UNIT 1

Introduction

Mobile Application Development

Mobile application development is a term used to denote the act or process by which application software is developed for mobile devices, such as personal digital assistants, enterprise digital assistants or mobile phones.

Mobility

• Transforming user experience (UE) from confines of a desk to the convenience of anytime-anywhere.

• Spontaneity, ubiquity and indispensability.

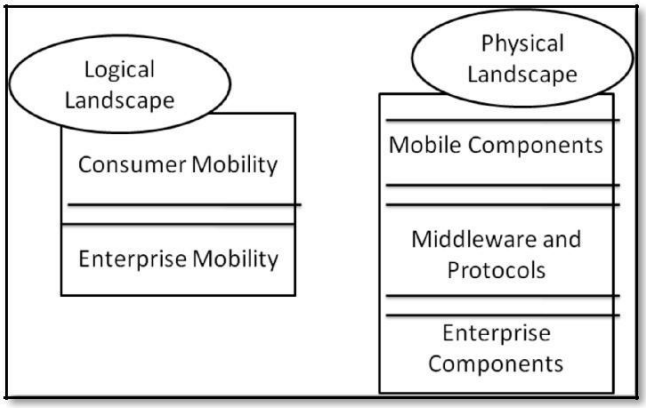
UNIT 1

Mobility Panorama:

It is classified into

(1) Logical Landscape

(2) Physical Landscape



Logical Landscape:

It describes the rationale behind mobility for different stakeholders.

Consumer Mobility:

- Focused toward the end user.
- Comprise mobility solutions such as social networking, games, shopping, bidding & utilities.

Enterprise Mobility:

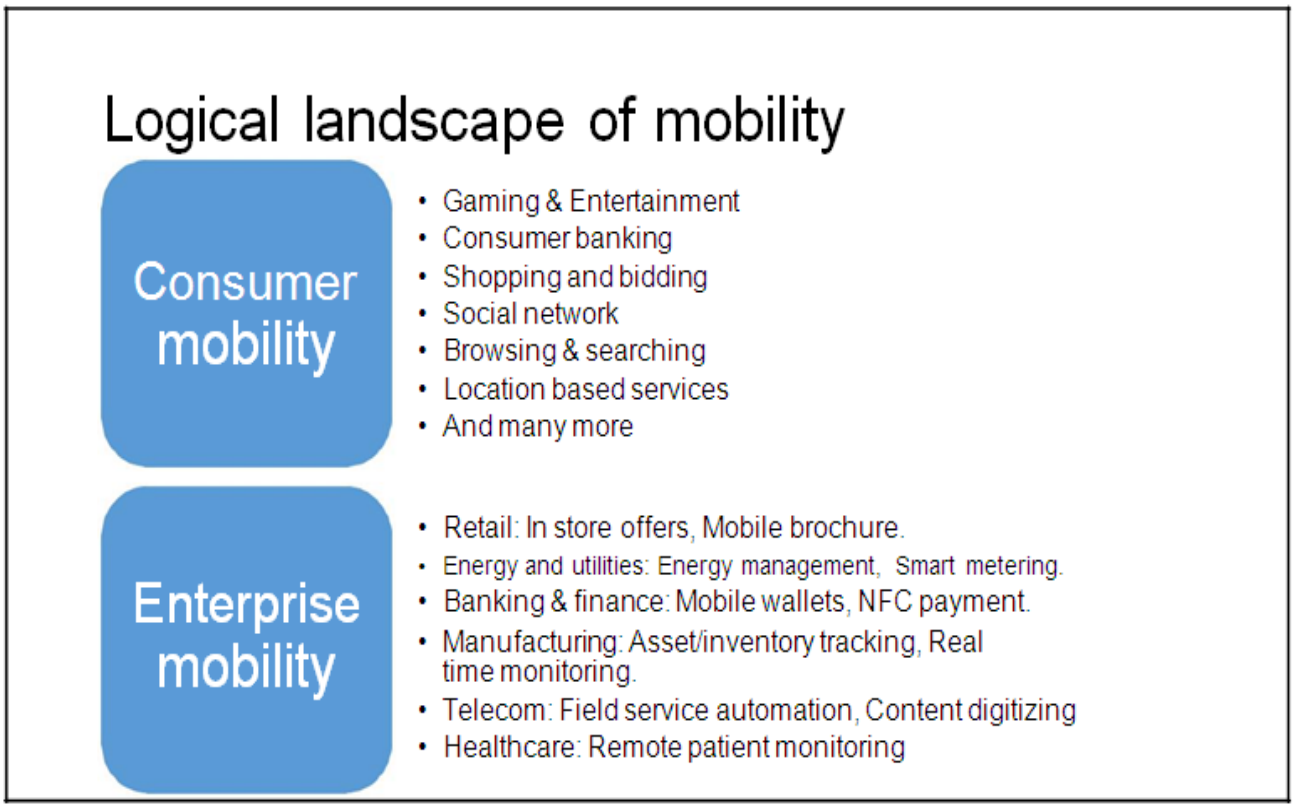
- Focused toward various stakeholders of an organization such as vendors, partners, suppliers, workforce and their end consumers.

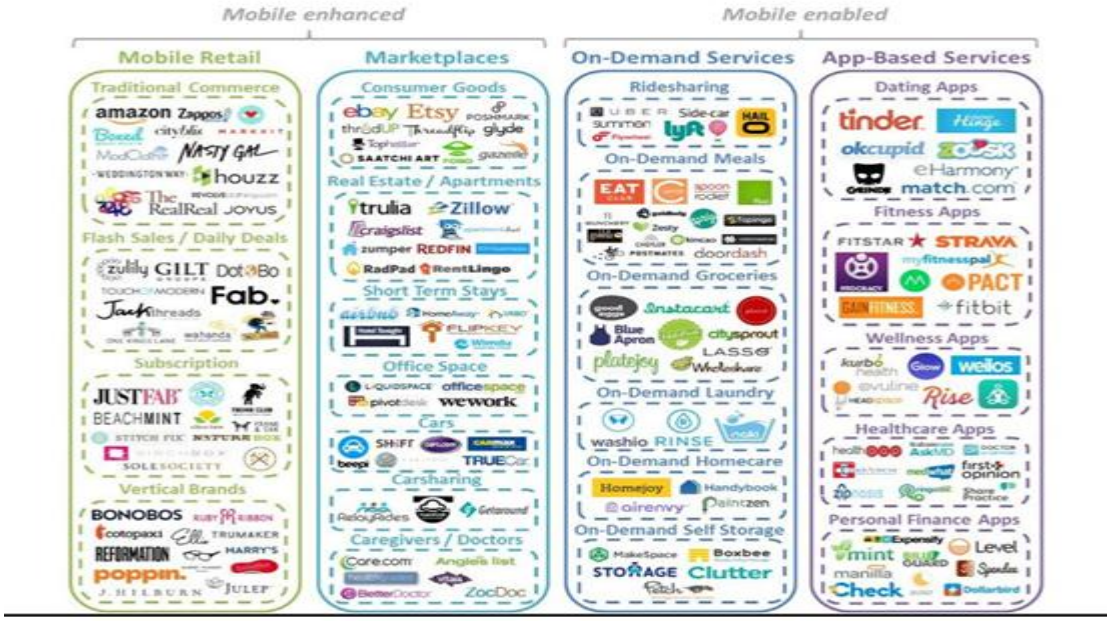
- Mobility is enabling enterprises to increase productivity of their workforce.

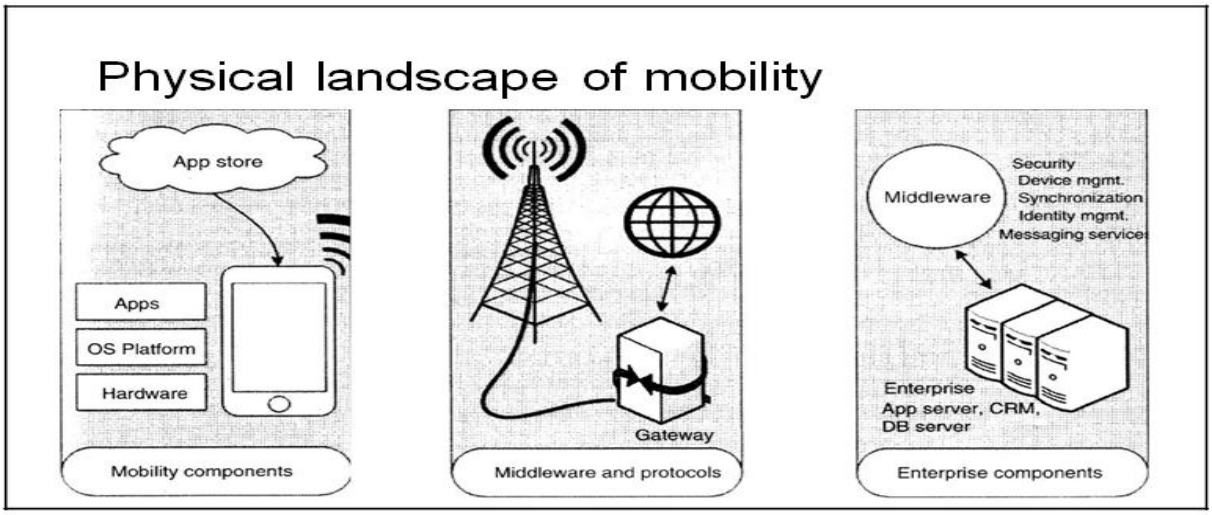
Physical Landscape:

It portrays the infrastructure that enables mobility.

UNIT 1


Logical landscape of mobility
Consumer mobility
Gaming & Entertainment
Consumer banking
Shopping and bidding
Social network
Browsing & searching
Location based services
And many more
Enterprise mobility
Retail: In store offers, Mobile brochure.
Energy and utilities: Energy management, Smart metering.
Banking & finance: Mobile wallets, NFC payment.
Manufacturing: Asset/inventory tracking, Real time monitoring.
Telecom: Field service automation, Content digitizing
Healthcare: Remote patient monitoring


Mobile enhanced
Mobile enabled
Mobile Retail
Marketplaces
On-Demand Services
App-Based Services
Traditional Commerce
Flash Sales / Daily Deals
Subscription
Vertical Brands
Consumer Goods
Real Estate / Apartments
Short Term Stays
Office Space
Cars
Carsharing
Caregivers / Doctors
Ridesharing
On-Demand Meals
On-Demand Groceries
On-Demand Laundry
On-Demand Homecare
On-Demand Self Storage
Dating Apps
Fitness Apps
Wellness Apps
Healthcare Apps
Personal Finance Apps


Physical landscape of mobility
App store
Apps
OS Platform
Hardware
Mobility components
Gateway
Middleware and protocols
Middleware
Security
Device mgmt.
Synchronization
Identity mgmt.
Messaging service
Enterprise
App server, CRM,
DB server
Enterprise components

Mobile Components:

- Key mobility components are mobile devices, mobile platforms, and mobile app stores.
- Mobile devices are the centre piece of mobility, and available in different shapes and sizes such as smart phones, tablets, phablets and smart watches.
- Mobile platforms such as Android and Applie iOS, are software stacks that power mobile devices.
- Mobile app stores are online market places of mobile apps. Ex.

  Google Play, App Store

Enterprise Components:

- Comprises hosts of servers, such as database servers and application servers that cater to enterprise portion of mobility solutions.
- Also comprise enterprise solutions that cater to the requirements of
  - data security

- data synchronization between mobile devices and enterprise servers and identity management

Middleware and Protocols:

- It acts as glue between mobility and enterprise components.

- Access mechanisms such as Wi-Fi, Bluetooth, Code Division Multiple Access (CDMA), General Packet Radio Service (GPRS), and GSM are some key components of this layer that allow mobile devices to communicate.

- Other key components are gateways such as WAP and SMS

  gateways that enable interaction between mobile devices and the Internet.

UNIT 1

Mobile Platforms

- Mobile OSs known as mobile platforms
- It is not just an OS but a software stack that typically comprises an OS, libraries and application development framework(s).
- The OS contributes to the core features of the platform such as
  - Memory management
  - Process management and
  - Various device drivers
- Libraries furnish the most required core functionality of the platform such as media libraries etc.
- The application development framework is the set of application programming interfaces (APIs) that in turn interact with the underlying libraries and are exposed to the developers for app development.
- Most popular mobile platforms are:

  Android
  Apple iOS
  Black Berry
  Windows phone

16CS306 and Composing Mobile Apps
UNIT 1

Mobile Platforms

Typical software stack of mobile platform

Operating System
- Memory management
- Process managements
- Various device driver

Libraries
- Media libraries
- Native data storage
- Rendering screen & drawing surfaces
- Graphics libraries

Application Development Framework
- Set of API
- Interact with underlying libraries
- Expose to developers for app development

UNIT 1

Mobile Apps Development

APP Development Approaches:

Three broad approaches:

1. Native
2. Web
3. Hybrid

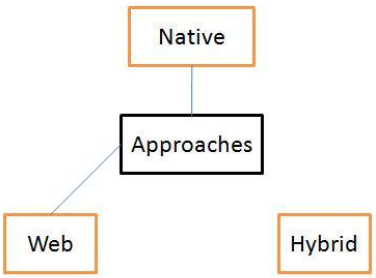


Figure: Mobile App Development Approaches

(1) Native Approach:

- native app,
- developed for native platform using platform specific APIs
- distributed through online app stores
- preferred when app requires a native look and feel

(2) Web Approach:

- mobile web app,

- developed using web technologies such as HTML5, CSS3, and JavaScript
- not installed on mobile device, gets rendered in an mobile browser, over the network
- preferred when app requires to cater to diverse devices using a single codebase
- no native L&F and no high-end device capabilities

☐ Hybrid Approach:

- mixed approach (features of native + web approaches)
- developed using mobile cross platforms
- hybrid platforms do not power device
- facilitate multiplatform development
- same codebase of a mobile app translated to fit into any of the supported native platforms

UNIT 1

App development approaches

- Rich user experience
- Platform specific
- Proven path for mobile apps

NATIVE

- App-like experience
- Leverages device capabilities
- Multiple platforms

HYBRID

- Fast development cycles
- Cross-platform
- Instant updates

WEB/HTML5

App development approaches

Overview of Android Platform

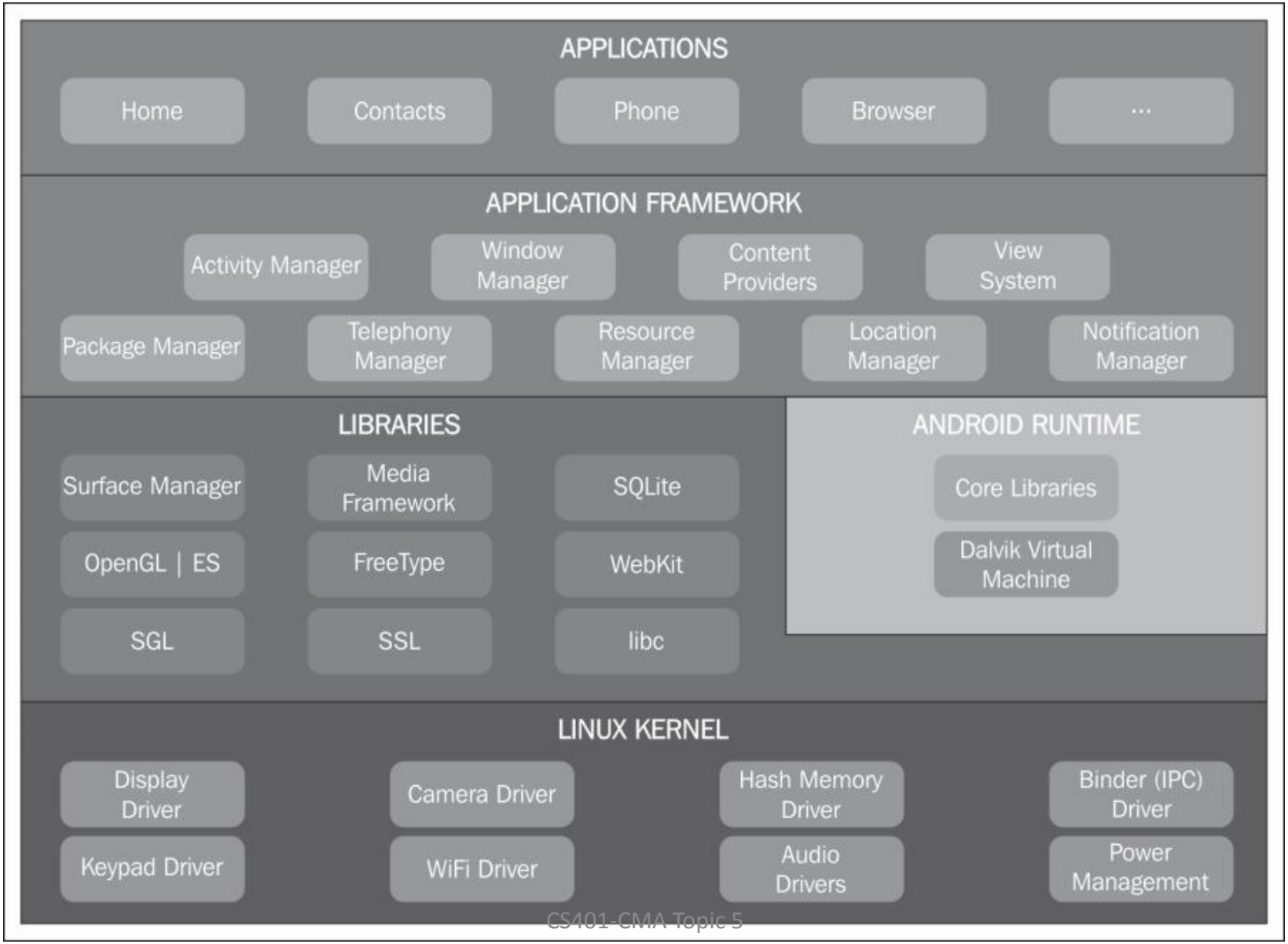
A SOFTWARE STACK FOR MOBILE DEVICES:

- OS KERNEL, SYSTEM LIBRARIES, APPLICATION FRAMEWORKS & KEY APPS

ANDROID SDK FOR CREATING APPS

- LIBRARIES & DEVELOPMENT TOOLS

Architecture



Linux Kernel – standard services

Security

Memory & process management

File & network I/O

Device drivers

Linux Kernel – Android-specific

Power management

Android shared memory

Low memory killer

Interprocess communication

And much more

Libraries

System C library
- Bionic libc

Surface Mgr.
- Display management

Media Framework
- Audio/ Video

Webkit
- browser engine

OpenGL
- Graphics engines

SQLite
- Relational database engine

Android Runtime

Two main components

- Core Java libraries
- Dalvik Virtual Machine

Core Java Libraries

basic java classes -- java.*, javax.*

app lifecycle -- android.*

Internet/Web services -- org. *

Unit testing -- junit.*

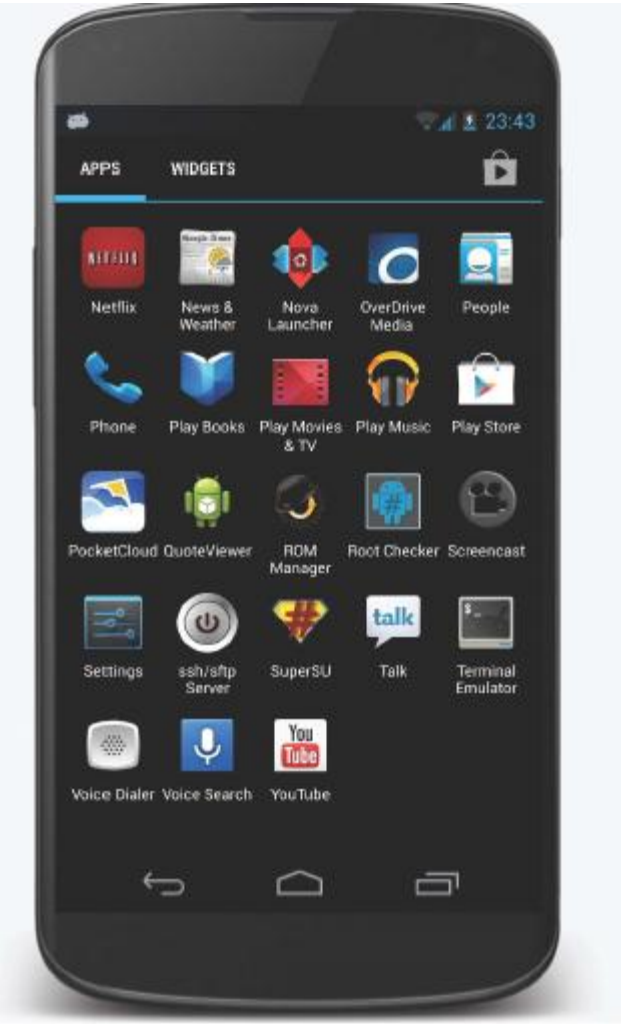
Dalvik Virtual Machine

Apps are executed by the dalvik virtual machine

Application Framework

Package Manager

Keeps track of app packages on device



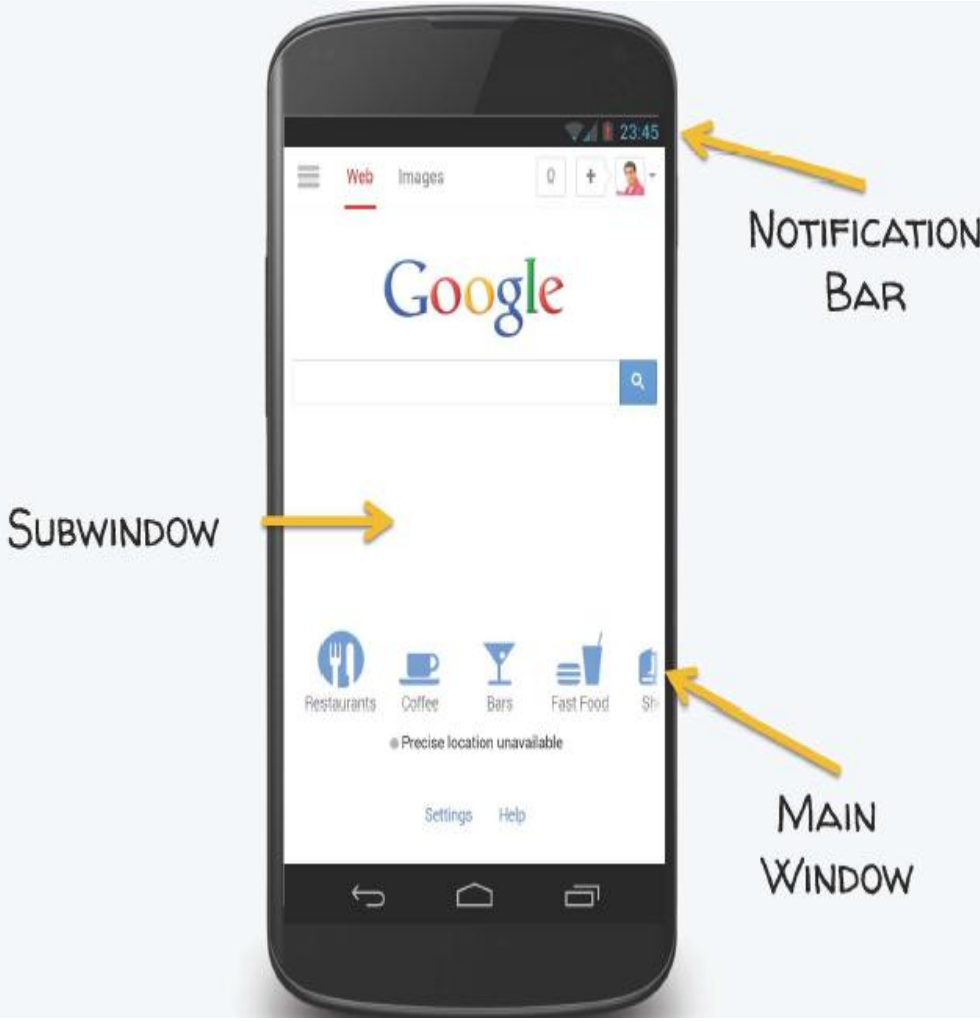
Window Manager

Manages the windows comprising an App

View System

Provides common user interface elements

- E.G., ICONS, TEXT ENTRY BOXES, BUTTONS AND MORE



Resource Manager

Manages non-compiled resources

E.g., strings, graphics, & layout files

Activity Manager

Manages app lifecycle and navigation stack

ContentProvider

Inter-application data sharing

LocationManager

Provides location & movement information

NotificationManager

Place notification icons in the status bar when important events occur

Applications

Standard apps include:

