SPPU New Syllabus

A Book Of

ANDROID PROGRAMMING

For T.Y.B.C.A. (Science) : Semester - VI

[Course Code BCA - 361 : Credits - 04]

CBCS Pattern

As Per New Syllabus, Effective from June 2021

Kamil Ajmal Khan

M.Sc. Computer Science
Assistant Professor, Computer application Department
Abeda Inamdar Senior College Pune

Dr. Ms. Manisha Bharambe

M.Sc. (Comp. Sci.), M.Phil, Ph.D. (Comp. Sci.)
Associate Professor, Computer Science Department
MES Abasaheb Garware College, Pune

Price ₹ 240.00

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

N5954

Syllabus ...

1. Introduction to Android **[06 Hrs]**

- 1.1 Overview
- 1.2 History
- 1.3 Features of Android
- 1.4 Architecture of Android
- 1.5 SDK Overview
- 1.6 Creating your first Android Application

2. Activities, Fragments and Intents **[10 Hrs]**

- 2.1 Introduction to Activities
- 2.2 Activity Lifecycle
- 2.3 Introduction to Intents
- 2.4 Linking Activities using Intents
- 2.5 Calling built-in applications using Intents
- 2.6 Introduction to Fragments
- 2.7 Adding Fragments Dynamically
- 2.8 Lifecycle of Fragment
- 2.9 Interaction Between Fragment
- 2.10 Toast

3. Android User Interface **[06 Hrs]**

- 3.1 Understanding the components of a screen
- 3.2 Adapting to Display Orientation
- 3.3 Split Screen / Multi-Screen Activities

4. Designing Your User Interface with Views **[13 Hrs]**

- 4.1 Using Basic Views
- 4.2 Using Picker Views
- 4.3 Using List Views to Display Long Lists
- 4.4 Understanding Specialized Fragments
- 4.5 Displaying Pictures and Menus
- 4.6 VideoView

5. Databases – SQLite, Messaging and E-mail **[14 Hrs]**

- 5.1 Introduction to SQLite
- 5.2 SQLiteOpenHelper and SQLite Database
- 5.3 Creating and Using database
- 5.4 Working with Cursors
- 5.5 Building and Executing Quries
- 5.6 SMS Messaging
- 5.7 Sending E-mail

6. Location-Based Services and Google Map **[11 Hrs]**

- 6.1 Display Google Maps
- 6.2 Getting Location Data
- 6.3 Monitoring a Location

■■■

Contents ...

■■■

1... Introduction to Android

Learning Objectives ...

Students will be able to:

- To learn the basic concept of Android Operating System.
- To understand History, Features and architecture of Android Operating System.
- To study JDK, SDK, ADT, AVD, Android Emulator and Versions of Android Operating System.
- To study Android Studio with its screen elements.
- To develop Android Applications using Android Studio.

1.1 OVERVIEW

- Technology generates more technology, and new technologies multiply with ever-increasing speed. Some of these new technologies will survive beyond a few years, but most will not.
- There is little worse than investing time and energy in acquiring a new skill that is obsolete on arrival or whose utility is short-lived. In this era, most of the people want to use smart devices for communication, planning and organizing their schedule for their private and professional life.
- These technologies are causing dramatic changes in the organization of information system. Android has been changed in smart device market. Android is a new generation mobile OS which runs on Linux kernel.
- Android device application developed is based on Java Programming. These codes are used to control smart device via Google–enabled java libraries. It is an important platform to develop mobile device application using software stack provided in the Google Android SDK.
- Android combines Operating System features like structured shared memory, preemptive multi-tasking, Unix User Identifiers (UIDs) and file permission with Java language and its class library. The Security platform is much better than J2ME or Blackberry Platforms.

- Programs can typically neither read nor write each other's code. The software developers at mobile development India have expertise in developing application based on Android java libraries.
- Android is an open-source operating system named Android Inc. Google has made the code for all the low-level "stuff" as well as the needed middleware to power and use an electronic device, and gave Android freely to anyone who wants to write code and build the operating system from it. There is even a full application framework included, so third-party apps can be built and installed, then made available for the user to run as they like.

1.2 HISTORY

- Android is a mobile operating system developed by Google, based on a modified version of the Linux kernel and other open source software and designed primarily for touch screen mobile devices such as smart phones and tablets.
- In addition, Google has further developed Android TV for television, Android Auto for cars and Wear OS for wrist watches, each with a specialized user interface. Variants of Android are also used on game consoles, digital cameras, PCs and other electronics.
- Initially developed by Andy Rubin founded Android Incorporation in Palo Alto, California, United States in October, 2003, which Google bought in 2005, and Android was unveiled in 2007, with the first commercial Android device launched in September 2008.
- The version history of the Android mobile began with the public release of the Android beta on November 5, 2007. The first commercial version, Android 1.0, was released on September 23, 2008. Android is continually developed by Google and the Open Handset Alliance (OHA) and it has seen a number of updates to its base operating system since the initial release. OHA is a business alliance that consists of 47 companies like Google, HTC, LG Electronics, Samsung Electronics etc. for developing standard platforms for mobile technology.
- Versions 1.0 and 1.1 were not released under specific code names, although Android 1.1 was unofficially known as Petit Four. Android code names are confectionery-themed and have been in alphabetical order since 2009's Android 1.5 Cupcake, with the most recent major version being Android 8.1 Oreo, released in December 2017.
- The operating system has since gone through multiple major releases, with the current version being 9.0 "Pie", released in August 2018. The core Android source code is known as Android Open Source Project (AOSP), and is primarily licensed under the Apache License.

1.3 FEATURES OF ANDROID

- The various features of Android are:
  1. **Application Framework:** It enables reuse and replacements of components
  2. **Dalvik Virtual Machine:** It is optimized for mobile device
  3. **Integrated Browser:** It is based on open source web kit engine
  4. **Handset layouts:** Optimized Graphics: It is powered by a custom 2D graphics Library 3D Graphics based on the OpenGL ES 1.0 Specification.
  5. **Storage:** SQLite, a lightweight relational database.
  6. **Media Support:** Android supports the following audio/video/still media formats: H.263, H.264 (in 3GP or MP4 container), MPEG - 4 SP, AMR, AMR - WB (in 3GP container), AAC, HE - AAC (in MP4 or 3GP container), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF and BMP.
  7. **Connectivity:** GSM/EDGE, IDEN, CDMA, EVDO, UMTS, Bluetooth, WiFi.
  8. **Messaging:** SMS and MMS are available forms of messaging, also support Android Cloud to Device Messaging Framework (C2DM).
  9. **Multiple Language Support:** Multiple Language Available in Android.
  10. **Additional Hardware Support:** Android can use video/still cameras, touch screens, GPS, accelerometers, gyroscopes, magnetometers.
  11. **Multi - touch:** Android has native support for multi - touch which was initially made available in handsets such as the HTC Hero.
  12. **Multitasking:** Multitasking of applications is available.
  13. **Voice based features**: Google search through voice has been available since initial release, voice actions for calling, texting, navigation etc.
  14. **Android Market (Google Play):** An online software store developed by Google for Android devices allows users to browse and download apps published by third party developers, hosted on Android Market.
  15. **Streaming Media Support:** RTP/RTSP streaming (3GPP PSS, ISMA), HTML progressive download, Adobe Flash Streaming (RTP) and HTTP Dynamic Streaming are supported by the Flash 10.1 plug-in.

1.4 ARCHITECTURE OF ANDROID

1.4.1 Overview of Stack

- Android operating system is a stack of software components. Main components of Android Operating system Architecture or Software Stack are Linux kernel, native libraries, Android Runtime, Application Framework and Applications.
- On top of the Linux kernel, there are the middleware, libraries and APIs written in C, and application software running on an application framework which includes Java-compatible libraries. Development of the Linux kernel continues independently of Android's other source code projects.

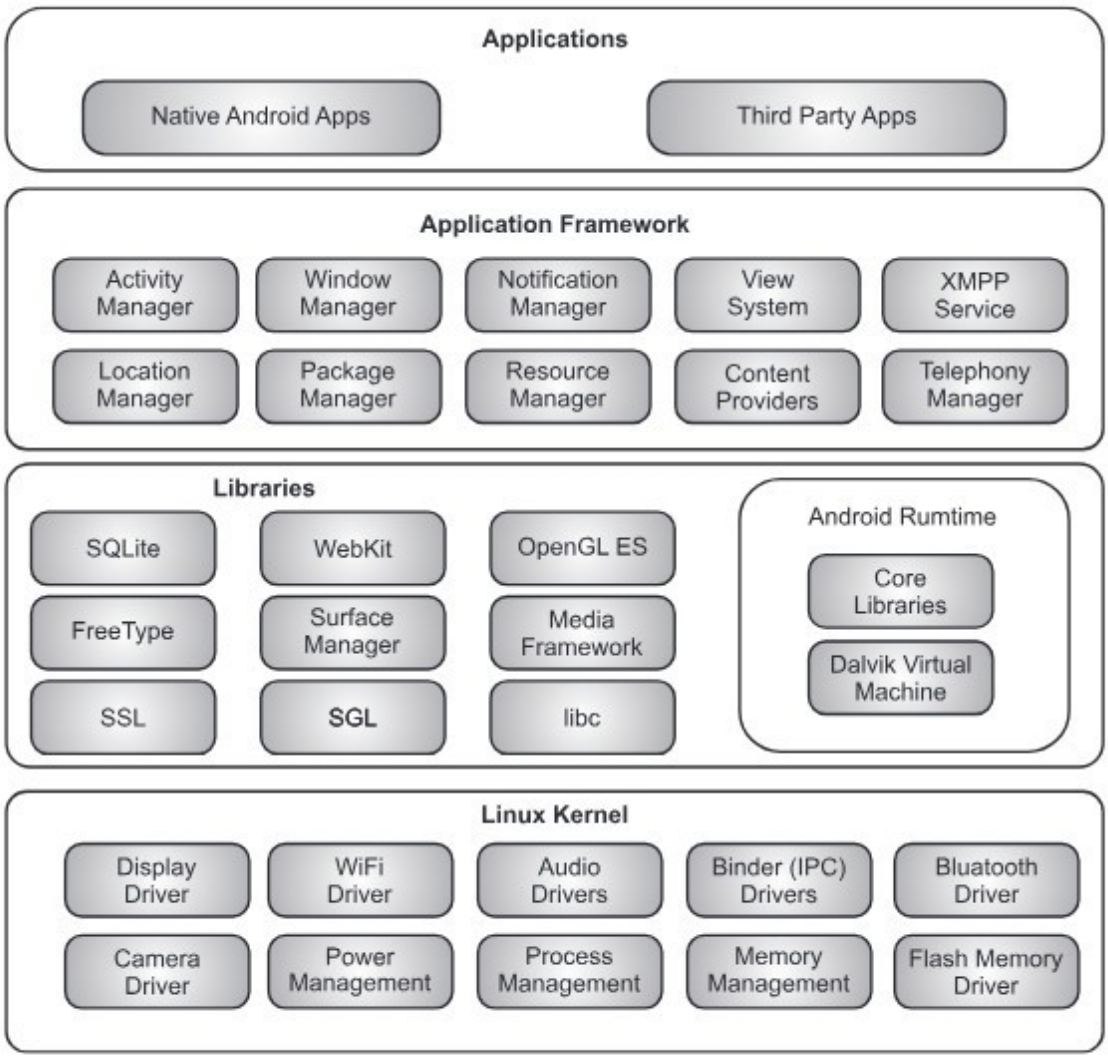


Fig. 1.1: Architecture of Android

- The Android OS is roughly divided into five sections in four main layers:
  - **Linux kernel:** This is the kernel on which Android is based. This layer contains all the low level device drivers for the various hardware components of an Android device.
  - **Libraries**: These contain all the code that provides the main features of an Android OS. For example, the SQLite library provides database support so that an application can use it for data storage. The WebKit library provides functionalities for web browsing.
  - **Android runtime:** At the same layer as the libraries, the Android runtime provides a set of core libraries that enable developers to write Android apps using the Java programming language.
  - **Application framework:** Exposes the various capabilities of the Android OS to application developers so that they can make use of them in their applications.

- **Applications:** At this top layer, you will find applications that ship with the Android device (such as Phone, Contacts, Browser, etc.), as well as applications that you download and install from the Android Market. Any applications that you write are located at this layer.

1.4.2 Linux Kernel

- Android devices use the Linux kernel, but every phone uses their own version of it. Linux kernel maintainers keep everything tidy and available, contributors (like Google) add or alter things to better meet their needs, and the people making the hardware contribute as well, because they need to develop hardware drivers for the parts they're using for the kernel version they're using.
- This is why it takes a while for independent Android developers and hackers to port new versions to older devices and get everything working. Drivers written to work with one version of the kernel for a phone might not work with a different version of software on the same phone.
- And that's important, because one of the kernel's main functions is to control the hardware. It's whole source code, with more options while building it than you can imagine, but in the end it's just the intermediary between the hardware and the software.
- When software needs the hardware to do anything, it sends a request to the kernel. And when we say anything, we mean anything. From the brightness of the screen, to the volume level, to initiating a call through the radio, even what's drawn on the display is ultimately controlled by the kernel.
- Linux kernel is responsible for device drivers, power management, memory management, device management and resource access.

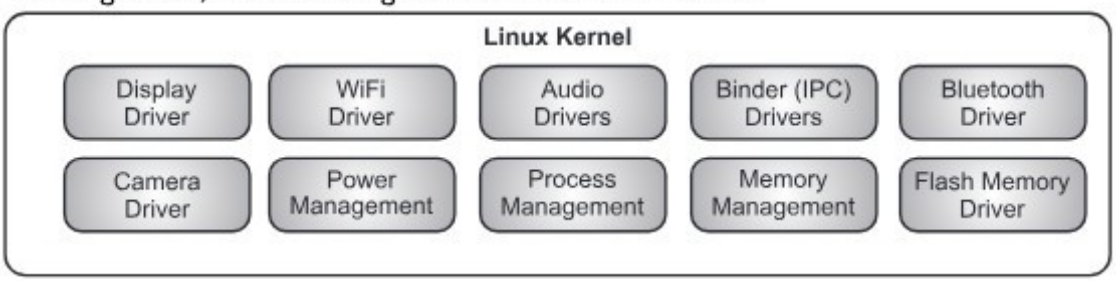


Fig. 1.2: Linux kernel

1.4.3 Native Libraries

- This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access.

- **A summary of some key core Android libraries available to the Android developer is as follows:**
  - **android.app:** Provides access to the application model and is the cornerstone of all Android applications.
  - **android.content:** Facilitates content access, publishing and messaging between applications and application components.
  - **android.database:** Used to access data published by content providers and includes SQLite database management classes.
  - **android.opengl:** A Java interface to the OpenGL ES 3D graphics rendering API.
  - **android.os:** Provides applications with access to standard operating system services including messages, system services and inter-process communication.
  - **android.text:** Used to render and manipulate text on a device display.
  - **android.view:** The fundamental building blocks of application user interfaces.
  - **android.widget:** A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
  - **android.webkit:** A set of classes intended to allow web-browsing capabilities to be built into applications.

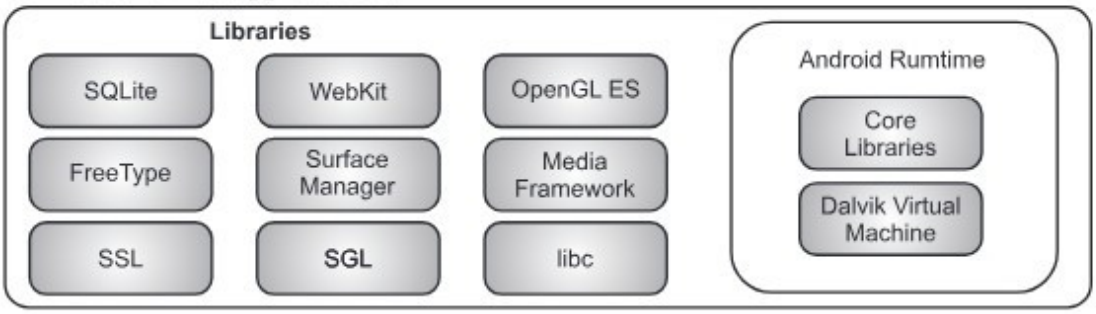


Fig. 1.3: Native Libraries of android

1.4.4 Android Runtime

(a) Dalvik Virtual Machine: The Android runtime also includes the Dalvik virtual machine, which enables every Android application to run in its own process with its own instance of the Dalvik virtual machine (Android applications are compiled into the Dalvik executables).

Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.

(b) Core Libraries: These are different from Java SE and Java ME libraries. But these libraries provide most of the functionalities defined in the Java SE libraries.

- Data Structure
- File Access
- Network Access
- Utilities
- Graphics

1.4.5 Application Framework

- The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.
- The Android framework includes the following key services:
  - **Activity Manager:** Controls all aspects of the application lifecycle and activity stack.
  - **Content Providers:** Allows applications to publish and share data with other applications.
  - **Resource Manager:** Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
  - **Notifications Manager:** Allows applications to display alerts and notifications to the user.
  - **View System:** An extensible set of views used to create application user interfaces.

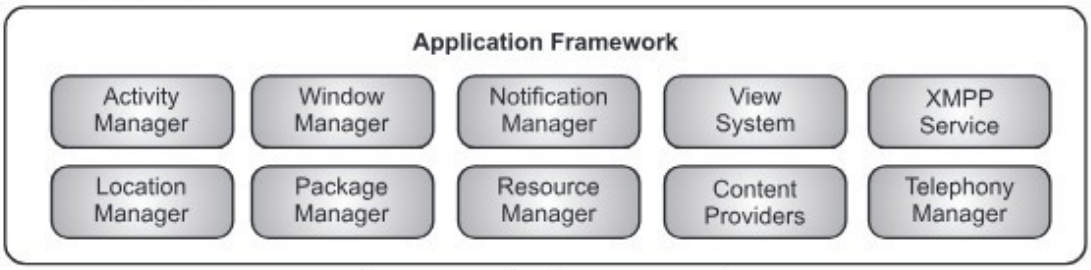


Fig. 1.4: Application Framework

1.4.6 Applications

- Applications are located at the top of the Android software stack are the applications.
- These comprise both the native applications provided with the particular Android implementation (for example, web browser and email applications) and the third party applications installed by the user after purchasing the device.

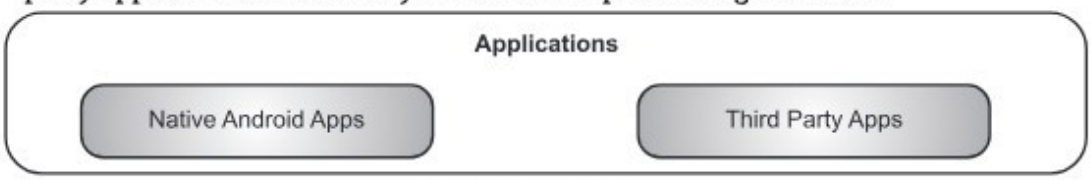


Fig. 1.5: Applications

1.5 SDK OVERVIEW

1.5.1 Android Platforms

- An SDK platform actually corresponds to an Android platform. Each platform includes a system image, libraries, and sample codes. An Android Virtual Device (AVD) is also created based on an SDK platform.

- Platforms are located in the platforms directory under the root directory of Android SDK (sdk/platforms/). You can download the Android SDK from http://developer.android.com/sdk/index.html.
- Once the SDK is downloaded, unzip its content (the android-sdk-windows folder) into the C:\Android\ folder, or whatever name you have given to the folder you just created.
- At least one Android platform must be installed into your environment. It is reasonable to download and install the latest platform version. After developing the application, you also should install older platforms and test under those platforms if you plan to publish the application for older versions.
- Android SDK platform contains packages/libraries to develop & build Android Application for specific versions. To compile your Application against specific version, to load specific widgets, views and tools for compilation, is done by SDK Platform. Hence, whenever you want to download packages for any new Android version like 8.0 or 7.0, always select and download SDK Platform for it.
- SDK Tools and Build-tools contain packages to build your Android Application and several updates/patches from Google for better Application Development. For specific API level (version) of Android, you have to download additional packages from the section **SDK Platforms**. You can easily understand which API Level/Android Version you want to install.
- For example, a mobile carrier or device manufacturer might offer additional API libraries that are supported by their own Android-powered devices. To develop using their libraries, you can install their Android SDK package by adding their SDK tools URL to the SDK Manager in the **SDK Update Sites**.

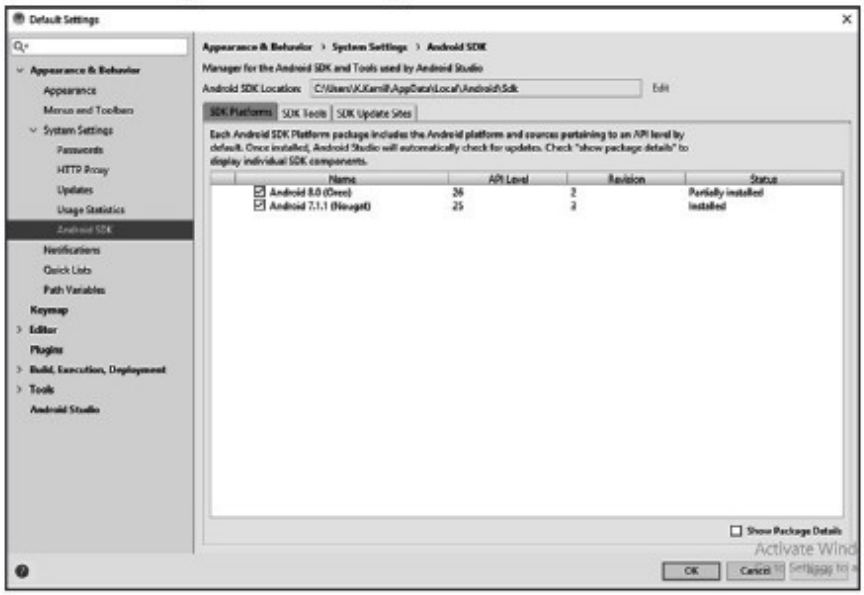


Fig. 1.6: Android Platform

1.5.2 Tools - (JDK, SDK, Android Studio, ADT, AVD, Android Emulator)

- For Android development, you can use a Mac, a Windows PC, or a Linux machine. All the tools needed are free and can be downloaded from the Web. Most of the examples provided in this book should work fine with the Android emulator, with the exception of a few examples that require access to the hardware.

1.5.2.1 JDK(Java Development Kit)

- Android Studio uses the Java tool chain to build, so you need to make sure that you have the Java Development Kit (JDK) installed on your computer before you start using Android Studio.
- It's quite possible that you already have the JDK installed on your computer, particularly if you're a seasoned Android or Java developer. If you already have the JDK installed on your computer, and you're running JDK version 1.6 or higher, then you can skip this section.
- However, you may want to download, install, and configure the latest JDK anyway. You can download the JDK from the following Oracle site:

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

Fig. 1.7: Installation Wizard for the JDK on Windows

Fig. 1.8: Select the JDK installation directory

- Make a note of where you are installing your JDK. Follow the prompts until the installation is complete. If prompted to install the Java Runtime Edition (JRE), choose the same directory where you installed the JDK.

Configuring Environmental Variables on Windows

- This section shows you how to configure Windows so that the JDK is found by Android Studio. On a computer running Windows, hold down the Windows key and press the Pause key to open the System window. Click the Advanced System Settings option, shown in Fig. 1.9.

Fig. 1.9: Windows System window

- Click the Environmental Variables button. In the System Variables list along the bottom navigate to the JAVA_HOME item. If the JAVA_HOME item does not exist, click **New** to create it. Otherwise, click **Edit**.
- Clicking either New or Edit displays a dialog box. Be sure to type JAVA_HOME in the Variable Name field. In the Variable Value field, type the location where you installed the JDK earlier (less any trailing slashes). Now click OK.

1.5.2.2 SDK (Software Development Kit)

- The Android SDK (Software Development Kit) is the most important software of android which is installed.
- The Android SDK provides to test android applications, the API libraries, an emulator, documentation, sample code, developer tools and tutorials which help you to build, test and debug apps of Android.

- Android SDK is made up of two main parts: the tools and the packages. When you first install the SDK, all you obtain are the base tools. Is helps you to develop applications. The packages are the records specific to a particular version of Android.

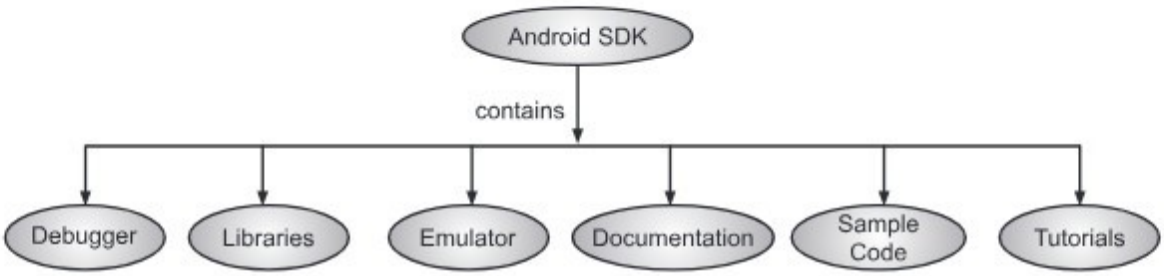


Fig. 1.10: SDK

Feature of Android SDK:

- No licensing, distribution or development fees.
- WiFi hardware access.
- Shared data store.
- Full multimedia hardware control etc.

1.5.2.3 Android Studio

- Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA.
- On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:
  - A flexible Gradle-based build system.
  - A fast and feature-rich emulator.
  - A unified environment where you can develop for all Android devices.
  - Instant Run to push changes to your running app without building a new APK.
  - Code templates and GitHub integration to help you build common app features and import sample code.
  - Extensive testing tools and frameworks.
  - Lint tools to catch performance, usability, version compatibility, and other problems.
  - C++ and NDK support.
  - Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine.

Downloading Android Studio is straightforward. Point your browser to this site:

www.developer.android.com/sdk/installing/studio.html

Now click the large green Download Android Studio for your OS button,

Fig. 1.11: Download Android Studio

- After the Installation Wizard begins, move through its screens by clicking the Next buttons until you reach the Choose Components screen. There, select all the component check boxes, shown in Fig. 1.12.

Fig. 1.12: Choose components

- Then click Next. Agree to the terms and conditions once again. When you reach the Configuration Settings: Install Locations screen, shown in Fig. 1.13, select the locations for Android Studio and the Android SDK. To be consistent, we chose to install Android Studio in C:\Java\astudio\ and the Android SDK in C:\Java\asdk\.

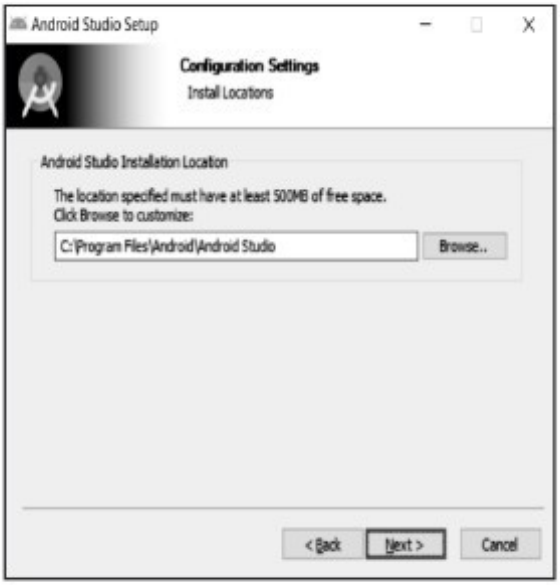


Fig. 1.13: Select locations for Android Studio and the SDK

- Click through several Next buttons as you install both Android Studio and the Android SDK. You should eventually arrive at the Completing the Android Studio Setup screen, shown in Fig. 1.14.
- The Start Android Studio check box enables Android Studio to launch after you click Finish. Make sure the check box is selected, and then go ahead and click Finish, and Android Studio will launch. Please note that from here on out, you will need to navigate to either the desktop icon or the Start menu to launch Android Studio.

Fig. 1.14: Completing the Android Studio setup

- Now that Android Studio 2 is installed, you need to adjust the settings and options using the following steps:
  - Click Continue at the Welcome screen and choose Standard from the Install Type selection screen shown in Fig. 1.15. Click Next to continue.

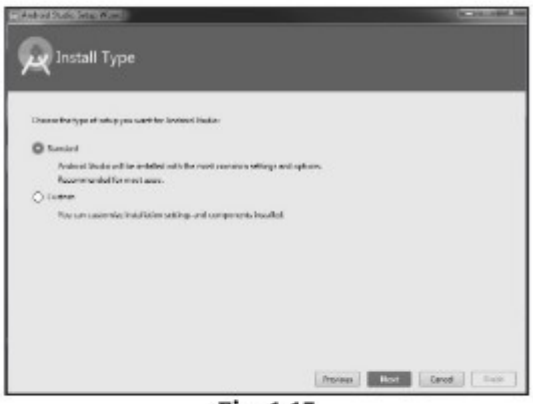


Fig. 1.15

- Click Finish on the Verify Settings screen, and Android Studio 2 finalizes the setup process. You know the process is complete when you are greeted with the Welcome to Android Studio screen (see Fig. 1.16)

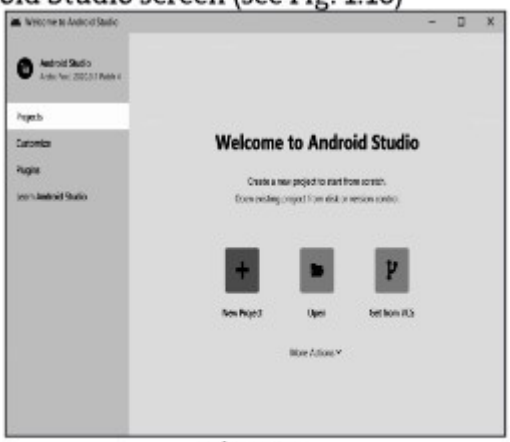


Fig. 1.16

- Now that Android Studio is set up, it's time to install the latest and greatest Android SDK.

Fig. 1.17

1.5.2.4 ADT (Android Development Tools)

- The important part of facilitating application development itself is to perform correct and functional setup of the development environment. For the purpose of development for Android it is necessary to connect Eclipse with the Android SDK.
- Linking is represented by the plug-in named Android Development Tool (ADT), which extends the capabilities of Eclipse and allows the rapid development of Android projects.
- With ADT, programmer receives a powerful integrated development environment with editor visual applications, own XML editors, and debug panels and APK packages for distributions of the applications.
- The ADT is an extension to the Eclipse IDE that supports the creation and debugging of Android applications. Using the ADT, you will be able to the following in Eclipse:
  - Create new Android Applications projects.
  - Access the tools for accessing your Android emulators and devices.
  - Compile and debug Android applications.
  - Export Android applications into Android Packages.
  - Create digital certificates for code-signing your APK.

1.5.2.5 AVD (Android Virtual Device)

- The Android Virtual Device Manager allows you to create Android Virtual Devices (AVDs), which you can then run to emulate a device on your computer. There's an important but subtle distinction between simulation and emulation.
- Simulation means that the virtual device is merely a façade that simulates how an actual physical device might behave, but does not run the targeted operating system.
- The iOS development environment uses simulation, and this is probably a good choice for iOS given the limited number of devices available for that platform.
- With emulation, however, your computer sets aside a block of memory to reproduce the environment found on the device that the emulator is emulating. Android Studio uses emulation, which means the Android Virtual Device Manager launches a sandboxed version of the Linux kernel and the entire Android stack in order to emulate the environment found on the physical Android device.
- Although emulation provides a much more faithful environment on which to test your apps than simulation does, booting up an AVD can drag into the minutes, depending on the speed of your computer.
- The good news is that after your emulator is active in memory, it remains responsive. Nevertheless, if you have an Android phone or tablet, we recommend using the physical device to test your apps, rather than using an AVD.

Fig. 1.18: AVD icon

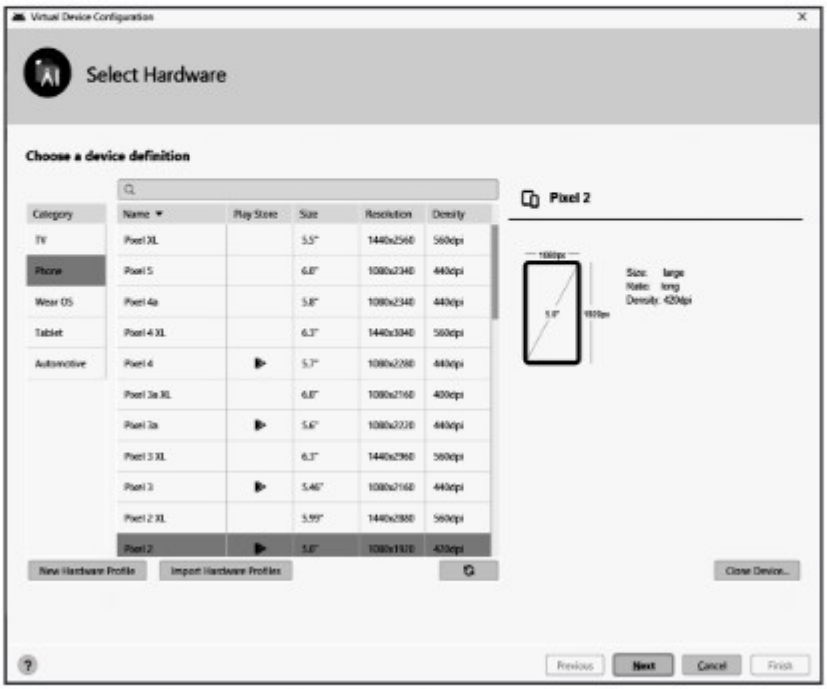


Fig. 1.19: Select the Galaxy Nexus hardware

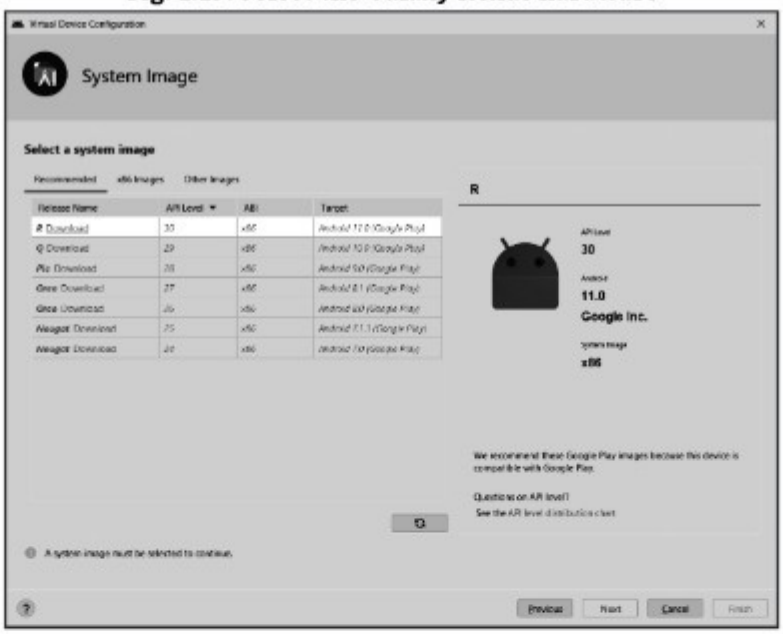


Fig. 1.20: Select the x86_64 system image

1.5.2.6 Android Emulator

- The Android emulator runs one full Android system stack, down to the kernel level, including a set of preinstalled applications (such as the dialer) that you can access from your applications.

- We can select the version of the Android system we wish to run the emulator by configuring AVDs and we can also modify the mobile device skin and key mappings.
- While launching the emulator at the runtime, we can use a variety of commands and options to control its behaviour.
- The Android emulator offers dynamic binary translation of device machine code to the OS and processor architecture of your development machine.
- The android emulator supports many hardware features likely to be found on mobile devices, including:
  - AN ARMv5 CPU and the corresponding Memory Managements Unit (MMU).
  - A 16-bit LCD display.
  - One or more keyboards.

