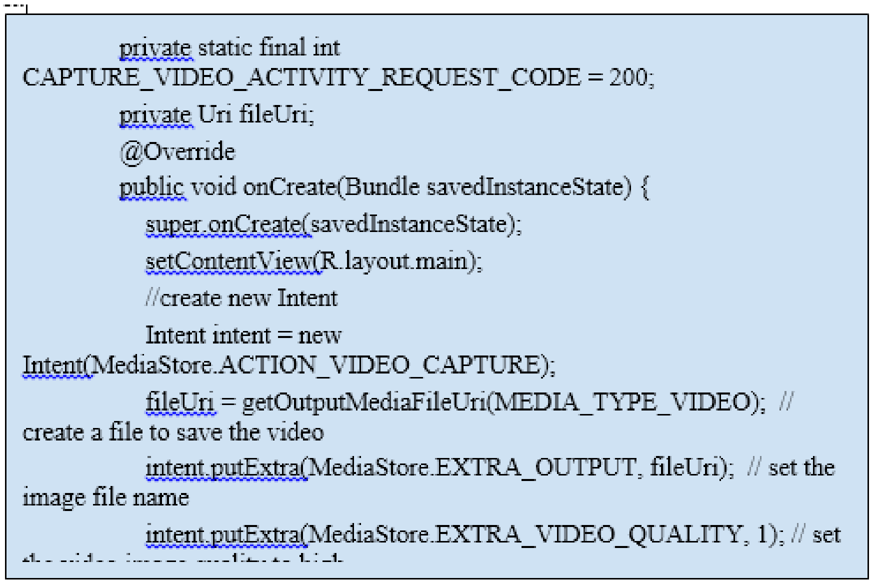
Using existing camera applications

It is also important to note that existing camera applications can be invoked through the Intent class to capture pictures and videos. The following example demonstrates how to construct an image capture intent and execute it.



```
        private static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE = 100;
        private Uri fileUri;

        @Override
        public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           setContentView(R.layout.main);

           // create Intent to take a picture and return control to the calling application
           Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
           fileUri = getOutputMediaFileUri(MEDIA_TYPE_IMAGE); // create a file to
save the image
           intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri); // set the image file
name
           // start the image capture Intent
           startActivityForResult(intent,
CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
        }
```

The following example demonstrates how to construct a video capture intent and execute it.



```
            private static final int
CAPTURE_VIDEO_ACTIVITY_REQUEST_CODE = 200;
            private Uri fileUri;
            @Override
            public void onCreate(Bundle savedInstanceState) {
               super.onCreate(savedInstanceState);
               setContentView(R.layout.main);
               //create new Intent
               Intent intent = new
Intent(MediaStore.ACTION_VIDEO_CAPTURE);
               fileUri = getOutputMediaFileUri(MEDIA_TYPE_VIDEO);  //
create a file to save the video
               intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);  // set the
image file name
               intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 1); // set
the video image quality to high
```

The general steps for creating a custom camera interface for your application are as follows: (***STEPS instead of bullets -> FLOW CHART)

- **Detect and Access Camera -** Create code to check for the existence of cameras and request access.
- **Create a Preview Class -** Create a camera preview class that extends SurfaceView and implements the SurfaceHolder interface. This class previews the live images from the camera.
- **Build a Preview Layout -** Once you have the camera preview class, create a view layout that incorporates the preview and the user interface controls you want.
- **Setup Listeners for Capture -** Connect listeners for your interface controls to start image or video capture in response to user actions, such as pressing a button.
- **Capture and Save Files -** Setup the code for capturing pictures or videos and saving the output.
- **Release the Camera -** After using the camera, your application must properly release it for use by other applications.

Camera hardware is a shared resource that must be carefully managed so your application does not collide with other applications that may also want to use it. The following sections discusses how to detect camera hardware, how to request access to a camera, how to capture pictures or video and how to release the camera when your application is done using it.

Activity

Activity 10.3

Write a code to play an audio file when a photo is open to view. Assume that the photo and the audio file are stored in the device memory.

Next, we will discuss the different formats/codes supported for images, audio and video files in the Android platform.

10.4 Supported Media Formats

When working with multimedia for Android application development, it is important to understand the core file formats and codec support that is provided (or in-built) in the Android platform. MediaCodec class is useful to access low-level media codecs, which are the encoder/decoder components. This class is part of the Android low-level multimedia support infrastructure. A codec processes input data to generate output data. Input data can be compressed data, raw audio data and raw video data. These input data are processed asynchronously and use buffers to store the output. Once the output is consumed, the buffers are released back to the codec.

Different codecs are supported with encoding parameters. These parameters include, resolution, bit-rate, and type of channel. Here we have summarized (in Table10.1), the codec support provided for image, audio and video. It is important to note that some mobile devices may provide support for additional formats/codecs not listed explicitly in Table10.1.

Table10.1: Core Media Support in Android for Audio, Video and Images. (Source: Content summarized based on the information given in https://developer.android.com/guide/appendix/media-formats.html)

Multimedia	Media CODEC
Audio	● AAC LC ● HE-AACv1 (AAC+) ● HE-AACv2 (enhanced AAC+) ● AMR-NB ● AMR-WB ● FLAC ● MP3 ● MIDI

	● Vorbis ● Opus ● wavw
Video	● H.263 ● H.264 AVC ● H.265 HEVC ● MPEG-4 SP ● VP8 ● VP9
Images	● JPEG ● GIF ● BMP ● PNG ● WebP

In this unit we have discussed how to incorporate multimedia functions when developing an Android application.

Unit summary

This unit covered the topics on how to integrate multimedia in different Android applications. The unit discussed how to utilize multimedia with appropriate examples and code snippets. To further enhance your skills and understanding activities and supplementary video were provided.

Unit 11

Saving Data on Android Devices

Introduction

In this unit you will learn about different methods to store data locally in Android. You will learn about how to use these different options and when to use them.

You will further learn about data management using SQLite. The videos available for this unit will guide you on these data related operations.

On the completion of this unit you will be able to integrate data management functionality to Android applications that you develop.

Upon completion of this unit you should be able to:

Outcomes

- Compare and contrast the different methods of persisting data locally.
- Write a program to access data in internal file system of an Android device and a SD card
- Use SQLite to create, alter update data tables and manipulate data.

Terminology

Persisting data:	denotes information that is infrequently accessed and not likely to be modified
SD Card:	Secure Digital (SD) is a non-volatile memory card format
SQLite:	C library used for database software

11.1 Android Storage Options

Android provides several options for you to save persistent application data. The solution you choose depends on your specific needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires.

Your data storage options are the following:

i. Shared Preferences - Store private primitive data as key-value pairs.

ii. Internal Storage - Store private data on the device memory.

iii. External Storage - Store public data on the shared external storage.

iv. SQLite Databases - Store structured data in a private database.

v. Remote server- Store data on a remotely hosted server.

In the following sections, we will discuss about the first four storage options mentioned above. We will however, focus only on the first four options and will not discuss about saving persistent data on cloud or other network location.

11.2 Shared Preferences

The SharedPreferences class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types. You can use SharedPreferences to save any primitive data: booleans, floats, ints, longs, and strings. This data will persist across user units (even if your application is killed). These data will only be removed only by uninstalling the application from the device or clearing the Application data via Settings menu. Also, data saved in Shared Preferences are private to this application only and not accessible by anyway for any other application on the device. We will discuss how and when to use shared preferences below.

When to use Shared Preferences?

If you need to save simple data for your application, the simplest and straight forward method is to use Shared Preferences. It is generally used to save simple data like integer, double, boolean, and short text. As Example, it is used to save application settings or user login info.

Using Shared Preferences

To get a SharedPreferences object for your application, use one of two methods:

getSharedPreferences() - Use this method if you need multiple preferences files identified by name, which you specify with the first parameter.

getPreferences() - Use this method if you need only one preferences file for your Activity. Because this will be the only preferences file for your Activity, it is not required to provide a name.

Write to Shared Preferences

To write to a shared preferences file, create a '*SharedPreferences.Editor*'by calling *edit()* on your SharedPreferences.

Pass the keys and values you want to write with methods such as putInt() and *putString()*. Then call *commit()* to save the changes. For example:

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt(getString(R.string.saved_high_score),
newHighScore);
editor.commit();
```

Read from Shared Preferences

To retrieve values from a shared preferences file, call methods such as *getInt()* and *getString()*, providing the key for the value you want, and optionally a default value to return if the key isn't present. For example:

```
SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);
int defaultValue =
getResources().getInteger(R.string.saved_high_score_de
fault);
long highScore =
sharedPref.getInt(getString(R.string.saved_high_score)
, defaultValue);
```

Here is an example that saves a preference for silent *keypress* mode in a calculator:

```
public class Calc extends Activity {
    public static final String PREFS_NAME =
                                   "MyPrefsFile";
    @Override
    protected void onCreate(Bundle state){
       super.onCreate(state);
       . . .
       // Restore preferences
       SharedPreferences settings =
getSharedPreferences(PREFS_NAME, 0);
       boolean silent =
settings.getBoolean("silentMode", false);
       setSilent(silent);
    }
    @Override
    protected void onStop(){
       super.onStop();

     // Need an Editor object to make preference changes.
      // All objects are from android.context.Context
      SharedPreferences settings =
getSharedPreferences(PREFS_NAME, 0);
      SharedPreferences.Editor editor =
settings.edit();
      editor.putBoolean("silentMode", mSilentMode);

      // Commit the edits!
      editor.commit();
    }
```

Delete Shared Preference Data

You have two options to delete data persisted as Shared Preference;

i. Delete a specific item

ii. Delete all data

Delete a specific item

```
SharedPreferences  sharedPref =
getApplicationContext().getSharedPreferences("MyAppDat
a", 0);
Editor sharedPrefEditor = sharedPref.edit();


sharedPrefEditor.remove(key); // key of the data you
want to delete
sharedPrefEditor.commit();
```

Delete all Data

```
SharedPreferences sharedPref =
getApplicationContext().getSharedPreferences("MyAppDat
a", 0);
Editor sharedPrefEditor = sharedPref.edit();
sharedPrefEditor.clear(); //clear all data inside
MyAppData Shared Preference File
sharedPrefEditor.commit();
```

The next section will discuss about the internal storage options available with Android.

Activity

Activity 11.1

Why do we need shared preferences to store persistent data?

11.3 Internal Storage

You can save files directly on the device's internal storage. By default, files saved to the internal storage are private to your application and other applications cannot access them (nor can the user). When the user uninstalls your application, these files are removed. Data will be removed only by uninstalling the application from the device.

When saving a file to internal storage, you can acquire the appropriate directory as a File by calling one of two methods:

getFilesDir() - Returns a '*File*' representing an internal directory for your app.

getCacheDir() - Returns a '*File*' representing an internal directory for your app's temporary cache files. Be sure to delete each file once it is no longer needed and implement a reasonable size limit for the amount of memory you use at any given time, such as 1MB. If the system begins running low on storage, it may delete your cache files without warning.

We will now discuss how and when to use internal storage.

When to use internal storage?

The amount of data in Internal Storage depends on the device. Therefore do not try to save a large persistent file because it may crash your application if there is not enough space available on the device. Preferably, keep any data under 1M such as text files or xml files.

Write files to internal storage

The following steps show how to create and write a private file to the internal storage:

Step 1 - Call *openFileOutput()* with the name of the file and the operating mode. This returns a FileOutputStream.

Step 2 - Write to the file with *write()*.

Step 3 - Close the stream with *close()*.

For example:

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME,
Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

MODE_PRIVATE will create the file (or replace a file of the same name) and make it private to your application. Other modes available are: MODE_APPEND, MODE_WORLD_READABLE, and MODE_WORLD_WRITEABLE.

Read from Internal Storage

To read a file from internal storage:

Step 1 - Call *openFileInput()* and pass it the name of the file to read. This returns a FileInputStream.

Step 2 - Read bytes from the file with *read()*.

Step 3 - Then close the stream with *close()*.

Tip: If you want to save a static file in your application at compile time, save the file in your project res/raw/ directory. You can open it with *openRawResource()*, passing the *R.raw.<filename>* resource ID. This method returns an *InputStream* that you can use to read the file (but you cannot write to the original file).

Delete Internal Storage Data

The following examples shows how to delete a file from the internal storage.

```
File fileDir = getFilesDir();
File file = new File(fileDir, "fileName");
file.delete();
```

Saving cache files

If you would like to cache some data, rather than store it persistently, you should use *getCacheDir()* to open a ‘File’ that represents the internal directory where your application should save temporary cache files.

When the device is low on internal storage space, Android may delete these cache files to recover space. However, you should not rely on the system to clean up these files for you. You should always maintain the cache files yourself and stay within a reasonable limit of space consumed, such as 1MB. When the user uninstalls your application, these files are removed.

Next, we will focus on external storage options available with Android.

11.4 External Storage

Every Android-compatible device supports a shared "external storage" that you can use to save files. This can be a removable storage media (such as an SD card) or an internal (non-removable) storage. Files saved to the external storage are world-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer. We will see how and when to use the external storage along with checking the availability of the media device and managing the visibility your files to other apps.

When to use External Storage?

Use External Storage whenever you need to save large files such as audio or video files and can be retrieved by repeatedly. Also you can use this if want your files to be shared through different application like statistics files.

Getting access to external storage

In order to read or write files on the external storage, your app must acquire the *READ_EXTERNAL_STORAGE or WRITE_EXTERNAL_*STORAGE system permissions.

```
<manifest ...>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAG
E" />
    ...
</manifest>
```

If you need to both read and write files, then you need to request only the *WRITE_EXTERNAL_STORAGE* permission, because it implicitly requires read access as well.

Caution: Currently, all apps have the ability to read the external storage without a special permission. However, this is going to be changed in a future release. If your app needs to read the external storage (but not write to it), then you will need to declare the *READ_EXTERNAL_STORAGE* permission. To ensure that your app continues to work as expected, you should declare this permission now, before the change takes effect.

```
<manifest ...>
    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE
" />
    ...
</manifest>
```

However, if your app uses the *WRITE_EXTERNAL_STORAGE* permission, then it implicitly has permission to read the external storage as well.

You do not need any permissions to save files on the internal storage. Your application always has permission to read and write files in its internal storage directory.

Checking media availability

Before you do any work with the external storage, you should always call *getExternalStorageState()* to check whether a compatible media is available. The media might be mounted to a computer, read-only, or in some other state. For example, here are a couple methods you can use to check the availability:

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state =
Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state =
Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(sta
te)) {
        return true;
    }
    return false;
}
```

The *getExternalStorageState()* method returns other states that you might want to check, such as whether the media is being shared (connected to a computer), is missing entirely, has been removed badly, etc. You can use these to notify the user with more information when your application needs to access the media.

Query Free Space

If you know ahead of time how much data you are saving, you can find out whether sufficient space is available without causing an *IOException* by calling *getFreeSpace()* or *getTotalSpace()*. These methods provide the current available space and the total space in the storage volume, respectively.

Saving files that can be shared with other apps

You may require sharing files across other apps. Sometimes, you may want to hide your files from others. We will now see how to share with or hide your files from other apps.

Hiding your files from the Media Scanner

To hide your files, include an empty file named *.nomedia* in your external files directory (note the dot prefix in the filename). This prevents media scanner from reading your media files and providing them to other apps through the MediaStore content provider. However, if your files are truly private to your app, you should save them in an app-private directory.

Generally, new files that the user may acquire through your app should be saved to a "public" location on the device where other apps can access them and the user can easily copy them from the device. When doing so, you should use to one of the shared public directories, such as Music/, Pictures/, and Ringtones/.

Saving files that are public to the users

To get a '*File'* representing the appropriate public directory, call *getExternalStoragePublicDirectory*(), passing it the type of directory you want, such as *DIRECTORY_MUSIC*, *DIRECTORY_PICTURES*, *DIRECTORY_RINGTONES*, or others. By saving your files to the corresponding media-type directory, the system's media scanner can properly categorize your files in the system (for instance, ringtones appear in system settings as ringtones, not as music).

For example, here is a method that creates a directory for a new photo album in the public pictures directory:

```
public File getAlbumStorageDir(String albumName) {
    // Get the directory for the user's public pictures directory.
    File file = new
File(Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES),
albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

Saving files that are app-private

If you want to save files that are private to your app, you can acquire the appropriate directory by calling *getExternalFilesDir()* and 'passing' it a name indicating the type of directory you would like. Each directory created this way is added to a parent directory that encapsulates all your app's external storage files, which the system deletes when the user uninstalls your app. For example, here's a method you can use to create a directory for an individual photo album:

```
public File getAlbumStorageDir(Context context, String
albumName) {
    // Get the directory for the app's private pictures directory.

    File file = new File(context.getExternalFilesDir(
            Environment.DIRECTORY_PICTURES),
albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

If none of the pre-defined sub-directory names suit your files, you can instead call *getExternalFilesDir()* and pass null. This returns the root directory for your app's private directory on the external storage.

Remember that *getExternalFilesDir()* creates a directory inside a directory that is deleted when the user uninstalls your app. If the files you are saving should remain available after the user uninstalls your app for instance when your app is a camera and the user will want to keep the photos. Instead use getExternalStoragePublicDirectory().

Caution: Although the directories provided by *getExternalFilesDir()* and *getExternalFilesDirs()* are not accessible by the MediaStore content provider, other apps with the *READ_EXTERNAL_STORAGE* permission can access all files on the external storage, including these. If you need to completely restrict access for your files, you should instead write your files to the internal storage.

Saving cache files

To open a ‘*File’* that represents the external storage directory where you should save cache files, call *getExternalCacheDir()*. If the user uninstalls your application, these files will be automatically deleted.