- Home – main screen
- Contacts – contacts database
- Phone – dial phone numbers
- Browser – view web pages
- Email reader –compose & read email messages

Setting-Up-Your-Android-Development-Environment

https://www.codeproject.com/Articles/797553/Setting-Up-Your-Android-Development-Environment

Mobile App Development flow

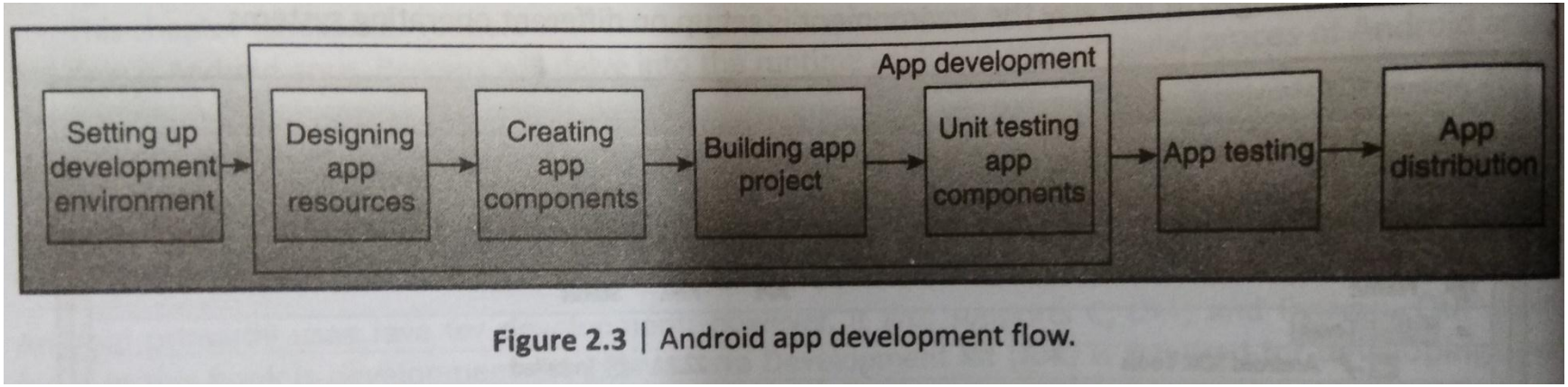


Figure 2.3 | Android app development flow.

UNIT 1

Setting up the Mobile app development

Difference between Emulator and Simulator

Emulator	Simulator	
What it mimics	Mobile device software Mobile device hardware Mobile operating system	Internal behavior of the device. It does not mimic hardware.
How to get it	It is generally provided by the device manufacturer.	It is generally provided by the device manufacturer or some other company.
Internal structure	It is written in machine-level assembly language.	It is written in high-level language.
Debugging	It is more suitable for debugging.	It is not suitable for debugging purpose.
Performance	Emulators are really slow. Emulating the actual hardware usually makes the software run slower than it would natively.	Faster than emulators.

Example	Google’s Android SDK	Apple’s iOS Simulator

Environment Set up

Start the Android application development on either of the following operating systems

- Microsoft Windows XP or later version.
- Mac OS X 10.5.8 or later version with Intel chip.
- Linux including GNU C Library 2.7 or later.

Second point is that all the required tools to develop Android applications are freely available and can be downloaded from the Web. Following is the list of software's you will need before you start your Android application programming.

- Java JDK5 or later version
- Android Studio

Here last two components are optional and if you are working on Windows machine then these components make easy while doing Java based application development.

UNIT 1

Set-up Java Development Kit (JDK)

You can download the latest version of Java JDK from Oracle's Java site − Java SE Downloads. You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains **java** and **javac**, typically java_install_dir/bin and java_install_dir respectively.

If you are running Windows and installed the JDK in C:\jdk1.8.0_102, you would have to put the following line in your C:\autoexec.bat file.

```
set PATH=C:\jdk1.8.0_102\bin;%PATH%
set JAVA_HOME=C:\jdk1.8.0_102
```

Alternatively, you could also right-click on *My Computer*, select *Properties*, then *Advanced*, then *Environment Variables*. Then, you would update the PATH value and press the OK button.

On Linux, if the SDK is installed in /usr/local/jdk1.8.0_102 and you use the C shell, you would put the following code into your **.cshrc** file.

```
setenv PATH /usr/local/jdk1.8.0_102/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.8.0_102
```

Alternatively, if you use Android studio, then it will know automatically where you have installed your Java.

Android IDEs

There are so many sophisticated Technologies are available to develop android applications, the familiar technologies, which are predominantly using tools as follows

- Android Studio
- Eclipse IDE(Deprecated)

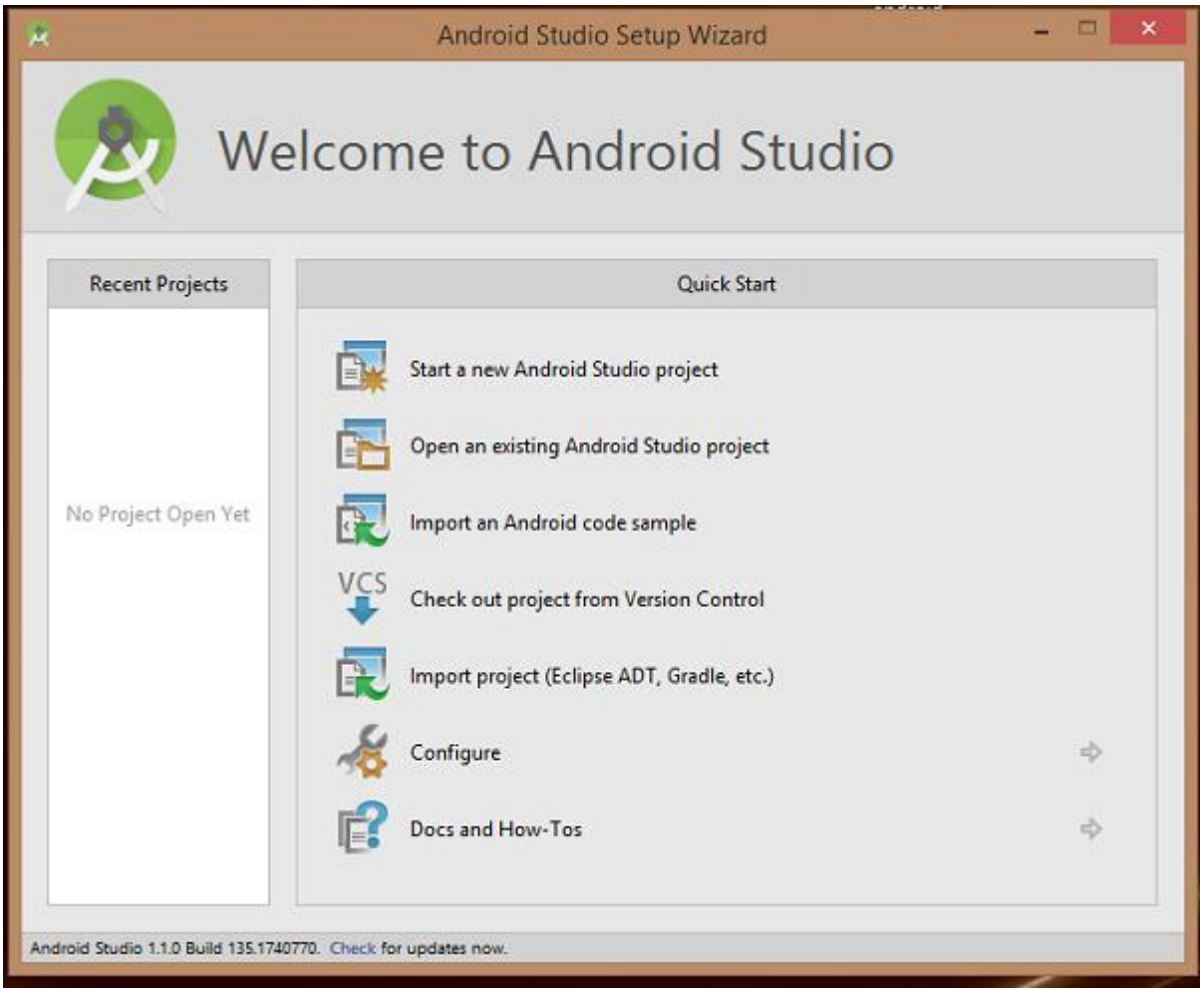
UNIT 1

Saying hello to android

simple Android Application which will print "Hello World!".

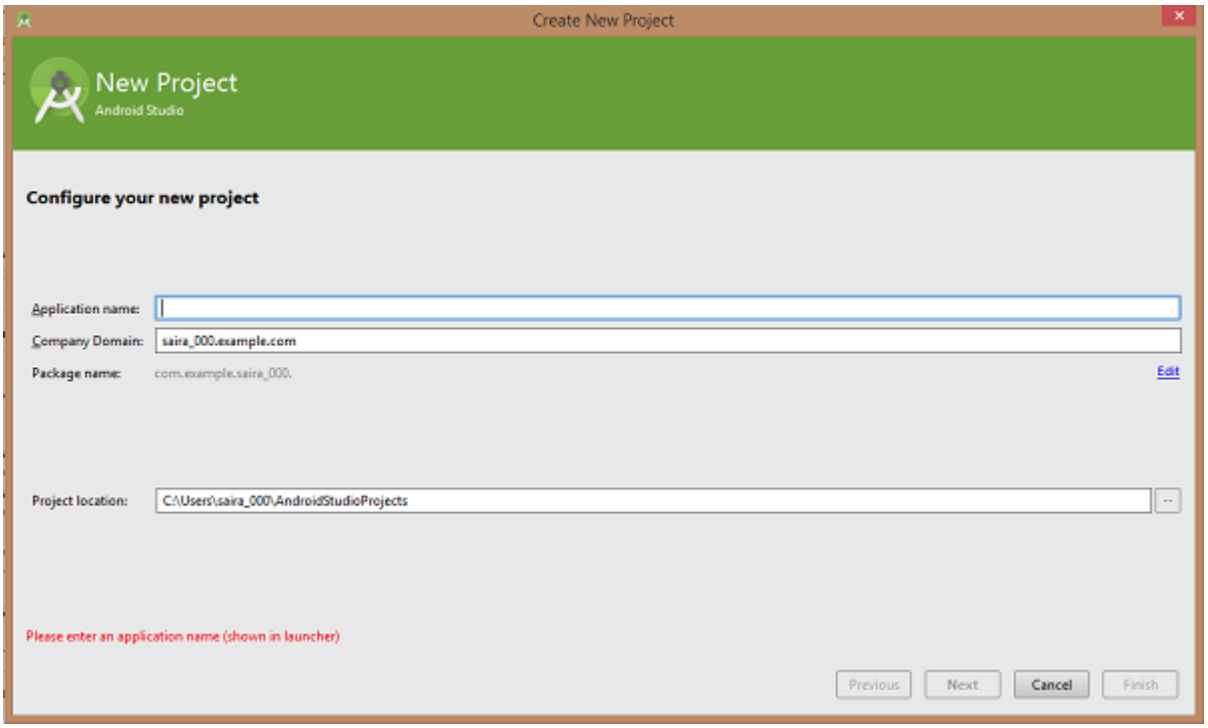
Create Android Application

The first step is to create a simple Android Application using Android studio. When you click on Android studio icon, it will show screen as shown below

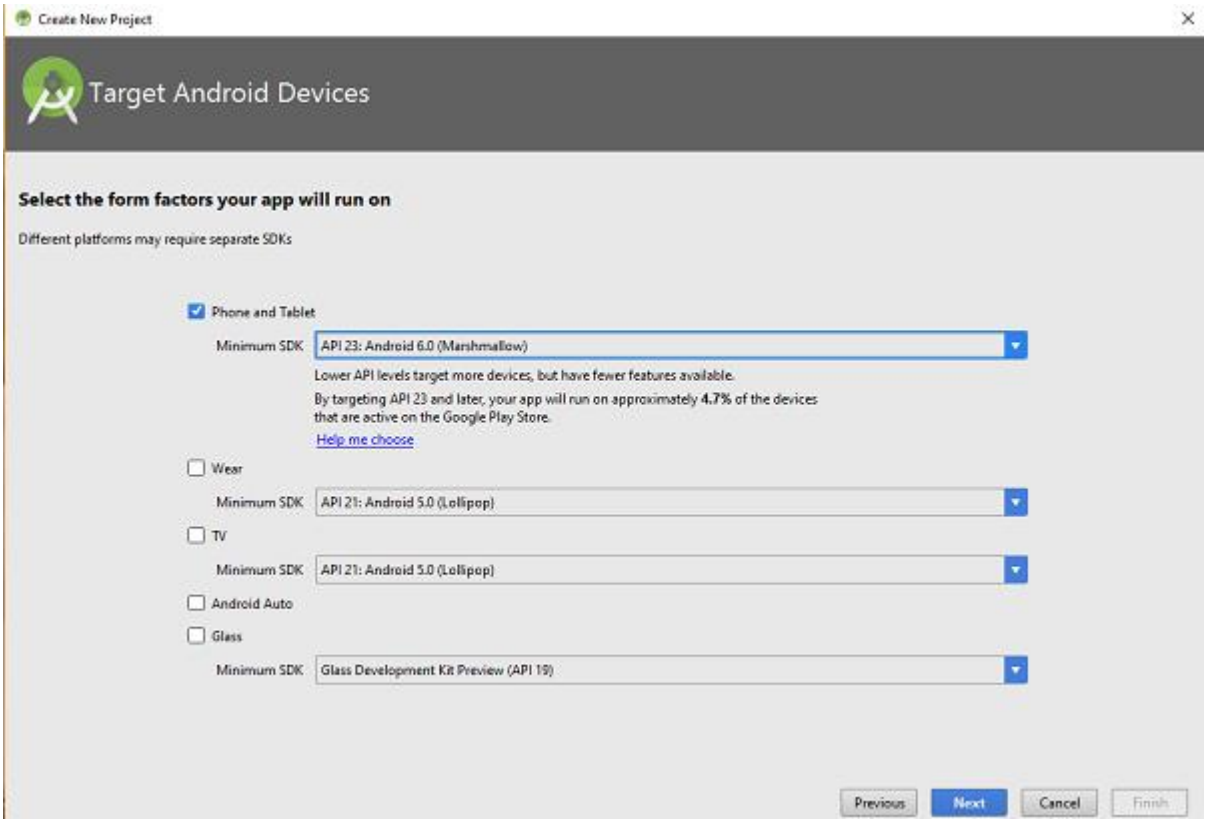


You can start your application development by calling start a new android studio project. in a new installation frame should ask Application name, package information and location of the project.–

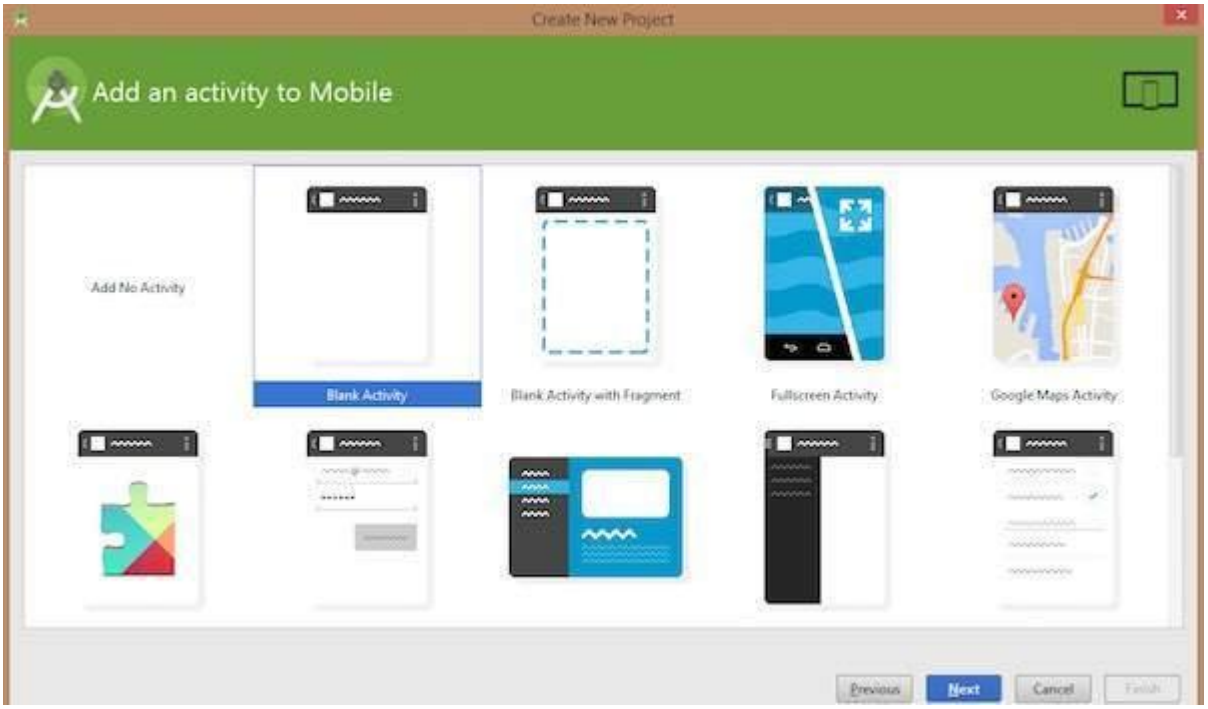
UNIT 1



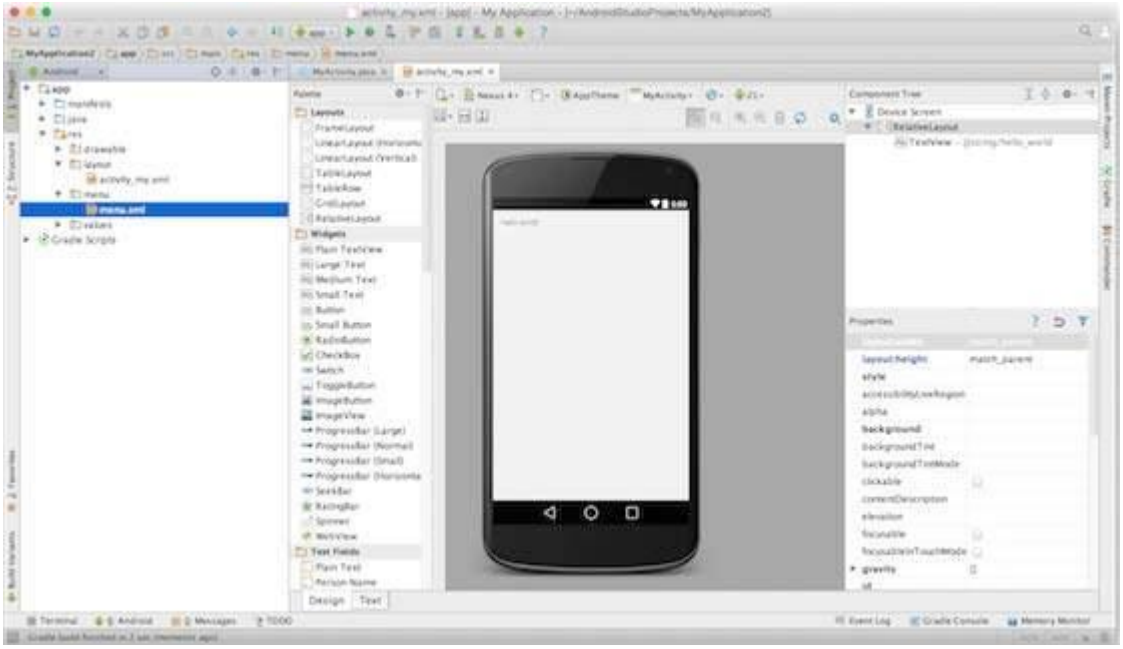
After entered application name, it going to be called select the form factors your application runs on, here need to specify Minimum SDK, in our tutorial, I have declared as API23: Android 6.0(Mashmallow) –



The next level of installation should contain selecting the activity to mobile, it specifies the default layout for Applications.



