

Unit 1

Introductory Concepts

1.1 Introduction

Do you spend ample amount of time sitting in front of the television? Do you read the newspaper in the morning? Do you have any book on your bedside table?

If your answer to all the questions asked above is 'NO', you are definitely an 'Androidian'. If your answer is 'YES', this unit could be a turning point in your living style!

'Staying connected' is the catchphrase of the twenty first century. Owning a cell phone is no longer a luxury but has become a necessity. Apples and blackberries are no longer just fruits. 'Google' is the engine of your daily life. What more? The wonderful memories of your life; the time spent with your family and friends can be captured instantaneously, stored and shared with your loved ones 24X7. Needless to say, photography has emerged as a hidden talent in every one. More than 300 million users globally are using social networking sites regularly and making friends is just a click away.

Interestingly the rationale of the cell phone has shifted from a verbal communication tool to a multimedia tool, often adopting the name "smart phone" rather than being called just a phone.

Today's young generation is exposed to technology immediately after their birth. With increasing demand, competition among the manufacturers, sellers, advertisers, etc. are also increasing rapidly. Demands of consumers for variety and improvement of existing applications is also burgeoning.

There are numerous operating systems of mobile devices such as, Apple's iOS, BlackBerry's OS, Nokia's Symbian, Hewlett-Packard's web OS (formerly Palm OS) and Microsoft's Windows etc. but Google's Android is the world's most widely used smart phone operating system with largest market share, dominating the web market for last several years.

So, what is Android? Why is it so popular? How to become an Android developer? These questions and some more questions will be answered in the succeeding sections.



Outcomes

Upon completion of this unit you will be able to:

- *Understand* the Android.
- *Know* the history of Android.
- *Grab* the idea of Android Architecture.
- *Install* the Android Studio and Java.
- *Recognize* the requirement of Android development.
- *Deal* with backward compatibility.



Terminology

Linux:	Linux is an open source operating system (OS) that is a freely distributable.
Open-Source:	Any software whose source code is made available by the copyright holder to provide the authority to study, changes, and distribute the software to anyone for any purpose.
Kernel:	A kernel is a computer program that makes up the central core of the OS. It is the first program that loads in the computer's memory on start up. It is responsible for managing the computer's hardware.
Android API:	An API is a collection of different classes and packages and an API Level is a number given to the every latest version of Android.
Android SDK:	The Android Software Development Kit (SDK) is a set of development libraries and tools used to develop an Android application.
IDE:	An integrated development environment (IDE) is a software application that provides complete convenience to programmers for software development. Generally, an IDE consists of a source code editor, build tools, debugger and intelligent code completion tools.
Android Studio:	It is an IDE provided by the Google to develop the Android application. It is preconfigured for Android development.

1.2 What is Android

Android is an open-source operating system for mobile devices such as smart phones, smart watches, tablets, and other Android enabled platforms such as Android TV and Android Auto. It is a Linux based operating system.

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

-By Andy Rubin (Founder of Android Inc.)

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



Activity

In case, you are using an Android device, check the Android version, Kernel version of your device.

Answer: -----

Tip: To check the version and kernel of your Android device, go to the settings of your device and select the “About Phone” option.

1.3 History of Android

In Oct, 2003, four computer experts (Andy Rubin, Nick Sears, Rich Miner and Chris White) founded a software development organisation Android Inc. in Palo Alto, California, USA. They wanted to make a Linux based operating system that can work on digital cameras which can be connected with computers. But this plan wasn't as successful as they thought, so they turned towards smart phones.

In Aug, 2005, Google purchased the Android Inc. at a very huge unrevealed amount and became the proprietor of the company. In Nov, 2007, Google disclosed a consortium of different mobile technology providers named Open Handset Alliance (OHA) that includes mobile hardware manufacturers (HTC, Motorola etc.), chipset manufacturers (Qualcomm, Texas Instruments etc.), and telecommunication service providers (T-Mobile etc.). There were 34 different companies in OHA consortium that agreed to provide such a mobile device which does not belongs to single company as iPhone from Apple. But for couple of years Google could not bring any mobile under the OHA consortium. In Oct, 2008, HTC brought first smart phone “HTC Dream” in the market which was commercially available. At the time when the first version of the android was unveiled, only 35 Android apps were accessible. But today, more than 1.5 million Android applications are available in the market.

Android has been released in many versions since its inception. Before commercialisation, many internal alpha versions were released on the name of fictional robots (like Astro Boy, Bender, R2-D2 etc.). On Nov 5, 2007 Google released first beta version of Android whose Software

Development Kit (SDK) was released on Nov 12, 2007. Since then, Nov 5th is considered as Android's Birthday.

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

API level is mainly the Android version used as an alternative to the Android version name (e.g. 3.0, 4.0, 4.4, etc.) where we apply integer numbers. This number keeps on increasing with each version, for e.g. Android 1.5 is API Level 3; Android 1.6 is API Level 4, and so on. "Figure 1.1" is providing the details of evolution of Android with product name, version name, release date and API level.



Figure 1.1: Android Version Evolution
(CC BY 4.0, Graphic Era Hill University, Dehradun, India. 2016)

With their main features, different versions of Android are given in the following Table 1.1:

Table 1.1: Version history of Android

SN	Version Name	Features
----	--------------	----------

Visual Programming

1.	API Level1	This was the first commercial version of Android implemented on the mobile device HTC Dream. There were many features in that device like Android Market, Web Browser, Digital Camera, Gmail, Google Maps, Google Search, Google Talk, Voice Dialler, Google Contacts, Google Calendar, Media Player, Wi-Fi and Bluetooth support etc.
1.	API Level2	This version was internally known as “Petite Four”. This version resolves many bugs of previous version and added additional features like save attachment in messages, show and hide dial pad etc.
2.	Cupcake	This was the first version whose code name was on the name of a bakery product. Cupcake was based on Linux Kernel 2.6.27. It had the features like third party virtual keyboard, screen Widgets, copy and paste in the browser, autorotation, upload facility on YouTube and Picasa, auto pairing for Bluetooth, video recording and playback in 3GP and MPEG-4 formats, etc.
3.	Donut	This version was based on Linux kernel 2.6.29. Donut was having the capability of speech and gesture support, selecting the multiple photos for deletion, support for WVGA screen resolution, etc.
4.	Éclair	First version of Eclair (API Level 5) was having features like Microsoft Exchange email support, Bluetooth 2.1, HTML5, Google Map 3.1.2, Live wallpapers, optimized hardware speed, support for more resolutions, double tap zoom, camera features like flash support, digital zoom, white balance, colour effect, etc. Second and third versions of Eclair were Android 2.0.1 (API Level 6) and Android 2.1 (API Level 7), which were released on Dec 3, 2009 and Jan 12, 2010 respectively, with some technological advancement and bug fixes in previous API.
5.	Froyo	It was based on Linux kernel 2.6.32. It was having the features like JIT compilation, Android Cloud to Device Messaging (C2DM), push notification, USB tethering and Wi-Fi hotspot, support for alphanumeric passwords, installing apps in external memory, Adobe Flash support, etc. It is also known as “Frozen Yogurt”.
6.	Gingerbread	It was based on Linux kernel 2.6.35. First version of Gingerbread (API Level 9) was having updated user interface, support for WXGA resolution, NFC and native code development; new download manager, concurrent garbage collection, native support for new sensors like Gyroscope and Barometer, etc. Second version of Gingerbread was Android 2.3.3–2.3.7 (API Level 10) which was released on Feb 9, 2011

		with some bug fixes and advancement like support for voice and video chat using Google Talk, etc.
7.	Honey-comb	It was the first tablet oriented Android update based on Linux kernel 2.6.36. “Motorola Xoom” tablet was the first device to run this update. This version was having the features like holographic interface, System Bar, Action Bar, soft navigation button at the bottom of the screen, two pane contact and email UI, support for multi-core processors, encryption of all user data, etc. Second and third versions of Honeycomb were Android 3.1 (API Level 12) and Android 3.2 (API Level 13), which were released on May 10, 2011 and Jul 15, 2011 respectively with some bug fixes and enhancements.
8.	Ice Cream Sandwich	It was based on Linux kernel 3.0.1. It was the last version that was supporting Adobe Flash Player. First version of Ice Cream Sandwich (API Level 14) was having the features like accessing app from lock screen, real time speech to text dictation, face unlock, built-in photo editor, Wi-Fi Direct, shut down app by swipe from recent menu, integrated screenshot capture, etc. Second version of Ice Cream Sandwich was Android 4.0.3–4.0.4 (API Level 15) which was released on Dec 16, 2011 with some bug fixes and improvements.
9.	Jelly Bean	First version of Jelly Bean (API Level 16) was based on Linux kernel 3.0.31. It was having “Buttery Smooth” UI and other advancements. Second version of Jelly Bean was Android 4.2 – 4.2.2 (API Level 17). It was based on Linux kernel 3.4.0 and released on Nov 13, 2012 with some features like Group Messaging etc. Third version of Jelly Bean was Android 4.3 – 4.3.1(API Level 18) which was released on Jul 24, 2013 with some features like 4K resolution support, native emoji support, Dial pad auto complete, etc.
10.	Kitkat	First version of Kitkat was Android 4.4 - 4.4.4(API Level 19). This was based on Linux kernel 3.10. This was optimized for larger range of devices than previous versions. Recommended RAM for Kitkat is 512 MB but it can run on minimum 340 MB of RAM. It was having different advanced features such as public API for developing text messaging clients; disable access to battery by third party, very elegant UI and much more. Second version of Kitkat was Android 4.4W (API Level 20) which was designed for wearable extensions like smart watch.
11.	Lollipop	First version of Lollipop was Android 5.0-5.0.2 (API Level 21). It was based on Linux kernel 3.16.1 and built around material design under project Volta to

Visual Programming

		improve the battery life. It supports 64 bit CPU, trace based JIT compilation, refreshed lock screen and notification tray; third party apps can modify the external storage; recently used apps remembered after restarting the device; audio I/O through USB, smart lock features and HD voice calls gives it an appeal. Second version of Lollipop was Android 5.1 (API Level 22) which was released on Mar 9, 2015 with official support for multiple SIM cards, high definition voice calls, replicate the silent mode which was removed in API Level 21, native Wi-Fi calling etc.
12.	Marsh-mallow	Marshmallow is based on Linux kernel 3.18.10. It is released under the code name Android M. This is the latest updated version of Android which is having the features like native figure print reader, App standby feature, Doze mode, Now on Tap feature, USB Type-C support, MIDI support, 184 new emoji, etc.
13.	Android N	Upcoming version of android will come under the Project N whose Developer's Preview has been released.



Activity

Complete the crossword with Android version name using the following distinct features of different versions of Android.

Hint:

Across:

1. First version whose name was on the name of a bakery product.
2. First version having native support for new sensors like gyroscope and barometer.
3. First version having the capability of speech and gesture support.
4. It was the last version that supports Adobe Flash Player.
5. This is the Android version 4.1 to 4.3.1.
6. First version to build around material design.
7. First version that is having USB tethering and Wi-Fi hotspot.

1.4 Android Architecture

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

As you can see in the figure 1.2 that Android Architecture (also called Software Stack) has the following layers:

1. Linux Kernel
2. Hardware Abstraction Layer (HAL)
3. Native Libraries
4. Android Run Time
5. Android Application Framework
6. Application Layer

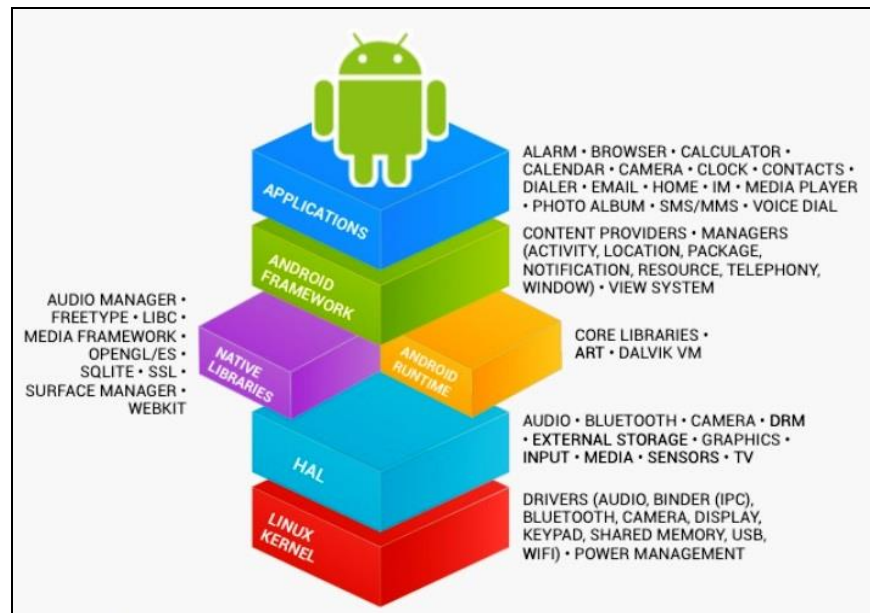


Figure 1.2: Android System Architecture
(Source: <https://source.android.com/source/index.html>)

Let us understand each layer one by one:

1. **Linux Kernel:** Android is design in the top of Linux Kernel which is open source. Being an open source is the best part of Linux. Basic services like process management, memory management, security management, power management, and providing hardware driver for different devices (like Bluetooth, WI-FI, Camera etc.) are managed by Linux Kernel. Latest Version of Android operating system is Marshmallow which is based on Linux kernel version 3.18.10.
2. **Hardware Abstraction Layer (HAL):** The Hardware Abstraction Layer (HAL) provides an interface for hardware vendors to define and implement the drivers for specific hardware without affecting lower level features.
3. **Native Libraries:** Native libraries run over the HAL and it consist various C / C++ library like libc. It also includes following standard libraries:
 - a. *Secure Sockets Layer (SSL):* It is responsible for Internet security.

Visual Programming

- b. *Graphics Library*: OpenGL and SGL used to create 2D and 3D graphics.
 - c. *WebKit*: It is open source web browser engine that gives the functionality to render the web content.
 - d. *SQLite*: This open source RDBMS which is designed to be embedded in Android devices.
 - e. *Media Library*: These libraries are used to play the audio/video media. etc.
- 4. Android Run Time:** It includes DVM, ART and Core Libraries which help your apps to run on an Android mobile device.
- a. *DVM (Dalvik Virtual Machine)*: It is a modified Java virtual machine (JVM) which is introduced for low end devices to run application objects efficiently. It gives the power to a device to become an Android device. It is a register based virtual machine that is optimized to run multiple objects efficiently. It depends on the Linux kernel for efficiently execute the instances because memory management and thread management is part of the Linux kernel. DVM executes the Dalvik Executable Code (.dex), which is optimized to take least memory and processing resources.
 - b. *ART (Android Run Time)*: ART is the successor of Dalvik Virtual machine. ART is the managed runtime system that helps to run applications and system services. ART and its predecessor Dalvik were originally created specifically for the Android project. ART is compatible with DVM, so it helps to run Dalvik Executable codes. This feature is introduced in Android 4.4. Currently it is available on some new Android devices, but there are some techniques that do not work on ART. ART has the following features:
 - i. Ahead-of-time (AOT) compilation
 - ii. Improved garbage collection
 - iii. Improvements in development and debugging
 - c. *Core Libraries*: Core libraries are the collection of Android specific core java library rather than Java ME and Java SE libraries. However, most of the features of core libraries are similar to Java SE library.
- 5. Android Application Framework:** Application framework contains the classes used to create an Android application. This behaves as an abstraction layer for hardware access. It also manages application resources and user interface. Content provider, activity manager, fragment manager, telephony manager, location manager, package manager, notification manager and view system are the parts of Android Application Framework.
- 6. Application Layer:** Application is the top layer of Android architecture. Every application (like Contacts, Browsers, etc.), whether it is native application or third party application, is run in Application layer. Preinstalled applications provided by the vendors

are called native apps and applications developed by another developer are called third party applications. In application layer, third party apps can replace the native apps. This is the beauty of Android.

1.5 Why develop for Android?

Why you should not develop for the Android? Today, mobile has fundamentally changed the way of people interact. You all also know that today more than 50 % mobile market is acquired by Android operating system. Android was launched with 34 applications only, but now more than 2 million applications are available on the Play Store and other Android market places. It is the future of handheld devices, Television and Auto. More than this, developers are also thinking to embed the Android in home appliances and other devices of human endeavour. As far as concern about Android development, following are some thinkable points that justify this question:

1. There is no need of any certificate to become an Android developer. Only you need basic knowledge of Java, XML (Extensible Mark-up Language; designed to store and carrying data) and SQL (Structure Query Language; used to communicate with database) programming to become an Android developer.
2. Google provides one window solution, as Play Store, to upload and download the application either free or with minimal charges.
3. For uploading and distributing the app, developers have no need of any approval of someone.
4. Developer is the owner of his / her app and has the total control on product.

Android has open source operating system, open source SDK and excellent documentation. So, why you should not develop for the Android?

Also, today more than one and half million Android devices are activating daily and since its inception Android became the multi-billion dollar industry. So again, why you should not develop for the Android?

1.6 Dealing with backward compatibility

When new version of Android comes into the market, older versions do not become obsolete so early. So, it becomes necessary to support old versions to target large number devices. You can take an idea of distribution of different Android versions from following distribution table (Figure 1.12.1) and pie chart (Figure 1.12.2):

Visual Programming

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.2%
4.1.x	Jelly Bean	16	7.8%
4.2.x		17	10.5%
4.3		18	3.0%
4.4	KitKat	19	33.4%
5.0	Lollipop	21	16.4%
5.1		22	19.4%
6.0	Marshmallow	23	4.6%

Figure 1.12.1 Android Version Distribution Table

(Source: <http://developer.android.com/about/dashboards/index.html>)

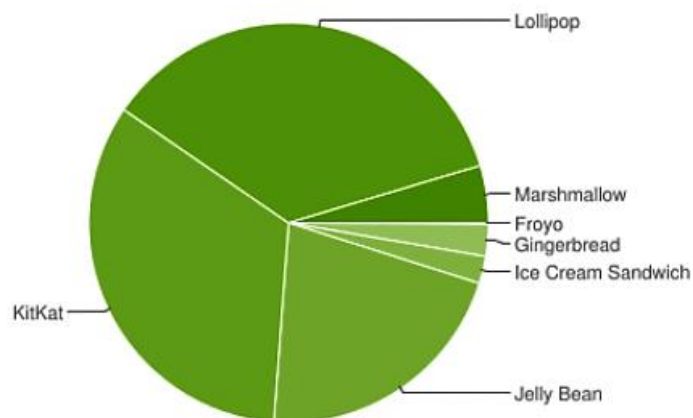


Figure 1.12.2 Android Version Distribution Pie-chart

(Source: <http://developer.android.com/about/dashboards/index.html>)

Note: This data is collected during a 7-day period ending on April 4, 2016. Any versions with less than 0.1% distribution are not shown.