Fig. 1.21: Run Button to Start Emulator

Fig. 1.22: Choosing a device and launching the emulator

- It will display following Emulator window:

Fig. 1.23: Emulator screenshot

1.5.3 Versions of Android

Sr. No.	Version	Description
1.	Android 1.0	Android 1.0, the first commercial version of the software, was released on September 23, 2008. The first commercially available Android device was the HTC Dream.
2.	Android 1.1	On February 9, 2009, the Android 1.1 update was released, initially for the HTC Dream only. Android 1.1 was known as "Petit Four" internally, though this name was not used officially.
3.	Android 1.5	On April 27, 2009, the Android 1.5 update was released based on Linux kernel 2.6.27. This was the first release to officially use a codename based on a dessert item ("Cupcake"), a theme which would be used for all releases henceforth.
4.	Android 1.6 Donut	On September 15, 2009, Android 1.6 – dubbed Donut – was released, based on Linux kernel 2.6.29.

contd. ...

5.	Android 2.0/2.1 Eclair	On October 26, 2009, the Android 2.0 SDK was released, based on Linux kernel 2.6.29 and codenamed Eclair.
6.	Android 2.2 Froyo	On May 20, 2010, the SDK for Android 2.2 (Froyo, short for frozen yogurt) was released, based on Linux kernel 2.6.32.
7.	Android 2.3/2.4 Gingerbread	On December 6, 2010, the Android 2.3 (Gingerbread) SDK was released, based on Linux kernel 2.6.35.
8.	Android 3.0/3.1/3.2 Honeycomb	On February 22, 2011, the Android 3.0 (Honeycomb) SDK – the first tablet-only Android update – was released, based on Linux kernel 2.6.36. The first device featuring this version, the Motorola Xoom tablet, was released on February 24, 2011.
9.	Android 4.0/4.0.1/4.0.2 Ice Cream Sandwich	The SDK for Android 4.0.1 (Ice Cream Sandwich), based on Linux kernel 3.0.1, was publicly released on October 19, 2011. Ice Cream Sandwich was the last version to officially support Adobe Systems' Flash player.
10.	Android 4.1/4.2/4.3 Jelly Bean	Google announced Android 4.1 (Jelly Bean) at the Google I/O conference on June 27, 2012. Based on Linux kernel 3.0.31, Jelly Bean was an incremental update with the primary aim of improving the functionality and performance of the user interface. Jelly Bean 4.2 was based on Linux kernel 3.4.0, and debuted on Google's Nexus 4 and Nexus 10, which were released on November 13, 2012. Google released Jelly Bean 4.3 under the slogan "An even sweeter Jelly Bean" on July 24, 2013.
11.	Android 4.4 KitKat	Google announced Android 4.4 KitKat on September 3, 2013 and released October 31, 2013.
12.	Android 5.0/5.1Lollipop	Android 5.0 "Lollipop" was unveiled under the codename "Android L" on June 25, 2014, during **Google I/O**. It became available as official **over-the-air** (OTA) updates on November 12, 2014, for select devices that run distributions of Android serviced by Google, including **Nexus** and **Google Play edition** devices. Its source code was made available on November 3, 2014.

contd. ...

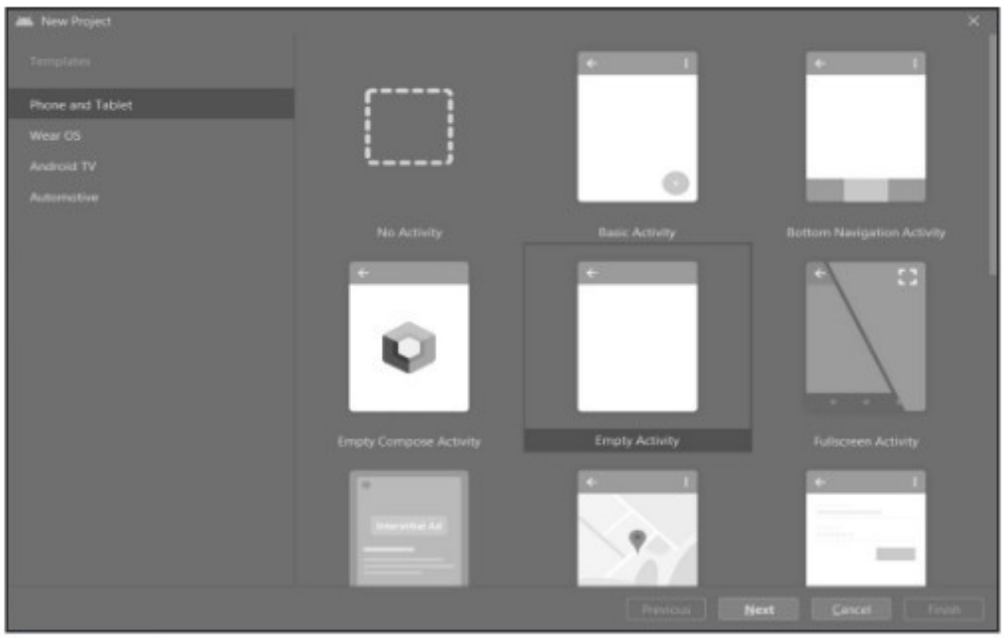
13.	Android 6.0 Marshmallow	Android 6.0 "Marshmallow" was unveiled under the codename "Android M" during Google I/O on May 28, 2015, for the Nexus 5 and Nexus 6 phones, Nexus 9 tablet, and Nexus Player set-top box, under the build number MPZ44Q. The third developer preview (MPA44G) was released on August 17, 2015 for the Nexus 5, Nexus 6, Nexus 9 and Nexus Player devices and was updated to MPA44I that brought fixes related to Android for Work profiles.
14.	Android 7.0/7.1 Nougat	Android "Nougat" (codenamed N in-development) is the major 7.0 release of the Android operating system. It was first released as a developer preview on March 9, 2016, with factory images for current Nexus devices, as well as with the new "Android Beta Program" which allows supported devices to be upgraded directly to the Android Nougat beta via over-the-air update. Final release was on August 22, 2016.
15.	Android 8.0/8.1 Oreo	Android Oreo is the 8^{th} major release of the Android operating system. It was first released as a developer preview on March 21, 2017, with factory images for current Nexus and Pixel devices. The final developer preview was released on July 24, 2017, with the stable version released in August 2017.
16.	Android 9.0 P	Android "P" is the upcoming ninth major version of the Android operating system. It was first announced by Google on March 7, 2018, and the first developer preview was released on the same day. Second preview, considered beta quality, was released on May 8, 2018. The final beta of Android P (fifth preview, also considered as a "Release Candidate") was released on July 25, 2018.
17.	Android 10.0 Quince Tart	Android 10 is the tenth major version of the Android operating system. The stable version of Android 10 was released on September 3, 2019.
18.	Android 11.0 Red Velvet Cake	Android 11 is the eleventh major version of the Android operating system. It was first announced by Google on February 19, 2020, and the first developer preview released on the same day. Android 11 Beta was postponed from being launched on June 3, 2020 to June 10, 2020.
19.	Android 12.0 Snow Cone	Android 12 is the twelfth major version of the Android operating system. It was first announced by Google on February 18, 2021, and the first developer preview released on the same day.

1.6 CREATING YOUR FIRST ANDROID APPLICATION

- To Create and Android Project Using Android Studio Follow the Below given Steps:

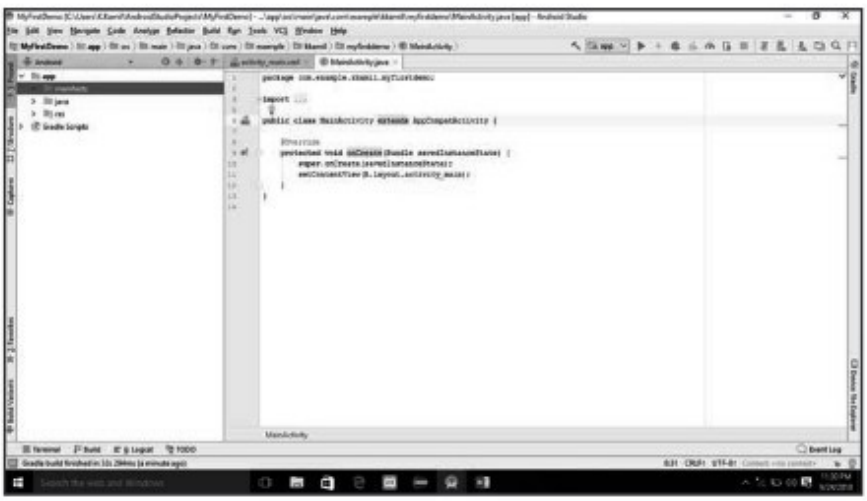
Step 1: Start Android Studio and Select Target Android Device.

Step 2: Click on Next



Step 3: Write Main Activity name Click on Finish Button

Step 4: Edit Activity_main.xml file



Step 5: Click on Run Button and that shows Emulator

Summary

- Android is an open source and Linux-based **Operating System** for mobile devices such as smartphones and tablet computers.
- Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It's the largest installed base of any mobile platform and growing fast. Every day more than 1 million new Android devices are activated worldwide.
- Android operating system is a stack of software components which is roughly divided into five sections namely Application Framework and Applications.
- Linux kernel is the heart of android architecture that exists at the root of android architecture. Linux kernel is responsible for device drivers, power management, memory management, device management and resource access.
- The Android SDK includes a virtual mobile device that runs on your computer. The emulator lets you prototype, develop and test Android applications without using a physical device.
- The Android emulator mimics all of the hardware and software features of a typical mobile device, except that it cannot place actual phone calls.

- An AVD is a virtual machine running Android on an emulated set of device specifications, including memory, screen size, CPU, etc.
- Android SDK Tools is a component for the Android SDK. It includes the complete set of development and debugging tools for Android. It is included with Android Studio.
- A project in Android Studio contains everything that defines your workspace for an app, from source code and assets, to test code and build configurations.
- ADT (Android Developer Tools) is a plug-in for Eclipse that provides a suite of tools that are integrated with the Eclipse IDE. It offers you access to many features that help you develop Android applications quickly.

Practice Questions

Q.I Multiple Choice Questions:

1. _____ provides a variety of navigation and control keys, which you can "press" using your mouse or keyboard to generate events for your application and also provides a screen in which your application is displayed, together with any other active Android applications.
   (a) Android Virtual Device (b) Emulator
   (c) Android Developer Tools (d) None of the above
2. Android has recently become the majority operating system on mobile smartphones today, because of _____.
   (a) Open source.
   (b) Versatile mobile OS (work on a vast majority of different types of phone hardware).
   (c) Inter application development.
   (d) All of the above.
3. The _____ Manager is an interface you can launch from Android Studio that helps you create and manage AVDs.
   (a) SDK (b) JDK
   (c) AVD (d) None of the above
4. _____ is a mobile operating system based on the Linux kernel and now developed by Google.
   (a) Android (b) Unix
   (c) iOS (d) All of the above

5. Android is primarily designed for _____ .
   (a) Smartphones
   (b) Table computers
   (c) Specialized user interface for Android TV
   (d) All of the above
6. The Android _____ provides you the API libraries and developer tools necessary to build, test, and debug apps for Android.
   (a) JDK (b) SDK
   (c) ADT (d) All of the above
7. ADT is a plug-in for _____.
   (a) Android Studio (b) Eclipse
   (c) AIDE (d) All of the above
8. Android application framework includes _____.
   (a) Activity Manager (b) Notifications Manager
   (c) Content Providers (d) All of the above

Answers

1. (b)	2. (d)	3. (c)	4. (a)	5. (d)	6. (b)	7. (b)	8. (d)

Q.II Answer the following questions:

1. Enlist SDK platforms in detail.
2. Describe ADT in detail.
3. Expalin the Android stack diagrammatically.
4. How to create an application in Android? Explain with example.
5. With the help of diagram describe Android architecture.

Q.III Answer the following questions in short:

1. Enlisr reasons for why use Android OS?
2. What are the features of Android?
3. Write short note on: History of Android.
4. What is SDK?
5. What is meant by AVD?
6. Write short note on: Android Emulator.
7. What is Android?
8. What is JDK?

Q.IV Define Terms:

1. Android runtime
2. Native libraries
3. Versions
4. Applications.

■■■

2... Activities, Fragments and Intents

Learning Objectives ...

Students will be able to:

- To learn Activity concept in Android with its Life Cycle.
- To study Intents in Android.
- To understand Fragment in Android with its Life Cycle.
- To learn the concept of toast in Android.

2.1 INTRODUCTION TO ACTIVITIES

- An activity is a window that contains the user interface of your applications. An application can have zero or more activities. Typically, applications have one or more activities, and the main aim of an activity is to interact with the user. From the moment an activity appears on the screen to the moment it is hidden, it goes through a number of stages, known as an activity's life cycle. Understanding the life cycle of an activity is vital to ensuring that your application works correctly.
- An activity represents a single screen with a user interface, in-short Activity performs actions on the screen. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails.
- If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.
- To create an activity, you create a Java class that extends the Activity base class:

```
package net.learnfromkamil.Activities;
import android.app.Activity;
import android.os.Bundle;
```

```
public class MainActivity extends Activity
{
   /** Called when the activity is first created. */
@Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
   }
}
```

- Your activity class would then load its UI component using the XML file defined in your res/layout folder. In this example, you would load the UI from the main.xml file:

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

- Every activity you have in your application must be declared in your AndroidManifest.xml file, like this:

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

<application
android:icon="@drawable/icon"android:label="@string/app_name">

<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>
<uses-sdkandroid:minSdkVersion="9" />

</manifest>
```

- The Activity base class defines a series of events that governs the life cycle of an activity. The Activity class defines the following events:
  - **onCreate():** Called when the activity is first created.
  - **onStart():** Called when the activity becomes visible to the user.
  - **onResume():** Called when the activity starts interacting with the user.
  - **onPause():** Called when the current activity is being paused and the previous activity is being resumed.

- **onStop():** Called when the activity is no longer visible to the user.
- **onDestroy():** Called before the activity is destroyed by the system (either manually or by the system to conserve memory).
- **onRestart():** Called when the activity has been stopped and is restarting again.

- Android is designed around the unique requirements of mobile applications. In particular, Android recognizes that resources (memory and battery, for example) are limited on most mobile devices, and provides mechanisms to conserve those resources. The mechanisms are evident in the Android Activity Lifecycle, which defines the states or events that an activity goes through from the time it is created until it finishes running.

2.2 ACTIVITY LIFECYCLE

- The lifecycle is shown diagrammatically in Fig. 2.1.

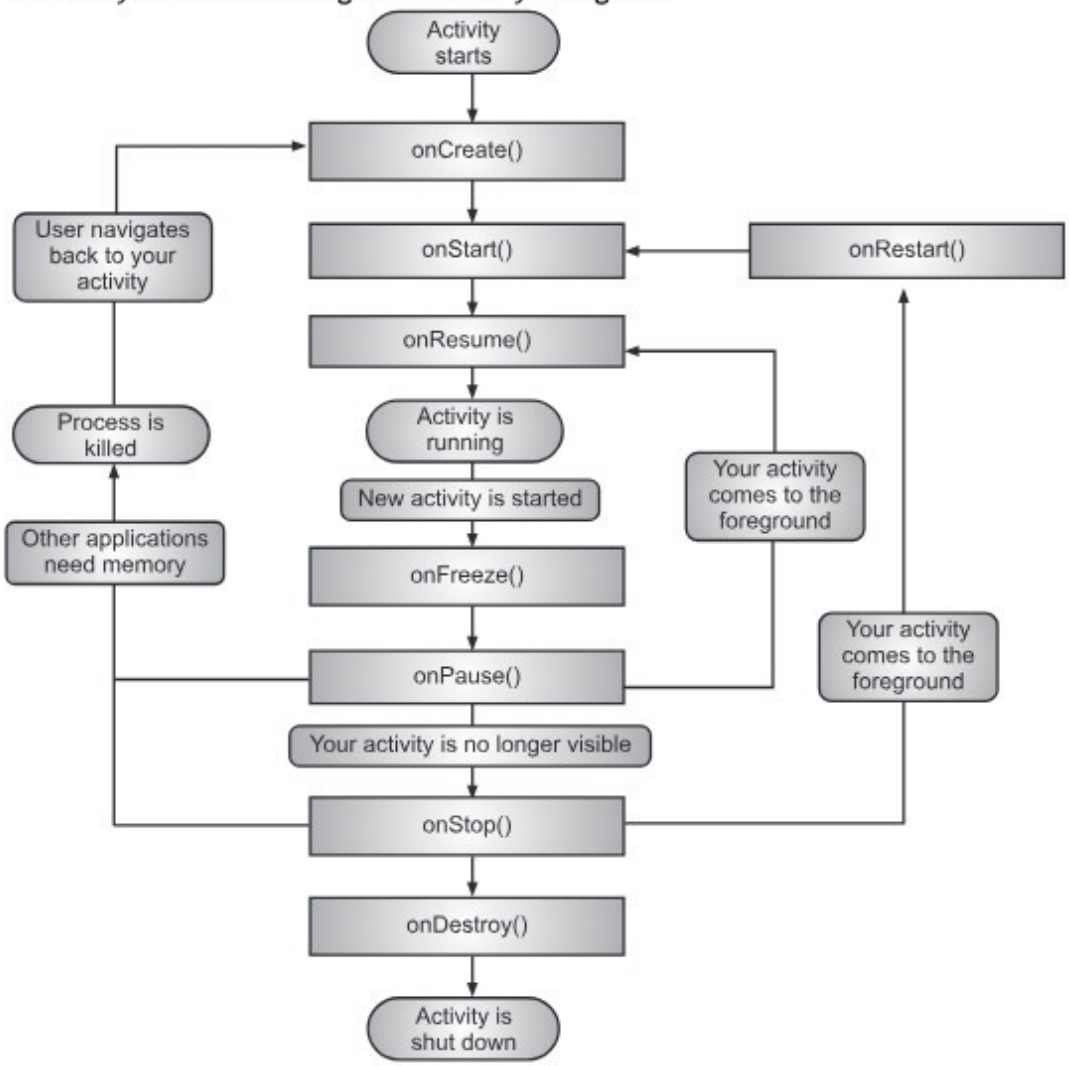


Fig. 2.1: Activity Lifecycle

- Your activity monitors and reacts to these events by instantiating methods that override the Activity class methods for each event:

  onCreate:

  - Called when your activity is first created. This is the place you normally create your views, open any persistent datafiles your activity needs to use, and in general initialize your activity.
  - When calling onCreate, the Android framework is passed a Bundle object that contains any activity state saved from when the activity ran before.

  onStart:

  - Called just before your activity becomes visible on the screen. Once onStart completes, if your activity can become the foreground activity on the screen, control will transfer to onResume.
  - If the activity cannot become the foreground activity for some reason, control transfers to the onStop method.

  onResume:

  - Called right after onStart if your activity is the foreground activity on the screen. At this point your activity is running and interacting with the user.
  - You are receiving keyboard and touch inputs, and the screen is displaying your user interface. onResume is also called if your activity loses the foreground to another activity, and that activity eventually exits, popping your activity back to the foreground.
  - This is where your activity would start (or resume) doing things that are needed to update the user interface (for example, receiving location updates or running an animation).

  onPause:

  - Called when Android is just about to resume a different activity, giving that activity the foreground. At this point your activity will no longer have access to the screen, so you should stop doing things that consume battery and CPU cycles unnecessarily.
  - If you are running an animation, no one is going to be able to see it, so you might as well suspend it until you get the screen back.
  - Your activity needs to take advantage of this method to store any state that you will need in case your activity gains the foreground again and it is not guaranteed that your activity will resume.

  onStop:

  - Called when your activity is no longer visible, either because another activity has taken the foreground or because your activity is being destroyed.

onDestroy:

- The last chance for your activity to do any processing before it is destroyed. Normally you'd get to this point because the activity is done and the framework called its finish method. But as mentioned earlier, the method might be called because Android has decided it needs the resources your activity is consuming.
- The best way to understand the various stages experienced by an activity is to in android studio select File -> New -> Create a new project, implement the various events, and then subject the activity to various user interactions.

Step 1 : Using Android Studio, create a new Android project and name it as shown in Fig. 2.2.

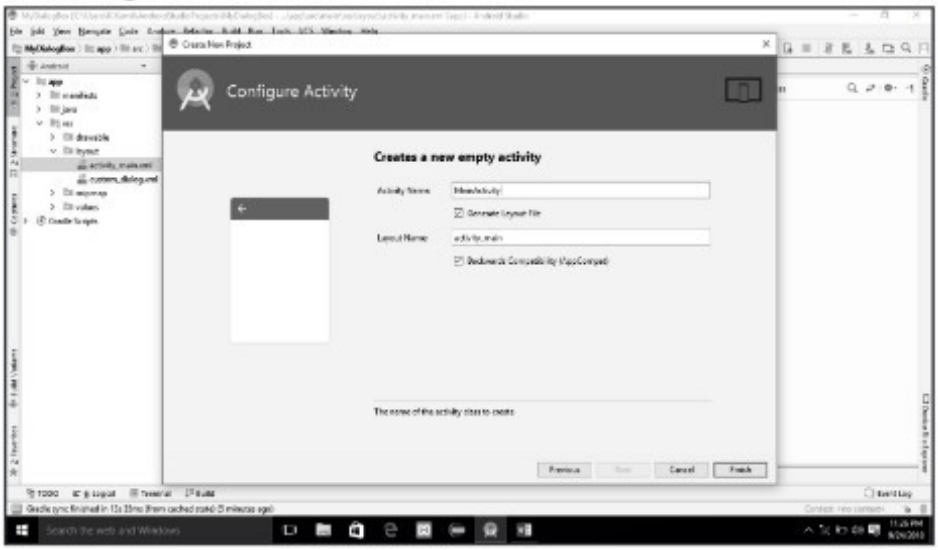


Fig. 2.2

Step 2 : In the MainActivity.java file, add the following statements in bold

```
package com.example.kkamil.learnfromkamil;
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
public class MainActivity extends AppCompatActivity
{
    String tag="Events";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(tag,"In on Create Event");
    }
```

```
    public void onStart()
    {
        super.onStart();
        Log.d(tag, "In the onStart() event");
    }
    public void onRestart()
    {
        super.onRestart();
        Log.d(tag, "In the onRestart() event");
    }
    public void onResume()
    {
        super.onResume();
        Log.d(tag, "In the onResume() event");
    }
    public void onPause()
    {
        super.onPause();
        Log.d(tag, "In the onPause() event");
    }
    public void onStop()
    {
        super.onStop();
        Log.d(tag, "In the onStop() event");
    }
    public void onDestroy()
    {
        super.onDestroy();
        Log.d(tag, "In the onDestroy() event");
    }
}
```

Step 3 : Press F11 to debug the application on the Android Emulator.

Step 4 : When the activity is first loaded, you should see the following in the LogCat window (click on the Debug perspective; see also Fig. 2.3):

11-27 13:02:36.783 2539-2539/com.example.kkamil.learnfromkamil D/Events: In onCreate() event

11-27 13:02:36.786 2539-2539/com.example.kkamil.learnfromkamil D/Events: In the onStart()event

11-27 13:02:36.789 2539-2539/com.example.kkamil.learnfromkamil D/Events: In the onResume()

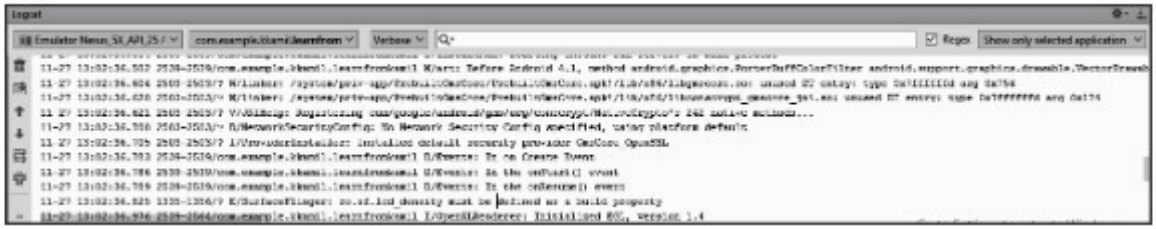


Fig. 2.3

2.3 INTRODUCTION TO INTENTS

- Intent is an abstract description of an operation to be performed.

Intent:

- The Android operating system uses an asynchronous messaging mechanism to match task requests with the appropriate Activity. Each request is packaged as an Intent. The intent generally consists of activities to be done, the parameter over which this activity is to be performed, and the application to perform this action.
- Examples of processes defined as intents:
  - Start of certain activities.
  - Displaying a web page.
  - Answering a call.
  - Dialing a number.
- Intentions can be divided into two categories: Implicit and Explicit.

1. **Explicit Intents:**
   - Explicit intent going to be connected internal world of application,suppose if you wants to connect one activity to another activity, we can do this quote by explicit intent, below image is connecting first activity to second activity by clicking button.
   - These intents designate the target component by its name and they are typically used for application-internal messages - such as an activity starting a subordinate service or launching a sister activity.

     For example:

```
// Explicit Intent by specifying its class name
Intent i = new Intent(FirstActivity.this, SecondActivity.class);

// Starts TargetActivity
startActivity(i);
```

2. **Implicit Intents:**

- These intents do not name a target and the field for the component name is left blank. Implicit intents are often used to activate components in other applications.

  For example:

```
Intent read1=new Intent();
read1.setAction(android.content.Intent.ACTION_VIEW);
read1.setData(ContactsContract.Contacts.CONTENT_URI);
startActivity(read1);
```

2.4 LINKING ACTIVITIES USING INTENTS

- An Android application can contain zero or more activities. When your application has more than one activity, you may need to navigate from one activity to another. In Android, you navigate between activities through what is known as **intent**.
- Following example shows how to link activities using intents:

Step 1 : Using the Activities project created earlier, add the following statements in bold to the AndroidManifest.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http:
//schemas.android.com/apk/res/android"
package="com.example.kkamil.learnfromkamil">
<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/AppTheme">
<activity android:name=".MainActivity">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<activity android:name=".Activity2">
<intent-filter>
<action android:name="android.intent.action.Activity2" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>

</application>

</manifest>
```

Step 2 : Right click on the package name under the java folder and select New -> Activity -> Empty Activity as shown in Fig. 2.4.

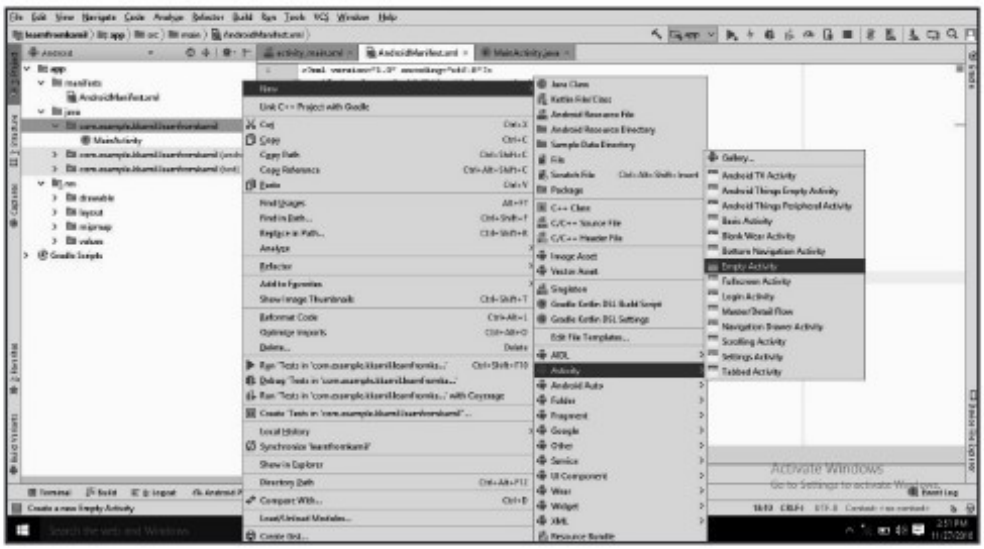


Fig. 2.4

Step 3 : Name the new class file Activity2 (see Fig. 2.5) and click **Finish.**

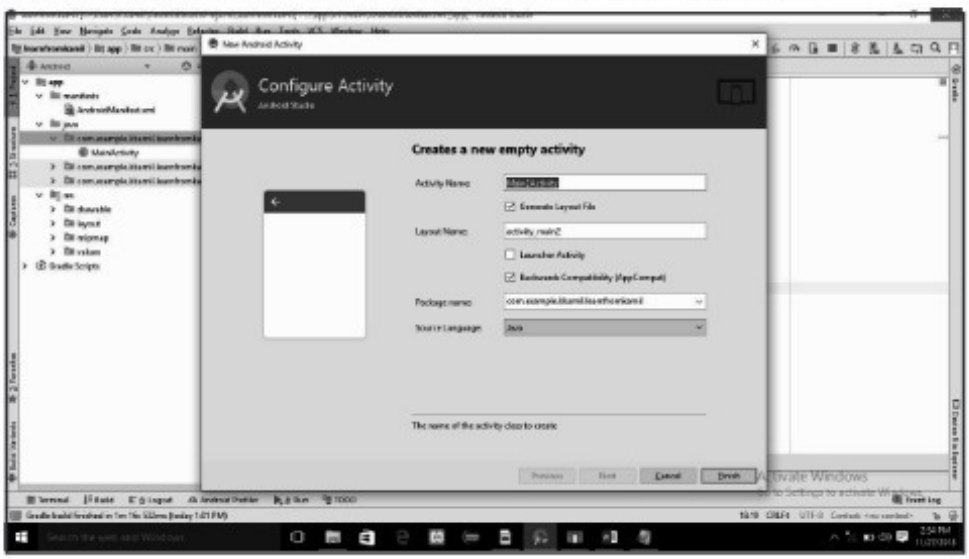


Fig. 2.5

Step 4 : Make a sure that activity_2.xml is created. Right-click on the res/layout folder if file not created then name the file activity_2.xml. The res/layout folder will now contain the activity_2.xml file (see Fig. 2.6).

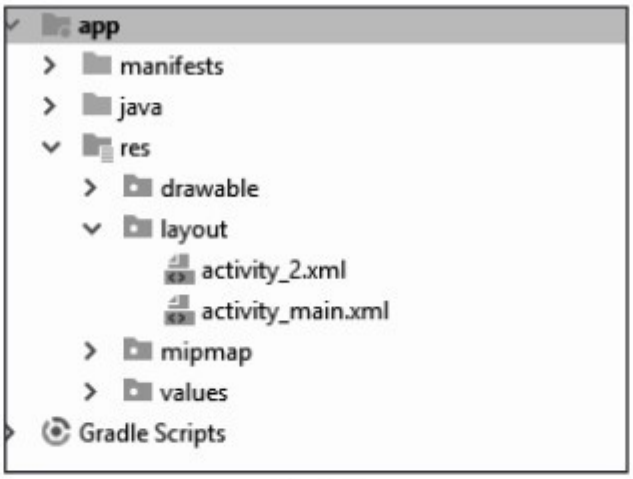


Fig. 2.6

Step 5 : Click on activity_2.xml and then click on Design tab and drag-drop Text View on Activity 2 GUI as follows:

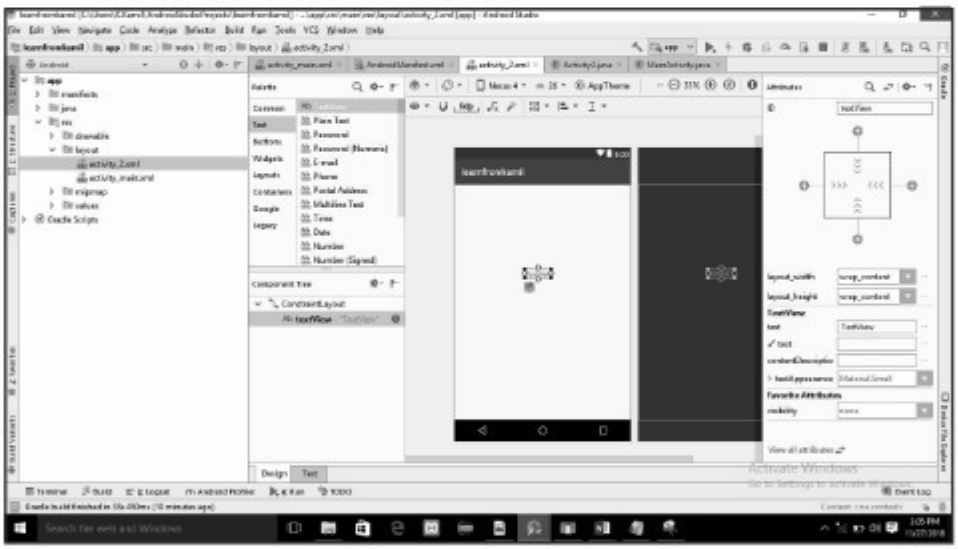


Fig. 2.7

And Edit Text as follow:

```
<TextView
android:id="@+id/textView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Activity 2 "
tools:layout_editor_absoluteX="153dp"
tools:layout_editor_absoluteY="185dp" />
```

Step 6 : In the Activity2.java file, add the following statements in bold:

```
package com.example.kkamil.learnfromkamil;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class Activity2 extends AppCompatActivity
{
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_2);
   }
}
```

Step 7 : Modify the MainActivity.java file as shown in bold:

```
package com.example.kkamil.learnfromkamil;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
public class MainActivity extends AppCompatActivity
{
@Override
public void onCreate(Bundle savedInstanceState)
{
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   }
@Override
public void onBackPressed()
{
   super.onBackPressed();
   Intent intent=new Intent(this,Activity2.class);
   startActivity(intent);
}
}
```

Step 8 : Debug the application on the Android Emulator. When the first activity is loaded, click the back button of the directional pad (see Fig. 2.8, on a real device this can be achieved by pressing back button). The second activity will now be loaded.

Fig. 2.8

2.5 CALLING BUILT-IN APPLICATIONS USING INTENTS

- Until this point, you have seen how to call activities within your own application. One of the key aspects of Android programming is using the intent to call activities from other applications.
- In particular, your application can call the many built-in applications that are included with an Android device.
- For example, if your application needs to enable a user to call a particular person saved in the Contacts application, you can simply use an Intent object to bring up the Contacts application, from which the user can select the person to call.
- This enables your application to present a consistent user experience, and enables you to avoid building another application to retrieve all the contacts in the Contacts application.

Step 1 : Using Android Studio, create a new Android project and name it Intents.

Step 2 : Add the following statements in bold to the main.xml file:

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

```
<Button
android:id="@+id/btn_webbrowser"
android:layout_width="match_parent"
android:layout_height="53dp"
android:layout_marginBottom="8dp"
android:layout_marginTop="48dp"
android:text="Web Browser"
app:layout_constraintBottom_toTopOf="@+id/button2"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.501"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent" />
<Button
android:id="@+id/button2"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginEnd="8dp"
android:layout_marginStart="8dp"
android:text="Call Button"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.502"
app:layout_constraintStart_toStartOf="parent"
tools:layout_editor_absoluteY="155dp" />
<Button
android:id="@+id/button3"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginStart="8dp"
android:layout_marginTop="28dp"
android:text="Show Map"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.0"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/button2" />
```

```
<Button
android:id="@+id/button4"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginBottom="156dp"
android:layout_marginTop="8dp"
android:text="Contacts"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.0"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/button3"
app:layout_constraintVertical_bias="1.0" />
</android.support.constraint.ConstraintLayout>
```

Step 3 : Add the following statements in bold to the MainActivity.java file:

```
package com.example.kkamil.intent;
import android.content.Intent;
import android.net.Uri;
import android.provider.ContactsContract;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity
{
   Button b1,b2,b3,b4;
   int request_Code= 1;
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      //---Web browser button---
      b1 = (Button) findViewById(R.id.btn_webbrowser);
```

```
        b1.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View arg0)
            {
                Intent    i    =    new    Intent(Intent.ACTION_VIEW,
                Uri.parse("http://www.amazon.com"));
                startActivity(i);
            }
        });
        //---Make calls button---
        b2 = (Button) findViewById(R.id.button2);
        b2.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View arg0)
        {
            Intent i = new
            Intent(Intent.ACTION_DIAL,
                                        Uri.parse("tel:+651234567"));
            startActivity(i);
        }
    });
    //---Show Map button---
    b3 = (Button) findViewById(R.id.button3);
    b3.setOnClickListener(new View.OnClickListener()
    {
        public void onClick(View arg0)
        {
            Intent i = new
            Intent(Intent.ACTION_VIEW,
                            Uri.parse("geo:37.827500,-122.481670"));
            startActivity(i);
            }
    });
        //---Choose Contact button---
    b4 = (Button) findViewById(R.id.button4);
```

```
        b4.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View arg0)
            {
                Intent i = new
                Intent(Intent.ACTION_PICK);
                i.setType(ContactsContract.Contacts.CONTENT_TYPE);
                startActivityForResult(i,request_Code);
            }
    });
}
}
```

Step 4 : Debug the application on the Android Emulator. Click on the Show Map button it display following results.