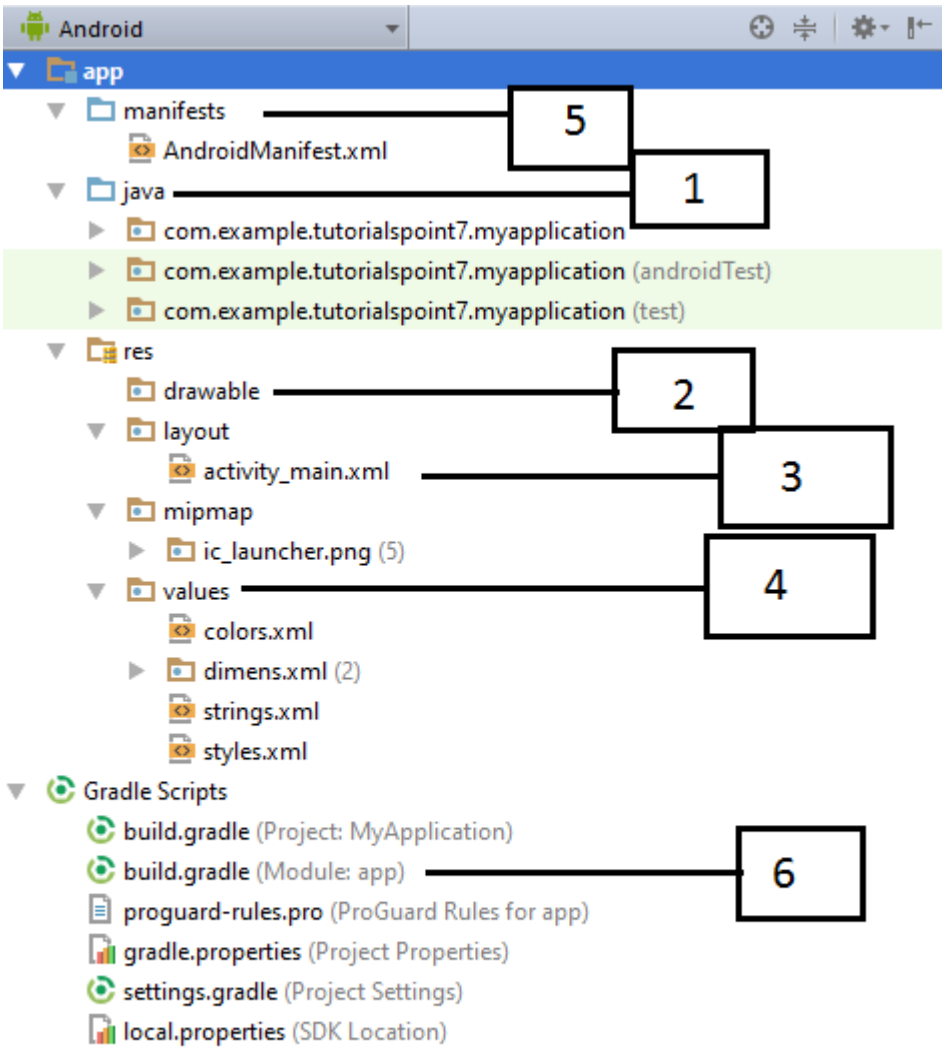
At the final stage it going to be open development tool to write the application code.



UNIT 1

Anatomy of Android Application

Before you run your app, you should be aware of a few directories and files in the Android project –



Running the Application

Try to run the **Hello World!** application just created. Then create **AVD** while doing environment set-up. To run the app from Android studio, open one of your project's activity files and click Run  icon from the tool bar. Android studio installs the app on AVD and starts it and if everything is fine with the set-up and application, it will display the Emulator window

UNIT 1

Projects Overview

1. Modules
2. Project Files
3. Project Structure Settings

A *project* in Android Studio contains everything that defines your workspace for an app, from source code and assets, to test code and build configurations. When you start a new project, Android Studio creates the necessary structure for all your files and makes them visible in the**Project** window on the left side of the IDE (click **View > Tool Windows > Project**). This page provides an overview of the key components inside your project.

Modules

A *module* is a collection of source files and build settings that allow you to divide your project into discrete units of functionality. Your project can have one or many modules and one module may use another module as a dependency. Each module can be independently built, tested, and debugged.

You can add a new module to your project by clicking **File > New > New Module**.

Android Studio offers a few distinct types of module:

Android app module

Provides a container for your app's source code, resource files, and app level settings such as the module-level build file and Android Manifest file. When you create a new project, the default module name is "app".

In the **Create New Module** window, Android Studio offers the following app modules:

- Phone & Tablet Module
- Android Wear Module
- Android TV Module
- Glass Module

They each provide essential files and some code templates that are appropriate for the corresponding app or device type.

Library module

UNIT 1

Provides a container for your reusable code, which you can use as a dependency in other app modules or import into other projects. Structurally, a library module is the same as an app module, but when built, it creates a code archive file instead of an APK, so it can't be installed on a device.

In the **Create New Module** window, Android Studio offers the following library modules:

- Android Library: This type of library can contain all file types supported in an Android project, including source code, resources, and manifest files. The build result is an Android Archive (AAR) file that you can add as a dependency for your Android app modules.
- Java Library: This type of library can contain only Java source files. The build result is an Java Archive (JAR) file that you can add as a dependency for your Andriod app modules or other Java projects.

Google Cloud module

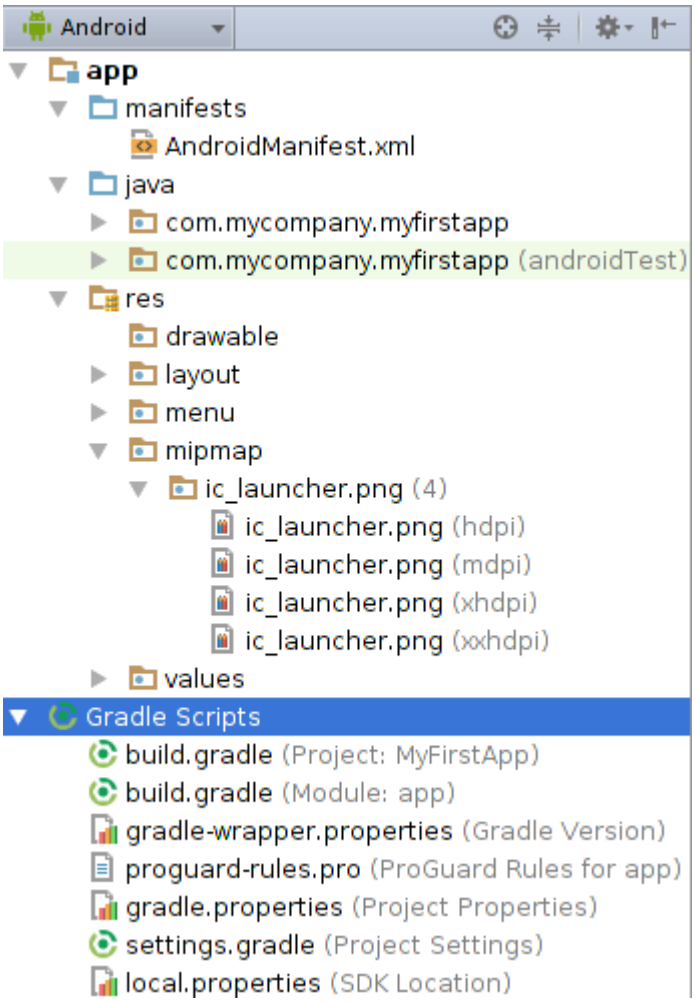
Provides a container for your Google Cloud backend code. This module adds the required code and dependencies for a Java App Engine backend that uses simple HTTP, Cloud Endpoints, and Cloud Messaging to connect to your app. You can develop your backend to provide cloud services your app needs.

Using Android Studio to create and develop your Google Cloud module lets you manage app code and backend code in the same project. You can also run and test your backend code locally, and use Android Studio to deploy your Google Cloud module.

```
dependencies {
  compile project(':my-library-module')
}
```

Project Files

UNIT 1



By default, Android Studio displays your project files in the **Android** view. This view does not reflect the actual file hierarchy on disk, but is organized by modules and file types to simplify navigation between key source files of your project, hiding certain files or directories that are not commonly used. Some of the structural changes compared to the structure on disk include the following:

- Shows all the project's build-related configuration files in a top-level **Gradle Script** group.
- Shows all manifest files for each module in a module-level group (when you have different manifest files for different product flavors and build types).
- Shows all alternative resource files in a single group, instead of in separate folders per resource qualifier. For example, all density versions of your launcher icon are visible side-by-side.

Within each Android app module, files are shown in the following groups:

UNIT 1

manifests

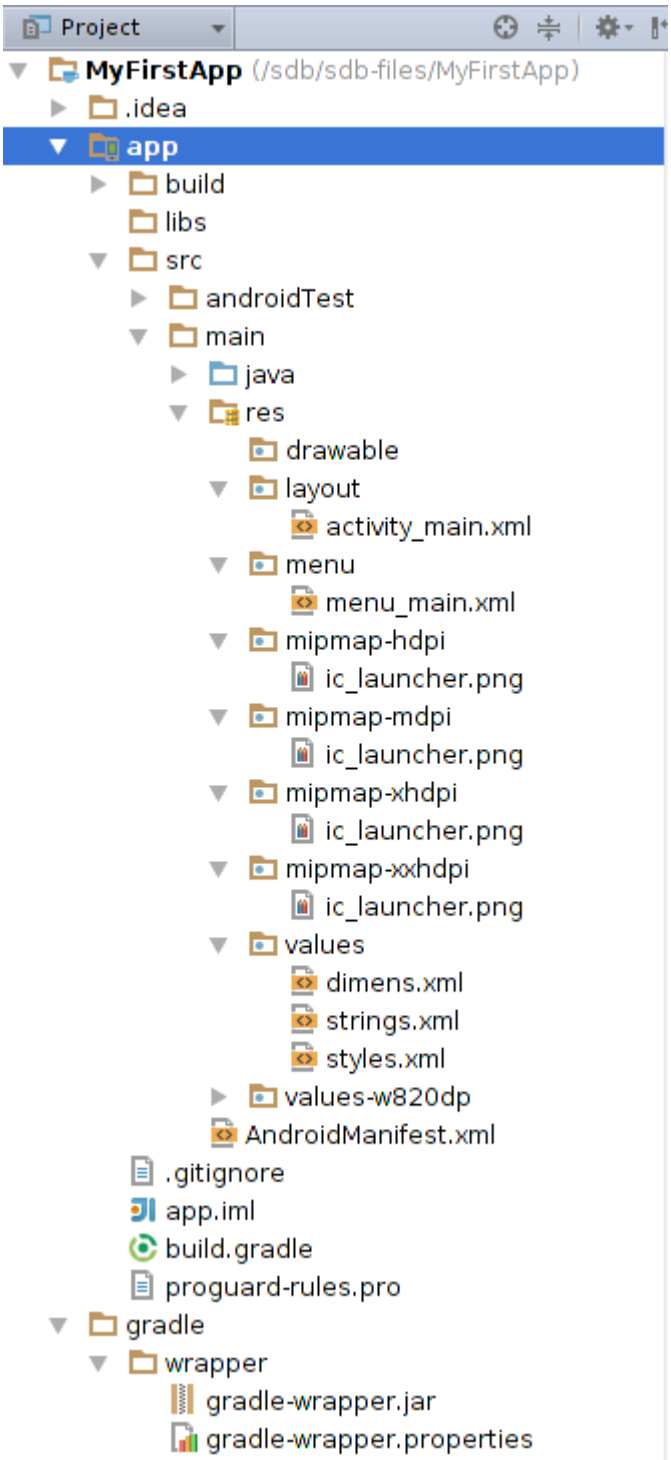
Contains the AndroidManifest.xml file.

java

Contains the Java source code files, separated by package names, including JUnit test code.

res

Contains all non-code resources, such as XML layouts, UI strings, and bitmap images, divided into corresponding sub-directoriesThe Android Project View



To see the actual file structure of the project including all files hidden from the Android view, select **Project**from the dropdown at the top of the **Project** window.

UNIT 1

When you select **Project** view, you can see a lot more files and directories. The most important of which are the following:

module-name/

build/

Contains build outputs.

libs/

Contains private libraries.

src/

Contains all code and resource files for the module in the following subdirectories:

androidTest/

Contains code for instrumentation tests that run on an Android device. For more information, see the Android Test documentation.

main/

Contains the "main" sourceset files: the Android code and resources shared by all build variants (files for other build variants reside in sibling directories, such as src/debug/ for the debug build type).

AndroidManifest.xml

Describes the nature of the application and each of its components. For more information, see the AndroidManifest.xmldocumentation.

java/

Contains Java code sources.

jni/

Contains native code using the Java Native Interface (JNI). gen/

Contains the Java files generated by Android Studio, such as your R.java file and interfaces created from AIDL files.

res/

UNIT 1

Contains application resources, such as drawable files, layout files, and UI string. assets/

Contains file that should be compiled into an .apk file as-is. You can navigate this directory in the same way as a typical file system using URIs and read files as a stream of bytes using the AssetManager . For example, this is a good location for textures and game data.

test/

Contains code for local tests that run on your host JVM.

build.gradle (module)

This defines the module-specific build configurations.

build.gradle (project)

This defines your build configuration that apply to all modules. This file is integral to the project, so you should maintain them in revision control with all other source code.

Project Structure Settings

To change various settings for your Android Studio project, open the **Project Structure** dialog by clicking **File > Project Structure**. It contains the following sections:

- **SDK Location:** Sets the location of the JDK, Android SDK, and Android NDK that your project uses.
- **Project:** Sets the version for Gradle and the Android plugin for Gradle, and the repository location name.
- **Developer Services:** Contains settings for Android Studio add-in components from Google or other third parties. See Developer Services, below.
- **Modules:** Allows you to edit module-specific build configurations, including the target and minimum SDK, the app signature, and library dependencies. See Modules, below.

Developer Services

The *Developer Services* section of the *Project Structure* dialog box contains configuration pages for several services that you can be use with your app. This section contains the following pages:

- **AdMob:** Allows you to turn on Google's AdMob component, which helps you understand your users and show them tailored advertisements.

- **Analytics:** Allows you to turn on Google Analytics, which helps you measure user interactions with your app across various devices and environments.
- **Authentication:** Allows users to use Google Sign-In to sign in to your app with their Google accounts.
- **Cloud:** Allows you to turn on Firebase cloud-based services for your app.
- **Notifications:** Allows you to use Google Cloud Messaging to communicate between your app and your server.

Turning on any of these services may cause Android Studio to add necessary dependencies and permissions to your app. Each configuration page lists these and other actions that Android Studio takes if you enable the associated service.

Modules

The *Modules* settings section lets you change configuration options for each of your project's modules. Each module's settings page is divided into the following tabs:

- **Properties:** Specifies the versions of the SDK and build tools to use to compile the module.
- **Signing:** Specifies the certificate to use to sign your APK.
- **Flavors:** Lets you create multiple build *flavors*, where each flavor specifies a set of configuration settings, such as the module's minimum and target SDK version, and the version code and version name. For example, you might define one flavor that has a minimum SDK of 15 and a target SDK of 21, and another flavor that has a minimum SDK of 19 and a target SDK of 23.
- **Build Types:** Lets you create and modify build configurations, as described in Configuring Gradle Builds. By default, every module has *debug* and*release* build types, but can define more as needed.
- **Dependencies:** Lists the library, file, and module dependencies for this module. You can add, modify, and delete dependencies from this pane

UNIT 2

App user interface designing

User interface design

User interface design or UI design generally refers to the visual layout of the elements that a user might interact with in a website, or technological product. This could be the control buttons of a radio, or the visual layout of a webpage. User interface designs must not only be attractive to potential users, but must also be functional and created with users in mind.

User interface design important for usability

User interface design can dramatically affect the usability and user experience of an application. If a user interface design is too complex or not adapted to targeted users, the user may not be able to find the information or service they are looking for. In website design, this can affect conversion rates. The layout of a user interface design should also be clearly set out for users so that elements can be found in a logical position by the user.

To optimize user interface design

User interface designs should be optimized so that the user can operate an application as quickly and easily as possible. Many experts believe that UI design should be simple and intuitive, often using metaphors from non-computer systems. With a more intuitive user interface design, users will be able to navigate around a website easily, finding the product or service they want quickly. One way to check the intuitiveness of a user interface design is through usability testing. The feedback from usability testing can then be used to optimize the user interface design of a prototype or final product.

To Design A Mobile App Using User Interface Design Principles

The Structure Principle

Design should organize the user interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.

The Simplicity Principle

The design should make simple, common tasks easy, communicating clearly and simply in the user's own language, and providing good shortcuts that are meaningfully related to longer procedures.

The Visibility Principle

The design should make all needed options and materials for a given task visible without distracting the user with extraneous or redundant information. Good designs don't overwhelm users with alternatives or confuse them with unneeded information.

The Feedback Principle

The design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.

The Tolerance Principle

The design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.

The Reuse Principle

The design should reuse internal and external components and behaviors, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember.

Mobile UI resources- Layout, UI elements, Draw-able

Android - UI Layouts

The basic building block for user interface is a **View** object which is created from the View class and 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 like buttons, text fields, etc.

The **ViewGroup** is a subclass of **View** and provides invisible container that hold other Views or other ViewGroups and define their layout properties.

At third level we have different layouts which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/ViewGroup** objects or you can declare your layout using simple XML file **main_layout.xml** which is located in the res/layout folder of your project.

Android Layout Types

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

Sr.No	Layout & Description
1	Linear Layout LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	Relative Layout RelativeLayout is a view group that displays child views in relative positions.

3	Table Layout TableLayout is a view that groups views into rows and columns.
4	Absolute Layout AbsoluteLayout enables you to specify the exact location of its children.
5	Frame Layout The FrameLayout is a placeholder on screen that you can use to display a single view.
6	List View ListView is a view group that displays a list of scrollable items.
7	Grid View GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and their are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the view.
2	**android:layout_width** This is the width of the layout.
3	**android:layout_height** This is the height of the layout
4	**android:layout_marginTop** This is the extra space on the top side of the layout.

5	**android:layout_marginBottom** This is the extra space on the bottom side of the layout.
6	**android:layout_marginLeft** This is the extra space on the left side of the layout.
7	**android:layout_marginRight** This is the extra space on the right side of the layout.
8	**android:layout_gravity** This specifies how child Views are positioned.
9	**android:layout_weight** This specifies how much of the extra space in the layout should be allocated to the View.
10	**android:layout_x** This specifies the x-coordinate of the layout.
11	**android:layout_y** This specifies the y-coordinate of the layout.
12	**android:layout_width** This is the width of the layout.
13	**android:layout_width** This is the width of the layout.
14	**android:paddingLeft** This is the left padding filled for the layout.
15	**android:paddingRight** This is the right padding filled for the layout.
16	**android:paddingTop** This is the top padding filled for the layout.

17	**android:paddingBottom** This is the bottom padding filled for the layout.

UNIT 2

Mobile UI resources- Draw-able, Menu

Android - UI 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, check box, zoom buttons, toggle buttons, and many more.

A **View** is an object that draws something on the screen that the user can interact with and a **ViewGroup** is an object that holds other View (and ViewGroup) objects in order to define the layout of the user interface.

You define your layout in an XML file which offers a human-readable structure for the layout, similar to HTML. For example, a simple vertical layout with a text view and a button looks like this −

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <TextView android:id="@+id/text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="I am a TextView" />

   <Button android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="I am a Button" />
</LinearLayout>
```

Android UI Controls

There are number of UI controls provided by Android that allow you to build the graphical user interface for your app.

Sr.No.	UI Control & Description

1	TextView This control is used to display text to the user.
2	EditText EditText is a predefined subclass of TextView that includes rich editing capabilities.
3	AutoCompleteTextView The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing.
4	Button A push-button that can be pressed, or clicked, by the user to perform an action.
5	ImageButton An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.
6	CheckBox An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive.
7	ToggleButton An on/off button with a light indicator.
8	RadioButton The RadioButton has two states: either checked or unchecked.
9	RadioGroup A RadioGroup is used to group together one or more RadioButtons.
10	ProgressBar The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
11	Spinner A drop-down list that allows users to select one value from a set.
12	TimePicker The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
13	DatePicker The DatePicker view enables users to select a date of the day.

Create UI 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, check box, zoom buttons, toggle buttons, and many more.

As explained in previous chapter, a view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is −

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

To create a UI Control/View/Widget you will have to define a view/widget in the layout file and assign it a unique ID as follows −

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <TextView android:id="@+id/text_id"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="I am a TextView" />
</LinearLayout>
```

Then finally create an instance of the Control object and capture it from the layout, use the following −

```
TextView myText = (TextView) findViewById(R.id.text_id);
```

Menus

Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, you should use the Menu APIs to present user actions and other options in your activities.

Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated *Menu* button. With this change, Android apps should migrate away from a dependence on the traditional 6-item menu panel and instead provide an app bar to present common user actions.

Although the design and user experience for some menu items have changed, the semantics to define a set of actions and options is still based on the Menu APIs. This guide shows how to create the three fundamental types of menus or action presentations on all versions of Android:

Options menu and app bar
: The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."

  See the section about Creating an Options Menu.

Context menu and contextual action mode
: A context menu is a floating menu that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.

  The contextual action mode displays action items that affect the selected content in a bar at the top of the screen and allows the user to select multiple items.

  See the section about Creating Contextual Menus.

Popup menu
: A popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command. Actions in a popup menu should **not** directly affect the corresponding content—that's what contextual actions are for. Rather, the popup menu is for extended actions that relate to regions of content in your activity.

  See the section about Creating a Popup Menu.

Defining a Menu in XML

For all menu types, Android provides a standard XML format to define menu items. Instead of building a menu in your activity's code, you should define a menu and all its items in an XML

menu resource. You can then inflate the menu resource (load it as a Menu object) in your activity or fragment.

Using a menu resource is a good practice for a few reasons:

- It's easier to visualize the menu structure in XML.
- It separates the content for the menu from your application's behavioral code.
- It allows you to create alternative menu configurations for different platform versions, screen sizes, and other configurations by leveraging the app resources framework.