Similar to *ContextCompat.getExternalFilesDirs()*, mentioned above, you can also access a cache directory on a secondary external storage (if available) by calling *ContextCompat.getExternalCacheDirs().*

Delete a File

You should always delete files that you no longer need. The most straightforward way to delete a file is to have the opened file reference call *delete()* on itself.

```
myFile.delete();
```

If the file is saved on internal storage, you can also ask the Context to locate and delete a file by calling *deleteFile():*

```
myContext.deleteFile(fileName);
```

Note: When the user uninstalls your app, the Android system deletes the following:

All files you saved on internal storage

All files you saved on external storage using *getExternalFilesDir()*.

However, you should manually delete all cached files created with *getCacheDir()* on a regular basis and also regularly delete other files you no longer need.

The next section will discuss about how to save data in SQLite databases, read the stored data when required and manipulating the stored data as per your requirement.

Activity

Activity 11.2

Compare and contrast data storage using internal storage and external storage options.

11.5 Saving Data in SQLite Databases

Saving data to a database is ideal for repeating or structured data, such as contact information. This class assumes helps you get started with SQLite databases on Android. The APIs you will need to use a database on Android are available in the android.database.sqlite package. We will discuss how and when to use databases as a storage option.

When to use SQLite DB?

You can use databases to store user data as per your app requirement. Otherwise, it does not have a specific reason to use a database.

Define a Schema and Contract

One of the main principles of SQL databases is the schema: a formal declaration of how the database is organized. The schema is reflected in the SQL statements that you use to create your database. You may find it helpful to create a companion class, known as a contract class, which explicitly specifies the layout of your schema in a systematic and self-documenting way.

A contract class is a container for constants that define names for URIs, tables, and columns. The contract class allows you to use the same constants across all the other classes in the same package. This let you change a column name in one place and have it propagate throughout your code.

A good way to organize a contract class is to put definitions that are global to your whole database in the root level of the class. Then create an inner class for each table that enumerates its columns.

Note: By implementing the BaseColumns interface, your inner class can inherit a primary key field called _ID that some Android classes such as cursor adaptors will expect it to have. It is not required, but this can help your database work harmoniously with the Android framework.

For example, this snippet defines the table name and column names for a single table:

```
public final class FeedReaderContract {
    // To prevent someone from accidentally instantiating the
// contract class,give it an empty constructor.
    public FeedReaderContract() {}

    /* Inner class that defines the table contents */
    public static abstract class FeedEntry implements
BaseColumns {
        public static final String TABLE_NAME =
"entry";
        public static final String
COLUMN_NAME_ENTRY_ID = "entryid";
        public static final String COLUMN_NAME_TITLE =
"title";
        public static final String
COLUMN_NAME_SUBTITLE = "subtitle";
        ...
    }
}
```

Create a Database Using a SQL Helper

Once you have defined how your database looks, you should implement methods that create and maintain the database and tables. Here are some typical statements that create and delete a table:

```
private static final String TEXT_TYPE = " TEXT";
private static final String COMMA_SEP = ",";
private static final String SQL_CREATE_ENTRIES =
    "CREATE TABLE " + FeedEntry.TABLE_NAME + " (" +
    FeedEntry._ID + " INTEGER PRIMARY KEY," +
    FeedEntry.COLUMN_NAME_ENTRY_ID + TEXT_TYPE +
COMMA_SEP +
    FeedEntry.COLUMN_NAME_TITLE + TEXT_TYPE +
COMMA_SEP +
    ... // Any other options for the CREATE command
    " )";

private static final String SQL_DELETE_ENTRIES =
    "DROP TABLE IF EXISTS " + FeedEntry.TABLE_NAME;
```

Just like files that you save on the device's internal storage, Android stores your database in private disk space associated with the application. Your data is secure, because by default this area is not accessible to other applications.

A useful set of APIs is available in the *SQLiteOpenHelper* class. When you use this class to obtain references to your database, the system performs the potentially long-running operations of creating and updating the database only when needed and not during app startup. All you need to do is call *getWritableDatabase()* or *getReadableDatabase()*.

Note: Because they can be long-running, be sure that you call *getWritableDatabase()* or *getReadableDatabase()* in a background thread, such as with AsyncTask or IntentService.

To use *SQLiteOpenHelper*, create a subclass that overrides the *onCreate()*, *onUpgrade()* and *onOpen()* callback methods. You may also want to implement *onDowngrade()*, but it's not required.

For example, here is an implementation of *SQLiteOpenHelper* that uses some of the commands shown above:

```
public class FeedReaderDbHelper extends
SQLiteOpenHelper {
    // If you change the database schema, you must
increment the database version.
    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME =
"FeedReader.db";

    public FeedReaderDbHelper(Context context) {
        super(context, DATABASE_NAME, null,
DATABASE_VERSION);
    }
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_ENTRIES);
    }
    public void onUpgrade(SQLiteDatabase db, int
oldVersion, int newVersion) {
        // This database is only a cache for online
data, so its upgrade policy is
        // to simply to discard the data and start
over
        db.execSQL(SQL_DELETE_ENTRIES);
        onCreate(db);
    }
    public void onDowngrade(SQLiteDatabase db, int
oldVersion, int newVersion) {
        onUpgrade(db, oldVersion, newVersion);
    }
}
```

To access your database, instantiate your subclass of *SQLiteOpenHelper*:

```
FeedReaderDbHelper mDbHelper = new
FeedReaderDbHelper(getContext());
```

Put Information into a Database

Insert data into the database by passing a ContentValues object to the *insert()* method:

```
// Gets the data repository in write mode
SQLiteDatabase db = mDbHelper.getWritableDatabase();

// Create a new map of values, where column names are the keys
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_ENTRY_ID, id);
values.put(FeedEntry.COLUMN_NAME_TITLE, title);
values.put(FeedEntry.COLUMN_NAME_CONTENT, content);

// Insert the new row, returning the primary key value of the new
//row
long newRowId;
newRowId = db.insert(
         FeedEntry.TABLE_NAME,
         FeedEntry.COLUMN_NAME_NULLABLE,
         values);
```

The first argument for *insert()* is simply the table name. The second argument provides the name of a column in which the framework can insert NULL in the event that the ContentValues is empty (if you instead set this to "null", then the framework will not insert a row when there are no values).

Read Information from a Database

To read from a database, use the *query()* method, passing it your selection criteria and desired columns. The method combines elements of *insert()* and *update()*, except the column list defines the data you want to fetch, rather than the data to insert. The results of the query are returned to you in a Cursor object.

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// Define a projection that specifies which columns
from the database
// you will actually use after this query.
String[] projection = {
    FeedEntry._ID,
    FeedEntry.COLUMN_NAME_TITLE,
    FeedEntry.COLUMN_NAME_UPDATED,
    ...
    };

// How you want the results sorted in the resulting
Cursor
String sortOrder =
    FeedEntry.COLUMN_NAME_UPDATED + " DESC";

Cursor c = db.query(
    FeedEntry.TABLE_NAME,  // The table to query
    projection,            // The columns to return
    selection,            // The columns for the WHERE
clause
    selectionArgs,        // The values for the WHERE
clause
```

```
    null,                     // don't group the rows
    null,                   // don't filter by row groups
    sortOrder               // The sort order
    );
```

To look at a row in the cursor, use one of the Cursor move methods, which you must always call before you begin reading values. Generally, you should start by calling *moveToFirst()*, which places the "read position" on the first entry in the results. For each row, you can read a column's value by calling one of the Cursor get methods, such as *getString()* or getLong(). For each of the get methods, you must pass the index position of the column you desire, which you can get by calling *getColumnIndex()* or *getColumnIndexOrThrow()*.

For example:

```
cursor.moveToFirst();
long itemId = cursor.getLong(
    cursor.getColumnIndexOrThrow(FeedEntry._ID)
);
```

Note: You can execute SQLite queries using the *SQLiteDatabase query()* methods, which accept various query parameters, such as the table to query, the projection, selection, columns, grouping, and others. For complex queries, such as those that require column aliases, you should use *SQLiteQueryBuilder*, which provides several convenient methods for building queries.

Delete Information from a Database

To delete rows from a table, you need to provide selection criteria that identify the rows. The database API provides a mechanism for creating selection criteria that protects against SQL injection. The mechanism divides the selection specification into a selection clause and selection arguments. The clause defines the columns to look at, and also allows you to combine column tests. The arguments are values to test against that are bound into the clause. Because the result is not handled the same as a regular SQL statement, it is immune to SQL injection.

```
// Define 'where' part of query.
String selection = FeedEntry.COLUMN_NAME_ENTRY_ID + "
LIKE ?";
// Specify arguments in placeholder order.
String[] selectionArgs = { String.valueOf(rowId) };
// Issue SQL statement.
db.delete(table_name, selection, selectionArgs);
```

Update a Database

When you need to modify a subset of your database values, use the *update()* method.

Updating the table combines the content values syntax of *insert()* with the where syntax of *delete()*.

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// New value for one column
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_TITLE, title);

// Which row to update, based on the ID
String selection = FeedEntry.COLUMN_NAME_ENTRY_ID + "
LIKE ?";
String[] selectionArgs = { String.valueOf(rowId) };

int count = db.update(
    FeedReaderDbHelper.FeedEntry.TABLE_NAME,
    values,
    selection,
    selectionArgs);
```

With that we have come to the end of this unit and now you are in a position to use data management on your apps.

Activity

Activity 11.3

Choose a popular game app and discuss how each of the storage options you learnt is used in that app.

Unit summary

In this unit we discussed about the four of the five methods used to persist data on Android devices. They are namely; Shared preferences, Internal storage, external storage and external database. We also learnt about when to use each of these options and how we can use them in apps. The important things to keep in your mind is that the privacy level of the files/data and the size of the data.

References

https://developer.android.com/guide/topics/sensors/sensors_overview.html

https://developer.android.com/training/location/index.html

CC by 2.5 – Android Developer Forum (developer.android.com)

Unit 12

Locating and Sensing

Introduction

In this unit you will be introduced to various sensors available in an Android device. The availability of a particular sensor depends on the device you use. You will learn how to use these sensors accessing relevant API’s to obtain data and use them in applications.

You will also learn about Google Maps and how location awareness can be achieved in applications.

A video is incorporated in this unit to help you to understand more about sensors and Google map.

Upon completion of this unit you should be able to:

Outcomes

- identify functionality of the common sensors available in Android devices.
- illustrate how sensors can be used in apps with examples.
- develop a small app to demonstrate the context awareness of the users with location.

Terminology

sensors:	measure motion, orientation, position, and various environmental conditions
context awareness:	refers to a general class of mobile systems that can sense their physical environment

12.1 Introduction to Sensors

Most Android devices have built-in sensors that measure motion, orientation, position, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful if you want to monitor three-dimensional device movement or positioning, or you want to monitor changes in the ambient environment near a device.

Some of these sensors are hardware-based and some are software-based. Hardware-based sensors are physical components built into a handset or tablet device. They derive their data by directly measuring specific environmental properties, such as acceleration, geomagnetic field strength, or angular change. Software-based sensors are not physical devices, although they mimic hardware-based sensors. Software-based sensors derive their data from one or more of the hardware-based sensors and are sometimes called virtual sensors or synthetic sensors. The linear acceleration sensor and the gravity sensor are examples of software-based sensors. Appendix 12.1 summarizes the sensor types that are supported by the Android platform.

The Android platform supports three broad categories of sensors:

- Motion sensors - These sensors measure acceleration forces and rotational forces along three axes (x,y,and z). List of Motion sensors that are supported on the Android platform is given in reference 12.2
- Environmental sensors - These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. Appendix 12.3 lists the environment sensors that are supported on the Android platform
- Position sensors - These sensors measure the physical position of a device. Appendix 12.4 lists the position sensors that are supported on the Android platform.

In the next section, we will discuss about the Android sensor framework.

12.2 Android Sensor Framework

The sensor framework provides several classes and interfaces that help you perform a wide variety of sensor-related tasks. For example, you can use the sensor framework to do the following:

- Determine which sensors are available on a device.
- Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.
- Acquire raw sensor data and define the minimum rate at which you acquire sensor data.
- Register and unregister sensor event listeners that monitor sensor changes.

Now, we will see the classes and interfaces available in the sensor framework.

Classes and Interfaces Available in the Sensor Framework

The sensor framework is part of Android. Hardware package includes the following classes and interfaces:

Sensor Manager

You can use this class to create an instance of the sensor service. This class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.

Sensor

You can use this class to create an instance of a specific sensor. This class provides various methods that let you determine a sensor's capabilities.

Sensor Event

The system uses this class to create a sensor event object, which provides information about a sensor event. A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.

Sensor Event Listener

You can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

In the next section we will learn how to identify different sensors available on a device and the capabilities of each sensor.

12.3 Identifying Sensors and sensor Capabilities

Identifying sensors and sensor capabilities at runtime is useful if your application has features that rely on specific sensor types or capabilities. For example, you may want to identify all of the sensors that are present on a device and disable any application features that rely on sensors that are not present. Likewise, you may want to identify all of the sensors of a given type so you can choose the sensor implementation that has the optimum performance for your application.

To identify the sensors that are on a device you first need to get a reference to the sensor service. To do this, you create an instance of the ‘SensorManager’ class by calling the *getSystemService()* method and passing in the SENSOR_SERVICE argument.

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
```

Next, you can get a listing of every sensor on a device by calling the `getSensorList()` method and using the TYPE_ALL constant.

```
List<Sensor> deviceSensors =
mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

If you want to list all of the sensors of a given type, you could use another constant instead of TYPE_ALL such as TYPE_GYROSCOPE, TYPE_LINEAR_ACCELERATION, or TYPE_GRAVITY.

You can also determine whether a specific type of sensor exists on a device by using the *getDefaultSensor()* method. If a default sensor does not exist for a given type of sensor, the method call returns null, which means the device does not have that type of sensor. The following code checks whether there's a magnetometer on a device:

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
if
(mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_
FIELD) != null){
  // Success! There's a magnetometer.
  }
else {
  // Failure! No magnetometer.
  }
```

In addition to listing the sensors that are on a device, you can use the public methods of the Sensor class to determine the capabilities and attributes of individual sensors. For example, you can use the *getResolution*() and *getMaximumRange*() methods to obtain a sensor's resolution and maximum range of measurement. You can also use the *getPower*() method to obtain a sensor's power requirements.

Now that we have learnt how to identify sensors and their capabilities, it is time to learn how to monitor sensor events. The next section will discuss about monitoring sensor events.

Activity

Activity 12.1

Find and list all the sensors available in your Android device.

12.4 Monitoring Sensor Events

A sensor event occurs every time a sensor detects a change in the parameters it is measuring. A sensor event provides you with four pieces of information: the name of the sensor that triggered the event, the timestamp for the event, the accuracy of the event, and the raw sensor data that triggered the event.

To monitor raw sensor data you need to implement two callback methods that are exposed through the *SensorEventListener* interface: *onAccuracyChanged*() and *onSensorChanged*(). The Android system calls these methods whenever the following occurs:

A sensor's accuracy changes.

In this case the system invokes the *onAccuracyChanged*() method, providing you with a reference to the *Sensor* object that changed and the new accuracy of the sensor.

A sensor reports a new value.

In this case the system invokes the *onSensorChanged*() method, providing you with a *SensorEvent* object. A *SensorEvent* object contains information about the new sensor data, including: the accuracy of the data, the sensor that generated the data, the timestamp at which the data was generated, and the new data that the sensor recorded.

The following code shows how to use the *onSensorChanged*() method to monitor data from the light sensor. This example displays the raw sensor data in a *TextView* that is defined in the main.xml file as sensor_data.