Fig. 2.9

2.6 INTRODUCTION TO FRAGMENTS

- A **Fragment** is a piece of an activity which enable more modular activity design. It will not be wrong if we say, a fragment is a kind of **sub-activity**.
- A Fragment represents a behavior or a portion of user interface in a FragmentActivity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).

- You create fragments by extending **Fragment** class and you can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a **<fragment>** element.
- Prior to fragment introduction, we had a limitation because we can show only a single activity on the screen at one given point in time. So we were not able to divide device screen and control different parts separately. But with the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time.
- Now we can have a single activity but each activity can comprise of multiple fragments which will have their own layout, events and complete life cycle.
- Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.

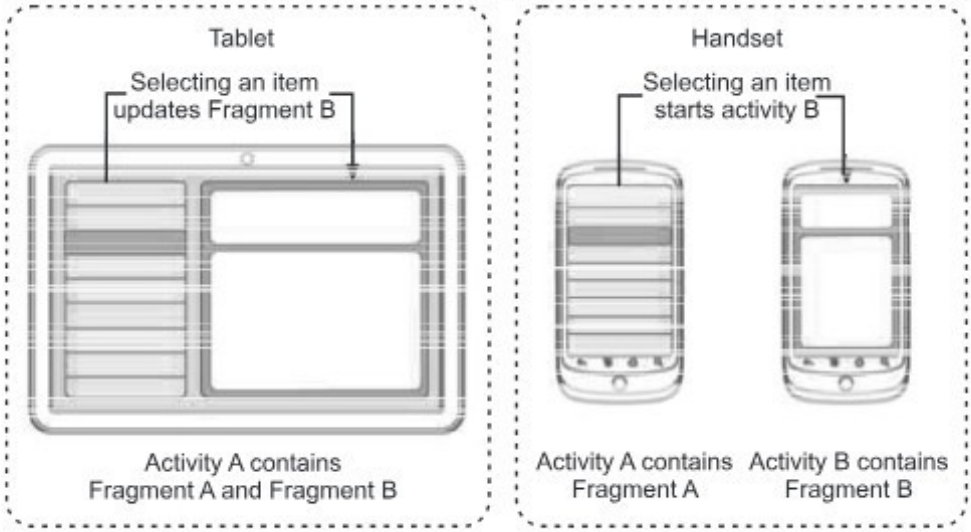


Fig. 2.10: Concept of Fragment

- The application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article.

Types of Fragments:

- Basically fragments are divided as three stages as shown below:
  - **Single frame fragments:** Single frame fragments are using for hand hold devices like mobiles. Here we can show only one fragment as a view.
  - **List fragments:** Fragments having special list view is called as list fragment.
  - **Fragments transaction:** Using with fragment transaction. We can move one fragment to another fragment.

- There are also a few subclasses that you might want to extend, instead of the base Fragment class:
  - **DialogFragment:** Displays a floating dialog. Using this class to create a dialog is a good alternative to using the dialog helper methods in the Activity class, because you can incorporate a fragment dialog into the back stack of fragments managed by the activity, allowing the user to return to a dismissed fragment.
  - **ListFragment:** Displays a list of items that are managed by an adapter (such as a SimpleCursorAdapter), similar to ListActivity. It provides several methods for managing a list view, such as the onListItemClick() callback to handle click events. (Note that the preferred method for displaying a list is to use RecyclerView instead of ListView. In this case you would need to create a fragment that includes a RecyclerView in its layout. See Create a List with RecyclerView to learn how.)
  - **PreferenceFragmentCompat:** Displays a hierarchy of Preference objects as a list, similar to PreferenceActivity. This is useful when creating a "settings" activity for your application.

Using Fragments

1. Using Android Studio, create a new Android project and name it Fragments.
2. In the res/layout folder, add a new layout resource file and name it `fragment1.xml`. Populate it with the following code. Be sure to change all instances of "com.jfdimarzio" to whatever package name your project is using.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#00FF00"
>
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="This is fragment #1"
android:textColor="#000000"
android:textSize="25sp" />

</LinearLayout>
```

3. Also in the res/layout folder, add another new layout resource file and name it `fragment2.xml`.

   Populate it as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#FFFE00"
>
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="This is fragment #2"
android:textColor="#000000"
android:textSize="25sp" />
</LinearLayout>
```

4. In `activity_main.xml`, add the bolded lines in the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:orientation="vertical"
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.jfdimarzio.fragments.MainActivity">
<fragment
android:name="com.jfdimarzio.fragments.Fragment1"
android:id="@+id/fragment1"
android:layout_weight="1"
android:layout_width="fill_parent"
android:layout_height="match_parent" />
<fragment
android:name="com.jfdimarzio.fragments.Fragment2"
android:id="@+id/fragment2"
```

```
android:layout_weight="1"
android:layout_width="fill_parent"
android:layout_height="match_parent" />
</LinearLayout>
```

5. Under the <Your Package Name>/fragments package name, add two Java class files and name them fragment1.java and fragment2.java.
6. Add the following code to Fragment1.java:

```
package com.jfdimarzio.fragments;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class Fragment1 extends Fragment
{
   @Override
   public View onCreateView(LayoutInflater inflater,
   ViewGroup container, Bundle savedInstanceState)
   {  //---Inflate the    layout for this fragment---
      return inflater.inflater(R.layout.fragment1, container, false);
   }
}
```

7. Add the following code to fragment2.java:

```
package com.jfdimarzio.fragments;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class Fragment2 extends Fragment
{
   @Override
   public View onCreateView(LayoutInflater inflater,
   ViewGroup container, Bundle savedInstanceState)
   {  //---Inflate the    layout    for    this    fragment---    return
      inflater.inflate(
      R.layout.fragment2, container, false);
   }
}
```

8. Press Shift+F9 to debug the application on the Android emulator. Figure 2.11 shows the two fragments contained within the activity.

Fig. 2.11

2.7 ADDING FRAGMENTS DYNAMICALLY

- While fragments enable you to compartmentalize your UI into various configurable parts, the real power of fragments is realized when you add them dynamically to activities during runtime.
- In reality, it is much more useful if you create fragments and add them to activities during runtime. This enables you to create a customizable user interface for your application. For example, if the application running on smartphone, you might fill an activity with single fragments; if the application is running on a tablet, you might then fill the activity with two or more fragments, as the tablet has mush more screen real estate compared to a smartphone.

Step 1 : Using the same project created in the previous section, modify the main.xml file by commenting out the two <fragment> elements. Be sure to change all instances of "com.jfdimarzio" to whatever package name your project is using.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:orientation="vertical"
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.jfdimarzio.fragments.MainActivity">
<!--
<fragment
android:name="com.jfdimarzio.fragments.Fragment1"
android:id="@+id/fragment1"
android:layout_weight="1"
android:layout_width="fill_parent"
android:layout_height="match_parent" />
<fragment
android:name="com.jfdimarzio.fragments.Fragment2"
android:id="@+id/fragment2"
android:layout_weight="1"
android:layout_width="fill_parent"
android:layout_height="match_parent" />
-->
</LinearLayout>
```

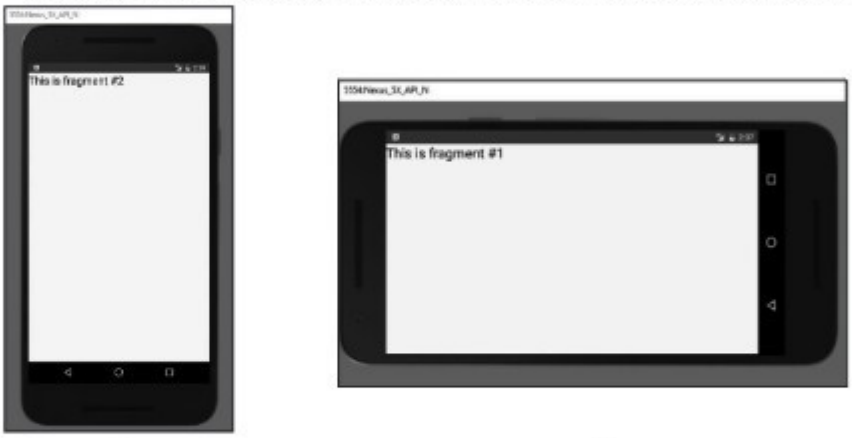
Step 2 : Add the bolded lines in the following code to the `MainActivity.java` file:

Fig. 2.12

```
package com.jfdimarzio.fragments;
import android.app.Activity;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.os.Bundle;
import android.util.DisplayMetrics;
public class MainActivity extends Activity
{
    /** Called when the activity is first created. */ @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
```

```
        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction fragmentTransaction =
        fragmentManager.beginTransaction(); //---get the current
        display info---
        DisplayMetrics display =
        this.getResources().getDisplayMetrics();
        int width = display.widthPixels;
        int height = display.heightPixels;
        if (width> height)
        {
           //---landscape mode---
           Fragment1 fragment1 = new Fragment1();
           android.R.id.content refers to the content
           view of the activity fragmentTransaction.replace(
           android.R.id.content, fragment1);
        }
        else
        {
           //---portrait mode---
           Fragment2 fragment2 = new Fragment2();
           fragmentTransaction.replace(
           android.R.id.content, fragment2);
        }
        fragmentTransaction.commit();
    }
}
```

Step 3 : Press Shift + F9 to run the application on the Android emulator. Observe that when the emulator is in portrait mode, Fragment 2 is displayed (see Fig. 2.13(a)). If you press Ctrl+Left to change the orientation of the emulator to landscape, Fragment 1 is shown instead (see Fig. 2.13(b)).



(a) (b)

Fig. 2.13

2.8 LIFECYCLE OF FRAGMENT

- Android fragments have their own life cycle very similar to an android activity. This section briefs different stages of its life cycle.
- The Fragment lifecycle begins when it is attached to an activity. Below we have shown the complete lifecycle of fragment in form of a flow chart.

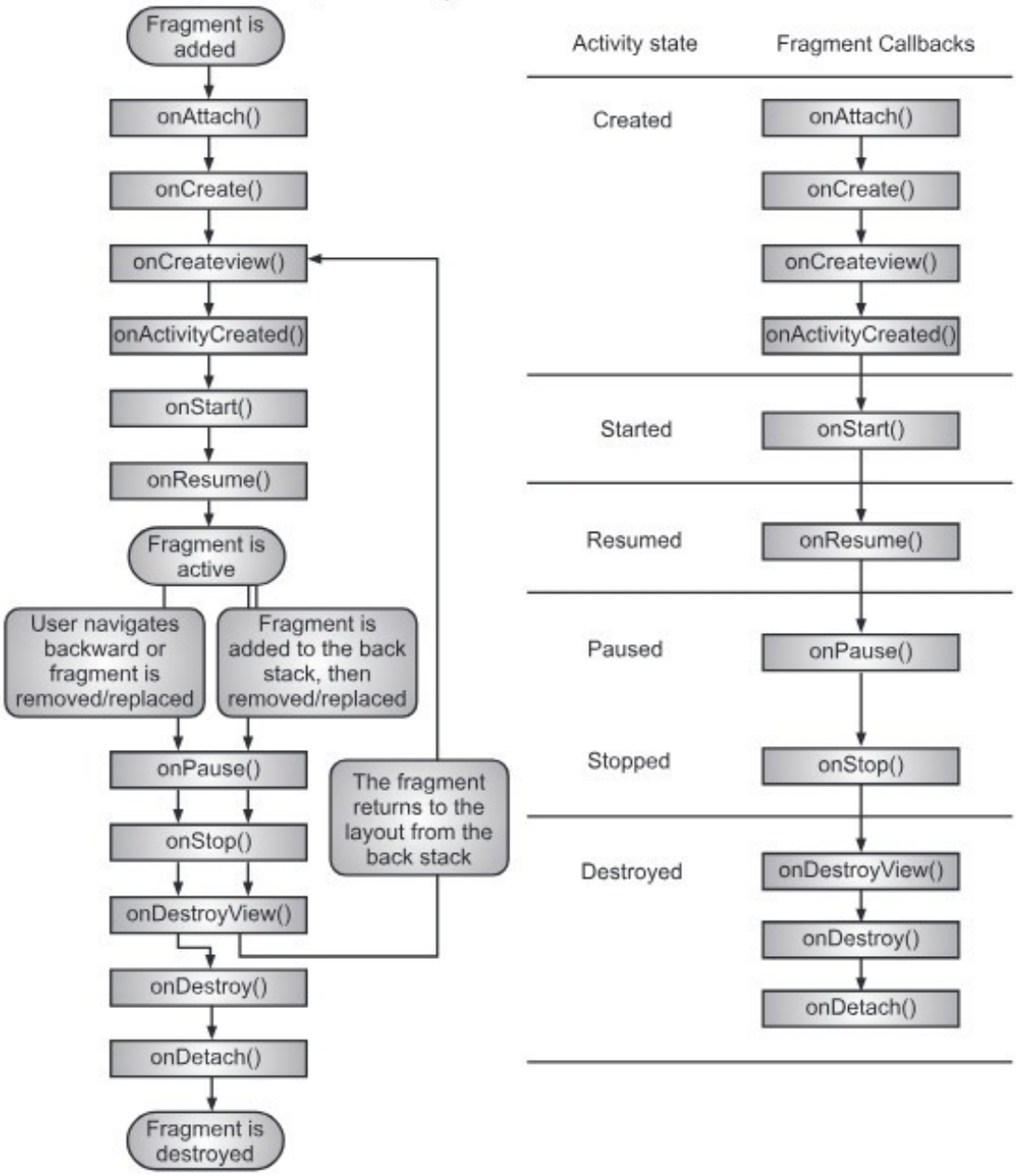


Fig. 2.14

- Here is the list of methods which you can to override in your Fragment class:

1. onAttach():

- The fragment instance is associated with an activity instance. This method is called first, even before onCreate() method. This method let us know that our Fragment has been attached to an activity.
- Below is the example code of onAttach() method.

```
@Override
public void onAttach(Activity activity)
{
    super.onAttach(activity);
    // add your code here which executes when fragment instance is
    associated
}
```

2. onCreate():

- This will be called when creating the fragment. It means when a new fragment instance initializes, which always happens after it attaches to the host.
- Below is the example code of onCreate() method.

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    // add your code here which executes when fragment's instance
    initializes
}
```

3. onCreateView():

- The will be called when it's time for the fragment to draw its UI (User Interface) for the first time. To draw a UI for our fragment we must return a View component from this method that is the root of our fragment's layout. We can also return null if the fragment does not provide a UI.
- Below is the example code of onCreateView() method.

```
@Override
public View onCreateView(LayoutInflaterinflater, ViewGroup container,
Bundle savedInstanceState)
{
    View v = inflater.inflate(R.layout.fragment_test, container, false);
    // add your code here to draw the UI for the first time means in this
    method we can get the reference of the views which are created // in
    our xml file

    return v;
}
```

4. **onViewCreated():**

- This will be called after onCreateView() method. This method is particularly useful when inheriting the onCreateView() method implementation but we need to configure the resulting views such as with a ListFragment and when to set up an adapter.
- Below is the example code of onViewCreated() method.

```
@Override
public void onViewCreated(View view, Bundle savedInstanceState)
{
    super.onViewCreated(view, savedInstanceState);
    // add your code here which executes after the execution of
    onCreateView() method.
}
```

5. **onActivityCreated():**

- This method is called after the onCreateView() method when the host activity is created. This method indicates that the activity's onCreate() has completed.
- Below is the example code of onActivityCreated() method.

```
@Override
public void onActivityCreated(Bundle savedInstanceState)
{
    super.onActivityCreated(savedInstanceState);
    // add your code here which executes when the host activity is created.
}
```

6. **onStart():**

- This method is called once the fragment gets visible.
- Below is the example code of onStart() method.

```
@Override
public void onStart()
{
    super.onStart();
    // add your code here which executes when the Fragment gets visible.
}
```

7. **onResume():**

- This method is called when the Fragment is visible and intractable.
- Below is the example code of onResume() method.

```
@Override
public void onResume()
{
    super.onResume();
    // add your code here which executes when the Fragment is visible and
    intractable.
}
```

8. onPause():

- This method is the first indication that the user is leaving the current fragment or fragment is no longer intractable. It occurs when any Fragment Transition processed or Fragment is removed.
- Below is the example code of onPause() method.

```
@Override
public void onPause()
{
    super.onPause();
    // add your code here which executes when user leaving the current
    fragment or fragment is no longer intractable.
}
```

9. onStop():

- This method is called after onPause() method. Fragment going to be stopped by calling onStop(). This method calls when the Fragment is no longer visible.
- It occurs either after the fragment is about to be removed or Fragment Transition is processed(replace Fragment) or when the host activity stops.
- Below is the example code of onStop() method.

```
@Override
public void onStop()
{
    super.onStop();
    // add your code here which executes Fragment going to be stopped.
}
```

10. onDestroyView():

- This method is called when the view and other related resources created in onCreateView() method is removed from the activity's view hierarchy and destroyed.
- Below is the example code of onDestroyView() method.

```
@Override
public void onDestroyView()
{
    super.onDestroyView();
    // add your code here which executes when the view's and other related
    resources created in onCreateView() method are removed
}
```

11. onDestroy():

- This method is called to do final clean up of the Fragment's state but not guaranteed to be called by the Android platform. This method called after onDestroyView() method.

- Below is the example code of onDestroy() method.

```
@Override
public void onDestroy()
{
    super.onDestroy();
    // add your code here which executes when the final clean up for the
    Fragment's state is needed.
}
```

12. onDetach():

- This method called after onDestroy() method to notify that the fragment has been disassociated from its hosting activity means Fragment is detached from its host Activity.
- Below is the example code of onDetach() method.

```
@Override
public void onDetach()
{
    super.onDetach();
    // add your code here which executes when fragment has been
    disassociated from its hosting activity
}
```

2.9 INTERACTIONS BETWEEN FRAGMENTS

- Very often, an activity may contain one or more fragments working together to present a coherent UI to the user. In this case, it is very important for fragments to communicate with one another and exchange data.
- A Fragment represents behaviour or a portion of user interface in an Activity. Fragments must be embedded with activities, they cannot run independent.
- Single Activity can use multiple fragments and fragment usable in several activities.Each fragments has its own lifecycle, closely associated with the lifecycle of its host activity.
- A fragment may be a static a part of an activity or instantiated automatically during the activity's creation or you can create(), add(), and remove() fragments dynamically in an activity at run-time.

1. Using the same project created in the previous section, add the following bolded statement to the Fragment1.xml file. Be sure to change all instances of "com.jfdimarzio" to whatever package name your project is using.

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

```
android:layout_height="fill_parent"
android:background="#00FF00" >
<TextView
android:id="@+id/lblFragment1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="This is fragment #1"
android:textColor="#000000"
android:textSize="25sp" />
</LinearLayout>
```

2. Add the following bolded lines to `fragment2.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#FFFE00" >
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="This is fragment #2"
android:textColor="#000000"
android:textSize="25sp" />
<Button
android:id="@+id/btnGetText"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Get text in Fragment #1"
android:textColor="#000000"
android:onClick="onClick" />
</LinearLayout>
```

3. Return the two fragments to `main.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:orientation="vertical"
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.jfdimarzio.fragments.MainActivity">
<fragment
android:name="com.jfdimarzio.fragments.Fragment1"
android:id="@+id/fragment1"
android:layout_weight="1"
android:layout_width="fill_parent"
android:layout_height="match_parent" />
<fragment
android:name="com.jfdimarzio.fragments.Fragment2"
android:id="@+id/fragment2"
android:layout_weight="1"
android:layout_width="fill_parent"
android:layout_height="match_parent" />
</LinearLayout>
```

4. Modify the `MainActivity.java` file by commenting out the code that you added in the earlier sections. It should look like this after modification:

```
public class MainActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        /*
        FragmentManager    fragmentManager    =    getFragmentManager();
        FragmentTransaction fragmentTransaction =
        fragmentManager.beginTransaction();
        //---get the current display info---
        DisplayMetrics display = this.getResources().getDisplayMetrics();
        int width = display.widthPixels;
        int height = display.heightPixels;
        if (width> height)
        {
            //---landscape mode---
            Fragment1 fragment1 = new Fragment1();
            android.R.id.content refers to the content
            view of the activity fragmentTransaction.replace(
            android.R.id.content, fragment1);
        }
```

```
        else
        {
            //---portrait mode---
            Fragment2 fragment2 = new Fragment2();
            fragmentTransaction.replace(
            android.R.id.content, fragment2);
        }
        fragmentTransaction.commit();
        */
    }
}
```

5. Add the following bolded statements to the `Fragment2.java` file:

```
package com.jfdimarzio.fragments;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
public class Fragment2 extends Fragment
{
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState)
    {
        //---Inflate the layout for this fragment---
        return inflater.inflate(
        R.layout.fragment2, container, false);
    }
    @Override
    public void onStart()
    {
        super.onStart();
        //---Button view---
        Button btnGetText = (Button)
        getActivity().findViewById(R.id.btnGetText);
```

```
        btnGetText.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View v)
            {
                TextView lbl = (TextView)
                getActivity().findViewById(R.id.lblFragment1);
                Toast.makeText(getActivity(), lbl.getText(),
                                                Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

6. Press Shift+F9 to debug the application on the Android emulator. In the second fragment on the right, click the button. You should see the Toast class displaying the text this is fragment #1.

2.10 TOAST

- In android, Toast is a small popup notification that is used to display information about the operation which we performed in our app.
- The Toast will show the message for a small period of time and it will disappear automatically after a timeout.
- The size of Toast will be adjusted based on the space required for the message and it will be displayed on the top of the main content of activity for a short period of time.
- For example, some of the apps will show a message like "**Press again to exit**" in toast, when we pressed a back button on the home page or showing a message like "**saved successfully**" toast when we click on the button to save the details.
- The syntax of creating a **Toast** in android applications:

```
Toast.makeText(context, "message", duration).show();
```

- There are two ways to define the Toast **duration** which as follows:
  1. LENGTH_SHORT
  2. LENGTH_LONG

Summary

- The Android activity life cycle defines the states or events that an activity goes through from its creation until its end. The activity monitors and reacts to these events by executing methods that override the activity class methods for each event.
- The intent generally consists of activities to be done, the parameter over which this activity is to be performed and the application to perform this action.
- Intents can be either explicit or implicit. An explicit intent calls a specific service or activity explicitly; whereas an implicit intent just gives the definition of the required service.

- An Intent object is a bundle of information which is used by the component that receives the intent as well as information used by the Android system. An intent filter is an expression in an app's manifest file that specifies the type of intents that the component would like to receive.
- Fragment represents a portion of user interface in an Activity.
- Various methods of Activity life cycle includes onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy() and onRestart().
- Toast is a small popup notification that is used to display information about the operation which we performed in our app.

Practice Questions

Q.I Multiple Choice Questions:

1. _____ dictates the UI and handles the user interaction to the smart phone screen.
   (a) Intents (b) Fragments
   (c) Activities (d) None of the Mentioned
2. An _____ is a messaging object that you can use to request an action from an app component which basically an intention to do an action.
   (a) activity (b) intent
   (c) fragment (d) None of the Mentioned
3. A _____ in Android is a component which can be used over an activity to define an independent modular UI component attached to the activity and its functions independently, but as it is linked to the Activity.
   (a) Intent (b) Fragment
   (c) Activities (d) None of the Mentioned
4. An _____ object is a bundle of information which is used by the component that receives the intent as well as information used by the Android system.
   (a) Intent (b) Fragment
   (c) Activity (d) All of the Mentioned
5. Android Intent is the _____ that is passed between components such as activities, content providers, broadcast receivers, services etc.
   (a) message (b) request
   (c) query (d) All of the Mentioned
6. A fragment can be used in _____ activities.
   (a) single (b) multiple
   (c) Both (a) and (b) (d) None of the Mentioned
7. When you just have to tell what action you want to perform without worrying which component will perform it, then you can use _____ intent.
   (a) explicit (b) implicit
   (c) hybrid (d) All of the Mentioned

8. An _____ is an expression in an application's manifest file that specifies the type of intents that the component would like to receive.
   (a) activity filter (b) fragment filter
   (c) intent filter (d) None of the Mentioned
9. A _____ is an independent Android component which can be used by an activity which encapsulates functionality so that it is easier to reuse within activities and layouts.
   (a) Intent (b) Fragment
   (c) Activity (d) All of the Mentioned

Answers

1. (c)	2. (b)	3. (b)	4. (a)	5. (a)	6. (b)	7. (b)	8. (c)	9. (b)

Q.II Answer the following Questions:

1. What is activity?
2. What is intent?
3. With the help of diagram describe activity life cycle.
4. How to linking activities using intents? Explin with example.
5. Describe lifecycle of fragment diagrammatically.
6. Enlist type of intents.
7. How to calling built-in applications using intents? Explain using example.

Q.III Answer the following questions in short:

1. Write short note on: Adding Fragments dynamically.
2. Write short note on: Interaction between fragments?
3. What is meant by fragment?

Q.IV Define the terms:

1. Intent
2. Fragment
3. Toast

■■■

3...

Android User Interface

Learning Objectives ...

Students will be able to:

- To Learn Basic Concepts of Android UI.
- To Understand Layouts and Layout Managers in Android.
- To Learn Screen in Android.

3.1 UNDERSTANDING THE COMPONENTS OF A SCREEN

- The term Android device covers a vast array of tablet and smart phone products with different screen sizes and resolutions.
- As a result, application user interfaces now must be carefully designed to ensure correct presentation on as wide a range of display sizes as possible. A key part of this is ensuring that the user interface layouts resize correctly when run on different devices.
- It is also important to keep in mind that the majority of Android based smartphone and tablets can be held by the user in both portrait and landscape orientations. A well-designed user interface should be able to adapt to such changes and make sensible layout changes to utilize the available screen space in each orientation.
- The basic unit of an Android application is an activity and it displays the user interface of your application.
- The activity may contain widgets such as buttons, labels, textboxes, and so on. Typically, you define your UI using an XML file (for example, the activity_main.xml file located in the res/layout folder of your project), which looks similar to what is shown in here.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:fitsSystemWindows="true"
tools:context="com.jfdimarzio.helloworld.MainActivity">
<android.support.design.widget.AppBarLayout
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:theme="@style/AppTheme.AppBarOverlay">
<android.support.v7.widget.Toolbar
android:id="@+id/toolbar"
android:layout_width="match_parent"
android:layout_height="?attr/actionBarSize"
android:background="?attr/colorPrimary"
app:popupTheme="@style/AppTheme.PopupOverlay"/>
</android.support.design.widget.AppBarLayout>
<include layout="@layout/content_main"/>
<android.support.design.widget.FloatingActionButton
android:id="@+id/fab"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="bottom|end"
android:layout_margin="@dimen/fab_margin"
android:src="@android:drawable/ic_dialog_email"/>
</android.support.design.widget.CoordinatorLayout>
```

- During runtime, you load the XML UI in the onCreate() method handler in your Activity class, using the setContentView() method of the Activity class:

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

3.1.1 Views and View Groups

- An activity contains views and ViewGroups.
- A view is a widget that has an appearance on screen. Examples of views are buttons, labels, and text boxes. A view derives from the base class android.view.View.

- The basic building block for User Interface (UI) is a View. Views are user interface elements that display data and respond to user actions. The Android SDK provides a set of pre-built views that can be used to construct a user interface such as CheckBox, ProgressBar etc.
- One or more views can be grouped into a ViewGroup. A ViewGroup (which is itself a special type of view) provides the layout in which you can order the appearance and sequence of views. Examples of ViewGroups include RadioGroup and ScrollView.
- A ViewGroup derives from the base class android.view.ViewGroup.
- Another type of ViewGroup is a Layout. A Layout is another container that derives from android .view.ViewGroup and is used as a container for other views. However, whereas the purpose of a ViewGroup is to group views logically such as a group of buttons with a similar purpose a Layout is used to group and arrange views visually on the screen.
- In short, the SDK also includes a set of views referred to as Layouts. A layout defines the structure for a user interface in your app, such as an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects.
- A View usually draws something the user can see and interact with. Where as a ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects.
- The Layouts available in Android are LinearLayout, TableLayout, FrameLayout, RelativeLayout, AbsoluteLayout, GridLayout, etc.
- LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
- RelativeLayout is a viewgroup that displays child views in relative positions.
- TableLayout is a view that groups views into rows and columns.
- AbsoluteLayout enables you to specify the exact location of its children.
- The FrameLayout is a placeholder on screen that you can use to display a single view. The FrameLayout is the most basic of the Android layouts. FrameLayouts are built to hold one view. As with all things related to development, there is no hard rule that FrameLayouts can't be used to hold multiple views. However, there is a reason why FrameLayouts were built the way they were.

3.1.2 LinearLayout

- The LinearLayout arranges views in a single column or a single row as shown in Fig. 3.1. Child views can be arranged either horizontally or vertically, which explains the need for two different layouts. In LinearLayout, the one for horizontal rows of views and one for vertical columns of views.

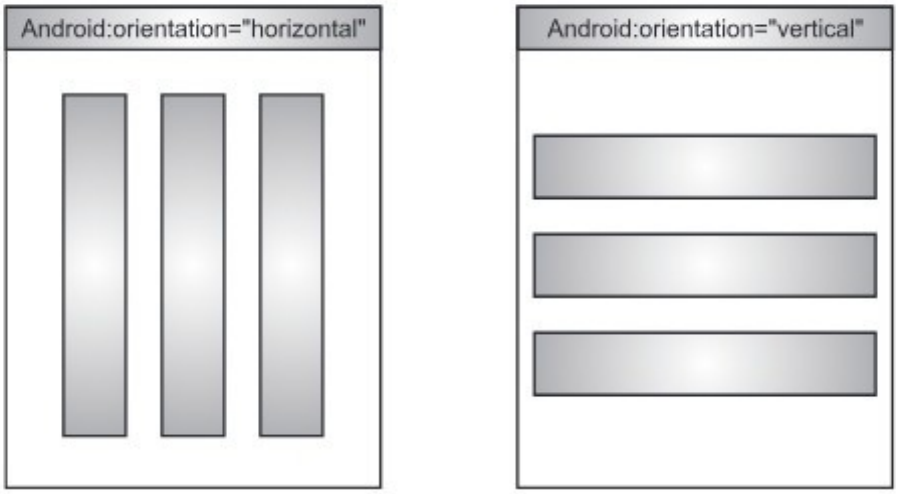


Fig. 3.1: LinearLayout in Android

- To see how LinearLayout works, consider the following elements typically contained in the activity_main.xml file:

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

- In the activity_main.xml file, observe that the root element is <LinearLayout> and it has a <TextView> element contained within it. The <LinearLayout> element controls the order in which the views contained within it appear.
- Each view and ViewGroup has a set of common attributes, some of which are described in Table 3.1.

Table 3.1: Common Attributes Used in Views and ViewGroups

Sr. No.	Attribute	Description
1.	layout_width	Specifies the width of the view or ViewGroup.
2.	layout_height	Specifies the height of the view or ViewGroup.
3.	layout_marginTop	Specifies extra space on the top side of the view or ViewGroup.
4.	layout_marginBottom	Specifies extra space on the bottom side of the view or ViewGroup.

contd ...

5.	layout_marginLeft	Specifies extra space on the left side of the view or ViewGroup.
6.	layout_marginRight	Specifies extra space on the right side of the view or ViewGroup.
7.	layout_gravity	Specifies how child views are positioned.
8.	layout_weight	Specifies how much of the extra space in the layout should be allocated to the view.
9.	layout_x	Specifies the x co-ordinate of the view or ViewGroup.
10.	layout_y	Specifies the y co-ordinate of the view or ViewGroup.

- For example, the width of the <TextView> element fills the entire width of its parent (which is the screen in this case) using the fill_parent constant. Its height is indicated by the wrap_content constant, which means that its height is the height of its content (in this case, the text contained within it). If you don't want the <TextView> view to occupy the entire row, you can set its layout_width attribute to wrap_content, like this:

```
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/hello"/>
```

- The preceding code sets the width of the view to be equal to the width of the text contained within it.
- Consider the layout in the next code snippet, which shows two views with their width explicitly stated as a measurement, and their heights set to the height of their contents.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">
<TextView
android:layout_width="100dp"
android:layout_height="wrap_content"
android:text="@string/hello"/>
<Button
android:layout_width="160dp"
android:layout_height="wrap_content"
android:text="Button"
android:onClick="onClick" />
</LinearLayout>
```

- Fig. 3.2 shows the screen of the Nexus5 (from the emulator). It has a 5-inch screen (diagonally), with a screen width of 2.72 inches. Its resolution is 1080 (width) ·1920 (height) pixels. The pixel density of a screen varies according to screen size and resolution.

Fig. 3.2: Screen of Nexus5

- To test how the views defined in the XML file look when displayed on screens of different densities, create two Android Virtual Devices (AVDs) with different screen resolutions and abstracted LCD densities. Fig. 3.3 (a) shows an AVD with 1080 × 1920 resolution and LCD density of 480.
- Fig. 3.3 (b) shows another AVD with 768 ·1280 resolution and LCD density of 320. Using the dp (density-independent pixel) unit ensures that your views are always displayed in the right proportion regardless of the screen density. Android automatically scales the size of the view depending on the density of the screen.

(a) (b)

Fig. 3.3: Android Virtual Device with difference Screen Resolutions

How to Convert dp to px?

- The formula for converting dp to px (pixels) is as follows:
  Actual pixels = dp * (dpi/160), where dpi is either 120, 160, 240, or 320.
  Therefore, in the case of the Button on a 235 dpi screen, its actual width is 160 * (240/160) = 240 px. When run on the 180 dpi emulator (regarded as a 160 dpi device), its actual pixel width is now 160 * (160/160) = 160 px. In this case, one dp is equivalent to one px.
- To prove that this is indeed correct, you can use the getWidth() method of a View object to get its width in pixels:

```
public void onClick(View view)
{
    Toast.makeText(this,
    String.valueOf(view.getWidth()),Toast.LENGTH_LONG).show();
}
```

- What if instead of using dp you now specify the size using pixels (px)?

```
<TextView
android:layout_width="100px"
android:layout_height="wrap_content"
android:text="@string/hello"/>

<Button
android:layout_width="160px"
android:layout_height="wrap_content"
android:text="Click Me"
android:onClick="onClick"/>
```

- Fig. 3.4 (a) shows how the Label and Button appear on a 480 dpi screen. Fig. 3.4 (b) shows the same views on a 320 dpi screen. In this case, Android does not perform any conversion because all the sizes are specified in pixels. If you use pixels for view sizes, the views appear smaller on a device with a high dpi screen than a screen with a lower dpi (assuming screen sizes are the same).

(a) (b)

Fig. 3.4: Label and Button on 480 dpi Screen

- The preceding example also specifies that the orientation of the layou t is vertical:

```
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">
```

- The default orientation layout is horizontal, so if you omit the android:orientation attribute, the views appear as shown in Fig. 3.5.