To define the menu, create an XML file inside your project's `res/menu/` directory and build the menu with the following elements:

`<menu>`
: Defines a Menu, which is a container for menu items. A `<menu>` element must be the root node for the file and can hold one or more `<item>` and `<group>` elements.

`<item>`
: Creates a MenuItem, which represents a single item in a menu. This element may contain a nested `<menu>` element in order to create a submenu.

`<group>`
: An optional, invisible container for `<item>` elements. It allows you to categorize menu items so they share properties such as active state and visibility. For more information, see the section about Creating Menu Groups.

Here's an example menu named `game_menu.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

The `<item>` element supports several attributes you can use to define an item's appearance and behavior. The items in the above menu include the following attributes:

`android:id`
: A resource ID that's unique to the item, which allows the application to recognize the item when the user selects it.

`android:icon`
: A reference to a drawable to use as the item's icon.

`android:title`
: A reference to a string to use as the item's title.

`android:showAsAction`
: Specifies when and how this item should appear as an action item in the app bar.

These are the most important attributes you should use, but there are many more available. For information about all the supported attributes, see the Menu Resource document.

You can add a submenu to an item in any menu (except a submenu) by adding a `<menu>` element as the child of an `<item>`. Submenus are useful when your application has a lot of functions that can be organized into topics, like items in a PC application's menu bar (File, Edit, View, etc.). For example:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
          android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                  android:title="@string/create_new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
    </item>
</menu>
```

UNIT 2

Activity- states and life cycle

Android system initiates its program with in an **Activity** starting with a call on *onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram: (*image courtesy : android.com*)

The Activity class defines the following call backs i.e. events. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

Sr.No	Callback & Description
1	**onCreate()** This is the first callback and called when the activity is first created.
2	**onStart()** This callback is called when the activity becomes visible to the user.
3	**onResume()** This is called when the user starts interacting with the application.
4	**onPause()** The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
5	**onStop()** This callback is called when the activity is no longer visible.
6	**onDestroy()**

	This callback is called before the activity is destroyed by the system.
7	**onRestart()** This callback is called when the activity restarts after stopping it.

Example

This example will take you through simple steps to show Android application activity life cycle. Follow the following steps to modify the Android application we created in *Hello World Example* chapter −

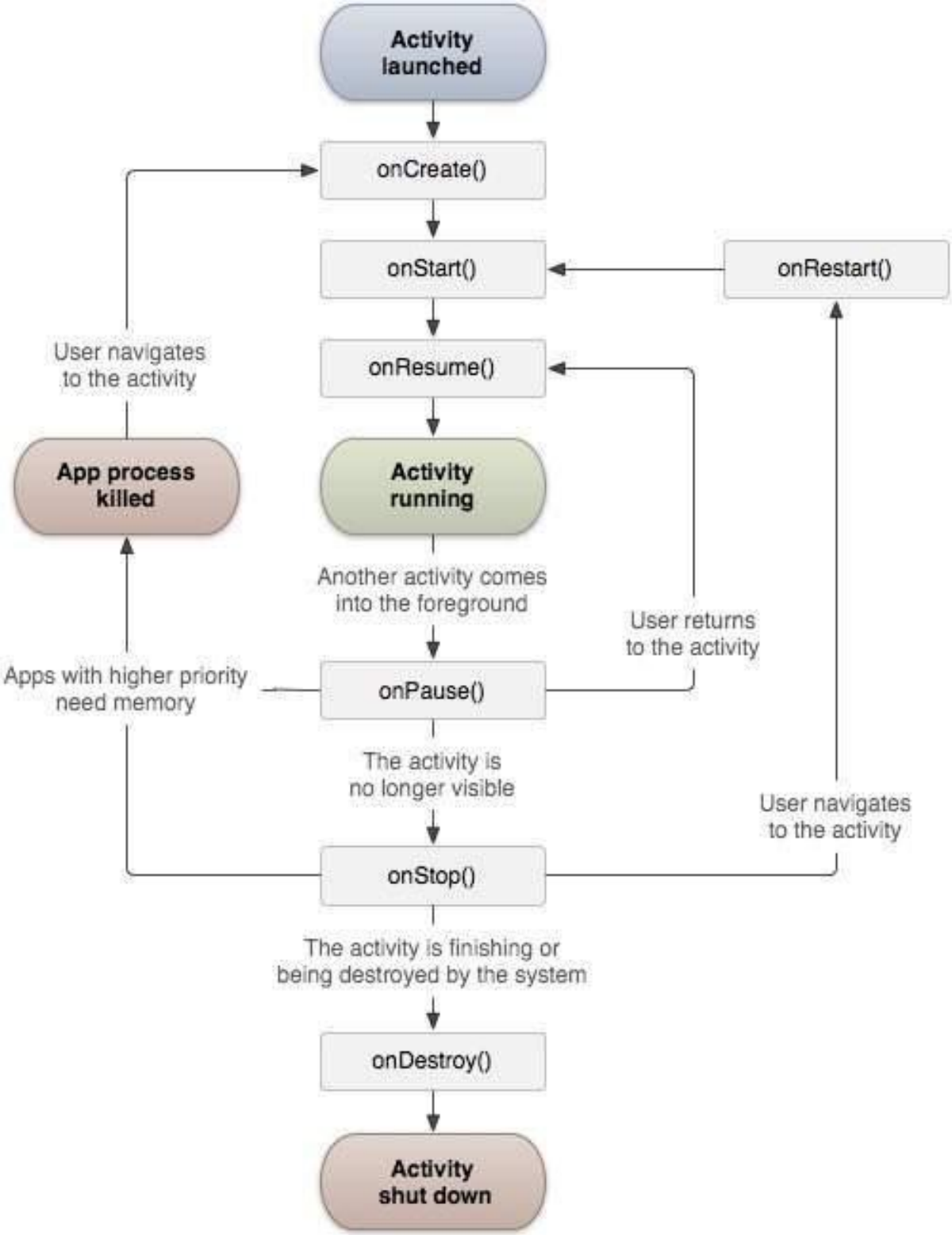
Step	Description
1	You will use Android studio to create an Android application and name it as *HelloWorld* under a package *com.example.helloworld* as explained in the *Hello World Example* chapter.
2	Modify main activity file *MainActivity.java* as explained below. Keep rest of the files unchanged.
3	Run the application to launch Android emulator and verify the result of the changes done in the application.

An activity class loads all the UI component using the XML file available in *res/layout* folder of the project. Following statement loads UI components from *res/layout/activity_main.xml file*:

```
setContentView(R.layout.activity_main);
```

An application can have one or more activities without any restrictions. Every activity you define for your application must be declared in your *AndroidManifest.xml* file and the main activity for your app must be declared in the manifest with an <intent-filter> that includes the MAIN action and LAUNCHER category as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.tutorialspoint7.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```


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

UNIT 2

App functionality beyond user interface- Services - states and lifecycle

Android - Services

A **service** is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states −

Sr.No.	State & Description
1	**Started** A service is **started** when an application component, such as an activity, starts it by calling *startService()*. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
2	**Bound** A service is **bound** when an application component binds to it by calling *bindService()*. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

A service has life cycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage. The following diagram on the left shows the life cycle when the service is created with startService() and the diagram on the right shows the life cycle when the service is created with bindService(): *(image courtesy : android.com)*

To create an service, you create a Java class that extends the Service base class or one of its existing subclasses. The **Service** base class defines various callback methods and the most important are given below. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

Sr.No.	Callback & Description

1	**onStartCommand()** The system calls this method when another component, such as an activity, requests that the service be started, by calling *startService()*. If you implement this method, it is your responsibility to stop the service when its work is done, by calling *stopSelf()* or *stopService()* methods.
2	**onBind()** The system calls this method when another component wants to bind with the service by calling *bindService()*. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an *IBinder* object. You must always implement this method, but if you don't want to allow binding, then you should return *null*.
3	**onUnbind()** The system calls this method when all clients have disconnected from a particular interface published by the service.
4	**onRebind()** The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its *onUnbind(Intent)*.
5	**onCreate()** The system calls this method when the service is first created using *onStartCommand()* or *onBind()*. This call is required to perform one-time set-up.
6	**onDestroy()** The system calls 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, receivers, etc.

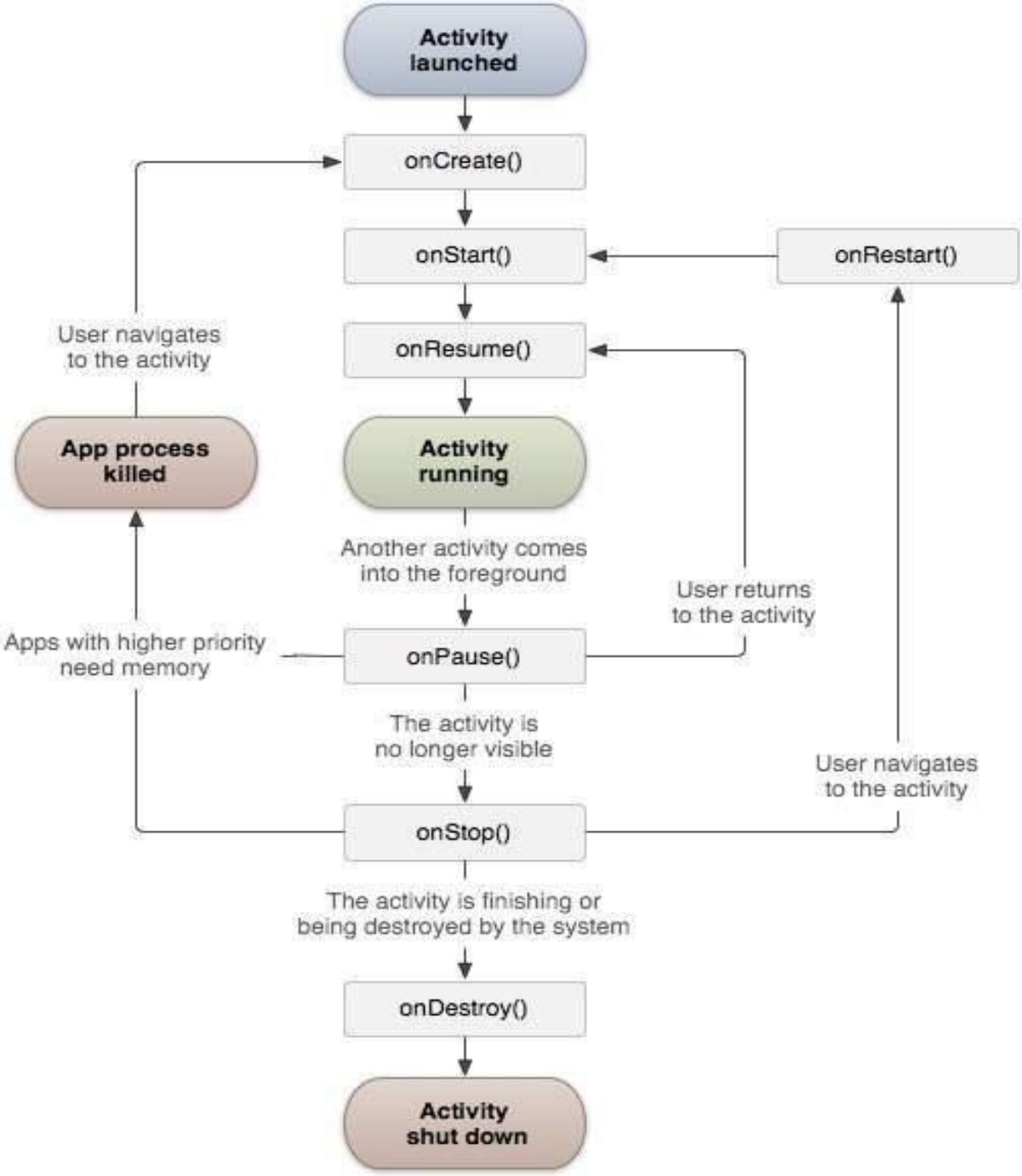
The following skeleton service demonstrates each of the life cycle methods −

```
package com.tutorialspoint;

import android.app.Service;
import android.os.IBinder;
import android.content.Intent;
import android.os.Bundle;

public class HelloService extends Service {

   /** indicates how to behave if the service is killed */
   int mStartMode;

   /** interface for clients that bind */
   IBinder mBinder;

   /** indicates whether onRebind should be used */
   boolean mAllowRebind;
```

```
/** Called when the service is being created. */
@Override
public void onCreate() {

}

/** The service is starting, due to a call to startService() */
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
   return mStartMode;
}

/** A client is binding to the service with bindService() */
@Override
public IBinder onBind(Intent intent) {
   return mBinder;
}

/** Called when all clients have unbound with unbindService() */
@Override
public boolean onUnbind(Intent intent) {
   return mAllowRebind;
}

/** Called when a client is binding to the service with bindService()*/
@Override
public void onRebind(Intent intent) {

}

/** Called when The service is no longer used and is being destroyed */
@Override
public void onDestroy() {
```

UNIT 2

Interaction amongst activities



```
ackage com.example.tutorialspoint7.myapplication;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.View;

public class MainActivity extends Activity {
   String msg = "Android : ";

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d(msg, "The onCreate() event");
   }

   public void startService(View view) {
      startService(new Intent(getBaseContext(), MyService.class));
   }

   // Method to stop the service
   public void stopService(View view) {
      stopService(new Intent(getBaseContext(), MyService.class));
   }
}
```

Following is the content of **MyService.java**. This file can have implementation of one or more methods associated with Service based on requirements. For now we are going to implement only two methods *onStartCommand()* and *onDestroy()* −

```
package com.example.tutorialspoint7.myapplication;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.widget.Toast;

/**
   * Created by TutorialsPoint7 on 8/23/2016.
*/

public class MyService extends Service {
   @Nullable
   @Override
   public IBinder onBind(Intent intent) {
      return null;
   }
```

```
   @Override
   public int onStartCommand(Intent intent, int flags, int startId) {
      // Let it continue running until it is stopped.
      Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
      return START_STICKY;
   }

   @Override
   public void onDestroy() {
      super.onDestroy();
      Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
   }
}
```

Following will the modified content of *AndroidManifest.xml* file. Here we have added <service.../> tag to include our service −

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.tutorialspoint7.myapplication">

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">

      <activity android:name=".MainActivity">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

      <service android:name=".MyService" />
   </application>

</manifest>
```

UNIT 2

App functionality beyond user interface-

Threads, Async task, Services

Creating multi-thread applications for Android application development is a challenging task for many Android developers. Single and multi-threading approaches are used to create complex Android enterprise mobile apps, as they help to streamline functional operation of the code. But sometimes it is necessary to update the UI from the background thread about the operations performed. If you ever tried to access an UI element from a background thread, you have already noticed that an exception is thrown. This article will explain how to notify activity with the information posted by another thread.

For creating multi-thread apps, Android by default does not allow the developer to modify the UI outside of the main thread. This problem is faced by many coders and if you still managed to do it you'd be breaking the second rule of the single-threaded model which is "*do not access the Android UI toolkit from outside the UI thread*", as stated here http://developer.android.com/guide/components/processes-and-threads.html.

Problem

While creating complex multi-thread functions in an enterprise android app, the information generated from these threads are not notified in the UI resulting in mismatch of app business logic and crashing of the application.

Solution

- Implement a Handler class, override method handleMessage() which will read messages from thread queue
- Next post message using sendMessage() method in worker thread

There are many situations when it is required to have a thread running in the background and send information to main Activity's UI thread. From the architectural level we can use two different approaches for notifying thread activity.

1. Use of Android AsyncTask class
2. Start a new thread

   Using AsyncTask is very convenient as there might be situations when you really need to

construct a worker thread by yourself. In such situation, you will need to send some information back to Activity thread. Keep in mind that Android doesn't allow other threads to modify any content of main UI thread as stated above. Instead you're required to wrap data into messages and send them through message queue.
You can implement this operation in two parts:

Part 1 – Add Handler

Add an instance of Handler class to your MapActivity instance.

```
public class MyMap extends MapActivity {
. . . .
public Handler _handler = new Handler() {
@Override
public void handleMessage(Message msg) {
Log.d(TAG, String.format("Handler.handleMessage(): msg=%s", msg));
// This is where main activity thread receives messages
// Put here your handling of incoming messages posted by other threads
super.handleMessage(msg);
}
};
. . . .
}
```

Part 2 – Post Message

In the worker thread post a message to activity main queue whenever you need Add handler class instance to your MapActivity instance.

```
/**
* Performs background job
*/
class MyThreadRunner implements Runnable {
// @Override
public void run() {
while (!Thread.currentThread().isInterrupted()) {
// Just dummy message -- real implementation will put some meaningful data in it
Message msg = Message.obtain();
msg.what = 999;
MyMap.this._handler.sendMessage(msg);
// Dummy code to simulate delay while working with remote server
try {
Thread.sleep(5000);
} catch (InterruptedException e) {
Thread.currentThread().interrupt();
}
}
}
}
```

AsyncTask

AsyncTask enables proper and easy use of the UI thread. This class allows you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

AsyncTask is designed to be a helper class around Thread and Handler and does not constitute a generic threading framework. AsyncTasks should ideally be used for short operations (a few seconds at the most.) If you need to keep threads running for long periods of time, it is highly recommended you use the various APIs provided by the java.util.concurrent package such as Executor, ThreadPoolExecutor and FutureTask.

An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called Params, Progress and Result, and 4 steps, called onPreExecute, doInBackground, onProgressUpdate and onPostExecute.

public abstract class AsyncTask
extends Object

java.lang.Object
  ↳ android.os.AsyncTask<Params, Progress, Result>

AsyncTask must be subclassed to be used. The subclass will override at least one method (doInBackground(Params...)), and most often will override a second one (onPostExecute(Result).)

AsyncTask's generic types

The three types used by an asynchronous task are the following:

1. Params, the type of the parameters sent to the task upon execution.
2. Progress, the type of the progress units published during the background computation.
3. Result, the type of the result of the background computation.

The 4 steps

When an asynchronous task is executed, the task goes through 4 steps:

1. onPreExecute(), invoked on the UI thread before the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
2. doInBackground(Params...), invoked on the background thread immediately after onPreExecute() finishes executing. This step is used to perform background computation

that can take a long time. The parameters of the asynchronous task are passed to this step. The result of the computation must be returned by this step and will be passed back to the last step. This step can also use publishProgress(Progress...) to publish one or more units of progress. These values are published on the UI thread, in the onProgressUpdate(Progress...) step.

3. onProgressUpdate(Progress...), invoked on the UI thread after a call to publishProgress(Progress...). The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing. For instance, it can be used to animate a progress bar or show logs in a text field.
4. onPostExecute(Result), invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

Threading rules

There are a few threading rules that must be followed for this class to work properly:

- The AsyncTask class must be loaded on the UI thread. This is done automatically as of JELLY_BEAN.
- The task instance must be created on the UI thread.
- execute(Params...) must be invoked on the UI thread.
- Do not call onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress...) manually.
- The task can be executed only once (an exception will be thrown if a second execution is attempted.)

UNIT 2

App functionality beyond user interface- Async task, Services

AsyncTask

AsyncTask enables proper and easy use of the UI thread. This class allows you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

AsyncTask is designed to be a helper class around Thread and Handler and does not constitute a generic threading framework. AsyncTasks should ideally be used for short operations (a few seconds at the most.) If you need to keep threads running for long periods of time, it is highly recommended you use the various APIs provided by the java.util.concurrent package such as Executor, ThreadPoolExecutor and FutureTask.

An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called Params, Progress and Result, and 4 steps, called onPreExecute, doInBackground, onProgressUpdate and onPostExecute.

public abstract class AsyncTask
extends Object

java.lang.Object
  ↳ android.os.AsyncTask<Params, Progress, Result>
AsyncTask must be subclassed to be used. The subclass will override at least one method (doInBackground(Params...)), and most often will override a second one (onPostExecute(Result).)

AsyncTask's generic types

The three types used by an asynchronous task are the following:

1. Params, the type of the parameters sent to the task upon execution.
2. Progress, the type of the progress units published during the background computation.

3. Result, the type of the result of the background computation.

The 4 steps

When an asynchronous task is executed, the task goes through 4 steps:

1. onPreExecute(), invoked on the UI thread before the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
2. doInBackground(Params...), invoked on the background thread immediately after onPreExecute() finishes executing. This step is used to perform background computation that can take a long time. The parameters of the asynchronous task are passed to this step. The result of the computation must be returned by this step and will be passed back to the last step. This step can also use publishProgress(Progress...) to publish one or more units of progress. These values are published on the UI thread, in the onProgressUpdate(Progress...) step.
3. onProgressUpdate(Progress...), invoked on the UI thread after a call to publishProgress(Progress...). The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing. For instance, it can be used to animate a progress bar or show logs in a text field.
4. onPostExecute(Result), invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

Threading rules

There are a few threading rules that must be followed for this class to work properly:

- The AsyncTask class must be loaded on the UI thread. This is done automatically as of JELLY_BEAN.
- The task instance must be created on the UI thread.
- execute(Params...) must be invoked on the UI thread.
- Do not call onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress...) manually.
- The task can be executed only once (an exception will be thrown if a second execution is attempted.)

Notifications

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.

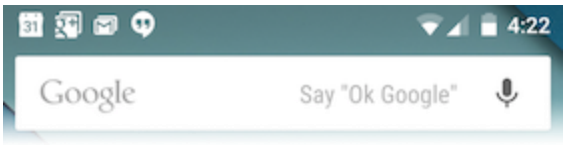


Figure 1. Notifications in the notification area.

Design Considerations

Notifications, as an important part of the Android user interface, have their own design guidelines. The material design changes introduced in Android 5.0 (API level 21) are of particular importance, and you should review the Material Design training for more information. To learn how to design notifications and their interactions, read the Notifications design guide.

Creating a Notification

Notifications in Android o

The Android O Developer Preview introduces new features and capabilities for users and developers related to notifications, including notification channels. To learn about the new changes, see Android O for Developers.

A Notification object *must* contain the following:

- A small icon, set by setSmallIcon()
- A title, set by setContentTitle()
- Detail text, set by setContentText()

Optional notification contents and settings

All other notification settings and contents are optional. To learn more about them, see the reference documentation for NotificationCompat.Builder.

Notification actions

Although they're optional, should add at least one action to your notification. An action allows users to go directly from the notification to an Activity in your application, where they can look at one or more events or do further work.