Reading

For more Android statistics follow the following link:

<http://developer.android.com/about/dashboards/index.html>

To make your application is backward compatible to previous versions with best features and functionality; you must use Android Support Library in your applications. To install the Android support library to Android SDK follow the below mentioned steps:

1. Open the “SDK Manager” (Figure 1.11) and select “Support Libraries” from “Extras” section.
 2. Now click on the “Install packages...” button.
-



Reading

To understand all features of Android Support Library follow the following link:

<http://developer.android.com/tools/support-library/index.html>

Unit summary



Summary

In this unit you learned about the Android and its capabilities. You also explored the history of Android, different API levels of Android, Architecture of Android. You have also learned about setting up the Android development environment by installing the JDK, Eclipse, and ADT Bundle. In the last, you have learned about downloading and installing the Android SDK, support library and build tools using Android SDK Manager.

Assignment



Assignment

Q1. What is Android?

Answer: Android is an open source Linux based operating system which is a product of Google Inc.(after acquiring from Android Inc.). Initially it was developed for touch screen mobile phones, but now it is also available for tablets, smart watch and Android Auto too. It is written in C, C++ and Java. For developing the Android applications Android SDK is available with all supportive tools. You can also state that, android is a system that includes an open source operating system, an open source development platform and devices that run the operating system and applications created for it. It has become the multi-billion dollar industry since its inception, so there is lots of space for earning for developers and users.

Q2. Explain the role of Linux Kernel in Android.

Answer: Linux Kernel is the foundation component of Android platform. It is there to handle the hardware; means it helps the software part of the Android System to interact with the hardware. Because all hardware drivers (display driver, keypad driver, camera driver, Wi-Fi driver, Bluetooth driver, etc.) are inbuilt in the kernel, the android runtime does

not need to worry about the hardware handling. It is the lowest layer of Android architecture and it serves as the abstraction layer to other layers

Q3. Describe the role of Dalvik Virtual Machine and Android Runtime.

Answer:Dalvik Virtual Machine (DVM) is a register based virtual machine that is used by Android system to run the Dalvik executable code (.dex file) which is a compiled code of Android. It used in Android System similarly as JVM works for Java to execute the byte code (.class file). DVM doesn't work with .class files. One thing that you must know is that implicitly .dex file is generated by the .class file after highly optimizing the .class file for low memory and least processing power. It gives the power to a device to become an Android device.

Android Runtime (ART) is the successor of Dalvik Virtual Machine (DVM). It has some advantage over DVM including ahead-of-time compilation (AOT), improved garbage collection and other development and debugging enhancements.

Q4. Enlist the different features of Android operating system.

Answer:Android has many features that make it different from other platforms. Those features are as follows:

- It is an open source platform, so you don't need any license or permission.
- It has an open marketplace Play Store for distributing your applications.
- Google play provides free and upfront purchase of your applications.
- There is no need of any special approval for app distribution.
- You don't need any special certificate to become an Android developer. Only you need basic knowledge of Java and little knowledge of XML and SQL.
- Developer is the sole proprietor of his / her application and has full control on the app.

Assessment



Assessment

Problem 1. Why should you be the developer of Android?

Problem 2. Describe the role Android Support Library.

Problem 3. What are the similarities and differences in Eclipse and Android Studio IDEs? Explain.

Problem 4. Analyse the both IDEs (Eclipse and Android Studio). After analysing, which IDE will you prefer to choose for Android application development? Justify your answer.

Problem 5. What are the different native libraries in android architecture?

Unit 2

First application and development environment

2.1 Introduction

In previous chapter, you have learnt about the Android history and Android SDK installation. In this chapter you will learn about the basic constructs of the Android application as well as the steps to create and run an Android project. For creating an Android application you need some basic knowledge of Java and XML programming languages. You need Java programming for creating the logic part and XML programming for creating the design part (User Interface). However, you can code both logic and design part in Java, but it becomes very easy to deal with logic and design separately. You can also program in native languages C and C++ using NDK (Native Development Kit). You will code in Java programming and can use either Eclipse or Android Studio as an IDE (Integrated Development Environment). In this chapter you will also learn about creating an AVD (Android Virtual Device) and running your application on the created AVD.

Upon completion of this unit you will be able to:



Outcomes

- Create First Android application.
- Configure the AVD.
- Save Launch configurations.
- Debug Android project.
- Run Android application.
- Understand tools used in Android application development.
- Have Knowledge of different files created in the development process.
- Add Permissions to the Manifest file.



Terminology

- Activity:** An Activity is a user interface provided as a screen to interact with the application.
- Resource:** Resources are the supplementary files and contents that your program uses, such as bitmap images, layouts, UI strings, animation, etc.
- Prospective:** Every working window of an IDE contains at least one perspective. It contains a set of editors, and other windows, those have a set of functionality to accomplish a specific task.

Visual Programming

Package:	A package is a method of organizing the classes into a single name (or folder) that provides the concept of modular programming. It can contain distinct class files or JAR files (Java Archive File that contains multiple compressed class files under a single name).
Debugging:	It is the procedure of finding the defects in a source code and removing them.
Workspace:	Workspace is a folder created by the Eclipse or Android Studio where all projects and projects related data get stored.
Graphical Layout:	Graphical layout is an arrangement of visual elements (e.g. Buttons, Labels, and Text Fields etc.) on a page or a screen.
APK File:	Android application package (APK) is the file format that packages the .dex file and different resource files. It is used by the Android OS for installation an application in the Android devices.

2.2 Creating first Android application

Now, after setting up the development environment, this is the time to move towards the Android application creation. Following sections will enable you to create a new Android project in both IDEs (Eclipse and Android Studio):

2.2.1 Creating a new Android project in Android Studio

First thing that you must know is that in Android studio every project has its own separate instance i.e. each project has its own window of Android studio. So when you start Android Studio first time, it wouldn't allow you to enter without creating a new project (shown in Figure 1.10 of section 1.6.3 of UNIT 1). When you click on the "Start a new Android Studio project" option, you will be switched to the "New project" window as shown in Figure 2.5.

If you are creating a new project from while working on another project, then the steps are as follows:

1. Select the "New Project..." from "New" submenu of "File" menu as shown in Figure 2.4.

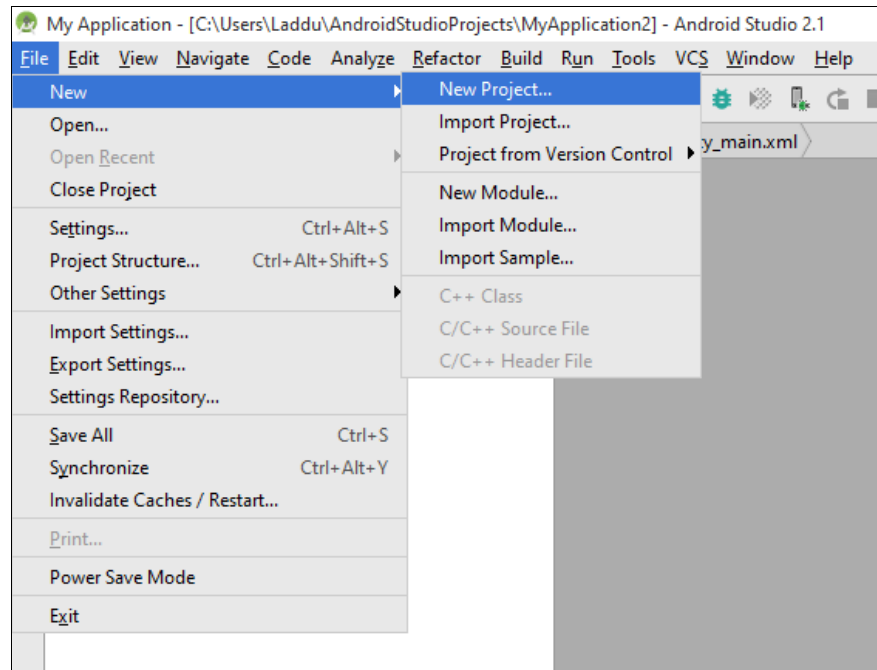


Figure 2.4: Creating a new project in Android Studio

2. First window for project setup will ask you to enter the name of the application and your company domain. Company domain name is used to assign the unique package name to your application because package name is going to identify your app in the application store uniquely. Here you will assign a project location (working directory) of your project where all the project work will be saved. It is shown in Figure 2.5.

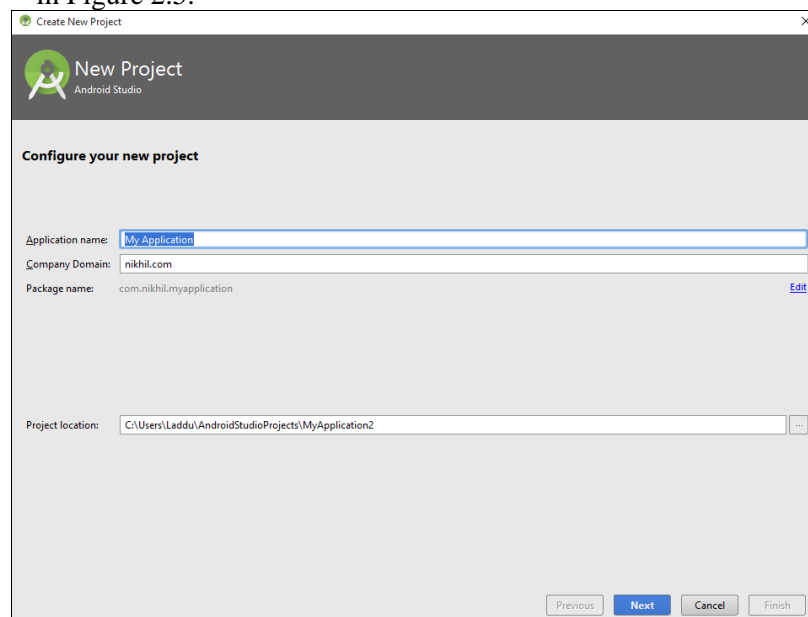


Figure 2.5: Defining the name and the package of your new application

3. When you click on the next button, following window (as shown in Figure 2.6) will appear on the screen:

Visual Programming

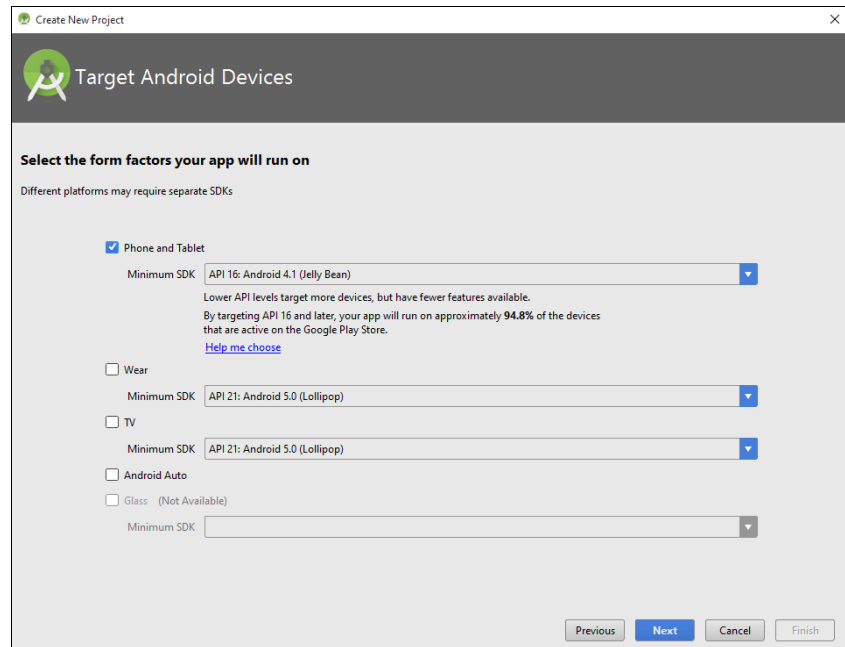


Figure 2.6: Selecting the platform and minimum SDK for your application

In this window (Figure 2.6), select the minimum SDK for targeted Android devices. Minimum SDK specifies the minimum Android version to which your application is going to support.

4. When you click on the next button, a window will pop up with different Activity templates. Select the appropriate Activity template and click on the next button (as shown in Figure 2.7)

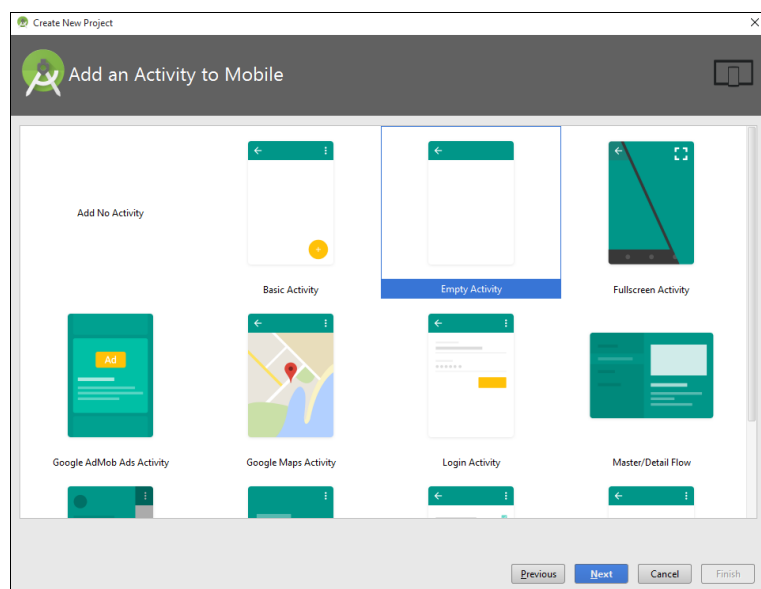


Figure 2.7: Selecting a default Activity template for your home screen

5. When you click on the next button, the “Customize the Activity” window will pop up. This will ask you to enter Activity file name (A Java file that is having the logic part of the Activity) and Layout file name (An XML file that is having the design part of the Activity). It is shown in Figure 2.8.

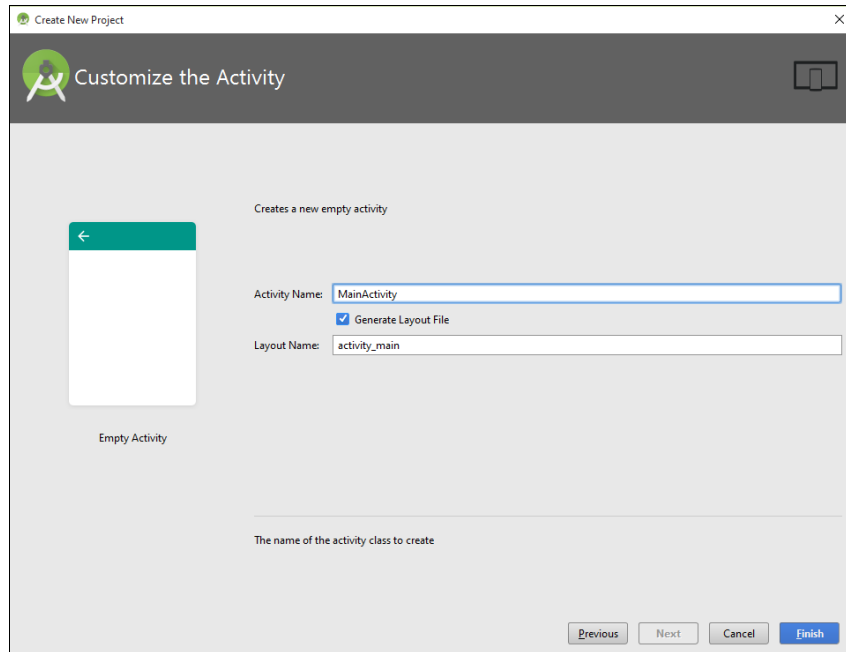


Figure 2.8: Assigning the name to the Activity and its layout file

6. When you click on the finish button, a working window with text editor and different tools will be displaying on the screen (as shown in Figure 2.9).

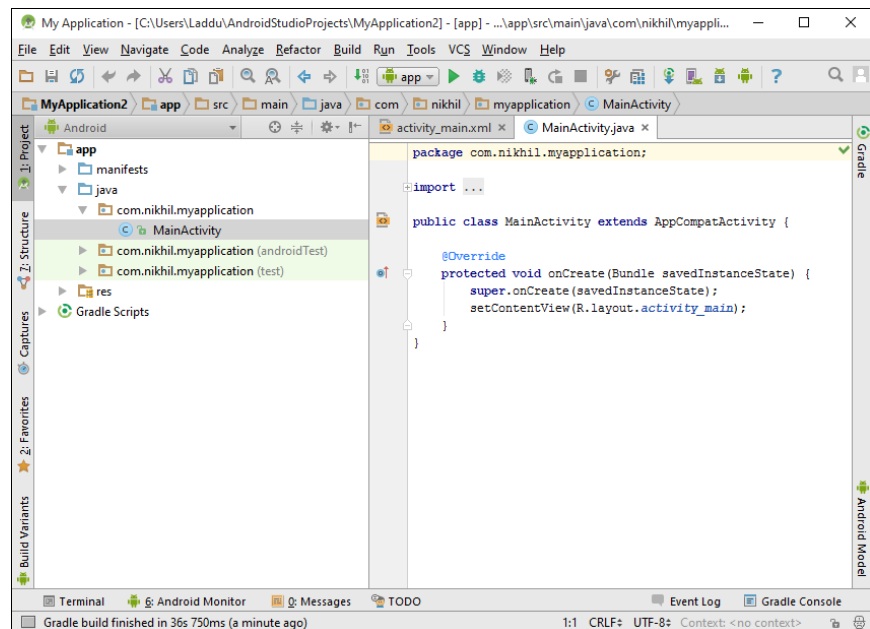


Figure 2.9: working window of Android Studio

Testing of App on multiple physical devices at one place is not possible. Therefore, Android SDK provides an emulator to test your application against to all versions of Android. An emulator provides virtual environment to test your applications. It eliminates the requirement of a real physical device. This emulator uses an Android Virtual Device

Visual Programming

(AVD) to run, so it can be used only when an AVD has been created. Following sections explain how to create and configure an AVD.

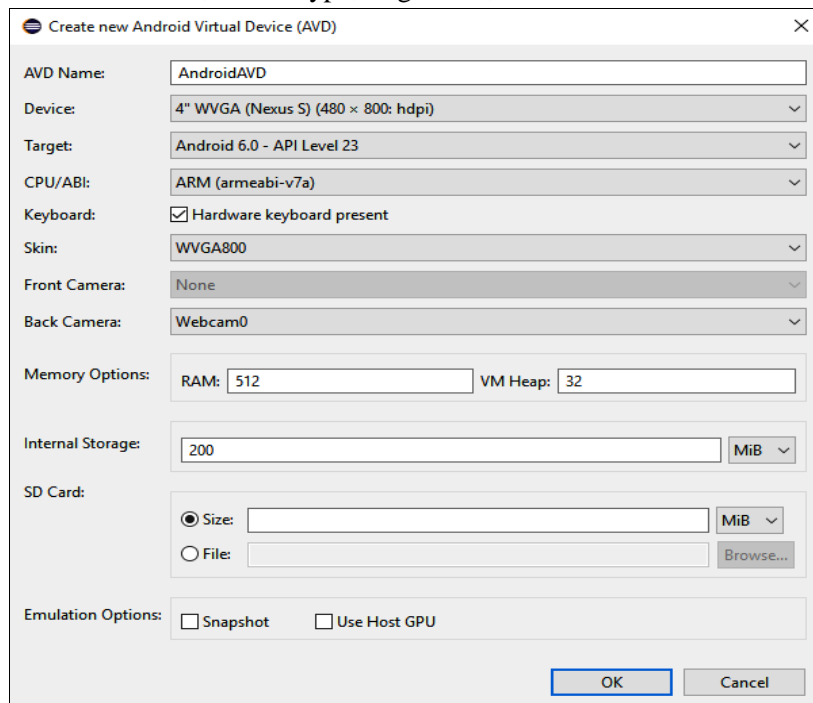
2.3 Creating Android Virtual Device

In this section, you will learn to create a new Android virtual Device in both IDEs (Eclipse and Android Studio).