Fig. 3.5: LinearLayout

- In LinearLayout, you can apply the layout_weight and layout_gravity attributes to views contained within it, as the modifications to activity_main.xml in the following code snippet show:

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

android:orientation="vertical">
<Button
android:layout_width="160dp"
android:layout_height="0dp"
android:text="Button"
android:layout_gravity="left"
android:layout_weight="1"/>
```

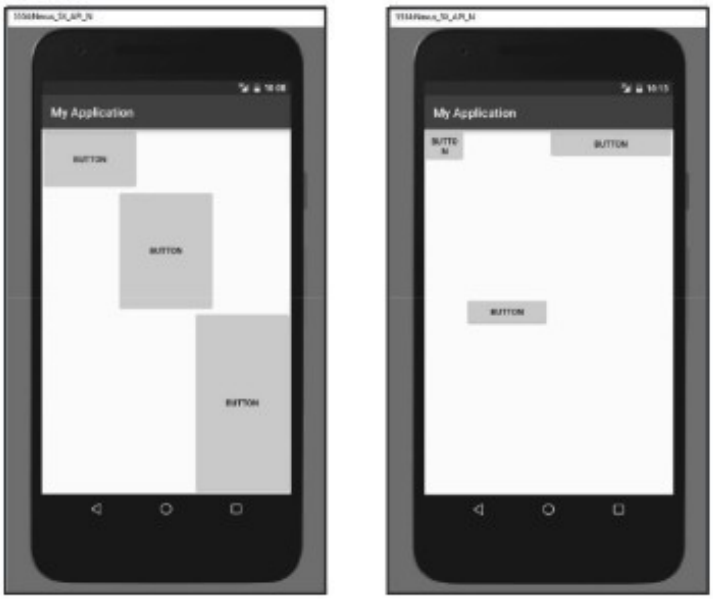
```
<Button
android:layout_width="160dp"
android:layout_height="0dp"
android:text="Button"
android:layout_gravity="center"
android:layout_weight="2"/>
<Button
android:layout_width="160dp"
android:layout_height="0dp"
android:text="Button"
android:layout_gravity="right"
android:layout_weight="3"/>
</LinearLayout>
```

- Fig. 3.6 shows the positioning of the views as well as their heights. The layout_gravity attribute indicates the positions the views should gravitate toward, whereas the layout_weight attribute specifies the distribution of available space. In the preceding example, the three buttons occupy about 16.6 percent (1/(1+2+3) * 100), 33.3 percent (2/(1+2+3) * 100), and 50 percent (3/(1+2+3) * 100) of the available height, respectively.
- If you change the orientation of the LinearLayout to horizontal (as shown in the following code snippet), you need to change the width of each view to 0 dp. The views display as shown in Fig. 3.6.

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

android:orientation="horizontal">
<Button
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Button"
android:layout_gravity="left"
android:layout_weight="1" />
<Button
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="Button"
android:layout_gravity="center_horizontal"
android:layout_weight="2" />
<Button
android:layout_width="0dp"
```

```
android:layout_height="wrap_content"
android:text="Button"
android:layout_gravity="right"
android:layout_weight="3" />
</LinearLayout>
```



(a) (b)

Fig. 3.6: LinearLayout (Horizontal)

Combine Two LinearLayouts with Different Orientations:

- Use two LinearLayouts, one with a vertical orientation and one with a horizontal orientation to place three TextViews and three Buttons in the configuration shown in Fig. 3.7.

Fig. 3.7: LinearLayout with difference Orientations

Step 1 : Create a new layout resource file named linearlayouts_example.xml. By default, this new file will be created with a LinearLayout (Vertical).

Step 2 : Place three stacked TextViews in a single column within the LinearLayout.

Step 3 : Place a LinearLayout (Horizontal) under the third TextView.

Step 4 : Place three buttons in a row within the LinearLayout (Horizontal).

3.1.3 AbsoluteLayout

- An AbsoluteLayout lets you specify exact locations (x/y co-ordinates) of its children. AbsoluteLayouts are less flexible and harder to maintain than other types of layouts without absolute positioning.

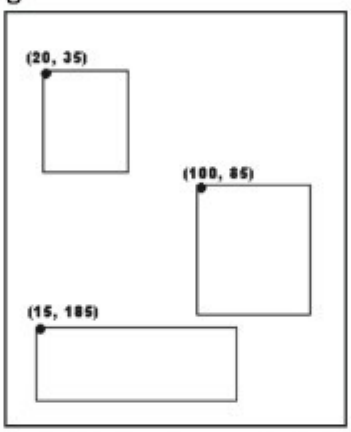


Fig. 3.8: AbsoluteLayout in Android

- Below is the example code of AbsoluteLayout in which we design a login screen with two field user name and password and one button for login. We set the all views using x and y co-ordinates of the screen and set the values in px (pixels). Below is the final output and code:

Step 1 : Create a new project and name it AbsoluteLayout Example.

Select File ·New ·New Project. Fill the forms and click "Finish" button.

Step 2 : Open res ·layout ·activity_main.xml (or) main.xml and add following code. Here we are designing a login form inside AbsoluteLayout.

```
<AbsoluteLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TextView
android:layout_x="110px"
android:layout_y="110px"
android:text="User Name"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
<EditText
```

```
android:layout_x="250px"
android:layout_y="80px"
android:width="100px"
android:layout_width="200dp"
android:layout_height="wrap_content"/>
<TextView
android:layout_x="110px"
android:layout_y="200px"
android:text="Password"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
<EditText
android:layout_x="250px"
android:layout_y="150px"
android:width="100px"
android:layout_width="200dp"
android:layout_height="wrap_content"/>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Log In"
android:layout_x="300px"
android:layout_y="300px"/>
</AbsoluteLayout>
```

Step 3 : Now Open java package -MainActivity.java and paste the below code.

```
package example.abhiandriod.absolytelayoutexample;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends AppCompatActivity{

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

@Override
public boolean onCreateOptionsMenu(Menu menu) {
//Inflate the menu; this adds items to the action bar if it is
present.
```

```
getMenuInflater().inflate(R.menu.menu_main, menu);
return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item){
//Handle action bar item clicks here. The action bar will
//automatically handle clicks on the Home/Up button, so long
// as you specify a parent activity in AndroidManifest.xml.
int id=item.getItemId();

//noinspection SimplifiableIfStatement
if (id == R.id.action_settings) {
return true;
}

return super.onOptionsItemSelected(item);
}
}
```

Step 4 : Now Open Manifests and click on AndroidManifest.xml and paste the following code.

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

<application
android:allowBackup="true"
android:icon="@mipmap/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>
```

Step 5 : Lastly Open res -values -strings.xml and paste the below code.

```
<resources>
<string name="app_name">AbsolyteLayoutExample</string>
<string name="hello_world">Hello world!</string>
<string name="action_settings">Settings</string>
</resources>
```

Output:

Fig. 3.9: AbsoluteLayout

3.1.4 TableLayout

- The TableLayout Layout groups views into rows and columns.
- You use the <TableRow> element to designate a row in the table. Each row can contain one or more views. Each view you place within a row forms a cell. The width of each column is determined by the largest width of each cell in that column.

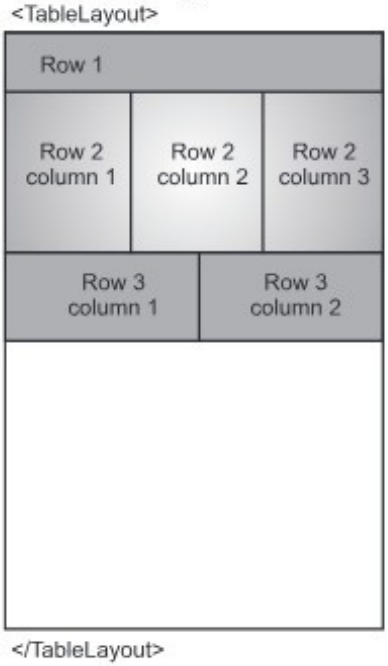


Fig. 3.10: TableLayout in Android

- Consider the content of activity_main.xml shown here:

```
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_height="fill_parent"
android:layout_width="fill_parent"><TableRow>
<TextView
android:text="User Name:"
android:width="120dp"
/>
<EditText
android:id="@+id/txtUserName"android:width="200dp"/>
</TableRow>
<TableRow>
<TextView
android:text="Password:"
/>
<EditText
android:id="@+id/txtPassword"
android:inputType="textPassword"
/>
</TableRow>
<TableRow>
<TextView />
<CheckBox android:id="@+id/chkRememberPassword"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="RememberPassword"
/>
</TableRow>
<TableRow>
<Button
android:id="@+id/buttonSignIn"android:text="Log In"/>
</TableRow>
</TableLayout>
```

- Fig. 3.9 shows how the preceding code appears when rendered on the Android emulator.

(a) (b)

Fig. 3.11: TableLayout Screen

3.1.5 RelativeLayout

- The RelativeLayout layout enables you to specify how child views are positioned relative to each other.
- In RelativeLayout a group of child views in which each view is positioned and aligned relative to other views within the view group. In other words, the positions of the child views can be described in relation to each other or to the parent view group.

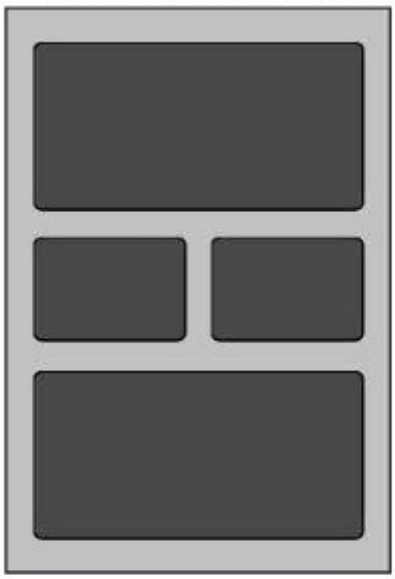


Fig. 3.12: RelativeLayout in Android

- Consider the following activity_main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
android:id="@+id/RLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"

xmlns:android="http://schemas.android.com/apk/res/android">
<TextView
android:id="@+id/lblComments"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Comments"
android:layout_alignParentTop="true"
android:layout_alignParentStart="true"/>
<EditText
android:id="@+id/txtComments"
android:layout_width="fill_parent"
android:layout_height="170dp"
android:textSize="18sp"
android:layout_alignStart="@+id/lblComments"
android:layout_below="@+id/lblComments"
android:layout_centerHorizontal="true"/>
<Button
android:id="@+id/btnSave"
android:layout_width="125dp"
android:layout_height="wrap_content"
android:text="Save"
android:layout_below="@+id/txtComments"
android:layout_alignEnd="@+id/txtComments"/>
<Button
android:id="@+id/btnCancel"
android:layout_width="124dp"
android:layout_height="wrap_content"
android:text="Cancel"
android:layout_below="@+id/txtComments"
android:layout_alignStart="@+id/txtComments"/>
</RelativeLayout>
```

- Notice that each view embedded within the RelativeLayout has attributes that enable it to align with another view. These attributes are as follows:

```
layout_alignParentTop
layout_alignParentStart
layout_alignStart
layout_alignEnd
layout_below
layout_centerHorizontal
```

- The value for each of these attributes is the ID for the view that you are referencing. The preceding XML UI creates the screen shown in Fig. 3.11.

Fig. 3.13: RelativeLayout Screen

3.1.6 FrameLayout

- The FrameLayout layout is a placeholder on screen that you can use to display a single view. Views that you add to a FrameLayout are always anchored to the top left of the layout.

Fig. 3.14: FrameLayout in Android

- Consider the following content in main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
android:id="@+id/RLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
xmlns:android=http://schemas.android.com/apk/res/android>
<TextView
android:id="@+id/lblComments"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello, Android!"
android:layout_alignParentTop="true"
android:layout_alignParentStart="true"/>
<FrameLayout
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignStart="@+id/lblComments"android:layout_below="@+id/
lblComments"
android:layout_centerHorizontal="true">
<ImageView
android:src="@mipmap/butterfly"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
</FrameLayout>
</RelativeLayout>
```

- Here, you have a FrameLayout within a RelativeLayout. Within the FrameLayout, you embed an ImageView. The UI is shown in Fig. 3.13.

Fig. 3.15: FrameLayout within RelativeLayout

- If you add another view (such as a Button view) within the FrameLayout, the view overlaps the previous view (See Fig. 3.14)

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
android:id="@+id/RLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/res/android">
<TextView
android:id="@+id/lblComments"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello, Android!"
android:layout_alignParentTop="true"
android:layout_alignParentStart="true"/>
<FrameLayout
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignStart="@+id/lblComments"
android:layout_below="@+id/lblComments"
android:layout_centerHorizontal="true">
<ImageView
android:src="@mipmap/butterfly"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
<Button
android:layout_width="124dp"
android:layout_height="wrap_content"
android:text="PrintPicture" />
</FrameLayout>
```

Fig. 3.16: Overlapping of FrameLayout

3.1.7 ScrollView

- A ScrollView is a special type of FrameLayout in that it enables users to scroll through a list of views that occupy more space than the physical display. The ScrollView can contain only one child view or ViewGroup, which normally is a LinearLayout.
- In ScrollView a group that contains one other child view and enables scrolling the child view.

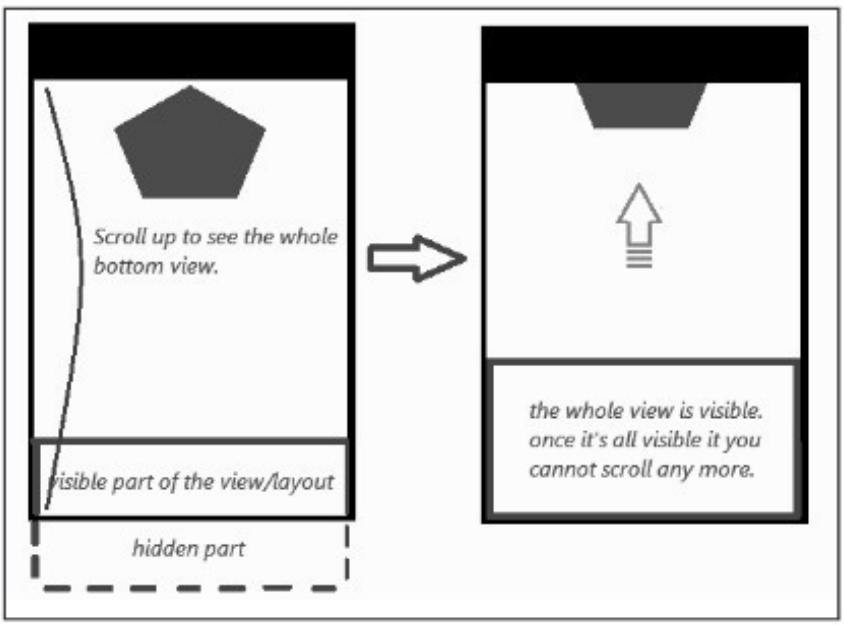


Fig. 3.17: AbsoluteLayout in Android

- The following main.xml content shows a ScrollView containing a LinearLayout, which is contains some Button and EditText views:

```
<ScrollView
android:layout_width="fill_parent"
android:layout_height="fill_parent"

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

<LinearLayout
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:orientation="vertical">
<Button
android:id="@+id/button1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 1" />
```

```
<Button
android:id="@+id/button2"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 2" />
<Button
android:id="@+id/button3"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 3" />
<EditText
android:id="@+id/txt"
android:layout_width="fill_parent"
android:layout_height="600dp"/>
<Button
android:id="@+id/button4"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 4" />
<Button
android:id="@+id/button5"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Button 5" />
</LinearLayout>
</ScrollView>
```

- If you load the preceding code on the Android emulator, you see something like what's shown in Fig. 3.18.

Fig. 3.18: AbsoluteLayout Screen

- Because the EditText automatically gets the focus, it fills up the entire activity (as the height was set to 600dp). To prevent it from getting the focus, add the following two bolded attributes to the

```
<LinearLayout> element:
<LinearLayout
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:orientation="vertical"
android:focusable="true"
android:focusableInTouchMode="true">
```

- Now you are able to view the buttons and scroll through the list of views (See Fig. 3.19).

Fig. 3.19: ScrollView with LinearLayout

- Sometimes you might want EditText to automatically get the focus, but you do not want the soft input panel (keyboard) to appear automatically (which happens on a real device). To prevent the keyboard from appearing, add the following bolded attribute to the <activity> element in the AndroidManifest.xml file:

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

3.1.8 ConstraintLayout

- ConstraintLayout is a layout on Android that gives you adaptable and flexible ways to create views for your apps.
- ConstraintLayout, which is now the default layout in Android Studio, gives you many ways to place objects.
- Advantages of using ConstraintLayout in Android:
  1. ConstraintLayout provides you the ability to completely design your UI with the drag and drop feature provided by the Android Studio design editor.
  2. It helps to improve the UI performance over other layouts.
  3. With the help of ConstraintLayout, we can control the group of widgets through a single line of code.
  4. With the help of ConstraintLayout, we can easily add animations to the UI components which we used in our app.
- Step by Step Implementation for adding ConstraintLayout in Android

Step 1 : Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. Note that select Java as the programming language.

Step 2 : Adding dependency for using ConstraintLayout in Android

Navigate to the app -> Gradle scripts -> build.gradle file and add the below dependency to it in the dependencies section.

implementation 'androidx.constraintlayout:constraintlayout:2.0.4'

Now sync your project and we will move towards working with activity_main.xml.

Step 3 : Working with the activity_main.xml file

Navigate to the app > res > layout > activity_main.xml and add the below code to that file. Below is the code for the activity_main.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MyActivity">
<TextView
   android:id="@+id/textView"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_marginLeft="16dp"
```

```
    android:layout_marginRight="16dp"
    android:gravity="center"
    android:padding="10dp"
    android:text="Kamil Khan"
    android:textColor="@color/black"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Fig. 3.20: Constraint Layout

3.2 ADAPTING TO DISPLAY ORIENTATION

- One of the key features of modern smartphones is their ability to switch screen orientation, and Android is no exception. Android supports two screen orientations namely portrait and landscape.
- By default, when you change the display orientation of your Android device, the current activity automatically redraws its content in the new orientation. This is because the onCreate() method of the activity is fired whenever there is a change in display orientation.
- However, when the views are redrawn, they may be drawn in their original locations (depending on the layouts elected). Fig. 3.20 shows the previous example displayed in landscape mode.

Fig. 3.21: Landscape mode

- In general, you can employ two techniques to handle changes in screen orientation:
  1. **Anchoring:** The easiest way is to "anchor" your views to the four edges of the screen. When the screen orientation changes, the views can anchor neatly to the edges.
  2. **Resizing and repositioning:** Whereas anchoring and centralizing are simple techniques to ensure that views can handle changes in screen orientation, the ultimate technique is resizing each and every view according to the current screen orientation.

3.2.1 Anchoring Views

- Anchoring can be easily achieved by using RelativeLayout. Consider the following main.xml file, which contains five Button views embedded within the <RelativeLayout> element:

```
<RelativeLayout
android:layout_width="fill_parent"
android:layout_height="fill_parent"

xmlns:android="http://schemas.android.com/apk/res/android">
<Button
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Top Left"
android:layout_alignParentStart="true"
android:layout_alignParentTop="true"/>
<Button
android:id="@+id/button2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

```
android:text="Top Right"
android:layout_alignParentTop="true"
android:layout_alignParentEnd="true"/>
<Button
android:id="@+id/button3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bottom Left"
android:layout_alignParentStart="true"
android:layout_alignParentBottom="true"/>
<Button
android:id="@+id/button4"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bottom Right"
android:layout_alignParentEnd="true"
android:layout_alignParentBottom="true"/>
<Button
android:id="@+id/button5"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Middle"
android:layout_centerVertical="true"
android:layout_centerHorizontal="true"/>
</RelativeLayout>
```

- Note the following attributes found in the various Button views:
  - layout_alignParentStart: Aligns the view to the left of the parent view.
  - layout_alignParentEnd: Aligns the view to the right of the parent view.
  - layout_alignParentTop: Aligns the view to the top of the parent view.
  - layout_alignParentBottom: Aligns the view to the bottom of the parent view.
  - layout_centerVertical: Centers the view vertically within its parent view.
  - layout_centerHorizontal: Centers the view horizontally within its parent view.
- Fig. 3.21 (a) shows the activity when viewed in portrait mode.
- When the screen orientation changes to landscape mode, the four buttons are aligned to the four edges of the screen, and the center button is centered in the middle of the screen with its width fully stretched (See Fig. 3.21 (b)).

(a) Portrait Mode

(b) Landscape Mode

Fig. 3.22

3.2.2 Resizing and Repositioning

- Resizing and repositioning in Android Studio are the techniques to ensure that views can be handle changes in screen orientation (portrait and landscape).
- An easier way to customize the UI based on screen orientation is to create a separate res/layout folder containing the XML files for the UI of each orientation.
- To support landscape mode, you can create a new folder in the **res** folder and name it as layout-land (representing landscape).
- The main.xml file contained within the layout folder defines the UI for the activity in portrait mode, where as the main.xml file in the layout-land folder defines the UI in landscape mode.

- The following code shows the content of main.xml under the layout folder:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
android:layout_width="fill_parent"
android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/res/android">
<Button
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Top Left"
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true"/>
<Button
android:id="@+id/button2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Top Right"
android:layout_alignParentTop="true"
android:layout_alignParentRight="true"/>
<Button
android:id="@+id/button3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bottom Left"
android:layout_alignParentLeft="true"
android:layout_alignParentBottom="true"/>
<Button
android:id="@+id/button4"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bottom Right"
android:layout_alignParentRight="true"
android:layout_alignParentBottom="true"/>
<Button
android:id="@+id/button5"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Middle"
android:layout_centerVertical="true"
android:layout_centerHorizontal="true"/>
</RelativeLayout>
```

- The following shows the content of main.xml under the layout-land folder:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
android:layout_width="fill_parent"
android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/res/android">
<Button
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Top Left"
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true"/>
<Button
android:id="@+id/button2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Top Right"
android:layout_alignParentTop="true"
android:layout_alignParentRight="true"/>
<Button
android:id="@+id/button3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bottom Left"
android:layout_alignParentLeft="true"
android:layout_alignParentBottom="true"/>
<Button
android:id="@+id/button4"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bottom Right"
android:layout_alignParentRight="true"
android:layout_alignParentBottom="true"/>
<Button
android:id="@+id/button5"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Middle"
android:layout_centerVertical="true"
android:layout_centerHorizontal="true"/>
```

```
<Button
android:id="@+id/button6"
android:layout_width="180px"
android:layout_height="wrap_content"
android:text="Top Middle"
android:layout_centerVertical="true"
android:layout_centerHorizontal="true"
android:layout_alignParentTop="true"/>
<Button
android:id="@+id/button7"
android:layout_width="180px"
android:layout_height="wrap_content"
android:text="Bottom Middle"
android:layout_centerVertical="true"
android:layout_centerHorizontal="true"
android:layout_alignParentBottom="true"/>
</RelativeLayout>
```

Output:

Fig. 3.23

3.3 SPLIT SCREEN / MULTI-SCREEN ACTIVITIES

- Android enables multiple apps to share the same screen simultaneously. The system can display two apps side by side (split-screen mode), one app in a small window overlaying other apps (picture-in-picture mode), or individual apps in separate movable, resizeable windows (free-form mode).
- Android 7.0 provides split-screen mode on handheld devices and picture-in-picture mode on TVs.

- Split-screen mode fills the screen with two apps, showing them either side by side or one above the other. Users can drag the divider separating the two apps to make one larger and the other smaller.
- Manufacturers of larger devices can choose to enable free-form mode, in which users can freely resize each activity.
- You can configure how your app handles multi-window display by specifying your activity's minimum allowable dimensions. You can also disable multi-window display for your app by setting resizeableActivity="false" to ensure the system always shows your app full screen.
- Android 8.0 extends picture-in-picture mode to handheld devices.
- Android 12 makes multi-window mode standard behavior.
  1. On large screens (sw >= 600dp), the platform supports all apps in multi-window mode regardless of app configuration. If resizeableActivity="false", the app is put into compatibility mode when necessary to accommodate display dimensions.
  2. On small screens (sw < 600dp), the system checks an activity's minWidth and minHeight to determine whether the activity can run in multi-window mode. If resizeableActivity="false", the app is prevented from running in multi-window mode regardless of minimum width and height.
- In multi-window mode on Android 9 (API level 28) and lower, only the activity the user has most recently interacted with is active at a given time. This activity is considered topmost, and is the only activity in the RESUMED state. All other visible activities are STARTED but are not RESUMED. However, the system gives these visible but not resumed activities higher priority than activities that are not visible. If the user interacts with one of the visible activities, that activity is resumed, and the previously topmost activity enters the STARTED state.
- Multi-resume is also available on select devices running Android 9. To opt in to multi-resume on those devices, add the following manifest metadata:

```
<meta-data android:name="android.allow_multiple_resumed_activities"
                                                    android:value="true" />
```

Example:

```
@Override
public void onTopResumedActivityChanged(boolean topResumed)
{
    if (topResumed)
    {
        // Top resumed activity
        // Can be a signal to re-acquire exclusive resources
    } else
    {
        // No longer the top resumed activity
    }
}
```

- This method is invoked when an activity gains or loses the top resumed activity position. This is important to know when an activity uses a shared singleton resource, such as the microphone or camera.
- Set this attribute in your manifest's or element to enable or disable multi-window display:

```
android:resizeableActivity=["true" | "false"]
```

- This attribute is set to true, the activity can be launched in split-screen and freeform modes. If the attribute is set to false, the activity does not support multi-window mode. If this value is false, and the user attempts to launch the activity in multi-window mode, the activity takes over the full screen.
- The following code shows how to specify an activity's default size and location, and its minimum size, when the activity is displayed in freeform mode:
- In Android activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity"
android:resizeableActivity="true">

<LinearLayout
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical">

<ListView
android:id="@+id/list"
android:layout_width="match_parent"
android:layout_height="0dp"
android:layout_weight="1"/>

</LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Summary

- An activity displays the user interface of your application, which may contain widgets such as buttons, labels, text boxes and so on.
- A flexible layout is very important, but you should also design different layouts that optimize the user experience for the available space on different devices such as phones and tablets. So Android allows you to provide alternative layout files that the system applies at runtime based on the current device's screen size.

- The pixel density is the number of pixels within a physical area of the screen and is referred to as dpi (dots per inch).
- Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as button, textfield, checkbox, toggle button, and many more.
- LinearLayouts are one of the simplest and common types of layouts used by Android developer to keep controls within their user interfaces. The LinearLayout works as much as its name implies, it organizes the controls either a vertical or horizontal pattern. When the layout's orientation is set to vertical, all child controls within organized in a single column; and when the layout's orientation is set to horizontal, all child controls within in a single row.
- AbsoluteLayout is based on the simple idea of placing each control at an absolute position. We specify for the exact x and y coordinates on the screen for every control.
- The purpose of the FrameLayout is to allocate an area of screen. Frame layout is used for displaying a single view. If more than child views are added, they will, by default, appear on top of each other, positioned in the top left hand corner of the layout area. Alternate positioning of individual child views can be achieved by setting gravity values on every child. For example, setting a center vertical gravity on a child will cause it to be positioned in the vertical center of the containing FrameLayout view.
- FrameLayout is designed to display a single item at once. We can have multiple elements within a FrameLayout but each element will be positioned based on the top left of the screen. Elements which overlap will be displayed overlapping.
- TableLayout: Arranges child views into a grid format of columns and rows. Each row within a table is represented by a TableRow object child, which contains a view object for each cell. TableLayout organizes content into columns and rows. The rows are defined in the layout XML code, and the columns are determined automatically by Android, which is done by creating at least one column for each element.
- ScrollView is a special type of FrameLayout in that it enables users to scroll through a list of views that occupy more space than the physical display allows.
- RelativeLayout: This is probably the most powerful and flexible of the layout managers, which allows child views to be positioned relative both to each other and the containing layout view through the specification of alignments and margins on child views.
- One of the key features of modern smartphones is their ability to switch screen orientation, and Android is no exception. Android supports two screen orientations namely portrait and landscape.
- onPause() method is always fired whenever an activity is killed or pushed into the background.
- LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.

Practice Questions

Q.I Multiple Choice Questions:

1. A _____ defines the structure for a User Interface (UI) in mobile application, such as in an activity and all elements in the layout are built using a hierarchy of View and ViewGroup objects.
   (a) layout (b) screen
   (c) window (d) None of the Mentioned
2. The _____ arranges views in a single column or a single row. Child views can be arranged either vertically or horizontally.
   (a) FrameLayout (b) AbsoluteLayout
   (c) LinearLayout (d) None of the Mentioned
3. A _____ resource defines the format and look for a UI which can be applied to an individual View (from within a layout file) or to an entire Activity or application (from within the manifest file).
   (a) Theme (b) Style
   (c) Both (a) and (b) (d) None of the Mentioned
4. The TableLayout groups views into _____ and _____ .
   (a) rows (b) columns
   (c) layouts (d) Both (a) and (b)
5. Themethod returns a Display object representing the screen of the device.
   (a) getDefaultDisplay() (b) getDefaultShow()
   (c) Both (a) and (b) (d) None of the Mentioned
6. The _____ is represented as of Android 5.0 via the Toolbar view group, can be freely positioned into mobile application layout file and can display the activity title, icon, actions which can be triggered, additional views and other interactive items. It can also be used for navigation in mobile application.
   (a) titlebar (b) actionbar
   (c) menubar (d) All of the Mentioned
7. A _____ usually draws something the user can see and interact with, where as a _____ is an invisible container that defines the layout structure for View and other ViewGroup objects.
   (a) View (b) ViewGroup
   (c) Both (a) and (b) (d) None of the Mentioned
8. The _____ is a placeholder on screen that you can use to display a single view.
   (a) FrameLayout (b) AbsoluteLayout
   (c) LinearLayout (d) None of the Mentioned

9. Three ways to persist activity state use _____ methods.
   (a) onPause()
   (b) onSaveInstanceState
   (c) onRetainNonConfigurationInstance()
   (d) All of the Mentioned
10. A _____ is nothing but an Android style applied to an entire Activity or application, rather than an individual View.
   (a) Theme (b) Style
   (c) Both (a) and (b) (d) None of the Mentioned
11. Anchoring can be easily achieved by using _____, to preserve the state of an activity, you could always implement the onPause() method, and then use your own ways to preserve the state of your activity, such as using a database, internal or external file storage and so on.
   (a) FrameLayout (b) ScrollLayout
   (c) RelativeLayout (d) None of the Mentioned

Answers

1. (a)	2. (c)	3. (b)	4. (d)	5. (a)	6. (b)	7. (c)	8. (a)	9. (d)	10. (a)
11. (c)									

Q.II Answer the following Questions in short:

1. What is UI?
2. What is layout? Enlist Layout Managers.
3. What is view?
4. What is Action bar?
5. Describe the following LayoutManagers with example:
   (i) LinearLayout
   (ii) TableLaout.

Q.III Answer the following Questions:

1. Explain the term utilizing action bar in detail.
2. What is ScrollView? Explain with example.
3. With the help of diagram describe FrameLayout.
4. Explain the term 'anchoring views' in detail.

Q.IV Short notes:

1. ViewsGroups
2. ConstraintLayout
3. FrameLayout
4. Adapting to display orientation

■■■

4...

Designing Your User Interface with Views

Learning Objectives ...

Students will be able to:

- To Learn UI with views.
- To Study TextView, EditView, RadioButton etc. views with example.
- To Understand ListView, PickerView with Example.
- To Learn Specialized Fragments in Android.

4.1 USINGBASIC VIEWS

- The View is a base class for all UI components in Android. For example, the EditText class is used to accept the input from users in Android apps, which is a subclass of View.
- The ViewGroup will provide an invisible container to hold other Views or ViewGroups and to define the layout properties. Both View and ViewGroup subclasses together will play a key role to create a layout in Android applications.
- In Android operating system the various views that you can use to design the user interface for your applications includes Basic views (commonly used views such as the TextView, EditText, and ButtonViews), Picker views (that enable users to select from a list, such as the Time Picker and DatePicker views), List views (that display a long list of items, such as the ListView and the SpinnerView) and Specialized fragments (that perform specific functions).
- To get started, this section explores some of the basic views that you can use to design the UI of your Android applications:
  1. TextView
  2. Button

3. ImageButton
4. EditText
5. CheckBox
6. Switch
7. ToggleButton
8. RadioButton
9. RadioGroup

- These basic views enable you to display text information, as well as perform some basic selection.
- The following sections explore all these views in more detail.

4.1.1 TextView

- A TextView displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing.
- When you create a new Android project, Android Studio always creates the activity_main.xml file (located in the res/layout folder), which contains a <TextView> element:

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

- You use the TextView view to display text to the user. This is the most basic view and one that you will frequently use when you develop Android applications. If you need to allow users to edit that the use this control.

4.1.2 Button, ImageButton, EditText, CheckBox, Switch, ToggleButton, RadioButton and RadioGroup Views

- Besides the TextView view, which you will likely use the most often, there are some other basic views that are frequently using:
  1. **Button:** Represents a push-button widget.
  2. **ImageButton:** Similar to the Buttonview, except that it also displays an image.
  3. **EditText:** A subclass of the TextView view, which allows users to edit its text content.
  4. **CheckBox:** A special type of button that has two states: checked or unchecked.

5. **ToggleButton:** Displays checked/unchecked states using a light indicator.
6. **Switch:** It is a two-state user interface element that is used to display **ON (Checked)** or **OFF (Unchecked)** states as a button. The Switch element is useful for the users to change the settings between two states either ON or OFF.
7. **RadioGroup and RadioButton:** The RadioButton has two states: either checked or unchecked. A RadioGroup is used to group one or more RadioButton views, thereby allowing only one RadioButton to be checked within the RadioGroup.

Example using the Basic Views:

Step 1 : Using Android Studio, create an Android project and name it **BasicViews1**.

Step 2 : Modify the activity_main.xml file located in the res/layout folder by adding the following elements shown in bold:

```
<?xml version="1.0"encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">
<Button android:id="@+id/btnSave"
android:layout_width="fill_parent"
android:layout_height="wrap_content"android:text="save"/>
<Button android:id="@+id/btnOpen"
android:layout_width="wrap_content"
android:layout_height="wrap_content"android:text="Open"/>
<ImageButton android:id="@+id/btnImg1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"android:src="@mipmap/ic_laun
cher"/>
<EditText android:id="@+id/txtName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"/>
<CheckBox android:id="@+id/chkAutosave"
android:layout_width="fill_parent"
android:layout_height="wrap_content"android:text="Autosave" />
<CheckBox android:id="@+id/star"
style="?android:attr/starStyle"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
<RadioGroup android:id="@+id/rdbGp1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"android:orientation="vertica
l">
```

```
<RadioButton android:id="@+id/rdb1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"android:text="Option 1" />
<RadioButton android:id="@+id/rdb2"
android:layout_width="fill_parent"
android:layout_height="wrap_content"android:text="Option 2" />
</RadioGroup>
<ToggleButton android:id="@+id/toggle1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
</LinearLayout>
```

Step 3 : To see the views in action, debug the project in Android Studio by pressing Shift+F9.

Fig. 4.1 (a) shows the various views displayed in the Android emulator.

Step 4 : Click each of the views and note how they vary in look and feel. Fig. 4.1 (b) shows the following changes to the view:

(i) The first CheckBox view (Autosave) is checked.

(ii) The second CheckBox View (star) is selected.

(iii) The second RadioButton (Option2) is selected.

(iv) The ToggleButton is turned on.

(a) (b)

Fig. 4.1: Views in the Android Emulator

Handling View Events:

Step1 : Using the BasicViews1 project you created in the previous Try It Out, modify the MainActivity.java file by adding the following bolded statements:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;
import android.widget.ToggleButton;
public class MainActivity extends AppCompatActivity {
@Override
Protected void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
//---Button view---
Button btnOpen = (Button) findViewById(R.id.btnOpen);
btnOpen.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
DisplayToast("You have clicked the Open button");
}
});
//---Button view---
Button btnSave = (Button) findViewById(R.id.btnSave);
btnSave.setOnClickListener(new View.OnClickListener(){
public void onClick(View v){
DisplayToast("You have clicked the Save button");
}
});
//---CheckBox---
CheckBox checkBox = (CheckBox)
findViewById(R.id.chkAutosave);
checkBox.setOnClickListener(new View.OnClickListener(){
public void onClick(View v){
if (((CheckBox)v).isChecked())
DisplayToast("CheckBox is checked");
else
DisplayToast("CheckBox is unchecked");
}
});
```

```
//---RadioButton---
RadioGroup radioGroup = (RadioGroup) findViewById(R.id.rdbGp1);
radioGroup.setOnCheckedChangeListener(
new RadioGroup.OnCheckedChangeListener()
{
public void onCheckedChanged(RadioGroup group, int checkedId) {
RadioButton rb1 = (RadioButton) findViewById(R.id.rdb1);
if (rb1.isChecked()){
DisplayToast("Option 1 checked!");
} else {
DisplayToast("Option 2 checked!");
}
}
});
//---ToggleButton---

ToggleButton toggleButton =
(ToggleButton) findViewById(R.id.toggle1);
toggleButton.setOnClickListener(newView.OnClickListener()
{
public void onClick(View v){
if (((ToggleButton)v).isChecked())
DisplayToast("Toggle button is On");
else
DisplayToast("Toggle button is Off");
}
});
}
private void DisplayToast(String msg)
{
Toast.makeText(getBaseContext(), msg,
Toast.LENGTH_SHORT).show();
}
}
```

Step 2 : Press Shift+F9 to debug the project on the Android emulator.

Step 3 : Click each of the views and observe the message displayed in the Toast window.

4.1.3 ProgressBar View

- The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.

- For example, you might be downloading some data from the web and need to update the user about the status of the download. In this case, the ProgressBar view is a good choice.
- The following activity demonstrates how to use the ProgressBar view.

Example: Using the ProgressBar View.

Step 1 : Using Android Studio, create an Android project and name it BasicViews2.

Step 2 : Modify the activity_main.xml file located in the res/layout folder by adding the following code in bold:

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

Step 3 : In the MainActivity.java file, add the following bolded statements:

```
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ProgressBar;
public class MainActivity extends AppCompatActivity {
static int progress;
ProgressBar progressBar;
int progressStatus = 0;
Handler handler = new Handler();
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
progress = 0;
progressBar = (ProgressBar) findViewById(R.id.progressbar);
//---do some work in background thread--- new
Thread(newRunnable()
{
public void run()
{
```

```
//---do some work here---while (progressStatus < 10)
{
progressStatus = doSomeWork();
}
//---hides the progress bar---handler.post(new Runnable()
{
public void run()
{

//---0 - VISIBLE; 4 - INVISIBLE; 8 - GONE---
progressBar.setVisibility(View.GONE);
}
});
}
//---do some long running work here---
private int doSomeWork()
{
try {
//---simulate doing some work---Thread.sleep(500);
} catch (InterruptedException e)
{
e.printStackTrace();
}
return ++progress;
}
}).start();
}
}
```

Step 4 : Press Shift+F9 to debug the project on the Android emulator. Fig. 4.2 shows the ProgressBar animating. After about five seconds, it disappears.

Fig. 4.2: ProgressBar View

Customizing the ProgressBar View:

Step 1 : Using the BasicViews2 project created in the previous Try It Out, modify the activity_main.xml file as shown here:

```
<?xml version="1.0"encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">
<ProgressBar android:id="@+id/progressbar"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
style="@android:style/Widget.ProgressBar.Horizontal"/>
</LinearLayout>
```

Step 2 : Modify the MainActivity.java file by adding the following bolded statements:

```
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ProgressBar;
public class MainActivity extends AppCompatActivity { static int
progress;
ProgressBar progressBar;
int progressStatus= 0;
Handler handler = new Handler();
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
progress = 0;
progressBar = (ProgressBar) findViewById(R.id.progressbar);
progressBar.setMax(200);
//---do some work in background thread---new Thread(new
Runnable()
{
public void run()
{
//---do some work here---
```

```
while (progressStatus < 100)
{
progressStatus = doSomeWork();
//---Update the progress bar---handler.post(new Runnable()
{
public void run(){
progressBar.setProgress(progressStatus);
}
});
}
//---hides the progress bar---handler.post(new Runnable()
{
public void run()
{
}
});
}
//---0 - VISIBLE; 4 - INVISIBLE; 8 - GONE---
progressBar.setVisibility(View.GONE);
//---do some long running work here---private int doSomeWork()
{
try {
//---simulate doing some work---Thread.sleep(500);
} catch (InterruptedException e)
{
e.printStackTrace();
}
return ++progress;
}
}).start();
}
}
```

Step 3 : Press Shift+F9 to debug the project on the Android emulator.

Step 4 : Fig. 4.3 shows the ProgressBar displaying the progress. The ProgressBar disappears when the progress reaches 50 percent.

Fig. 4.3

4.1.4 AutoCompleteTextView View

- The AutoCompleteTextView is a view that is similar to EditText (in fact it is a subclass of EditText), except that it automatically shows a list of completion suggestions while the user is typing.

  Example: Using the AutoCompleteTextView.

  Step 1 : Using Android Studio, create an Android project and name it BasicViews3.

  Step 2 : Modify the activity_main.xml file located in the res/layout folder as shown here in bold:

```
<?xml version="1.0"encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"android:text="NameofPresiden
t" />
<AutoCompleteTextView
android:id="@+id/txtCountries"
android:layout_width="fill_parent"
android:layout_height="wrap_content"/>
</LinearLayout>
```

Step 3 : Add the following bolded statements to the MainActivity.java file:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity{
String[] presidents = {
"Dwight D. Eisenhower",
"John F. Kennedy",
"Lyndon B. Johnson",
"Richard Nixon",
"Gerald Ford",
"Jimmy Carter",
"Ronald Reagan",
"George H. W. Bush",
"Bill Clinton",
"George W. Bush",
"Barack Obama"
};
/** Called when the activity is first created. */ @Override
public void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_dropdown_item_1line, presidents);
AutoCompleteTextView textView = (AutoCompleteTextView)
findViewById(R.id.txtCountries);
textView.setThreshold(3);
textView.setAdapter(adapter);
}
}
```

Step 4 : Press Shift+F9 to debug the application on the Android emulator. As shown in Fig. 4.4, a list of matching names appears as you type into the AutoCompleteTextView field.

Fig. 4.4: AutoCompleteTextView

4.2 USING PICKER VIEWS

- Selecting a date and time is one of the common tasks you need to perform in a mobile application. Android supports this functionality through the TimePicker and DatePicker views.

4.2.1 TimePicker View

- The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode. The following example shows you how to use the TimePicker in the latest version of the Android SDK. When you are creating the project for this sample, be sure that you choose an SDK that is level 23 or greater.

Example: Using the TimePicker View.

Step 1 : Using Android Studio, create an Android project and name it BasicViews4.

Step 2 : Modify the activity_main.xml file located in the res/layout folder by adding the following bolded lines:

```
<?xml version="1.0"encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">
<TimePicker android:id="@+id/timePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
<Button android:id="@+id/btnSet"
android:layout_width="wrap_content"
android:layout_height="wrap_content"android:text="I am all set!"
android:onClick="onClick" />
</LinearLayout>
```

Step 3 : Press Shift+F9 to debug the application on the Android emulator. Fig. 4.5 shows the TimePicker in action. You can use the numeric keypad or the time widget on the screen to change the hour and minute.

Fig. 4.5: TimePicker View

Step 4 : Back in Android Studio, add the following bolded statements to the MainActivity.java file:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TimePicker;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity{
TimePicker timePicker;
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
timePicker = (TimePicker) findViewById(R.id.timePicker);
timePicker.setIs24HourView(true);
}
public void onClick(View view) {
Toast.makeText(getBaseContext(),
"Time selected:" + timePicker.getHour() +
":" + timePicker.getMinute(),
Toast.LENGTH_SHORT).show();
}
}
```

Step 5 : Press Shift+F9 to debug the application on the Android emulator. This time, the TimePicker is displayed in the 24-hour format. Clicking the Button displays the time that you have set in the TimePicker (See Fig. 4.6).

Fig. 4.6: TimePicker View in 24 hour Format

Example: Using a Dialog to Display the TimePicker View:

Step 1 : Using the BasicViews4 project created in the previous Try It Out, modify the MainActivity.java file as shown here:

```
import android.app.TimePickerDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TimePicker;
import java.util.Calendar;
public class MainActivity extends AppCompatActivity {
TimePickerDialog.OnTimeSetListener dialogListener =
newTimePickerDialog
.OnTimeSetListener() {
@Override
public void onTimeSet(TimePicker timePicker, int i, int i1){
}
};
Calendar cal = Calendar.getInstance();
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
showTimeDialog();
}
public void showTimeDialog(){
new
TimePickerDialog(MainActivity.this,dialogListener,cal.get(Calend
ar.HOUR_OF_DAY), cal.get(Calendar.MINUTE), false).show();
{}; }
}
```

Step 2 : Modify the activity_main.xml to look as follows:

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

Step 3 : Press Shift+F9 to debug the application on the Android emulator. When the activity is loaded, you can see the TimePicker displayed in a dialog window (See Fig. 4.7).

Fig. 4.7: TimePicker View in Dialog Window

4.2.2 DatePicker View

- Another view that is similar to the TimePicker is the DatePicker. Using the DatePicker, you can enable users to select a particular date on the activity.
- The following example shows you how to use the DatePicker.

Example: Using the DatePicker View.

Step 1 : Using the BasicViews4 project created earlier, modify the activity_main.xml file as shown here:

```
<?xml version="1.0"encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">
<Button android:id="@+id/btnSet"
android:layout_width="wrap_content"
android:layout_height="wrap_content"android:text="I am all set!"
android:onClick="onClick" />
<DatePicker android:id="@+id/datePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>

<TimePicker android:id="@+id/timePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
</LinearLayout>
```

Step 2 : Add the following bolded statements to the MainActivity.java file:

```
import android.app.TimePickerDialog;
import android.icu.text.SimpleDateFormat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.DatePicker;
import android.widget.TimePicker;
import android.widget.Toast;
import java.util.Date;
public class MainActivity extends AppCompatActivity{
TimePicker timePicker;
DatePicker datePicker;
int hour, minute;
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
timePicker = (TimePicker) findViewById(R.id.timePicker);
timePicker.setIs24HourView(true);
datePicker = (DatePicker) findViewById(R.id.datePicker);
}
private  TimePickerDialog.OnTimeSetListener  mTimeSetListener  =
new TimePickerDialog.OnTimeSetListener()
{
public void onTimeSet(
TimePicker view, int hourOfDay, intminuteOfHour)
{
hour  = hourOfDay;
minute = minuteOfHour;
SimpleDateFormat  timeFormat  =  newSimpleDateFormat("hh:mm  aa");
Date date = new Date();
String strDate = timeFormat.format(date);

Toast.makeText(getBaseContext(),
"You have selected " + strDate,
```

```
Toast.LENGTH_SHORT).show();
}
};
public void onClick(View view) {
Toast.makeText(getBaseContext(),
"Date selected:" +(datePicker.getMonth()+1)
+ "/" + datePicker.getDayOfMonth() +"/" + datePicker.getYear() +
"\n" +
"Time    selected:"    +    timePicker.getHour()    +    ":"    +
timePicker.getMinute(),
Toast.LENGTH_SHORT).show();
}
}
```

Step 3 : Press Shift+F9 to debug the application on the Android emulator. After the date is set, clicking the Button displays the date set (See Fig. 4.8).

Fig. 4.8: DatePicker View

4.3 USING LIST VIEWS TO DISPLAY LONG LISTS

- List views are views that enable you to display a long list of items. In Android, there are two types of list views: ListView and SpinnerView. Both are useful for displaying long lists of items.

4.3.1 ListView View

- The ListView displays a list of items in a vertically scrolling list. The following code demonstrates how to display a list of items using the ListView.

Example: Displaying a LongList of Items Using the ListView.

Step 1 : Using Android Studio, create an Android project and name it BasicViews5.

Step 2 : Modify the MainActivity.java file by inserting the bolded statements shown here:

```
import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;
public class MainActivity extends ListActivity{
String[] presidents = {
"Dwight D. Eisenhower",
"John F. Kennedy",
"Lyndon B. Johnson",
"RichardNixon",
"Gerald Ford",
"Jimmy Carter",
"Ronald Reagan",
"George H. W. Bush",
"Bill Clinton",
"George W. Bush",
"Barack Obama"
};
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
setListAdapter(newArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, presidents));
}
public void onListItemClick(
ListView parent, Viewv, int position, long id)
{
Toast.makeText(this,
"You have selected" + presidents[position],
Toast.LENGTH_SHORT).show();
}
}
```

Step 3 : Press Shift+F9 to debug the application on the Android emulator. Fig. 4.9 shows the activity displaying the list of presidents' names.

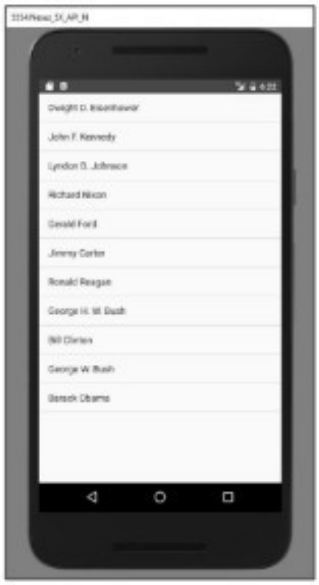


Fig. 4.9: ListView

Step 4 : Click an item. A message containing the item selected is displayed.

Customizing the ListView

- The ListView is a versatile view that you can further customize.

  Example: Enabling Filtering and Multi-Item Support in the ListView.

  Step 1 : Using the BasicViews5 project created in the previous section, add the following bolded statements to the MainActivity.java file:

```
public void onCreate(Bundle savedInstanceState)
{super.onCreate(savedInstanceState);
ListView lstView = getListView();
lstView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
lstView.setTextFilterEnabled(true);
setListAdapter(newArrayAdapter<String>(this,
android.R.layout.simple_list_item_checked, presidents));

}
```

  Step 2 : Press Shift+F9 to debug the application on the Android emulator. You can now click each item to display the check icon next to it (see Fig. 4.10).

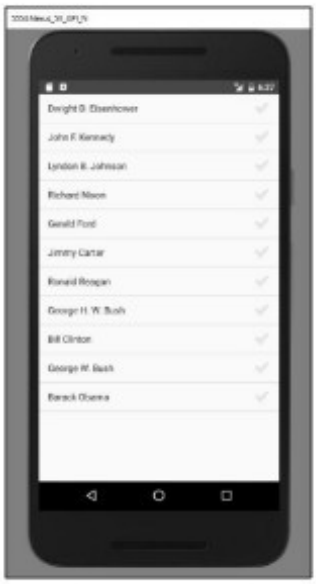


Fig. 4.10: Multi-item support in the ListView

4.3.2 Using the Spinner View

- The ListView displays a long list of items in an activity, but you might want the user interface to display other views, i.e. you do not have the additional space for a full-screen view, such as the ListView.
- In such cases, you should use the SpinnerView. The SpinnerView displays one item at a time from a list and enables users to choose from them.

Example: Using the SpinnerView to Display One Item at a Time.

Step 1 : Using Android Studio, create an Android project and name it BasicViews6.

Step 2 : Modify the activity_main.xml file located in the res/layout folder as shown here:

```
<?xml version="1.0"encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">
<Spinner
android:id="@+id/spinner1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"android:drawSelectorOnTop="t
rue"/>
</LinearLayout>
```

Step 3 : Add the following bolded lines to the strings.xml file located in the res/values folder:

```
<resources>
<string name="hello">Hello World, BasicViews6Activity!</string>
<string name="app_name">BasicViews6</string>
<string-array name="presidents_array">
<item>Dwight D. Eisenhower</item>
<item>John F. Kennedy</item>
<item>Lyndon B. Johnson</item>
<item>Richard Nixon</item>
<item>Gerald Ford</item>
<item>Jimmy Carter</item>
<item>Ronald Reagan</item>
<item>George H.W.Bush</item>
<item>Bill Clinton</item>
<item>George W. Bush</item>
<item>Barack Obama</item>
</string-array>
</resources>
```

Step 4 : Add the following bolded statements to the MainActivity.java file:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity{
String[]presidents;
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
presidents=
getResources().getStringArray(R.array.presidents_array);

Spinner s1 = (Spinner) findViewById(R.id.spinner1);
ArrayAdapter<String> adapter = newArrayAdapter<String>(this,
android.R.layout.simple_spinner_item,                presidents);
s1.setAdapter(adapter);
s1.setOnItemSelectedListener(newAdapterView.OnItemSelectedListen
er()
```

```
{
@Override
public void onItemSelected(AdapterView<?> arg0,
View arg1, int arg2, long arg3)
{
int index = arg0.getSelectedItemPosition();
Toast.makeText(getBaseContext(),
"You have selected item : " + presidents[index],
Toast.LENGTH_SHORT).show();
}
@Override
public void onNothingSelected(AdapterView<?> arg0){}
});
}
}
```

Step 5 : Press Shift+F9 to debug the application on the Android emulator. Click SpinnerView and you see a pop-up displaying the list of presidents' names (See Fig. 4.11). Clicking an item displays a message showing the selected item.

(a) (b)

Fig. 4.11: SpinnerView

4.4 UNDERSTANDING SPECIALIZED FRAGMENTS

- As you have learned, fragments are really "mini-activities" that have their own lifecycles. To create a fragment, you need a class that extends the Fragment base class.

- In addition to the Fragment base class, you can also extend from some other subclasses of the Fragment base class to create more specialized fragments.
- The following sections discuss the three subclasses of Fragment:
  1. ListFragment
  2. DialogFragment.
  3. PreferenceFragment.

4.4.1 Using a ListFragment

- A list fragment is a fragment that contains a ListView, which displays a list of items from a data source, such as an array or a cursor. A list fragment is useful because it's common to have one fragment that contains a list of items (such as a list of RSS postings), and another fragment that displays details about the selected posting. To create a ListFragment, you need to extend the ListFragment base class.

Example: Creating and Using a List Fragment.

Step 1 : Using Android Studio, create an Android project and name it ListFragmentExample.

Step 2 : Modify the activity_main.xml file as shown in bold. (Please replace instances of com.jfdimarzio with references your project's package:

```
<?xml version="1.0"encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="horizontal" >
<fragment
android:name="com.jfdimarzio.listfragmentexample.Fragment1"
android:id="@+id/fragment1"
android:layout_weight="0.5"
android:layout_width="0dp"
android:layout_height="200dp"/>
<fragment
android:name="com.jfdimarzio.listfragmentexample.Fragment1"
android:id="@+id/fragment2"
android:layout_weight="0.5"
android:layout_width="0dp"
android:layout_height="300dp"/>
</LinearLayout>
```

Step 3 : Add an XML file to the res/layout folder and name it fragment1.xml.

Step 4 : Populate the fragment1.xml as follows:

```
<?xml version="1.0"encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<ListView
android:id="@id/android:list"
android:layout_width="match_parent"
android:layout_height="match_parent"android:layout_weight="1"
android:drawSelectorOnTop="false"/>
</LinearLayout>
```

Step 5 : Add a Java Class file to the package and name it Fragment1.

Step 6 : Populate the Fragment1.java file as follows:

```
import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;
public class Fragment1 extends ListFragment {String[]
presidents={
"Dwight D. Eisenhower",
"John F. Kennedy",
"Lyndon B. Johnson",
"Richard Nixon",
"Gerald Ford",
"Jimmy Carter",
"Ronald Reagan",
"George H. W. Bush",
"Bill Clinton",
"George W. Bush",
"Barack Obama"
};
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState){
return inflater.inflate(R.layout.fragment1, container, false);
}
```

```
@Override
public void onCreate(Bundle savedInstanceState)
{super.onCreate(savedInstanceState); setListAdapter(new
ArrayAdapter<String>(getActivity(),
android.R.layout.simple_list_item_1, presidents));
}
public void onListItemClick(ListView parent, View v, int
position, long id)
{
Toast.makeText(getActivity(),
"You have selected " + presidents[position],
Toast.LENGTH_SHORT).show();
}
}
```

Step 7 : Press Shift+F9 to debug the application on the Android emulator. Fig. 4.12 (a) shows the two ListFragments displaying the two lists of presidents' names.

Step 8 : Click any of the items in the two ListView views, and you see a message (See Fig. 4.12 (b)).

(a) (b)

Fig. 4.12: ListFragments

4.4.2 Using a DialogFragment

- A dialog fragment floats on top of an activity and is displayed modally. DialogFragments are useful when you need to obtain the user's response before continuing with execution. To create a DialogFragment, you must extend the DialogFragment base class.

Example: Creating and Using a DialogFragment.

Step 1 : Using Android Studio, create an Android project and name it DialogFragmentExample.

Step 2 : Add a Java Class file under the package and name it Fragment1.

Step 3 : Populate the Fragment1.java file as follows:

```
import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;
public class Fragment1 extends DialogFragment {
static Fragment1 newInstance(String title){
Fragment1 fragment = new Fragment1();
Bundle args = newBundle();
args.putString("title", title);
fragment.setArguments(args);
return fragment;
}
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
String title = getArguments().getString("title");
return new AlertDialog.Builder(getActivity())
.setIcon(R.mipmap.ic_launcher)
.setTitle(title)
.setPositiveButton("OK",
new DialogInterface.OnClickListener(){
public void onClick(DialogInterface dialog,int whichButton){
((MainActivity)getActivity()).doPositiveClick();
}
});
.setNegativeButton("Cancel",
new DialogInterface.OnClickListener(){
public void onClick(DialogInterface dialog,int whichButton){
((MainActivity) getActivity()).doNegativeClick();
}
}).create();
}
}
```

Step 4 : Populate the MainActivity.java file as shown here in bold:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.util.Log;
public class MainActivity extends AppCompatActivity{
@Override
protected void onCreate(Bundle savedInstanceState)
{super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Fragment1 dialogFragment = Fragment1.newInstance( "Are you sure
you want to do this?");
dialogFragment.show(getFragmentManager(),"dialog");
}
public void doPositiveClick(){
//---perform steps when user clicks on OK---
Log.d("DialogFragmentExample", "User clicks on OK");
}
public void doNegativeClick(){
//---perform steps when user clicks on Cancel---
Log.d("DialogFragmentExample", "User clicks on Cancel");
}
}
```

Step 5 : Press Shift+F9 to debug the application on the Android emulator. Fig. 4.13 shows the fragment displayed as an alert dialog. Click either OK or Cancel and observe the message displayed.

Fig. 4.13

4.4.3 Using a PreferenceFragment

- Typically, in your Android applications you provide preferences for users to personalize the application. For example, you might allow users to save the login

credentials that they use to access their web resources. Also, you could save information, such as how often the feeds must be refreshed (For example, in an RSS reader application), and so on.

- In Android, you can use the PreferenceActivity base class to display an activity for the user to edit the preferences. In Android 3.0 and later, you can use the PreferenceFragment class to do the same thing.

  Example: Creating and Using a PreferenceFragment.

  Step 1 : Using Android Studio, create an Android project and name it **PreferenceFragmentExample**.

  Step 2 : Create a new xml directory under the res folder and then add a new XML resource file to it.

  Name the XML file preferences.xml.

  Step 3 : Populate the preferences.xml file as follows:

```
<?xml version="1.0"encoding="utf-8"?>
<PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android">
<PreferenceCategory android:title="Category1">
<CheckBoxPreference android:title="Checkbox"
android:defaultValue="false"
android:summary="True of False"
android:key="checkboxPref"/>
</PreferenceCategory>
<PreferenceCategory          android:title="Category          2">
<EditTextPreference android:name="EditText"
android:summary="Enter a string"
android:defaultValue="[Enter a string here]"
android:title="Edit Text"
android:key="editTextPref" />
<RingtonePreference
android:name="Ringtone Preference"
android:summary="Select a ringtone"
android:title="Ringtones"
android:key="ringtonePref" />
<PreferenceScreen
android:title="Second Preference Screen"
android:summary=
"Click here   to     go     to     the    second Preference
   Screen"   android:key="secondPrefScreenPref">
<EditTextPreference
android:name="EditText"
android:summary="Enter a string"
```

```
android:title="Edit Text (second Screen)"
android:key="secondEditTextPref" />
</PreferenceScreen>
</PreferenceCategory>
</PreferenceScreen>
```

Step 4 : Add a Java Class file to the package and name it **Fragment1**.

Step 5 : Populate the Fragment1.java file as follows:

```
import android.os.Bundle;
import android.preference.PreferenceFragment;
public class Fragment1 extends
PreferenceFragment {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
//---load the preferences from an XML file---
addPreferencesFromResource(R.xml.preferences);
}
}
```

Step 6 : Modify the MainActivity.java file as shown in bold:

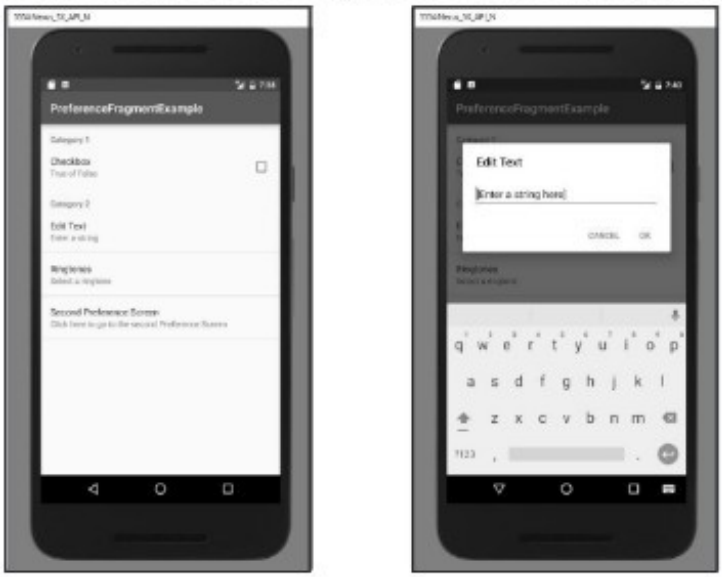
```
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity { @Override
protected void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();
Fragment1 fragment1 = new Fragment1();
fragmentTransaction.replace(android.R.id.content, fragment1);
fragmentTransaction.addToBackStack(null);fragmentTransaction.com
mit();
}
}
```

Step 7 : Press Shift+F9 to debug the application on the Android emulator. Following Fig. shows the PreferenceFragment displaying the list of preferences that the user can modify.

Step 8 : When the Edit Text preference is clicked, a pop-up is displayed (See Fig. 4.14 (a)).

Step 9 : Clicking Edit Text (Second Screen) causes a second preference screen to be displayed (See Fig. 4.14 (b)).

Step 10 : To dismiss the preference fragment, click the Back button on the emulator.



(a) (b)

Fig. 4.14: PreferenceFragment

4.5 DISPLAYING PICTURES AND MENUS

4.5.1 Using Image Views to Display Pictures

- So far, all the views you have seen are used to display text information. However, you can use the ImageView, ImageSwitcher, and GridView views for displaying images. The following sections discuss each view in detail.

4.5.1.1 Gallery and ImageView views

GalleryView

- Android Gallery is a View commonly used to display items in a horizontally scrolling list that locks the current selection at the center. In this section we'll display a horizontal list of images and when a user clicks an image, it will be displayed in the center of the screen.
- The items of Gallery are populated from an Adapter, similar to ListView, in which ListView items were populated from an Adapter.
- We need to create an Adapter class which extends BaseAdapter class and override getView() method.
- getView() method called automatically for all items of Gallery.
- The layout for the Gallery is defined as follows:

```
<Gallery android:id="@+id/gallery1"
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
```

- It belongs to android.widget.Gallery class. However this class is deprecated now.

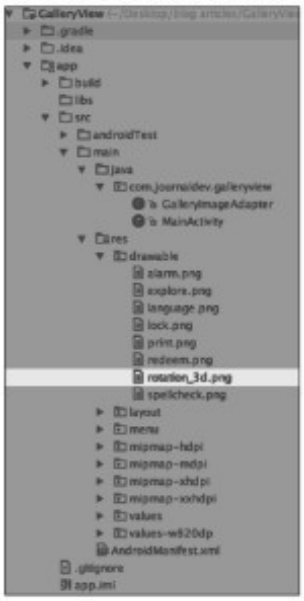


Fig. 4.15: Project Structure for GalleryView

- The layout of the MainActivity is given below:

```
main.xml
<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"
android:orientation="vertical">
<Gallery android:id="@+id/gallery"
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
<ImageView android:id="@+id/imageView"
android:layout_marginTop="100dp"
android:layout_width="250dp"
android:layout_gravity="center_horizontal"
android:layout_height="250dp"
android:src="@drawable/alarm" />
</LinearLayout>
```

- The android:src points to the first image from the left in the gallery.
- The MainActivity.java is given below:

```
package com.journaldev.galleryview;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.AdapterView;
import android.widget.Gallery;
import android.widget.ImageView;
public class MainActivity extends AppCompatActivity {
ImageView selectedImage;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
Gallery gallery = (Gallery) findViewById(R.id.gallery);
selectedImage=(ImageView)findViewById(R.id.imageView);
gallery.setSpacing(1);
final GalleryImageAdapter galleryImageAdapter = new
GalleryImageAdapter(this);
gallery.setAdapter(galleryImageAdapter);
gallery.setOnItemClickListener(new AdapterView.OnItemClickListener()
{
public void onItemClick(AdapterView<?> parent, View v, int position,
long id)
{ // show the selected Image
selectedImage.setImageResource(galleryImageAdapter.mImageIds[position])
; } }); } }
```

- We need to create the GalleryImageAdapter class which extends the BaseAdapter class. This will bind to the Gallery view with a series of ImageView views. The BaseAdapter class will work as a bridge between an AdapterView and also the data source that feeds data into it.
- For the GalleryImageAdapter class, following methods are implemented:
  - getCount()
  - getItem()
  - getItemId()
  - getView()
- The GalleryImageAdapter class is given below:

```
package com.journaldev.galleryview;
import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.Gallery;
import android.widget.ImageView;
public class GalleryImageAdapter extends BaseAdapter {
private Context mContext;
```

```
public GalleryImageAdapter(Context context) { mContext = context; }
public int getCount() { return mImageIds.length; }
public Object getItem(int position) { return position; }
public long getItemId(int position) { return position; }
// Override this method according to your need
public View getView(int index, View view, ViewGroup viewGroup) {
//   TODO   Auto-generated   method   stub   ImageView   i   =   new
ImageView(mContext);
i.setImageResource(mImageIds[index]);
i.setLayoutParams(new Gallery.LayoutParams(200, 200));
i.setScaleType(ImageView.ScaleType.FIT_XY); return i;
}
public Integer[] mImageIds = { R.drawable.alarm, R.drawable.explore,
R.drawable.language,          R.drawable.lock,          R.drawable.print,
R.drawable.rotation_3d,
R.drawable.spellcheck, R.drawable.redeem }; }
```

Output:

Fig. 4.16: GalleryView

ImageView:

- Android ImageView is used to display an image file. In android ImageView is also commonly used to display image such as Bitmap or vector.
- The following example shows you how to use the ImageView view to display an image. **Example:** Using the ImageView.

  Step 1 : Using Android Studio, create a new Android project and name it **Image**.

  Step 2 : Add an image to your project under the res/mipmap folder as shown in Fig. 4.17.

(The imageI used in this example is butterfly.png from http://jfdimarzio.com/butterfly.png.) Please note that you must be in project view to drag and drop images into the res/mipmap folder.

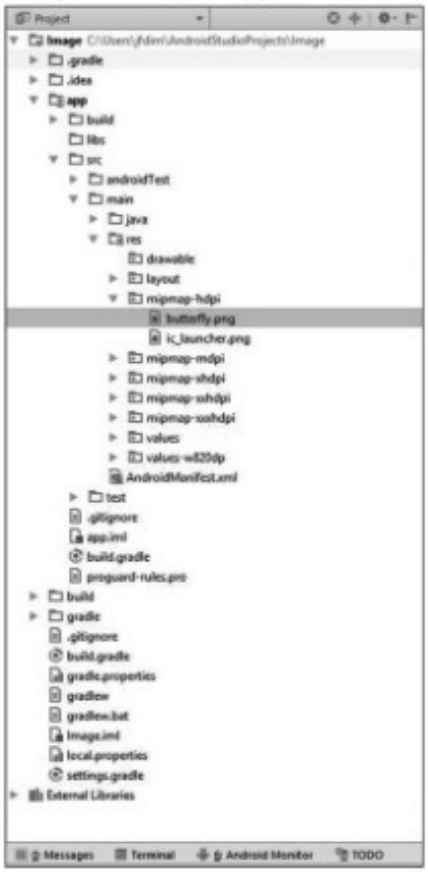


Fig. 4.17

Step 3 : Modify the activity_main.xml file as shown in bold:

```
<?xml version="1.0"encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_main"
android:layout_width="wrap_content"
android:layout_height="match_parent"
tools:context="com.jfdimarzio.image.MainActivity">
<android.support.v7.widget.AppCompatImageViewandroid:layout_widt
h="match_parent"
android:layout_height="match_parent"
android:src="@mipmap/butterfly" />
</LinearLayout>
```

Step 4 : Press Shift+F9 to debug the application on the Android emulator. Fig. 4.18 shows the image displayed in the ImageView.

Fig. 4.18: Image in ImageView

4.5.1.2 ImageSwitcher

- The previous section demonstrated how to use the ImageView to display an image. However, sometimes you don't want an image to appear abruptly when the user opens the view.
- For example, you might want to apply some animation to an image when it transitions from one image to another. In this case, you need to use the ImageSwitcher.

Example: Using the ImageSwitcher View:

Step 1 : Using Android Studio, create a new Android project and name it ImageSwitcher.

Step 2 : Modify the activity_main.xml file by adding the following bolded statements. Please be sure to change all instances of com.jfdimarzio to reflect the package you use in your project:

```
<?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.jfdimarzio.imageswitcher.MainActivity">
<Button
android:text="View Windows"
android:layout_width="wrap_content"
android:layout_height="wrap_content"android:id="@+id/button2"
/>
<ImageSwitcher
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_alignParentStart="true"
android:layout_below="@+id/button2"
android:id="@+id/imageSwitcher">
</ImageSwitcher>
<Button
android:text="View Butterfly"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/button"
android:layout_alignParentTop="true"
android:layout_alignParentEnd="true"
/>
</RelativeLayout>
```

Step 3 : Add two images to your res/mipmap folder. For this example, we added an image named butterfly.png and an image named windows.jpg.

Step 4 : Add the following bolded statements to the MainActivity.javafile:

```
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.ImageSwitcher;
import android.widget.ImageView;
import android.widget.Toast;
import android.widget.ViewSwitcher;
public class MainActivity extends AppCompatActivity
{ private ImageSwitcher imgSwitcher;
private Button btnViewWindows,btnViewButterfly;
@Override
protected void onCreate(Bundle saved InstanceState)
```

```
{ super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
imgSwitcher = (ImageSwitcher) findViewById(R.id.imageSwitcher);
imgSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
android.R.anim.fade_in));
imgSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
android.R.anim.fade_out));
btnViewWindows = (Button) findViewById(R.id.button2);
btnViewButterfly = (Button) findViewById(R.id.button);
imgSwitcher.setFactory(new ViewSwitcher.ViewFactory()
{ @Override
public View makeView(){
ImageView myView = new ImageView(getApplicationContext());
myView.setScaleType(ImageView.ScaleType.FIT_CENTER);
myView.setLayoutParams(new ImageSwitcher.LayoutParams(
ActionBar.LayoutParams.WRAP_CONTENT,
ActionBar.LayoutParams.WRAP_CONTENT)); return myView;
      }
});
btnViewWindows.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v){
Toast.makeText(getApplicationContext()
"View Windows",Toast.LENGTH_LONG).show();
imgSwitcher.setImageResource(R.mipmap.windows);
}
});
btnViewButterfly.setOnClickListener(new View.OnClickListener()
{ @Override
public void onClick(Viewv){
Toast.makeText(getApplicationContext(), "ViewButterfly",
Toast.LENGTH_LONG).show();imgSwitcher.setImageResource(R.mipmap.
butterfly);
}
});

}
}
```

Step 5 : Press Shift+F9 to debug the application on the Android emulator. Fig. 4.19 shows the ImageSwitcher and Button views, with no images loaded.

Fig. 4.19

Step 6 : Click the View Windows button. You see the windows image and a Toast indicating that the image is being viewed, as shown in Fig. 4.20.

Fig. 4.20

Step 7 : Click the View Butterfly button. You see a fade-out and a fade-in animation from the windows image to the butterfly image. You also see a Toast indicating that the image is being viewed (as shown in Fig. 4.20).

How it Works?

- To use the ImageSwitcher view, you need to implement the ViewFactory interface, which creates the views for use with the ImageSwitcher view. For this, you need to implement the makeView()method:

```
imgSwitcher.setFactory(new ViewSwitcher.ViewFactory()
{ @Override
public View makeView(){
ImageView myView = new ImageView(getApplicationContext());
myView.setScaleType(ImageView.ScaleType.FIT_CENTER);
myView.setLayoutParams(newImageSwitcher.
LayoutParams(ActionBar.LayoutParams.WRAP_CONTENT,ActionBar.LayoutParams.WR
AP_CONTENT)); return myView;
}
});
```

Fig. 4.21

- This method creates a new View to be added in the ImageSwitcher view, which in this case is an ImageView.
- In the onCreate() method, you get a reference to the ImageSwitcher view and set the animation, specifying how images should "fade" in and out of the view.
- Finally, when an image is selected from the Gallery view, the image is displayed in the ImageSwitcher view:

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
imgSwitcher = (ImageSwitcher) findViewById(R.id.imageSwitcher);
imgSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
android.R.anim.fade_in));imgSwitcher.setOutAnimation(AnimationUtils.loa
dAnimation(this,
android.R.anim.fade_out));btnViewWindows = (Button)
                                                findViewById(R.id.button2);
btnViewButterfly = (Button) findViewById(R.id.button);
...
}
```

4.5.1.3 GridView

- The GridView shows items in a two-dimensional scrolling grid. You can use the GridView together with an ImageView to display a series of images.

  Example: Using the GridView View.

  Step 1 : Using Android Studio, create a new Android project and name it Grid.

  Step 2 : Drag and drop a series of images into the res/mipmap folder (see step 3 in the previous example for the images).

  Step 3 : Populate the activity_main.xml file with the following content:

```
<?xml version="1.0"encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_main"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.jfdimarzio.grid.MainActivity"
tools:layout_editor_absoluteX="0dp"
tools:layout_editor_absoluteY="81dp">
<GridView
android:layout_width="384dp"
android:layout_height="511dp"
tools:layout_editor_absoluteX="0dp"tools:layout_editor_absoluteY
="0dp"
app:layout_constraintLeft_toLeftOf="@+id/activity_main"
tools:layout_constraintLeft_creator="0"
app:layout_constraintTop_toTopOf="@+id/activity_main"
tools:layout_constraintTop_creator="0"
app:layout_constraintRight_toRightOf="@+id/activity_main"
tools:layout_constraintRight_creator="0"
app:layout_constraintBottom_toBottomOf="@+id/activity_main"
tools:layout_constraintBottom_creator="0"
android:id="@+id/gridview" />
</android.support.constraint.ConstraintLayout>
```

  Step 4 : Add the following bolded statements to the MainActivity.java file:

```
import android.content.Context;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
```

```
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity{
//---the images to display---
Integer[] imageIDs={
R.mipmap.butterfly,R.mipmap.windows,R.mipmap.ic_launcher
};
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
GridView gridView = (GridView) findViewById(R.id.gridview);
gridView.setAdapter(new ImageAdapter(this));
gridView.setOnItemClickListener(new
AdapterView.OnItemClickListener()
{
public void onItemClick(AdapterView parent,
View v, int position, long id)
{
}
});
}
Toast.makeText(getBaseContext(),"pic"+(position+1)+"selected",
Toast.LENGTH_SHORT).show();
public class ImageAdapter extends BaseAdapter
{
private Context context;
public ImageAdapter(Context c)
{
context = c;
}
//---returns the number of images---public int getCount(){
return imageIDs.length;
}
//---returns the item---
public Object getItem(int position) {
return position;
}
```

```
//---returns the ID of an item---public long getItemId(int
position){
return position;
}
//---returns an ImageView view---
public View getView(int position, View convertView, ViewGroup
parent)
{
ImageView imageView;
if (convertView == null){
imageView = new ImageView(context);
imageView.setLayoutParams(new
GridView.LayoutParams(150, 150));
imageView.setScaleType(
ImageView.ScaleType.CENTER_CROP);imageView.setPadding(5, 5, 5,
5);
} else {
imageView = (ImageView) convertView;
}
imageView.setImageResource(imageIDs[position]);return imageView;
}
}
}
```

Step 5 : Press Shift+F9 to debug the application on the Android emulator. Fig. 4.22 shows the GridView displaying all the images.

How it Works?

- Like the ImageSwitcher example, you first implement the ImageAdapter class and then bind it to the GridView:

```
GridView gridView = (GridView)
findViewById(R.id.gridview);
gridView.setAdapter(new ImageAdapter(this));
gridView.setOnItemClickListener(newOnItemClickListener(){
public void onItemClick(AdapterView parent, View v, int position, long id)
{
Toast.makeText(getBaseContext(),
"pic" + (position +1) + " selected",
Toast.LENGTH_SHORT).show();
}
});
```

Fig. 4.22: GridView

4.5.2 Using Menus with Views

- Menus are useful for displaying additional options that are not directly visible on the main User Interface (UI) of an application.
- In android, Menu is a part of User Interface (UI) component which is used to handle some common functionality around the application.
- By using Menus in our applications, we can provide better and consistent user experience throughout the application.
- In android application menu is one of the important user interface entity which provides some action options for a particular view.
- There are two main types of menus in Android:
  1. **Options menu:** This menu displays information related to the current activity. In Android, you activate the options menu by pressing the Menu button.
  2. **Context menu:** This menu displays information related to a particular view on an activity. In Android, you tap and hold a context menu to activate it.

4.5.2.1 Creating the Helper Methods

- Before you go ahead and create your options and context menus, you need to create two helper methods. One creates a list of items to show inside a menu, where as the other handles the event that is fired when the user selects an item inside the menu.

  Example: Creating the Menu Helper Methods.

  Step 1 : Using Android Studio, create a new Android project and name it **Menus**.

  Step 2 : In the MainActivity.java file, add the following bolded statements:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
private void createMenu(Menu menu){
MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");
{
mnu1.setAlphabeticShortcut('a');
}
MenuItem mnu2 = menu.add(0, 1, 1, "Item 2");
{
mnu2.setAlphabeticShortcut('b');
}
MenuItem mnu3 = menu.add(0, 2, 2, "Item 3");
{
mnu3.setAlphabeticShortcut('c');
}
MenuItem mnu4 = menu.add(0, 3, 3, "Item 4");
{
mnu4.setAlphabeticShortcut('d');
}
menu.add(0, 4, 4, "Item5");
menu.add(0, 5, 5, "Item6");

menu.add(0, 6, 6, "Item7");
}
private boolean MenuChoice(MenuItem item) {
switch (item.getItemId()){
case 0:
Toast.makeText(this, "You clicked on Item1",
Toast.LENGTH_LONG).show();
return true;
case 1:
Toast.makeText(this,       "You       clicked       on       Item
2",Toast.LENGTH_LONG).show();
return true;
```

```
case 2:
Toast.makeText(this,"You clicked on Item3",
Toast.LENGTH_LONG).show();
return true;
case 3:
Toast.makeText(this, "You clicked on Item4",
Toast.LENGTH_LONG).show();
return true;
case 4:
Toast.makeText(this, "You clicked on Item5",
Toast.LENGTH_LONG).show();
return true;
case 5:
Toast.makeText(this,"You clicked on Item6",
Toast.LENGTH_LONG).show();
return true;
case 6:
Toast.makeText(this, "You clicked on Item7",
Toast.LENGTH_LONG).show();
return true;
}
return false;
}
}
```

4.5.2.2 Options Menu

- You are now ready to modify the application to display the options menu when the user presses the Menu key on the Android device.

  Example: Displaying an Options Menu.

  Step 1 : Using the same project created in the previous section, add the following bolded statements to the

```
MainActivity.javafile:
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
super.onCreateOptionsMenu(menu); createMenu(menu);
return true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
return menuChoice(item);
}
private void createMenu(Menu menu){
MenuItem mnu1 = menu.add(0, 0, 0, "Item 1”);
{
mnu1.setAlphabeticShortcut('a');
}
MenuItem mnu2 = menu.add(0, 1, 1, "Item 2");
{
mnu2.setAlphabeticShortcut('b');
}
MenuItem mnu3 = menu.add(0, 2, 2, "Item 3");
{
mnu3.setAlphabeticShortcut('c');
}
MenuItem mnu4 = menu.add(0, 3, 3, "Item 4");
{
mnu4.setAlphabeticShortcut('d');
}
menu.add(0, 4, 4, "Item 5");
menu.add(0, 5, 5, "Item 6");
menu.add(0, 6, 6, "Item 7");
}
private boolean menuChoice(MenuItem item) {
switch (item.getItemId()){
case 0:
Toast.makeText(this, "You clicked on Item 1",
Toast.LENGTH_LONG).show();
return true;
case 1:
Toast.makeText(this, "You clicked on Item 2",
Toast.LENGTH_LONG).show();
return true;
```

```
case 2:
Toast.makeText(this, "You clicked on Item 3",
Toast.LENGTH_LONG).show();
return true;
case3:
Toast.makeText(this, "You clicked on Item 4",
Toast.LENGTH_LONG).show();
return true;
case 4:
Toast.makeText(this, "You clicked on Item 5",
Toast.LENGTH_LONG).show();
return true;
case 5:
Toast.makeText(this, "You clicked on Item 6",
Toast.LENGTH_LONG).show();
return true;
case 6:
Toast.makeText(this, "You clicked on Item 7",
Toast.LENGTH_LONG).show();
return true;
}
return false;
}
}
```

Step 2 : Press Shift+F9 to debug the application on the Android emulator. Fig. 4.23 shows the options menu that displays when you click the Menu button. To select a menu item, either click an individual item or use its shortcut key (A to D, and applicable only to the first four items).

Fig. 4.23

4.5.2.3 Context Menu

- The previous section showed how the options menu is displayed when the user press the Menu button. In addition to the options menu, you can also display a context menu.
- A context menu is usually associated with a view on an activity. A context menu is displayed when the user taps and holds an item. For example, if the user taps a Button view and holds it for a few seconds, a context menu can be displayed.

Example: Displaying a Context Menu.

Step 1 : Using the same project from the previous example, add the following bolded statements to the activity_main.xml file:

```
<?xml version="1.0"encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_main"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.jfdimarzio.menus.MainActivity"
tools:layout_editor_absoluteX="0dp"
tools:layout_editor_absoluteY="81dp">
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello World!"
tools:layout_editor_absoluteX="154dp"
tools:layout_editor_absoluteY="247dp"
app:layout_constraintLeft_toLeftOf="@+id/activity_main"
tools:layout_constraintLeft_creator="0"
app:layout_constraintTop_toTopOf="@+id/activity_main"
tools:layout_constraintTop_creator="0"
app:layout_constraintRight_toRightOf="@+id/activity_main"
tools:layout_constraintRight_creator="0"
app:layout_constraintBottom_toBottomOf="@+id/activity_main"
tools:layout_constraintBottom_creator="0"/>
<Button
android:text="Button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
tools:layout_editor_absoluteX="148dp"
tools:layout_editor_absoluteY="102dp"
```

```
android:id="@+id/button"
app:layout_constraintLeft_toLeftOf="@+id/activity_main"
tools:layout_constraintLeft_creator="0"
app:layout_constraintRight_toRightOf="@+id/activity_main"tools:l
ayout_constraintRight_creator="0"/>
</android.support.constraint.ConstraintLayout>
```

Step 2 : Add the following bolded statements to the MenusActivity.java file:

```
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState){
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Button btn = (Button) findViewById(R.id.button);
btn.setOnCreateContextMenuListener(this);
}
@Override
public void onCreateContextMenu(ContextMenu menu,
View view, ContextMenu. ContextMenuInfo menuInfo)
{
super.onCreateContextMenu(menu, view, menuInfo);
createMenu(menu);
}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
super.onCreateOptionsMenu(menu); createMenu(menu);
return true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
return menuChoice(item);
}
private void createMenu(Menu menu){
```

```
MenuItem mnu1 = menu.add(0, 0, 0, "Item 1");
{
mnu1.setAlphabeticShortcut('a');
}
MenuItem mnu2= menu.add(0, 1, 1, "Item 2");
{
mnu2.setAlphabticShortcut('b');
}
MenuItem mnu3= menu.add(0, 2, 2, "Item 3");
{
mnu3.setAlphabeticShortcut('c');
}
MenuItem mnu4= menu.add(0, 3, 3, "Item 4");
{
mnu4.setAlphabeticShortcut('d');
}
menu.add(0, 4, 4, "Item 5");
menu.add(0, 5, 5, "Item 6");
menu.add(0, 6, 6, "Item 7");
}
private boolean menuChoice(MenuItem item) {switch
(item.getItemId()){
case 0:
Toast.makeText(this, "You clicked on Item 1",
Toast.LENGTH_LONG).show();
return true;
case 1:
Toast.makeText(this, "You clicked on Item 2",
Toast.LENGTH_LONG).show();
return true;
case 2:
Toast.makeText(this, "You clicked on Item 3",
Toast.LENGTH_LONG).show();
return true;
case 3:
Toast.makeText(this, "You clicked on Item 4",
Toast.LENGTH_LONG).show();
return true;
case 4:
Toast.makeText(this, "You clicked on Item 5",
Toast.LENGTH_LONG).show();
return true;
```

```
case 5:
Toast.makeText(this, "You clicked on Item 6",
Toast.LENGTH_LONG).show();
return true;
case 6:
Toast.makeText(this, "You clicked on Item 7",
Toast.LENGTH_LONG).show();
return true;
}
return false;
}
}
```

Step 3 : Press Shift+F9 to debug the application on the Android emulator. Fig. 4.24 shows the context menu that displays when you click and hold the Button view.

Fig. 4.24

4.6 VideoVIEW

- VideoView is used to display a video file. It can load images from various sources (such as content providers or resources) taking care of computing its measurement from the video so that it can be used for any layout manager, providing display options such as scaling and tinting.
- VideoView does not retain its full state when going into the background.
- In particular it does not restore the current play position and play state. Applications should save and restore these in onSaveInstanceState(Bundle) and onRestoreInstanceState(Bundle).

Example:

```
<VideoView
android:id="@+id/simpleVideoView"
android:layout_width="fill_parent"
android:layout_height="fill_parent" />
```

4.6.1 Play Video From URL with using VideoView

- setVideoUri(Uri uri): This method is used to set the absolute path of the video file which is going to be played. This method takes a Uri object as an argument.
- Below we set the uri of video which is saved in Android Studio:

  Step 1 : Create a new directory in res folder and name it raw.

  Step 2 : Save a video name fishvideo in raw folder.

  Step 3 : Now use the below code to set the path for the video using setVideoUri() method in VideoView.

```
// initiate a video view
VideoView simpleVideoView = (VideoView)
findViewById(R.id.simpleVideoView);
simpleVideoView.setVideoURI(Uri.parse("android.resource://" +
getPackageName() + "/" + R.raw.fishvideo));
```

- Setting Video From Online Web Source:

  Step 1 : First add internet permission in Manifest.xml file. We will need to add this so as to access the video through Internet. Open AndroidManifest.xml and add the below code:

```
<!--Add this before application tag in AndroidManifest.xml-->
<uses-permission android:name="android.permission.INTERNET" />
```

  Step 2 : Add the basic VideoVideo XML code in activity_main.xml or activity.xml

  Step 3 : Use the below code to access the Video from our website

```
package abhiandroid.com.videofromwebsource;
import android.app.ProgressDialog;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.VideoView;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Uri uri = Uri.parse("/ui/wp-
content/uploads/2016/04/videoviewtestingvideo.mp4");
```

```
VideoView simpleVideoView = (VideoView)
findViewById(R.id.simpleVideoView);
// initiate a video view
   simpleVideoView.setVideoURI(uri);
   simpleVideoView.start();
   }
}
```

4.6.2 VideoView Create

Step 1 : Create a new project in Android Studio and name it VideoViewExample

Step 2 : Open res -> layout -> xml (or) main.xml and add following code:

In this step we open an xml file and add the code to display a VideoView in our activity.

```
<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=".MainActivity">
<VideoView
android:id="@+id/simpleVideoView"
android:layout_width="match_parent"
android:layout_height="match_parent" />
</RelativeLayout>
```

Step 3 : Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code to initiate the video view and create an object of MediaController to control the video playback.

```
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.res.Configuration;
import android.media.MediaPlayer;
import android.media.MediaPlayer.OnPreparedListener;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
```

```
import android.view.View;
import android.widget.MediaController;
import android.widget.Toast;
import android.widget.VideoView;
public class MainActivity extends Activity {
       VideoView simpleVideoView;
       MediaController mediaControls;
       @Override
       protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_main);
          // Find your VideoView in your video_main.xml layout
          simpleVideoView = (VideoView)
findViewById(R.id.simpleVideoView);
          if (mediaControls == null) {
          // create an object of media controller class
          mediaControls = new
MediaController(MainActivity.this);
   mediaControls.setAnchorView(simpleVideoView);
          }
        // set the media controller for video view
          simpleVideoView.setMediaController(mediaControls);
        // set the uri for the video view
   simpleVideoView.setVideoURI(Uri.parse("android.resource://" +
getPackageName() + "/" + R.raw.fishvideo));
        // start a video
          simpleVideoView.start();
        // implement on completion listener on video view
          simpleVideoView.setOnCompletionListener(new
MediaPlayer.OnCompletionListener() {
       @Override
       public void onCompletion(MediaPlayer mp) {
          Toast.makeText(getApplicationContext(), "Thank
You...!!!", Toast.LENGTH_LONG).show();
// display a toast when an video is completed
            }
        });
        simpleVideoView.setOnErrorListener(new
MediaPlayer.OnErrorListener() {
       @Override
       public boolean onError(MediaPlayer mp, int what, int
extra) {
```

```
            Toast.makeText(getApplicationContext(), "Oops An Error
Occur While Playing Video...!!!", Toast.LENGTH_LONG).show();
// display a toast when an error is occured while playing an
video
                return false;
            }
        });
    }
}
```

Summary

- The Android TextView component is a View subclass which is capable of showing text. Being a subclass of View the TextView component can be used in your Android app's GUI inside a ViewGroup, or as the content view of an activity.
- The View is a base class for all UI components in Android and it is used to create an interactive UI components such as TextView, EditText, Checkbox, RadioButton, etc. and it responsible for event handling and drawing.
- The basic views that you can use to design the UI of your Android applications contains TextView, EditText, Button, ImageButton, CheckBox, ToggleButton, RadioButton, RadioGroup.
- The TextView view is used to display text to the user. A TextView displays text to the user and optionally allows them to edit it.
- Button is a user interface control which is used to perform an action whenever the user click or tap on it. Generally, Buttons in Android will contains a text or an icon or both and perform an action when user touches it.
- EditText is a user interface control which is used to allow the user to enter or modify the text. In Android, EditText control is an extended version of TextView control with an additional features and it is used to allow users to enter input values.
- CheckBox is a two states button that can be either checked (ON) or unchecked (OFF) and it will allow users to toggle between the two states (ON/OFF) based on the requirements. Generally, we can use multiple CheckBox controls in Android application to allow users to select one or more options from the set of values.
- Android Toggle Button is used to display on and off state on a button.
- In Android, RadioButton is a two states button that can be either checked or unchecked and it's a same as CheckBox control, except that it will allow only one option to select from the group of options.
- In Android, RadioGroup is used to group one or more radio buttons into separate groups based on our requirements. RadioGroup contains several radio buttons, marking one radio button as checked makes all other radio buttons as unchecked.

- In Android, Progress Bar is a user interface control which is used to indicate the progress of an operation. For example, downloading a file, uploading a file.
- In Android, AutoComplete TextView is an editable textview which is used to show the list of suggestions based on the user typing text. The list of suggestions will be shown as a dropdown menu from which the user can choose an item to replace the content of textbox.
- Selecting the date and time is one of the common tasks you need to perform in a mobile application. Android supports this functionality through the TimePicker and DatePicker views.
- In Android, TimePicker is a widget for selecting the time of day, in either 24-hour or AM/PM mode. In android, DatePicker is a control which will allow users to select the date by day, month and year in our application user interface.
- Android ListView is a view which groups several items and display them in vertical scrollable list.
- In Android, Spinner is a view which allows a user to select one value from the list of values. The spinner in android will behave same like drop down list in other programming languages.
- A fragment that displays a list of items by binding to a data source such as an array or Cursor, and exposes event handlers when the user selects an item.
- ListFragment hosts a ListView object that can be bound to different data sources, typically either an array or a Cursor holding query results.
- A DialogFragment is a fragment that displays a dialog window, floating on top of its activity's window. This fragment contains a Dialog object, which it displays as appropriate based on the fragment's state.
- PreferenceFragment shows a hierarchy of Preference objects as lists.
- Android provides views which can be used to display images from various sources and provide transitions between them. Some of these views are the ImageView and the ImageSwitcher.
- In Android you can use the ImageView component to display images in your Application.
- Android Gallery is a View commonly used to display items in a horizontally scrolling list that locks the current selection at the center.
- Android ImageSwitcher provides an animation over images to transition from one image to another. In order to use ImageSwitcher, we need to implement ImageSwitcher component in.xml file.
- GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid. The grid items are automatically inserted to the layout using a ListAdapter.
- In Android, Menu is a part of User Interface (UI) component which is used to handle some common functionality around the application. By using Menus in our applications, we can provide better and consistent user experience throughout the application.

- Android Option Menus are the primary menus of android. They can be used for settings, search, delete item etc.
- Android context menu appears when user press long click on the element. It is also known as floating menu. It affects the selected content while doing action on it. It doesn't support item shortcuts and icons.
- The Options Menu is the primary collection of menu items for an activity. It's where you should place actions that have a global impact on the app, such as "Search", "Compose email", and "Settings."
- A Context menu is a floating menu that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.
- A Popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu. Popup menus this is also a floating window. It displays a list of items anchored to the view that the user clicked.
- The GridView shows items in a two-dimensional scrolling grid. You can use the GridView together with an ImageView to display a series of images.

Practice Questions

Q.I Multiple Choice Questions:

1. _____ is used to display text to the user.
   (a) TextView (b) EditText
   (c) Both (a) and (b) (d) None of the Mentioned
2. _____ is a drop-down list that allows users to select one value from a set.
   (a) CheckBox (b) RadioButton
   (c) Spinner (d) All of the Mentioned
3. _____ hosts a ListView object that can be bound to different data sources, typically either an array or a Cursor holding query results.
   (a) DialogFragment (b) PreferenceFragment
   (c) ListFragment (d) All of the Mentioned
4. _____ is a view which groups several items and display them in vertical scrollable list.
   (a) TextView (b) ListView
   (c) Button (d) None of the Mentioned
5. The AutoCompleteTextView is a view that is similar to _____, except that it shows a list of completion suggestions automatically while the user is typing.
   (a) EditText (b) TextText
   (c) CheckBox (d) All of the Mentioned

6. The _____ and the _____ views provide a high level of functionality to display images in a user interface.
   (a) ImageView (b) ImageSwitcher
   (c) Both(a) and (b) (d) None of the Mentioned
7. The _____ is a view that shows items (such as images) in a center locked, horizontal scrolling list.
   (a) Gallery (b) GridView
   (c) ImageView (d) All of the Mentioned
8. _____ is a specialized ViewSwitcher which will provide a smooth transition animation effect to the images while switching from one image to another.
   (a) ImageView (b) ImageSwitcher
   (c) Both (a) and (b) (d) None of the Mentioned
9. _____ shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a ListAdapter.
   (a) GridView (b) ImageView
   (c) TextView (d) All of the Mentioned
10. _____ menu displays information related to the current activity.
   (a) Contexts (b) Options
   (c) Popups (d) All of the Mentioned
11. _____ are useful for displaying additional options that are not directly visible on the main UI of an application.
   (a) Images (b) Menus
   (c) Gallery (d) None of the Mentioned
12. Android support _____ types of menus.
   (a) Context (b) Option
   (c) Popup (d) All of the Mentioned
13. _____ is a ViewGroup that displays items in a two-dimensional, scrollable format.
   (a) GridView (b) ListView
   (c) Both(a) and (b) (d) None of the Above

Answers

1. (a)	2. (c)	3. (c)	4. (b)	5. (a)	6. (c)	7. (a)	8. (b)	9. (a)	10. (b)
11. (b)	12. (d)	13. (a)							

Q.II Short Answer the following Questions:

1. What is the TextView?
2. With the help of example explain RadioButton.
3. What is meant by button? Describe with example.

4. What is the EditText?
5. Explain the AutoCompleteTextView in detail.
6. Write short note on: ProgressBar.
7. What is ToggleButton? How to create it? Explain with example.
8. Describe the following terms:
   (i) DatePicker
   (ii) TimePicker.
9. What is TextView?
10. What is meant by ProgressBar? Describe With Example.
11. What is the gallery in Android?
12. What is GridView in Android?
13. What is ImageView in Android?
14. What is menu? Enlist its types.

Q.III Long Answer the following Questions:

1. What is ListView? How to use it in Android application? Explain with example.
2. Explain ListFragment with example.
3. What is DialogFragment? Explain its uses.
4. With the help of example explain spinner.
5. With the help of diagram describe the following terms:
   (i) ImageSwitcher
   (ii) ContextMenu
6. Explain GridView with the help of example.

Q.IV Short notes :

1. Button
2. CheckBox
3. RadioGroup
4. PreferenceFragment
5. Creating the helper methods.

■■■

5...

Databases SQLite Messaging and E-mail

Learning Objectives ...

Students will be able to:

- To learn Basic Concepts of SQLite Database.
- To study Operations (Insert, Update, Delete etc.) on SQLite Database.
- To understand Cursor Concept with Example.

5.1 INTRODUCTION TO SQLite

- Persisting data is an important topic in application development, as users expect to reuse the data sometime at a later stage. For Android, there are primarily three basic ways of persisting data:
  1. A lightweight mechanism known as shared preferences to save small chunks of data.
  2. Traditional file systems.
  3. A relational database management system through the support of SQLite databases.
- DataBase is one of the ways of storing data. However we should prefer using a DB only if the data we are dealing with is structured data like Employee details of an organization.
- Android uses the SQLite database system. The database that you create for an application is only accessible to itself; other applications will not be able to access it.
- SQLite is an open source light weight Relational Database Management System (RDBMS) to perform database operations, such as storing, updating, retrieving data from database.
- Android SDK comes with built in support for SQLite Database. Android SQLite Database is a light weight database mainly useful for embedded applications, it reads and writes directly to disk files.

- Generally in our Android applications, Shared Preferences, Internal Storage and External Storage options are useful to store and maintain the small amount of data. In case, if we want to deal with large amount of data, then SQLite database is the preferable option to store and maintain the data in structured format.
- By default, Android comes with built-in SQLite Database support so we don't need to do any configurations. Just like we save the files on device's internal storage, Android stores our database in a private disk space that's associated to our application and the data is secure, because by default this area is not accessible to other applications.
- The package android.database.sqlite contains all the required API's to use SQLite database in our android applications.
- All the classes and interfaces that are required to work with Android SQLite Database are available in the package, android.database.sqlite.
- The SQLite Database class represents a file on the Android device. You can control the name of the file which contains the DB and it will be a single file rather than multiple scattered files.
- Let's review some of the important classes that we'll use when working with an SQLite database:

Table 5.1: Classes of SQLite Database

Sr. No.	Class	Description
1.	SQLiteDatabase	represents the Android SQLite database.
2.	SQLiteOpenHelper	a helper class that manages the Android SQLite DB (DATABASE).
3.	SQLiteQuery	represents an Android SQLite DB query.
4.	SQLiteStatement	represents an Android SQLite statement.
5.	SQLiteCursor	exposes the results from a query, use to iterate through the results from the query.
6.	SQLiteQueryBuilder	a helper class to build and manage queries
7.	SQLiteException	represents an Android SQLite Exceptions.

Features of SQLite:

1. **Zero-configuration:** SQLite is designed in such a manner that it requires no configuration file. It requires no installation steps or initial setup; it has no server process running and no recovery steps to take even if it crashes.
2. **Cross-platform:** Database files from one system can be moved to a system running a different architecture without any hassle. This is possible because the database file format is binary and all the machines use the same format.

3. **Compact:** A SQLite database is a single ordinary disk file; it comes without a server and is designed to be light weight and simple. These attributes lead to a very light weight database engine.
4. **No-copyright:** The source code of SQLite is in the public domain; you are free to modify, distribute, and even sell the code.

5.2 SQLiteOpenHelper AND SQLite DATABASE

1. **SQLiteOpenHelper:**

- The SQLiteOpenHelper class is the first and most essential class of Android to work with SQLite databases; it is present in the android.database.SQLite namespace.
- SQLiteOpenHelper is a helper class that is designed for extension and to implement the tasks and actions you see as important when creating, opening, and using a database. This helper class is provided by the Android framework to work with the SQLite database and helps in managing the database creation and version management.
- For managing all the operations related to the database, a helper class has been given and is called SQLiteOpenHelper. It automatically manages the creation and update of the database.
- Its syntax is given below:

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

2. **SQLiteDatabase:**

- SQLiteDatabase is the base class required to work with a SQLite database in Android and provides methods to open, query, update, and close the database.
- SQLiteDatabase has methods to create, delete, execute SQL commands, and perform other common database management tasks.
- SQLite is an open-source relational database i.e. used to perform database operations on android devices such as storing, manipulating or retrieving persistent data from the database.

Create Database and Tables using SQLiteHelper:

- In Android, by using SQLiteOpenHelper class we can easily create required database and tables for our application. To use SQLiteOpenHelper, we need to create a subclass that overrides the onCreate() and onUpgrade() call-back methods.

- Following is the code snippet of creating database and tables using SQLiteOpenHelper class in our Android application.

```
public class DbHandler extends SQLiteOpenHelper
{
    private static final int DB_VERSION = 1;
    private static final String DB_NAME = "usersdb";
    private static final String TABLE_Users = "userdetails";
    private static final String KEY_ID = "id";
    private static final String KEY_NAME = "name";
    private static final String KEY_LOC = "location";
    private static final String KEY_DESG = "designation";
    public DbHandler(Context context)
    {
        super(context, DB_NAME, null, DB_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db)
    {
        String CREATE_TABLE = "CREATE TABLE" + TABLE_Users + "("
        +KEY_ID+"INTEGER PRIMARY KEY AUTOINCREMENT,"+KEY_NAME + "TEXT,"
        + KEY_LOC + "TEXT,"
        + KEY_DESG + " TEXT"+ ")";
        db.execSQL(CREATE_TABLE);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
    newVersion)
    {
        //Drop older table if exist
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_Users);
        // Create tables again
        onCreate(db);
    }
}
```

- If you observe above code snippet, we are creating database "usersdb" and table "userdetails" using SQLiteOpenHelper class by overriding onCreate and onUpgrade methods.

Creating, Opening and Closing Database:

- Superclass does the hardwork of setting up the database. The onCreate() method is called automatically by the runtime when it needs to create the database. This operation runs only once, when the app first Launches and the database has not yet been created.
- The code in listing 5-11 demonstrates using Database Helper to open and close the database. The constructor saves an instance of Context, which is passed to DatabaseHelper. The open() method initializes the helper and uses it to get an instance of the database, while the close() method uses the helper to close the database. Add this code after all the member variable.

```
//---opens the database---
public DBAdapter open()throws SQLException
{
    db = DBHelper.getwritableDatabase();
    return this;
}
//---closes the database---public void close()
{
    DBHelper.close();
}
```

5.3 CREATING AND USING DATABASE

- Android uses the SQLite database system. The database that you create for an application is only accessible to itself; other applications will not be able to access it.
- In this section, you find out how to create programmatically a SQLite database in your Android application. For Android, this SQLite database is always stored in the /data/data/<package_name>/databases folder.

Creating the DBAdapter Helper Class:

- A good practice for dealing with databases is to create a helper class to encapsulate all the complexities of accessing the data so that it is transparent to the calling code. For this section, you create a helper class called DBAdapter, which creates, opens, closes, and uses a SQLite database.
- In this example, you are going to create a database named MyDB containing one table named contacts. This table has three columns namely_id, name, and e-mail.

Creating the Database Helper Class:

Step 1 : Using Android Studio, create an Android project and name it Databases.

Step 2 : Add a new Java Class file to the package and name it DBAdapter (See Fig. 5.1).

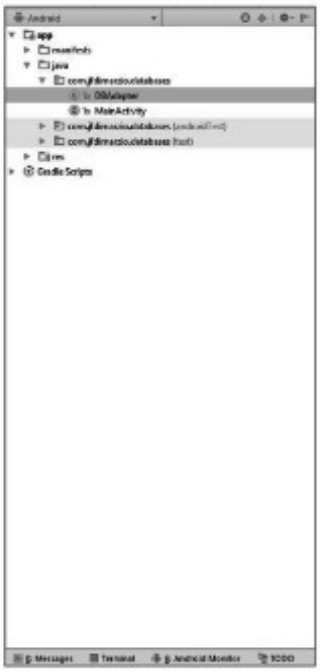


Fig. 5.1

Step 3 : Add the following bolded statements to the DBAdapter.java file:

```
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;
public class DBAdapter{
static final String KEY_ROWID = "_id";
static final String KEY_NAME = "name";
static final String KEY_EMAIL = "e-mail";
static final String TAG = "DBAdapter";
static final String DATABASE_NAME = "MyDB";
static final String DATABASE_TABLE = "contacts";
static final int DATABASE_VERSION = 1;
static final String DATABASE_CREATE =
"create table contacts (_id integer primary key autoincrement,"
+ "name text not null, e-mail text not null);";
final Context context;
DatabaseHelper DBHelper;
SQLiteDatabase db;
```

```
public DBAdapter(Context ctx)
{
this.context = ctx;
DBHelper = new DatabaseHelper(context);
}
private static class DatabaseHelper extends SQLiteOpenHelper
{
DatabaseHelper(Context context)
{
super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
@Override
public void onCreate(SQLiteDatabase db)
{
try {
db.execSQL(DATABASE_CREATE);
} catch (SQLException e)
{ e.printStackTrace();
}

}
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion)
{
Log.w(TAG, "Upgrading database from version" + oldVersion + "to"
+ newVersion + ", which will destroy all old data");
db.execSQL("DROP TABLE IF EXISTS contacts");
onCreate(db);
}
}
//---opens the database---
public DBAdapter open() throws SQLException
{
db =  DBHelper.getWritableDatabase();
return this;
}
//---closes the database---public void close()
{
DBHelper.close();
}
```

```
//---insert a contact into the database---
public long insertContact(String name, String email)
{
ContentValues initialValues = new ContentValues();
initialValues.put(KEY_NAME, name);
initialValues.put(KEY_EMAIL, email);
return db.insert(DATABASE_TABLE, null, initialValues);
}
//---deletes a particular contact---public boolean
deleteContact(long rowId)
{
return db.delete(DATABASE_TABLE, KEY_ROWID + "=" +rowId, null) > 0;
}
//---retrieves all the contacts---public Cursor getAllContacts()
{
return db.query(DATABASE_TABLE, new String[] {KEY_ROWID,
   KEY_NAME, KEY_EMAIL}, null, null, null, null, null);
}
//---retrieves a particular contact---
public Cursor getContact(long rowId) throws SQLException
{
Cursor mCursor=
db.query(true, DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME,
KEY_EMAIL},
KEY_ROWID + "=" + rowId, null, null, null, null, null);
if (mCursor!=null){
mCursor.moveToFirst();
}
return mCursor;
}
//---updates a contact---
public boolean updateContact(long rowId, String name, String email)
{
ContentValues args = new ContentValues();
args.put(KEY_NAME, name);
args.put(KEY_EMAIL, email);
return db.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId,
null)>
0;
}
}
```

Using the Database Programmatically:

- With the DBAdapter helper class created, you are now ready to use the database. In the following sections, you will learn how to perform the regular CRUD (Create, Read, Update and Delete) operations commonly associated with databases.

Adding Contacts:

- The following example demonstrates how you can add a contact to the table.

  Example: Adding Contacts to a Table.

  Step 1 : Using the same project created earlier, add the following bolded statements to the MainActivity.java file:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity{
@Override
protected void onCreate(Bundle savedInstanceState)
{super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
DBAdapter db = newDBAdapter(this);
//---add a contact---db.open();
long id = db.insertContact("Jennifer
Ann","jenniferann@jfdimarzio.com");
id = db.insertContact("Oscar Diggs", "oscar@oscardiggs.com");
db.close();
}
}
```

  Step 2 : Press Shift+F9 to debug the application on the Android emulator.

Retrieving All the Contacts:

- To retrieve all the contacts in the contacts table, use the getAllContacts() method of the DBAdapter class, as the following example shows.

  Example: Retrieving All Contacts from a Table.

  Step 1 : Using the same project created earlier, add the following bolded statements to the MainActivity.java file:

```
import android.database.Cursor;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.activity_main);DBAdapter db = new
DBAdapter(this);
//---add a contact---
db.open();
long id = db.insertContact("Jennifer Ann",
"jenniferann@jfdimarzio.com");
id = db.insertContact("Oscar Diggs",
"oscar@oscardiggs.com"); db.close();
db.open();
Cursor c = db.getAllContacts();
if (c.moveToFirst())
{
do {
DisplayContact(c);
} while (c.moveToNext());
}
db.close();
}
public void DisplayContact(Cursor c)
{
Toast.makeText(this, "id: " + c.getString(0) + "\n" + "Name: " +
c.getString(1) + "\n" + "Email: " + c.getString(2),
Toast.LENGTH_LONG).show();
}
}
```

Step 2 : Press Shift+F9 to debug the application on the Android emulator. Fig. 5.2 shows the Toast class displaying the contacts retrieved from the database.

Fig. 5.2

Retrieving a Single Contact:

- To retrieve a single contact using its ID, call the getContact() method of the DBAdapter class, as the following example shows.

  Example: Retrieving a Contact from a Table.

  Step 1 : Using the same project created earlier, add the following bolded statements to the MainActivity.java file:

```
@Override
public void onCreate(Bundle savedInstanceState){
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
DBAdapter db = new DBAdapter(this);
/*
//---add a contact---
...
//--get all contacts---
...
db.close();
*/
//---get a contact---
db.open();
Cursor c = db.getContact(2);
if (c.moveToFirst())
DisplayContact(c);
else
Toast.makeText(this, "No contact found",
Toast.LENGTH_LONG).show();
db.close();
}
```

  Step 2 : Press Shift+F9 to debug the application on the Android emulator. The details of the second contact are displayed using the Toast class.

Updating a Contact:

- To update a particular contact, call the updateContact() method in the DBAdapter class by passing the ID of the contact you want to update, as the following example shows.

  Step 1 : Using the same project created earlier, add the following bolded statements to the MainActivity.java file:

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
```

```
DBAdapter db = newDBAdapter(this);
/*
//---add a contact---
...
//--get all contacts---
...
//---get a contact---
...
db.close();
*/
//---update contact---
db.open();
if (db.updateContact(1, "Oscar Diggs", "oscar@oscardiggs.com"))
Toast.makeText(this, "Update successful.",
Toast.LENGTH_LONG).show();
else
Toast.makeText(this, "Update failed.", Toast.LENGTH_LONG).show();
db.close();
}
```

Step 2 : Press Shift+F9 to debug the application on the Android emulator. A message is displayed if the update is successful.

Deleting a Contact:

- To delete a contact, use the deleteContact() method in the DBAdapter class by passing the ID of the contact you want to update, as the following example shows.

Step 1 : Using the same project created earlier, add the following bolded statements to the MainActivity.java file:

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
DBAdapter db = new DBAdapter(this);
/*
//---add a contact---
...
//--get all contacts---
...
//---get a contact---
...
//---update contact---
...
db.close();
*/
```

```
//---delete a contact---
db.open();
if(db.deleteContact(1))
Toast.makeText(this,              "Delete              successful.",
Toast.LENGTH_LONG).show();
else
Toast.makeText(this, "Delete failed.", Toast.LENGTH_LONG).show();
db.close();
}
```

Step 2 : Press Shift+F9 to debug the application on the Android emulator. A message is displayed if the deletion was successful.

5.4 WORKING WITH CURSORS

- The basic purpose of a cursor is to point to a single row of the result fetched by the query. A cursor is a pointer into one row of that structured data.
- The Cursor class provides methods for moving the cursor through the data structure, and methods to get the data from the fields in each row.
- A query recovers a Cursor object. A Cursor object depicts the result of a query and fundamentally points to one row of the result of the query.
- With this method, Android can buffer the results of the query in a productive manner; as it doesn't need to load all of the data into memory.
- To obtain the elements of the resulting query, you can use the getCount() method. To navigate a mid individual data rows, you can utilize the moveToFirst() and moveToNext() methods.
- To close the Cursor object, the close() method call will be used. A database query returns a cursor. This interface provides random read-write access to the result set. It points to a row of the query result that enables Android to buffer the results effectively since now it is not required to load all the data in the memory.
- The pointer of the returned cursor points to the 0^{th} location, which is known as the first location of the cursor. We need to call the moveToFirst() method on the Cursor object; it takes the cursor pointer to the first location. Now we can access the data present in the first record.
- The following code issued to fetch all records:

```
public Cursor fetch()
{
   String[]     columns     =     new     String[]     {DatabaseHelper._ID,
   DatabaseHelper.SUBJECT, DatabaseHelper.DESC};
   Cursor   cursor   =   database.query(DatabaseHelper.TABLE_NAME,   columns,
   null, null, null, null, null);
```

```
    if (cursor != null)
    {
        cursor.moveToFirst();
    }
return cursor;
}
```

5.5 BUILDING AND EXECUTING QUERIES

1. Building the Create Query:

- To create a table with the desired columns, we will build a query statement and execute it. The statement will contain the table name, different table columns, and respective datatype.
- We will now look at methods for creating a new database and also upgrading an existing database according to the needs of the application:

```
private class CustomSQLiteOpenHelper extends SQLiteOpenHelper
{
    public CustomSQLiteOpenHelper(Context context)
    {
        super(context, DB_NAME, null, DB_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db)
    {
        String newTableQueryString = "create table"
        + TABLE_NAME +" ("
        + TABLE_ROW_ID
        + " integer primary key autoincrement not null,"
        + TABLE_ROW_NAME
        + "text not null,"
        + TABLE_ROW_PHONENUM
        + "text not null,"
        + TABLE_ROW_EMAIL
        +"text not null,"
        + TABLE_ROW_PHOTOID
        +" BLOB"+ ");";
        db.execSQL(newTableQueryString);
    }
    @Override
    public   void   onUpgrade(SQLiteDatabase   db,   int   oldVersion,   int
    newVersion)
    {
        String DROP_TABLE = "DROP TABLE IF EXISTS " + TABLE_NAME;
        db.execSQL(DROP_TABLE);
        onCreate(db);
    }
}
```

2. Executing Queries Insert Query:

- To insert a new row of data in the database table, we need to use either the insert() method or we can make an insert query statement and use the execute() method:

```
public void addRow(ContactModel contactObj)
{
    ContentValues values = prepareData(contactObj);
    try
    {
        db.insert(TABLE_NAME, null, values);
    }
    catch (Exception e)
    {
        Log.e("DB ERROR", e.toString());
        e.printStackTrace();
    }
}
```

- In case if our table name is wrong, SQLite will give a log no such table message and the exception, android.database.sqlite.SQLiteException. The addRow() method is used to insert contact details in the database row; notice that the parameter of the method is an object of ContactModel.

3. Delete Query:

- To delete a particular row of data from our database table, we need to provide the primary key to uniquely identify the data set to be removed:

```
public void deleteRow(int rowID)
{
    try
    {
        db.delete(TABLE_NAME, TABLE_ROW_ID
        + "=" + rowID, null);
    } catch (Exception e)
    {
        Log.e("DB ERROR", e.toString());
        e.printStackTrace();
    }
}
```

- This method uses the SQLiteDatabase delete() method to delete the row of the given ID in the table:

```
public int delete (String table, String whereClause, String[]
whereArgs)
```

4. Update Query:

- To update an existing value, we need to use the update() method with the required parameters:

```
public void updateRow(int rowId, ContactModel contactObj)
{
    ContentValues values = prepareData(contactObj);
    String whereClause = TABLE_ROW_ID +"=?";
    String whereArgs[] = new String[] {String.valueOf(rowId)};
    db.update(TABLE_NAME, values, whereClause, whereArgs);
}
```

- Generally, we need the primary key, in our case the rowed parameter, to identify the row to be modified. An SQLiteDatabase update() method is used to modify the existing data of zero or more rows in a database table:

```
public int update (String table, ContentValues values, String
whereClause, String[] whereArgs)
```

Android SQLite Database Example:

- Following is the example of creating the SQLite database, insert and show the details from SQLite database into android list view using SQLiteOpenHelper class.
- Create a new android application using android studio and give names as SQLiteExample.
- Once we create an application, create a class file DbHandler.java in \java\com.tutlane.sqliteexample path to implement SQLite database related activities for that right click on your application folder
- Go to New -select Java Class and give name as DbHandler.java.
- Once we create a new class file DbHandler.java, open it and write the code like as shown below:

DbHandler.java:

```
package com.tutlane.sqliteexample;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import java.util.ArrayList;
import java.util.HashMap;

/**
*Created by tutlane on 06-01-2018.
*/
```

```
public class DbHandler extends SQLiteOpenHelper
{
    private static final int DB_VERSION = 1;
    private static final String DB_NAME = "usersdb";
    private static final String TABLE_Users = "userdetails";
    private static final String KEY_ID = "id";
    private static final String KEY_NAME = "name";
    private static final String KEY_LOC = "location";
    private static final String KEY_DESG = "designation";
    public DbHandler(Context context)
    {
        super(context, DB_NAME, null, DB_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db)
    {
        String CREATE_TABLE = "CREATE TABLE" + TABLE_Users + "("
        + KEY_ID + "INTEGER PRIMARY KEY AUTOINCREMENT," + KEY_NAME + "TEXT,"
        + KEY_LOC + "TEXT,"
        + KEY_DESG + " TEXT"+ ")";
        db.execSQL(CREATE_TABLE);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
    newVersion)
    {
        //Drop older table if exist
        db.execSQL("DROP TABLE IF EXISTS" + TABLE_Users);
        // Create tables again onCreate(db);
    }
    //****CRUD(Create, Read, Update, Delete) Operations*****//
    // Adding new User Details
    void insertUserDetails(String name, String location, String
    designation)
    {
        //Get the Data Repository in write mode
        SQLiteDatabase db = this.getWritableDatabase();
        //Create a new map of values, where column names are the
        keysContentValues cValues = new ContentValues();
        cValues.put(KEY_NAME, name);
        cValues.put(KEY_LOC, location);
        cValues.put(KEY_DESG, designation);
```

```
    // Insert the new row, returning the primary key value of the new
    row long newRowId = db.insert(TABLE_Users, null, cValues);
    db.close();
}
//Get User Details
public ArrayList<HashMap<String, String>> GetUsers()
{
    SQLiteDatabase db = this.getWritableDatabase();
    ArrayList<HashMap<String, String>> userList = new ArrayList<>();
    String  query  =  "SELECT  name,  location,  designation  FROM  "+
    TABLE_Users;
    Cursor cursor = db.rawQuery(query, null);
    while (cursor.moveToNext())
    {
        HashMap<String, String> user = new HashMap<>();
        user.put("name",cursor.getString(cursor.getColumnIndex
                                                        (KEY_NAME)));
        user.put("designation",cursor.getString(cursor.getColumnIndex
                                                        (KEY_DESG)));
        user.put("location",cursor.getString(cursor.getColumnIndex
                                                         (KEY_LOC)));
        userList.add(user);
    }
    return userList;
}
//Get User Details based on userid
public ArrayList<HashMap<String, String>> GetUserByUserId(int userid)
{
    SQLiteDatabase db = this.getWritableDatabase();
    ArrayList<HashMap<String, String>>userList = new ArrayList<>();
    String  query  =  "SELECT  name,  location,  designation  FROM  "+
    TABLE_Users;
    Cursor cursor = db.query(TABLE_Users, new String[]
    {
        KEY_NAME, KEY_LOC, KEY_DESG}, KEY_ID+"=?",
        new String[]{String.valueOf(userid)},null, null, null, null);
        if (cursor.moveToNext())
        {
            HashMap<String,String> user = new HashMap<>();
            user.put("name",cursor.getString(cursor.getColumnIndex
                                                        (KEY_NAME)));
            user.put("designation",cursor.getString(cursor.getColumnIndex
                                                        (KEY_DESG)));
```

```
            user.put("location",cursor.getString(cursor.getColumnIndex
                                                        (KEY_LOC)));
            userList.add(user);
        }
        return userList;
    }
    // Delete User Details
    public void DeleteUser(int userid)
    {
        SQLiteDatabase db = this.getWritableDatabase();
        db.delete(TABLE_Users, KEY_ID+" = ?",new String[]
                                        {String.valueOf(userid)});
        db.close();
    }
    // Update User Details
    public int UpdateUserDetails(String location, String designation,
    int id)
    {
        SQLiteDatabasedb = this.getWritableDatabase();
        ContentValues cVals = new ContentValues();
        cVals.put(KEY_LOC, location);
        cVals.put(KEY_DESG, designation);
        int count = db.update(TABLE_Users,cVals,KEY_ID+"=?", new
        String[]{String.valueOf(id)});
        return count;
    }
}
```

- If you observe above code, we implemented all SQLite Database related activities to perform CRUD operations in android application.
- Now open activity_main.xml file from \res\layout folder path and write the code like as shown below.

activity_main.xml

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

android:id="@+id/fstTxt"
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"android:layout_marginLeft="100dp"
android:layout_marginTop="150dp"
android:text="Name" />
<EditText
android:id="@+id/txtName"
android:layout_width="wrap_content"
android:layout_height="wrap_content"android:layout_marginLeft="100dp"
android:ems="10"/>
<TextView
android:id="@+id/secTxt"
android:layout_width="wrap_content"
android:layout_height="wrap_content"android:text="Location"
android:layout_marginLeft="100dp"/>
<EditText
android:id="@+id/txtLocation"
android:layout_width="wrap_content"
android:layout_height="wrap_content"android:layout_marginLeft="100dp"
android:ems="10" />
<TextView
android:id="@+id/thirdTxt"
android:layout_width="wrap_content"
android:layout_height="wrap_content"android:text="Designation"
android:layout_marginLeft="100dp"/>
<EditText
android:id="@+id/txtDesignation"
android:layout_width="wrap_content"
android:layout_height="wrap_content"android:layout_marginLeft="100dp"
android:ems="10" />
<Button
android:id="@+id/btnSave"
android:layout_width="wrap_content"
android:layout_height="wrap_content"android:layout_marginLeft="100dp"
android:text="Save" />
</LinearLayout>
```

- Now we will create another layout resource file details.xml in \res\layout path to show the details in custom listview from SQLite Database for that right click on your layout folder Go to New select Layout Resource File and give name as details.xml.

- Once we create a new layout resource file details.xml, open it and write the code like as shown below:

details.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="fill_parent"
android:orientation="vertical">
<ListView
android:id="@+id/user_list"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:dividerHeight="1dp"/>
<Button
android:id="@+id/btnBack"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="20dp"
android:text="Back" />
</LinearLayout>
```

- Create an another layout file (list_row.xml) in /res/layout folder to show the data in listview, for that right click on layout folder add new Layout resource file ·Give name as list_row.xml and write the code like as shown below.

list_row.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:orientation="horizontal"
android:padding="5dip" >
<TextView
android:id="@+id/name"
android:layout_width="wrap_content"
android:layout_height="wrap_content
android:textStyle="bold"
android:textSize="17dp" />
<TextView
android:id="@+id/designation"
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@id/name"
android:layout_marginTop="7dp"
android:textColor="#343434"
android:textSize="14dp" />
<TextView
android:id="@+id/location"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignBaseline="@+id/designation"
android:layout_alignBottom="@+id/designation"
android:layout_alignParentRight="true"
android:textColor="#343434"
android:textSize="14dp" />
</RelativeLayout>
```

- Now open your main activity file MainActivity.java from \java\com.tutlane.sqliteexample path and write the code like as shown below:

MainActivity.java

```
package com.tutlane.sqliteexample;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity
{
    EditText name, loc, desig;
    Button saveBtn;
    Intent intent;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        name = (EditText)findViewById(R.id.txtName);
        loc = (EditText)findViewById(R.id.txtLocation);
        desig = (EditText)findViewById(R.id.txtDesignation);
        saveBtn = (Button)findViewById(R.id.btnSave);
```

```
        saveBtn.setOnClickListener(new View.OnClickListener()
        {
           @Override
           public void onClick(View v)
           {
               String username = name.getText().toString()+"\n";
               String location = loc.getText().toString();
               String designation = desig.getText().toString();
               DbHandler dbHandler = new DbHandler(MainActivity.this);
               dbHandler.insertUserDetails(username,location,designation);
               intent = new Intent(MainActivity.this,DetailsActivity.class);
               startActivity(intent);
               Toast.makeText(getApplicationContext(),"Details Inserted
               Successfully",Toast.LENGTH_SHORT).show();
           }
        });
    }
}
```

- If you observe above code, we are taking entered user details and inserting into SQLite database and redirecting the user to another activity.
- Now we will create another activity file DetailsActivity.java in \java\com.tutlane.sqliteexample path to show the details from SQLite database for that right click on your application folder ·Go to New ·select Java Class and give name as DetailsActivity.java.
- Once we create a new activity file DetailsActivity.java, open it and write the code like as shown below.

DetailsActivity.java

```
package com.tutlane.sqliteexample;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import java.util.ArrayList;
import java.util.HashMap;
public class DetailsActivity extends AppCompatActivity
{
    Intent intent;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.details);
        DbHandler db = newDbHandler(this);
        ArrayList<HashMap<String, String>> userList = db.GetUsers();
        ListView lv = (ListView) findViewById(R.id.user_list);
        ListAdapter adapter = new SimpleAdapter(DetailsActivity.this,
        userList,
        R.layout.list_row,newString[]{"name","designation","location"}, new
        int[]{R.id.name, R.id.designation, R.id.location});
        lv.setAdapter(adapter);
        Button back = (Button)findViewById(R.id.btnBack);
        back.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                intent = new Intent(DetailsActivity.this, MainActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

- If you observe above code, we are getting the details from SQLite database and binding the details to android listview. Now we need to add this newly created activity in **AndroidManifest.xml** file as shown below.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.tutlane.sqliteexample">
<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/AppTheme">
<activity android:name=".MainActivity">
<intent-filter>
```

```
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
<activity android:name=".DetailsActivity"
android:label="SQLite Example-Details"></activity>
</application>
</manifest>
```

- If you observe above example, we are saving entered details in SQLite database and redirecting the user to another activity file (DetailsActivity.java) to show the users details and added all the activities in AndroidManifest.xml file.

Output of Android SQLite Database Example:

- When we run above example in Android Emulator, we will get a result like as shown below.

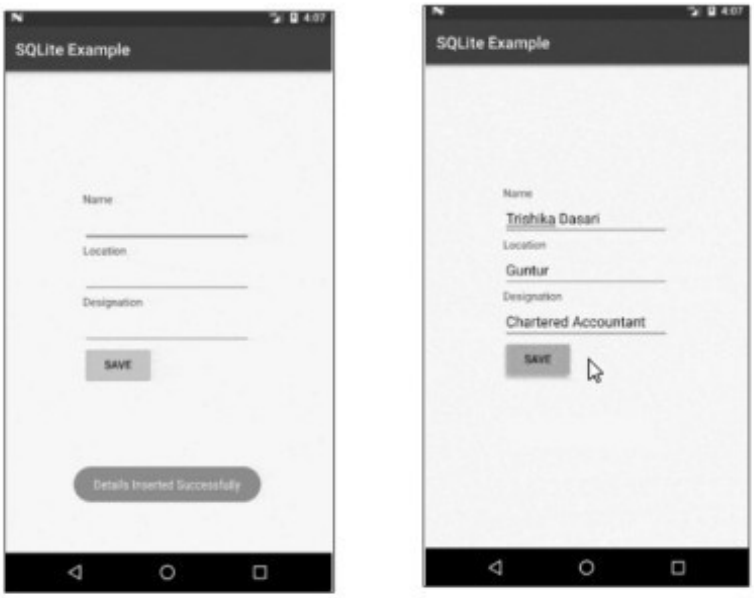


Fig. 5.3: Android SQLite Database

5.6 SMS MESSAGING

- Once, your basic Android application is up and running, the next interesting thing you can add to it is the capability to communicate with the outside world.
- You may want your application to send an SMS message to another phone when an event happens (such as when you reach a particular geographical location), or you may wish to access a Web service that provides certain services (such as currency exchange, weather, etc.).
- In this chapter, you learn how to send and receive SMS messages programmatically from within your Android application.

- SMS (Short Message Service) messaging is one of the main functions on a mobile phone today for some users, it's as necessary as the device itself.
- Today, any mobile phone you buy will have SMS messaging capabilities, and nearly all users of any age know how to send and receive such messages.
- Android comes with a built-in SMS application that enables you to send and receive SMS messages. However, in some cases, you might want to integrate SMS capabilities into your Android application. For example, you might want to write an application that automatically sends an SMS message at regular time intervals.
- For example, this would be useful if you wanted to track the location of your kids simply give them an Android device that sends out an SMS message containing its geographical location every 30 minutes.
- Now you know if they really went to the library after school! (Of course, such a capability also means you would have to pay the fees incurred from sending all those SMS messages.)
- This section describes how you can programmatically send and receive SMS messages in your Android applications. The good news for Android developers is that you don't need a real device to test SMS messaging: The free Android emulator provides that capability.
- In fact, when looking at your emulator window, the four-digit number that appears above your emulator is its "phone number." The first emulator session that you open is typically 5554, with each subsequent session being incremented by 1.

5.6.1 Sending SMS Messages Programmatically

- The first example explains how to send SMS messages programmatically from within your application. Using this approach, your application can automatically send an SMS message to a recipient without user intervention.
- The following example shows you how.

Step 1 : Using Android Studio, create a new Android project and name it SMS.

Step 2 : Replace the TextView with the following bolded statements in the activity_main.xml file. Be sure to replace instances of com.jfdimarzio with the package used in your project:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_main"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.jfdimarzio.sms.MainActivity">
```

```
<Button
android:text="Send SMS"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/btnSendSMS"
app:layout_constraintLeft_toLeftOf="@+id/activity_main"
app:layout_constraintTop_toTopOf="@+id/activity_main"
android:layout_marginTop="16dp"
app:layout_constraintRight_toRightOf="@+id/activity_main"
app:layout_constraintBottom_toBottomOf="@+id/activity_main"
android:layout_marginBottom="16dp"
android:onClick="onClick" />
</android.support.constraint.ConstraintLayout>
```

Step 3 : In the AndroidManifest.xml file, add the following bolded statements. Be sure to replace instances of com.jfdimarzio with the package used in your project:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.jfdimarzio.sms">
<uses-permission android:name="android.permission.SEND_SMS"/>
<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:supportsRtl="true"
android:theme="@style/AppTheme">
activity android:name=".MainActivity">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
/activity>
</application>
</manifest>
```

Step 4 : Add the following statements in bold to the MainActivity.java file:

```
import android.Manifest;
import android.content.pm.PackageManager;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity
{
    final private int REQUEST_SEND_SMS = 123;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if (ContextCompat.checkSelfPermission(this,
        Manifest.permission.SEND_SMS)
        != PackageManager.PERMISSION_GRANTED)
        {
            ActivityCompat.requestPermissions(this,
            new String[]{Manifest.permission.SEND_SMS},
                                                REQUEST_SEND_SMS);
        }
    }
    @Override
    public   void   onRequestPermissionsResult(int   requestCode,
    String[] permissions, int[] grantResults)
    {
        switch (requestCode)
        {
            caseREQUEST_SEND_SMS:
            if (grantResults[0] ==
                                PackageManager.PERMISSION_GRANTED)
            {
                Toast.makeText(MainActivity.this, "Permission
                Granted",Toast.LENGTH_SHORT).show();
            }
            else
            {
                Toast.makeText(MainActivity.this,"Permission
                                Denied",Toast.LENGTH_SHORT).show();
            }
            break;
```

```
            default:
            super.onRequestPermissionsResult(requestCode,
            permissions, grantResults);
        }
    }
    public void onClick(Viewv)
    {
        //---the "phone number" of your emulator should be 5554--
        -sendSMS("5554", "Hello my friends!");
    }
    //---sends an SMS message---
    private void send SMS(String phoneNumber, String message)
    {
        SmsManager sms= SmsManager.getDefault();
        sms.sendTextMessage(phoneNumber, null, message, null, null);
    }
}
```

Step 5 : Press Shift+F9 to debug the application on the Android emulator. Grant the application permission to SEND_SMS when asked.

Step 6 : Click the Send SMS button to send an SMS message. Fig. 5.4 shows the SMS message received (view it by opening the messaging app on the emulator).

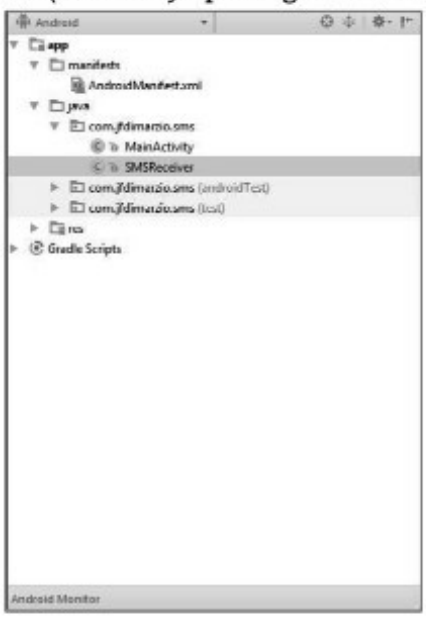


Fig. 5.4

- To send an SMS message programmatically, you use the SmsManager class. Unlike other classes, you do not directly instantiate this class. Instead, you call the getDefault() static method to obtain an SmsManager object.

- You then send the SMS message using the sendTextMessage()method:

```
//---sends an SMS message---
private void sendSMS(String phoneNumber, String message)
{
    SmsManager sms= SmsManager.getDefault();
    sms.sendTextMessage(phoneNumber, null, message, null, null);
}
```

- Following are the five arguments to the sendTextMessage()method:
  1. destinationAddress: Phone number of the recipient.
  2. scAddress: Service center address; use null for default SMSC.
  3. text: Content of the SMS message.
  4. sentIntent: Pending intent to invoke when the message is sent.
  5. deliveryIntent: Pending intent to invoke when the message has been delivered.

5.6.2 Getting Feedback after Sending a Message

- In the previous section, you learned how to programmatically send SMS messages using the SmsManager class; but how do you know that the message has been sent correctly? To do so, you can create two PendingIntent objects to monitor the status of the SMS message-sending process.
- These two PendingIntent objects are passed to the last two arguments of the sendTextMessage() method. The following code snippets show how you can monitor the status of the SMS message being sent:

```
package net.learn2develop.SMS;
import android.app.Activity;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.Toast;
public class SMSActivity extends Activity
{
    String SENT = "SMS_SENT";
    String DELIVERED = "SMS_DELIVERED";
    PendingIntent sentPI, deliveredPI;
    BroadcastReceiver smsSentReceiver, smsDeliveredReceiver;
```

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    sentPI = PendingIntent.getBroadcast(this, 0, new Intent(SENT), 0);
    deliveredPI = PendingIntent.getBroadcast
                                    (this, 0, new Intent(DELIVERED), 0);
}
@Override
public void onResume()
{
    super.onResume();
    //---create the BroadcastReceiver when the SMS is sent---
    smsSentReceiver = newBroadcastReceiver()
    {
        @Override
        public void onReceive(Context arg0, Intent arg1)
        {
            switch(getResultCode())
            {
                case Activity.RESULT_OK:
                Toast.makeText(getBaseContext(), "SMS sent",
                                            Toast.LENGTH_SHORT).show();
                break;
                case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                Toast.makeText(getBaseContext(), “Generic failure”,
                                            Toast.LENGTH_SHORT).show();
                break;
                case SmsManager.RESULT_ERROR_NO_SERVICE:
                Toast.makeText(getBaseContext(), "No service",
                                            Toast.LENGTH_SHORT).show();
                break;
                case SmsManager.RESULT_ERROR_NULL_PDU:
                                        Toast.makeText(getBaseContext(),
                                "Null PDU", Toast.LENGTH_SHORT).show();
                break;
                case SmsManager.RESULT_ERROR_RADIO_OFF:
                                        Toast.makeText(getBaseContext(),
                                "Radio off", Toast.LENGTH_SHORT).show();
                break;
            }
        }
    };
```

```
        //---create the BroadcastReceiver when the SMS is delivered---
        smsDeliveredReceiver = new BroadcastReceiver()
        {
            @Override
            public void onReceive(Context arg0, Intent arg1)
            {
                switch (getResultCode())
                {
                    case Activity.RESULT_OK: Toast.makeText(getBaseContext(),
                                "SMS delivered", Toast.LENGTH_SHORT).show();
                    break;
                    case Activity.RESULT_CANCELED:
                         Toast.makeText(getBaseContext(), "SMS not delivered",
                                                    Toast.LENGTH_SHORT).show();
                    break;
                }
            }
        };
        //---register the two BroadcastReceivers---
        registerReceiver(smsDeliveredReceiver,
             new IntentFilter(DELIVERED)); registerReceiver(smsSentReceiver,
                                                    new IntentFilter(SENT));
    }
    @Override
    public void onPause()
    {
        super.onPause();
        //---unregister the two BroadcastReceivers--
        unregisterReceiver(smsSentReceiver);
    unregisterReceiver(smsDeliveredReceiver);
    }
    public void onClick(View v)
    {
        sendSMS("5556","Hellomyfriends!");
    }
    //---sends an SMS message to another device---
    private void sendSMS(String phoneNumber,String message)
    {
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(phoneNumber,     null,     message,     sentPI,
        deliveredPI);
    }
}
```

- The preceding example created two PendingIntent objects in the onCreate() method:

```
sentPI = PendingIntent.getBroadcast(this, 0, new Intent(SENT), 0);
deliveredPI = PendingIntent.getBroadcast(this, 0, new Intent(DELIVERED),
0);
```

- These two PendingIntent objects will be used to send broadcasts later when an SMS message has been sent ("SMS_SENT") and delivered ("SMS_DELIVERED").
- In the onResume() method, you then created and registered two BroadcastReceivers. These two BroadcastReceivers listen for intents that match "SMS_SENT" and "SMS_DELIVERED" (which are fired by the SmsManager when the message has been sent and delivered, respectively):

```
//---register the two BroadcastReceivers---
registerReceiver(smsDeliveredReceiver, new IntentFilter(DELIVERED));
registerReceiver(smsSentReceiver, new IntentFilter(SENT));
```

- Within each BroadcastReceiver you override the onReceive() method and get the current result code.
- The two PendingIntent objects are passed in to the last two arguments of the sendTextMessage() method:

```
SmsManager sms=SmsManager.getDefault(); sms.sendTextMessage(phoneNumber,
null, message, sentPI, deliveredPI);
```

- In this case, whether a message has been sent correctly or failed to be delivered, you will be notified of its status via the two PendingIntent objects.
- Finally, in the onPause() method, you unregister the two BroadcastReceivers objects.

5.6.3 Sending SMS Messages Using Intent

- Using the SmsManager class, you can send SMS messages from within your application without the need to involve the built-in Messaging application. However, sometimes it would be easier if you could simply invoke the built-in Messaging application and let it handle sending the message.
- To activate the built-in Messaging application from within your application, you can use an Intent object with the MIME type "vnd.android-dir/mms-sms", as shown in the following code snippet:

```
Intent I = new Intent(android.content.Intent.ACTION_VIEW);
i.putExtra("address", "5556; 5558; 5560");
i.putExtra("sms_body", "Hello my friends!");
i.setType("vnd.android-dir/mms-sms");
startActivity(i);
```

- This code invokes the Messaging application directly. Note that you can send your SMS to multiple Recipients by separating each phone number with a semicolon (in the putExtra() method). The numbers are separated using commas in the Messaging application.

5.6.4 Receiving SMS Messages

- Besides sending SMS messages from your Android applications, you can also receive incoming SMS messages from within your applications by using a BroadcastReceiver object. This is useful when you want your application to perform an action when a certain SMS message is received.
- For example, you might want to track the location of your phone in case it is lost or stolen. In this case, you can write an application that automatically listens for SMS messages containing some secret code.
- When that message is received, you can then send an SMS message containing the location's coordinates back to the sender.

Step 1 : Using the same project created in the previous section, add the following bolded statements to the AndroidManifest.xml file. Please be sure to replace all instances of com.jfdimarzio with the package used in your project:

```
<?xml version="1.0"encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.jfdimarzio.sms">
<uses-permissionandroid:name="android.permission.SEND_SMS"/>
<uses-permissionandroid:name="android.permission.RECEIVE_SMS"/>
<application android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:supportsRtl="true" android:theme="@style/AppTheme">
<activity
android:name=".MainActivity">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
/activity>
<receiver android:name=".SMSReceiver"
android:exported="true"
android:permission="android.permission.BROADCAST_SMS">
<intent-filter android:priority="9000">
<action android:name="android.provider.Telephony.SMS_RECEIVED"/>
</intent-filter>
</receiver>
</application>
</manifest>
```

Step 2 : In the SRC folder of the project, add a new Class file to the package name and call it SMSReceiver (see Fig. 5.5).

Fig. 5.5: Addition of SMSReceiver Class

Step 3 : Code the SMSReceiver.java file as follows:

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.util.Log;
import android.widget.Toast;
public class SMSReceiver extends BroadcastReceiver
{
@Override
public void onReceive(Context context, Intent intent)
{
//---get the SMS message passed in---Bundle bundle =
intent.getExtras(); SmsMessage[] msgs = null;
String str = "SMS from "; if (bundle != null)
{
//---retrieve the SMS message received---
msgs = Telephony.Sms.Intents.getMessagesFromIntent(intent); for
(int i=0; i<msgs.length; i++){
str += msgs[i].getMessageBody().toString();
}
```

```
//---get the message body---
str += msgs[i].getMessageBody().toString();
}
//---display the new SMS message---Toast.makeText(context, str,
Toast.LENGTH_SHORT).show(); Log.d("SMSReceiver", str);
}
}
}
```

Step 4 : Press Shift+F9 to debug the application on the Android emulator.

Step 5 : Using the More setting on the emulator, select Phone and Send Message, send a message to the emulator. Your application should be able to receive the message and display it using the Toast class (see Fig. 5.6).

Fig. 5.6

- Besides sending SMS messages from your Android applications, you can also receive incoming SMS messages from within your applications by using a BroadcastReceiver object. This is useful when you want your application to perform an action when a certain SMS message is received.
- To receive SMS messages, use the onReceive() method of the BroadcastReceiver class. The Android framework sends out system broadcasts of events such as receiving an SMS message, containing intents that are meant to be received using a BroadcastReceiver.

Preventing the Messaging Application from Receiving a Message

- In the previous section, you might have noticed that every time you send an SMS message to the emulator (or device), both your application and the built-in application receive it. This is because when an SMS message is received, all applications (including the Messaging application) on the Android device take turns handling the incoming message.

- Sometimes, however, this is not the behavior you want. For example, you might want your application to receive the message and prevent it from being sent to other applications. This is very useful, especially if you are building some kind of tracking application.
- The solution is very simple. To prevent an incoming message from being handled by the built-in Messaging application, your application needs to handle the message before the Messaging app has the chance to do it. To do this, add the android:priority attribute to the <intent-filter> element, like this:

```
<receiver android:name=".SMSReceiver">
<intent-filter android:priority="100">
<action android:name = "android.provider.Telephony.SMS_RECEIVED"/>
</intent-filter>
</receiver>
```

- Set this attribute to a high number, such as 100. The higher the number, the earlier Android executes your application. When an incoming message is received, your application executes first, you can decide what to do with the message.
- To prevent other applications from seeing the message, simply call the abortBroadcast() method in your BroadcastReceiver class:

```
@Override
public void onReceive(Context context, Intent intent)
{
//---get the SMS message passed in---Bundle bundle =
intent.getExtras(); SmsMessage[] msgs = null;
String str = "SMS from "; if (bundle != null)
{
//---retrieve the SMS message received---
Object[] pdus = (Object[]) bundle.get("pdus");
msgs = new SmsMessage[pdus.length];
for (int i=0; i<msgs.length; i++){
msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]); if (i==0){
//---get the sender address/phone number---str +=
msgs[i].getOriginatingAddress();
str += ":";
}
//---get the message body---
str += msgs[i].getMessageBody().toString();
}
//---display the new SMS message---Toast.makeText(context, str,
Toast.LENGTH_SHORT).show(); Log.d("SMSReceiver", str);
//---stop the SMS message from being broadcasted---
this.abortBroadcast();
}
}
```

Updating an Activity from a BroadcastReceiver:

- The previous section demonstrates how you can use a BroadcastReceiver class to listen for incoming SMS messages and then use the Toast class to display the received SMS message. Often, you'll want to send the SMS message back to the main activity of your application.
- For example, you might want to display the message in a TextView.
- The following example demonstrates how you can do this. **Example:** Creating a View-Based Application Project.

Step 1 : Using the same project from the previous section, add the following bolded lines to the activity_ main.xml file. Be sure to replace all instances of com.jfdimarzio with the package used in your project:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_main"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.jfdimarzio.sms.MainActivity">
<Button
android:text="Send SMS" android:layout_width="wrap_content"
android:layout_height="wrap_content"android:id="@+id/btnSendSMS"
android:onClick="onClick"
app:layout_constraintLeft_toLeftOf="@+id/activity_main"
app:layout_constraintTop_toTopOf="@+id/activity_main"
android:layout_marginTop="16dp"
app:layout_constraintRight_toRightOf="@+id/activity_main"
app:layout_constraintBottom_toBottomOf="@+id/activity_main"
android:layout_marginBottom="16dp"/>
<TextView
android:text="TextView" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:id="@+id/textView"
app:layout_constraintLeft_toLeftOf="@+id/btnSendSMS"
app:layout_constraintTop_toBottomOf="@+id/btnSendSMS"
android:layout_marginTop="8dp"
app:layout_constraintRight_toRightOf="@+id/btnSendSMS"
app:layout_constraintBottom_toBottomOf="@+id/activity_main"androi
d:layout_marginBottom="16dp"/>
</android.support.constraint.ConstraintLayout>
```

Step 2 : Add the following bolded statements to the SMSReceiver.javafile:

```
import android.content.BroadcastReceiver; import
android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage; import android.util.Log;
import android.widget.Toast;
public class SMSReceiver extends BroadcastReceiver
{
@Override
public void onReceive(Context context, Intent intent)
{
//---get the SMS message passed in---Bundle bundle =
intent.getExtras(); SmsMessage[] msgs = null;
String str = "SMS from "; if (bundle != null)
{
//---retrieve the SMS message received---Object[] pdus =
(Object[]) bundle.get("pdus"); msgs = new
SmsMessage[pdus.length];
for (int i=0; i<msgs.length; i++){
msgs[i] =  SmsMessage.createFromPdu((byte[])pdus[i]);
if (i==0){
//---get the sender address/phone number---str +=
msgs[i].getOriginatingAddress();
str += ":";
}
//---get the message body---
str += msgs[i].getMessageBody().toString();
}
//---display the new SMS message---Toast.makeText(context, str,
Toast.LENGTH_SHORT).show(); Log.d("SMSReceiver", str); Intent
broadcastIntent =  new Intent();
broadcastIntent.setAction("SMS_RECEIVED_ACTION");
broadcastIntent.putExtra("sms", str);
context.sendBroadcast(broadcastIntent);
}
}
}
```

Step 3 : Add the following bolded statements to the MainActivity.java file:

```
import android.Manifest;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.pm.PackageManager;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;
import static android.Manifest.permission_group.SMS;
public class MainActivity extends AppCompatActivity
{
final private int REQUEST_SEND_SMS = 123;
final private int REQUEST_REC_SMS = 321;
BroadcastReceiver smsSentReceiver;
IntentFilter intentFilter;
private BroadcastReceiver intentReceiver = new
BroadcastReceiver() {
@Override
public void onReceive(Context context, Intent intent){
//---display the SMS received in the TextView---
TextView SMSes = (TextView) findViewById(R.id.textView);
SMSes.setText(intent.getExtras().getString("sms"));
}
};
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
if (ContextCompat.checkSelfPermission(this,
Manifest.permission.SEND_SMS)
```

```
!= PackageManager.PERMISSION_GRANTED ) {
ActivityCompat.requestPermissions(this,
new String[]{Manifest.permission.SEND_SMS}, REQUEST_SEND_SMS);
}
if (ContextCompat.checkSelfPermission(this,
Manifest.permission.RECEIVE_SMS)
!= PackageManager.PERMISSION_GRANTED ) {
ActivityCompat.requestPermissions(this,
new String[]{Manifest.permission.RECEIVE_SMS}, REQUEST_REC_SMS);
}
intentFilter = new IntentFilter();
intentFilter.addAction("SMS_RECEIVED_ACTION");
}
@Override
public void onResume() { super.onResume();
//---register the receiver---
registerReceiver(intentReceiver,intentFilter);
}
@Override
public void onPause() { super.onPause();
//---unregister the receiver---
unregisterReceiver(intentReceiver);
}
@Override
public void onRequestPermissionsResult(int requestCode, String[]
permissions, int[] grantResults){
switch (requestCode) { case REQUEST_SEND_SMS:
if (grantResults[0] == PackageManager.PERMISSION_GRANTED)
{ Toast.makeText(MainActivity.this,
"SEND Permission Granted", Toast.LENGTH_SHORT).show();
} else {
Toast.makeText(MainActivity.this,
"SEND Permission Denied", Toast.LENGTH_SHORT).show();
}
break;
case REQUEST_REC_SMS:
if (grantResults[0] == PackageManager.PERMISSION_GRANTED)
```

```
{ Toast.makeText(MainActivity.this,
"RECEIVE Permission Granted", Toast.LENGTH_SHORT).show();
} else {
Toast.makeText(MainActivity.this,
"RECEIVE Permission Denied", Toast.LENGTH_SHORT).show();
}
break;default:
super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
}
}
public void onClick(View v){ sendSMS("5554", "Hello my
friends!");
}
//---sends an SMS message to another device---
private void sendSMS(String phoneNumber, String message)
{
SmsManager sms = SmsManager.getDefault();
sms.sendTextMessage(phoneNumber, null, message, null, null);
}
}
```

Step 4 : Press Shift+F9 to debug the application on the Android emulator. Using the DDMS, send an SMS message to the emulator. Fig. 5.7 shows the Toast class displaying the message received, and the TextView showing the message received.

Fig. 5.7

5.6.5 Caveats and Warnings

- Although the capability to send and receive SMS messages makes Android a very compelling platform for developing sophisticated applications, this flexibility comes with a price.
- A seemingly innocent application might send SMS messages behind the scene without the user knowing, as demonstrated by a recent case of an SMS-based Trojan Android application.
- The app claims to be a media player, but when it's installed it sends SMS messages to a premium-rate number, resulting in huge phone bills for the user.
- The user needs to explicitly give permissions (such as accessing the Internet, sending and receiving SMS messages, and so on) to your application; however, the request for permissions is shown only at installation time.
- If the user clicks the Install button, he or she is considered to have granted the application permission to send and receive SMS messages. This is dangerous because after the application is installed it can send and receive SMS messages without ever prompting the user again.
- In addition to this, the application also can "sniff" for incoming SMS messages. For example, based on the techniques you learned from the previous section, you can easily write an application that checks for certain keywords in the SMS message.
- When an SMS message contains the keyword you are looking for, you can then use the Location Manager to obtain your geographical location and then send the coordinates back to the sender of the SMS message. The sender could then easily track your location.

5.7 SENDING E-MAIL

- All these tasks can be done easily without the user knowing it! That said, users should try to avoid installing Android applications that come from uncertain sources, such as from unknown websites or strangers.
- Like SMS messaging, Android also supports e-mail. The Gmail/Email application on Android enables you to configure an email account using POP3 or IMAP.
- Besides sending and receiving emails using the Gmail/Email application, you can also send e-mail messages programmatically from within your Android application.
- For the following example to work properly, you must configure the Email app on your emulator. Simply click the Email app on the emulator and follow the on-screen prompts to set up the application. If you do not, you receive a message stating that no application is configured to handle the E-mail intent.

Step 1 : Using Android Studio, create a new Android project and name it **E-MAILS**.

Step 2 : Add the following bolded statements to the activity_main.xml file, replacing the TextView. Please be sure to replace all instances of com.jfdimarzio with the package used in your project:

```
<?xml version="1.0"encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android= "http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_main"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.jfdimarzio.emails.MainActivity">
<Button
android:text="Send Email"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/btnSendEmail"
app:layout_constraintLeft_toLeftOf="@+id/activity_main"
app:layout_constraintTop_toTopOf="@+id/activity_main"
app:layout_constraintRight_toRightOf="@+id/activity_main"
app:layout_constraintBottom_toBottomOf="@+id/activity_main"/>
</android.support.constraint.ConstraintLayout>
```

Step 3 : Add the following bolded statements to the MainActivity.java file:

```
import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
public class MainActivity extends AppCompatActivity{
@Override
protected void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
public void onClick(View v) {
//---replace the following email addresses with real ones---
String[] to =
{"someguy@example.com", "anotherguy@example.com"}; String[] cc =
{"busybody@example.com"};
sendEmail(to, cc, "Hello", "Hello my friends!");
}
```

```
private void sendEmail(String[] emailAddresses, String[]
carbonCopies, String subject, String message)
{
Intent emailIntent = new Intent(Intent.ACTION_SEND);
emailIntent.setData(Uri.parse("mailto:")); String[] to =
emailAddresses; String[] cc = carbonCopies;
emailIntent.putExtra(Intent.EXTRA_EMAIL, to);
emailIntent.putExtra(Intent.EXTRA_CC, cc);
emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
emailIntent.putExtra(Intent.EXTRA_TEXT, message);
emailIntent.setType("message/rfc822");
startActivity(Intent.createChooser(emailIntent, "Email"));
}
}
```

Step 4 : Press Shift+F9 to test the application on the Android emulator/device (ensure that you have configured your email before trying this example).

Step 5 : Click the Send Email button. If you configured the Email service on your emulator, you should see the Email application launched in your emulator/device. Otherwise you see the message shown in Fig. 5.8.

Fig. 5.8

Summary

- SQLite is an open source SQL database that stores data to a text file on a device. Android comes in with built-in SQLite database implementation.
- In the mobile platform, SQLite is a very popular choice across various platforms because of its light weight nature. Apple uses it in the iPhone and Google in the Android operating system.

- User may search/request for particular data within a database by utilizing what is known as a query. In Android a helper class to manage database creation and version management.
- A SQLite statement is written in SQL, which is issued to a database to retrieve data or to create, insert, update, or delete data in the database.
- The SQLiteOpenHelper is responsible for opening database if exist, creating database if it does not exists and upgrading if required.
- The SQLiteOpenHelper only require the DATABASE_NAME to create database. After extending SQLiteOpenHelper you will need to implement its methods onCreate, onUpgrade and constructor.
- The CREATE TABLE command is used to create a new table in a SQLite database.
- For performing any database operation, you have to provide the implementation of onCreate() and onUpgrade() methods of SQLiteOpenHelperclass.
- SQLiteDatabase contains methods to be performed on sqlite database such as create, insert, update, delete, etc.
- In Android, we can insert data into SQLite database by using insert() method.
- In Android, we can read the data from SQLite database using query() method.
- In Android, we can update the data in SQLite database using update() method.
- Cursor objects are like file pointers; they allow random access to query results.
- A Cursor object depicts the result of a query and fundamentally points to one row of the result of the query.
- SMS stands for Short Message Service. SMS is used to send text messages to mobile phones. SMS messaging is one of the main popular applications on a mobile phone today.
- In android, we can easily send an email from our android application using existing email clients such as GMAIL, Outlook, etc.
- Android smartphones can send and receive messages to or from any other phone that supports Short Message Service (SMS).
- To send an SMS message programmatically, you use the SmsManager class.
- You can use Android Intent to send SMS by calling built-in SMS functionality of the Android and use ACTION_VIEW action to launch SMS client installed on your Android device.
- Like SMS messaging, Android also supports email. The Gmail/Email application on Android enables you to configure an email account using POP3 or IMAP.
- To listen for incoming SMS messages, you create a BroadcastReceiver class.
- The BroadcastReceiver class enables your application to receive intents sent by other applications using the sendBroadcast() method.

Practice Questions

Q.I Multiple Choice Questions:

1. _____ is used to perform database operations on Android such as storing, manipulating or retrieving persistent data from the database.
   (a) MySQL (b) SQL
   (c) SQLite (d) All of the Mentioned
2. _____ class provides the functionality to use the SQLite database.
   (a) SQLiteOpenHelper (b) SQLiteCloseHelper
   (c) SQLiteFetchHelper (d) All of the Mentioned
3. The method creates a new database if the required database is not present.
   (a) onUpgrade() (b) onCreateTable()
   (c) onCreate() (d) None of the Mentioned
4. In android, we can read the data from SQLite database using _____ method.
   (a) insert() (b) query()
   (c) update() (d) All of the Mentioned
5. A _____ represents the entire result set of the query.
   (a) Cursor (b) Trigger
   (c) Function (d) None of the Mentioned
6. CRUD stands for _____.
   (a) Create, Read (b) Update, Delete
   (c) None of the Mentioned (d) Both (a) and (b)
7. Cursor navigation methods includes _____.
   (a) moveToFirst (b) moveToNext
   (c) Both (a) and (b) (c) None of the Mentioned
8. _____ is a text messaging service in Android mobile-device.
   (a) SMS (b) MMS
   (c) Both (a) and (b) (d) None of Mentioned
9. To send an SMS message programmatically, you use the _____ class.
   (a) MmsManager (b) SmsManager
   (c) Both (a) and (b) (d) None of Mentioned
10. Receiving SMS messages implement a _____ method.
    (a) BroadcastSender (b) BroadcastOnReceiver
    (c) BroadcastReceiver (d) None of the Mentioned
11. When an incoming SMS message is received, the _____ method is fired.
    (a) onReceive() (b) onBroadcast()
    (c) onSend() (d) None of the Mentioned

12. _____ is an activity action which specifies that we are sending some data.
    (a) ACTION_RECEIVE (b) ACTION_SEND
    (c) ACTION_BROADCAST (d) All of the Mentioned

Answers

1. (c)	2. (a)	3. (c)	4. (b)	5. (a)	6. (d)	7. (c)	8. (a)	9. (b)	10. (c)
11. (a)	12. (b)								

Q.II Answer the following Questions in short:

1. What is meant by database?
2. What is SQLite?
3. Enlist feature of SQLite.
4. What is SMS?
5. What is email?
6. What is meant by SMS messaging?
7. How to close database using SQLite?

Q.III Answer the following Questions:

1. How to create database in SQLite? Explain with example.
2. How to update record in SQLite?
3. What is cursor in SQLite?
4. How to delete a record in SQLite? Explain with example.
5. How to sending email? Explain with example.
6. Name the two ways in which you can send SMS messages in your Android application.
7. How to send messages programmatically?
8. Describe how to send email in Android using Intent.

Q.IV Short note:

1. SQLiteDatabase
2. SQLiteOpenHelper
3. Caveats and Warnings
4. Receiving SMS messages

■■■

6... Location-Based Services and Google Map

Learning Objectives ...

Students will be able to:

- To learn Basic Concepts of Google Map.
- To study Displaying Map, Changing Views of Map, Adding Markers on Map etc.

6.1 DISPLAY GOOGLE MAPS

- You have seen the explosive growth of mobile apps in recent years. One category of apps that is very popular is Location-Based Services, commonly known as LBS. LBS apps track your location, and might offer additional services such as locating amenities nearby, offering suggestions for route planning, and soon. Of course, one of the key ingredients in an LBS app is maps, which present a visual representation of your location.
- This chapter shows you how to make use of Google Maps in your Android application, as well as how to manipulate the map view programmatically. In addition, you find out how to obtain your geographical location using the LocationManager class available in the Android SDK.
- Google Maps is one of the many applications bundled with the Android platform. In addition to simply using the Maps application, you can also embed it into your own applications and make it do some very cool things.
- This section describes how to use Google Maps in your Android applications and programmatically perform the following:
  1. Change the views of Google Maps.
  2. Obtain the latitude and longitude of locations in Google Maps.
  3. Perform geocoding and reverse geocoding (translating an address to latitude and longitude and vice versa).

6.1.1 Creating the Project

- To get started, you need to first create an Android project so that you can display Google Maps in your activity.
- Using Android Studio, create an Android project and name it LBS.
- From the Create New Project Wizard, select Google Maps Activity as shown in Fig. 6.1.

Fig. 6.1: Project Wizard Screen

6.1.2 Obtaining the Maps API Key

- Beginning with the Android SDK release v1.0, you need to apply for a free Google Maps API key before you can integrate Google Maps into your Android application.
- When you apply for the key, you must also agree to Google's terms of use, so be sure to read them carefully.
- To get a Google Maps key, open the google_maps_api.xml file that was created in your LBS project. Within this file is a link to create a new Google Maps key.
- Simply copy and paste the link into your browser and follow the instructions. Make note of the key that Google gives you because you need it later in this project.

6.1.3 Displaying the Map

- The following example shows you how to display Google Maps in Android application.

Example 1: Displaying Google Maps.

Step 1 : Using the project created in the previous section, replace the YOUR_KEY_HERE placeholder in the google_maps_api.xml with your Google Maps key.

Step 2 : Press Shift+F9 to debug the application on the Android emulator. Fig. 6.2 shows Google Maps displayed in the application's activity. (Keep in mind, depending on the specs of your computer, this example might take a while to load the first time you run it.)

Fig. 6.2: Google Map

6.1.4 Displaying the Zoom Control

- The previous section showed how you can display Google Maps in your Android application. You can pan the map to any desired location and it updates on-the-fly.
- However, there is no way to use the emulator to zoom in or out from a particular location (on a real Android device you can pinch the map to zoom it).
- Thus, in this section, you find out how you can enable users to zoom in or out of the map using the built-in zoom controls.

Step 1 : Using the project created in the previous section, add the following bolded statement to activity_maps.xml. Please be sure to replace all instances of com.jfdimarzio with a reference to the package used in your application:

```
<fragment
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
xmlns:map="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:id="@+id/map"
tools:context="com.jfdimarzio.locationservices.MapsActivity"
android:name="com.google.android.gms.maps.SupportMapFragment"
map:uiZoomControls="true"
/>
```

Step 2 : Press Shift+F9 to debug the application on the Android emulator. Observe the built-in zoom controls that appear at the bottom of the map when you click and drag the map (see Fig. 6.3). You can click the minus (–) icon to zoom out of the map, and the plus (+) icon to zoom in to the map.

Fig. 6.3: Zoom Controls on Google Map

Programmatically Zooming In or Out of the Map:

Step 1 : Using the project created in the previous section, add the following bolded statements to the

MapsActivity.javafile:

```
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;
import android.view.KeyEvent;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback {
private GoogleMap mMap;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_maps);
```

```
//Obtain the SupportMapFragment and get notified
//when the map is ready to be used.
SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
mapFragment.getMapAsync(this);
}
@Override
public void onMapReady(GoogleMap googleMap) {
mMap = googleMap;
//Add a marker in Sydney and move the camera LatLng sydney = new
LatLng(-34, 151);
mMap.addMarker(new MarkerOptions().position(sydney).title("Marker
in Sydney"));
mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
}
public boolean onKeyDown(int keyCode, KeyEvent event)
{
Switch (keyCode)
{
case KeyEvent.KEYCODE_3:
mMap.animateCamera(CameraUpdateFactory.zoomIn());
break;
case KeyEvent.KEYCODE_1:
mMap.animateCamera(CameraUpdateFactory.zoomOut());
break;
}
return super.onKeyDown(keyCode, event);
}
}
```

Step 2 : Press Shift+F9 to debug the application on the Android emulator. You can now zoom into the map by clicking the numeric 3 key on the emulator. To zoom out of the map, click the numeric 1 key.

6.1.5 Changing Views

- By default, Google Maps is displayed in map view, which is basically drawings of streets and places of interest.

- You can also set Google Maps to display in satellite view using the setMapType() method of the Google Map class.

```
public void onMapReady(GoogleMap googleMap)
{
    mMap = googleMap;
    //Add a marker in Sydney and move the camera LatLng sydney = new
    LatLng(-34, 151); mMap.addMarker(newMarkerOptions().position(sydney).
    title("Marker in Sydney"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
```

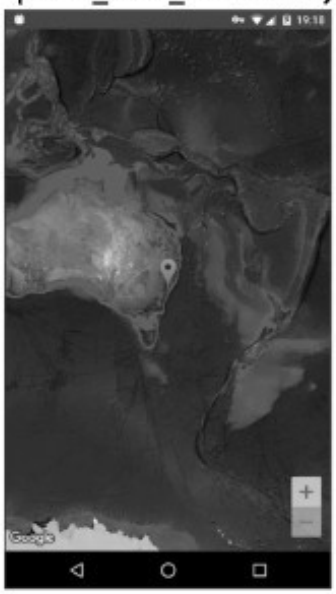


Fig. 6.4: Google Maps Displayed in Satellite View

6.1.6 Navigating to a Specific Location

- By default, Google Maps displays the map of Australia when it is first loaded. However, you can set Google Maps to display a particular location.
- To do so, you can use the moveCamera() method of the Google Map class.

Step 1 : Using the project created in the previous section, change the following bolded statements to the MapsActivity.javafile:

```
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
```

```
public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback {
private GoogleMap mMap;
@Override
protected void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
setContentView(R.layout.activity_maps);
//Obtain the SupportMapFragment and get notified
//when the map is ready to be used.
SupportMapFragment mapFragment=
(SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
mapFragment.getMapAsync(this);
}
@Override
public void onMapReady(GoogleMap googleMap) {
mMap =googleMap;
LatLng boston = new LatLng(42.3601, -71.0589);
mMap.addMarker(newMarkerOptions().position(boston).
title("Boston, Mass"));
mMap.moveCamera(CameraUpdateFactory.newLatLng(boston));
}
}
```

Step 2 : Press Shift+F9 to debug the application on the Android emulator. When the map is loaded, observe that it now animates to a particular location in Boston, Massachusetts (see Fig. 6.5).

Fig. 6.5: A particular location on Google Map

6.1.7 Adding Markers

- Markers indicate single locations on the map. The purpose of the Marker class is to allow locations to be marked on a map.
- Markers are added to a map by obtaining a reference to the Google Map object associated with a map and then making a call to the addMarker() method of that object instance.
- You can customize your markers by changing the default color, or replacing the marker icon with a custom image. The default marker uses a standard icon, common to the Google Maps look and feel.
- The position of a marker is defined via Longitude and Latitude. Markers can be configured in a number of ways, including specifying a title, text and an icon. Markers may also be made to be "draggable", allowing the user to move the marker to different positions on a map.
- Adding markers to a map to indicate places of interest enables the users to easily locate the places they are searching. The following coding shows how to add a marker to Google Maps.

Adding Markers to the Map:

Step 1 : Create a (ill image containing a pushpin and copy it into the residrawable-mdpi folder of the project.

For most effective, we make the background of the image transparent hence it does not block parts of the map when the image is added to the map.

Step 2 : Add the following codes in bold to the MainActivity.java file in previous activity:

```
package com.pkg.LBS;
import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Point;
import com.google.androld.maps.Overlay;
import java.util.List;
```

```
public class MainActivity extends MapActivity (
MapView mapView;
MapController mc;
GeoPaint p;
class MapOverlay extends com_google.android.maps.Overlay
{
@Override
public boolean draw(Canvas canvas, MapView mapView, boolean
shadow, long when)
{
super.draw(canvas, mapView, shadow);
//---translate the GeoPoint to screen pixels---Point screenPts =
new Point();
mapView.getProjection().toPixels(p, screenPts);
//---add the marker--
Bitmap bmp=BitmapFactory.decodeResource(getResources(),
R.drawable. pushpin);
canvas.drawBitmap(omp, screenPts.x, screenPts.y-50, null);
return true;
}
}
/** Called when the activity is first created.
@Override
public void onCreate(Bundle savedInstanceState)
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
mapView = (MdpView) lindViewByld(R.id.mapView);
mapView.setBuiltInZoomControls(true);//mapView.setSatellite(true)
;
//mapView.setStreetView(true);

me = mapView.getController();
String coordinates[] = {"1.352566007", "103,789215871"};
double lat = Double.parseDouble(coordinates[0]);
double lng = Double.parseDouble(coordinates[1]);
p = new GeoPoint((int) (lat * 1E6), (int) (lng * 1E6));
mc.animateTo(p);
mc.setZoom(13);
```

```
//---Add a location marker--
MapOverlay mapOverlay =. new MapOverlay();
List<Overlay> listOfOverlays = mapView.getOverlays();
listiClOverlays.clear();
listofOverlays.add(mapOverlay);
mapview.invalidate();
public boolean onKeyDown(int keyCode, KeyEvent event)
MapController me = mapView.getController();
switch (keyCode)
{
case KeyEvent.KEYCODE_1:
mc.zoomIn();break;
case KeyEvent.KEYCODE_2:
mc.zoomOut();break;
}
return super.onKeyDown(keyCode, event);
}
@Override
protected Boolean isRouteDisplayed(){
// TODO Auto-generated method stub
return false;
}
}
```

Step 3 : Press F11 to debug the application on the Android Emulator.

Mechanism:

- To add a marker to the map, we required to define a class that extends the Overlay class:

```
class MapOverlay extends com.google.android.maps.Overlay
{
    @Override
    publicbooleandraw(Canvascanvas,MapViewmapView,booleanshadow,longwhen)
    {
        //…
    }
}
```

Fig. 6.6: Added Markers to Google Map

6.1.8 Getting the Location that was Touched

- After using Google Maps for a while, you might want to know the latitude and longitude of a location corresponding to the position on the screen that was just touched. Knowing this information is very useful because you can determine a location's address a process known as reverse geocoding (you find out how this is done in the next section).
- To get the latitude and longitude of a point on the Google Map that was touched, you must set a onMapClickListener:

```
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;
import android.util.Log;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback {
private GoogleMap mMap;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_maps);
```

```
//Obtain the SupportMapFragment and get notified
//when the map is ready to be used.
SupportMapFragment mapFragment =
(SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
mapFragment.getMapAsync(this);
}
@Override
public void onMapReady(GoogleMap googleMap) {
mMap =googleMap;
LatLng boston = new LatLng(42.3601, -71.0589);
mMap.addMarker(new          MarkerOptions().position(boston).title("Boston,
Mass"));
mMap.moveCamera(CameraUpdateFactory.newLatLng(boston));
mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
@Override
public void onMapClick(LatLng point) {
Log.d("DEBUG","Mapclicked[" + point.latitude + "/" + point.longitude +
"]");
}
});
}
}
```

- You should see a logcat entry similar to this if you run the preceding code:

```
D/DEBUG:Map clicked[37.15198779979302/-83.76536171883345]
```

6.1.9 Geocoding and Reverse Geocoding

- As mentioned in the preceding section, if you know the latitude and longitude of a location, you can find out its address using a process known as reverse geocoding.
- Google Maps in Android supports reverse geocoding via the Geocoder class.
- The following code snippet shows how you can retrieve the address of a location just touched using the getFromLocation() method:

```
import android.location.Address;
import android.location.Geocoder;
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;
import android.widget.Toast;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
```

```
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
import java.io.IOException;
import java.util.List;
import java.util.Locale;
public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback {
private GoogleMap mMap;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_maps);
//Obtain the SupportMapFragment and get notified
//when the map is ready to be used.
SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
mapFragment.getMapAsync(this);
}
@Override
public void onMapReady(GoogleMap googleMap) {
mMap = googleMap;
LatLng boston = new LatLng(42.3601, -71.0589);
mMap.addMarker(new MarkerOptions().position(boston).title("Boston,
Mass"));
mMap.moveCamera(CameraUpdateFactory.newLatLng(boston));
mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
@Override
public void onMapClick(LatLng point) {
Geocoder geoCoder = new Geocoder( getBaseContext(), Locale.getDefault());
try{
List<Address> addresses = geoCoder.getFromLocation(
point.latitude,point.longitude,1);
String add ="";
If (addresses.size() > 0)
{
for (int i=0; i<addresses.get(0).getMaxAddressLineIndex(); i++)
add += addresses.get(0).getAddressLine(i) + "\n";
}
Toast.makeText(getBaseContext(), add, Toast.LENGTH_SHORT).show();
}
Catch (IOException e){ e.printStackTrace();
}
}
});
}
}
```

- The Geocoder object converts the latitude and longitude into an address using the getFromLocation() method. After the address is obtained, you display it using the Toast class. Keep in mind that pin will not move. In this example, we are only getting the address of a location that you touch.
- Fig. 6.7 shows the application displaying the address of a location that was touched on the map.

Fig. 6.7: Displayed the Address of a Location on Google Map

6.2 GETTING LOCATION DATA

- Nowadays, mobile devices are commonly equipped with GPS receivers. Because of the many satellites orbiting the earth, you can use a GPS receiver to find your location easily. However, GPS (Global Positioning System) requires a clear sky to work and hence does not always work indoors or where satellites can't penetrate (such as a tunnel through a mountain).
- Another effective way to locate your position is through cell tower triangulation. When a mobile phone is switched on, it is constantly in contact with base stations surrounding it.
- By knowing the identity of cell towers, it is possible to translate this information into a physical location through the use of various databases containing the cell towers' identities and their exact geographical locations.
- The advantage of cell tower triangulation is that it works indoors, without the need to obtain information from satellites.
- However, it is not as precise as GPS because its accuracy depends on overlapping signal coverage, which varies quite a bit. Cell tower triangulation works best in densely populated areas where the cell towers are closely located.

Example 2: Navigating the Map to a Specific Location.

Step 1 : Using the same project created in the previous section, add the following bolded statements to the MapsActivity.java file:

```
import android.content.Context;
import android.content.pm.PackageManager;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.support.v4.app.ActivityCompat;
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;
import android.widget.Toast;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
public class MapsActivity extends FragmentActivity
implements OnMapReadyCallback {
final private int REQUEST_COURSE_ACCESS = 123;
boolean permissionGranted = false;
private GoogleMap mMap;
LocationManager lm;
LocationListener locationListener;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_maps);
//Obtain the SupportMapFragment and get notified
//when the map is ready to be used.
SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
mapFragment.getMapAsync(this);
}
```

```
@Override
public void onPause() {
super.onPause();
//---remove the location listener---
if (ActivityCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_FINE_LOCATION)
!= PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_COARSE_LOCATION)
!= PackageManager.PERMISSION_GRANTED) {
ActivityCompat.requestPermissions(this, new String[]{
android.Manifest.permission.ACCESS_COARSE_LOCATION},
REQUEST_COURSE_ACCESS);
return;
}else{
permissionGranted = true;
}
if(permissionGranted) {
lm.removeUpdates(locationListener);
}
}
@Override
public void onMapReady(GoogleMap googleMap) {
mMap = googleMap;
lm = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
locationListener = new MyLocationListener();
if (ActivityCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_FINE_LOCATION)
!= PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_COARSE_LOCATION)
!= PackageManager.PERMISSION_GRANTED) {
ActivityCompat.requestPermissions(this,
new String[]{
android.Manifest.permission.ACCESS_COARSE_LOCATION},
REQUEST_COURSE_ACCESS);
return;
}else{
permissionGranted = true;
}
```

```
if(permissionGranted) {
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
locationListener);
}
}
@Override
public void onRequestPermissionsResult(int requestCode, String[]
permissions, int[]grantResults) {
switch(requestCode){
case REQUEST_COURSE_ACCESS:
if(grantResults[0] == PackageManager.PERMISSION_GRANTED) {
permissionGranted = true;
} else {
permissionGranted = false;
}
break;default:
super.onRequestPermissionsResult(requestCode, permissions,
}
}
grantResults);
private class MyLocationListener implements LocationListener
{
public void onLocationChanged(Location loc) {
if(loc != null)
{
Toast.makeText(getBaseContext(),
"Location changed : Lat: "+ loc.getLatitude() + Lng:"
+ loc.getLongitude(), Toast.LENGTH_SHORT).show();
LatLng p = new LatLng(
(int) (loc.getLatitude()),
(int) (loc.getLongitude()));
mMap.moveCamera(CameraUpdateFactory.newLatLng(p));
mMap.animateCamera(CameraUpdateFactory.zoomTo(7));
}
}
```

```
public void onProviderDisabled(String provider) {
}
public void onProviderEnabled(String provider) {
}
public void onStatusChanged(String provider, int status, Bundle
extras) {
}
}
}
```

Step 2 : Add the following bolded line to the AndroidManifest.xml file. Please be sure to replace all instances of com.jfdimarzio with a reference to the package used in your application:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.jfdimarzio.locationservices">
<!--
The ACCESS_COARSE/FINE_LOCATION permissions are not required to
use Google Maps Android API v2, but you must specify either
coarse or fine location permissions for the 'MyLocation'
functionality.
-->
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:supportsRtl="true"
android:theme="@style/AppTheme">
<!--
The API key for Google Maps-based APIs is defined as a string
resource (See the file "res/values/google_maps_api.xml").
Note that the API key is linked to the encryption key used to
sign the APK.
```

```
You need a different API key for each encryption key, including
the release key that is used to sign the APK for publishing.
You can define the keys for the debug and release targets in
src/debug/ and src/release/.
-->
<meta-data
android:name="com.google.android.geo.API_KEY"
android:value="@string/google_maps_key"/>
<activity
android:name=".MapsActivity"
android:label="@string/title_activity_maps">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
</application>
</manifest>
```

Step 3 : Press Shift+F9 to debug the application on the Android emulator.

Step 4 : To simulate GPS data received by the Android emulator, you use the Location Controls tool on the right-hand side of the emulator.

Step 5 : Observe that the map on the emulator now animates to another location (see Fig. 6.8). This proves that the application has received the GPS data.

Fig. 6.8: Map Animated to another Location

6.3 MONITORING A LOCATION

- One very cool feature of the LocationManager class is its ability to monitor a specific location. This is achieved using the addProximityAlert() method.
- The following code snippet shows how to monitor a particular location such that if the user is within a five-meter radius from that location, your application will fire an intent to launch the web browser:

```
import android.app.PendingIntent;
import android.content.Intent;
import android.net.Uri;
//---use the LocationManager class to obtain locations data---lm
= (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
//---PendingIntent to launch activity if the user is within
//some locations---
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, new
Intent(android.content.Intent.ACTION_VIEW,Uri.parse("http://www.amazon.com
")), 0);
lm.addProximityAlert(37.422006, -122.084095, 5, -1, pendingIntent);
```

- The addProximityAlert() method takes five arguments namely: Latitude, Longitude, Radius (in meters), Expiration (duration for which the proximity alert is valid, after which it is deleted; –1 for no expiration, and Pending intent.
- Note that if the Android device's screen goes to sleep, the proximity is also checked once every four minutes in order to preserve the battery life of the device.

Summary

- Google Maps is one of the many applications bundled with the Android platform. Android allows to integrate Google maps in the application.
- A Marker is an icon placed at a particular point on the map's surface. A marker icon is drawn oriented against the device's screen rather than the map's surface; i.e., it will not necessarily change orientation due to map rotations, tilting, or zooming.
- Markers indicate single locations on the map. You can customize your markers by changing the default color, or replacing the marker icon with a custom image. Markers are added to the map with the addMarker() method.
- Using the Geocoder class in the Android framework location APIs, you can convert an address to the corresponding geographic coordinates. This process is called geocoding.
- Alternatively, you can convert a geographic location to an address. The address lookup feature is also known as reverse geocoding.

- The getFromLocation() method to convert a geographic location to an address. The method returns an estimated street address corresponding to a given latitude and longitude.
- The location data available to an Android device includes the current location of the device – pinpointed using a combination of technologies the direction and method of movement, and whether the device has moved across a predefined geographical boundary.
- Google Map is a mapping and navigation application for desktop and mobile devices from Google. Google Map is the main class of the Google Maps Android API.
- Google Maps is displayed in map view, which is basically drawings of streets and places of interest by default. We can also set Google Maps to display in satellite view using the setSatellite() method of the Map View class.
- When developing a location aware application for Android, you can utilize GPS and Android's Network Location Provider to acquire the user location.
- On a map, the technique used to display way markers, the current location and other points of interest, is Overlays. To add an overlay to the map, we have to override the onTouchEvent() method within the MapOverlay class. This method is fired every time the user touches the map.
- Using the MotionEvent parameter, you can determine whether the user has lifted a finger from the screen using the getAction() method.
- There is another feature of the LocationManager class is its ability to monitor a specific location. This is achieved by using the addProximityAlert() method.

Practice Questions

Q.I Multiple Choice Questions:

1. _____ is a Web-based service that provides detailed information about geographical regions and sites around the world.
   (a) Google Maps (b) Apple Maps
   (c) Here Maps (d) All of the Mentioned
2. _____ is the process of finding the geographical coordinates (latitude and longitude) of given address or location.
   (a) Reverse Geocoding (b) Geocoding
   (c) Both (a) and (b) (d) All of the Mentioned
3. _____ identifies locations on the map and used to notify the user of locations on a map and take the form of either a standard or custom icon.
   (a) Maps (b) API Keys
   (c) Markers (d) None of the Mentioned

4. To add an overlay to the map, we have to override the _____ method within the MapOverlay class and method is fired every time the user touches the map.
   (a) addMarker() (b) onTouchEvent()
   (c) addProximityAlert() (d) None of the Mentioned
5. The feature of the LocationManager class is its ability to monitor a specific location and this is achieved by using themethod.
   (a) getFromLocation() (b) onTouchEvent()
   (c) addProximityAlert() (d) None of the Mentioned

Answers

1. (a)	2. (b)	3. (c)	4. (b)	5. (c)

Q.II Answer the following Questions in short:

1. What is meant by Google map?
2. How to create Google map?
3. What is marker? How to add markers in Google map.

Q.III Answer the following Questions:

1. Describe the term navigating to a specific location.
2. With the help of example describe getting location data.
3. Explain the term displaying Google map in detail.
4. How to changing views in Google maps? Describe with example.

Q.IV Short notes

1. Geocoding and Reverse Geocoding.
2. Monitoring a location in Google map.

■■■