A notification can provide multiple actions. Always define the action that's triggered when the user clicks the notification; usually this action opens an Activity in your application. Also add buttons to the notification that perform additional actions such as snoozing an alarm or responding immediately to a text message; this feature is available as of Android 4.1. If use additional action buttons, you must also make their functionality available in an Activity in your app; see the section Handling compatibility for more details.

Inside a Notification, the action itself is defined by a PendingIntent containing an Intent that starts an Activity in your application. To associate the PendingIntent with a gesture, call the appropriate method of NotificationCompat.Builder. For example, if want to start Activity when the user clicks the notification text in the notification drawer, add the PendingIntent by calling setContentIntent().

Starting an Activity when the user clicks the notification is the most common action scenario. You can also start an Activity when the user dismisses a notification. In Android 4.1 and later, you can start an Activity from an action button. To learn more, read the reference guide for NotificationCompat.Builder.

Notification priority

If you wish, you can set the priority of a notification. The priority acts as a hint to the device UI about how the notification should be displayed. To set a notification's priority, call NotificationCompat.Builder.setPriority() and pass in one of the NotificationCompat priority constants. There are five priority levels, ranging from PRIORITY_MIN (-2) to PRIORITY_MAX (2); if not set, the priority defaults to PRIORITY_DEFAULT (0).

For information about setting an appropriate priority level, see "Correctly set and manage notification priority" in the Notifications Design guide.

Creating a simple notification

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 =
        new NotificationCompat.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 = new Intent(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());
```

Android - Broadcast Receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

There are following two important steps to make BroadcastReceiver works for the system broadcasted intents −

- Creating the Broadcast Receiver.
- Registering Broadcast Receiver

There is one additional steps in case you are going to implement your custom intents then you will have to create and broadcast those intents.

Creating the Broadcast Receiver

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and overriding the onReceive() method where each message is received as a **Intent** object parameter.

```
public class MyReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show();
  }
}
```

Registering Broadcast Receiver

An application listens for specific broadcast intents by registering a broadcast receiver in *AndroidManifest.xml* file. Consider we are going to register *MyReceiver* for system generated event ACTION_BOOT_COMPLETED which is fired by the system once the Android system has completed the boot process.

```
<application
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme" >
  <receiver android:name="MyReceiver">

    <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED">
      </action>
    </intent-filter>

  </receiver>
</application>
```

Now whenever your Android device gets booted, it will be intercepted by BroadcastReceiver *MyReceiver* and implemented logic inside *onReceive()* will be executed.

There are several system generated events defined as final static fields in the **Intent** class. The following table lists a few important system events.

Sr.No	Event Constant & Description
1	**android.intent.action.BATTERY_CHANGED** Sticky broadcast containing the charging state, level, and other information about the battery.
2	**android.intent.action.BATTERY_LOW** Indicates low battery condition on the device.
3	**android.intent.action.BATTERY_OKAY** Indicates the battery is now okay after being low.
4	**android.intent.action.BOOT_COMPLETED** This is broadcast once, after the system has finished booting.
5	**android.intent.action.BUG_REPORT** Show activity for reporting a bug.
6	**android.intent.action.CALL** Perform a call to someone specified by the data.
7	**android.intent.action.CALL_BUTTON** The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.

8	**android.intent.action.DATE_CHANGED** The date has changed.
9	**android.intent.action.REBOOT** Have the device reboot.

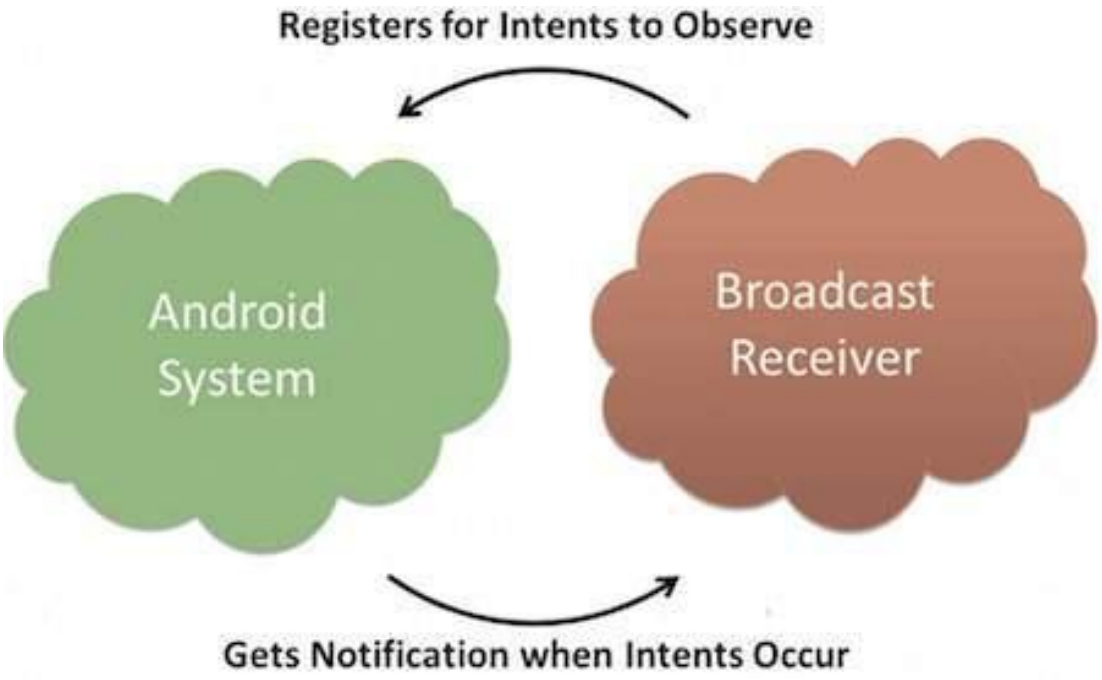
Broadcasting Custom Intents

If you want your application itself should generate and send custom intents then you will have to create and send those intents by using the *sendBroadcast()* method inside your activity class. If you use the *sendStickyBroadcast(Intent)* method, the Intent is **sticky**, meaning the *Intent* you are sending stays around after the broadcast is complete.

```
public void broadcastIntent(View view) {
   Intent intent = new Intent();
   intent.setAction("com.tutorialspoint.CUSTOM_INTENT");
   sendBroadcast(intent);
}
```

This intent *com.tutorialspoint.CUSTOM_INTENT* can also be registered in similar way as we have regsitered system generated intent.

```
<application
   android:icon="@drawable/ic_launcher"
   android:label="@string/app_name"
   android:theme="@style/AppTheme" >
   <receiver android:name="MyReceiver">

      <intent-filter>
         <action android:name="com.tutorialspoint.CUSTOM_INTENT">
         </action>
      </intent-filter>

   </receiver>
</application>
```


Registers for Intents to Observe
Android System
Broadcast Receiver
Gets Notification when Intents Occur

Telephony.Sms

public static final class Telephony.Sms extends Object implements BaseColumns, Telephony.TextBasedSmsColumns

java.lang.Object
  ↳ android.provider.Telephony.Sms

Contains all text-based SMS messages.

Nested classes	
class	Telephony.Sms.Conversations Contains all sent text-based SMS messages in the SMS app.
class	Telephony.Sms.Draft Contains all sent text-based SMS messages in the SMS app.
class	Telephony.Sms.Inbox Contains all text-based SMS messages in the SMS app inbox.
class	Telephony.Sms.Intents Contains constants for SMS related Intents that are broadcast.
class	Telephony.Sms.Outbox Contains all pending outgoing text-based SMS messages.
class	Telephony.Sms.Sent Contains all sent text-based SMS messages in the SMS app.

SmsManager

public final class SmsManager extends Object

java.lang.Object
  ↳ android.telephony.SmsManager

Manages SMS operations such as

sending data,

text, and

pdu SMS messages.

Get this object by calling the static method getDefault().

For information about how to behave as the default SMS app on Android 4.4 (API level 19) and higher, see Telephony.

UNIT 3

Native Data Handling

What is native code for Android devices, and what is the NDK?

Android apps run within the Dalvik virtual machine, which interprets device-agnostic, cross-platform commands into instructions for the specific device that it is running on.The speed and memory overhead is a worthwhile tradeoff. In some cases, developers need the absolute fastest performance possible. The NDK allows embedding C and C++ components within Android apps, allowing the most performance-intensive pieces to be as close to the hardware as possible. This comes at a cost, though — using native code complicates development. There are more tools to use and infrastructure to set up. Also, some details that were handled by the Dalvik virtual machine must now be handled by the developer. For these reasons, native code should be used only when necessary.

When native code is needed

There are times that using native code can be advantageous, such as processing data or computing physics and graphics for games. Access to existing native libraries, as well as high-performance code, can also be good reasons.

Uses for the native code

Game engine developers often dive right in to native code. The limited speed and memory of mobile devices means native code may be necessary to squeeze every bit of potential out for them

The native-activity sample resides under the NDK installation root, in samples/native-activity. It is a very simple example of a purely native application, with no Java source code. In the absence of any Java source, the Java compiler still creates an executable stub for the virtual machine to run.

```
#include <EGL/egl.h>
#include <GLES/gl.h>
```

#include <android/sensor.h>
#include <android/log.h>
#include <android_native_app_glue>

Create a new Native Activity project

In this tutorial, you'll first create a new Android Native Activity project and then build and run the default app in the Visual Studio Emulator for Android.

To create a new project

1. Open Visual Studio. On the menu bar, choose **File**, **New**, **Project**.
2. In the **New Project** dialog box, under **Templates**, choose **Visual C++**, **Cross Platform**, and then choose the **Native-Activity Application (Android)** template.
3. Give the app a name like MyAndroidApp, and then choose **OK**.

   Visual Studio creates the new solution and opens Solution Explorer.

The new Android Native Activity app solution includes two projects:

- **MyAndroidApp.NativeActivity** contains the references and glue code for your app to run as a Native Activity on Android. The implementation of the entry points from the glue code are in main.cpp. Precompiled headers are in pch.h. This Native Activity app project is compiled into a shared library .so file which is picked up by the Packaging project.
- **MyAndroidApp.Packaging** creates the .apk file for deployment on an Android device or emulator. This contains the resources and AndroidManifest.xml file where you set manifest properties. It also contains the build.xml file that controls the Ant build process. It's set as the startup project by default, so that it can be deployed and run directly from Visual Studio.

Build and run the default Android Native Activity app

Build and run the app generated by the template to verify your installation and setup. For this initial test, run the app on one of the device profiles installed by the Visual Studio Emulator for Android. If you prefer to test your app on another target, you can load the target emulator or connect the device to your computer.

To build and run the default Native Activity app

1. If it is not already selected, choose **x86** from the **Solution Platforms** dropdown list.

   If the **Solution Platforms** list isn't showing, choose **Solution Platforms** from the **Add/Remove Buttons** list, and then choose your platform.

2. On the menu bar, choose **Build**, **Build Solution**.

The Output window displays the output of the build process for the two projects in the solution.

3. Choose one of the VS Emulator Android Phone (x86) profiles as your deployment target.

   If you have installed other emulators or connected an Android device, you can choose them in the deployment target dropdown list.

4. Press F5 to start debugging, or Shift+F5 to start without debugging.

   Visual Studio starts the emulator, which takes a few seconds to load and deploy your code. Once your app has started, you can set breakpoints and use the debugger to step through code, examine locals, and watch values.

5. Press Shift + F5 to stop debugging.

   The emulator is a separate process that continues to run. You can edit, compile, and deploy your code multiple times to the same emulator.

UNIT 3

On Device File I/O

sr.No	Part & Description
1	**prefix** This is always set to content://
2	**authority** This specifies the name of the content provider, for example *contacts*, *browser* etc. For third-party content providers, this could be the fully qualified name, such as *com.tutorialspoint.statusprovider*
3	**data_type** This indicates the type of data that this particular provider provides. For example, if you are getting all the contacts from the *Contacts* content provider, then the data path would be *people* and URI would look like this*content://contacts/people*
4	**id** This specifies the specific record requested. For example, if you are looking for contact number 5 in the Contacts content provider then URI would look like this *content://contacts/people/5*.

UNIT 3

SharedPreferences

Saving Key-Value Sets

1. Get a Handle to a SharedPreferences
2. Write to Shared Preferences
3. Read from Shared Preferences

Using Shared Preferences

If you have a relatively small collection of key-values that you'd like to save, you should use the `SharedPreferences` APIs. A `SharedPreferences` object points to a file containing key-value pairs and provides simple methods to read and write them. Each `SharedPreferences` file is managed by the framework and can be private or shared.

Get a Handle to a SharedPreferences

create a new shared preference file or access an existing one by calling one of two methods:

- `getSharedPreferences()` — Use this if you need multiple shared preference files identified by name, which you specify with the first parameter. You can call this from any `Context` in your app.
- `getPreferences()` — Use this from an `Activity` if you need to use only one shared preference file for the activity. Because this retrieves a default shared preference file that belongs to the activity, you don't need to supply a name.

For example, the following code is executed inside a `Fragment`. It accesses the shared preferences file that's identified by the resource string `R.string.preference_file_key` and opens it using the private mode so the file is accessible by only your app.

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
        getString(R.string.preference_file_key),
Context.MODE_PRIVATE);
```

When naming your shared preference files, you should use a name that's uniquely identifiable to your app, such as `"com.example.myapp.PREFERENCE_FILE_KEY"`

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_default);
long highScore =
sharedPref.getInt(getString(R.string.saved_high_score),
defaultValue);
```

SQLITE

What is SQLite?
SQLite is a software library that provides a relational database management system.

The lite in SQLite means light weight in terms of setup, database administration, and required resource.

SQLite has the following noticeable features**: self-contained, serverless, zero-configuration, transactional.**

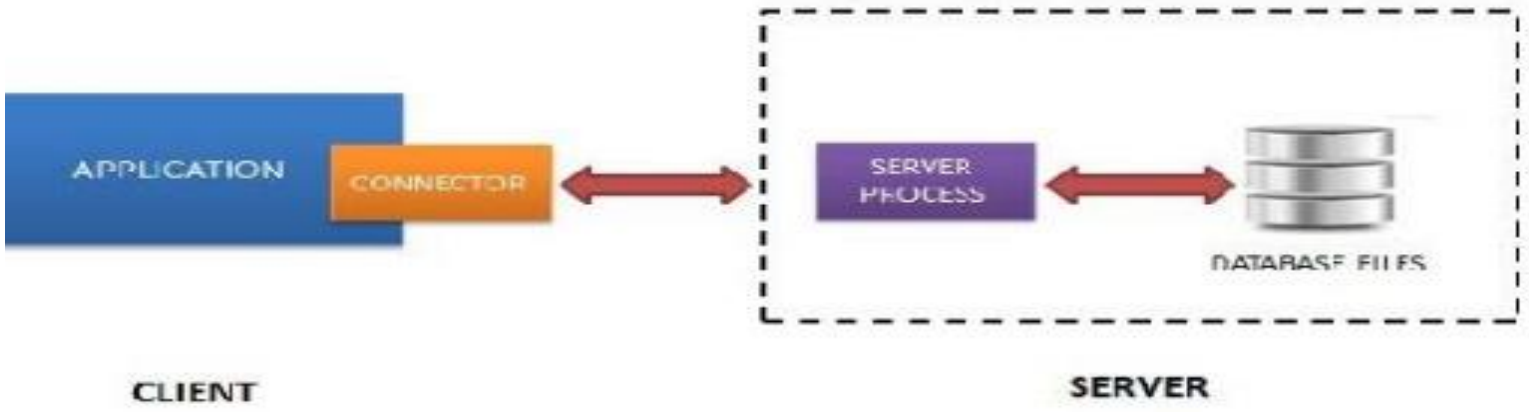
SERVERLESS
Normally, an RDBMS such as MySQL, PostgreSQL, etc., requires a separate server process to operate. The applications that want to access the database server use TCP/IP protocol to send and receive requests. This is called client/server architecture.

TRANSACTIONAL

All transactions in SQLite are fully ACID-compliant. It means all queries and changes are Atomic, Consistent, Isolated, and Durable. In other words, all changes within a transaction take place completely or not at all even when an unexpected situation like application crash, power failure, or operating system crash occurs.

SQLite distinctive features SQLite uses dynamic types for tables. It means you can store any value in any column, regardless of the data type. SQLite allows a single database connection to access multiple database files simultaneously. This brings many nice features like joining tables in different databases or copying data between databases in a single command. SQLite is capable of creating in-memory databases which are very fast to work with.

The following diagram illustrates the RDBMS client/server architecture:



RDBMS Client Server Architecture

SQLite does NOT work this way.

SQLite does NOT require a server to run.

SQLite database is integrated with the application that accesses the database. The applications interact with the SQLite database read and write directly from the database files stored on disk.

The following diagram illustrates the SQLite server-less architecture:

What is SQLite

SELF-CONTAINED

SQLite is self-contained means it requires minimal support from the operating system or external library. This makes SQLite usable in any environments especially in embedded devices like iPhones, Android phones, game consoles, handheld media players, etc. SQLite is developed using ANSI-C. The source code is available as a big sqlite3.c and its header file sqlite3.h. If you want to develop an application that uses SQLite, you just need to drop these files into your project and compile it with your code.

ZERO-CONFIGURATION

Because of the serverless architecture, you don’t need to “install” SQLite before using it. There is no server process that needs to be configured, started, and stopped. In addition, SQLite does not use any configuration files.

UNIT 3

SQLite

SQLite is a opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.

SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

Database - Package

The main package is android.database.sqlite that contains the classes to manage your own databases

Database - Creation

In order to create a database you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object.Its syntax is given below

```
SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",MODE_PRIVATE,null);
```

Apart from this , there are other functions available in the database package , that does this job. They are listed below

Sr.No	Method & Description
1	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)** This method only opens the existing database with the appropriate flag mode. The common flags mode could be OPEN_READWRITE OPEN_READONLY
2	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)** It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases
3	**openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)** It not only opens but create the database if it not exists. This method is equivalent to openDatabase method.

4	**openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)** This method is similar to above method but it takes the File object as a path rather then a string. It is equivalent to file.getPath()

Database - Insertion

we can create table or insert data into table using execSQL method defined in SQLiteDatabase class. Its syntax is given below

```
mydatabase.execSQL("CREATE TABLE IF NOT EXISTS
TutorialsPoint(Username VARCHAR,Password VARCHAR);");
mydatabase.execSQL("INSERT INTO TutorialsPoint
VALUES('admin','admin');");
```

This will insert some values into our table in our database. Another method that also does the same job but take some additional parameter is given below

Sr.No	Method & Description
1	**execSQL(String sql, Object[] bindArgs)** This method not only insert data , but also used to update or modify already existing data in database using bind arguments

Database - Fetching

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

```
Cursor resultSet = mydatbase.rawQuery("Select * from
TutorialsPoint",null);
resultSet.moveToFirst();
String username = resultSet.getString(0);
String password = resultSet.getString(1);
```

There are other functions available in the Cursor class that allows us to effectively retrieve the data. That includes

Sr.No	Method & Description
1	**getColumnCount()** This method return the total number of columns of the table.

2	**getColumnIndex(String columnName)** This method returns the index number of a column by specifying the name of the column
3	**getColumnName(int columnIndex)** This method returns the name of the column by specifying the index of the column
4	**getColumnNames()** This method returns the array of all the column names of the table.
5	**getCount()** This method returns the total number of rows in the cursor
6	**getPosition()** This method returns the current position of the cursor in the table
7	**isClosed()** This method returns true if the cursor is closed and return false otherwise

Database - Helper class

For managing all the operations related to the database , an helper class has been given and is called SQLiteOpenHelper. It automatically manages the creation and update of the database. Its syntax is given below

```
public class DBHelper extends SQLiteOpenHelper {
   public DBHelper(){
      super(context,DATABASE_NAME,null,1);
   }
   public void onCreate(SQLiteDatabase db) {}
   public void onUpgrade(SQLiteDatabase database, int
oldVersion, int newVersion) {}
}
```

Example

Here is an example demonstrating the use of SQLite Database. It creates a basic contacts applications that allows insertion, deletion and modification of contacts.

To experiment with this example, you need to run this on an actual device on which camera is supported.

Steps	Description
1	You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.

2	Modify src/MainActivity.java file to get references of all the XML components and populate the contacts on listView.
3	Create new src/DBHelper.java that will manage the database work
4	Create a new Activity as DisplayContact.java that will display the contact on the screen
5	Modify the res/layout/activity_main to add respective XML components
6	Modify the res/layout/activity_display_contact.xml to add respective XML components
7	Modify the res/values/string.xml to add necessary string components
8	Modify the res/menu/display_contact.xml to add necessary menu components
9	Create a new menu as res/menu/mainmenu.xml to add the insert contact option
10	Run the application and choose a running android device and install the application on it and verify the results.

UNIT 3

Enterprise data access via Internet

One of the biggest challenges to delivering enterprise mobile solutions is connecting to enterprise data. Customers and employees need access to their data to be productive. Since mobile solutions mean access from anywhere, how does an enterprise mobile app always have access to the data that makes it useful.

The requirements for a solution should address these points:

1. Threat Protection
2. Edge Authentication
3. Configure (Not Code) Security
4. Single Sign-on User Authentication
5. Device Authentication
6. OAuth Support

UNIT 3

Enterprise data access via Intranet

Applications can access a content provider indirectly with an Intent. The application does not call any of the methods of ContentResolver or ContentProvider. Instead, it sends an intent that starts an activity, which is often part of the provider's own application.

The destination activity is in charge of retrieving and displaying the data in its UI. Depending on the action in the intent, the destination activity may also prompt the user to make modifications to the provider's data.

An intent may also contain "extras" data that the destination activity displays in the UI; the user then has the option of changing this data before using it to modify the data in the provider.

Implementing a content provider involves always the following steps:

1. Create a class that extends ContentProvider
2. Create a contract class
3. Create the UriMatcher definition
4. Implement the onCreate() method
5. Implement the getType() method
6. Implement the CRUD methods

Graphics and animation

Animation in android is possible from many ways. In this chapter we will discuss one easy and widely used way of making animation called tweened animation.

Tween Animation

Tween Animation takes some parameters such as start value , end value, size , time duration , rotation angle e.t.c and perform the required animation on that object. It can be applied to any type of object. So in order to use this , android has provided us a class called Animation.

In order to perform animation in android , we are going to call a static function loadAnimation() of the class AnimationUtils. We are going to receive the result in an instance of Animation Object. Its syntax is as follows −

```
Animation animation = AnimationUtils.loadAnimation(getApplicationContext(),
  R.anim.myanimation);
```

This animation class has many useful functions which are listed below package com.example.sairamkrishna.myapplication;−

Sr.No	Method & Description
1	**start() -**This method starts the animation.
2	**setDuration(long duration)-** This method sets the duration of an animation.
3	**getDuration() -**This method gets the duration which is set by above method
4	**end()-** This method ends the animation.
5	**cancel() -**This method cancels the animation.