2.3.1 Creating an AVD in Eclipse

You can create as many Android Virtual Devices as you want with different configuration. Attempt the following steps to create an AVD in Eclipse IDE:

1. Launch the Eclipse and select “Android Virtual Device Manager” from “Window” menu. The “Android Virtual Device Manager” window will appear. If you have already created some AVD, that will be showed in the list. For creating a new AVD, there are two tabs in this window: “Android Virtual Devices” and “Device Definitions”. In “Device Definitions” tab, a list of some known device definitions is available. You can select any one of these definitions to create an AVD. Otherwise, you can create a new customized AVD by clicking on the “Create...” button in “Android Virtual Devices” tab.
2. When you click on the “Create...” button, “Create New Android VirtualDevice” window will appear on the screen (as shown in Figure 2.10). In this window you will need to enter the details such as AVD name, device type, target API, etc.



The screenshot shows the "Create new Android Virtual Device (AVD)" dialog box. It contains the following fields and options:

- AVD Name:** Text field with "AndroidAVD" entered.
- Device:** Dropdown menu showing "4" WVGA (Nexus S) (480 x 800: hdpi)".
- Target:** Dropdown menu showing "Android 6.0 - API Level 23".
- CPU/ABI:** Dropdown menu showing "ARM (armeabi-v7a)".
- Keyboard:** Check box labeled "Hardware keyboard present" which is checked.
- Skin:** Dropdown menu showing "WVGA800".
- Front Camera:** Dropdown menu showing "None".
- Back Camera:** Dropdown menu showing "Webcam0".
- Memory Options:** Two text fields: "RAM:" with "512" and "VM Heap:" with "32".
- Internal Storage:** Text field with "200" and a "MiB" dropdown.
- SD Card:** Radio button "Size:" is selected, followed by a text field and a "MiB" dropdown. There is also an unselected "File:" radio button with a "Browse..." button next to it.
- Emulation Options:** Two unchecked checkboxes: "Snapshot" and "Use Host GPU".
- Buttons:** "OK" and "Cancel" buttons at the bottom right.

Figure 2.10: Creating a New AVD

3. After filling all the details of AVD, click on the “OK” button to create a new AVD. Now your AVD will appear on the list of existing AVDs in the “Android Virtual Device Manager” window (as shown in Figure 2.11).

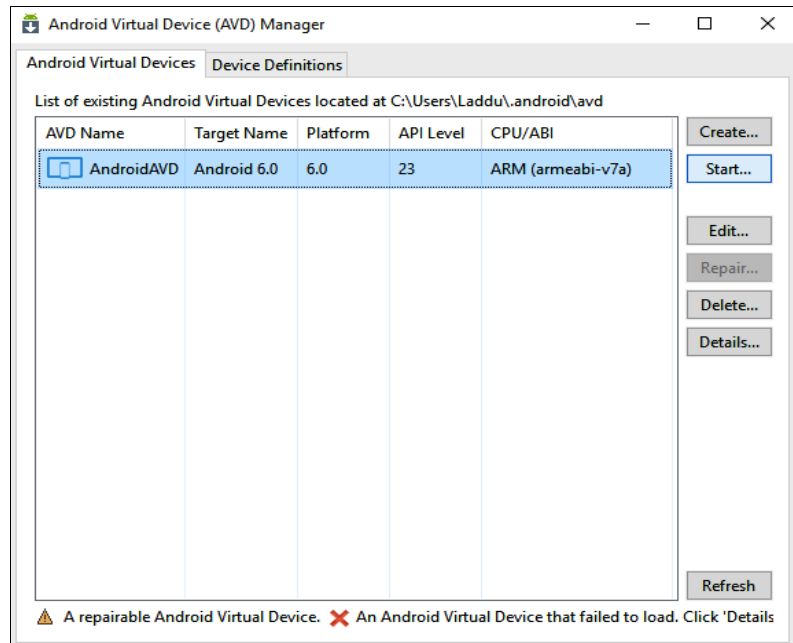


Figure 2.11: List of Existing AVDs in AVD Manager Window

2.3.2 Creating an AVD in Android Studio

1. To create an AVD in Android Studio, Select the “AVD Manager” option in the “Android” submenu of “Tools” menu. An AVD Manager window will pop up. If you have already created some AVD, then it will be show in the list of AVDs. To create a new AVD click on the “Create New AVD” button. After clicking on the button, the “Virtual Device Configuration” window will pop up on the screen as show in Figure 2.12.

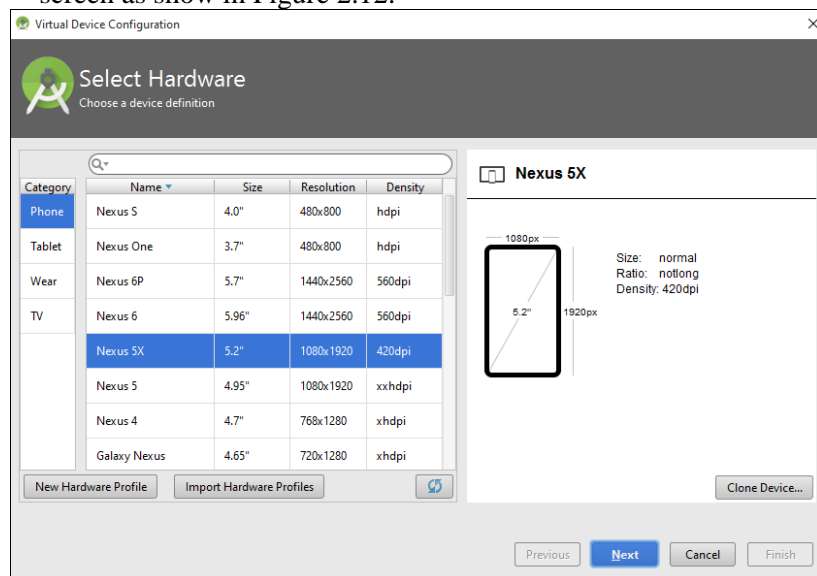


Figure 2.12: Choosing device definition

2. Few definitions of some standard Android devices are listed here, as shown in the Figure 2.12. Either select any standard definition or create your own definition by clicking on the “New Hardware Profile” button. If you are creating your own hardware profile, then

Visual Programming

“Configure Hardware Profile” window will pop up where you can fill all necessary details of your device definition and provide a name to the definition. After setting the configuration click on the “Finish” button and then “Next” button in the next window (“Select a system image” window). This leads to a new window “Verify Configuration” where you should provide the name to the created AVD (for example NIK_AVD). Here you can change some attributes of AVD such that Android Version, resolution and orientation of the screen, etc. and complete the job by selecting the “Finish” button.

3. Now your created definition will be available in the list of AVDs in AVD manager window as shown in Figure 2.13.

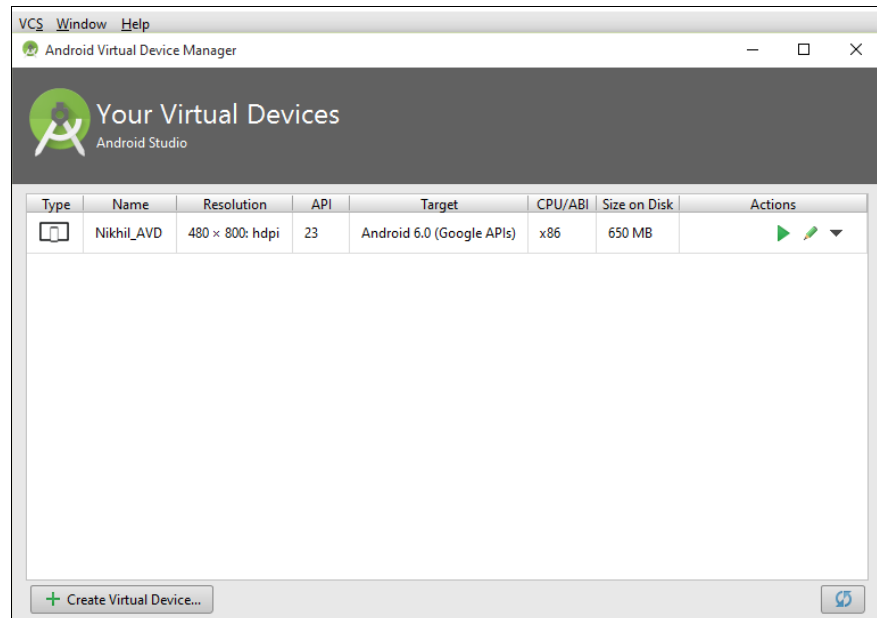


Figure 2.13: AVD Manager is showing the created AVD

AVD has been created. The upcoming sections will discuss the process to run the app in the created AVD.

2.4 Creating and saving launch configuration

Whenever an app is created, you need to run and test the app against an AVD repeatedly by selecting the AVD on each launch. This extra overhead of selecting the AVD at each launch can be eliminated by tying up a specific AVD to a project. For this purpose, you need to save the launch configuration for the project (or application).

2.4.2 Creating and saving the launch configuration in Android Studio

For creating and saving the launch configuration for a project in Android Studio, pursue the following steps:

1. In Android studio, when you select the “Edit Configuration...” option from the “Run” menu, a “Run/Debug Configuration” window will pop up on the screen as shown in Figure 2.16.

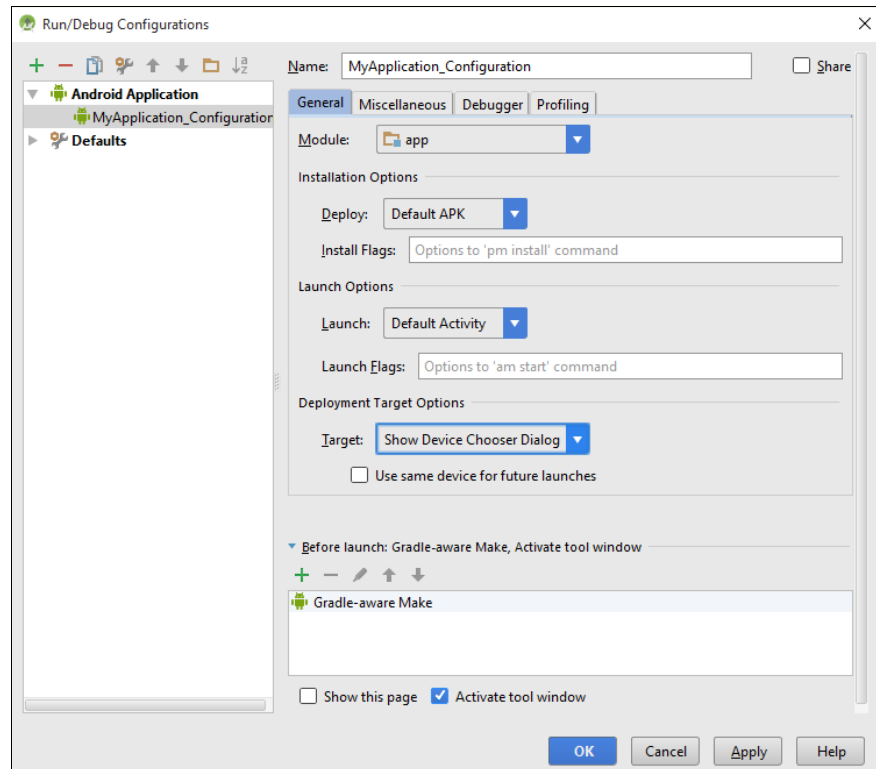


Figure 2.16: Setting the Run and Debug configuration in Android Studio

2. Click on the plus sign (+) button (window shown in Figure 2.16), to add new run configuration. Now assign new name to the run configuration, select the module (Application Name) to attach with the configuration, select the default launch Activity, select the default device to run and test the application, etc.
3. Now click on the “Apply” and “OK” button respectively. A new default launch configuration for the project in Android Studio will be created.

2.6 Understanding the development platform

Before moving further, you must have a better understanding of basic ingredients of the Android project. You must know about different files that create during app development in the Eclipse environment.

2.6.1 Introduction to important Android project files

Following files have a significant importance in Android application development:

2.6.1.1 AndroidManifest.xml

AndroidManifest.xml file plays a very important role in Android development. This is the heart of any android application. It has all details about the application including package of the app, version of the app, minimum and maximum Android SDK versions supported by the app, themes, permissions, activities, intents, broadcast receivers, content

Visual Programming

providers and much more. Following is the code outline of the AndroidManifest.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.nikhil.myfirstapp"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="23"/>
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

AndroidManifest.xml

Some of the components of AndroidManifest.xml file are described below:

- a. **<manifest>:** This is root element of the manifest file. As you can see in the above that whole structure of manifest file is enclosed within it. It must contain at least an <application> element and two attributes; “xmlns:android” and “package”.
- b. **<uses-sdk>:** This element defines the maximum and minimum SDK version that must be available in your device in which you are going to install this app.
- c. **<application>:** This element is used to represent the information of the application (including title, theme, icon etc.). This element contains the specifiers of the components of an application (including Activity, Service, Broadcast Receiver and content Provider).
- d. **<activity>:** This tag is used to specify the presence of an Activity that can be displayed. It has two other sub tags; intent-filter and action.
- e. **<intent-filter>:** Intent-filter is used to specify that which Intent can be used to start the Activity, Service or Broadcast Receivers.
- f. **<action>:** This tag specifies the generic action to be performed by an intent-filter.
- g. **<category>:** This tag adds a category name to an intent-filter.

- h. **<permission>**: This tag is used to add permissions to an application. There can be multiple permission tags.



Reading

There are many more elements and attributes of AndroidManifest.xml file. For more details of AndroidManifest.xml file follow the following link:

<http://developer.android.com/guide/topics/manifest/manifest-intro.html>

2.6.1.2 MainActivity.java

This file contains the logic part of the main Activity of the application. As you can see in the code, first line represents the package of your application. Next two lines imports the minimum necessary classes. The MainActivity class extends the Activity class to inherit the methods of Activity class because MainActivity is going to be an Activity. In the next line, an overridden method onCreate() is used to execute the things on creation of the Activity. Next to this, method setContentView() is called to render the graphical components (a View or a Layout) on the screen.

```
package com.nikhil.myfirstapp;
import android.app.Activity;
import android.os.Bundle;
publicclass MainActivity extends Activity {
    protectedvoid onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

MainActivity.java

2.6.1.3 activity_main.xml

This is a XML file which defines the layout for an Activity. Every Activity needs a layout file to display the contents on the screen. There are many layouts available for arranging the graphical components on the screen in a specific manner. A layout can contain graphical components (like TextView, Button, EditText, etc.). This file resides in the layout folder.

```
<RelativeLayoutxmlns: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:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="com.nikhil.myfirstapp.MainActivity">

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

Visual Programming

```
android:layout_height="wrap_content"
android:text="@string/hello_world"/>

</RelativeLayout>
```