```
public class SensorActivity extends Activity
implements SensorEventListener {
  private SensorManager mSensorManager;
  private Sensor mLight;

  @Override
  public final void onCreate(Bundle
savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
    mLight =
mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
  }
  @Override
  public final void onAccuracyChanged(Sensor sensor,
int accuracy) {
    // Do something here if sensor accuracy changes.
  }

  @Override
  public final void onSensorChanged(SensorEvent event)
{
    // The light sensor returns a single value.
    // Many sensors return 3 values, one for each
axis.
    float lux = event.values[0];
    // Do something with this sensor value.
  }

  @Override
  protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mLight,
SensorManager.SENSOR_DELAY_NORMAL);
  }

  @Override
  protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
  }
}
```

In this example, the default data delay (SENSOR_DELAY_NORMAL) is specified when the *registerListener*() method is invoked. The data delay (or sampling rate) controls the interval at which sensor events are sent to your application via the *onSensorChanged*() callback method.

Next, we will discuss about the sensor coordinate system.

12.5 Sensor Coordinate System

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (Figure 12.1).

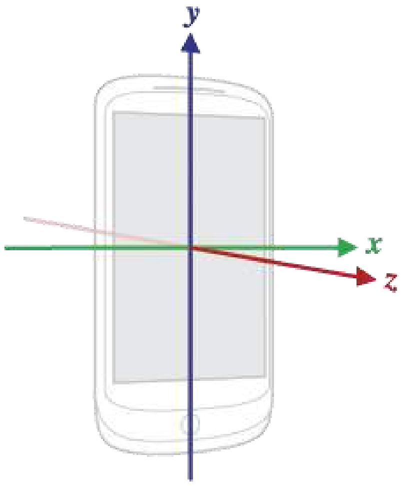


Figure 12.1- Coordinate system used by the Sensor API

The most important point to understand about this coordinate system is that the axes are not swapped when the device's screen orientation changes—that is, the sensor's coordinate system never changes as the device moves.

Another point to understand is that your application must not assume that a device's natural (default) orientation is portrait. The natural orientation for many tablet devices is landscape. And the sensor coordinate system is always based on the natural orientation of a device.

In the next section, we will discuss about the best practices for accessing and using sensors.

12.6 Best Practices for Accessing and Using Sensors

Under this topic we will be explaining some of the best practices that you should follow when accessing and using sensors.

Unregister sensor listeners

Be sure to unregister a sensor's listener when you have finished using the sensor or when the sensor activity pauses. If a sensor listener is registered and its activity is paused, the sensor will continue to acquire data and use battery resources unless you unregister the sensor. Also there could be some exception the application runtime, if activity get destroyed without unregistering the sensor manager. The following code snippet shows how to use the *onPause*() method to unregister a listener:

```
private SensorManager mSensorManager;
  ...
@Override
protected void onPause() {
  super.onPause();
  mSensorManager.unregisterListener(this);
}
```

Do not test your code on the emulator

The default Android emulator cannot emulate sensors. You should test your sensor code on a physical device or emulator which capable to emulate the sensors as well. There are, however, sensor simulators that you can use to simulate sensor output.

Do not block the onSensorChanged() method

Sensor data can change at a high rate, which means the system may call the *onSensorChanged*(*SensorEvent*) method quite often. As a best practice, you should do as little as possible within the *onSensorChanged*(*SensorEvent*) method so you don't block it. If your application requires you to do any data filtering or reduction of sensor data, you should perform that work outside of the *onSensorChanged*(*SensorEvent*) method.

Verify sensors before you use them

Always verify that a sensor exists on a device before you attempt to acquire data from it. Don't assume that a sensor exists simply because it's a frequently-used sensor. Device manufacturers are not required to provide any particular sensors in their devices.

Choose sensor delays carefully

When you register a sensor with the *registerListener*() method, be sure you choose a delivery rate that is suitable for your application or use-case. Sensors can provide data at very high rates. Allowing the system to send extra data that you don't need wastes system resources and uses battery power.

Next, we will discuss about some of the commonly used sensors.

12.7 Commonly Used Sensors

There are many different types of sensors. Under this topic we will be explaining some of the commonly used sensors accelerometer, Gravity Sensor, Gyroscope and Proximity Sensor.

Accelerometer

An acceleration sensor measures the acceleration applied to the device, including the force of gravity. The following code shows you how to get an instance of the default acceleration sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
  ...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
mSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROME
TER);
```

It should be noted that the force of gravity (g) is always influencing the measured acceleration. For this reason, when the device is sitting on a table (and not accelerating), the accelerometer reads a magnitude of g = 9.81 m/s2. Similarly, when the device is in free fall and therefore rapidly accelerating toward the ground at 9.81 m/s2, its accelerometer reads a magnitude of g = 0 m/s2. Therefore, to measure the real acceleration of the device, the contribution of the force of gravity must be removed from the accelerometer data. This can be achieved by applying a high-pass filter. Conversely, a low-pass filter can be used to isolate the force of gravity. The following example shows how you can do this:

```
public void onSensorChanged(SensorEvent event){
  // In this example, alpha is calculated as t / (t + dT),
  // where t is the low-pass filter's time-constant and
  // dT is the event delivery rate.

  final float alpha = 0.8;

  // Isolate the force of gravity with the low-pass filter.
  gravity[0] = alpha * gravity[0] + (1 - alpha) *
event.values[0];
  gravity[1] = alpha * gravity[1] + (1 - alpha) *
event.values[1];
  gravity[2] = alpha * gravity[2] + (1 - alpha) *
event.values[2];
```

```
  // Remove the gravity contribution with the high-pass filter.
  linear_acceleration[0] = event.values[0] -
gravity[0];
  linear_acceleration[1] = event.values[1] -
gravity[1];
  linear_acceleration[2] = event.values[2] -
gravity[2];
}
```

In general, the accelerometer is a good sensor to use if you are monitoring device motion. Almost every Android-powered handset and tablet has an accelerometer, and it uses about 10 times less power than the other motion sensors. One drawback is that you might have to implement low-pass and high-pass filters to eliminate gravitational forces and reduce noise.

Gravity Sensor

The gravity sensor provides a three dimensional vector indicating the direction and magnitude of gravity. The following code shows you how to get an instance of the default gravity sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
mSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
```

The units are the same as those used by the acceleration sensor (m/s2), and the coordinate system is the same as the one used by the acceleration sensor

Gyroscope

The gyroscope measures the rate of rotation in rad/s around a device's x, y, and z axis. The following code shows you how to get an instance of the default gyroscope:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
mSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE)
;
```

The sensor's coordinate system is the same as the one used for the acceleration sensor. Rotation is positive in the counter-clockwise direction.

Standard gyroscopes provide raw rotational data without any filtering or correction for noise and drift (bias). In practice, gyroscope noise and drift will introduce errors that need to be compensated for. You usually determine the drift (bias) and noise by monitoring other sensors, such as the gravity sensor or accelerometer.

Proximity Sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
mSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
;
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

```
public class SensorActivity extends Activity
implements SensorEventListener {
  private SensorManager mSensorManager;
  private Sensor mProximity;

  @Override
  public final void onCreate(Bundle
savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Get an instance of the sensor service, and use
that to get an instance of
    // a particular sensor.
    mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
    mProximity =
mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
;
  }

  @Override
  public final void onAccuracyChanged(Sensor sensor,
int accuracy) {
    // Do something here if sensor accuracy changes.
  }

  @Override
  public final void onSensorChanged(SensorEvent event)
{
    float distance = event.values[0];
    // Do something with this sensor data.
  }

  @Override
  protected void onResume() {
    // Register a listener for the sensor.
    super.onResume();
    mSensorManager.registerListener(this, mProximity,
SensorManager.SENSOR_DELAY_NORMAL);
  }

  @Override
  protected void onPause() {
    // Be sure to unregister the sensor when the
activity pauses.
    super.onPause();
    mSensorManager.unregisterListener(this);
  }
}
```

Next we will discuss about how to make your apps location-aware.

Activity

Activity 12.2

What are the sensors/location details required in following category of apps?

- ✓ Car racing game
- ✓ Weather predictor
- ✓ Navigation app
- ✓ Video chat app

12.8 Making Your App Location-Aware

One of the unique features of mobile applications is location awareness. Mobile users take their devices with them everywhere, and adding location awareness to your app offers users a more contextual experience. The location APIs available in Google Play services facilitate adding location awareness to your app with automated location tracking, geofencing (a facility that monitors whether a device is near to a location of interest), and activity recognition.

Android offers two ways to add location awareness to your apps; one through the Google Play services location APIs and the other is through the Android framework location API. However, the former is preferred over the latter as a way of adding location awareness to your app.

In this section, we will discuss how to use the Google Play services location APIs in your app to get the current location, and get periodic location updates. In addition the same API's can be used to maintain addresses and geofences. However, we will not discuss about addresses and geofences here.

Specify App Permissions

Apps that use location services must request location permissions. Android offers two location permissions: ACCESS_COARSE_LOCATION and ACCESS_FINE_LOCATION. The permission you choose determines the accuracy of the location returned by the API. If you specify ACCESS_COARSE_LOCATION, the API returns a location with an accuracy approximately equivalent to a city block.

Connect to Google Play Services

To connect to the API, you need to create an instance of the Google Play services API client. For details about using the client, see the guide to Accessing Google APIs.

In your activity's onCreate() method, create an instance of Google API Client, using the GoogleApiClient.Builder class to add the LocationServices API, as the following code snippet shows.

```
// Create an instance of GoogleAPIClient.
if (mGoogleApiClient == null) {
    mGoogleApiClient = new
GoogleApiClient.Builder(this)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .addApi(LocationServices.API)
        .build();}
```

To connect, call *connect*() from the activity's *onStart*() method. To disconnect, call *disconnect*() from the activity's onStop() method. The following snippet shows an example of how to use both of these methods.

```
protected void onStart() {
    mGoogleApiClient.connect();
    super.onStart();
}
protected void onStop() {
    mGoogleApiClient.disconnect();
    super.onStop();}
```

In the next section we will discuss about how to obtain the last known location of a particular Android device.

12.9 Getting the Last Known Location

Use the fused location provider to retrieve the device's last known location. The fused location provider manages the underlying location technology and provides a simple API so that you can specify requirements at a high level, like high accuracy or low power. It also optimizes the device's use of battery power.

This service requires only coarse location. Request this permission with the uses-permission element in your app manifest, as the following code snippet shows:

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/andr
oid"
 package="com.google.android.gms.location.sample.basic
locationsample" >

<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATIO
N"/>
</manifest>
```

To request the last known location, call the *getLastLocation*() method, passing it your instance of the GoogleApiClient object. Do this in the *onConnected*() callback provided by Google API Client, which is called when the client is ready. The following code snippet illustrates the request and a simple handling of the response:

```
public class MainActivity extends ActionBarActivity
implements
        ConnectionCallbacks,
OnConnectionFailedListener {
    ...
    @Override
    public void onConnected(Bundle connectionHint) {
        mLastLocation =
LocationServices.FusedLocationApi.getLastLocation(
                mGoogleApiClient);
        if (mLastLocation != null) {
            mLatitudeText.setText(String.valueOf(mLast
Location.getLatitude()));
```

```
            mLongitudeText.setText(String.valueOf(mLas
tLocation.getLongitude()));
        }
    }
}
```

The *getLastLocation*() method returns a Location object from which you can retrieve the latitude and longitude coordinates of a geographic location.

Now that we are aware of how to get the location details, we can discuss about changing location settings. Next section will discuss about changing location settings.

12.10 Changing Location Settings

To change location settings, coarse location detection is sufficient. Request this permission with the uses-permission element in your app manifest, as shown in the following example:

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/andr
oid"  package="com.google.android.gms.location.sample.
locationupdates" >

<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATIO
N"/>
</manifest>
```

Set Up a Location Request

To store parameters for requests to the fused location provider, create a Location Request. The parameters determine the level of accuracy for location requests. Here you will learn to set the update interval, fastest update interval, and priority, as described below:

Update interval

setInterval() - This method sets the rate in milliseconds at which your app prefers to receive location updates.

Fastest update interval

setFastestInterval() - This method sets the fastest rate in milliseconds at which your app can handle location updates. You need to set this rate because other apps also affect the rate at which updates are sent. The Google Play services location APIs send out updates at the fastest rate that any app has requested with *setInterval*(). If this rate is faster than your app can handle, you may encounter problems with UI flicker or data overflow. To prevent this, call *setFastestInterval*() to set an upper limit to the update rate.

Priority

setPriority() - This method sets the priority of the request, which gives the Google Play services location services a strong hint about which location sources to use. The following values are supported:

- PRIORITY_BALANCED_POWER_ACCURACY - Use this setting to request location precision to within a city block (approximately 100 meters). This is considered a coarse level of accuracy, and is likely to consume less power. With this setting, the location services are likely to use WiFi and cell tower positioning.
- PRIORITY_HIGH_ACCURACY - Use this setting to request the most precise location possible. With this setting, the location services are more likely to use GPS to determine the location.
- PRIORITY_LOW_POWER - Use this setting to request city-level precision, which is an accuracy of approximately 10 kilometers. This is considered a coarse level of accuracy, and is likely to consume less power.
- PRIORITY_NO_POWER - Use this setting if you need negligible impact on power consumption, but want to receive location updates when available. With this setting, your app does not trigger any location updates, but receives locations triggered by other apps.
- Create the location request and set the parameters as shown in this code sample:

```
protected void createLocationRequest() {
    LocationRequest mLocationRequest = new
LocationRequest();
    mLocationRequest.setInterval(10000);
    mLocationRequest.setFastestInterval(5000);
    mLocationRequest.setPriority(LocationRequest.PRIOR
ITY_HIGH_ACCURACY);
}
```

Get Current Location Settings

To do this, create a *LocationSettingsRequest*.Builder, and add one or more location requests. The following code snippet shows how to add the location request that was created in the previous step:

```
LocationSettingsRequest.Builder builder = new
LocationSettingsRequest.Builder()
     .addLocationRequest(mLocationRequest);

Next check whether the current location settings are
satisfied:

PendingResult<LocationSettingsResult> result
=          LocationServices.SettingsApi.checkLocationSe
ttings(mGoogleClient,builder.build());
```

When the PendingResult returns, your app can check the location settings by looking at the status code from the LocationSettingsResult object. To get even more details about the the current state of the relevant location settings, your app can call the LocationSettingsResult object's *getLocationSettingsStates*() method.

Prompt the User to Change Location Settings

To determine whether the location settings are appropriate for the location request, check the status code from the LocationSettingsResult object. A status code of RESOLUTION_REQUIRED indicates that the settings must be changed. To prompt the user for permission to modify the location settings, call *startResolutionForResult*(Activity, int). This method brings up a dialog asking for the user's permission to modify location settings. The following code snippet shows how to check the location settings, and how to call *startResolutionForResult*(Activity, int).

```
result.setResultCallback(new
ResultCallback<LocationSettingsResult>()) {
     @Override
     public void onResult(LocationSettingsResult
result) {
         final Status status = result.getStatus();
         final LocationSettingsStates =
result.getLocationSettingsStates();
         switch (status.getStatusCode()) {
             case LocationSettingsStatusCodes.SUCCESS:
                 // All location settings are
satisfied. The client can
                 // initialize location requests here.
                 ...
                 break;
             case
LocationSettingsStatusCodes.RESOLUTION_REQUIRED:
                 // Location settings are not
satisfied, but this can be fixed
                 // by showing the user a dialog.
                 try {
                     // Show the dialog by calling
startResolutionForResult(),
                     // and check the result in
onActivityResult().
                     status.startResolutionForResult(
                         OuterClass.this,
                         REQUEST_CHECK_SETTINGS);
                 } catch (SendIntentException e) {
                     // Ignore the error.
                 }
                 break;
             case
LocationSettingsStatusCodes.SETTINGS_CHANGE_UNAVAILABL
E:
                 // Location settings are not
satisfied. However, we have no way
```

```
                    // to fix the settings so we won't
show the dialog.
                    ...
                    break;
            }
        }
 });
```

Next we will discuss how to receive location updates on a device.

12.11 Receiving Location Updates

If your app can continuously track location, it can deliver more relevant information to the user. For example, if your app helps the user find their way while walking or driving, or if your app tracks the location of assets, it needs to get the location of the device at regular intervals. As well as the geographical location (latitude and longitude), you may want to give the user further information such as the bearing (horizontal direction of travel), altitude, or velocity of the device. This information, and more, is available in the Location object that your app can retrieve from the fused location provider.

While you can get a device's location with *getLastLocation*(), a more direct approach is to request periodic updates from the fused location provider. In response, the API updates your app periodically with the best available location, based on the currently-available location providers such as WiFi and GPS (Global Positioning System). The accuracy of the location is determined by the providers, the location permissions you've requested, and the options you set in the location request.

The starting point of location update would be to get the last known location of the device. This also ensures that the app has a known location before starting the periodic location updates. The snippets in the following sections assume that your app has already retrieved the last known location and stored it as a Location object in the global variable *mCurrentLocation*.

For this service you require fine location detection, so that your app can get as precise a location as possible from the available location providers. Request this permission with the uses-permission element in your app manifest, as shown in the following example:

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/andr
oid"    package="com.google.android.gms.location.sampl
e.locationupdates" >

<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"
/>
</manifest>
```

Request Location Updates

Before requesting location updates, your app must connect to location services and make a location request. Once a location request is in place you can start the regular updates by calling *requestLocationUpdates*(). Do this in the *onConnected*() callback provided by Google API Client, which is called when the client is ready.

Depending on the form of the request, the fused location provider either invokes the LocationListener.*onLocationChanged*() callback method and passes it a Location object, or issues a PendingIntent that contains the location in its extended data. The accuracy and frequency of the updates are affected by the location permissions you've requested and the options you set in the location request object.

This section shows you how to get the update using the LocationListener callback approach. Call *requestLocationUpdates*(), passing it your instance of the GoogleApiClient, the LocationRequest object, and a LocationListener. Define a *startLocationUpdates*() method, called from the *onConnected*() callback, as shown in the following code sample:

```
@Override
public void onConnected(Bundle connectionHint) {
    ...
    if (mRequestingLocationUpdates) {
        startLocationUpdates();
    }
}

protected void startLocationUpdates() {
    LocationServices.FusedLocationApi.requestLocationU
pdates(
            mGoogleApiClient, mLocationRequest, this);
}
```

Notice that the above code snippet refers to a boolean flag, *mRequestingLocationUpdates*, used to track whether the user has turned location updates on or off.

Define the Location Update Callback

The fused location provider invokes the LocationListener.*onLocationChanged*() callback method. The incoming argument is a Location object containing the location's latitude and longitude. The following snippet shows how to implement the LocationListener interface and define the method, then get the timestamp of the location update and display the latitude, longitude and timestamp on your app's user interface:

```
public class MainActivity extends ActionBarActivity
implements
```

```
        ConnectionCallbacks,
OnConnectionFailedListener, LocationListener {
    ...
    @Override
    public void onLocationChanged(Location location) {
        mCurrentLocation = location;
        mLastUpdateTime =
DateFormat.getTimeInstance().format(new Date());
        updateUI();
    }

    private void updateUI()
{        mLatitudeTextView.setText(String.valueOf(mCur
rentLocation.getLatitude()));        mLongitudeTextVie
w.setText(String.valueOf(mCurrentLocation.getLongitude
()));
        mLastUpdateTimeTextView.setText(mLastUpdateTim
e);
    }
}
```

Stop Location Updates

Consider whether you want to stop the location updates when the activity is no longer in focus, such as when the user switches to another app or to a different activity in the same app. This can be handy to reduce power consumption, provided the app doesn't need to collect information even when it's running in the background. This section shows how you can stop the updates in the activity's *onPause*() method.

To stop location updates, call *removeLocationUpdates*(), passing it your instance of the GoogleApiClient object and a LocationListener, as shown in the following code sample:

```
@Override
protected void onPause() {
    super.onPause();
    stopLocationUpdates();
}

protected void stopLocationUpdates() {
    LocationServices.FusedLocationApi.removeLocationUp
dates(
            mGoogleApiClient, this);
}
```

Use a boolean, *mRequestingLocationUpdates*, to track whether location updates are currently turned on. In the activity's *onResume*() method, check whether location updates are currently active, and activate them if not:

```
@Override
public void onResume() {
    super.onResume();
    if (mGoogleApiClient.isConnected() &&
!mRequestingLocationUpdates) {
        startLocationUpdates();
    }
}
```

12.12 Adding Google Maps to Your App

Google maps are the most popular method of displaying maps. You can easily incorporate Google maps to your app. The following link shows you a step by step approach to incorporate Google maps to your app.

Activity

Activity 12.3

Develop a small app that uses GPS information and your current location (in terms of latitude and longitude)

Unit summary

In this unit, first we discussed about the various sensors available with Android framework. We also learnt about some important sensors and how to use them in an application. You will have to get familiar with the sensor coordinate system and the best practices when you develop apps. It will be useful to make your applications more efficient.

Thereafter, we discussed about how to incorporate location awareness to an app. We discussed about getting the current location of a device and also updating a location. Also we learnt about how to incorporate Google maps to your app. Sensor types supported by the Android platform are at the end of this unit as an Annexure.

References

https://developer.android.com/guide/topics/sensors/sensors_overview.html

https://developer.android.com/training/location/index.html

CC by 2.5 – Android Developer Forum (developer.android.com)

Unit 13

Connectivity and the cloud

Introduction

This unit will help you to understand the process of connecting your application to the world beyond the user's device. You will learn how to connect your application to other devices in the area and to the Internet, how to take backup and sync your applications data and how to deal with Push notifications in this unit.

Upon completion of this unit you should be able to:

Outcomes

- explain the modes of connecting an Android application with different devices wirelessly
- write a program to connect the app with outside world
- discuss the advantages of sending push notifications over polling method
- write a program to send Push notifications from the Android application

Terminology

cloud:	remote servers hosted on the Internet to store, manage, and process data
polling:	actively sampling the status of an external device by a client program as a synchronous activity
connectivity:	being connected to Internet

13.1 Connecting devices wirelessly

First of all we will see what a wireless network is. It is any type of computer network that uses wireless data connections for connecting network nodes. Besides enabling communication with the cloud, Android's wireless APIs also enable communication with other devices on the same local network, and even devices which are not on a network, but are physically nearby. The addition of Network Service Discovery (NSD) takes this further by allowing an application to seek out a nearby device running services with which it can communicate.

Integrating this functionality into your application helps you provide a wide range of features, such as playing games with users in the same room, pulling images from a networked NSD-enabled webcam, or remotely logging into other machines on the same network.

Using Network Service Discovery

Adding Network Service Discovery (NSD) to your app allows your users to identify other devices on the local network that support the services your application requests. This is useful for a variety of peer-to-peer applications such as file sharing or multi-player gaming. Android's NSD APIs simplify the effort required for you to implement such features.

Creating peer-to-peer (P2P) Connections with Wi-Fi

The Wi-Fi P2P APIs allow applications to connect to nearby devices without needing to connect to a network or hotspot. Wi-Fi P2P allows your application to quickly find and interact with nearby devices, at a range beyond the capabilities of Bluetooth. Also it provides fast data transfer with more security than the Bluetooth.

Using Wi-Fi P2P for Service Discovery

Let’ssee how to discover services published by nearby devices without being on the same network using Wi-Fi P2P. Using Wi-Fi Peer-to-Peer (P2P) Service Discovery allows you to discover the services of nearby devices directly without being connected to a network. You can also advertise the services running on your device. These capabilities help you communicate between apps, even when no local network or hotspot is available. While this set of APIs is similar in purpose to the Network Service Discovery APIs outlined in a previous section, implementing them in code is very different.

13.2 Performing network operations

This section explains the basic tasks involved in connecting to the network, monitoring the network connection (including connection changes), and giving users control over an app's network usage. It also describes how to parse and consume XML data.

These fundamental building blocks will enable you to create Android applications that download content and parse data efficiently, while minimizing network traffic.

Activity

Activity 13.1

State the dependencies and prerequisites when performing network operations.

Connecting to the Network

To perform the network operations, your application manifest must include the following permissions:

```
<uses-
permissionandroid:name="android.permission.INTERNET"/>
<uses-
permissionandroid:name="android.permission.ACCESS_NETWORK
_STATE"/>
```

Managing Network Usage

If your application performs a lot of network operations, you should provide user settings that allow users to control your app’s data habits, such as how often your app syncs data, whether to perform uploads/downloads only when on Wi-Fi, whether to use data while roaming, and so on. With these controls available to them, users are much less likely to disable your app’s access to background data when they approach their limits, because they can instead precisely control how much data your app uses.

Optimizing Network Data Usage

Over the life of a smartphone, the cost of a cellular data plan can easily exceed the cost of the device itself. From Android 7.0 (API level 24), users can enable Data Saver on a device-wide basis in order optimize their device's data usage, and use less data. This ability is especially useful when roaming, near the end of the billing cycle, or for a small prepaid data pack. Though this has been introduced as OS feature with Android 7.0, various vendors have introduced their own data saving mechanisms like ultra-data saving mode, on other Android versions which support VPN.

When a user enables Data Saver in Settings and the device is on a metered network, the system blocks background data usage and signals apps to use less data in the foreground wherever possible. Users can white-list specific apps to allow background metered data usage even when Data Saver is turned on.

Parsing XML Data

Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form. XML is a popular format for sharing data on the internet. Websites that frequently update their content, such as news sites or blogs, often provide an XML feed so that external programs can keep abreast of content changes. Uploading and parsing XML data is a common task for network-connected apps.

13.3 Considerations when transferring data

In mobile devices battery and memory is a limited resource. For your app to be considered ‘good’, it should seek to limit its impact on the battery life of its device.

By taking steps such as batching network requests, disabling background service updates when you lose connectivity, or reducing the rate of such updates when the battery level is low, you can ensure that the impact of your app on battery life is minimized, without compromising the user experience.

Optimizing Downloads for Efficient Network Access

Using the wireless radio to transfer data is potentially one of your app's most significant sources of battery drain. To minimize the battery drain associated with network activity, it's critical that you understand how your connectivity model will affect the underlying radio hardware. It goes on to propose ways to minimize your data connections, use prefetching, and bundle your transfers in order to minimize the battery drain associated with your data transfers

Minimizing the effect of regular updates

The optimal frequency of regular updates will vary based on device state, network connectivity, user behavior, and explicit user preferences.

Optimizing battery life discusses how to build battery-efficient apps that modify their refresh frequency based on the state of the host device. That includes disabling background service updates when you lose connectivity and reducing the rate of updates when the battery level is low.

How your refresh frequency can be varied to best mitigate the effect of background updates on the underlying wireless radio state machine.

Redundant downloads are redundant

The most fundamental way to reduce your downloads is to download only what you need. In terms of data, that means implementing REST APIs that allow you to specify query criteria that limit the returned data by using parameters such as the time of your last update.

Similarly, when downloading images, it is a good practice to reduce the size of the images server-side, rather than downloading full-sized images that are reduced on the client.

Modifying your download patterns based on the connectivity Type

When it comes to impact on battery life, not all connection types are created equal. Not only does the Wi-Fi radio use significantly less battery than its wireless radio counterparts, but the radios used in different wireless radio technologies have different battery implications

13.4 Syncing to the cloud with information delivery models

The Wearable Data Layer API, which is part of Google Play services, provides a communication channel for your handheld and wearable apps. The API consists of a set of data objects that the system can send and synchronize over the wire and listeners that notify your apps of important events with the data layer

Video – V11: Connectivity and the cloud

In this video you will be learning how to integrate cloud based services to your application.

URL: https://tinyurl.com/y86wfhhw

13.5 Push notification

A notification is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

Polling happens when the phone goes to the server with a certain interval and asks if there are any messages to process. Basically there are two ways to get messages to the phone:

1. Push messages (the server contacts the phone and tells it there are messages waiting)

2. Polling service (the phone contacts the server and ask for messages)

Creating a notification

You can specify the UI information and actions for a notification in a NotificationCompat.Builder object. To create the notification you cancall NotificationCompat.Builder.build(), which returns a Notification object containing your specifications. To issue the notification, you pass the Notification object to the system by calling NotificationManager.notify()

The following snippet illustrates a simple notification that specifies an activity to open when the user clicks the notification. Notice that the code creates a TaskStackBuilder object and uses it to create the PendingIntent for the action. This pattern is explained in more detail in the section Preserving Navigation when Starting an Activity:

```
NotificationCompat.Builder mBuilder =
        newNotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!");
// Creates an explicit intent for an Activity in your app
Intent resultIntent
=newIntent(this,ResultActivity.class);
// The stack builder object will contain an artificial back //stack
for the started Activity.
// This ensures that navigating backward from the Activity leads
//out of your application to the Home //screen.
TaskStackBuilder stackBuilder
=TaskStackBuilder.create(this);
// Adds the back stack for the Intent (but not the Intent //itself)
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent that starts the Activity to the top of the
//stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
        stackBuilder.getPendingIntent(
            0,
            PendingIntent.FLAG_UPDATE_CURRENT
        );
mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
// mId allows you to update the notification later on.
mNotificationManager.notify(mId, mBuilder.build());
```

Google Cloud Messaging as an alternative to polling

Every time an app polls a server to check if an update is required, you activate the wireless radio, drawing power unnecessarily, for up to 20 seconds on a typical 3G connection.

Compared to polling, where your app must regularly ping the server to query for new data, Google cloud messaging model allows your app to create a new connection only when it knows there is data to download.

The result is a reduction in unnecessary connections, and a reduced latency for updated data within your application.

Google Cloud Messaging for Android (GCM) is a lightweight mechanism used to transmit data from a server to a particular app instance. Using GCM, your server can notify your app running on a particular device that there is new data available for it.

GCM is implemented using a persistent TCP/IP connection. While it's possible to implement your own push service, it's best practice to use GCM. This minimizes the number of persistent connections and allows the platform to optimize bandwidth and minimize the associated impact on battery life

Google Cloud Messaging (GCM) is a free service that enables developers to send messages between servers and client apps. This includes downstream messages from servers to client apps, and upstream messages from client apps to servers.

For example, a lightweight downstream message could inform a client app that there is new data to be fetched from the server, as in the case of a "new email" notification. For use cases such as instant messaging, a GCM message can transfer up to 4kb of payload to the client app. The GCM service handles all aspects of queuing of messages and delivery to and from the target client app.

A GCM implementation includes a Google connection server, an app server in your environment that interacts with the connection server via HTTP or XMPP protocol, and a client app.

Activity

Activity 13.2

Select a commonly used Android application and identify how the application connect with other devices, with internet, how the application syncs and backup data.

Unit summary

Summary

This unit provided you an insight of how to connect different devices to your application wirelessly. Furthermore how to transfer data without draining the battery and how to get started with syncing to the cloud using information delivery models were discussed. At the end of the unit the drawbacks of using polling methods over push notifications and the Android code snippets used for sending push notifications were explained.

Unit 14

Publish to Android Market

Introduction

By following this unit you will gain theoretical knowledge on the revenue and distribution models, process of launching an Android application in a distributed environment. Knowledge gained from this unit will help you to make an informed choice of the business models to cater to specific requirements when launching a developed Android application to the market.

The provided video will demonstrate you how to publish an Android application to the Google Play.

Upon completion of this unit you should be able to:

Outcomes

- translate the appropriate distribution model to reach target audience
- describe available business models
- select appropriate business models for the applications
- demonstrate how to deploy the developed application in Android Market

Terminology

Localization: the process of making something local in character or restricting it to a particular place

14.1 How can you obtain an Android application?

As you already learnt in the previous units, Android mobile applications are software programs that may be installed on portable computing devices such as smartphones, tablets, some digital set-top boxes, laptops etc. Mobile applications market is the place where the buyers (consumers/users) and sellers (app developers) of mobile applications meet. These mobile application markets are dedicated retail platforms known as AppStores which can be accessible through the consumer’s device. The main objective of an AppStore is to serve as a host to initially source online to distribute to potential users through a data connection and accessing device.

An end-user can obtain an Android application in two main ways. Applications may be pre-installed or downloaded on-demand. Pre-installed mobile applications are selected by device manufacturers and usually include the following.

- utilities (for example, calendars, alarm clocks, camera apps)
- services (for example, weather apps, Google maps, a compass, world clock)
- entertainment (for example, music, video, games)
- communications applications (for example, chat applications)

Mobile applications may be downloaded and installed by consumers in several ways:

- Via the device - end-user can directly access the device manufacturer’s app store through a menu on the device, and download and install a mobile application. Access may be enabled through 3G or Wi fi networks (in Wifi-enabled devices).
- Via the Internet directly onto the mobile device - the end-user can access the manufacturer’s app store via the web browser on his/her device.
- Via the Internet and transfer to the mobile device - the end-user can access the manufacturer’s app store via the web browser on a personal computer and then transfer over Bluetooth or an external memory to install in the mobile device.

14.2 App Stores

Mobile applications are commonly made available through aggregators with online stores. Mobile applications aggregators are not new to the communications industry. Online stores offering mobile phone ringtones, themes and other applications have existed since the late 1990s.

App stores may be categorized as:

Device manufacturers—including Apple’s App Store, Nokia’s Ovi, and Blackberry’s App World. These stores can be used only by consumers with the appropriate manufacturer’s device and proprietary software.

Operating system developer—including Android Market and Microsoft Windows Mobile. These stores can be accessed by consumers with devices from multiple handset manufacturers via the proprietary operating system software (OS). For example, Android mobile applications can be used on Motorola, HTC and Samsung devices etc. which have Android Operating system.

Mobile network operator—including Telstra, Verizon and Optus. These stores can only be accessed by consumers with service contracts with the network operator. Consumers can use multiple handset brands to access these stores.

Independent—including app stores operated as independent commercial concerns, or by developers such as GetJar and Mobango. Access to these stores is not dependent on the brand of device used, service provider or proprietary software.

Activity

Activity 14.1

Based on the discussion in Sections 14.1 and 14.2, compare the AppStore classifications for publishing the following two applications.

AppA - health diary application to update calorie intake, number of hours of exercise etc for Hutch mobile users.

AppB - advanced colouring app which can edit and use existing art for children

14.3 Revenue Models

Revenue models are different ways that entrepreneurs can earn money from their mobile applications. Under this topic we will be explaining three revenue models: Free Model, Paid model and Paymium model.

Free Model

Free model is also referred to as freemium model. In this model, users don’t pay to download or use an application. The main advantage of removing the barrier of price increases the likelihood that users in the target market will download and try. This initiation can help increase awareness for the application and grow the distribution among the target user base.

The main source of revenue for freemium model is through advertisements. Applications can display advertisements and in return generate “ad revenue”. It is also important to select appropriate advertisements to suit the application and its target market. Inappropriate advertisements can reduce the interactions with the users and their retention to continue usage.

Paid Model

In this model, users pay once to download an application. Then, use the functionalities provided by that application without any further payments. The paid model is suitable for users who prefer to pay once to get the full application experience, without in-app purchases. We will discuss the in-app model in the next section. Often the application developers need to pay careful attention to include premium experiences through attractive design and functionality for applications intend to be distributed using paid model. Since there is only a one-time payment it is important to drive the marketing strategy to acquire more users. It is advisable that the developers make sure that their app's title, icon, description, preview, screenshots, and other marketing communications effectively showcase the premium nature of the application.

Paymium Model

Paymium model is essentially a combination of the paid and freemium models. Users pay to download the application and have the option to buy additional features, content, or services as they continue to use the application. Paymium model is suitable for users who would like to engage interacting with the application step-by-step depending on their requirements. Unlike in the paid model, this model allows a user to decide whether to acquire more features as and when needed rather than to pay a hefty price. In-app purchases can reduce the chances of disappointments in paid model where the user is not fully satisfied with the functionalities and features offered by an application. Another important aspect is the developers have the flexibility to add more customizable features.

Offering Subscriptions

This model offer an easy experience for digital subscriptions. In-app purchase APIs provide a simple, standardized way to implement auto-renewable and non-renewing subscriptions to content or services. With in-app purchase subscriptions, it is possible to the price and duration of subscriptions. Duration of subscriptions may be seven days, one month, two months, three months, six months, or one year. Often AppStores allow the users to manage the subscription of their applications. For example, in Apple AppStore a user can manage the subscriptions through the Apple user ID account.

Auto-Renewable Subscriptions: This subscription type gives users access to content that is regularly updated. At the end of each subscription duration, the subscription will renew automatically until a user chooses to turn off auto-renewal. Often free trials are offered for such applications using this business model. The length of the subscription determines how long the free trial can be. For example, for a monthly subscription you can offer users a 7-day or 1-month free trial. When users sign up for a subscription with a free trial, their subscription will begin immediately but they won't be billed until the free trial period is over. They will then continue to be billed on a recurring basis, unless the users turn off auto-renewal.

Non-Renewing Subscriptions: This subscription type give users access to content or services for a limited duration Non-renewing subscriptions require users to renew each time a subscription ends and may notify them when subscriptions is due to expire with a prompt to purchase a new subscription. Often free trials are offered for such applications using this business model. When users sign up for a subscription they won't be billed until the free trial period is over. Then the subscription will be billed to a pre-defined period and not on a recurring basis as there is no auto-renewal.

Activity

Activity 14.2

Identify the revenue model for the Equalizer Android application. Include the knowledge gained from the previous sections.

14.4 Google Play

The Google Play Store or Google Play (originally as the Android Market) is a digital distribution service operated and developed by Google. It serves as the official app store for the Android operating system. GooglePlay allows users to browse and download applications developed with the Android SDK and published through Google. In Android devices, Google Play Store is an official pre-installed application. This pre-installed application provides access to the Google Play store for users to browse and download music, books, magazines, movies, television programs, and other applications from Google Play.

The Devices segment of Google Play is not accessible through the Play Store. The Play Store application is not open source. Only Android devices that comply with Google's compatibility requirements may install and access Google's closed-source Play Store application, subject to entering into a free-of-charge licensing agreement with Google.

Applications are available through Google Play use freemium or paid business models. They can be downloaded directly to an Android or Google TV device through the Play Store mobile app, or by deploying the application to a device from the Google Play website. Many applications can be targeted to specific users based on a particular hardware attribute of their device, such as a motion sensor (for motion-dependent games) or a front-facing camera (for online video calling). Such specific application are allowed to download onto devices with appropriate in-built hardware or sufficient capabilities.

14.5 Process of Publishing an Android Application

Once the application is developed, the next process is to make it available for the users to download and use it. To reliably distribute an Android application for the users to download, an AppStore is required. The steps of the publishing process can be summarized as follows.

1. Select an appropriate AppStore

2. Read and understand the policies and agreements of the selected AppStore
3. Quality test
4. Determine the content rating for the Android application
5. Determine the country (or the countries) to distribute
6. Confirm the overall size, platform and the screen compatibility ranges
7. Decide the revenue model
8. Decide how to bill or collect the revenue (e.g. In-App or using Google Pay)
9. Set the price (or prices)
10. Localization
11. Prepare promotional graphics, videos and screencasts
12. Build and upload
13. Plan for Beta release
14. Complete AppStore listing
15. Support users after launch

Video – V12: Publish to Android Market

This video shows the steps to follow when building your application to release and releasing the application to users.

URL: **https://tinyurl.com/Publish-to-Android-Market**

Activity

Activity 14.3

Based on the content covered in Section 14.3, 14.4 and 14.5, in your opinion what are the most significant steps that can have a significant impact (in terms of popularity and revenue loss/gain) if the revenue model is changed during the publishing process of the Android application.

Unit summary

This unit covered the topics on revenue and distribution models, process of launching an Android application in a distributed environment. To further enhance your understanding on these topics activities and a supplementary video were provided.

References

1. Australian Communications and Media Authority, “Emerging Business Models in the Digital Economy - The Mobile Applications Market”, May 2011, downloaded from: http://www.acma.gov.au/webwr/_assets/main/lib310665/emerging_business_models.pdf.

2. Nielsen, "Smartphones: So Many Apps, So Much Time," July 2014.

3.“Choosing a Business Model”, downloaded from: https://developer.apple.com/app-store/business-models/

Unit 15

Performance

Introduction

In this unit you will learn to analyze the performance of an Android application. First you will identify and use tools to visualize performance which would enable you to analyze performance in various aspects of an Android application. Then you will learn how to improve the performance by optimizing memory usage and minimizing the power consumption by selecting appropriate techniques.

Upon completion of this unit you should be able to:

Outcomes

- identify tools to visualize performance of an Android app
- analyse performance of a developed application to identify the drawbacks
- apply techniques to reduce memory usage when programming for Android
- analyse battery life and apply techniques to optimize battery usage

Terminology

profiling:	analyse performance using a tool
optimize:	make the most effective use of something

15.1 Performance Profiling

With Android applications, it is possible to perform various things on devices such as running tasks in background, playing music or videos, or connecting with different networks like, Wi-Fi, 4G, and Bluetooth. So applications running in the device may consume its resources in many ways. For example displaying pixels on the screen involves four primary pieces of hardware. In simple terms the Central Processing Unit (CPU) computes display lists, the Graphical Processing Unit (GPU) renders images to the display, the memory stores images and data, and the battery provides electrical power. Each of these pieces of hardware has constraints; pushing or exceeding those constraints causes your app to be slow, have bad display performance, or exhaust the battery. Performance profiling means investigating and analyzing a mobile application’s runtime behavior to decide how to optimize the performance the programs involved. In the next section, we will discuss a popular profiling tool and its sub-tools.

Performance Profiling Tools

To discover what causes your specific performance problems, you need to investigate in depth, i.e. use tools to collect data about your app execution behavior, organize that data in lists and graphics, and analyze what you see. Your mobile device together with Android Studio provides profiling tools to record and visualize the rendering, do computations, analyze memory and battery usage of your app. Android Monitor, which is introduced below is one such tool.

15.2 Android Monitor Overview

Android Monitor provides various sub-tools that you can use to profile the performance of an app so that you can optimize, debug, and improve them. It lets you monitor the following aspects of your apps from a hardware device or the Android Emulator:

- Log messages, either system or user defined
- Memory, CPU, and GPU usage
- Network traffic (hardware device only)

It lets you capture data as your app runs and stores it in a file that you can analyze in various viewers. You can also capture screenshots and videos of your app as it runs.

Android Monitor has a main window that contains performance monitors such as logcat, Memory, CPU, GPU, and Network Monitors. From this window, you can select a device and app process to work with, terminate an app, collect *dumpsys* system information, and create screenshots and videos of the running app. dumpsys is an Android tool that runs on the device and dumps information about the status of system services.

Usage of these performance monitors will be described in the screen cast on ‘Performance monitors’. A brief description of each performance monitor with a screen shot is given below for you to see what it looks like.

Those screen shots are taken from the weather app shown in Figure 15.1 below.

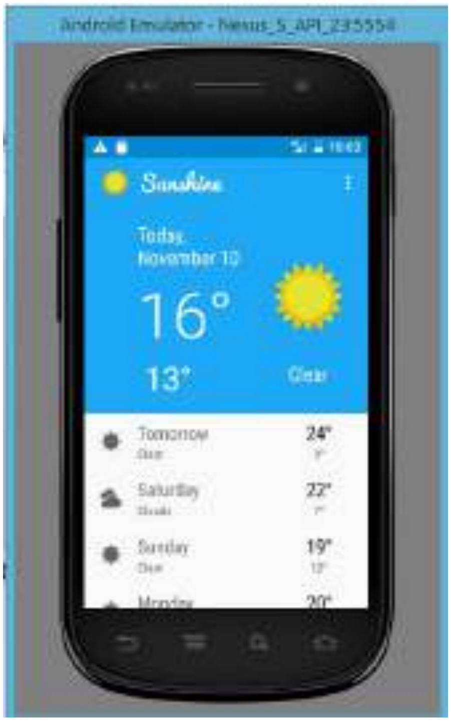


Figure 15. 1 weather app

Video -V13: Performance Profiling

In this video you will see how different profiling tools in Android Monitor are used. These tools are, CPU monitor, GPU monitor, memory monitor, network monitor and logcat.

URL: **https://tinyurl.com/Performance-Profiling**

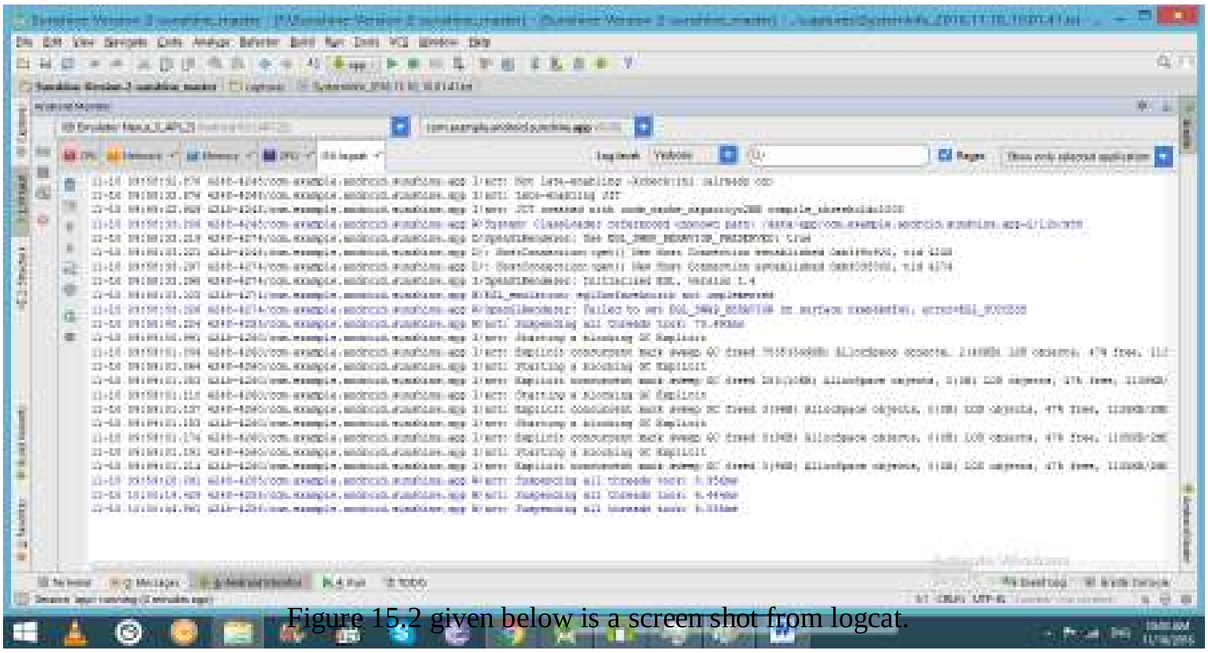


Figure 15.2 given below is a screen shot from logcat.

Figure 15.2 Logcat

Memory Monitor – We use the Memory Monitor an in figure 15.3 to evaluate memory usage and find de-allocated objects, locate memory leaks, and track the amount of memory that the connected device is using.

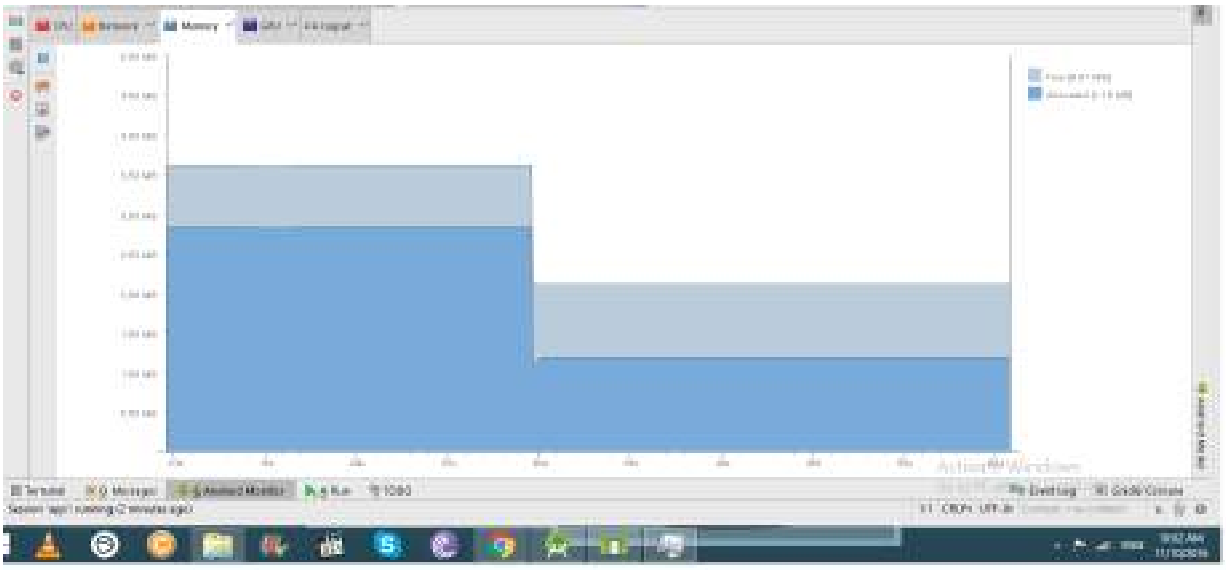


Figure 15.3 Android Monitor – Memory Footprint by running Weather app

```
Applications Memory Usage (kB):

Uptime: 269104 Realtime: 269104

 ** MEMINFO in pid 4248 [com.example.android.sunshine.app] **

Pss  Private  Private  Swapped     Heap     HeapHeap

                 Total     Dirty     Clean     Dirty     Size     Alloc     Free

                ------    ------    ------    ------    ------    ------    ------

  Native Heap     3361      3276         0         0     14336     13865       470

Dalvik Heap     1634      1524         0         0     3684      2920      764

Dalvik Other      341       288         0         0

        Stack      120       120         0         0

       Cursor        4         4         0         0

Ashmem         2         0         0         0

    Other dev        4         0         4         0

     .so mmap     1109       300         0         0

    .apkmmap      168         0         8         0

    .ttfmmap      113         0        72         0

    .dexmmap     2112         4      2108         0

    .oat mmap     1057         0       152         0

    .art mmap      834       436         0         0

   Other mmap       38         8         4         0

      Unknown      169       168         0         0

        TOTAL    11066      6128      2348         0     18020     16785      1234


App Summary

Pss(KB)
                       ------

           Java Heap:     1960

         Native Heap:     3276

                Code:     2644

               Stack:      120

            Graphics:        0

       Private Other:      476

              System:     2590


               TOTAL:    11066      TOTAL SWAP (KB):        0
```

CPU Monitor – As in Figure 15.4, CPU Monitor can be used to display CPU usage in real time and the percentage of total CPU time (including all cores) used in user and kernel mode.

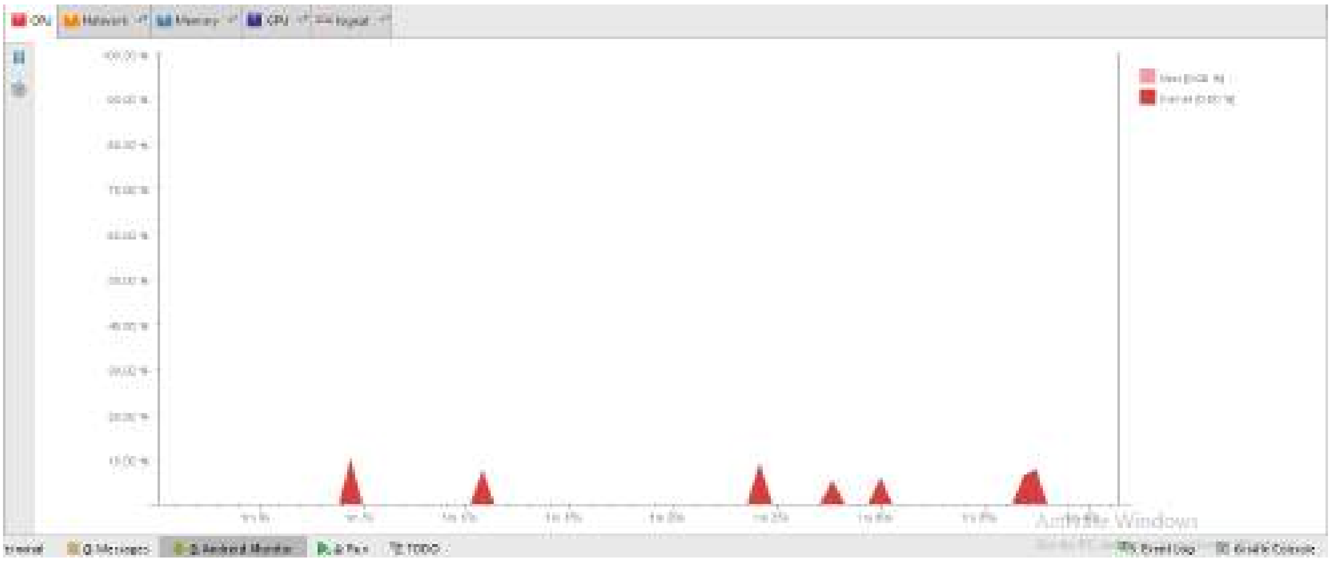


Figure 15.4 CPU Monitor

GPU Monitor – You can use the GPU Monitor as in figure 15.5 for a visual representation of how much time it takes to render the frames of a UI window. This information can be used to optimize the code that displays graphics and conserve memory.

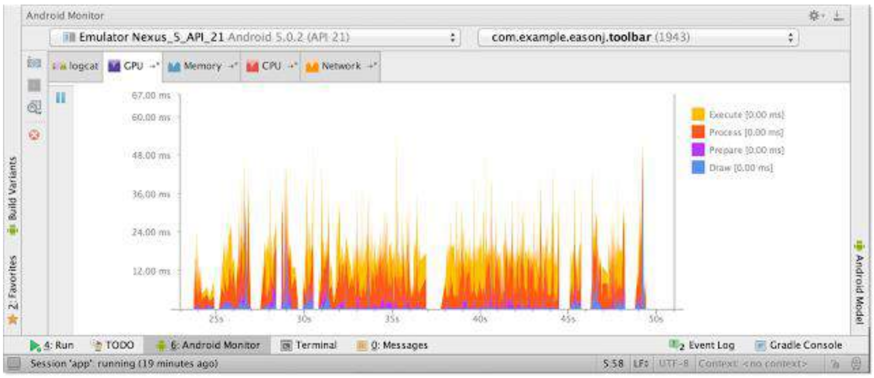
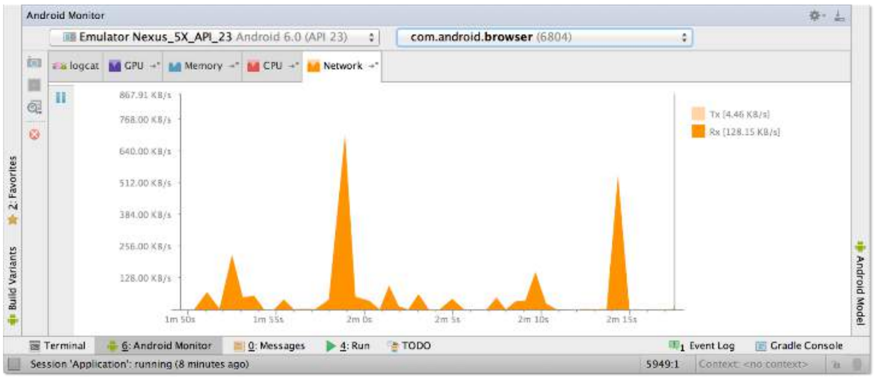


Figure 15.5 GPU monitor



Emulator Nexus_5X_API_23 Android 6.0 (API 23)

com.android.browser (6804)

Session 'Application': running (8 minutes ago)

5949:1

Context: <no context>

Figure 15.6 Android

Activity

Activity 15.1

What are the sub-tools provided by the Android Monitor to analyse the performance of an app?

Data Analysis

Android Monitor also lets you capture various types of data about your app while it is running and stores it in a file, which you can access later. It lists these files in the *Captures* window and you may observe how it works in the screen cast provided for this unit.

15.3 Android Monitor Basics

In this section you will be guided how to use the Android Monitor step by step. The screen cast on Android Monitor Basics will demonstrate this activity.

Android Monitor is integrated into the Android Studio main window:

- To display Android Monitor, click 6: Android Monitor, which by default is at the bottom of the main window.
- To hide Android Monitor, click 6: Android Monitor again.
- Or select **View**>**Tool Windows**>**Android Monitor**.

Note: If you don't see the sidebar buttons, you can display them by selecting **View**>**Tool Buttons**.

A screen shot of Android Monitor is given below in Figure 15.7:

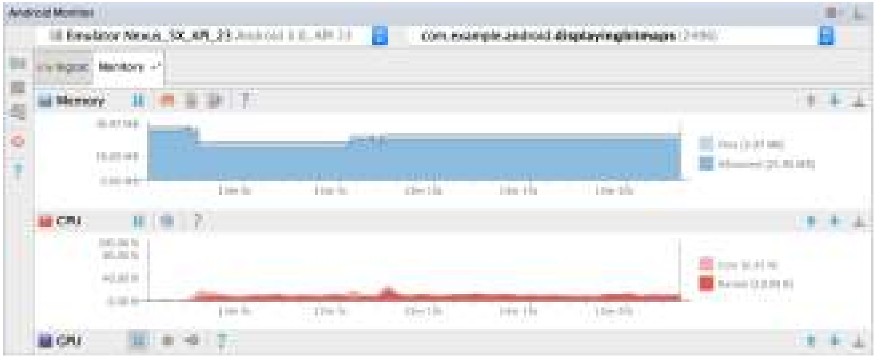


Figure 15.7 Android monitor

Before start using Android Monitor, you need to set up the environment, as well as the hardware device or emulator. All of the monitors require the following:

- If you want to run your app on a hardware device (as opposed to the emulator), connect it to the USB port. Make sure your development computer detects your device, which often happens automatically when you connect it.
- Enable ADB integration by selecting **Tools**>**Android**>**Enable ADB Integration**. **Enable ADB Integration** should have a check mark next to it in the menu to indicate it's enabled.

All but the logcat Monitor have these additional requirements:

- EnableUSB debugging in **Developer Options** on the device or emulator.

  In your app, set the debuggable property to true in the manifest or build.gradle file (it’s initially set by default).

The GPU Monitor has this requirement as well:

- For Android 5.0 (API level 21) and Android 5.1 (API level 22), in **Developer Options** on the device or emulator, set **Profile GPU rendering** to ‘**In adb shell dumpsysgfxinfo**’.

The Network Monitor and the Video Capture tool work with a hardware device only, not with the emulator.

15.4 Profiling a Running App in Android Monitor

After you have met the prerequisites and connected a hardware device, you are ready to profile an app in Android Monitor. To start;

1. Open an app project and run the app on a device or emulator.
2. Display Android Monitor and click the tab for the monitor you want to view.
3. follow the instructions about using the monitor:
   - o logcat Monitor

- Memory Monitor
- CPU Monitor
- GPU Monitor
- Network Monitor

Switching between Devices and Apps

By default, Android Monitor displays data for your most recently run app. You can switch to another device and app as needed. In addition to currently running apps, you can view information about apps that are no longer running so you can continue to see any information about them that you gathered previously.

At the top of the Android Monitor main window, there are two menus listing devices and processes. To switch to another device, process, or both:

1. Select the device or emulator.

   The Device menu lists the devices and emulators that are running or have run during your current unit. There are various status messages that can appear in the Device menu:

   - **DISCONNECTED -** You closed an emulator or unplugged a device from the computer.
   - **UNAUTHORIZED -** A device needs you to accept the incoming computer connection. For example, if the connected device displays an *Allow USB Debugging* dialog, click **OK** to allow the connection.
   - **OFFLINE -** Android Monitor can’t communicate with a device, even though it has detected that device.

2. Select the process.

   The Process menu lists the processes that are running or have run during your current unit. If a process is no longer running, the menu displays status of **DEAD**.

Terminating an App and removing it from a Device

To stop an app you have run from Android Studio, select the device and the process in the Android Monitor menus and click ‘Terminate Application’. The process status changes to **DEAD** in the Processes menu. The emulator or device may continue to run, but the app closes. Any running monitors in Android Monitor would stop.

To remove an app from a device you use for development, use the normal uninstall procedure on the device.

If you run a new version of an app from Android Studio that’s been already installed on a hardware device, the device displays an *Application*

Installation Failed dialog. Click **OK** to install the new version of the app.

From the Android Monitor main window, you can also do the following:

- Examine dumpsys system information.
- Take a screen capture of the device.
- Record a video from the screen.

15.5 How Android Manages Memory

Random-access memory (RAM) is a valuable resource in any software development environment, but it is even more valuable on a mobile operating system where physical memory is often constrained. Although Android's Dalvik virtual machine or Android Runtime (Introduced later, with KitKat) performs routine garbage collection, this doesnot allow you to ignore when and where your app allocates and releases memory. Techniques such as sharing memory, allocating and reclaiming app memory, restricting app memory and switching apps have been implemented in Android. In this course these techniques are not discussed in detail.

You may read the following pages found in Android Developer Forum https://developer.android.com/training/articles/memory.html to understand how Android manages app processes and memory allocation, and how you can proactively reduce memory usage while developing for Android.

You should consider RAM constraints throughout all phases of development, especially during app design. There are many ways you can design and write code that lead to more efficient results, through aggregation of the same techniques applied over and over.

Here are few techniques that you can apply while designing and implementing your app to make it more memory efficient.

Use services sparingly

If your app needs a service to perform work in the background, do not keep it running unless it is actively performing a job. Bbe careful to never leak your service by failing to stop it when its work is over.

Release memory when your user interface becomes hidden

When the user navigates to a different app and your user interface (UI) is no longer visible, you should release any resources that are used by only your UI. Releasing UI resources at this time can significantly increase the system's capacity for cached processes, which has a direct impact on the quality of the user experience.

Release memory as memory becomes tight

During any stage of your app's lifecycle, the *onTrimMemory()* callback also tells you when the overall device memory is getting low. You should respond by further releasing resources based on the memory levels delivered by *onTrimMemory()*.

Because the *onTrimMemory()* callback was added in API level 14, you can use the *onLowMemory()*callback as a fallback for older versions.

Check how much memory you should use

As mentioned earlier, each Android-powered device has a different amount of RAM available to the system and thus provides a different heap limit for each app. You can call *getMemoryClass()* to get an estimate of your app's available heap in megabytes. If your app tries to allocate more memory than is available here, it will receive an *OutOfMemoryError*.

Large heap size is not the same on all devices so that when running on devices that have limited RAM, the large heap size may be exactly the same as the regular heap size. So even if you request the large heap size, you should call *getMemoryClass()* to check the regular heap size and strive to stay below that limit always.

Avoid wasting memory with bitmaps

When you load a bitmap, keep it in RAM only at the resolution you need for the current device's screen. Scale it down if the original bitmap is a higher resolution. You may keep in mind that an increase in bitmap resolution results in a corresponding in-memory needed, because both the X and Y dimension increase.

Use optimized data containers

Take advantage of optimized containers in the Android framework, such as *SparseArray*, *SparseBooleanArray*, and *LongSparseArray*. The generic *HashMap*implementation can be quite memory inefficient because it needs a separate entry object for every mapping.

Be aware of memory overhead

Be knowledgeable about the cost and overhead of the language and libraries you are using, and keep this information in mind when you design your app, from start to finish. Often, things on the surface that look trivial may in fact have a large amount of overhead.

Be careful with code abstractions

Often, developers use abstractions simply as a "good programming practice," because abstractions can improve code flexibility and maintenance. However, generally they require a fair amount more code that needs to be executed, requiring more time and more RAM for that code to be mapped into memory. Soyou may avoid abstractions if they are not giving a significant benefit.

Be careful about using external libraries

External library code is often not written for mobile environments and can be inefficient when used for work on a mobile client. At the very least, when you decide to use an external library, you should assume you are taking on a significant porting and maintenance burden to optimize the library for mobile. Plan for that work up-front and analyse the library in terms of code size and RAM footprint before deciding to use it at all.

Optimize overall performance

Some of the actions can be taken to optimize your app's performance in various ways to improve its responsiveness and battery efficiency are given below.

- Avoid creating unnecessary objects
- Prefer static over virtual
- Use static final for constants
- Avoid internal getters/setters
- Use enhanced for loop syntax
- Consider package instead of private access with private inner classes
- Avoid using floating-point
- Use native methods carefully
- measure your existing performance

Further details on implementing these techniques are given in the Android Developer Forum in https://developer.android.com/training/best-performance.html

Use zipalign on your final Android Application Package (APK)

Android Application Package (APK) is the packaging format used by Android operating system in distribution and installation of Android apps. *Zipalign*is a tool that optimize the way an application is packaged.

If you do any post-processing of an APK generated by a build system, then you must run *zipalign* on it to have it re-aligned. Failing to do so can cause your app to require significantly more RAM, because things like resources can no longer be mapped from the APK.

Use multiple processes

If it is appropriate for your app, an advanced technique that may help you manage your app's memory is dividing components of your app into multiple processes. This technique must always be used carefully and **most apps should not run multiple processes**, as it can easily increase—rather than decrease—your RAM footprint if done incorrectly. It is primarily useful to apps that may run significant work in the background as well as the foreground and can manage those operations separately.

Activity

Activity 15.2

List down what techniques you used to optimize memory in the app you developed. Briefly describe them in the on-line discussion forum under *managing app memory*. You should give feedback to what your peers have written and also invite them to give feedback to you.

15.6 Battery Analysis

The battery-life impact of performing application updates depends on the battery level and charging state of the device. The impact of performing updates while the device is charging over AC is negligible, so in most cases you can maximize your refresh rate whenever the device is connected to a wall charger. Conversely, if the device is discharging, reducing your update rate helps prolong the battery life.

Similarly, you can check the battery charge level, potentially reducing the frequency of or even stopping the updates when the battery charge is nearly exhausted.

Battery Historian Walkthrough

This walkthrough shows the basic usage and workflow for the Batterystats tool and the Battmery Historian script.

Batterystats collects battery data from your device, and Battery Historian converts that data into an HTML visualization that you can view in your Browser. Batterystats is part of the Android framework, and Battery Historian script is open-sourced and available on GitHub at https://github.com/google/battery-historian.

It is good for:

- Showing you where and how processes are drawing current from the battery and
- Identifying tasks in your app that could be deferred or even removed to improve battery life.

Video – V14: Battery Analysis

This video shows you how to analyse battery power and generate a report.

URL: **https://tinyurl.com/ya2lsmko**

Battery Historian Charts

The Battery Historian chart graphically illustrate power-relevant events over time.

Each row shows a colored bar segment when a system component is active and thus drawing current from the battery. The chart does *not* show *how much* battery was used by the component, only that the app was active.

Charts are organized by category.

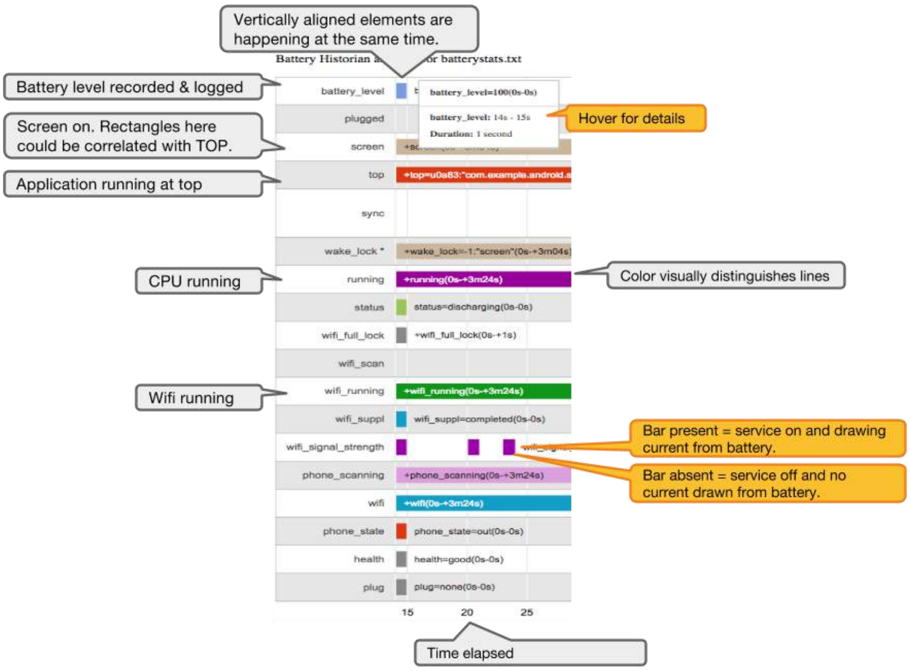

Vertically aligned elements are happening at the same time.
Battery Historian a... or batterystats.txt
Battery level recorded & logged
Screen on. Rectangles here could be correlated with TOP.
Application running at top
CPU running
Wifi running
battery_level
plugged
screen
top
sync
wake_lock *
running
status
wifi_full_lock
wifi_scan
wifi_running
wifi_suppl
wifi_signal_strength
phone_scanning
wifi
phone_state
health
plug
battery_level=100(0s-0s)
battery_level: 14s - 15s
Duration: 1 second
+top=u0a83:"com.example.android.s
+wake_lock=-1:"screen"(0s-+3m04s)
+running(0s-+3m24s)
status=discharging(0s-0s)
+wifi_full_lock(0s-+1s)
+wifi_running(0s-+3m24s)
wifi_suppl=completed(0s-0s)
+phone_scanning(0s-+3m24s)
+wifi(0s-+3m24s)
phone_state=out(0s-0s)
health=good(0s-0s)
plug=none(0s-0s)
15
20
25
Hover for details
Color visually distinguishes lines
Bar present = service on and drawing current from battery.
Bar absent = service off and no current drawn from battery.
Time elapsed

Figure 6 Battery Historian Chart

Battery usage categories

Given below is a list of battery usage categories.

- **battery_level**: When the battery level was recorded and logged. Reported in percent, where 093 is 93%. Provides an overall measure of how fast the battery is draining.
- **top**: The application running at the top; usually, this is the application that is visible to the user. If you want to measure battery drain while your app is active, make sure it is the top app. If you want to measure battery drain while your app is in the background, make sure it's*not* the top app.
- **wifi_running**: Shows that the Wi-Fi network connection was active.
- **screen**: Screen is turned on.
- **phone_in_call**: Recorded when the phone is in a call.
- **wake_lock**: App wakes up, grabs a lock, does small work, then goes back to sleep. This is one of the most important pieces of information. Waking up the phone is expensive, so if you see lots of short bars here, that might be a problem.

- **running**: Shows when the CPU is awake. Check whether it is awake and asleep when you expect it to be.
- **wake_reason**: The last thing that caused the kernel to wake up. If it's your app, determine whether it was necessary.
- **mobile_radio**: Shows when the radio was on. Starting the radio is battery expensive. Many narrow bars close to each other can indicate opportunities for batching and other optimizations.
- **gps**: Indicates when the GPS was on. Make sure this is what you expect.
- **sync:** Shows when an app was syncing with a backend. The sync bar also shows which app did the syncing. For users, this can show apps where they might turn syncing off to save battery. Developers should sync as little as possible and only as often as necessary.

Working with Batterystats and Battery Historian

You should watch the screen cast and do the following steps to learn how to use Batterystats and Battery Historian tools.

For more details you may go to following links in Android Developer Forum.

- https://developer.android.com/studio/profile/battery-historian.html
- https://developer.android.com/training/.../battery-monitoring.html

15.7 Optimizing Battery Life

For your app to be ‘good’, it should seek to limit its impact on the battery life of its device. After studying this section, you will be able to build apps that modify their functionality and behavior based on the state of its device.

By taking steps such as batching network requests, disabling background service updates when you lose connectivity, or reducing the rate of such updates when the battery level is low, you can ensure that the impact of your app on battery life is minimized, without compromising the user experience.

Here we briefly describe the steps you can take to optimize battery life. You should go to Android Developer Studio for more details.

(https://developer.android.com/training/monitoring-device-state/index.html) What is described in detail in this page are summarized below.

1. Reducing Network Battery Drain

Requests that your app makes to the network are a major cause of battery drain because they turn on the power-hungry mobile or Wi-Fi radios. Beyond the power needed to send and receive packets, these radios spend extra power just turning on and keeping awake. Something as simple as a network request every 15 seconds can keep the mobile radio on continuously and quickly use up battery power.

2. Optimizing for Doze and App Standby

Starting from Android 6.0 (API level 23), Android introduces two power-saving features that extend battery life for users by managing how apps behave when a device is not connected to a power source. *Doze* reduces battery consumption by deferring background CPU and network activity for apps when the device is unused for long periods of time. *App Standby* defers background network activity for apps with which the user has not recently interacted.

3. Monitoring the Battery Level and Charging State

When you are altering the frequency of your background updates to reduce the effect of those updates on battery life, check the current battery level.You can stopyour updates when the battery charge is nearly exhausted.

4. Determining and Monitoring the Docking State and Type

Android devices can be docked into several different kinds of docks. These include car or home docks and digital versus analog docks. The dock-state is typically closely linked to the charging state as many docks provide power to docked devices.

5. Determining and Monitoring the Connectivity Status

Some of the most common uses for repeating alarms and background services is to schedule regular updates of application data from Internet resources, cache data, or execute long running downloads. But if you arenot connected to the Internet, or the connection is too slow to complete adownload there in no use waking the device to schedule the update.

You can use the *ConnectivityManage*r to check the Internet connectivity.

6. Manipulating Broadcast Receivers On Demand

The simplest way to monitor device state changes is to create a *BroadcastReceiver* for each state you are monitoring and register each of them in your application manifest. Then within each of these receivers you simply reschedule your recurring alarms based on the current device state.

A side-effect of this approach is that your app will wake the device each time any of these receivers is triggered—potentially much more frequently than required.

Activity

Activity 15.3

- List down names of the techniques listed above that can be used in the app you developed.
- How can you change your app to optimize the battery life?

Unit summary

There are five main performance monitors provided by the tool, Android Monitor - and they can make you visualize the behavior and performance of your app. There are many ways that you can improve the performance of your application but optimizing memory and battery life are very important. Several techniques for optimizing memory and battery life were discussed in this unit.

Unit 16

Security

Introduction

Android is considered to be the most widely used mobile operating system in the world today. Being Open Source it is very much vulnerable for security breaches if the security is not managed properly.

In this unit, first you will learn about security provided by the operating system and how to analyse an application to understand what security features are built into it. Then you will study device management policies, which will enable you to design and develop applications for devices that enforce security policies.

Upon completion of this unit you should be able to:

Outcomes

- identify possible security concerns of an Android application
- identify the device management policies implemented in different setups
- identify how security can be enhanced by using device management policies
- design and develop applications that enforces security policies on devices

Terminology

malicious app:	software that brings harm to the mobile device
cryptography:	coding or decoding messages to keep them secure

16.1 Security Concerns of an Android Application

There are many security concerns regarding Android applications. One major security threat is over the limit access permission given and requested by apps. When an app is downloaded from Google Play, users ignore the extent of permission this app should have on their devices. Very often, app developers also do not have a clear understanding as to what permissions a mobile application actually needs and request overzealous and irrelevant permission. In any operating system, this kind of situations expose the user to a source of potential risks.

However, risks increase when users download apps from unidentified sources to avoid paying the fee. Anyone can create a malicious app and upload it on the Internet. This can result in downloading a malicious app or one that has been modified to automatically install a virus on Android devices.

Fact that Android is Open Source makes it more vulnerable for malware and malicious software attacks. However, being Open Source there is a large community of experts reviewing it and developing patches. Anyway, users are being hacked without them knowing that they are hacked. Another major security threat faced by Android platform due to the option of customizing the operating system. Device manufacturers can modify the OS to make it function optimally on their device. Moreover, users also can modify the OS, integrating customization layers or launchers. These practices leads to more security breaches.

Another major issue with Android is fragmentation. It means that there exist multiple versions of Android, even on latest devices. Since some devices are never updated to the latest version they will not have the latest security updates. It is also difficult to take appropriate security measures or educate the users about potential vulnerabilities because user experience on each device is different.

16.2 Security Provided by the OS

Android has many security features built into the operating system that significantly reduce the frequency and impact of application security issues we already discussed. Latest versions of the operating system is designed in a way that you can build your apps with default system and file permissions without worrying too much about security.

Some of the core security features provided by Android OS are:

- Android Application Sandbox that isolates your app data and code execution from other apps by running each application other than system apps with a different user.
- Permission reflects Linux permissions groups and corresponding user related to app will be assign to particular user group to assign permissions. So, access restrictions guaranteed even in the kernel level.
- An application framework which provides common security functionality such as cryptography, permissions, and secure inter process communication
- Techniques to mitigate risks associated with common memory management errors
- An encrypted file system that can be enabled to protect data on lost or stolen devices
- User-granted permissions to restrict access to system features and user data
- Application-defined permissions to control application data on a per-app basis.

- Android 6.0 and higher versions ask from user to allow or deny individual permissions for application dynamically when needed.

However, it is important for you to be familiar with the Android security best practices as given in https://developer.android.com/training/best-security.html. This web page describe best practices to be followed when storing data, using permissions, using networks, validating inputs, handling user data, using web view, using cryptography, using inter process communication, and dynamically loading code. Following these practices will reduce many security issues that may adversely affect the users.

Activity

Activity 16.1

Briefly describe different methods recommended as Android best practices to store data.

Security in a virtual machine

Unlike in many other Virtual Machine environments the Dalvik VM in Android does not provide a security boundary. The application sandbox is implemented at the OS level, so Dalvik can interoperate with native code in the same application without any security constraints.

As there is limited storage on mobile devices, it is common for developers to build modular applications and use dynamic class loading. When doing this, you must consider both the source where you retrieve your application logic and where you store it locally. Dynamic class loading from sources that are not verified should not be done as that code might be modified to include malicious behavior.

Security in native code

It is recommended to use the Android SDK for application development, rather than using native development kit. Applications built with native code are more complex, less portable, and more like to include common memory-corruption errors such as buffer overflows.

16.3 Information Leakage

It is widely known that many mobile applications share data with third parties without the knowledge of users. There are many reports and research papers about in-app advertising that leak potentially sensitive personal information on millions of mobile phone users. These data may include how much money users make, whether or not they have kids, and what their political affiliations are.

Following is a list of situations how information leakage in mobile apps take place.

- Mobile app developers choose to accept in-app advertisements inside their app
- Advertisement networks pay a fee to app developers in order to show advertisements and monitor user activity. User data could be device models, geolocations, etc. This information is provided to advertisers select where to place ads
- Advertisers instruct ad networks to show their ads based on topic targeting (such as “movies”), interest targeting (such as usage patterns and previous click throughs), and demographic targeting (such as estimated age range)
- The ad network displays ads to appropriate mobile app users and receives payment from advertisers for successful views or click throughs by the recipients
- In-app ads are displayed unencrypted as part of the app’s GUI. So that mobile app developers can access the targeted ad content delivered to its own app users and then reverse-engineer that data to construct a profile of their app customer

In one research [1], to test what is being leaked, researchers had created a custom-built Android app that they installed on more than 200 participants’ phones. From that they have been able to review the accuracy of personalized advertisements served to test subjects from the Google mobile ad network, AdMob, based on users’ personal interests and demographic profiles.

Many researchers [1] [2] have found that the root cause of the privacy leakage is the lack of isolation between the advertisements and mobile apps. It is reported that adopting HTTPS would not do anything to protect the advertisement traffic.

Activity

Activity 16.2

This activity is to be done in LMS for this course in the given discussion forum.

Briefly describe main findings of a research paper or a report on information leakage from mobile devices. Please give the reference and access date and time with URL, if it was accessed on-line.

16.4 Device management policies

Device management policies control features in mobile devices and computers. To use them, first you have to define the type of policy to support at the functional level. Policies may cover screen-lock password strength, expiration timeout, encryption, etc.

Any operating system would have its own device management policies. For mobile devices, these policies can be created by using templates that contain recommended or custom settings, and then deploying them to devices. Since Android 2.2 (API level 8), the Android platform offers system-level device management capabilities through the Device Administration APIs so that the application can be configured to ensure a strong screen-lock password before displaying restricted content to the user.

Device management policies implemented in different set-ups

‘Android for Work’ is a program for supporting enterprise use of Android. You can develop apps for Android for Work to take advantages of built in security and management features in Android. Enterprise mobility management (EMM) providers and enterprise application developers can accessed it via APIs.

Apps built on Android for Work include data security, app security and device security as explained below.

- **Data security**—Business data is separated in a work profile and protected device-wide on work-managed devices. So data leakage prevention policies can easily be applied.
- **Apps security**—Work apps are deployed through ‘Google Play for Work’ preventing installation of apps from unknown sources and apply app configurations.
- **Device security**— ‘Android for Work’ devices are protected with disk encryption, lockscreen, remote attestation services, and hardware-backed key store.

Using ‘Android for Work’, organizations can choose what devices, APIs, and framework they want to use to develop apps which may enable;

- Building apps to help employees be more productive in scenarios such as Bring Your Own Device (BYOD), Corporate Owned Personally Enabled (COPE) devices, and Corporate-Owned, Single-Use (COSU) devices.
- Connecting with leading enterprise mobility management (EMM) partners to help integrate Android in your business.

You may use Android for Work Developer Guide, https://developer.android.com/work/guide.html to learn to create apps that best utilize features in Android.

Moreover, ‘Android for Work’ offers a partner program for developers through the Android for Work DevHub (https://enterprise.google.com/android/developers/applyDevHub/). It provides exclusive access to beta features and developer events, along with access to a community of Android developers making enterprise apps.

Designing and developing applications that enforces security policies on devices

In this section, you will learn how to create a security-aware application that manages access to its content by enforcing device management policies. To do that we will discuss how to,

1. define and declare your policy
2. create a device administration receiver
3. activate the device administrator
4. implement the device policy controller

Define and declare your policy

First, you need to define the kinds of policy to support at the functional level such as screen-lock password strength, expiration timeout, encryption, etc.

Then, you must declare the selected policy set, which will be enforced by the application, in the `res/xml/device_admin.xml` file. The Android manifest should also reference the declared policy set.

Each declared policy corresponds to some number of related device policy methods in `DevicePolicyManager` (e.g. defining minimum password length and minimum number of uppercase characters). If an application attempts to invoke methods whose corresponding policy is not declared in the XML, this will result in a `SecurityException` at runtime.

Create a device administration receiver

You must create a Device Administration broadcast receiver, which gets notification of events related to the policies you have declared to support. An application can selectively override callback methods.

Activate the device administrator

Before enforcing any policies, the user needs to manually activate the application as a device administrator. It is a good practice to include the explanatory text to highlight to users why the application is requesting to be a device administrator. It can be done by specifying the EXTRA_ADD_EXPLANATION extra in the intent.

Implement the device policy controller

After the device administrator is activated successfully, the application then configures Device Policy Manager with the requested policy. However, new policies are being added to Android with each release. It is appropriate to perform version checks in your application if using new policies while supporting older versions of the platform.

More details and necessary code snippets can be found at https://developer.android.com/work/device-management-policy.html.

Activity

Activity 16.3

What is the importance of Network security configuration?

Unit summary

In this unit we discussed importance of security in Android apps, the security provided by the operating system, problem of information leakage and how to develop secure apps by adhering to the security policies.

References

1. The Price of Free: Privacy Leakage in Personalized Mobile In-App Ads Wei Meng, Ren Ding, Simon P. Chung, Steven Han, and Wenke Lee

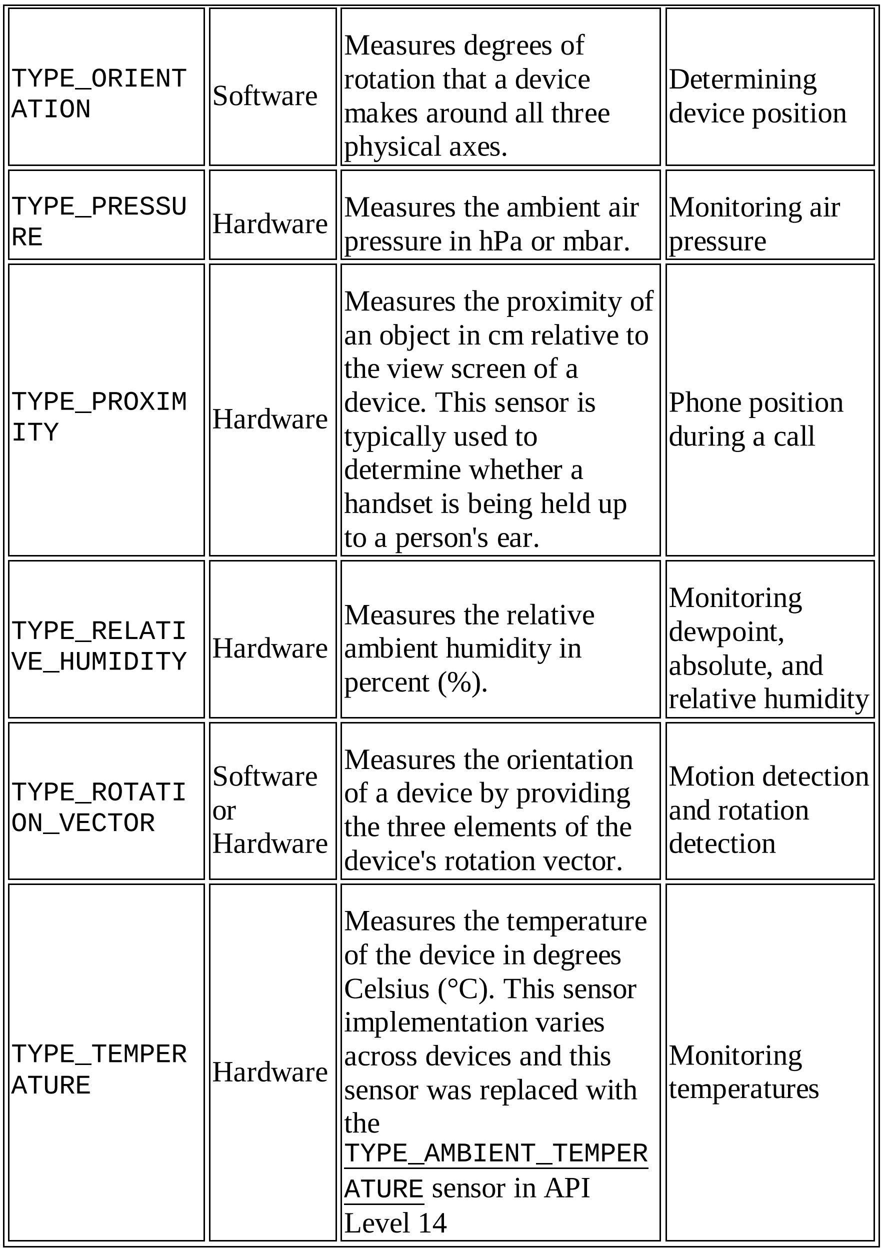
2. Privacy Capsules: Preventing information leaks by mobile apps Raul Herbster, Scott DellaTorre, Peter Druschel, Bobby Bhattacharjee

Appendix -A

Sensor types

Table 12.1 Sensor types supported by the Android platform.

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.)
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C).	Monitoring air temperatures
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes.	Motion detection (shake, tilt, etc.)
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes.	Rotation detection (spin, turn, etc.)
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes, excluding the force of gravity.	Monitoring acceleration along a single axis
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes in μT.	Creating a compass



TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes.	Determining device position
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14	Monitoring temperatures

Table 12.2 Motion sensors that are supported on the Android platform.

Sensor	Sensor event data	Description	Units of measure
TYPE_ACCELEROMETER	SensorEvent.values[0]	Acceleration force along the x axis (including gravity).	m/s^2
	SensorEvent.values[1]	Acceleration force along the y axis (including gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (including gravity).	
TYPE_GRAVITY	SensorEvent.values[0]	Force of gravity along the x axis.	m/s^2
	SensorEvent.values[1]	Force of gravity along the y axis.	
	SensorEvent.values[2]	Force of gravity along the z axis.	
TYPE_GYROSCOPE	SensorEvent.values[0]	Rate of rotation around the x axis.	rad/s
	SensorEvent.values[1]	Rate of rotation around the y axis.	
	SensorEvent.values[2]	Rate of rotation around the z axis.	
TYPE_GYROSCOPE_UNCALIBRATED	SensorEvent.values[0]	Rate of rotation (without drift compensation) around the x axis.	rad/s
	SensorEvent.values[1]	Rate of rotation (without drift compensation) around the y axis.	
	SensorEvent.values[2]	Rate of rotation (without drift compensation) around the z axis.	
	SensorEvent.values[3]	Estimated drift around the x axis.	

	SensorEvent.values[4]	Estimated drift around the y axis.	
	SensorEvent.values[5]	Estimated drift around the z axis.	
TYPE_LINEAR_ACCELERATION	SensorEvent.values[0]	Acceleration force along the x axis (excluding gravity).	m/s^2
	SensorEvent.values[1]	Acceleration force along the y axis (excluding gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (excluding gravity).	
TYPE_ROTATION_VECTOR	SensorEvent.values[0]	Rotation vector component along the x axis (x * sin(θ/2)).	Unitless
	SensorEvent.values[1]	Rotation vector component along the y axis (y * sin(θ/2)).	
	SensorEvent.values[2]	Rotation vector component along the z axis (z * sin(θ/2)).	
	SensorEvent.values[3]	Scalar component of the rotation vector ((cos(θ/2)).[1]	
TYPE_SIGNIFICANT_MOTION	N/A	N/A	N/A
TYPE_STEP_COUNTER	SensorEvent.values[0]	Number of steps taken by the user since the last reboot while the sensor was activated.	Steps
TYPE_STEP_DETECTOR	N/A	N/A	N/A

[1] The scalar component is an optional value.

Table 12.3 Environment sensors that are supported on the Android platform.

Sensor	Sensor event data	Units of measure	Data description
TYPE_AMBIENT_TEMPERATURE	event.values[0]	°C	Ambient air temperature.
TYPE_LIGHT	event.values[0]	lx	Illuminance.
TYPE_PRESSURE	event.values[0]	hPa or mbar	Ambient air pressure.
TYPE_RELATIVE_HUMIDITY	event.values[0]	%	Ambient relative humidity.
TYPE_TEMPERATURE	event.values[0]	°C	Device temperature.[1]

[1] Implementations vary from device to device. This sensor was deprecated in Android 4.0 (API Level 14).

Table 12.4. Position sensors that are supported on the Android platform.

Sensor	Sensor event data	Description	Units of measure
TYPE_GAME_ROTATION_VECTOR	SensorEvent.values[0]	Rotation vector component along the x axis (x * sin(θ/2)).	Unitless
	SensorEvent.values[1]	Rotation vector component along the y axis (y * sin(θ/2)).	
	SensorEvent.values[2]	Rotation vector component along the z axis (z * sin(θ/2)).	
TYPE_GEOMAGNETIC_ROTATION_VECTOR	SensorEvent.values[0]	Rotation vector component along the x axis (x * sin(θ/2)).	Unitless
	SensorEvent.values[1]	Rotation vector component along the y axis (y * sin(θ/2)).	

	SensorEvent.values[2]	Rotation vector component along the z axis (z * sin(θ/2)).	
TYPE_MAGNETIC_FIELD	SensorEvent.values[0]	Geomagnetic field strength along the x axis.	μT
	SensorEvent.values[1]	Geomagnetic field strength along the y axis.	
	SensorEvent.values[2]	Geomagnetic field strength along the z axis.	
TYPE_MAGNETIC_FIELD_UNCALIBRATED	SensorEvent.values[0]	Geomagnetic field strength (without hard iron calibration) along the x axis.	μT
	SensorEvent.values[1]	Geomagnetic field strength (without hard iron calibration) along the y axis.	
	SensorEvent.values[2]	Geomagnetic field strength (without hard iron calibration) along the z axis.	
	SensorEvent.values[3]	Iron bias estimation along the x axis.	
	SensorEvent.values[4]	Iron bias estimation along the y axis.	
	SensorEvent.values[5]	Iron bias estimation along the z axis.	
TYPE_ORIENTATION1	SensorEvent.values[0]	Azimuth (angle around the z-axis).	Degrees
	SensorEvent.values[1]	Pitch (angle around the x-axis).	
	SensorEvent.values[2]	Roll (angle around the y-axis).	
TYPE_PROXIMITY	SensorEvent.values[0]	Distance from object.[2]	cm

[1] This sensor was deprecated in Android 2.2 (API level 8), and this sensor type was deprecated in Android 4.4W (API level 20). The sensor framework provides alternate methods for acquiring device orientation, which are discussed in Computing the Device's Orientation.

[2] Some proximity sensors provide only binary values representing near and far.

Appendix - B

Answers to Activities

This section provides the answer guide for the given activities in this book.

UNIT 01

Activity 1.1

1. You should be able to access this information in the same way regardless of the device's version.
2. Open the "App drawer" – Press the button at the bottom of your phone, in the center to view the entire list of apps installed in your device.
3. Scroll through the list and find "Settings" icon and tap it.
4. Scroll through the "Setting" and find "About phone" or "About tablet" option and tap it.
5. Look for "Android version", "Kernel version" and other relevant fields.

Activity 1.2

Across: 1. Cupcake 2. Gingerbread 3. Donut 4. Ice Cream Sandwich 5. Jelly Bean 6. Lollypop 7. Froyo

Down: 1. Kitkat 2. Éclair 3. Honeycomb 4. Marshmallow

Activity 1.3

Android is a powerful Operating System which supports many applications in Smart Phones. It offers many options not found in comparable mobile operating systems. For instance, there are some features such as Near Field Communication (NFC), storage and battery swap and media support. Here you have to identify such features to get an idea about how strong Android operating is.

UNIT 02

Activity 2.1

Linux Kernel is the foundation component of Android platform. It is there to handle the hardware; means it helps the software part of the Android System to interact with the hardware. Because all hardware drivers (display driver, keypad driver, camera driver, Wi-Fi driver, Bluetooth driver, etc.) are inbuilt in the kernel, the android runtime does not need to worry about the hardware handling. It is the lowest layer of Android architecture and it serves as the abstraction layer to other layers.

Activity 2.2:

Dalvik Virtual Machine (DVM) is a register based virtual machine that is used by Android system to run the Dalvik executable code (.dex file) which is a compiled code of Android. It used in Android System similarly as JVM works for Java to execute the byte code (.classfile). DVM doesn't work with .class files. One thing that you must know is that implicitly .dex file is generated by the .class file after highly optimizing the .class file for low memory and least processing power. It gives the power to a device to become an Android device.

Android Runtime (ART) is the successor of Dalvik Virtual Machine (DVM). It has some advantage over DVM including ahead-of-time compilation (AOT), improved garbage collection and other development and debugging enhancements.

Activity 2.3:

	Advantages	**Disadvantages**
Native	• Silky smooth performance • Best user experience • App icon available on the device • Can receive push notifications • Runs inside the operating system • Can use the platform APIs	• Developers need to know each of the platforms languages • Source code only works on the targeted platform • Slower to market due to multiple source codes

Web	• Cross platform • Single code base • Fast to production • Lower development cost	• Sluggish performance • Require loading • Network connection required • Not available in the app stores • Extremely limited API access Lives in the browser
Hybrid	• Single source code • Access to all platforms • Less time to deployment • Available in the app store • Has application icon on the device	• Dependent on such as phone gap • Middleware may be slow to update • More bug prone • Some bug fixes need middleware updates • Some bug fixes are outside of your control • Slower performance • More issues from device fragmentation

Feature	Native	HTMAL5	Hybrid
Graphics	Native APIs	HTML, Canvas, SVG	HTML, Canvas, SVG
App performance	Fast	Moderate	Moderate
Distribution	App Store/Market	Web	App Store/Market
Native look and feel	Native	Emulated	Emulated
Camera	Yes	No	Yes
Push Notifications	Yes	No	Yes
File upload	Yes	Yes	Yes
Contacts, calendar	Yes	No	Yes
Connectivity	Online and offline	Mostly online	Online and offline
Development skills needed	XML, Java	HTML5, CSS, Javascript	HTML5, CSS, Javascript
Geolocation	Yes	Yes	Yes

UNIT 03

Activity 3.1:

A fragment is essentially a modular section of an activity. Fragment has its own lifecycle and input events, and which can be added or removed based on its design.

(Source: https://developer.android.com/guide/components/fragments.html)

Activity 3.2:

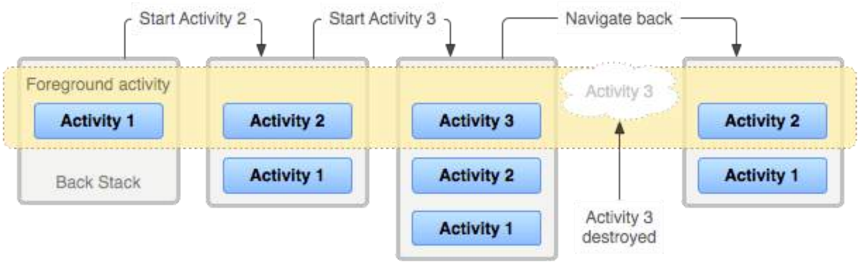


Figure 3.5: A representation of how each new activity in a task adds an item to the back stack. (Source: https://developer.android.com/guide/components/tasks-and-back-stack.html)

Activities in the stack are never rearranged, only pushed and popped from the stack—pushed onto the stack when started by the current activity and popped off when the user leaves it using the *Back* button. When the user presses the *Back* button, the current activity is destroyed and the previous activity resumes.

Activity 3.3:

1. A) True
2. B) False
3. A) True
4. A) True
5. A) True

UNIT 04

Activity 4.1:

You can explore cross platforms such as PhoneGap, Xamarin, Rhomobile, Appcelerator Titanium, MoSync, Alpha Anywhere, Corona, Qt, and 5App ect…

The most attention is given for PhoneGap, Appcelerator and Xamarin

PhoneGap has several features and uses of it. It is free in the open-source framework. It can be used to build native apps for multiple platforms. It is built using HTML, CSS and JavaScript to cater larger communities. PhoneGap API allows access to device features such as Accelerometer, Camera, Microphone, File system and few more.

Activity 4.2:

You must install one Android platform and platform tools prior to Android development.

UNIT 05

Activity 5.1

Identify the element that is not part of the basic application component of an Android application.

○ Activities
○ Services
○ Content Providers
○ Screencast Receivers
○ Broadcast Receivers

Answer:

○ Screencast Receivers

Activity 5.2

State the intent types that is available in Android.

Answer: Explicit Intent and Implicit Intent

Activity 5.3

Explain the role of AndroidManifest.xml file in an Android application.

Answer: Every application must have an `AndroidManifest.xml` file (with precisely that name) in its root directory. The manifest file provides essential information about your app to the Android system, which the system must have before it can run any of the app's code.

UNIT 06

Activity 6.1

1. Create a simple "Hello World Program"

Note: If you find it difficult to follow the instructions below, please refer to the video on ''Android development" for step by step instructions.

In the file **app** > **java** > **com.example.myfirstapp** > **MainActivity.java**, add the sendMessage() method stub as shown below:

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /** Called when the user taps the Send button */
    public void sendMessage(View view) {
        // Do something in response to button
    }
}
```