```
import android.app.Activity;
import android.os.Bundle;
```

```
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;
import android.widget.Toast;

public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  public void clockwise(View view){
    ImageView image = (ImageView)findViewById(R.id.imageView);
    Animation animation =
AnimationUtils.loadAnimation(getApplicationContext(),
      R.anim.myanimation);
    image.startAnimation(animation);
  }

  public void zoom(View view){
    ImageView image = (ImageView)findViewById(R.id.imageView);
    Animation animation1 =
AnimationUtils.loadAnimation(getApplicationContext(),
      R.anim.clockwise);
    image.startAnimation(animation1);
  }

  public void fade(View view){
    ImageView image = (ImageView)findViewById(R.id.imageView);
    Animation animation1 =
      AnimationUtils.loadAnimation(getApplicationContext(),
        R.anim.fade);
    image.startAnimation(animation1);
  }

  public void blink(View view){
    ImageView image = (ImageView)findViewById(R.id.imageView);
    Animation animation1 =
      AnimationUtils.loadAnimation(getApplicationContext(),
```

```
            R.anim.blink);
        image.startAnimation(animation1);
    }

    public void move(View view){
        ImageView image = (ImageView)findViewById(R.id.imageView);
        Animation animation1 =
            AnimationUtils.loadAnimation(getApplicationContext(), R.anim.move);
        image.startAnimation(animation1);
    }

    public void slide(View view){
        ImageView image = (ImageView)findViewById(R.id.imageView);
        Animation animation1 =
            AnimationUtils.loadAnimation(getApplicationContext(), R.anim.slide);
        image.startAnimation(animation1);
    }
}
```

Custom views, canvas

Android offers a great list of pre-built widgets like Button, TextView, EditText, ListView, CheckBox, RadioButton, Gallery, Spinner, AutoCompleteTextView etc. which you can use directly in your Android application development, but there may be a situation when you are not satisfied with existing functionality of any of the available widgets. Android provides you with means of creating your own custom components which you can customized to suit your needs.

Creating a Simple Custom Component

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *myapplication* under a package *com.example.tutorialspoint7.myapplication* as explained in the *Hello World Example* chapter.
2	Create an XML *res/values/attrs.xml* file to define new attributes along with their data type.
3	Create *src/mainactivity.java* file and add the code to define your custom component
4	Modify *res/layout/activity_main.xml* file and add the code to create Colour compound view instance along with few default attributes and new attributes.
5	Run the application to launch Android emulator and verify the result of the changes done in the application.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <declare-styleable name="TimeView">
    <declare-styleable name="TimeView">
      <attr name="title" format="string" />
      <attr name="setColor" format="boolean"/>
    </declare-styleable>
  </declare-styleable>
</resources>
```

Change the layout file used by the activity to the following.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  xmlns:custom="http://schemas.android.com/apk/res-auto"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity" >

  <com.example.tutorialspoint7.myapplication.TimeView
    android:id="@+id/timeView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#fff"
    android:textSize="40sp"
    custom:title="my time view"
    custom:setColor="true" />

  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/simple"
    android:layout_below="@id/timeView"
    android:layout_marginTop="10dp"
```

Animation APIs

The API allows to define for arbitrary object properties a start and end value and apply a time-based change to this attribute. This API can be applied on any Java object not only on Views.

Animator and AnimatorListener

The superclass of the animation API is the Animator class. Typically the ObjectAnimator class is used to modify the attributes of an object.

The ViewPropertyAnimator class introduced in Android 3.1 provides a simpler access to typical animations which are performed on Views.

The animate() method on a View will return the ViewPropertyAnimator object. This object allows to perform simultaneous animations. It has a fluent API and allows to set the duration of the animation.

The target of ViewPropertyAnimator is to provide a very simple API for typical animations.

he following code shows an example of the usage of this method.

```
// Using hardware layer
myView.animate().translationX(400).withLayer();
```

For performance optimization you can also let ViewPropertyAnimator use a hardware layout.

```
// Using hardware layer
myView.animate().translationX(400).withLayer();
```

You can also directly define a Runnable to be executed at the start and the end of the animation.

```
// StartAction
myView.animate().translationX(100).withStartAction(new Runnable(){
  public void run(){
    viewer.setTranslationX(100-myView.getWidth());
    // Do something
  }
});

// EndAction
myView.animate().alpha(0).withStartAction(new Runnable(){
  public void run(){
    // Remove the view from the layout called parent
    parent.removeView(myView);
  }
});
```

1.4. Layout animations

The LayoutTransition class allows to set animations on a layout container and a changes on the Views of this container will be animated.

```
package com.example.android.layoutanimation;

import android.animation.LayoutTransition;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;

public class MainActivity extends Activity {

  private ViewGroup viewGroup;

  @Override
```

```
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    LayoutTransition l = new LayoutTransition();
    l.enableTransitionType(LayoutTransition.CHANGING);
    viewGroup = (ViewGroup) findViewById(R.id.container);
    viewGroup.setLayoutTransition(l);

  }

  public void onClick(View view) {
    viewGroup.addView(new Button(this));
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.activity_main, menu);
    return true;
  }
}
```

Animations for Activity transition **PAGE TOP**

Animations can be applied to Views but it is also possible to apply them on the transition between activities.

The ActivityOptions class allows to define default or customer animations.

```
public void onClick(View view) {
  Intent intent = new Intent(this, SecondActivity.class);
  ActivityOptions options = ActivityOptions.makeScaleUpAnimation(view, 0,
      0, view.getWidth(), view.getHeight());
  startActivity(intent, options.toBundle());
}
```

Multimedia

Android Multimedia

Android multimedia api deals in playing and controlling the audio and video.

Playing Audio in android Example
Playing Video in android Example

Play and control the audio files in android by the help of **MediaPlayer class**.

MediaPlayer class

The **android.media.MediaPlayer** class is used to control the audio or video files.

Methods of MediaPlayer class

There are many methods of MediaPlayer class. Some of them are as follows:

Method	Description
public void setDataSource(String path)	sets the data source (file path or http url) to use.
public void prepare()	prepares the player for playback synchronously.
public void start()	it starts or resumes the playback.
public void stop()	it stops the playback.
public void pause()	it pauses the playback.
public boolean isPlaying()	checks if media player is playing.
public void seekTo(int millis)	seeks to specified time in miliseconds.

public void setLooping(boolean looping)	sets the player for looping or non-looping.
public boolean isLooping()	checks if the player is looping or non-looping.
public void selectTrack(int index)	it selects a track for the specified index.
public int getCurrentPosition()	returns the current playback position.
public int getDuration()	returns duration of the file.
public void setVolume(float leftVolume,float rightVolume)	sets the volume on this player.

```
package com.example.audiomediaplayer1;


import android.media.MediaPlayer;

import android.net.Uri;

import android.os.Bundle;

import android.app.Activity;

import android.view.Menu;

import android.widget.MediaController;

import android.widget.VideoView;


public class MainActivity extends Activity {


    @Override

    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);


        MediaPlayer mp=new MediaPlayer();
        try{
            mp.setDataSource("/sdcard/Music/maine.mp3");
//Write your location here
            mp.prepare();
            mp.start();


        }catch(Exception e){e.printStackTrace();}


    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
     }
```

Audio/video and images

MediaController and **VideoView** classes, we can play the video files in android

MediaController class

The **android.widget.MediaController** is a view that contains media controls like play/pause, previous, next, fast-forward, rewind etc.

VideoView class

The **android.widget.VideoView** class provides methods to play and control the video player. The commonly used methods of VideoView class are as follows:

Method	Description
public void setMediaController(MediaController controller)	sets the media controller to the video view.
public void setVideoURI (Uri uri)	sets the URI of the video file.
public void start()	starts the video view.
public void stopPlayback()	stops the playback.
public void pause()	pauses the playback.
public void suspend()	suspends the playback.
public void resume()	resumes the playback.
public void seekTo(int millis)	seeks to specified time in miliseconds.

RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android
"

xmlns:tools="http://schemas.android.com/tools"

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <VideoView
        android:id="@+id/videoView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_centerVertical="true" />

</RelativeLayout>


package com.example.video1;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.MediaController;
import android.widget.VideoView;
```

```
public class MainActivity extends Activity {


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


        VideoView videoView =(VideoView)findViewById(R.id.videoView1);


            //Creating MediaController
        MediaController mediaController= new MediaController(this);
            mediaController.setAnchorView(videoView);


            //specify the location of media file
          Uri uri=Uri.parse(Environment.getExternalStorageDirectory().getPath()
+"/media/1.mp4");


            //Setting MediaController and URI, then starting the videoView
          videoView.setMediaController(mediaController);
          videoView.setVideoURI(uri);
          videoView.requestFocus();
          videoView.start();
```

```
    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }

}
```

Playback and record

Android provides many ways to control playback of audio/video files and streams. One of this way is through a class called MediaPlayer.

Android is providing MediaPlayer class to access built-in mediaplayer services like playing audio,video e.t.c. In order to use MediaPlayer, we have to call a static Method create() of this class. This method returns an instance of MediaPlayer class. Its syntax is as follows

MediaPlayer mediaPlayer = MediaPlayer.create(this, R.raw.song);

The second parameter is the name of the song that you want to play. You have to make a new folder under your project with name **raw** and place the music file into it.

Once you have created the Mediaplayer object you can call some methods to start or stop the music. These methods are listed below.

mediaPlayer.start();

mediaPlayer.pause();

On call to **start()** method, the music will start playing from the beginning. If this method is called again after the **pause()** method, the music would start playing from where it is left and not from the beginning.

In order to start music from the beginning, you have to call **reset()** method. Its syntax is given below.

mediaPlayer.reset();

Apart from the start and pause method, there are other methods provided by this class for better dealing with audio/video files. These methods are listed below −

isPlaying()

1 This method just returns true/false indicating the song is playing or not

seekTo(position)

2 This method takes an integer, and move song to that particular second

getCurrentDuration()

3 This method returns the current position of song in milliseconds

getDuration()

4 This method returns the total time duration of song in milliseconds

reset()

5 This method resets the media player

release()

6 This method releases any resource attached with MediaPlayer object

setVolume(float leftVolume, float rightVolume)

7 This method sets the up down volume for this player

setDataSource(FileDescriptor fd)

8 This method sets the data source of audio/video file

selectTrack(int index)

9 This method takes an integer, and select the track from the list on that particular index

getTrackInfo()

10 This method returns an array of track information

MediaRecorder

1. Requesting permission to record audio
2. Creating and running a MediaRecorder
3. Using MediaMuxer to record multiple channels
4. Adding metadata
5. Sample code

The Android multimedia framework includes support for capturing and encoding a variety of common audio and video formats. You can use the MediaRecorder APIs if supported by the device hardware.

This document shows you how to use MediaRecorder to write an application that captures audio from a device microphone, save the audio, and play it back

Requesting permission to record audio

To be able to record, app must tell the user that it will access the device's audio input. <uses-permission android:name="android.permission.RECORD_AUDIO" />

Creating and running a MediaRecorder

Initialize a new instance of MediaRecorder with the following calls:

Set the audio source using setAudioSource().

Note: Most of the audio sources (including DEFAULT) apply processing to the audio signal. To record raw audio select UNPROCESSED. Some devices do not support unprocessed input. Call AudioManager.getProperty("PROPERTY_SUPPORT_AUDIO_SOURCE_UNPROCESSED") first to verify it's available. If it is not, try using

VOICE_RECOGNITION instead, which does not employ AGC or noise suppression.

```
package com.example.sairamkrishna.myapplication;

import android.app.Activity;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;

import android.widget.Button;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;
import java.util.concurrent.TimeUnit;


public class MainActivity extends Activity {
  private Button b1,b2,b3,b4;
  private ImageView iv;
  private MediaPlayer mediaPlayer;

  private double startTime = 0;
  private double finalTime = 0;

  private Handler myHandler = new Handler();;
  private int forwardTime = 5000;
  private int backwardTime = 5000;
  private SeekBar seekbar;
  private TextView tx1,tx2,tx3;

  public static int oneTimeOnly = 0;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    b1 = (Button) findViewById(R.id.button);
    b2 = (Button) findViewById(R.id.button2);
```

```
    b3 = (Button)findViewById(R.id.button3);
    b4 = (Button)findViewById(R.id.button4);
    iv = (ImageView)findViewById(R.id.imageView);

    tx1 = (TextView)findViewById(R.id.textView2);
    tx2 = (TextView)findViewById(R.id.textView3);
    tx3 = (TextView)findViewById(R.id.textView4);
    tx3.setText("Song.mp3");

    mediaPlayer = MediaPlayer.create(this, R.raw.song);
    seekbar = (SeekBar)findViewById(R.id.seekBar);
    seekbar.setClickable(false);
    b2.setEnabled(false);

    b3.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Toast.makeText(getApplicationContext(), "Playing
          sound",Toast.LENGTH_SHORT).show();
        mediaPlayer.start();

        finalTime = mediaPlayer.getDuration();
        startTime = mediaPlayer.getCurrentPosition();

        if (oneTimeOnly == 0) {
          seekbar.setMax((int) finalTime);
          oneTimeOnly = 1;
        }

        tx2.setText(String.format("%d min, %d sec",
          TimeUnit.MILLISECONDS.toMinutes((long) finalTime),
          TimeUnit.MILLISECONDS.toSeconds((long) finalTime) -

TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes((long)
            finalTime)))
        );

        tx1.setText(String.format("%d min, %d sec",
          TimeUnit.MILLISECONDS.toMinutes((long) startTime),
          TimeUnit.MILLISECONDS.toSeconds((long) startTime) -
```

```
TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes((long)
            startTime)))
        );

        seekbar.setProgress((int)startTime);
        myHandler.postDelayed(UpdateSongTime,100);
        b2.setEnabled(true);
        b3.setEnabled(false);
      }
    });

    b2.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Toast.makeText(getApplicationContext(), "Pausing
          sound",Toast.LENGTH_SHORT).show();
        mediaPlayer.pause();
        b2.setEnabled(false);
        b3.setEnabled(true);
      }
    });

    b1.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        int temp = (int)startTime;

        if((temp+forwardTime)<=finalTime){
          startTime = startTime + forwardTime;
          mediaPlayer.seekTo((int) startTime);
          Toast.makeText(getApplicationContext(),"You have Jumped forward
5
            seconds",Toast.LENGTH_SHORT).show();
        }else{
          Toast.makeText(getApplicationContext(),"Cannot jump forward 5
            seconds",Toast.LENGTH_SHORT).show();
        }
      }
    });
```

```
    b4.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        int temp = (int)startTime;

        if((temp-backwardTime)>0){
          startTime = startTime - backwardTime;
          mediaPlayer.seekTo((int) startTime);
          Toast.makeText(getApplicationContext(),"You have Jumped
backward 5
            seconds",Toast.LENGTH_SHORT).show();
        }else{
          Toast.makeText(getApplicationContext(),"Cannot jump backward 5
            seconds",Toast.LENGTH_SHORT).show();
        }
      }
    });
  }

  private Runnable UpdateSongTime = new Runnable() {
    public void run() {
      startTime = mediaPlayer.getCurrentPosition();
      tx1.setText(String.format("%d min, %d sec",
        TimeUnit.MILLISECONDS.toMinutes((long) startTime),
        TimeUnit.MILLISECONDS.toSeconds((long) startTime) -
        TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.
        toMinutes((long) startTime)))
      );
      seekbar.setProgress((int)startTime);
      myHandler.postDelayed(this, 100);
    }
  };
}
```

Following is the modified content of the xml **res/layout/activity_main.xml**.

In the below code **abc**indicates the logo of tutorialspoint.com

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
```

```
   android:layout_height="match_parent"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

   <TextView android:text="Music Palyer"
android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/textview"
      android:textSize="35dp"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true" />

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorials point"
      android:id="@+id/textView"
      android:layout_below="@+id/textview"
      android:layout_centerHorizontal="true"
      android:textColor="#ff7aff24"
      android:textSize="35dp" />

   <ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageView"
      android:layout_below="@+id/textView"
      android:layout_centerHorizontal="true"
      android:src="@drawable/abc"/>

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/forward"
      android:id="@+id/button"
      android:layout_alignParentBottom="true"
      android:layout_alignParentLeft="true"
      android:layout_alignParentStart="true" />
```

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/pause"
  android:id="@+id/button2"
  android:layout_alignParentBottom="true"
  android:layout_alignLeft="@+id/imageView"
  android:layout_alignStart="@+id/imageView" />

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/back"
  android:id="@+id/button3"
  android:layout_alignTop="@+id/button2"
  android:layout_toRightOf="@+id/button2"
  android:layout_toEndOf="@+id/button2" />

<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/rewind"
  android:id="@+id/button4"
  android:layout_alignTop="@+id/button3"
  android:layout_toRightOf="@+id/button3"
  android:layout_toEndOf="@+id/button3" />

<SeekBar
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/seekBar"
  android:layout_alignLeft="@+id/textview"
  android:layout_alignStart="@+id/textview"
  android:layout_alignRight="@+id/textview"
  android:layout_alignEnd="@+id/textview"
  android:layout_above="@+id/button" />

<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
```

```
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:text="Small Text"
    android:id="@+id/textView2"
    android:layout_above="@+id/seekBar"
    android:layout_toLeftOf="@+id/textView"
    android:layout_toStartOf="@+id/textView" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:text="Small Text"
    android:id="@+id/textView3"
    android:layout_above="@+id/seekBar"
    android:layout_alignRight="@+id/button4"
    android:layout_alignEnd="@+id/button4" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Medium Text"
    android:id="@+id/textView4"
    android:layout_alignBaseline="@+id/textView2"
    android:layout_alignBottom="@+id/textView2"
    android:layout_centerHorizontal="true" />
```

LOCATION AWARENESS

Android location APIs make it easy for you to build location-aware applications, without needing to focus on the details of the underlying location technology.

This becomes possible with the help of **Google Play services**, which facilitates adding location awareness to your app with automated location tracking, geofencing, and activity recognition.

This tutorial shows you how to use Location Services in your APP to get the current location, get periodic location updates, look up addresses etc.

The Location Object

The **Location** object represents a geographic location which can consist of a latitude, longitude, time stamp, and other information such as bearing, altitude and velocity. There are following important methods which you can use with Location object to get location specific information −

Sr.No.	Method & Description
1	**float distanceTo(Location dest)** Returns the approximate distance in meters between this location and the given location.
2	**float getAccuracy()** Get the estimated accuracy of this location, in meters.
3	**double getAltitude()** Get the altitude if available, in meters above sea level.
4	**float getBearing()**

	Get the bearing, in degrees.
5	**double getLatitude()** Get the latitude, in degrees.
6	**double getLongitude()** Get the longitude, in degrees.
7	**float getSpeed()** Get the speed if it is available, in meters/second over ground.
8	**boolean hasAccuracy()** True if this location has an accuracy.
9	**boolean hasAltitude()** True if this location has an altitude.
10	**boolean hasBearing()** True if this location has a bearing.
11	**boolean hasSpeed()** True if this location has a speed.
12	**void reset()** Clears the contents of the location.
13	**void setAccuracy(float accuracy)**

	Set the estimated accuracy of this location, meters.
14	**void setAltitude(double altitude)** Set the altitude, in meters above sea level.
15	**void setBearing(float bearing)** Set the bearing, in degrees.
16	**void setLatitude(double latitude)** Set the latitude, in degrees.
17	**void setLongitude(double longitude)** Set the longitude, in degrees.
18	**void setSpeed(float speed)** Set the speed, in meters/second over ground.
19	**String toString()** Returns a string containing a concise, human-readable description of this object.

Get the Current Location

To get the current location, create a location client which is **LocationClient**object, connect it to Location Services using **connect()** method, and then call its **getLastLocation()** method. This method returns the most recent location in the form of **Location** object that contains latitude and longitude coordinates and other information as explained above. To have location based functionality in your activity, you will have to implement two interfaces −

- GooglePlayServicesClient.ConnectionCallbacks
- GooglePlayServicesClient.OnConnectionFailedListener

These interfaces provide following important callback methods, which you need to implement in your activity class −

Sr.No.	Callback Methods & Description
1	**abstract void onConnected(Bundle connectionHint)** This callback method is called when location service is connected to the location client successful use **connect()** method to connect to the location client.
2	**abstract void onDisconnected()** This callback method is called when the client is disconnected. You will use **disconnect()** method the location client.
3	**abstract void onConnectionFailed(ConnectionResult result)** This callback method is called when there was an error connecting the client to the service.

You should create the location client in **onCreate()** method of your activity class, then connect it in **onStart()**, so that Location Services maintains the current location while your activity is fully visible. You should disconnect the client in **onStop()** method, so that when your app is not visible, Location Services is not maintaining the current location. This helps in saving battery power up-to a large extent.

Get the Updated Location

If you are willing to have location updates, then apart from above mentioned interfaces, you will need to implement **LocationListener** interface as well. This interface provide following callback method, which you need to implement in your activity class −

Sr.No.	Callback Method & Description
1	**abstract void onLocationChanged(Location location)** This callback method is used for receiving notifications from the LocationClient when the locatio

Location Quality of Service

The **LocationRequest** object is used to request a quality of service (QoS) for location updates from the **LocationClient**. There are following useful setter methods which you can use to handle QoS. There are equivalent getter methods available which you can check in Android official documentation.

Sr.No.	Method & Description
1	**setExpirationDuration(long millis)** Set the duration of this request, in milliseconds.
2	**setExpirationTime(long millis)** Set the request expiration time, in millisecond since boot.
3	**setFastestInterval(long millis)** Explicitly set the fastest interval for location updates, in milliseconds.
4	**setInterval(long millis)** Set the desired interval for active location updates, in milliseconds.
5	**setNumUpdates(int numUpdates)** Set the number of location updates.
6	**setPriority(int priority)** Set the priority of the request.