activity_main.xml

2.6.1.4 strings.xml

Remember that, Android does not recommend any hard coded string in the layout file. Strings should be saved separately as a resource in a resource file strings.xml. This file is found in the “values” folder under the “res” folder.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">MyFirstApp</string>
<string name="hello_world">Hello world!</string>
<string name="action_settings">Settings</string>
</resources>
```

strings.xml

2.6.1.5 R.java

Android recommends that the logic part should be written in Java and design part should be written in XML, as by using these different technologies it gets easier to manage layouts, views, and other resources separately. The graphical components coded in XML are going to be used in Java code. But, how can it be done? The answer is – by using R.java. R.java is a Java file that is created automatically by Android development environment during the programming. It works as a bridge between logic part and design part. It interprets the references of UI components of XML file to the Java file and to other XML files. It is recommended that do not make any change in the R.java file because you can alter the reference of any graphical component by mistake.

2.6.2 Introduction to different Android tools

Android development environment is a bundle of multiple tools. Each tool has its own significance and plays an important role in Android development. Some Android tools are described below:

2.6.2.1 DVM and ART

Dalvik Virtual Machine (DVM) is used by Android system to run the compiled code of Android (that is called Dalvik executable or .dex file). It is used in Android system similarly as JVM is used in Java to execute the byte code (.class file). Dalvik executable (.dex file) is highly optimized .class file; optimized for low memory and least processing power. Remember, DVM doesn't work with .class files.

Android Runtime (ART) is the successor of Dalvik Virtual Machine (DVM). It has some advantage over DVM including ahead-of-time compilation (AOT), improved garbage collection and other development and debugging enhancements.



Reading

For more details follow the section 1.4 of UNIT-1 and track the following link:

<https://source.android.com/devices/tech/dalvik/>

2.6.2.2 AVD Manager

The AVD Manager provides you an interface that is used to create and manage the AVDs.



Reading

For more details refer the previous section 2.3 and follow the below mentioned links:

<http://developer.android.com/tools/help/avd-manager.html> and

<http://developer.android.com/tools/devices/managing-avds.html>

2.6.2.3 Android SDK Manager

The SDK Manager is used to install, uninstall and manage different SDK versions, support libraries, examples and documentation.



Reading

For more details refer the previous section 1.6.4 of previous unit 1 and follow the following link:

<http://developer.android.com/tools/help/sdk-manager.html>

2.6.2.4 Android Emulator

The Android Emulator is a virtual mobile device that runs in your computer. The emulator lets you run, test and debug your application. It provides you a mobile platform which does not need any real mobile hardware. You can connect to the internet; can make a call and SMS with the help of Android Emulator. It is a DVM implementation that needs an AVD instance created by the AVD Manager to show all above mentioned features. The file that installs in an Android Emulator (or Real Device) is .apk (Application Package) file. “.apk” generally is created automatically in build process or you can create it by the export option. “.apk” file is combination of different files including “.dex” file, compiled and non-compiled resources and AndroidManifest.xml file (Figure 2.19).

Visual Programming

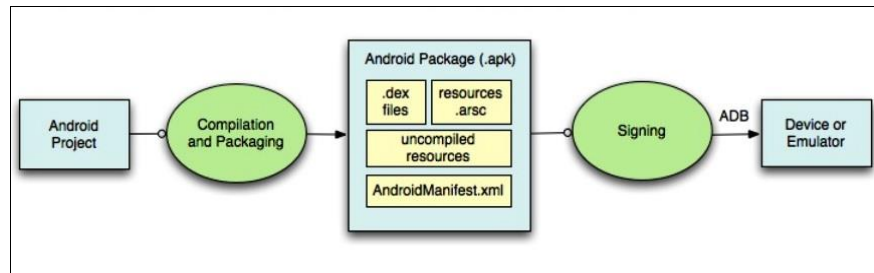


Figure 2.19: Build and run Process

(Source: <http://developer.android.com/tools/building/index.html>)



Reading

For more details of building and run process of the app go after the following links:

<http://developer.android.com/tools/help/emulator.html> and
<http://developer.android.com/tools/devices/emulator.html>

2.6.2.5 DDMS

Dalvik Debug Monitor Server (DDMS) is a debugging tool that helps you to understand and monitor the active processes, view the heap and the stack, looking into the threads and the logcat, incoming call and SMS spoofing and more. It is shown in Figure 1.9 of UNIT-1.



Reading

To know more about the Dalvik Debug Monitor Server, follow the link:

<http://developer.android.com/tools/debugging/ddms.html>

2.7 Building blocks of Android application

Building blocks or application components of an Android application defines the behaviour of the application. Each Android application has four types of components; each component plays a specific role and has its own lifecycle. They are described below.

2.7.1 Activity

An Activity is a user interface provided as a screen with which a user interacts. For example, if you open a media player app, then the screen that allows you to play, pause, next or previous the song is an Activity. An Activity can either be fit into the whole screen or can float over other screens. An Android application can have multiple Activities linked to each other. The very first Activity that renders on the screen after opening the application is called the default Activity.

2.7.2 Services

A service is an application component that runs in the background and does not need any user interface. For example, if you are using a web browser and decide to download a file, then file would be downloaded in

the background while you continue your interaction with the web browser. This is being done due to an application component called service. An Activity can start a service. You can also bind a service to an Activity to interact with it.

2.7.3 Content Providers

A content provider is a component that manages the application data stored in SQLite database, file system, on the web or any other storage location accessed by your application. Using the content provider other applications can access or modify the data of your application. For example, Android system can provide the contact information to the messaging app through the content provider. It is also helpful in managing and querying the private data of the app.

2.7.4 Broadcast Receivers

A broadcast receiver is used to broadcast the announcements to whole system whether it is notification to status bar to alert the user or it is a message to another application or Activity. For example, if you either receive a notification on status bar about completion of download or receiving a text message about the new update of the app, it happens because of the broadcast receivers. Even, a broadcast receiver can start a service on occurrence of a predefined event.

2.8 Adding permissions to the AndroidManifest.java file

Permissions in the application are restrictions that limit the access of device data, code and resources. This limitation is forced to protect vital data and program that can be altered or damaged. These permissions are set in AndroidManifest.xml file.

Examples of some permission are as follows:

- android.permission.BLUETOOTH,
- android.permission.VIBRATE,
- android.permission.SET_WALLPAPER,
- android.permission.FLASHLIGHT and many more.

There are numerous features in Android and each feature is secured by permission. Permission is declared with in a <uses-permission> element. When the application is installed, it seeks permissions from the user. If user grants the permissions then application can use the protected features else it doesn't use it. An application can also protect the activities, services, broadcast receivers, and content providers with permissions.



Reading

For more details of how to add permissions to a Manifest file, trace the following link:

<http://developer.android.com/guide/topics/manifest/uses-permission-element.html>

<http://developer.android.com/guide/topics/manifest/permission-element.html>

Unit summary



Summary

In this unit you learned about creating a new Android project in Eclipse and Android Studio IDEs. You have also been trained about creating the Android Virtual Devices to run, test and debug your application. Configuring and saving the launch configuration and associating an AVD with your project is also going to make your debugging and testing task easier. You also learnt about different project files that are created during the development process of an Android application. You also got an idea about different development tools, Android application components and adding permissions to your application.

Assignment



Assignment

Q 1: What is the difference between “Minimum Required SDK”, “Target SDK” and “Compile With”.

Answer:

Minimum Required SDK: The new versions of Android often provide best APIs for your application but you must continue to support previous versions of Android until older version devices get updated. So, during development process select the lowest version from the drop down list. As lower the version you select, more the devices it supports.

Target SDK: Select the highest API level that the application is going to work with. Since the application does not have forward compatibility issue for target API level. It is recommended to set highest level of API for Target SDK. Your application can still work with older versions up to Minimum SDK level that is set by you.

Compile With: You should select the most recent version of installed SDK to compile the project.

Q 2: Explain the role of R.java file.

Answer:

Android recommends that the logic part should be written in Java and design part should be written in XML, as by using these different technologies it gets easier to manage layouts, views, and other resources separately. The graphical components coded in XML are going to be used in Java code. But, how can it be done? The answer is – by using R.java. R.java is a Java file that is created automatically by Android development environment during the programming. It works as a bridge between logic part and design part. It interprets the references of UI components of XML file to the Java file and to other XML files. It is recommended not make any change in the file because you can alter the reference of any graphical component by mistake.

Q 3: Explain the role of AndroiManifest.xml file in an Android application.

Answer: AndroidManifest.xml file plays a very important role in Android development. This is the heart of any android application. It has all details about the application including package of the app, version of the app, minimum and maximum Android SDK versions supported by the app, themes, permissions, activities, intents, broadcast receivers, content providers and much more.

Q 4: What are different basic components of an Android application?

Answer: Building blocks or application components of an Android app defines the behaviour of the apps. Each Android application has four types of components; each component plays a specific role and has its own lifecycle. Each component is described below.

Visual Programming

An Activity is a user interface provided as a screen with which a user interacts. For example, if you open a media player app, then the screen that allows you to play, pause, next or previous the song is an Activity. An Activity can either be fit into the whole screen or can float over other screens. An Android application can have multiple Activities linked to each other. The very first Activity that renders on the screen after opening the app is called main Activity (or default Activity).

A service is an application component that runs in the background and does not need any user interface. For example, if you are using a web browser and decide to download a file, then file would be downloaded in the background while you continue your interaction with the web browser. This is being done due to an application component called service. An Activity can start a service. You can also bind a service to an Activity to interact with it.

A content provider is a component that manages the application data stored in SQLite database, file system, on the web or any other storage location accessed by your application. Using the content provider other apps can access or modify the data. For example, Android system can provide the contact information to the messaging app through the content provider. It is also helpful in managing and querying the private data of the app.

A broadcast receiver is used to broadcast the announcements to whole system whether it is notification to status bar to alert the user or it is a message to another application or Activity. For example, if you receive a notification on status bar about completion of download receiving a text message or about the new update of the app, it happens because of the broadcast receivers. Even, a broadcast receiver can start a service on occurrence of a predefined event.

Q 5: What is the role of an Android emulator?

Answer: The Android Emulator is a virtual mobile device that runs in your computer. The emulator lets you run, test and debug your application. It provides you a mobile platform which does not need any real mobile hardware. You can connect to the internet; can make a call and SMS with the help of Android Emulator. It is a DVM implementation that needs an AVD instance created by the AVD Manager to show all above mentioned features.

Assessment



Assessment

Problem 1: Why do you manage the logic part and the design part separately? Which type of files contains the logic part and design part?

Problem 2: What is an .APK file? What are the ingredients of it? How is it created? Explain.

Problem 3: What are the permissions? How can you add permissions to a Manifest file?

Unit 3

Using Views and Layouts

3.1 Introduction

For any application two most important aspects are user interface and logical functionality of the application. User interface gives the first impression to the user before using its functionality. User interface of an application must be user friendly and interactive that should serve its features very effectively.

In Android, a user generally communicates to an Android device with the help of an application that provide its characteristics through a set of Activities. The graphical design of an Activity is a collection of different graphical components (such as Button, Text Fields, etc.) called views. Usually, Activity uses the layout managers (such as Linear Layout, Relative Layout, etc.) to contain these views. Layout managers help the Activity to arrange the views on the screen. These layout managers are also called view groups.

There are variety of views and view groups available in Android. These views and view groups come under the classes View and ViewGroup respectively. There are a number of events associated with class View and ViewGroup that can be handled programmatically.

View and ViewGroups are not enough to build all user interfaces. Android gives you several other graphical components to design standard layout. These graphical components (including Action Bar, Dialogs, Action Bar and Status Notification) provide a bunch of APIs to program.

In this unit, you will learn to design the graphical layout (such as Linear Layout, Relative Layout, etc.) of an Activity. You will also grab some knowledge about distinct Views and bind the data to the views using AdapterView class. You will also learn about some measurement units of measuring the component sizes.

Upon completion of this unit you will be able to:

- *Design* the layout of an Activity.
- *Use* different Views and ViewGroups.
- *Manipulate* the XML file.
- *Apply* measurement units.
- *Exercise* distinct layout attributes.
- *Implement* AdapterView class.
- *Handle* the events.



Outcomes



Terminology

GUI:	A Graphical User Interface (GUI) is a set of menus, icons, commands, views and layouts through which a user communicates with an application or program.
View:	In Android, a view is a GUI component that user uses to interact with the screen such as Button, Text Fields, Labels, etc.
Layout:	A layout is an arrangement of different View and other graphical components on the screen. It is also called a View Group.
Event:	An event is an action sensed by a program such as pressing a button, releasing a button, selecting an item, etc.
XML:	XML (Extensible Mark-up Language) is mark-up language for documents for having organized information. XML is an HTML like language which provides a readable structure of the document (text and pictures) for the layout.
XML Attributes:	XML attributes are usually used to define XML element, or to deliver the extra information about the elements.
Reference:	A reference is a value (such as an address) that is used by a program to access an object (such as a view, layout, variable, record, image, etc.) in computer memory.
Listener:	A listener is an object that is used to receive a notification when a specific event takes place.
Interface:	An interface defines a convention by including a set of unimplemented methods (abstract method declarations). The methods in an interface need to be defined by the class that will be implementing the interface.

3.2 Views

Generally, an Activity includes several views and view groups in different styles and arrangements. A view is a GUI component that user uses to interact with the screen. Views are the interactive components and are also called the input controls. A view group is also a GUI component that holds other views and view groups to describe the layout of the interface on the screen. All views and view groups derived from two classes “android.view.View” and “android.view.ViewGroup” respectively. There are a number of subclasses of View and ViewGroup that provides commonly used input / output controls (such as labels, buttons, text fields, etc.) and different layouts (such as linear layout, relative layout, etc.).

To design the user interface, each component is defined by the hierarchy of Views and ViewGroups objects (as shown in Figure

Visual Programming

3.1). View groups are just like a container that is used to manage the order and appearance of other views and view groups.

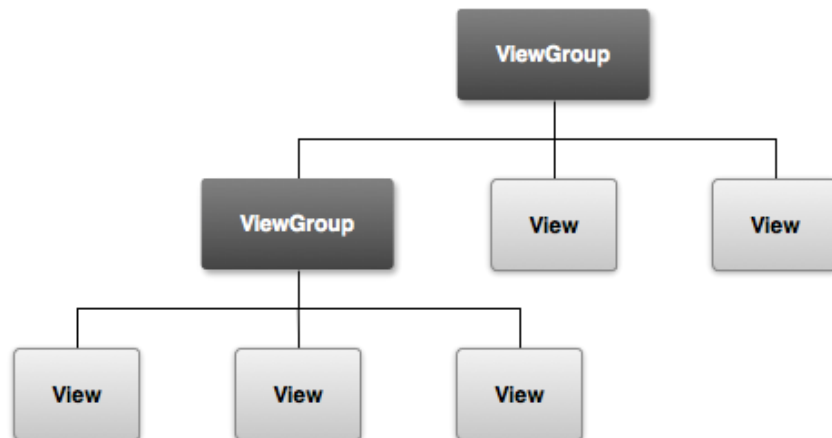


Figure 3.1: Illustration of a view hierarchy, which defines a UI layout.
(Source: <http://developer.android.com/guide/topics/ui/overview.html>)

You can add views in the Activity class and design the layout with the help of java code. But, the most effective and simplest way is to define the layout using an XML (Extensible Mark-up Language) file. XML is an HTML like language which provides a readable structure of the code for the layout.

Android provides a variety of views. There are three categories of views: basic views, picker views and list views (Figure 3.2). Basic views are the common views like Button view, EditText view and TextView view. The picker views (such as DatePicker view and TimePicker view) permit the user to select the content from a view. The list views (such as SpinnerView, etc.) are used to exhibit the list of items. Generally, you will be using XML layouts to add an input control (or view).

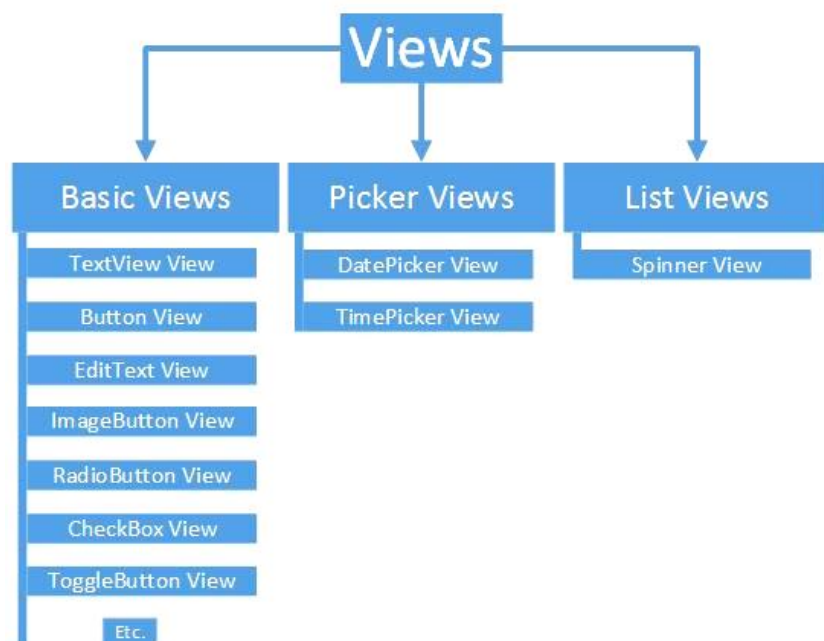


Figure 3.2 Categories of Views

Different views are described in following sections:

3.2.1 TextView view

A TextView view is used to show the text to the user on the Activity screen of the Android application. Generally, whenever you create a new project either in the Eclipse or Android Studio, the default Activity of the application includes a TextView in the file activity_main.xml.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical">
<TextView
android:id="@+id/textView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/hello_world"
android:textSize="20sp"/>
</LinearLayout>
```

activity_main.xml

Due to the above code the appearance of the TextView on the screen is as shown in Figure 3.3.

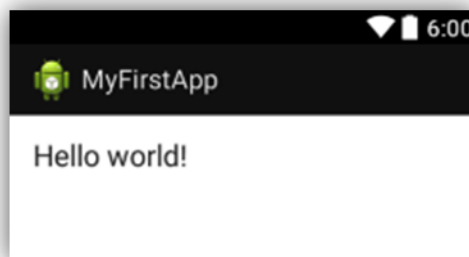


Figure 3.3: Text View on the Activity

The XML code shown above in bold is for declaring a TextView (or Label) with different attributes such as layout_height, layout_width, text and textSize. In the XML code, TextView element is written under the LinearLayout element. Different layouts will be discussed in the upcoming section 3.3. Every view has some attributes associated with it; some of them are common for all views and some are specific to an individual view. Table 3.1 is showing some of the XML attributes that are used for TextView element.

Table 3.1 XML attributes of TextView element

Attribute	Description
android: gravity	It is used to align the text according to X & Y axis.
android: hint	It is used to set hint text if no text appears.
android: lines	It is used to set the number of lines for the text.

Visual Programming

android: text	It sets the text to be displayed.
android: height	It is used to set the height of the text view.
android: width	It is used to set the width of the text view.
android: textSize	It allows you to set the size of the text.
android: textAppearance	It is used to set the text size, style and colour from already defined resource file.
android: id	It is used to set the id of a view. Every view must have a unique id attribute which helps to make it identifiable in many views.

There are many measurement units (such as sp, dp, px, etc.) which are generally used for different measuring attributes such as height, width, size, etc. These measurement units are described in detail in section 3.4.

To get the reference of the TextView in Java file, use findViewById() method of Activity class and assign the reference to the reference variable of android.widget.TextView class. Example,

```
package com.nikhil.myfirstapp;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView tv = (TextView) findViewById(R.id.textView1);
    }
}
```

MainActivity.java

Here, R is automatically created Java file for different XML resources such as layouts, views, strings, colours, themes, images, etc. It helps Java program to access the resources created in XML.

3.2.2 EditText view

The EditText view allows you to input editable text. It can be created either by using <EditTextView> element of XML or by using android.widget.EditText class of Java. For inputting the text you can touch the text field on the screen and it places the cursor into it and display the virtual keyboard. You can select, copy, cut, paste or delete the inputted text.

Following XML code snippet is used for inclusion of the EditText view in the XML layout file:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
```

```

android:orientation="vertical">
<EditText
  android:id="@+id/editText1"
  android:layout_width="258dp"
  android:layout_height="wrap_content"
  android:inputType="text"
  android:ems="10"
  android:hint="@string/editText">
</EditText>
</LinearLayout>

```

activity_main.xml

Above code will display the EditText view in the Activity as shown in Figure 3.4.

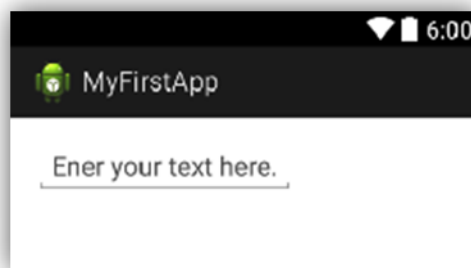


Figure 3.4: EditText View on the Activity

To get the reference of the EditText view, use the reference variable of `android.widget.EditText` class. For example,

```

package com.nikhil.myfirstapp;

import android.app.Activity;
import android.os.Bundle;
import android.widget.EditText;

public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        EditText et = (EditText)findViewById(R.id.editText1);
    }
}

```

MainActivity.java

3.2.3 Button view

Button is an input control that is used to communicate the action to the application when user touches it. It consists of an icon or text or both. On the basis of what you want to add to your button, the XML declaration of Button would be as following code snippets.

```

<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">
  <Button

```

Visual Programming

```
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Button"/>
</LinearLayout>
```

activity_main.xml

On the Activity screen, button will look as shown in Figure 3.5.

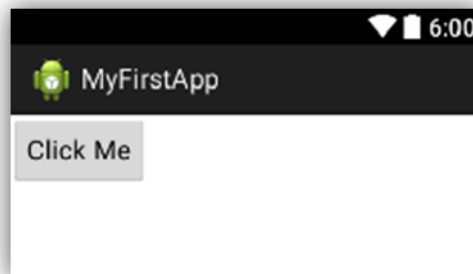


Figure 3.5: Button View on the Activity

In Java file, you can get the reference of Button view in the object of `android.widget.Button` class. You can set the `OnClickListener` on the button and can define the functionality of the button by overriding the `onClick()` method.

```
package com.nikhil.myfirstapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity implements OnClickListener {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button b = (Button) findViewById(R.id.button1);
        b.setOnClickListener(this);
    }

    public void onClick(View v) {
        Toast.makeText(MainActivity.this, "Hello, Dear!",
            Toast.LENGTH_LONG).show();
    }
}
```

MainActivity.java

In the above written code the `OnClickListener` is an interface that has an unimplemented method `onClick()` which is used to define the functionality of the Button.

You can also call a method from the XML code on the `onClick` event. Use “`onClick`” attribute in a view declaration and provide the method name (that is defined in Java code) as a value to the “`onClick`” attribute.

Here you have no need to set the OnClickListener explicitly. Following code depicts it:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical">
<Button
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/button"
android:onClick="method1"/>
</LinearLayout>
```

activity_main.xml

If you add the onClick attribute to a view in the XML code then the code in Java Activity class will be:

```
package com.nikhil.myfirstapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void method1(View v) {
        Toast.makeText(MainActivity.this, "Hello, Dear!",
        Toast.LENGTH_LONG).show();
    }
}
```

MainActivity.java

Here, makeText() is a static method of Toast class that is used to toast a popup message on the screen. Note that, when you add the method in XML file, then in Java code it takes the object of View class as an argument. In both the cases, on the click of the button, a toast message "Hello Dear!" will be displayed on the screen (as shown in Figure 3.6).