Now return to the **activity_main.xml** file to call this method from the button:

1. Click to select the button in the Layout Editor.

2. In the **Properties** window, locate the **onClick** property and select **sendMessage [MainActivity]** from the drop-down list.

3. In `MainActivity.java`, add the `EXTRA_MESSAGE` constant and the `sendMessage()` code, as shown here:

```
public class MainActivity extends
AppCompatActivity {
    public static final String EXTRA_MESSAGE =
"com.example.myfirstapp.MESSAGE";
    @Override
    protected void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /** Called when the user taps the Send
button */
    public void sendMessage(View view) {
        Intent intent = new Intent(this,
DisplayMessageActivity.class);
        EditText editText = (EditText)
findViewById(R.id.editText);
        String message =
editText.getText().toString();
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivity(intent);
    }
}
```

Display the message

1. In `DisplayMessageActivity.java`, add the following code to the `onCreate()` method:

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_display_messag
e);

    // Get the Intent that started this activity
and extract the string
    Intent intent = getIntent();
    String message =
intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

    // Capture the layout's TextView and set the
string as its text
    TextView textView = (TextView)
findViewById(R.id.textView);
    textView.setText(message);
}
```

2. Press Alt + Enter (or Option + Return on Mac) to import missing classes. Your imports should end up as the following:

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.TextView;
```

Add up navigation

```
<activity android:name=".DisplayMessageActivity"
          android:parentActivityName=".MainActivity" >
    <!-- The meta-data tag is required if you support
API level 15 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

Run the app

Now run the app again by clicking **Apply Changes** ⚡ in the toolbar. When it opens, type a message ”Hello World” in the text field, and tap **Send** to see the message appear in the second activity.

Now you have created your first “Hello World” Program.

Activity 6.2

Build a simple app and run the application on

a. Emulator

b. Actual Device

Answer: Please check the following URL for the step by step tutorial on how you can run an application in the Emulator and the Actual Device:

UNIT 07

Activity 7.1

Android devices come in many shapes and sizes all around the world. With a wide range of device types, to reach a huge audience, your app needs to adapt to various device configurations. Some of the important variations that you should consider include different languages, screen sizes, and versions of the Android platform.

Activity 7.2
Latest versions of Android often provide great APIs for your application. You should continue to support older versions of Android until more devices get updated. For this activity, you have to find version of the Android, code name, API level and the distribution rate for different platform versions. For ex: 7.0, Nougat, API level 24 and Distribution rate 8.9%

Activity 7.3
To calculate resolution of your mobile screen (Pixel density), you have to know vertical and horizontal pixel counts and your diagonal screen size then apply the below formula.

Samsung Galaxy S4
W – 1080 pixels
H – 1920 pixels
Dp – 2202.9 pixels
Di – 5 inches

Resolution (Pixel density) of the Samsung Galaxy S4 is **441**

UNIT 09

Activity 9.1

Differentiate the use of local test from instrumented test when performing a unit test of an Android application.

Answer : Local tests are the unit tests run on the local machines only. These tests are compiled to run locally on JVM to minimize the execution time. Instrumented tests are the unit tests run on Android device or emulator.

Activity 9.2

Create an Android application "`MyApp`" with a class "`ConversionUtil`" to perform the given two functionalities.

- To convert centimeters into inches [write a method `ConvertCmtoInch`()]
- To convert inches into centimeters [write a method `ConvertInchtoCm`()]

Then write local unit tests to check whether the written functionalities provide the expected output. Use the values given as inputs and expected output to test the method.

Functionality to test	Input	Output
Convert centimeters into inches	10 centimeters	3.93701 inches
Convert inches into centimeters	10 inches	25.4 centimeters

Answer :
First you need to create a new Android Project by providing MyApp as the Application Name.

Then you need to check whether the testing environment is set in your application. If not you need to go to Module Settings in your application and import JUnit.

Then you need to write the class ConversionUtil with two methods ConvertCmtoInch() and ConvertInchtoCm() with necessary logic in them. A sample code snippet for method ConvertCmtoInch() is given below.

```
public double ConvertCmtoInch(double cm){
    return  cm * 0.393701;
{
```

Similarly write the method to convert inches into centimeters.

Then you need to write the unit tests to test the two functionalities. You can refer the video given under unit 9, for the steps to be followed.

Activity 9.3

Go to the settings of your device and Scroll down to the bottom of the about screen and find the Build number. Tap the Build number field seven times to enable Developer Options.

UNIT 10

Activity 10.1

Refer the given link of valid or invalid state transitions from the MediaPlayer
https://developer.android.com/reference/android/media/MediaPlayer.html #Valid_and_Invalid_States

Activity 10.2

Google Music Play

- Play stored music files
- Streaming music files

- Download music files
- Upload music files
- Support MP3, AAC, WMA, FLAC, Ogg, or ALAC file formats
- Support offline playback

Ustream

- Play stored video files
- Streaming video files
- Download video files
  - Support MOV, MP4, AVI, OGM,MPEG2 and WMV file formats with video codecs: H264, H263, MPEG4 (and variants), VP6, VP8, THEORA, WMV and audio codecs: MP3, AAC-LC, Nellymoser, PCM (16 bit max), Speex, Vorbis, WMV
- Support offline playback

Netflix

- Play stored video files
- Streaming video files
- Download video files
- Support AVC, MP4, file formats with H.264, WMA, WMV3, VC-1 Advanced Profiling encoding
- Support offline playback

XfinityTV

- Play stored video files
- Streaming video files
- Download video files
- Support DVR functions
- Support offline playback

Activity 10.3

```
Button boton = (Button) findViewById(R.id.boton);
boton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
 MediaPlayer mp = MediaPlayer.create(TestSonido.this,
R.raw.slayer);
 mp.start();
}
});
```

UNIT 14

Activity 14.1
Depending on whether the publishing is intended for proliferation among certain types of devices, device manufacturer based AppStores are more viable for both AppA and AppB. If the proliferation is expected as device agnostic, then for AppA, independent AppStores are more viable platforms. Since there is no specifications to select an appropriate mobile operator or a supporting operating system, such specific AppStores are not desirable.

Activity 14.2
Equalizer is a free app. It does not require a payment to install.

Activity 14.3
Free apps do not necessarily ensure a large user base. However, a user is more likely to install a free app. If a subscription is introduced the users expect value-for-money . Therefore, would require additional features etc. Depending on the visions of whether to popularize, sustain the market with new and emerging competitors, retain a significantly larger user base or to increase the revenue can be considered as important aspects to change from one revenue model to another.

UNIT 15

Activity 15.1
Q: What are the sub-tools provided by the Android Monitor to analyse the performance of an app?
A: Android Monitor provides various sub-tools to profile the performance of an app. They are the performance monitors such as logcat, Memory, CPU, GPU, and Network Monitors. *Dumpsys* is an Android tool that runs on the device and dumps information about the status of system services.

Activity 15.2
Answer depends on the developed app during the course. Many techniques are given under 'How Android Manages Memory'

Activity 15.3
Answer depends on the developed app during the course.

UNIT 16

Activity 16.1
Q:Briefly describe different methods recommended as Android best practices to store data.
Answer: Shared Preferences

Store Private Primitive data in key value pairs
Internal Storage
Store private data on the device memory
External Storage
Store public data on the shared external storage
SQLite Databases
Store structured data in a private database
Network Connection
Store data on the web with your own network server

Activity 16.2
Q:Briefly describe main findings of a research paper or a report on information leakage from mobile devices. Please give the reference and access date and time with URL, if it was accessed on-line.

Two example answers are given below.

- Mobile app developers choose to accept in-app advertisements inside their app
- Advertisement networks pay a fee to app developers in order to show advertisements and monitor user activity. User data could be device models, geolocations, etc. This information is provided to advertisers select where to place ads

Activity 16.3

Importance of the Network Security Configuration feature is that, it lets apps customize their network security settings in a safe, declarative configuration file without modifying app code. These security settings can be configured for specific applications in specific domains.