Now for example, if your application wants high accuracy location it should create a location request with **setPriority(int)** set to PRIORITY_HIGH_ACCURACY and **setInterval(long)** to 5 seconds. You can also use bigger interval and/or other priorities like PRIORITY_LOW_POWER for to request "city" level accuracy or PRIORITY_BALANCED_POWER_ACCURACY for "block" level accuracy.

Activities should strongly consider removing all location request when entering the background (for example at onPause()), or at least swap the request to a larger interval and lower quality to save power consumption.

Displaying a Location Address

Once you have **Location** object, you can use **Geocoder.getFromLocation()**method to get an address for a given latitude and longitude. This method is synchronous, and may take a long time to do its work, so you should call the method from the **doInBackground()** method of an **AsyncTask** class.

The **AsyncTask** must be subclassed to be used and the subclass will override **doInBackground(Params...)** method to perform a task in the background and **onPostExecute(Result)** method is invoked on the UI thread after the background computation finishes and at the time to display the result. There is one more important method available in AyncTask which is **execute(Params... params)**, this method executes the task with the specified parameters.

Example

Following example shows you in practical how to to use Location Services in your app to get the current location and its equivalent addresses etc.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Create Android Application

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *Tutorialspoint* under a package *com.example.tutorialspoint7.myapplication*.
2	add *src/GPSTracker.java* file and add required code.
3	Modify *src/MainActivity.java* file and add required code as shown below to take care of getting current l equivalent address.
4	Modify layout XML file *res/layout/activity_main.xml* to add all GUI components which include three bu views to show location/address.
5	Modify *res/values/strings.xml* to define required constant values

6	Modify *AndroidManifest.xml* as shown below
7	Run the application to launch Android emulator and verify the result of the changes done in the applicat

Following is the content of the modified main activity file **MainActivity.java**.

```
package com.example.tutorialspoint7.myapplication;

import android.Manifest;
import android.app.Activity;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
import android.test.mock.MockPackageManager;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {

   Button btnShowLocation;
   private static final int REQUEST_CODE_PERMISSION = 2;
   String mPermission = Manifest.permission.ACCESS_FINE_LOCATION;

   // GPSTracker class
   GPSTracker gps;

   @Override
```

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  try {
    if (ActivityCompat.checkSelfPermission(this, mPermission)
      != MockPackageManager.PERMISSION_GRANTED) {

      ActivityCompat.requestPermissions(this, new String[]{mPermission},
        REQUEST_CODE_PERMISSION);

      // If any permission above not allowed by user, this condition will
        execute every time, else your else part will work
    }
  } catch (Exception e) {
    e.printStackTrace();
  }

  btnShowLocation = (Button) findViewById(R.id.button);

  // show location button click event
  btnShowLocation.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View arg0) {
      // create class object
```

```
            gps = new GPSTracker(MainActivity.this);

            // check if GPS enabled
            if(gps.canGetLocation()){

              double latitude = gps.getLatitude();
              double longitude = gps.getLongitude();

              // \n is for new line
              Toast.makeText(getApplicationContext(), "Your Location is - \nLat: "
                + latitude + "\nLong: " + longitude, Toast.LENGTH_LONG).show();
            }else{
              // can't get location
              // GPS or Network is not enabled
              // Ask user to enable GPS/network in settings
              gps.showSettingsAlert();
            }

          }
        });
      }
}
```

Following is the content of the modified main activity file **GPSTracker.java**.

```
package com.example.tutorialspoint7.myapplication;

import android.app.AlertDialog;
```

```
import android.app.Service;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;
import android.provider.Settings;
import android.util.Log;

public class GPSTracker extends Service implements LocationListener {

  private final Context mContext;

  // flag for GPS status
  boolean isGPSEnabled = false;

  // flag for network status
  boolean isNetworkEnabled = false;

  // flag for GPS status
  boolean canGetLocation = false;

  Location location; // location
```

```
    double latitude; // latitude
    double longitude; // longitude

    // The minimum distance to change Updates in meters
    private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 10; // 10 meters

    // The minimum time between updates in milliseconds
    private static final long MIN_TIME_BW_UPDATES = 1000 * 60 * 1; // 1 minute

    // Declaring a Location Manager
    protected LocationManager locationManager;

    public GPSTracker(Context context) {
        this.mContext = context;
        getLocation();
    }

    public Location getLocation() {
        try {
            locationManager = (LocationManager)
mContext.getSystemService(LOCATION_SERVICE);

            // getting GPS status
            isGPSEnabled =
locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);

            // getting network status
```

```
isNetworkEnabled = locationManager
   .isProviderEnabled(LocationManager.NETWORK_PROVIDER);


if (!isGPSEnabled && !isNetworkEnabled) {
   // no network provider is enabled
} else {
   this.canGetLocation = true;
   // First get location from Network Provider
   if (isNetworkEnabled) {
      locationManager.requestLocationUpdates(
         LocationManager.NETWORK_PROVIDER,
         MIN_TIME_BW_UPDATES,
         MIN_DISTANCE_CHANGE_FOR_UPDATES, this);


      Log.d("Network", "Network");
      if (locationManager != null) {
         location = locationManager
            .getLastKnownLocation(LocationManager.NETWORK_PROVIDER);


         if (location != null) {
            latitude = location.getLatitude();
            longitude = location.getLongitude();
         }
      }
   }
```

```
        // if GPS Enabled get lat/long using GPS Services
        if (isGPSEnabled) {
          if (location == null) {
            locationManager.requestLocationUpdates(
              LocationManager.GPS_PROVIDER,
              MIN_TIME_BW_UPDATES,
              MIN_DISTANCE_CHANGE_FOR_UPDATES, this);

            Log.d("GPS Enabled", "GPS Enabled");
            if (locationManager != null) {
              location = locationManager
                .getLastKnownLocation(LocationManager.GPS_PROVIDER);

              if (location != null) {
                latitude = location.getLatitude();
                longitude = location.getLongitude();
              }
            }
          }
        }

    } catch (Exception e) {
      e.printStackTrace();
    }
```

```
    return location;
}

/**
    * Stop using GPS listener
    * Calling this function will stop using GPS in your app
* */

public void stopUsingGPS(){
    if(locationManager != null){
        locationManager.removeUpdates(GPSTracker.this);
    }
}

/**
    * Function to get latitude
* */

public double getLatitude(){
    if(location != null){
        latitude = location.getLatitude();
    }

    // return latitude
    return latitude;
}
```

```
/**
   * Function to get longitude
* */

public double getLongitude(){
   if(location != null){
      longitude = location.getLongitude();
   }

   // return longitude
   return longitude;
}

/**
   * Function to check GPS/wifi enabled
   * @return boolean
* */

public boolean canGetLocation() {
   return this.canGetLocation;
}

/**
   * Function to show settings alert dialog
   * On pressing Settings button will lauch Settings Options
```

```
* */

public void showSettingsAlert(){
  AlertDialog.Builder alertDialog = new AlertDialog.Builder(mContext);

  // Setting Dialog Title
  alertDialog.setTitle("GPS is settings");

  // Setting Dialog Message
  alertDialog.setMessage("GPS is not enabled. Do you want to go to settings menu?");

  // On pressing Settings button
  alertDialog.setPositiveButton("Settings", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog,int which) {
      Intent intent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
      mContext.startActivity(intent);
    }
  });

  // on pressing cancel button
  alertDialog.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int which) {
      dialog.cancel();
    }
  });
```

```
    // Showing Alert Message
    alertDialog.show();
  }

  @Override
  public void onLocationChanged(Location location) {
  }

  @Override
  public void onProviderDisabled(String provider) {
  }

  @Override
  public void onProviderEnabled(String provider) {
  }

  @Override
  public void onStatusChanged(String provider, int status, Bundle extras) {
  }

  @Override
  public IBinder onBind(Intent arg0) {
    return null;
  }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<?xml version = "1.0" encoding = "utf-8"?>
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
   android:layout_width = "fill_parent"
   android:layout_height = "fill_parent"
   android:orientation = "vertical" >


   <Button
      android:id = "@+id/button"
      android:layout_width = "fill_parent"
      android:layout_height = "wrap_content"
      android:text = "getlocation"/>


</LinearLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants −

```
<?xml version = "1.0" encoding = "utf-8"?>
<resources>
   <string name = "app_name">Tutorialspoint</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

```
<?xml version = "1.0" encoding = "utf-8"?>
<manifest xmlns:android = "http://schemas.android.com/apk/res/android"
   package = "com.example.tutorialspoint7.myapplication">
   <uses-permission android:name = "android.permission.ACCESS_FINE_LOCATION" />
   <uses-permission android:name = "android.permission.INTERNET" />
   <application
```

```
    android:allowBackup = "true"
    android:icon = "@mipmap/ic_launcher"
    android:label = "@string/app_name"
    android:supportsRtl = "true"
    android:theme = "@style/AppTheme">

    <activity android:name = ".MainActivity">
      <intent-filter>
        <action android:name = "android.intent.action.MAIN" />

        <category android:name = "android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

Motion sensors

The Android platform provides several sensors that let you monitor the motion of a device.

The sensors' possible architectures vary by sensor type:

- The gravity, linear acceleration, rotation vector, significant motion, step counter, and step detector sensors are either hardware-based or software-based.
- The accelerometer and gyroscope sensors are always hardware-based.

Most Android-powered devices have an accelerometer, and many now include a gyroscope. The availability of the software-based sensors is more variable because they often rely on one or more hardware sensors to derive their data. Depending on the device, these software-based sensors can derive their data either from the accelerometer and magnetometer or from the gyroscope.

Motion sensors are useful for monitoring device movement, such as tilt, shake, rotation, or swing. The movement is usually a reflection of direct user input (for example, a user steering a car in a game or a user controlling a ball in a game), but it can also be a reflection of the physical environment in which the device is sitting (for example, moving with you while you drive your car). In the first case, you are monitoring motion relative to the device's frame of reference or your application's frame of reference; in the second case you are monitoring motion relative to the world's frame of reference. Motion sensors by themselves are not typically used to monitor device position, but they can be used with other sensors, such as the geomagnetic field sensor, to determine a device's position relative to the world's frame of reference (see Position Sensors for more information).

All of the motion sensors return multi-dimensional arrays of sensor values for each SensorEvent. For example, during a single sensor event the accelerometer returns acceleration force data for the three coordinate axes, and the gyroscope returns rate of rotation data for the three coordinate axes. These data values are returned in a float array (values) along with other SensorEvent parameters. Table 1 summarizes the motion sensors that are available on the Android platform.

The rotation vector sensor and the gravity sensor are the most frequently used sensors for motion detection and monitoring. The rotational vector sensor is particularly versatile and can be used for a wide range of motion-related tasks, such as detecting gestures, monitoring angular change, and monitoring relative orientation changes. For example, the rotational vector sensor is ideal if you are developing a game, an augmented reality application, a 2-dimensional or 3-dimensional compass, or a camera stabilization app. In most cases, using these sensors is a better choice than using the accelerometer and geomagnetic field sensor or the orientation sensor.

Android Open Source Project sensors

The Android Open Source Project (AOSP) provides three software-based motion sensors: a gravity sensor, a linear acceleration sensor, and a rotation vector sensor. These sensors were updated in Android 4.0 and now use a device's gyroscope (in addition to other sensors) to improve stability and performance. If you want to try these sensors, you can identify them by using the getVendor() method and the getVersion() method (the vendor is Google LLC; the version number is 3). Identifying these sensors by vendor and version number is necessary because the Android system considers these three sensors to be secondary sensors. For example, if a device manufacturer provides their own gravity sensor, then the AOSP gravity sensor shows up as a secondary gravity sensor. All three of these sensors rely on a gyroscope: if a device does not have a gyroscope, these sensors do not show up and are not available for use.

Use the gravity sensor

The gravity sensor provides a three dimensional vector indicating the direction and magnitude of gravity. Typically, this sensor is used to determine the device's relative orientation in space. The following code shows you how to get an instance of the default gravity sensor:

KOTLINJAVA

```
val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
val sensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY)
```

The units are the same as those used by the acceleration sensor (m/s^2), and the coordinate system is the same as the one used by the acceleration sensor.

Note: When a device is at rest, the output of the gravity sensor should be identical to that of the accelerometer.

Use the linear accelerometer

The linear acceleration sensor provides you with a three-dimensional vector representing acceleration along each device axis, excluding gravity. You can use this value to perform gesture

detection. The value can also serve as input to an inertial navigation system, which uses dead reckoning. The following code shows you how to get an instance of the default linear acceleration sensor:

KOTLINJAVA

```
val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
val sensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION)
```

Conceptually, this sensor provides you with acceleration data according to the following relationship:

```
linear acceleration = acceleration - acceleration due to gravity
```

You typically use this sensor when you want to obtain acceleration data without the influence of gravity. For example, you could use this sensor to see how fast your car is going. The linear acceleration sensor always has an offset, which you need to remove. The simplest way to do this is to build a calibration step into your application. During calibration you can ask the user to set the device on a table, and then read the offsets for all three axes. You can then subtract that offset from the acceleration sensor's direct readings to get the actual linear acceleration.

The sensor coordinate system is the same as the one used by the acceleration sensor, as are the units of measure (m/s^2).

Use the rotation vector sensor

The rotation vector represents the orientation of the device as a combination of an angle and an axis, in which the device has rotated through an angle θ around an axis (x, y, or z). The following code shows you how to get an instance of the default rotation vector sensor:

KOTLINJAVA

```
val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
val sensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR)
```

The three elements of the rotation vector are expressed as follows:

$$x \cdot \sin(\frac{\theta}{2})$$
$$y \cdot \sin(\frac{\theta}{2})$$
$$z \cdot \sin(\frac{\theta}{2})$$

Where the magnitude of the rotation vector is equal to sin(θ/2), and the direction of the rotation vector is equal to the direction of the axis of rotation.

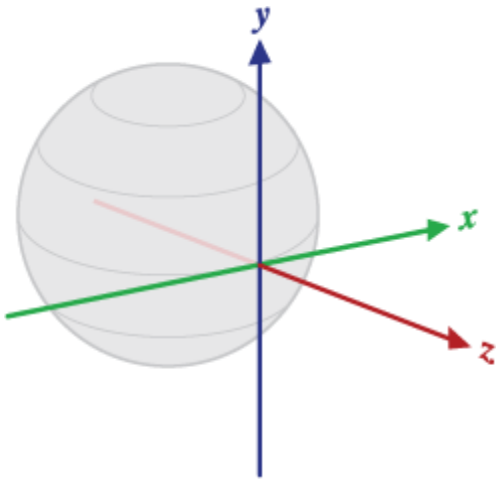


Figure 1. Coordinate system used by the rotation vector sensor.

The three elements of the rotation vector are equal to the last three components of a unit quaternion (cos(θ/2), x*sin(θ/2), y*sin(θ/2), z*sin(θ/2)). Elements of the rotation vector are unitless. The x, y, and z axes are defined in the same way as the acceleration sensor. The reference coordinate system is defined as a direct orthonormal basis (see figure 1). This coordinate system has the following characteristics:

- X is defined as the vector product Y x Z. It is tangential to the ground at the device's current location and points approximately East.
- Y is tangential to the ground at the device's current location and points toward the geomagnetic North Pole.
- Z points toward the sky and is perpendicular to the ground plane.

For a sample application that shows how to use the rotation vector sensor, see RotationVectorDemo.java.

Use the significant motion sensor

The significant motion sensor triggers an event each time significant motion is detected and then it disables itself. A significant motion is a motion that might lead to a change in the user's location; for example walking, biking, or sitting in a moving car. The following code shows you how to get an instance of the default significant motion sensor and how to register an event listener:

KOTLINJAVA

```
val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
val mSensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_SIGNIFICANT_MOTION)
val triggerEventListener = object : TriggerEventListener() {
    override fun onTrigger(event: TriggerEvent?) {
        // Do work
    }
}
mSensor?.also { sensor ->
    sensorManager.requestTriggerSensor(triggerEventListener, sensor)
}
```

For more information, see TriggerEventListener.

Use the step counter sensor

The step counter sensor provides the number of steps taken by the user since the last reboot while the sensor was activated. The step counter has more latency (up to 10 seconds) but more accuracy than the step detector sensor. The following code shows you how to get an instance of the default step counter sensor:

```
val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
val sensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER)
```

To preserve the battery on devices running your app, you should use the JobScheduler class to retrieve the current value from the step counter sensor at a specific interval. Although different types of apps require different sensor-reading intervals, you should make this interval as long as possible unless your app requires real-time data from the sensor.

Use the step detector sensor

The step detector sensor triggers an event each time the user takes a step. The latency is expected to be below 2 seconds. The following code shows you how to get an instance of the default step detector sensor:

KOTLINJAVA

```
val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
val sensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR)
```

Work with raw data

The following sensors provide your app with raw data about the linear and rotational forces being applied to the device. In order to use the values from these sensors effectively, you need to filter out factors from the environment, such as gravity. You might also need to apply a smoothing algorithm to the trend of values to reduce noise.

Use the 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:

KOTLINJAVA

```
val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
val sensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
```

Conceptually, an acceleration sensor determines the acceleration that is applied to a device (A_d) by measuring the forces that are applied to the sensor itself (F_s) using the following relationship:

$$A_D = -(\frac{1}{mass}) \sum F_S$$

However, the force of gravity is always influencing the measured acceleration according to the following relationship:

$$A_D = -g - (\frac{1}{mass}) \sum F_S$$

For this reason, when the device is sitting on a table (and not accelerating), the accelerometer reads a magnitude of g = 9.81 m/s^2. Similarly, when the device is in free fall and therefore rapidly accelerating toward the ground at 9.81 m/s^2, its accelerometer reads a magnitude of g = 0 m/s^2. 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:

KOTLINJAVA

```
override fun onSensorChanged(event: SensorEvent) {
    // 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.

    val alpha: Float = 0.8f

    // 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]
}
```

Note: You can use many different techniques to filter sensor data. The code sample above uses a simple filter constant (alpha) to create a low-pass filter. This filter constant is derived from a time constant (t), which is a rough representation of the latency that the filter adds to the sensor events, and the sensor's event delivery rate (dt). The code sample uses an alpha value of 0.8 for demonstration purposes. If you use this filtering method you may need to choose a different alpha value.

Accelerometers use the standard sensor coordinate system. In practice, this means that the following conditions apply when a device is laying flat on a table in its natural orientation:

- If you push the device on the left side (so it moves to the right), the x acceleration value is positive.
- If you push the device on the bottom (so it moves away from you), the y acceleration value is positive.
- If you push the device toward the sky with an acceleration of A m/s^2, the z acceleration value is equal to A + 9.81, which corresponds to the acceleration of the device (+A m/s^2) minus the force of gravity (-9.81 m/s^2).
- The stationary device will have an acceleration value of +9.81, which corresponds to the acceleration of the device (0 m/s^2 minus the force of gravity, which is -9.81 m/s^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.

The Android SDK provides a sample application that shows how to use the acceleration sensor (Accelerometer Play).

Use the 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:

KOTLINJAVA

```
val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
val sensor: Sensor? = sensorManager.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; that is, an observer looking from some positive location on the x, y or z axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter clockwise. This is the standard mathematical definition of positive rotation and is not the same as the definition for roll that is used by the orientation sensor.

Usually, the output of the gyroscope is integrated over time to calculate a rotation describing the change of angles over the timestep. For example:

KOTLINJAVA

```
// Create a constant to convert nanoseconds to seconds.
private val NS2S = 1.0f / 1000000000.0f
private val deltaRotationVector = FloatArray(4) { 0f }
private var timestamp: Float = 0f

override fun onSensorChanged(event: SensorEvent?) {
    // This timestep's delta rotation to be multiplied by the current rotation
    // after computing it from the gyro sample data.
    if (timestamp != 0f && event != null) {
        val dT = (event.timestamp - timestamp) * NS2S
        // Axis of the rotation sample, not normalized yet.
        var axisX: Float = event.values[0]
        var axisY: Float = event.values[1]
        var axisZ: Float = event.values[2]

        // Calculate the angular speed of the sample
        val omegaMagnitude: Float = sqrt(axisX * axisX + axisY * axisY + axisZ * axisZ)

        // Normalize the rotation vector if it's big enough to get the axis
        // (that is, EPSILON should represent your maximum allowable margin of error)
        if (omegaMagnitude > EPSILON) {
```

```
                axisX /= omegaMagnitude
                axisY /= omegaMagnitude
                axisZ /= omegaMagnitude
            }

            // Integrate around this axis with the angular speed by the timestep
            // in order to get a delta rotation from this sample over the timestep
            // We will convert this axis-angle representation of the delta rotation
            // into a quaternion before turning it into the rotation matrix.
            val thetaOverTwo: Float = omegaMagnitude * dT / 2.0f
            val sinThetaOverTwo: Float = sin(thetaOverTwo)
            val cosThetaOverTwo: Float = cos(thetaOverTwo)
            deltaRotationVector[0] = sinThetaOverTwo * axisX
            deltaRotationVector[1] = sinThetaOverTwo * axisY
            deltaRotationVector[2] = sinThetaOverTwo * axisZ
            deltaRotationVector[3] = cosThetaOverTwo
        }
        timestamp = event?.timestamp?.toFloat() ?: 0f
        val deltaRotationMatrix = FloatArray(9) { 0f }
        SensorManager.getRotationMatrixFromVector(deltaRotationMatrix, deltaRotationVector);
        // User code should concatenate the delta rotation we computed with the current rotation
        // in order to get the updated rotation.
        // rotationCurrent = rotationCurrent * deltaRotationMatrix;
    }
```

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.

Use the uncalibrated gyroscope

The uncalibrated gyroscope is similar to the gyroscope, except that no gyro-drift compensation is applied to the rate of rotation. Factory calibration and temperature compensation are still applied to the rate of rotation. The uncalibrated gyroscope is useful for post-processing and melding orientation data. In general,`gyroscope_event.values[0]` will be close to `uncalibrated_gyroscope_event.values[0] - uncalibrated_gyroscope_event.values[3]`. That is,

```
calibrated_x ~= uncalibrated_x - bias_estimate_x
```

Note: Uncalibrated sensors provide more raw results and may include some bias, but their measurements contain fewer jumps from corrections applied through calibration. Some applications may prefer these uncalibrated results as smoother and more reliable. For instance, if an application is attempting to conduct its own sensor fusion, introducing calibrations can actually distort results.

In addition to the rates of rotation, the uncalibrated gyroscope also provides the estimated drift around each axis. The following code shows you how to get an instance of the default uncalibrated gyroscope:

KOTLINJAVA