Visual Programming

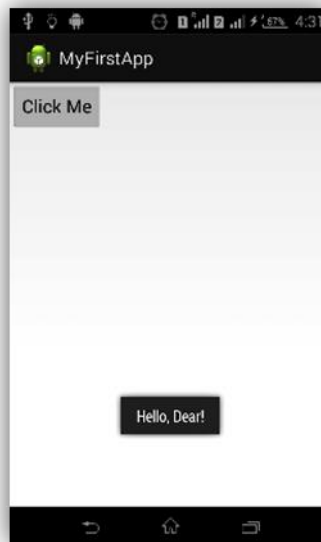


Figure 3.6 Activity is toasting a message on the button click

3.2.4 RadioButton view

The RadioButton view has two states; selected or unselected. Generally, radio buttons are used in groups. This group is called RadioGroup. In a RadioGroup all radio buttons are mutually exclusive; i.e. if you select one radio button in a radio group then others become unchecked automatically. Following XML code is using two RadioButtons in a RadioGroup.

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

<RadioGroup
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:id="@+id/r_Group">

    <RadioButton
        android:id="@+id/r_Button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/RB1"/>

    <RadioButton
        android:id="@+id/r_Button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/RB2"/>

</RadioGroup>
```

activity_main.xml

The look of radio button group in the Activity will be as Figure 3.7.

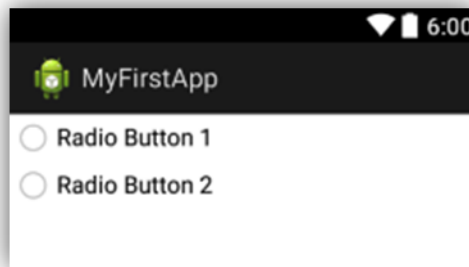


Figure 3.7 Radio Button group

In Activity class you will override the `setOnCheckedChangeListener()` method of `OnCheckedChangeListener` interface for generating the checked event. Following code is used to toast the corresponding messages on the screen when you select the different radio buttons in the radio group.

```
package com.nikhil.myfirstapp;

import android.app.Activity;
import android.os.Bundle;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.RadioGroup.OnCheckedChangeListener;
import android.widget.Toast;

public class MainActivity extends Activity
    implements OnCheckedChangeListener {
    RadioButton rb1, rb2;
    RadioGroup rg;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        rg = (RadioGroup) findViewById(R.id.r_Group);
        rb1 = (RadioButton) findViewById(R.id.r_Button1);
        rb2 = (RadioButton) findViewById(R.id.r_Button2);
        rg.setOnCheckedChangeListener(this);
    }
    public void onCheckedChanged(RadioGroup g, int id) {
        if (rb1.isChecked()) {
            Toast.makeText(MainActivity.this, "Radio Button
1 is checked", Toast.LENGTH_SHORT).show();
        }
        if (rb2.isChecked()) {
            Toast.makeText(MainActivity.this, "Radio Button
2 is checked", Toast.LENGTH_SHORT).show();
        }
    }
}
```

MainActivity.java

When you run the above mentioned codes in your Android device, it will show the messages as shown in Figure 3.8.

Visual Programming



Figure 3.8 Activity is toasting a message on click of the Radio Button

3.2.5 CheckBox view

The CheckBox view is similar to RadioButton view because it has two states; checked or unchecked. But, unlike RadioButton view, you can uncheck the checked CheckBox view or vice-versa by re-clicking on it. It means you can select multiple CheckBox views in a set of check boxes. Following XML code depicts the code of declaring CheckBox views.

```
<?xmlversion="1.0"encoding="utf-8"?>

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:orientation="vertical">

<CheckBox
android:id="@+id/checkBox1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/CB1"/>

<CheckBox
android:id="@+id/checkBox2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/CB2"/>

</LinearLayout>
```

activity_main.xml

On the execution of the above XML code the CheckBox views created will look as Figure 3.9.

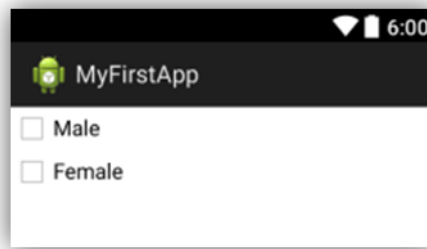


Figure 3.9 CheckBox Views

To perform the click event on the CheckBox views, you will implement the `OnClickListener` interface and override the `onClick()` method. In the following code snippet, the message would be displayed on the basis of CheckBox view is checked or not.

```
package com.nikhil.myfirstapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.CheckBox;
import android.widget.Toast;

public class MainActivity extends Activity implements OnClickListener {
    CheckBox cb1, cb2;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        cb1 = (CheckBox) findViewById(R.id.checkBox1);
        cb2 = (CheckBox) findViewById(R.id.checkBox2);
        cb1.setOnClickListener(this);
        cb2.setOnClickListener(this);
    }

    public void onClick(View v) {
        CheckBox cb = (CheckBox) v;
        int x = v.getId();
        if (x == R.id.checkBox1) {
            if (cb.isChecked())
                Toast.makeText(MainActivity.this, "Male
                Selected", Toast.LENGTH_SHORT).show();
            else
                Toast.makeText(MainActivity.this, "Male
                Unselected", Toast.LENGTH_SHORT).show();
        }
        if (x == R.id.checkBox2) {
            if (cb.isChecked())
                Toast.makeText(MainActivity.this,
                "Female Selected", Toast.LENGTH_SHORT).show();
            else
                Toast.makeText(MainActivity.this, "Female
                Unselected", Toast.LENGTH_SHORT).show();
        }
    }
}
```

MainActivity.java

Visual Programming

In the above mentioned code, the `isChecked()` method is used to check the checked or unchecked situation of the `CheckBox` view. Following Figure 3.10 is depicting the toasted message on the basis of event generated.

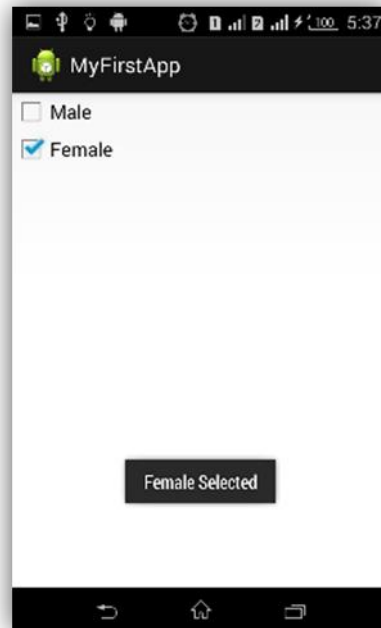


Figure 3.10 Activity is toasting a message on selection of a `CheckBox` view

3.2.6 ImageButton view

An `ImageButton` view is similar to the `Button` view except that it displays an image as an alternative of text. The image of `ImageButton` view can change during the different states of button (focused, selected, etc.). To add the image resource to surface of the button, you can either use the `android:src` attribute in the XML file or use the `setImageResource()` method to the Java file. Following code shows that how to add an `ImageButton` view in XML:

```
<?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="wrap_content"
    android:orientation="vertical">

    <ImageButton
        android:id="@+id/i_But"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:contentDescription="@string/IB"
        android:src="@drawable/image_selector"/>

</LinearLayout>
```

activity_main.xml

`ImageButton` view created by the above code will look as the Figure 3.11.

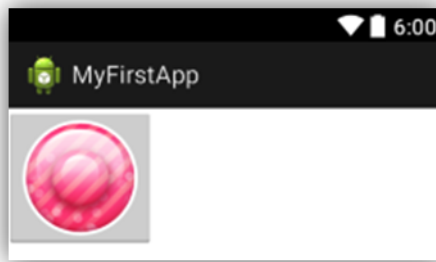


Figure 3.11: ImageButton view

In the above specified XML code, `android:src` attribute is used to provide the location of the image. Here, `image_selector` is drawable selector resource file which is a XML file that specifies the different images for different button states. Android will automatically switch the image according to the state of the button. XML code for XML drawable selector file is:

```
<?xmlversion="1.0"encoding="utf-8"?>
<selectorxmlns:android="http://schemas.android.com/apk/res/android">
  <itemandroid:state_pressed="true"
    android:drawable="@drawable/pink_button1" />
  <itemandroid:state_focused="true"
    android:drawable="@drawable/gold_button1" />
  <itemandroid:drawable="@drawable/pink_button1" />
</selector>
```

image_selector.xml

Selector file “`image_selector.xml`” will be saved in `res/drawable` folder. Images used in selector file (here, `pink_button1`, `pink_button2` and `gold_button1`) will also be saved in `res/drawable` folder.

To add the `onClick` listener to the `ImageButton`, use the following code:

```
packagecom.nikhil.myfirstapp;

importandroid.app.Activity;
importandroid.os.Bundle;
importandroid.view.View;
importandroid.view.View.OnClickListener;
importandroid.widget.ImageButton;
importandroid.widget.Toast;

publicclassMainActivityextends Activity implementsOnClickListener{
    protectedvoidonCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ImageButtonib=(ImageButton)findViewById(R.id.i_But);
        ib.setOnClickListener(this);
    }
    publicvoidonClick(View v) {
        Toast.makeText(MainActivity.this, "Image button is clicked!",
        Toast.LENGTH_LONG).show();
    }
}
```

MainActivity.java

Visual Programming

On the click of the ImageButton view the Activity window will toast a message on the screen and change the image according to the state of the ImageButton. It as shown in Figure 3.12.

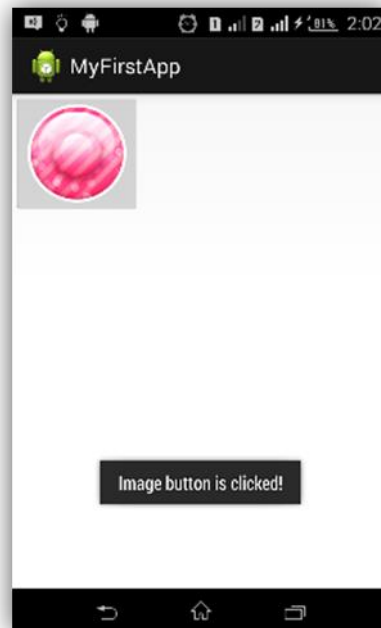


Figure 3.12: Activity is toasting a message on the click of ImageButton view

3.2.7 ToggleButton view

A ToggleButton view is a two state button which is generally used to change the settings between two states. By default it has a light indicators and ON and OFF text on it for both states. You can change the ON and OFF text by using android:textOn and android:textOff attributes of XML. XML code for adding the ToggleButton view is:

```
<?xmlversion="1.0"encoding="utf-8"?>

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:orientation="vertical">

<ToggleButton
android:id="@+id/t_Button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textOn="Silent On"
android:textOff="Silent Off"
android:text="@string/TB"/>

</LinearLayout>
```

activity_main.xml

After execution of code, the Toggle Button view will look like as the Figure 3.13.

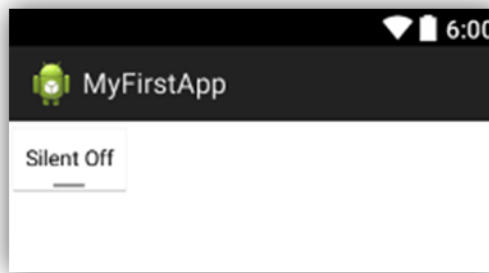


Figure 3.13: ToggleButton view

To add the `OnClickListener` to the `ToggleButton` view and to toast the messages on click, use the following Java code:

```
package com.nikhil.myfirstapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Toast;
import android.widget.ToggleButton;

public class MainActivity extends Activity implements OnClickListener {
    ToggleButton tb;

    Protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tb = (ToggleButton) findViewById(R.id.tb_Button);
        tb.setOnClickListener(this);
    }

    Public void onClick(View v) {
        if (tb.isChecked()) {
            Toast.makeText(MainActivity.this, "Silent Mode
is ON", Toast.LENGTH_LONG).show();
        }
        else {
            Toast.makeText(MainActivity.this, "Silent Mode
is OFF", Toast.LENGTH_LONG).show();
        }
    }
}
```

MainActivity.java

On the execution of the code, when you click on the `ToggleButton` view, the Activity will toast a message on the screen and change the state of the `ToggleButton` with a light indication. It is showing in Figure 3.14.



Figure 3.14: Activity is toasting a message and changing the state of the ToggleButton view with a light indication



Reading

There is one more kind of toggle button, called Switch, which is introduced in Android 4.0 (API Level-14). To know more about Switch view pursue the following link:

<http://developer.android.com/reference/android/widget/Switch.html>

Generally, when user enters the time and date in a text field in an application, then he / she doesn't know the exact format of time and date to be entered. Android provides two controls for picking the time and the date in exact format that is needed by the application. These controls are TimePicker view and DatePicker view. You can select each part of the control such that hour, minute and AM/PM for the time, and year, month and day for the date. These picker views are elaborated below:

3.2.8 TimePicker view

The TimePicker view provides you a mechanism of selecting hours and minutes of time by a single input control. You can select the time either in 24 Hours format or in AM / PM format.

Following XML code is used to declare the TimePicker view:

```
<?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="wrap_content"
  android:theme="@style/AppBaseTheme"
  android:orientation="vertical">
```

```

<TimePicker
    android:id="@+id/timePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button"/>
</LinearLayout>

```

activity_main.xml

TimePicker view declared in XML code will look as shown in Figure 3.15.

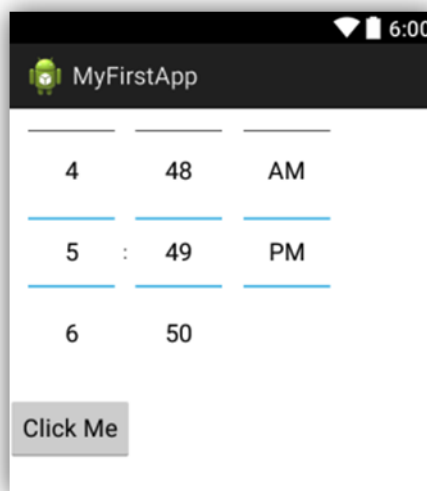


Figure 3.15 TimePicker View

To get the reference of TimePicker view and to display the selected time as toast message in the Activity, use the following Java code:

```

package com.nikhil.myfirstapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TimePicker;
import android.widget.Toast;

public class MainActivity extends Activity implements OnClickListener {
    Button b;
    TimePicker tp;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tp = (TimePicker) findViewById(R.id.timePicker1);
    }
}

```


Visual Programming

```
// To show the 24 Hour view of TimePicker view
// uncomment the following line of code.
// tp.setIs24HourView(true);
b = (Button) findViewById(R.id.button1);
b.setOnClickListener(this);
}
public void onClick(View v) {
    Toast.makeText(MainActivity.this, "Time Selected is :
"+tp.getCurrentHour()+" : "+
tp.getCurrentMinute(), Toast.LENGTH_LONG).show();
}
}
```

MainActivity.java

Here, in the Java code two methods `getCurrentHour()` and `getCurrentMinute()` are used to get the hour and minute respectively from the TimePicker view.

At the time of execution of the application, after selecting the required time when you will click on the “Click Me” button, a toast message will be displayed on the screen as shown in Figure 3.16.

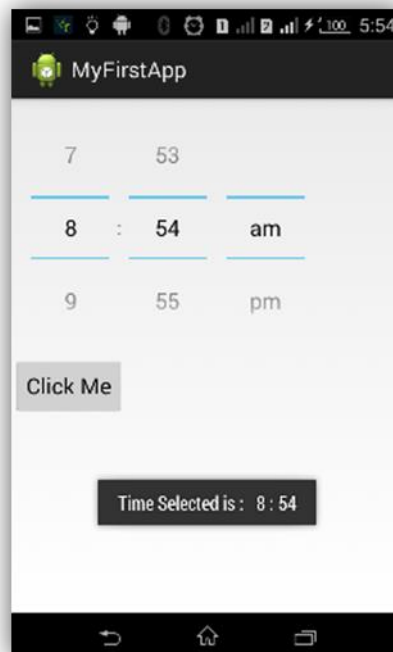


Figure 3.16: Time selected from TimePicker view

3.2.9 DatePicker view

The DatePicker view enables you to pick the date from an input control. It allows you to pick day, month and year separately. Following XML code is used to declare the DatePicker view:

```
<?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="wrap_content">
```

```

android:theme="@style/AppBaseTheme"
android:orientation="vertical">

<DatePicker
    android:id="@+id/datePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button"/>
</LinearLayout>

```

activity_main.xml

Graphically, the DatePicker view will look as shown in Figure 3.17.

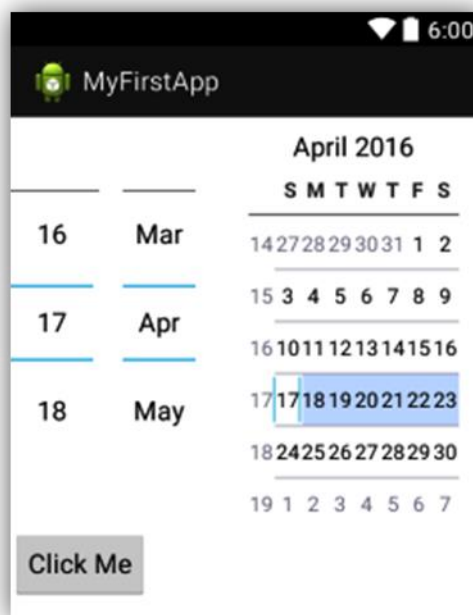


Figure 3.17: DatePicker view

To get the reference of DatePicker view in Activity's Java code, use the following code snippet:

```

package com.nikhil.myfirstapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.Toast;

public class MainActivity extends Activity implements OnClickListener {
    Button b;
    DatePicker dp;
}

```

Visual Programming

```
Protectedvoid onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    dp = (DatePicker) findViewById(R.id.datePicker1);  
    b = (Button) findViewById(R.id.button1);  
    b.setOnClickListener(this);  
}  
  
publicvoid onClick(View v) {  
    Toast.makeText(MainActivity.this, "My birth date is : " +  
    dp.getDayOfMonth() + " / " + (dp.getMonth()+1) + " / " + dp.getYear(),  
    Toast.LENGTH_LONG).show();  
}  
}
```

MainActivity.java

Here, three methods `getDayOfMonth()`, `getMonth()` and `getYear()` are used to get the day, month and year respectively from the `DatePicker` view. Remember, for January, `getMonth()` method returns value 0. So, we add value 1 to it to show the exact month.

When you will run the app, after selecting the date from `DatePicker` view, the activity will toast a message as shown in Figure 3.18.



Figure 3.18 Date selected from `DatePicker` view

3.2.10 Spinner view

The `Spinner` view is an input control that provides you a drop-down list to select an item from a set of items. Generally, it shows a default value but you can select a new one from the drop-down list. The XML code to add a `Spinner` view into the Activity, is 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="wrap_content"
android:theme="@style/AppBaseTheme"
android:orientation="vertical">
<Spinner
  android:id="@+id/spinner1"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"/>
</LinearLayout>

```

activity_main.xml

As you know, the Spinner view needs a list of items to be displayed. So you need an array of strings to show. You can do this by initializing a string array in the Java code or you can save the array of strings as a resource in the XML file. To create a string array as resource, create a string array in the strings.xml file by using <string-array> element as shown in following code.

```

<?xmlversion="1.0"encoding="utf-8"?>
<resources>
<stringname="app_name">MyFirstApp</string>
<stringname="hello_world">Hello world!</string>
<stringname="action_settings">Settings</string>
<string-arrayname="fruit_names">
  <item>Green Apple</item>
  <item>Yellow Banana</item>
  <item>Yellow Mango</item>
  <item>Red Apple</item>
  <item>Green Banana</item>
  <item>Green Mango</item>
  <item>Green Papaya</item>
  <item>Yellow Papaya</item>
  <item>Green Peach</item>
  <item>Green Guava</item>
</string-array>
</resources>

```

strings.xml

To use the Spinner in the Java code and to create the event on selection of the item, use the following code snippet.

```

package com.nikhil.myfirstapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;

public class MainActivity extends Activity
    implements OnItemSelectedListener {
    Spinner spinner;
    String fruits[];
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}

```

Visual Programming

```
setContentView(R.layout.activity_main);
fruits =
getResources().getStringArray(R.array.fruit_names);
spinner = (Spinner) findViewById(R.id.spinner1);
ArrayAdapter<String>adapter =
new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_dropdown_item, fruits);
spinner.setAdapter(adapter);
spinner.setOnItemSelectedListener(this);
}
public void onItemSelected(AdapterView<?> parent, View
view, int position, long id) {
    int i = parent.getSelectedItemPosition();
    Toast.makeText(MainActivity.this, "You have selected " +
fruits[i], Toast.LENGTH_LONG).show();
}
public void onNothingSelected(AdapterView<?> parent) {
}
}
```

MainActivity.java

You can fetch the names of fruits programmatically from the strings.xml by using `getStringArray()` method:

```
fruits = getResources().getStringArray(R.array.fruit_names);
```

Here, you will override the `onItemSelected()` method of `OnItemSelectedListener` interface to create the functionality for selecting an item from the Spinner view.

Following Figure 3.20.1 and Figure 3.20.2 is showing the execution of the application.



Figure 3.20.1: Drop-Down list in Spinner view



Figure 3.20.2: Selecting an item from Spinner view

3.3 Layouts

A layout is used to define the arrangement of input controls (called views) on the screen for a user interface (UI). This arrangement can

be either for an Activity or an app widget. There are two ways of defining the layout for your Android application:

- By declaring the UI elements in XML or
- By creating the View or ViewGroup objects programmatically at run time in Java.

You are free to use either or both methods for declaring and managing the UI. But it is convenient to design the layout in XML and keep application logic separate in Java. When you design the layout in XML, Android allows you to get preview of your UI design in Layout tab. Keeping the UI separate from your application code enables you to modify the design without modifying and compiling the Java source code. Moreover, XML makes it easier to visualize the UI structure; that makes debugging easier. In Android, XML has almost same names vocabulary for declaring the UI elements and attributes as used for classes and methods in Java.

Generally, in XML, all views and view groups are declared inside the Layout element. For example,

```
<?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="wrap_content"
android:theme="@style/AppBaseTheme"
android:orientation="vertical">
<Button>..... </Button>
<EditText>.....</EditText>
.
.
.
</LinearLayout>
```

activity_main.xml

Remember that, in Android, layout file (or any xml file) is stored as resource. Following code snippet can be used to retrieve the resource file in Java code:

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

When an Activity starts, onCreate() method is called. Here, setContentView() method is used to load and render the layout resource file on the screen.

Some of the layouts are described below:

3.3.1 Linear Layout

A LinearLayout is used to arrange the graphical components on screen vertically or horizontally. For setting the direction of rendering the components horizontally or vertically, use the android:orientation

Visual Programming

attribute. A LinearLayout can contain other layouts or vice-versa. Other nested layouts can further arrange their components accordingly.

A LinearLayout has many attributes that can be applied on it. Following table is describing the distinct attributes of LinearLayout:

Table 3.2 XML attributes of LinearLayout

Attribute	Description
android: orientation	This attribute is used to set the orientation of rendering of the graphical components on the screen.
android: gravity	It specifies that how child elements will be positioned.
android: layout_weight	It specifies that how much additional space is to be assigned to its child view.
android: background	It is used to set the background of the layout (ViewGroup) or view. You can specify the path of any image or a color.
android: layout_width	It specifies the width of the View or ViewGroup.
android: layout_height	It specifies the height of the View or ViewGroup.
android: layout_marginTop	It is used to specify the space above of the View or ViewGroup.
android: layout_marginBottom	It is used to specify the space below of the View or ViewGroup.
android: layout_marginStart	It is used to specify the space before of the View or ViewGroup.
android: layout_marginEnd	It is used to specify the space after of the View or ViewGroup.
android: id	This attribute allows you to specify the unique id for the layout. This will help you to identify the layout in many layouts while adding it in the Activity class.
adroid: layout_x	It is used to specify the X-Coordinate of the View or ViewGroup.
adroid: layout_y	It is used to specify the Y-Coordinate of the View or ViewGroup.

Some of the above mentioned attributes are common for different layouts and views.

Following XML code snippet is showing the linear layout applied on two TextView views, two EditText views and a Button view. The orientation of layout is vertical.

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

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:theme="@style/AppBaseTheme"
    android:orientation="vertical">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/email"/>

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textEmailAddress">
        <requestFocus/>
    </EditText>

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pass"/>

    <EditText
        android:id="@+id/editText2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPassword"/>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/login"/>
</LinearLayout>

```

activity_main.xml

The graphical view of the above mentioned XML code will be as shown in Figure 3.21.

Visual Programming

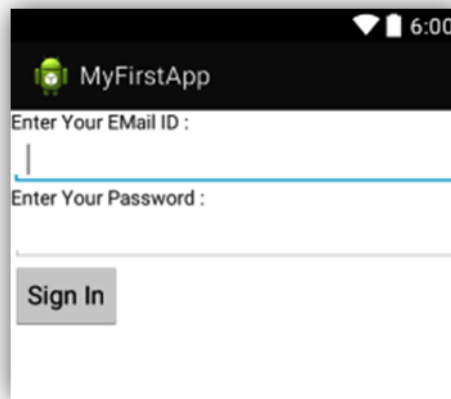


Figure 3.21: Linear Layout

Here, for the width and the height, two constants `match_parent` and `wrap_content` are used which are discussed in the section 3.4.

3.3.2 Relative Layout

In a `RelativeLayout`, child views are displayed in relative positions of other views or the parent (i.e. screen). For example, if you have added an `EditText` view, then you can add a `Button` view in either left, right, top or bottom of the `EditText` view. In other words, you can add a view in relation to the previously added views. You can also arrange the view in relation to the parent (i.e. screen) such as left, right or bottom of the screen. A `RelativeLayout` has many attributes; some of them are common to `LinearLayout`.

Important thing to remember is that the parent of any `View` or `ViewGroup` is the container of it. This container can be either `ViewGroup` (`Layout`) or the `Screen`.

Following table describes some attributes of `RelativeLayout`.

Table 3.3 XML attributes of `RelativeLayout`

Attribute	Description
<code>android:layout_alignParentTop</code>	It is used to align the view at the parent's top.
<code>android:layout_alignParentBelow</code>	It is used to align the view at the parent's below.
<code>android:layout_alignParentLeft</code>	It is used to align the view at the parent's left.
<code>android:layout_alignParentRight</code>	It is used to align the view at the parent's Right.
<code>android:layout_alignLeft</code>	It is used to align the view to the right of the other view.
<code>android:layout_alignRight</code>	It is used to align the view in right of the other view.
<code>android:layout_alignBottom</code>	It is used to align the view to the bottom of the other view.

android:layout_alignTop	It is used to align the view to the top of the other view.
android: id	This attribute allows you to specify the unique id for the layout.
Android:layout_centerHorizontal	It is used to place the view horizontally in the centre of the layout.
Android:layout_centerInParent	It is used to place the view in the centre of the parent view in the layout.
Android:layout_centerVertical	It is used to place the view vertically in the centre of the layout.
android:layout_toRightOf	It is used to position a view to the right of the already placed view.
android:layout_toLeftOf	It is used to position a view to the left of the already placed view.
android:layout_below	It is used to position a view to the below of the already placed view.
android:layout_above	It is used to position a view to the above of the already placed view.

Following XML code snippet is showing the use of RelativeLayout:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/RelativeLayout1"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android:theme="@style/AppBaseTheme">
<TextView
android:id="@+id/textView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentStart="true"
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true"
android:layout_marginTop="10dp"
android:layout_marginLeft="5dp"
android:layout_marginStart="5dp"
android:text="@string/mobileno" />
<EditText
android:id="@+id/editText1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentRight="true"
android:layout_alignParentEnd="true"
android:layout_marginTop="10dp"
android:ems="10"
```

Visual Programming

```
android:inputType="number">
<requestFocus/>
</EditText>
<Button
android:id="@+id/button1"
android:layout_width="90dp"
android:layout_height="wrap_content"
android:layout_below="@+id/editText1"
android:layout_marginTop="20dp"
android:text="@string/cancel"/>
<Button
android:id="@+id/button2"
android:layout_width="90dp"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:layout_alignBottom="@+id/button1"
android:text="@string/reset"/>
<Button
android:id="@+id/button3"
android:layout_width="90dp"
android:layout_height="wrap_content"
android:layout_alignBottom="@+id/button2"
android:layout_alignParentRight="true"
android:layout_alignParentEnd="true"
android:text="@string/ok"/>
</RelativeLayout>
```

activity_main.xml

The layout of the components in RelativeLayout in above mentioned XML code will look like Figure 3.22.

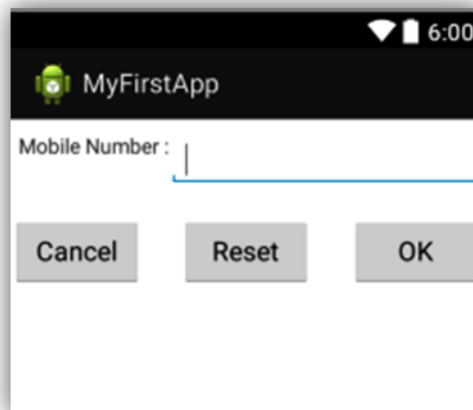


Figure 3.22: RelativeLayout

In the above code the `<requestFocus>` element is used to focus the cursor or control by default on the view at the time of loading the activity.



Reading

Many more layouts are there to be used in Android development such as Frame Layout, ListView layout, GridView layout, Table Layout, etc. To know more about them, follow the following link:

<http://developer.android.com/guide/topics/ui/declaring-layout.html>

<http://developer.android.com/reference/android/view/ViewGroup.html>

Note that, a layout gives you a bundle of functionality related to it. You don't need to get worried about a series of functions that layouts provides automatically. For example, if you are using a WebView, then there is no need of managing the connection, to get worried about the rendering of web page components, etc. It'll handle these things on its own.

3.4 Measurement Units

For specify the size of different components (such as Views, GroupViews or text) on the Android user interface, there are some measurement units:

1. **dp (Density Independent Pixel):** One inch of physical screen of the device is equivalent to 160 dp. This measurement unit is recommended for specifying the size of Views in layout. It is also represented as *dip*. It adjusts the size of the Views accordingly on screens of different density. (Density of the screen is equal to total number of pixels in per square inch of the screen.)
2. **sp (Scale Independent Pixel):** It is similar to dp i.e. one inch of physical screen of device is equal to 160 sp. It is recommended to specify the font size. It is different from the dp as it remembers the font size preferences of the user while dp does not. It can also adjust the font size of the text according to the different screens of different densities.
3. **pt (Point):** One inch of the physical screen is equals to 72 pt. It is not independent of the density of the screen.
4. **px (Pixel):** One px represents the one pixel of the screen. It is not recommended because the Views cannot be rendered accurately on the screens of different sizes.



Reading

To get more technical details of measurement units, pursue the following link:

<https://www.google.com/design/spec/layout/units-measurements.html#>

There are two predefined constants which are used to assign the size to the graphical components:

1. **match_parent:** This is a predefined constant which is used to define the size of a View or ViewGroup as big as its parent. match_parent is formerly known as fill_parent. It is renamed as match_parent after API Level 8 or higher. Although fill_parent is

Visual Programming

still in use but not recommended. Using a `match_parent` is called minus padding.

2. ***wrap_content***: This is also a predefined constant which is used to define the size of a View or a ViewGroup to be just big enough to surround its content. Using a `wrap_content` is called plus padding.

Unit summary



Summary

In this unit you learned about different kind of Views such as Button view, TextView view, EditText view, etc. Further, you have learned to design different layouts such as Linear layout, Relative layout and WebView layout. You have also learned to change the position, size and orientation of views and layouts, to handle the different events and listeners related to different views. This unit also gave you a brief introduction of different measurement units.

Assignment



Assignment

Q1. How a View is different from a ViewGroup? Explain.

Answer: Generally, an Activity includes several views and view groups in different styles and arrangements.

A view is a GUI component that user use to interact with the screen. Views are also called the input controls. Views are the interactive components. All views are derive from “android.view.View” class. Commonly used Views are TextView, Button, EditText, ToggleButton, etc.

A view group is also a GUI component that holds other views and view groups to describe the layout of the interface on the screen. All view groups are derive from “android.view.ViewGroup” class. View groups are also called the layouts. Commonly used ViewGroups are LinearLayout, RelativeLayout, etc. View groups are just like container that is used to manage the order and appearance of other views and view groups.

Q2. Explain the role of a Spinner view.

Answer: The Spinner view is an input control that provides you a

Visual Programming

drop-down list to select an item from a set of items. Generally, it shows a default value but you can select a new one from the drop-down list. The Spinner view needs a list of items to be displayed from an array of strings. You can do this by initializing a string array in the Java code or you can save the array of strings as a resource in the XML file. Generally, an Adapter class object provides the values to the drop down list of a Spinner.

Q3. Describe all measurement units and their specification.

Answer: For specify the size of different components (such as Views, GroupViews or text) on the Android user interface, there are some measurement units which are as follows:

dp (Density Independent Pixel) :One inch of physical screen of the device is equivalent to 160 dp. This measurement unit is recommended for specifying the size of Views in layout. It is also represented as dip.

sp (Scale Independent Pixel): It is similar to dp i.e. one inch of physical screen of device is equal to 160 sp. It is recommended to specify the font size.

pt (Point): One inch of the physical screen is equals to 72 pt. It is not independent of the density of the screen.

px (Pixel): One px represents the one pixel of the screen. It is not recommended because the Views cannot be rendered accurately on screens of different sizes.

There are two predefined constants **match_parent** and **wrap_content** which are used to assign the size to the graphical components. The **match_parent** is a predefined constant which is used to define the size of a View or ViewGroup as big as its parent, it was formerly known as **fill_parent**. The **wrap_content** is also a predefined constant which is used to define the size of a View or a ViewGroup to be just big enough to surround its content.

Q4. Explain the WebView Layout.

Answer:A WebView is layout is used to display the web pages in

the Activity of an application. It uses an inbuilt tool called WebKit to render the web pages or any online content in the Activity screen. The WebView allows you to navigate the history, carry out the text search, zoom in and zoom out the web page, etc. Remember that, WebView does not provide you the full features of a web browser such as navigation bar, address bar etc. It is just used to deliver a web page as a part of your application. There can be many situations where you need a WebView. One situation can be when you might need to update the information again and again (such as the licence agreements). Another situation can be when you need an internet to retrieve the information (such as the email).

Assessment



Assessment

Problem1. What is the difference between Linear Layout and Relative Layout? Explain.

Problem 2. Describe the role of AdapterView class in Spinner view.

Problem 3. Write the XML code for following layout in portrait view using Relative Layout:

Problem 4. Write an app for calculating the multiplication of two integers. You must read integers from two text fields (EditText view) and post the answer to label (TextView view) on the click of a button (Button view). This all must be done in a single Activity.

(*Hint:* Use the `getText()` to read a value from the EditText view and `setText()` method to put the value to the TextView view. Don't forget to convert the text to the integer while reading the values from text field and vice-versa when putting the values to the label.)

Problem 5. Write an app to calculate the percentage on the click of different radio buttons in a radio group. Input the value in the EditText view and display the answer in the text view. Remember that each radio button represents different percentage. For example, there can be three

Visual Programming

radio buttons for three different percentage such as 15%, 25% and 35%.

Unit 4

Activity

4.1 Introduction

In the last unit you have learned to design the user interface. In this unit you will learn in-depth about the Fragment and the Activity. Typically, an application has one or more activities in it. The key objective of an Activity is to provide an interface through which a user can communicate to the device. During its life span, an Activity goes through various stages of its life, called Activity life cycle. To understand the working of an application, you must understand the lifecycle of the Activity and different call-back methods of Activity's lifecycle. Although, you have been exposed to the concept of Activity in previous units, this unit you will learn more about how Activities work.

Along with the functionality of the application, look and feel of the application can also make a difference. You can change look and feel of your application with the help of Themes and Styles. In this unit, you will also learn about the themes and styles.

Upon completion of this unit you will be able to:



Outcomes

- *Get* detailed information of an Activity.
- *Understand* the Activity Lifecycle and its call-back methods.
- *Know* the use of Activity's back stack.
- *Define* Themes and Styles.
- *Apply* Themes and Styles.
-



Terminology

Default Activity: An Activity is called the default Activity or launcher Activity if it is the first Activity that is launched when an application starts.

Stack: A Stack is a data structure that is used to reverse the order of the elements (In your case, elements can be an Activity or a Fragment). It means it follows the "Last In First Out" approach for processing the elements; i.e. the element that is inserted in the last in the stack, will be the first that comes out of the stack.

Lifecycle: The lifecycle of a component is the series of

Visual Programming

different stages through which the component passes in its whole life (from its creation to its destruction).

Inheritance:	Inheritance is feature of Object Oriented programming languages which can be defined as the process where one class obtains the features of another class and add-on its features too.
Overriding:	Method overriding is a feature of Object Oriented programming languages in which a subclass re-implements the methods of the superclass.
State of the Activity:	The state of an Activity defines the status of the activity with its behaviour and visibility. Generally an Activity can be in resumed, paused or stopped states.
Call-back methods:	During the lifecycle of an Activity and a Fragment, the Android system calls a set of core methods called call-back methods.
Pixel:	A Pixel is the smallest displayable unit of a screen. A pixel stands for Picture Element.
Resolution:	Total number of pixels on the screen is called the Resolution of the screen. As resolution increases, the quality of display increases. Total number of pixels in per square unit of screen is called the density of the screen.

4.2 Activity

An Activity is an application component that provides you an interface to perform the operations. More simply, you can say that an Activity is a screen which is presented to the user by an application. An Activity can be a dialler screen, Google map screen, etc. When you open your application the very first screen that appear at front of you, is called a default Activity (or Main Activity). There can be more than one loosely coupled Activities in your application. Generally, as the complexity of your application increases, the need of number of Activities increases proportionally.

An Activity can start another Activity to perform some actions. If it does so then system stops the current Activity and preserves it in a stack (called Back Stack). When a new Activity starts, system run it on to the top of the stack and takes the user's focus. When a user tap on the back button, then the current Activity pops up from the stack and relinquish the user's focus and gets destroyed; then previous Activity comes into the focus. You can understand this explanation with the help of a diagram shown in Figure 4.1.

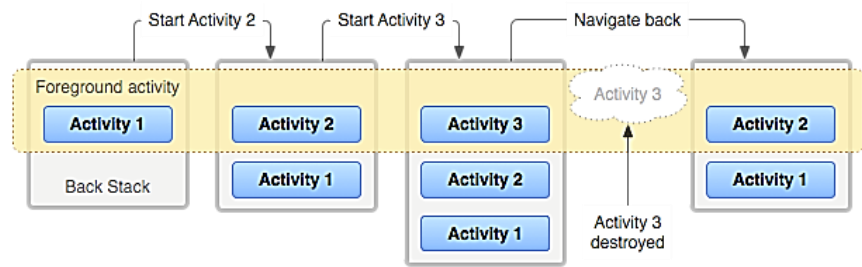


Figure 4.1: Implementation of Back Stack for multiple Activities of an app
(Source: <http://developer.android.com/guide/components/tasks-and-back-stack.html>)

4.2.1 Creating an Activity:

In unit 3, you have seen some examples of designing the user interface (or Activity) and performed some click events in the Activity class. Now, it is the time to explore more about the Activity class. To understand it in detail consider the following example.

```
package com.nikhil.myfirstapp;

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

public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

MainActivity.java

Here, in the code, the very first line specifies the package name where your class MainActivity is to be created. Next two lines are there to import the library classes, Activity and Bundle, in the program to use their functionality in your code. Activity class is included here to inherit its features in your MainActivity class. That is, your class must inherit the Activity class to use its methods in your defined class. The first method that is called when your Activity does load into the memory, is onCreate() method. This is an overridden method of Activity class that takes an object of Bundle class to load the previous state of the Activity. Notably, object of the Bundle class generally stores the state of the Activity.

To assign the graphical components (such as Views, Layouts, etc.) to the Activity and render them on to the screen, use the setContentView() method of Activity class. It takes the reference of the graphical component as an argument. For example, to add the button to the Activity append the following lines of code (shown in Bold) to the onCreate() method in the previous example.

```
Protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Button b= new Button(this);
    setContentView(b);
}
```

Visual Programming

But, generally, you add the graphical components in an XML layout file. This will make it easy to manage complex arrangement of graphical components in the Activity. Also, using an XML code is an easy and structured task. You have learned a lot about the layout file in the unit 3. If `activity_main.xml` is the name of the layout file, then it would be included in the Java code using following code (shown in Bold):

```
Protectedvoid onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
}
```

Remember that, an Activity should be registered in the Manifest file after its creation. To, register an Activity in the Manifest file use the `<activity>` tag within `<application>` tag. The `<activity>` tag consists of many attributes such as name, label, themes and permissions, to add the properties of the Activity. If you don't specify the `<activity>` tag in your Manifest file then the Activity will not display on the screen, and system will show a run time error when you will run the application. An `<activity>` tag contains an `<intent-filter>` tag to specify the Intents that can start the Activity. Intents are described in detail the upcoming unit 5. Example is given below,

```
<activity  
    android:name=".MainActivity"  
    android:label="@string/app_name">  
    <intent-filter>  
        <action android:name="android.intent.action.MAIN"/>  
        <category android:name="android.intent.category.LAUNCHER"/>  
    </intent-filter>  
</activity>
```



Note it!

An Activity can start another Activity or application. An Activity can accomplish this job with the help of a special feature of Android, called Intent. Intent will be discussed detail in the section 5.2 of unit 5.

To have proper understanding of the working of an Activity you need to first know lifecycle of the Activity.

4.2.2 Activity Lifecycle:

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

called for each Activity. The lifecycle of an Activity is managed by the Android run time system.

Generally an Activity can remain in following three states:

1. *Resumed*: This state is also called running state. In this state an Activity remains in the foreground of the screen and has the focus of the user.
2. *Paused*: When another Activity comes at the foreground of the Activity, then Activity exist in paused state. In this state Activity remains alive and partially visible but doesn't have focus of the user. When an Activity remains alive, it means the object of the Activity still holds in the memory and carries its involvement with the window manager.
3. *Stopped*: When an Activity goes into the background due to another Activity then it remains into the stopped state. In this state, while an Activity remains alive but it relinquishes the involvement with window manager. This means it is no longer visible to the user in the foreground or background.

Figures 4.2.1 and Figure 4.2.2 is depicting the state diagrams to represent the different stages of the Activity life cycle with different call-back methods:

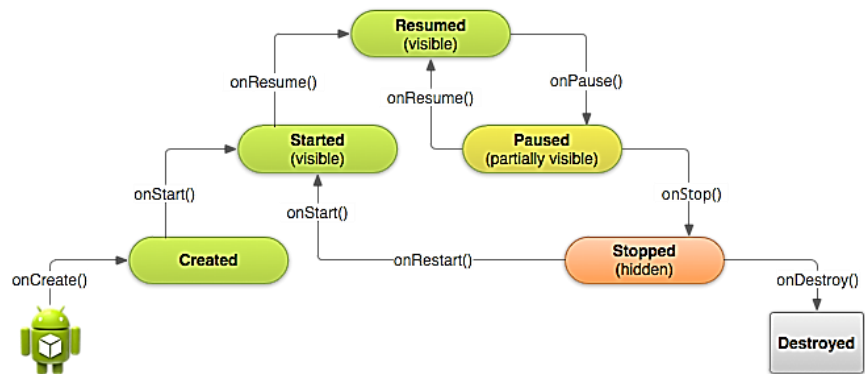


Figure 4.2.1: A simplified illustration of Activity Lifecycle
(Source: <http://developer.android.com/training/basics/activity-lifecycle/starting.html>)

Visual Programming

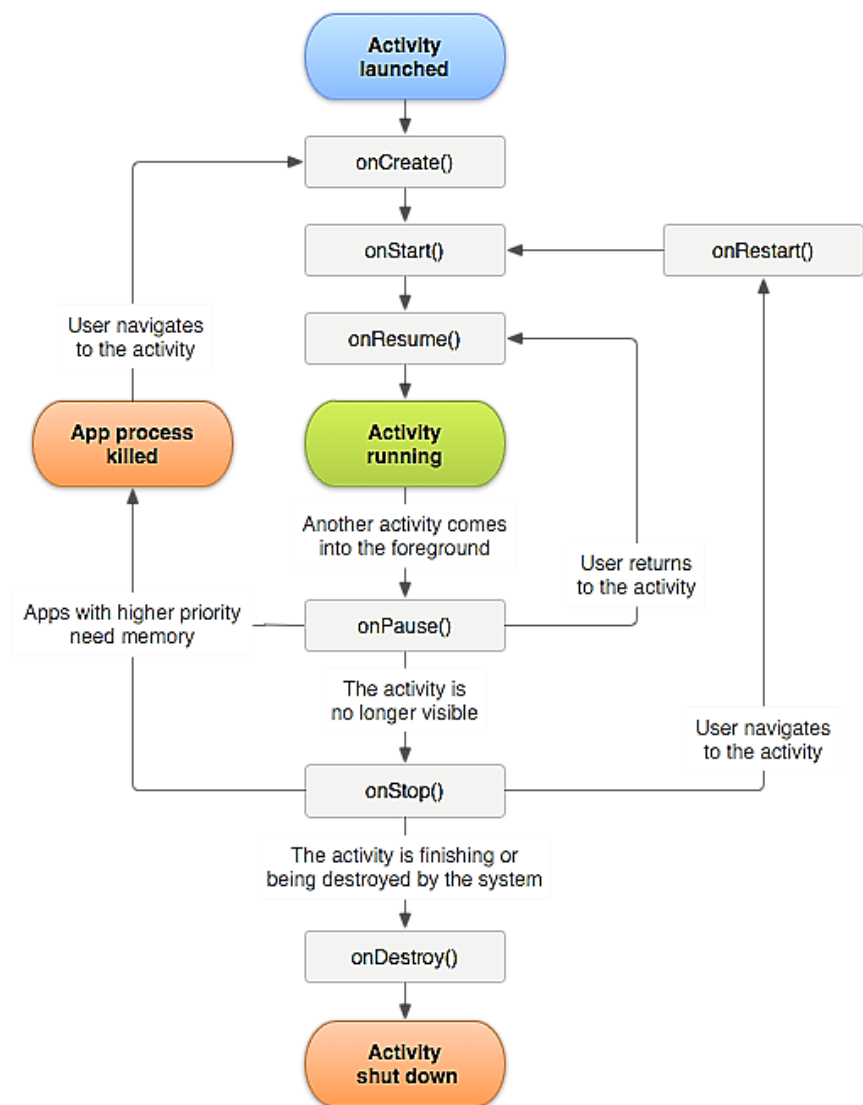


Figure 4.2.2: Activity Lifecycle

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

Following table is showing each lifecycle call-back method with details such as name of the method, description of the method, what method is called after the specified method, method is killable or not, etc.

Table 4.1: A summary of the activity lifecycle's call-back methods

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

Method	Description	Killable after?	Next
onCreate()	Called when the activity is first created. This is where you should do all of your normal static set up such as create views, bind data to lists, and so on.	No	onStart()
onRestart()	Called after the activity has been stopped, just prior to it	No	onStart()

	being started again.		
onStart()	Called just before the activity becomes visible to the user.	No	onResume() or onStop()
onResume()	Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack.	No	onPause()
onPause()	Called when the system is about to start resuming another activity.	Yes	onResume() or onStop()
onStop()	Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it.	Yes	onRestart() or onDestroy()
onDestroy()	Called before the activity is destroyed. This is the final call that the activity will receive	Yes	<i>Nothing</i>

In the table column “Killable after?” shows that system can kill the Activity after execution of specified method in the row. There are three call-back methods of an Activity’s lifecycle after that an Activity can be killed: onPause(), onStop() and onDestroy(). In the shortage of memory, the Android runtime system can kill the Activity in the paused state without calling the onStop() and onDestroy() methods.

Following skeleton code is showing each basic call-back method of Activity:

```
package com.nikhil.myfirstapp;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Toast;

public class MainActivity extends Activity {

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast.makeText(MainActivity.this, "onCreate() is called.",
            Toast.LENGTH_SHORT).show();
    }

    protected void onStart() {
        super.onStart();
    }
}
```


Visual Programming

```
        Toast.makeText(MainActivity.this, "onStart() is called.",
Toast.LENGTH_SHORT).show();
    }
    protected void onRestart() {
        super.onRestart();
        Toast.makeText(MainActivity.this, "onRestart() is called.",
Toast.LENGTH_SHORT).show();
    }
    protected void onResume() {
        super.onResume();
        Toast.makeText(MainActivity.this, "onResume() is called.",
Toast.LENGTH_SHORT).show();
    }
    protected void onPause() {
        super.onPause();
        Toast.makeText(MainActivity.this, "onPause() is called.",
Toast.LENGTH_SHORT).show();
    }
    protected void onStop() {
        super.onStop();
        Toast.makeText(MainActivity.this, "onStop() is called.",
Toast.LENGTH_SHORT).show();
    }
    protected void onDestroy() {
        super.onDestroy();
        Toast.makeText(MainActivity.this, "onDestroy() is called.",
Toast.LENGTH_SHORT).show();
    }
}
```

MainActivity.java

When you will run the above mentioned code, a corresponding toast message will be displayed on the screen whenever a call-back method will be called. By executing this code you can make an idea in your mind about the order of execution of the methods. For example, whenever you will start the application, on the execution of call-back methods onCreate(), onStart() and onResume(), three consecutive toast message will be showing on the screen respectively. And when you will close the application when it is running, three consecutive toast message will be showing on the screen respectively, on the execution of call-back methods onPause(), onStop() and onDestroy(). Type of the toast messages will be depending on your actions with the application.

Saving the state of the Activity: Whenever an Activity resumes after calling the onPause() or onStop() methods, the Activity retains its previous state. This is true but question arises, how does it happens? This happens as the object of the Activity pushed into the stack when onStop() method is called. On the calling of onResume() method this object pops up from the stack and retains its position.

How can Android system save the state of the Activity if the Activity destroyed using onDestroy() method? This happens because Android can preserve the state of the Activity in an object of Bundle class by calling the onSaveInstanceState() method before calling the onDestroy() method. You can save the information of state in the Bundle class object as name-

value pairs, using methods like `putInt()` and `putString()`. After destroying the Activity by system if the user navigate back the Activity, then the system get the information by calling the `onCreate()` method or `onRestoreInstanceState()` method. These methods takes the previously stored object of Bundle class as an argument to recover the state information. In cases where Activity is created first time, the Bundle class object has null value because there is no stored information about state. Figure 4.3 is showing the whole mechanism of preserving and recovering the state information.

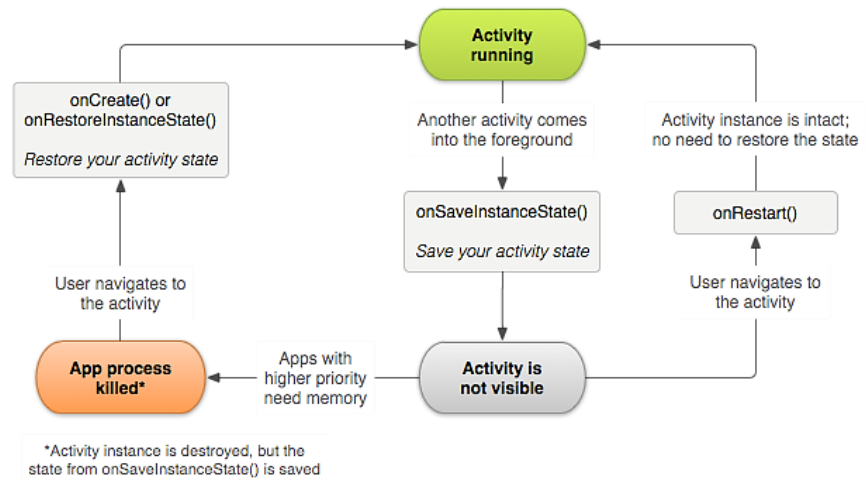


Figure 4.3: Saving the Activity state by using `onSaveInstanceState()` method
(Source: <http://developer.android.com/guide/components/activities.html>)



Reading

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

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

4.3 Using Themes and Styles:

A style is a collection of different attributes which are used to specify appearance and presentation of a View. A style can be a font size, font color, background color, height, padding, etc. A separate XML resource file is used to store the styles. A style is something like the Cascading Style Sheets used for web page designing.

4.3.1 Defining the Styles and the Themes:

You can use the style attributes to a View distinctly. For example, you can use different style attributes inline to a `TextView` as following:

```

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#0011DD"
    android:typeface="serif"
    
```

```
android:text="Hello World!"/>
```

The look of TextView will be as Figure 4.4:

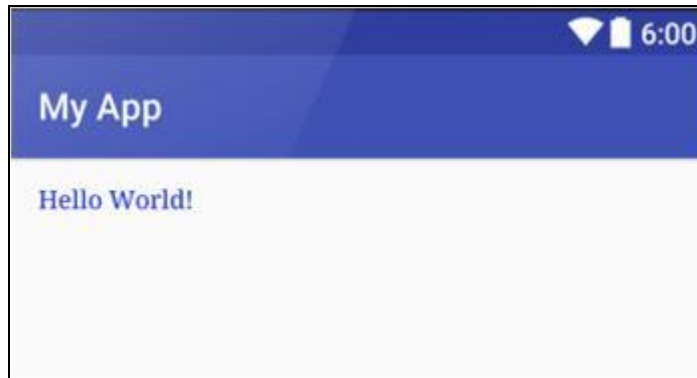


Figure 4.4: TextView after applying the style attributes

Suppose, you want to apply the same style attributes for many TextView views then this approach has a limitation. You need to have all the style attributes for each TextView inline. Solution is that you can save these attributes in a different resource file using a `<style>` tag and then use the style for the multiple TextView views as following:

Create an XML resource file in the `res/values/` folder and give that file an arbitrary name:

```
<?xmlversion="1.0"encoding="utf-8"?>
<resources>
<stylename="textView_style">
    <itemname="android:layout_width">fill_parent</item>
    <itemname="android:layout_height">wrap_content</item>
    <itemname="android:textColor">#0011DD</item>
    <itemname="android:typeface">sans</item>
</style>
</resources>
```

col_style.xml

And use the style attribute in the TextView tag as:

```
<TextView
style="@style/textView_style"
android:text="Hello World!"/>
```

This will produce the same result as Figure 4.4. Remember that, root element of the file must be `<resources>`.



Reading

To understand the different colour schemes and constants, explore the following two links:

<http://developer.android.com/reference/android/graphics/Color.html>

<http://developer.android.com/reference/android/R.color.html>

Theme: A Theme is a style that is applied to the whole Activity or whole application, rather than distinct Views. When you apply the style on the

entire activity then each view in the Activity will apply that style (if a view supports a particular attribute).

You can inherit the features and properties of a predefined style in your style. To inherit the properties, use the “parent” attribute of the <style> tag while you are defining your style. For example,

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<style name="new_style1"
parent="@android:style/TextAppearance.DeviceDefault">
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#0011DD</item>
    <item name="android:typeface">sans</item>
</style>
</resources>
```

Here, “android:style/TextAppearance.DeviceDefault” is a predefined theme (or style). The properties, which you specify in your file, will override the properties of parent theme.

When you want to override the properties of your own defined theme, then you will not use the parent attribute. You will inherit your own defined theme as follows:

```
<style name="new_style1.G">
    <item name="android:typeface">serif</item>
    <item name="android:textColor">#00FF00</item>
</style>
```

You can define this style in the same file or in a different file. You can continue inheriting like this as many times as you want.

4.3.2Applying the Styles and the Themes:

To apply a style to an individual view use the “style” attribute as follows:

```
<TextView
style="@style/textView_style"
android:text="Hello World!"/>
```

Here, style attribute is used without specifying the android namespace.

To apply the theme to whole application, specify the “android:theme” attribute in the <application> of AndroidManifest.xml file. For example,

```
<application android:theme="@style/new_style1">
```

And, if you want to apply the theme to an individual Activity then specify the “android:theme” attribute to the <activity> tag instead of <application> tag in the AndroidManifest.xml file.

```
<activity android:theme="@style/new_style1">
```



Activity

To understand the themes and styles with the help of this course's activity, pursue the following steps:

1. Create an application under com.nikhil.myapp package.
2. Write the following code in MainActivity.java file:

```
package com.nikhil.myapp;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Toast;

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

MainActivity.java

3. Write the following code in the strings.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">MyApp</string>
<string name="hello_world">
    A style is a collection of different attributes which are used to specify
    appearance and presentation of a View. A style can be a font size, font
    color, background color, height, padding, etc.
    A separate XML resource file is used to store the styles. A style is
    something like the Cascading Style Sheets used for web page designing.
</string>
</resources>
```

Strings.xml

4. Add a new XML resource file gehu_themes in the res/values/ folder:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<style name="graphic_style"
    parent="@android:style/TextAppearance.DeviceDefault.Large">
<item name="android:textColor">#008800</item>
<item name="android:textSize">20sp</item>
<item name="android:typeface">monospace</item>
</style>
</resources>
```

gehu_themes.xml

5. Write the following code in the activity_main.xml file:

```
<?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:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="com.nikhil.myapp.MainActivity">

```

```

<TextView
style="@style/graphic_style"
android:text="@string/hello_world"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true"/>
</RelativeLayout>

```

activity_main.xml

Now when you run the application and check the result, it should be like the following Figure 4.5.

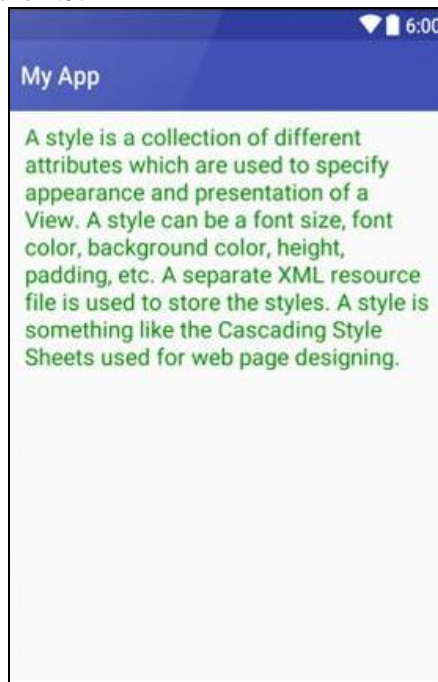


Figure 4.5: Output of the application created in Activity

Now, you will learn about a very interesting feature of the Android, called Fragment.

Unit summary



Summary

In this unit you learned about the working of the Activity, the Activity lifecycle and different call-back methods of it. You understood how an application uses the back stack. You learnt about defining and applying different styles and themes that makes your application more attractive.

Assignment



Assignment

Q1. Explain the different call-back methods of Activity Lifecycle.

Answer: Following call-back methods are used by Android runtime while running an Activity:

1. **onCreate():** Called when the activity is first created. This is where you should do all of your normal static set up such as create views, bind data to lists, and so on.
2. **onRestart():** Called after the activity has been stopped, just prior to it being started again.
3. **onStart():** Called just before the activity becomes visible to the user.
4. **onResume():** Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack.
5. **onPause():** Called when the system is about to start resuming another activity.
6. **onStop():** Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it.
7. **onDestroy():** Called before the activity is destroyed. This is the final call that the activity will receive

Q2. What is the difference between Styles and Themes? Explain.

Answer:A style is a collection of different attributes which are used to specify appearance and presentation of a View. A style can be a font size, font color, background color, height, padding, etc. A separate XML resource file is used to store the styles. A style is something like the Cascading Style Sheets used for web page designing.

A Theme is a style that is applied to the whole Activity or whole application, rather than distinct Views. When you apply the style on the entire activity then each view in the Activity will apply that style (if a view supports a particular attribute).

Assessment



Assessment

Problem1. Explain, how themes and styles are applied to an Activity.

Problem 2. Create an application to demonstrate the call-back methods of a Fragment lifecycle by displaying the toast messages on the changing of states.

Unit 5

Intents

5.1 Introduction

In previous units, you have learned to create and design the Activities. But rarely, a single Activity is enough for an Android application. An Android application is made up of multiple Activities, Services and Broadcast Receivers which are the core constituents of an Android application. It means there should be some kind of interaction between Activities. This is done through another important constituent of an Android app called Intent. Intent works like a glue between the different Activities to work together and gives the seamless performance to your application. Intents will be discussed thoroughly in this unit.

Other topics that will be covered in this unit are Menus and Dialog Boxes. Menus are used to provide the additional option which is generally not visible in the main user interface. And a dialog is a small pop-up window which is generally used to take an instant decision or to enter additional information. Both, menus and dialogs, make the user interface more interactive without consuming any substantial space on the screen.

Upon completion of this unit you will be able to:



Outcomes

- Use Intents.
- Link an Activity to another.
- Invoke inbuilt application.
- Define Intent-filters.
-



Terminology

Intent:	Intent is an object of Intent class that is used to start a new component or passing some data to another component.
App Component:	An application component can be an Activity, Service or Broadcast Receiver.
URI:	A URI (Uniform Resource Identifier) is a string that is used to identify a resource such as web browser, phone dialer or an Activity of Another app.
Intent-filter:	Intent filter stipulates the types of intents to which an Activity can respond. It specifies for the corresponding Activity to listen the specific types of actions of specific category.

Menu:	A Menu is a GUI control which is displayed in the form of a list of options or commands in the device's screen.
Dialog:	It is GUI control which is displayed in the form of small window having a title, action area and some buttons. It is used to communicate the information to the user or helps the user to take instantaneous decisions.
Bundle class:	A Bundle class object is used to wrap a set of strings in a parcel form.
Broadcast Receiver:	A Broadcast Receiver is an Android application component which is used to respond to the broadcasted messages of other applications.

5.2 Intent

Intent is an object that is used to pass the data or request to another application component (such as an Activity, a Service or a Broadcast Receiver). For example, it can help you to switch from one Activity to another by clicking on any UI component. An Intent object can be used:

- To start a new Activity.
- To pass the data to another Activity.
- To receive the result or data from another Activity.
- To start any background operation (i.e. Service).
- To bind a service with another component.
- To broadcast a message such as the ringing the phone or receiving an SMS (i.e. to start a Broadcast Receiver).

5.2.1 Types of Intents

Mainly, there are two types of intents: Explicit Intents and Implicit Intents.

Explicit Intents: Explicit Intents are used to call a specific component using fully qualified class name. It is used when you know which component you want to launch and don't want to give the free control to the user to choose a specific component to process the request. In abstract, it is used to start another component in your own application. Suppose, you have an application that has 2 activities: Activity A and activity B. You want to launch the activity B from activity A. In this case you define an explicit Intent in Activity A to target the Activity B and then use it to call the Activity B directly.

Implicit Intents: It is used when you have an idea of what you want to do, but you do not know which component should be launched. If you

Visual Programming

want to give the user an option to choose between a lists of components, then you must use Implicit Intents. Implicit Intents are sent to the Android system then it searches for all components which are registered for the specific action and the data type. If only one component is found, Android starts the component directly; for example, you have an application that uses the camera to take photos. One of the features of your application is that you give the user the possibility to send the photos he has taken. You do not know what kind of application a user has that can send photos, and you also want to give the user an option to choose which external application to be used if he / she have more than one application. In this case you should not use an explicit intent; instead of using the explicit intents, you should use an implicit Intent.

When you use an explicit Intent to start service or an activity, the system instantly starts the application component quantified in the object of Intent. But when you are using an implicit intent, then it is the responsibility of Android system to find the suitable component to start. The Android system does it by comparing the contents of the intent to the Intent Filters. Intent Filters are the expressions written under the <activity> element of the Manifest file to specify the type of Intent likely to be received by the component. The Android system starts the component if the intent matches an Intent Filter. If many Intent Filters are well-matched, the Android system shows a dialog on the screen from where a user can choose which application to use. It is shown in Figure 5.1.

When an explicit intent calls its target component the intent filters are not checked, on the other hand implicit intent calls its target component only if it matches to any of the component's filters.

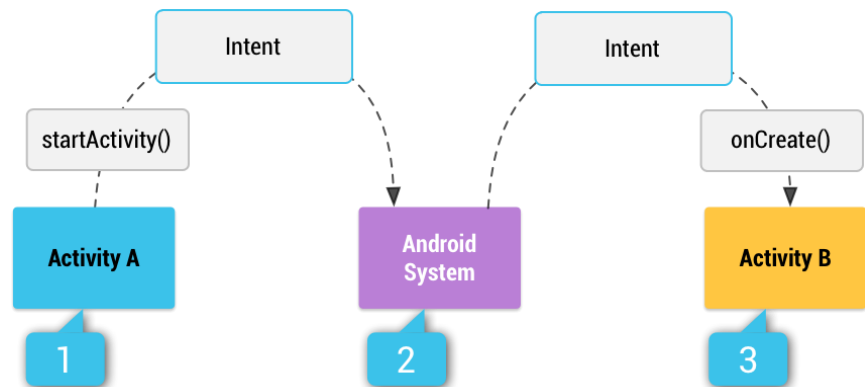


Figure 5.1: Delivery of an Implicit Intent via Android system to start an Activity
(Source: <http://developer.android.com/guide/components/intents-filters.html>)

5.2.2 Working with Intents

To work with the intents first you must understand the types of data that Intent can carry with it.

Information carried by Intents: An Intent object is received by the component as a bundle of information. The information can be used either to receive the Intent only or to take the decision by Android system

to decide which component should be start and what action to be performed. The information contained in the Intent object can be of the following types:

1. **Component Name:** It is the name of the target component. It can be a fully qualified path of the target component such as the Activity class name.
2. **Action:** An action is a string that is used by an Activity to specify an action to be perform. The Intent class has several action constants corresponding to different intents. Some of the common actions performed are ACTION_VIEW, ACTION_MAIN, ACTION_EDIT, etc. You can specify the action when creating the Intent object.
3. **Data:** It is a URI (Uniform Resource Locator) that references the data to be worked on. A URI is send in the pairing of action constant. The Data (URI) and the action pair defines the operation to be performed. For example, if the action is ACTION_DIAL then the URI “tel:123” is used to display the phone dialer with the given number 123 filled into it.
4. **Category:** It is a string that contains the extra information about the type of component that will handle the intent. For example, category CATEGORY_LAUNCHER is used to specify that the Activity is the launcher Activity; i.e. it is the first Activity to be displayed on the screen when the application is launched.
5. **Extras:** When you want to bind additional data (extra data) to the Intent object, putExtra() method is used. The data is wrapped in the object of the Bundle class in the form of key-value pairs.



Reading

For more details of the type of the information that can be contained by an Intent, explore the following link:

<http://developer.android.com/reference/android/content/Intent.html>

To understand the working of the Intents, first learn to link the Activities using Explicit Intents and Implicit Intents respectively.

5.2.2.1 Linking the Activities using Intents

As you know an Intent can be used either to start another component (such as an Activity) or to pass the data to another components. Generally, Activities are linked to each other with the help of an Intents. To understand this use the following examples.

1. **Explicit Intent to start another Activity:** For example, first create two activities in two different XML layout files; one is having a Button view and another is having a TextView view.

```
<LinearLayout
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"
```

Visual Programming

```
android:orientation="vertical"
tools:context="com.nikhil.explicit_intent.MainActivity">

<Button
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Start Activity"/>

</LinearLayout>
```

activity_main.xml

The output screen of the activity_main.xml file is shown in Figure 5.2.1.

```
<LinearLayout
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"
tools:context="com.nikhil.explicit_intent.SecondActivity">

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="You are welcomed in Second Activity."/>

</LinearLayout>
```

second.xml

The output screen of the activity_main.xml file is shown in Figure 5.2.2.

For the layout file second.xml, create the SecondActivity.java file.

```
package com.nikhil.explicit_intent;

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

public class SecondActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
    }
}
```

SecondActivity.java

For the main layout file activity_main.xml create the following java file ActivityMain.java. This java file has an Intent object that is used to start the activity SecondActivity which has a welcome message.

```
package com.nikhil.explicit_intent;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
```

```

import android.widget.Button;

public class MainActivity extends Activity {
    Button b1;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        b1=(Button) findViewById(R.id.button1);
        b1.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                Intent i = new
Intent(MainActivity.this,SecondActivity.class);
                startActivity(i);
            }
        });
    }
}

```

MainActivity.java

Whenever a new Activity is created, it needs to be registered in the Manifest file of the application.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.nikhil.explicit_intent"
android:versionCode="1"
android:versionName="1.0">

    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="23"/>
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".SecondActivity"></activity>
    </application>
</manifest>

```

AndroidManifest.xml

Visual Programming

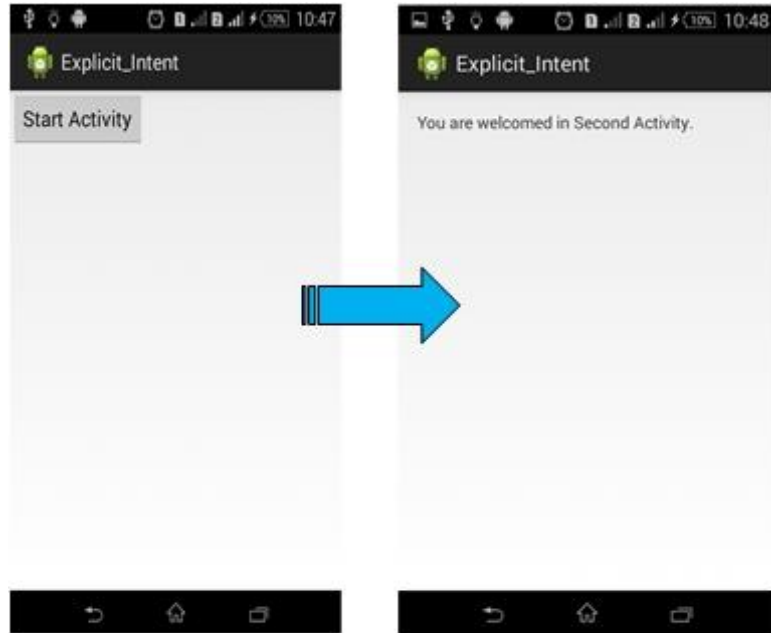


Figure 5.2.1: Main Activity

Figure 5.2.2: Second Activity

2. **Implicit Intent to start a web browser:** To understand the Implicit Intents, you will use the following code to start an inbuilt browser to open the specified web page. Remember, if there are more than one web browser in your device, then Android system will ask you to choose one.

The layout file of the Activity having only one button to start a web browser will be as follows:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/LinearLayout1"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical">

<Button
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/start_browser"/>

</LinearLayout>
```

activity_main.xml

Java code for the Activity will be as follows:

```
package com.nikhil.implicitintent;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
```

```

import android.view.View.OnClickListener;
import android.widget.Button;

publicclass MainActivity extends Activity {
    @Override
    protectedvoid onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button b1 = (Button) findViewById(R.id.button1);
        b1.setOnClickListener(new OnClickListener(){
            @Override
            publicvoid onClick(View arg0) {
                Intent i = new
                Intent(Intent.ACTION_VIEW,Uri.parse("https://www.facebook.com/
                dcaugepv"));
                startActivity(i);
            }
        });
    }
}

```

MainActivity.java

There will not be any changes in the Manifest file because you didn't include a new Activity in the application.

When you will run the above code, the output will be as shown in Figure 5.3.1 and Figure 5.3.2.



Figure 5.3.1: Main Activity



Figure 5.3.2: Web Browser showing a web page

5.2.2.2 Intent Filters

Intent Filter is a very important construct that you must learn to understand the working of Intents more profoundly. If an Intent is sent to the Android system, it will determine suitable applications for this Intent.

Visual Programming

If several components have been registered for this type of Intents, Android offers the user the choice to open one of them. This is due to Intent Filters. Intent filter stipulates the types of intents to which an Activity can respond. It specifies for the corresponding Activity to listen the specific types of actions of specific category. An Intent Filter declares the capabilities of a component. It specifies what an activity or service do and what types of broadcasts a Receiver can handle. It allows the corresponding component to receive Intents of the declared type. Intent Filters are typically defined via the AndroidManifest.xml file. For Broadcast Receiver it is also possible to define them in coding. An Intent Filter is defined by its category, action and data filters. It can also contain additional metadata.

If a component does not define an Intent filter, it can only be called by explicit Intents. There are two ways of defining Intent Filters; you can define Intent Filters in either your Manifest file or Broadcast Receiver. If you register the intent in Manifest file then Android registers the filter when your application gets installed. If you want to receive an intent at run time then use it for the Broadcast Receiver. To use the intent for Broadcast Receiver, you must define it at run time programmatically.

Unit summary



Summary

In this unit you learned about starting a new Activity from another Activity using Intents. Working of different types of Intents are also elaborated.

Assignment



Assignment

Q1. What are Intents? Differentiate between Explicit Intent and Implicit Intent with a suitable note.

Answer: Intent is an object that is used to pass the data or request to another application component (such as an Activity, a Service or a Broadcast Receiver).

Explicit Intents: Explicit Intents are used to call a specific component using fully qualified class name. When you know which component you want to launch and don't want to give the user a free control over which component to use. In abstract, it is used to start another component in your own application. For example, if you have an application that has 2 activities: Activity A and activity B. You want to launch activity B from activity A. In this case you define an explicit Intent in Activity A to target the Activity B and then use it to directly call it.

Implicit Intents: It is used when you have an idea of what you want to do, but you do not know which component should be launched. If you want to give the user an option to choose between a lists of components, then you must use Implicit Intents. Implicit Intents are send to the Android system then it searches for all components which are registered for the specific action and the data type. If only one component is found, Android starts the component directly; for example, you have an application that uses the camera to take photos. One of the features of your application is that you give the user the possibility to send the photos he has taken. You do not know what kind of application a user has that can send photos, and you also want to give the user an option to choose which external application to be used if he / she has more than one applications. In this case you would not use an explicit intent; instead of this you should use an implicit Intent.

Q2. What type of data, an Intent object can carry with it? Explain.

Answer:The information contained in the Intent object can be of the following:

1. **Component Name:** It is the name of the target component. It can be a fully qualified path of the target component such as the Activity class name.
2. **Action:** An action is a string that is used by an Activity to specify an action to be perform. The Intent class has several action constants corresponding to different intents. Some of the common actions performed are ACTION_VIEW, ACTION_MAIN, ACTION_EDIT, etc. You ca specify the action when creating the Intent object.
3. **Data:** It is a URI (Uniform Resource Locator) that references the data to be worked on. A URI is send in the pairing of action constant. The Data (URI) and the action pair defines the operation to be performed. For example, if the action is ACTION_DIAL then the URI tel:123 is used to display the phone dialer with the given number 123 filled into it.

Visual Programming

4. **Category:** It is a string that contains the extra information about the type of component that will handle the intent. For example, category CATEGORY_LAUNCHER is used to specify that the Activity is the launcher Activity; i.e. it is the first Activity to be displayed on the screen when the application is launched.
5. **Extras:** When you want to bind additional data (extra data) to the Intent object, putExtra() method is used. The data is wrapped in the object of the Bundle class in the form of key-value pairs.

Assessment



Assessment

Problem 1: Write an app to pass an integer (inputted in the text field by the user) from one Activity to another Activity on a click of a Button. On the target Activity print the factorial of the integer.

Problem 2: Write an app to start the inbuilt Camera on the click of a Button.

Problem 3. Write an app that is having two Activities. First Activity must have two EditText, a Button and a TextView. Second Activity must have two TextViews and a Button. User must input two numbers in both text fields of first Activity. On the click of the button first Activity should pass the values to another Activity. Second Activity must show the numbers in two different TextViews. On the click of the button of second Activity, it must return the addition of the numbers to the first Activity. Now first Activity must show the returning result in the label created in it.