```
val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
val sensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE_UNCALIBRATED)
```

UNIT 4

Debugging mobile apps

App Use interface starts with the core building block of android UI (User Interface) Activity. This is required for designing the screen make up.

An android app user interface consists of 4 components:

1. Screen layout
2. UI controls
3. Art work
4. Events

1. Programming
2. Non programming

Programming components of UI mean app components that have to be programmed using java such as an activity or event handling. **Non programming components** of UI typically include resources such as image files, XML files, icon files, screen layout files, and other media elements.

Android app development framework provides API that enables access of non-programming components inside programming components. App contain interface activities based on UI requirements. Apps activity is created by extending the activity class.

The activity of an app, being a central point to the app UI, hosts the event-handling logic and runtime interaction with the non-programming components.

UNIT 4

White Box Testing

White Box Testing is the testing of a software solution's internal coding and infrastructure. It focuses primarily on strengthening security, the flow of inputs and outputs through the application, and improving design and usability. White box testing is also known as Clear Box testing, Open Box testing, Structural testing, Transparent Box testing, Code-Based testing, and Glass Box testing.

White Box Testing is the testing of a software solution's internal coding and infrastructure. It focuses primarily on strengthening security, the flow of inputs and outputs through the application, and improving design and usability. White box testing is also known as Clear Box testing, Open Box testing, Structural testing, Transparent Box testing, Code-Based testing, and Glass Box testing.

What do you verify in White Box Testing?

White box testing involves the testing of the software code for the following:

- Internal security holes
- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object and function on an individual basis

The testing can be done at system, integration and unit levels of software development. One of the basic goals of whitebox testing is to verify a working flow for an application. It involves testing a series of predefined inputs against expected or desired outputs so that when a specific input does not result in the expected output, you have encountered a bug.

How do you perform White Box Testing?

To give you a simplified explanation of white box testing, we have divided it into two basic steps. This is what testers do when testing an application using the white box testing technique:

STEP 1) UNDERSTAND THE SOURCE CODE

Step 2) CREATE TEST CASES AND EXECUTE

White Box Testing Techniques

Statement Coverage - This technique requires every possible statement in the code to be tested at least once during the testing process.

Branch Coverage - This technique checks every possible path (if-else and other conditional loops) of a software application. Tools: An example of a tool that handles branch coverage testing for C, C++ and Java applications is TCAT-PATH

Advantages of White Box Testing

- Code optimization by finding hidden errors.
- White box tests cases can be easily automated.
- Testing is more thorough as all code paths are usually covered.
- Testing can start early in SDLC even if GUI is not available.

Disadvantages of White Box Testing

- White box testing can be quite complex and expensive.
- Developers who usually execute white box test cases detest it. The white box testing by developers is not detailed can lead to production errors.
- White box testing requires professional resources, with a detailed understanding of programming and implementation.
- White-box testing is time-consuming, bigger programming applications take the time to test fully.

UNIT 4

Black Box Testing

A blackbox test, also called a dynamic analysis security test (DAST test), is an invaluable part of any application security toolbox. Blackbox testing is a method for finding vulnerabilities and flaws in applications using the same techniques that hackers and malicious individuals might adopt when trying to breach application security. The blackbox test gets its name from the fact that testers have no access to applications source code or information about its architecture – they are testing blind or "in the dark", as it were. This is in contrast to a white box test, where testers can view the source code and understand the structure of the application.

Blackbox test techniques are helpful for finding certain vulnerabilities such as input/output validation problems, mistakes related to server configuration, and other problem specific to applications. But a blackbox test on its own cannot identify every vulnerability in the application – other forms of testing are required to fully vet an application before it goes live. And while black box testing can help improve application security, it can also be a drag on development timelines if it is not well-managed and easily integrated into the software development lifecycle (SDLC).

For developers who want the ability to quickly and easily perform a blackbox test during application development, CA Veracode provides a SaaS-based black box test service to help improve application security without hindering development timelines.

A blackbox test tool from CA Veracode

CA Veracode is a leading provider of application security solutions for today's software-driven world. Offering a comprehensive suite of solutions and services on a unified platform, CA Veracode helps organizations assess and improve the security of applications so they can confidently innovate with the software they build, buy and assemble.

CA Veracode provides a comprehensive application security solution, combining blackbox test tools with static analysis (white box test) solutions and software composition analysis, as well as web application perimeter monitoring and vendor application security testing.

As a cloud-based security solution, CA Veracode lets you access blackbox test functionality as needed, scaling effortlessly to meet the demands of development deadlines. CA Veracode is cost-efficient, too, requiring no hardware or software investments and no additional staff or security consultants. CA Veracode's blackbox test is supported by a world-class team of security specialists who are constantly updating testing methodologies.

Benefits of CA Veracode's blackbox test solution

When you add CA Veracode's blackbox test services to your testing protocol, you can:

- Quickly identify vulnerabilities by simulating the actions of a malicious attacker.
- Simplify application security testing – CA Veracode enables access to dynamic analysis through an online portal and returns results quickly.
- Prioritize remediation with a Fix-First Analysis that identifies the most urgent and critical flaws as well as the ones that can be fixed most quickly.
- Find hidden username/passwords, SQL strings, ODBC connectors and other sensitive information that hackers can exploit to gain unauthorized access to locations.

UNIT 4

Test automation of mobile apps

In the fast paced software development industry, mobile automation testing has become indispensable. The real value of mobile automation testing is realized when the development team can progress at a rapid pace, without the fear of breaking existing features. Such automation tests free developers to carry out refactoring confidently without fear.

Testing in the world of mobile applications presents its own set of challenges. It is a waste of time for manual test engineering teams to spend a considerable amount of time testing out each feature on so many different devices, mobile operating systems, and versions.

TOOLS OVERVIEW

Some tools available to us for mobile automation testing include:

1. Appium
2. Robotium
3. Selendroid
4. Calabash

APPIUM

Appium is an open source test automation tool for mobile applications. It is used for native app testing, hybrid app testing, and mobile web app testing. It supports most platforms including iOS, Android, Windows, Firefox OS and Mac. It supports running the apps on Android emulators, iOS simulators and real devices with Android, iOS, Windows, and Mac operating systems. This makes Appium a true cross-platform mobile automation testing tool. The underlying philosophy of Appium is that code should be reusable on various platforms. Appium is built on the idea that testing native apps should not involve including an SDK and recompiling your app. It provides standard automation APIs for all platforms. With Appium, you can write tests with any of your favorite development tools using many different programming languages like Javascript, Java, C#, Ruby, Python, PHP. It is

extremely easy to set up and works on the Selenium Web Driver API that specifies a client-server protocol called JSON Wire Protocol.

ROBOTIUM

Robotium is another test automation framework explicitly targeted for Android. It supports testing for both native and hybrid Android apps. It eases the tasks of writing powerful black box UI tests for Android applications. It is similar to Selenium, but is exclusive to Android. It integrates smoothly with Maven, Gradle or Ant to support running tests as a part of Continuous Integration. These tests can be executed on an Android emulator as well as on real Android devices. One aspect to note is that Robotium tests can only run on one device at a time. This means you cannot run one test against a host of devices, which can make scaling difficult. You can develop a Robotium test suite from Android Studio or Eclipse. Robotium was founded and developed by Renas Reda and was released in January 2010. It is currently at version 5.6.3.

SELENDROID

Selendroid is another mobile test automation framework for Android. It can be used to test a native Android app, a hybrid app, or a mobile web app. Here, tests are written in Selenium 2 Client API. It can be used on Android emulators and real devices. A significant benefit of Selendroid is that it can be added as a node into the Selenium Grid for scaling and parallel testing. It is fully compatible with JSON Wire Protocol, and can handle gestures and interact with multiple Android devices at the same time. The current stable version of Selendroid is 0.17.0.

CALABASH

Calabash is a Behavior-Driven Development (BDD) test automation framework. It can be used to create and execute automated acceptance tests both for Android and iOS. It is cross-platform, open source and free. It is executable on mobile devices and consists of libraries that enable test code to interact programmatically with native and hybrid apps. This functionality requires Ruby to be running on the machine. It is developed and maintained by Xamarin and can be run on Xamarin Test Cloud. It also supports Cucumber. It can be used with any Ruby-based test framework. Calabash is currently at version 0.5.4.

UNIT 4

Junit for Android

The following code shows a JUnit test using the JUnit 5 version. This test assumes that the MyClass class exists and has a multiply(int, int) method.

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class MyTests {

   @Test
   public void multiplicationOfZeroIntegersShouldReturnZero() {
      MyClass tester = new MyClass(); // MyClass is tested

      // assert statements
      assertEquals(0, tester.multiply(10, 0), "10 x 0 must be 0");
      assertEquals(0, tester.multiply(0, 10), "0 x 10 must be 0");
      assertEquals(0, tester.multiply(0, 0), "0 x 0 must be 0");
   }
}
```

Junit naming conventions

There are several potential naming conventions for JUnit tests. A widely-used solution for classes is to use the "Test" suffix at the end of test classes names.

As a general rule, a test name should explain what the test does. If that is done correctly, reading the actual implementation can be avoided.

One possible convention is to use the "should" in the test method name. For example, "ordersShouldBeCreated" or "menuShouldGetActive". This gives a hint what should happen if the test method is executed.

Run your test from command line

The org.junit.runner.JUnitCore class provides the runClasses() method. This method allows you to run one or several tests classes. As a return parameter you receive an object of the type org.junit.runner.Result. This object can be used to retrieve information about the tests.

The following class demonstrates how to run the MyClassTest. This class executes your test class and write potential failures to the console.

```
package de.vogella.junit.first;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;
public class MyTestRunner {
  public static void main(String[] args) {
    Result result = JUnitCore.runClasses(MyClassTest.class);
    for (Failure failure : result.getFailures()) {
    System.out.println(failure.toString());
    }
  }
}
```

Robotium

ANDROID WITH ROBOTIUM

Robotium is a test framework created to make it easy to write powerful and robust automatic black-box test cases for Android applications so test developers don't need any further information about the Android app's structure or implemented classes. All they need is the name of the main class and the path that links to it. With the support of Robotium, test case developers can write function, system and acceptance test scenarios, spanning multiple Android activities. This blog post is meant to serve as a mini-tutorial on setting up Robotium as an automated acceptance testing framework for Android applications. The post will summarize all the info related to setting up & writing tests in Robotium.

Robotium officially supports Android 1.6 and up. Robotium has full support for Activities, Dialogs, Toasts, Menus and Context Menus.

Robotium provides the following benefits:

You can develop powerful test cases, with minimal knowledge of the application under test.

The framework handles multiple Android activities automatically.

Minimal time needed to write solid test cases.

Readability of test cases is greatly improved, compared to standard instrumentation tests.

Test cases are more robust due to the run-time binding to GUI components.

Blazing fast test case execution.

Integrates smoothly with Maven or Ant to run tests as part of continuous integration

With Robotium it is possible to run test cases on applications that are pre-installed http://code.google.com/p/robotium/wiki/RobotiumForPreInstalledApps

Robotium can be integrated into continuous integration and can get code coverage for Robotium tests.(http://code.google.com/p/robotium/wiki/QuestionsAndAnswers)

CREATING AND RUNNING TESTCASES:

STEP 1: ROBOTIUM.JAR

STEP 2: CREATE JAVA CLASS

STEP 3: RUN TEST CASE

Eclipse: After all test cases are created right click on the test project Run As >> Run As Android JUnit Test.

ADB:

Use adb to Install the application apk

```
>> adb install ApplicationToTest.apk
```

Use adb to install the test project apk:

```
>> Adb install ExampleTesting.apk
```

UNIT 4

Monkey Talk

MonkeyTalk is a free and open source functional testing tool for iOS and Android mobile apps. The MonkeyTalkplatform consists of two primary components: MonkeyTalk IDE and MonkeyTalk Agents. MonkeyTalk IDE is Eclipse based tool that records, plays, edits, and manages functional test suites for iOS and Android applications running on simulators, emulators, and devices. Money Talk Agents are libraries for iOS and Android that must be linked into applications to be tested. The agents enable applications to record and play MonkeyTalk Commands. Each command performs a user interface action or verification step.

MonkeyTalkPro provides an inbuilt feature to instrument the build, which injects MonkeyTalkAgent to the built.

Otherwise, we can manually integrate the MonkeyTalkAgent by configuring the source code using eclipse or AndroidStudio before deploying the build. Before we start configuring the source code, we need to install "AspectJ" plugin in eclipse. Once the plugin is installed, open our project source code in eclipse.

Steps to instrument the Android build using eclipse as follows:

Step to be followed to create a test in MonkeyTalkIDE as follows:
Step 1: Create new TestProject by selecting (File→ New→ Project→ General→ Project).
Enter the Project name (eg. MyProject) and click finish.
Step 2: Now add Test to the project by right click on the project and select (New→ Script).
Enter the name for the test (MyTest) and select the project to which it needs to be added.
Now open the test file which we created (say "http://MyTest.mt")

Step 3: Now we have to import the instrumented apk which we already created, by selecting menu (App→ Select a .apk =ile to test) or using toolbar as below:

Step 4: After adding apk, Choose any of the connected device or emulators to test.

Once the device is selected the console below will show the message like this:

Step 5: Now install that apk in the selected device as follows:

Step 6: Now launch the app in connected device as follows:

Note: Once again click on connect device, as a result the record and playback button will be enabled to start the test.

Step 7: Now we can start recording by click on record button.

Whatever we work on the test app will be recorded by monkeytalk and generate the test script as follows:

Our test script can be view and edit in 3 ways as follows:

Table view

Monkey talk Script

JavaScript

Step 8: Once the recording is done. Click on Stop button and click Play button to view the playback of the script. (Before clicking on Play button, relaunch the application is recommendable).

Step 9: To do a complete test on our test app, we have to add some verify commands if needed.

Step 10: Test result can be viewed in the console window.

Advantages of Monkeytalk:

Open source Tool

Provide Record and Playback

Support both Android and iOS

Support Cross platform recording

Easy Readable Test script

Support Gestures

UNIT 5

Versioning

Versioning is the creation and management of multiple releases of a product, all of which have the same general function but are improved, upgraded or customised.

To define the version information for your Android app, set values for the version settings in the Gradle build files. These values are then merged into your app's manifest file during the build process.

Two settings are available, and you should always define values for both of them:

Version Code: An integer used as an internal version number. This number is used only to determine whether one version is more recent than another, with higher numbers indicating more recent versions.

Version Name: A string used as the version number shown to users. This setting can be specified as a raw string or as a reference to a string resource.

Semantic Versioning:

Semantic Versioning (referred to, for short, as SemVer), is a versioning system that has been on the rise over the last few years. In this scheme, version numbers and the way they change convey meaning about the underlying code and what has been modified from one version to the next.

SemVer is a 3-component system in the format of x.y.z where:

x stands for a major version

y stands for a minor version

z stands for a patch

So you have Major.Minor.Patch.

How to apply SemVer in Android ?

Based on the following changes we apply this versioning scheme

Major Version: API Changes, Adding new feature, Redesign the App, Increment the MAJOR after the MINOR version achieved the number 9

Minor Version: While adding a functionality, minor changes like color change or icon changes

Patch: Bug fixing

Precedence refers to how versions are compared to each other when ordered. Precedence MUST be calculated by separating the version into major, minor, patch and pre-release identifiers in that order

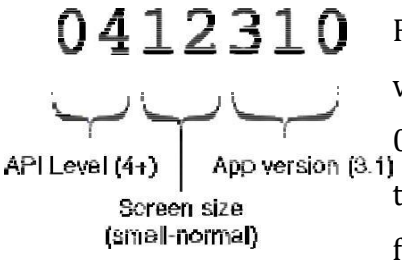
Pre-release:

A pre-release version may be denoted by appending a version classifier that starts with a "-". For example "1.1.0-alpha", "1.2.1-beta", "4.1.3-preview","1.1.8-demo"

When major, minor, and patch are equal, a pre-release version has lower precedence than a normal version. Example: 1.0.0-alpha < 1.0.0.

Using a version code scheme:

In order to allow different APKs to update their version codes independent of others. As per developer guidelines a version code with at least 7 digits.



For example, when the application version name is 3.1.0, version codes for an API level 4 would be something like 0412310.The first two digits are reserved for the API Level (4), the middle two digits are for either screen sizes or GL texture formats (12), and the last three digits are for the application's version name (3.1.0).

Automate the versioning scheme with Gradle:

You can automate the versioning scheme using the following snippet on your build.gradle.

```
apply plugin: 'com.android.application'

ext.majorVersion = 1
ext.minorVersion = 2
ext.patchVersion = 3
ext.preRelease= "DEMO"
 ext.minSdkVersion = 14

android {
   compileSdkVersion 23
   buildToolsVersion "23.0.2"
    defaultConfig {
      applicationId "droidmentor.sample"
      targetSdkVersion 23
      minSdkVersion project.ext.minSdkVersion versionCode
      generateVersionCode() // 140010203 versionName
      generateVersionName() // 1.2.3-DEMO
   }
}
 private Integer generateVersionCode() {
   return ext.minSdkVersion * 10000000 + ext.majorVersion * 10000 + ext.minorVersion *
100 + ext.patchVersion
}
 private String generateVersionName() {
   String versionName = "${ext.majorVersion}.${ext.minorVersion}.${ext.patchVersion}"
   if (ext.preRelease != null &amp;amp;&amp;amp; !ext.preRelease.isEmpty()) {
      versionName = versionName + "-" + ext.preRelease
   }
   return versionName;
}
```

UNIT 5

Packaging mobile apps

Once you are done building the mobile application, you can package the application targeting some common mobile platforms. ColdFusion builder packaged applications can access the native capabilities of the mobile platform. When you are building the applications, you will be writing only CFML code and not any device-specific native code.

Ideally, you will be using ColdFusion Builder for building the mobile application and ColdFusion Server for translating the ColdFusion code in your application to corresponding HTML/JavaScript code that can be packaged and installed on the device. ColdFusion builder gets this translation done through ColdFusion Server seamlessly with a few easy configuration settings. ColdFusion Builder helps in creating platform-specific installers (.apk and .ipa) by invoking the Cordova build service. However, you do not need to package the application targeting individual platforms if you are not using any hardware or device-specific functionalities.

See Types of mobile applications for all the supported types while building ColdFusion-based mobile applications. Information available in this chapter is only applicable for Type 1 and Type 3 deployments.

Supported mobile platforms

The ColdFusion builder currently supports packaging applications for the following mobile platforms:

Android 4.x or higher

iOS 6.0 or higher

Packaging applications using ColdFusion Builder

After creating your mobile application in ColdFusion Builder, you can generate a platform-specific package that can be installed on the mobile device (iOS and Android). ColdFusion

Builder sends the ColdFusion (.cfm) files to the ColdFusion Server, which converts the .cfm files to .html and .js files.

Global configuration requirements

The following sections describe the global configurations required to prepare ColdFusion Builder for creating platform-specific mobile applications. Step 1 – Get the required certificates

The ColdFusion Builder supports creating platform-specific builds for Android and iOS platforms. In order to package the mobile applications for these platforms, you need to configure the ColdFusion Builder to sign the applications with an appropriate developer/self-signed certificate. In the case of Android, providing the certificate details is optional as you can create an Android Application Package (APK) file for testing on your devices without signing it. However, testing the mobile application on iOS devices require you to have a developer certificate and a provisioning profile file.

Ensure that you follow the steps provided in this article to get started:

For iOS development

Create and download development provisioning profiles. Note that you need to first join the iOS developer program to generate developer certificate for testing your mobile applications.

For Android development

Create the keystore file for signing applications.

Step 2 – Provide the server and authentication details
Project-specific configuration requirements

The following sections describe the project-specific configurations required to prepare ColdFusion Builder for creating platform-specific mobile applications.

Step 1 Configuring the mobile project properties

If you have already created a ColdFusion Builder Mobile project (see Building Mobile Applications), right-click the project in the Navigator panel and click Properties.

Select the ColdFusion Mobile Project in the left pane to see the available properties for configuration.

Note: For packaging Server CFCs, go to the Miscellaneous tab and provide the application base URL.

Keep your CFM files and other supporting assets in a separate directory under the web root directory so that you can select just that directory.

Important: All the selected files must be present under the Server's web root directory or under web root's sub-directories. Also, it is mandatory to have an index.cfm file in your application.

UNIT 5

Distributing apps on mobile market place

Distributing through an app marketplace. Usually, to reach the broadest possible audience, you'd distribute your apps through a marketplace, such as Google Play. Google Play is the premier marketplace for Android apps and is particularly useful if you want to distribute your apps to a large global audience.

Distributing through an app marketplace

Usually, to reach the broadest possible audience, you'd distribute your apps through a marketplace, such as Google Play.

Google Play is the premier marketplace for Android apps and is particularly useful if you want to distribute your apps to a large global audience. However, you can distribute your apps through any app marketplace you want or use multiple marketplaces.

Unlike other forms of distribution, Google Play allows you to use the In-app Billing service and licensing service. The In-app Billing service makes it easy to sell in-app products like game jewels or app feature upgrades. The Licensing service helps prevent unauthorized installation and use of your apps.

Distributing your apps by email

A quick and easy way to release your apps is to send them to users by email. To do this, you prepare the app for release, attach it to an email, and send it to a user. When the user opens your email on their Android-powered device, the Android system recognizes the APK and displays an Install Now button in the email message. Users can install your app by touching the button. Users need to opt in for installing unknown apps if they haven't already to proceed with the installation.

Distributing apps through email is convenient if you're sending them to a few trusted users, as it provides few protections from piracy and unauthorized distribution; that is, anyone you send your apps to can simply forward them to others.

Distributing through a website

If you don't want to release your apps on a marketplace such as Google Play, you can make them available for download on your website or server, including on a private or enterprise server. To do this, first prepare your apps for release in the normal way, then host the release-ready APK files on your website and provide users with a download link. To install an app distributed in this way, users must opt-in for installing unknown apps.

User opt-in for installing unknown apps

Android protects users from inadvertent download and install of unknown apps, or apps from sources other than Google Play, which is trusted. Android blocks such installs until the user opts into allowing the installation of apps from other sources. The opt-in process depends on the version of Android running on the user's device: