MOBILE APPLICATION DEVELOPMENT

[As per New NEP Syllabus for 6th Semester BCA, Bengaluru City University, Bangalore University and Other Universities in Karnataka w.e.f 2021-22]

Dr. N. Kartik MCA, M.Tech, K-SET, Ph.D
Associate Prof., Department of Computer Application,
Presidency College (Autonomous), Bengaluru.

Hemalatha P. MCA
Asst. Professor, Bachelor of Computer Applications,
Benedictine Academy, Bengaluru.

Roopa H. R. MCA (Ph.D)
Asst. Professor in Dept. of Computer Science,
KLE Society's S Nijalingappa College, Bengaluru.

Vision book house
NEW DELHI • BANGALORE
(An Imprint of Himalaya Publishing House Pvt. Ltd.)

First Edition: 2024

Published by : Mrs. Meena Pandey
for HIMALAYA PUBLISHING HOUSE PVT. LTD.,
"Ramdoot", Dr. Bhalerao Marg, Girgaon, Mumbai - 400 004.
Phone: 022-23860170, 23863863; Fax: 022-23877178
E-mail: himpub@vsnl.com; Website: www.himpub.com

(Vision Book House, an Imprint of Himalaya Publishing House Pvt. Ltd.)

Branch Offices :

New Delhi : "Pooja Apartments", 4-B, Murari Lal Street, Ansari Road, Darya Ganj, New Delhi - 110 002.
Phone: 011-23270392, 23278631; Fax: 011-23256286

Nagpur : Kundanlal Chandak Industrial Estate, Ghat Road, Nagpur - 440 018.
Phone: 0712-2721216, 2721216

Bengaluru : Plot No. 91-33, 2nd Main Road, Seshadripuram, Behind Nataraja Theatre,
Bengaluru - 560 020. Phone: 080-41138821; Mobile: 09379847017, 09379847005

Hyderabad : No. 3-4-184, Lingampally, Besides Raghavendra Swamy Matham, Kachiguda,
Hyderabad - 500 027. Phone: 040-27560041, 27550139

Chennai : No. 34/4, Motilal Street, T. Nagar, Chennai - 600 017. Mobile: 09380460419

Pune : "Laxmi" Apartment, First Floor, No. 527, Mahurpura,
Shaniwarpeth (Near Prabhat Theatre), Pune - 411 030.
Phone: 020-24496323, 24496333; Mobile: 09370579333

Cuttack : Plot No. SF-7594, Sector-9, CDA Market Nagar, Cuttack - 753 014,
Odisha. Mobile: 09338746007

Kolkata : 3, S.M. Bose Road, Near Gate No. 5, Agarpara Railway Station,
North 24 Parganas, West Bengal - 700 109. Mobile: 09674536325

DTP by : SPS, Bengaluru.

Printed at : M/s. Girraj Printers, Hyderabad On behalf of HPH.

Preface

We have great pleasure in presenting First edition *"Mobile Application Development"* written for students of UG courses. The related matters are written in a simple and easily understandable.

This volume is an attempt to provide the students with thorough understanding of Mobile Application Development. We have presented the subject matter in a systematic manner with liberal use of charts and diagrams where ever necessary so as to make it interesting and sustain students' interest. We thank Almighty for showering his substantial blessings and giving us the determination in preparing this book.

In writing this book we have benefitted immensely from the studies of a number of books and the articles written by scholars spread over various books, journals and magazine. We are grateful to them.

We are sure this book will prove extremely useful to students and teachers alike. This book would not have seen the light, but for the grace of God and the blessings and support of our family members and friends.

We are quite confident, though, that the book can also be adopted and used successfully. We believe that the book is even suited for self-study. This book will provide an up-to-date first foundation for informed discussion of today's national and global issues.

We offer our gratitude to Vision Book House, an Imprint of Himalaya Publishing House Pvt. Ltd., who is leader in Commerce and Management publications. Our sincere regards to Mr. Niraj Pandey, Mr. Vijay Pandey, Bhagwan Singh, Mohsin Ahmed and Narendra Singh for interest shown and for the best effort put forth by the matter of publication of this book.

Finally, we express our sincere thank to SPS, Bengaluru for their excellent computer typesetting work and the printing.

We respectfully acknowledge that the critical comments and constructive suggestions for the improvement of this book are most welcome and will be greatly appreciated.

Bengaluru **Authors**

March, 2024

Syllabus

Unit – I Introduction to Mobile Technologies

Introduction: Brief History of Mobile Technologies, Different Mobile Technologies Key Mobile Application Services, Introducing Android, The Android Application Components, Exploring the Development Environment, Obtaining the Required Tools, Launching Your First Android Application, Exploring the IDE, Debugging Your Application, Publishing Your Application.

Using Activities: Fragments and Intents in Android: Working with Activities, Using Intents, Fragments, Using the Intent Object to Invoke Built-in Application.

Unit – II Working with the User Interface Using views

Working with the User Interface Using views

Understanding the Components of a Screen, Adapting to Display Orientation, Managing Changes to Screen Orientation, Utilizing the Action Bar, Creating the User Interface Programmatically Listening for UI Notification.

Using Basic Views: Using Picker Views, Using List Views to Display Long Lists, Understanding Specialized Fragments, Using Image Views to Display Pictures, Using Menus with Views Using WebView, Saving and Loading User Preferences, Persisting Data to Files, Creating and Using Databases.

Unit – III Designing User Interface

Designing User Interface Designing by Declaration, Creating the Opening Screen, Using Alternate Resources, Implementing an about Box, Applying a Theme, Adding a Menu, Adding Settings, Debugging with log Messages, Debugging with Debugger.

Unit – IV Creating Your Own Content

Creating Your Own Content Providers, Using the Content Provider, SMS Messaging, Sending Email, Displaying Maps, Getting Location Data, Monitoring a Location.

Putting SQL to Work Introducing SQLite, In and Out of SQLite, Hello Database, Data Binding, Using Content Provider, Implementing Content Provider. Reading/writing Local data, Accessing the Internal File System, Accessing the SD Card. Preparing App for Publishing, Deploying APK Files, Uploading in Market .. Consuming Web Services Using HTTP-Consuming JSON Services, Creating Your Own Services Binding Activities to Services, Understanding Threading.

Contents

Introduction to Mobile Technologies

UNIT - 1

Highlights

- Introduction
- Brief History of Mobile Technologies
- Different Mobile Technologies
- Key Mobile Application Services
- Introducing Android
- Android Application Components
- Exploring the Development Environment
- Obtaining the Required Tools
- Launching Your First Android Application
- Exploring the IDE
- Debugging Your Application
- Publishing Your Application
- Using Activities
- Fragments and Intents in Android
- Working with Activities
- Using Intents
- Using the Intent Object to Invoke
- Built–in Application

INTRODUCTION

The Latin term mobile means "ability to move" or portable. Therefore mobile technology means, technology that is portable.Mobile technology has gone a long journey from a simple two way pager to a device that fits in your pocket which can be used as a substitute for a computer.

Mobile application development is the process of creating software applications specifically designed to run on mobile devices such as smartphones and tablets.

Mobile application development originated in the early 1990s when the first mobile devices were introduced. However, those devices had limited capabilities and could only run simple applications such as calculators and address books. In 1996, the first mobile operating system called Palm OS was launched, which provided mobile app development platforms that enabled developers to create more advanced mobile applications and progressive web apps.

As the usage of mobile devices increased, the early 2000s saw significant growth in mobile application development. In 2007, Apple launched the iPhone, which introduced a new operating system iOS. This OS featured App development tools that enabled developers to create native mobile applications and publish them on the Apple App Store. That same year, Google also introduced the Android operating system, which led to the birth of Android development and the Google Play Store.

In recent years, mobile application development has undergone significant advancements from traditional software development by integrating new technologies like Augmented Reality (AR) and Virtual Reality (VR) into mobile applications and progressive web apps.

A Brief History of Mobile Technologies

The history of mobile technologies is a fascinating journey that has seen remarkable advancements over the years.

1. First Generation (1G):

1G refers to the first generation of analog cellular networks introduced in the early 1980s.These networks provided basic voice calling capabilities but had limited coverage and poor call quality.

2. Second Generation (2G)

2G systems, introduced in the late 1980s and early 1990s, marked the transition to digital cellular networks. 2G networks offered improved voice quality, security features, and introduced basic data services such as SMS (Short Message Service).

3. Third Generation (3G)

3G networks emerged in the early 2000s, offering higher data speeds and enabling advanced mobile services such as video calling and mobile internet browsing.

4. Fourth Generation (4G)

4G networks, introduced in the late 2000s and early 2010s, represented a significant leap in data speeds and network capacity. 4G networks enabled high-definition video streaming, online gaming, and other bandwidth-intensive applications on mobile devices.

5. Fifth Generation (5G)

5G is the latest generation of mobile technology, introduced in the 2010s and continuing to roll out globally.5G networks promise even faster data speeds, ultra-low latency, and support for massive connectivity through technologies like autonomous vehicles, augmented reality, and the Internet of Things (IoT).

DIFFERENT TYPES OF MOBILE TECHNOLOGIES

Mobile technologies are used in a variety of ways. There are many types of mobile technologies that can be used in the various mobile devices. Following are the few famous mobile technologies:

1. SMS

SMS, which stands for Short Message Service, is a text-based communication service. It is used to send messages to mobile devices.SMS allows users to send short text messages, typically limited to 160 characters per message.

Messages are transmitted over cellular networks using the signalling channels reserved for control purposes.

2. MMS

MMS (Multimedia Messaging Service) is a mobile technology that allows users to send multimedia content such as pictures, videos, audio clips, and longer text messages between mobile devices.

3. LTE (Long-Term Evolution)

Often referred to as 4G LTE, it provides high-speed wireless communication for mobile devices.LTE Advanced and LTE Advanced Pro are further enhancements offering faster data rates, improved coverage, and better performance.

4. 5G Technology

The latest generation of cellular network technology promising significantly faster data speeds, lower latency, and increased capacity compared to 4G LTE.

It enables revolutionary applications like autonomous vehicles, IoT, augmented reality (AR), virtual reality (VR), and high-definition video streaming.

5. GSM

GSM stands for the Global System for Mobile Communications. It's a type of mobile network that uses a technology called Global System for Mobile Communications (GSM). The GSM network is used by more than 80% of mobile phone users to make calls, send text messages and browse the internet.

6. CDMA

CDMA stands for Code Division Multiple Access, or CDMA. CDMA is a wireless technology standard that allows multiple users to simultaneously use the same frequency band. This is unlike other technologies, such as GSM, in which each user is assigned a specific frequency band. Only one user can communicate at any given time on that specific frequency.

7. Wi-Fi (Wireless Fidelity)

Wi-Fi is a wireless networking technology that allows us to connect to a network or to other computers or mobile devices across a wireless channel.Wi-Fi (Wireless Fidelity) is a generic acronym for a communication standard for a wireless network that functions as a Local Area Network without the use of cables or other types of cabling.

KEY MOBILE APPLICATION SERVICES

Mobile application services encompass a wide range of functionalities and features that enable mobile applications to work and provide value to users.These services can be broadly categorized into three groups:

1. Core Services

These are the essential services that all mobile applications need to function, such as:

a) ***User authentication and authorization:*** This service allows users to sign up for and log in to applications, and it controls what users can access within the application.

b) ***Data storage and management:*** This service allows applications to store and manage data, such as user preferences, app settings, and content.

c) ***Network communication:*** This service allows applications to send and receive data over the internet or other networks.

d) ***Device access:*** This service allows applications to access and use device features, such as the camera, microphone, and GPS.

2. Platform-specific Services

These are services that are specific to a particular mobile platform, such as Android or iOS. For example, Android offers services for accessing Google Play services, while iOS offers services for accessing Apple Pay.

3. Value-added Services

These are services that provide additional features and functionality to applications, such as:

a) ***Push notifications:*** This service allows applications to send notifications to users even when the application is not running.

b) ***Analytics:*** This service allows application developers to track how users are using their application and gather data to improve the application.

c) ***Social integration:*** This service allows users to connect their application accounts to social media accounts.

d) ***In-app payments:*** This service allows users to purchase goods and services within the application.

e) ***Location-based services:*** This service allows applications to use the device's location to provide location-based features, such as maps and navigation.

INTRODUCING ANDROID

A Powerful Mobile Operating System.Android is an open-source, Linux-based operating system specifically designed for mobile devices like smartphones and tablets. Developed by a consortium called the Open Handset Alliance (OHA) led by Google, it dominates the global mobile OS market with billions of users worldwide.

ANDROID VERSIONS

Android has gone through quite a number of updates since its first release.Table 1-1 shows the various versions of Android and their codenames.

Table 1-1: A Brief History of Android Versions

ANDROID VERSION	RELEASE DATE	CODENAME
1.1	February 9, 2009	
1.5	April 30, 2009	Cupcake
1.6	September 15, 2009	Donut
2.0/2.1	October 26, 2009	Éclair
2.2	May 20, 2010	Froyo
2.3	December 6, 2010	Gingerbread
3.0/3.1/3.2	February 22, 2011	Honeycomb
4.0	October 18, 2011	Ice Cream Sandwich

ANDROID VERSION	RELEASE DATE	CODENAME
4.1	July 9, 2012	Jelly Bean
4.4	October 31, 2013	KitKat
5.0	November 12, 2014	Lollipop
6.0	October 5, 2015	Marshmallow
7.0	TBD	Nougat

1. Android releases major updates once a year.Each version has a cute dessert codename (Cupcake, KitKat, Oreo, etc.).
2. The latest version of Android is Android 14, which was released on October 4, 2023. Android 14 is the 21st version of Android.

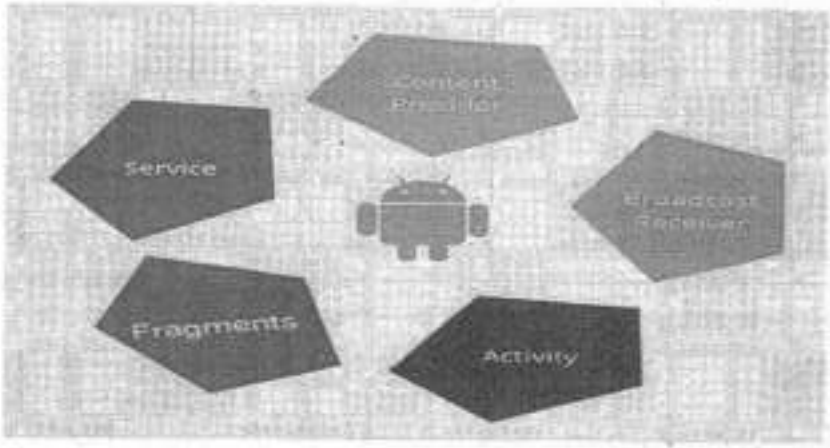
FEATURES OF ANDROID

As Android is open source and freely available to manufacturers for customization. Here are some key highlights:

1. ***Intuitive Interface:*** Touchscreen-friendly with familiar elements like home screen, app drawer, notifications, and widgets.
2. ***Customizable:*** Personalize your device with launchers, themes, wallpapers, and more.
3. ***Open Ecosystem:*** Download millions of free and paid apps from the Google Play Store.
4. ***Multitasking:*** Run multiple apps simultaneously and switch between them seamlessly.
5. ***Google Integration:*** Tightly integrated with Google services like Gmail, Maps, Drive, and more for a unified experience.
6. ***Notification Center:*** Stay informed with timely notifications from apps and system updates.
7. ***Security Features:*** Android offers various security features like sandboxing, fingerprint/face unlock, and encryption for data protection.
8. ***Accessibility Options:*** Provides options for users with disabilities, including screen readers, magnification, and voice control.

ANDROID APPLICATION COMPONENTS

In Android development, applications are built using various components that serve different purposes and interact with each other to create a cohesive user experience. The main components of an Android application are:



Activities

a) An activity represents a single screen with a user interface.

b) Handle user interaction with the screen (e.g., button clicks, text input).

c) Activities are the building blocks of an Android application, and multiple activities can be combined to create complex user interfaces and navigation flows.

d) Examples: Login screen, product details page, settings screen.

Services

a) A service is a component that runs in the background to perform long-running operations or handle tasks without a user interface.

b) Handle long-running tasks like playing music, syncing data, or handling notifications.

c) Don't have a UI themselves, but can interact with activities and fragments.

d) Examples: Music player service, background data sync service, notification service.

Content Providers

a) A content provider is a component that manages access to a structured set of data or content, such as a database or file system.

b) Content providers enable data sharing between applications and facilitate data storage, retrieval, and manipulation.

c) Examples: Contact list provider, music library provider, settings provider.

Broadcast Receivers

a) A broadcast receiver is a component that listens for system-wide broadcast messages or intents and responds to them accordingly.

b) Broadcast receivers are often used to trigger actions or start services in response to system events, such as incoming calls or battery low warnings.

c) Examples: Network connectivity changes, battery level changes, receiving SMS messages.

Fragments

a) A fragment is a reusable portion of a user interface or behavior that can be combined with other fragments within an activity.

b) Fragments are used to create flexible and responsive UI designs that adapt to different screen sizes and orientations.

c) Examples: Navigation bar, comment section, image slider.

EXPLORING THE ANDROID DEVELOPMENT ENVIRONMENT

Exploring the development environment in Android involves understanding the tools and components necessary for creating Android applications. Here's a roadmap to kick-start the exploration:

1. Choosing the Right Tools

a) *Android Studio:* The official and most widely used integrated development environment (IDE) by Google. Supports Java, Kotlin, C++, and offers built-in tools for layout editing, debugging, and app building.

b) *Visual Studio Code:* A versatile code editor with numerous Android extensions like Flutter and Kotlin support. Offers customization and flexibility.

c) *Android IDE for IntelliJ IDEA:* Another popular option based on IntelliJ IDEA platform, provides similar features to Android Studio with a different look and feel.

2. Setting Up the Environment:

a) *System Requirements:* Ensure your computer meets the recommended specifications for your chosen IDE.

b) *Download and Install:* Download the latest version of your chosen IDE and follow the installation instructions.

c) *SDK Setup:* Configure the Android SDK (Software Development Kit) within your IDE, downloading required tools and platform versions.

d) *Device or Emulator:* Choose a physical Android device or use an emulator to test and run your apps.

e) Developers can create virtual devices with different screen sizes, resolutions, Android versions, and hardware configurations to simulate various real-world devices.

3. Getting Familiar with the Interface

a) *Project Structure:* Understand the organization of your project files including code, resources, and configuration.

b) *Code Editor:* Learn the features and functionalities of your IDE's code editor, like syntax highlighting, code completion, and debugging tools.

c) *Layout Editor:* Explore tools for building user interfaces visually, using drag-and-drop elements and property panels.

d) *Build System:* Grasp how to compile and build your app for different devices and Android versions.

4. Run and Debugging Tools

a) Android Studio provides tools for running and debugging Android applications on physical devices and emulators.

b) Developers can deploy the application to a connected device or emulator directly from Android Studio and monitor the application's behaviour using the Logcat tool for logging messages and debugging.

Exploring these components within the Android development environment is essential for getting started with Android app development and building high-quality applications for the Android platform.

DEVELOPING AND EXECUTING THE FIRST ANDROID APPLICATION

Step 1 - System Requirements

The required tools to develop Android applications are open source and can be downloaded from the Web. Following is the list of software's required before you start your Android application programming.

a) Java JDK21 or latest version

b) Android Studio latest version

c) Android SDK

Step 2 - Setup Android Environment

Android Studio is the official IDE for android application development. In this section ,you learn the basic steps to create an android application using the android studio IDE. It is based on the IntelliJ IDEA IDE, provides a comprehensive set of tools for developers to design, build, test, and debug Android applications.

In order to launch android studio,make sure that JDK((Java Development Kit) is installed in the system, Install JDK via the following steps:

i. Download JDK

a) Goto Java SE download site

https://www.oracle.com/java/technologies/downloads

b) Click the "Oracle JDK Download" button.

c) Check "Accept License Agreement".

d) Choose the JDK as per your system requirement.

ii. Install JDK

a) Double click on download to setup and Run.

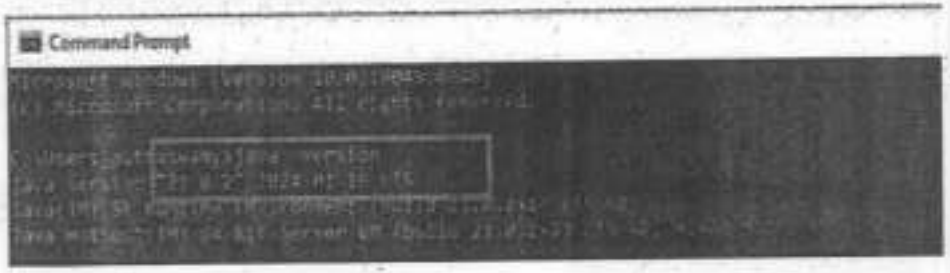
b) By default, JDK is installed in the directory "C:\Program Files\Java\jdk-21.0.", Accept the defaults.

iii. Verify the JDK Installation

Open a CMD via the following steps:

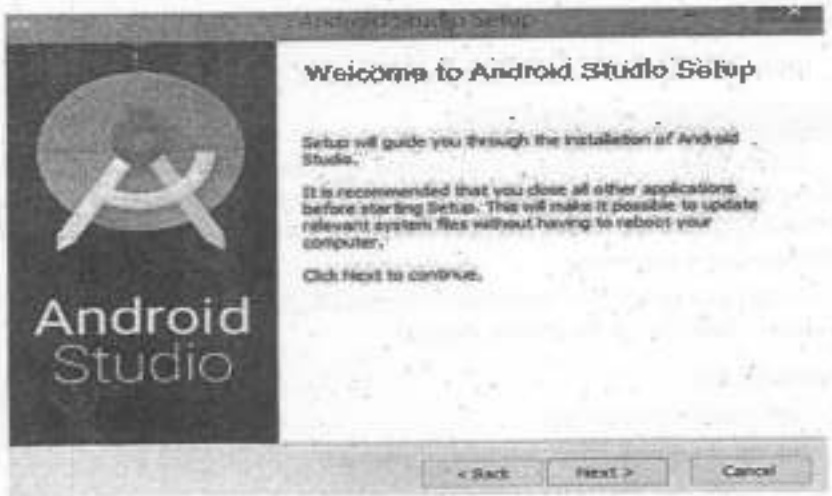
Click the Search button and Type "cmd" ? Choose "Command Prompt".

The following command to verify that JDK/JRE is properly installed and display their version.

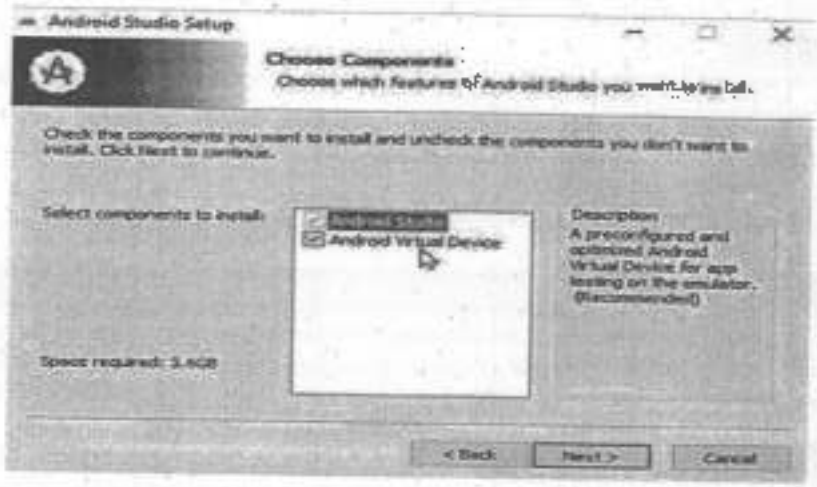


Step 3 : Install "Android Studio IDE"

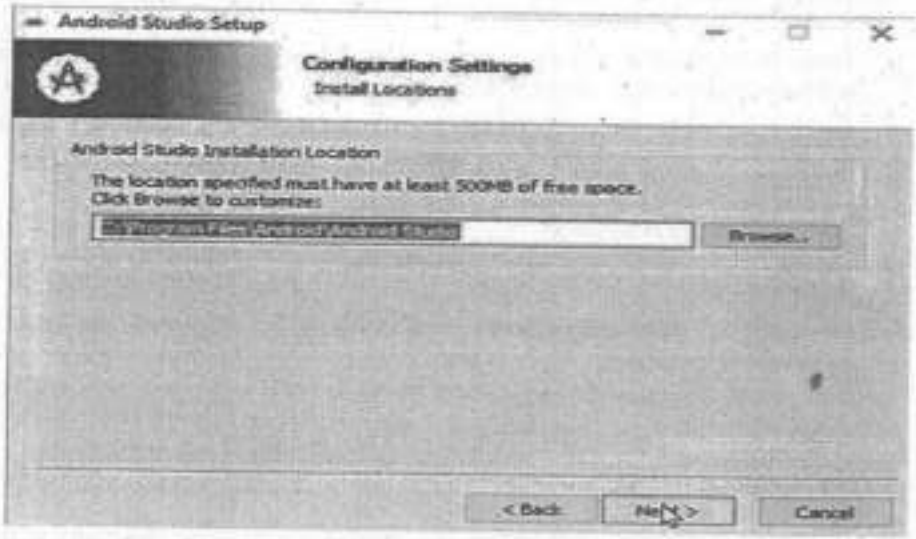
Open any web browser and navigate to the Android Studio @https://developer.android.com/studio.Click "Download Android Studio" to download the executable installer.



Once Android Studio is launched,its time to check the components, which are required to create applications,



Android Studio IDE by default will be installed in "C:\Program Files\Android\Android Studio".

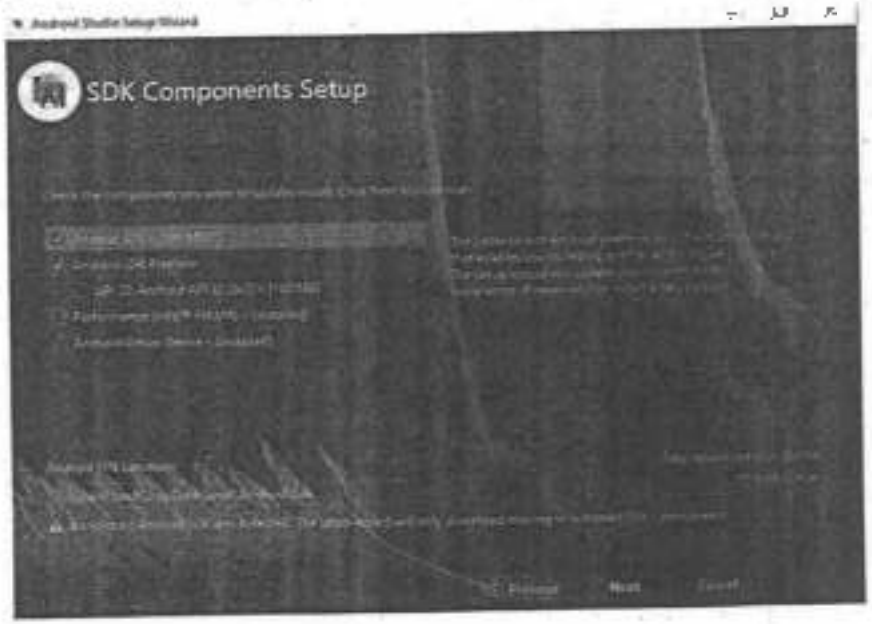


After done all above steps perfectly, you must get finish button

Step 4 :Installing the Android SDK

The Android SDK stands for a software development kit. It is a set of development tools that are used to develop applications. The Android SDK includes required libraries,debugger, emulator and more.

Android SDK by default will be installed in the"C:\Users\username\AppData\Local\Android\Sdk".

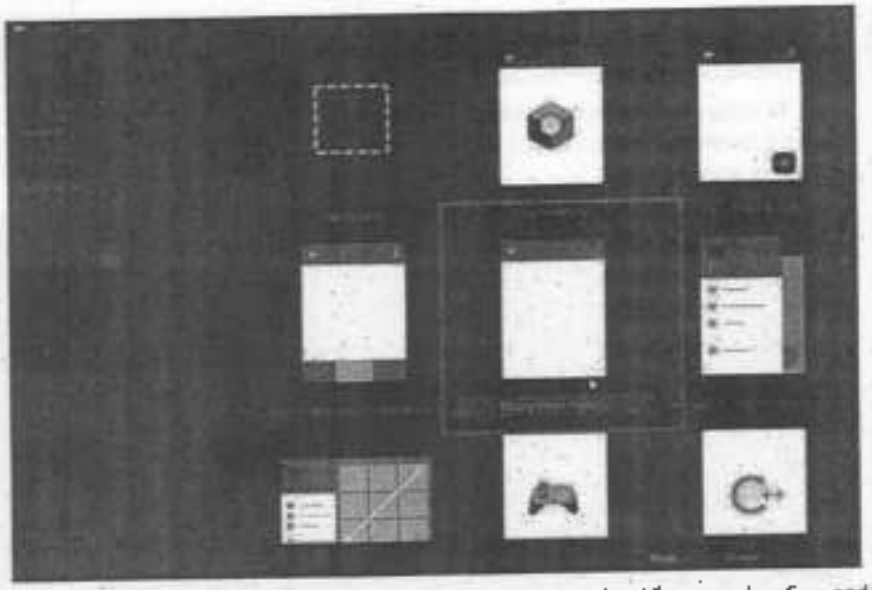


EXECUTING THE FIRST ANDROID APPLICATION

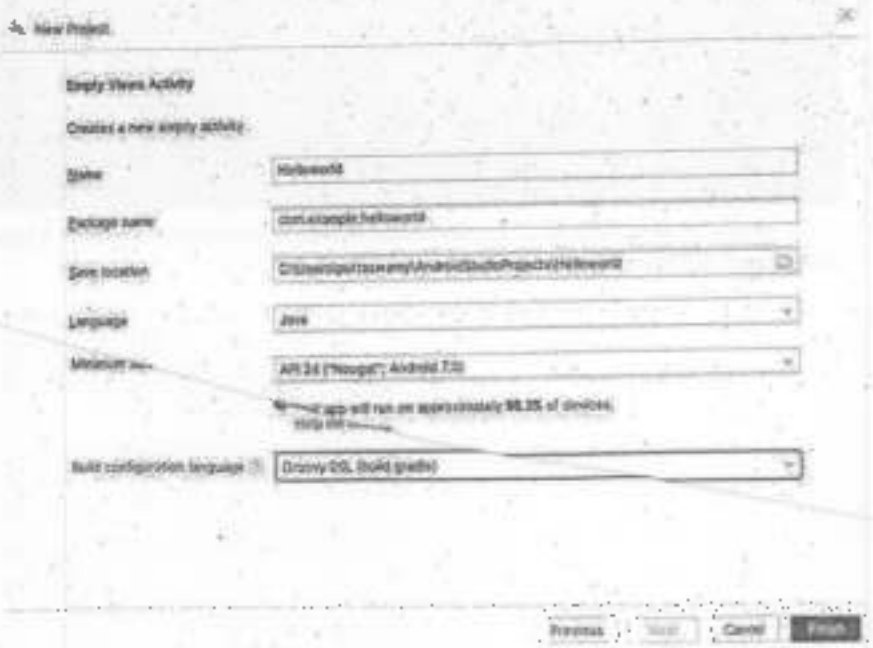
Development of Hello World Application

To design an android application to display Hello World,First step is to create a simple Android Application using Android studio. When you click on Android studio icon, it will show screen as shown below:

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



We'll finish creating the project by configuring some details about its name, location, and the API version of it.



SOURCE CODE

Main Activity File

1. The main activity code is a Java file MainActivity.java.The main activity file typically refers to the Java (or Kotlin) file that represents the main entry point of an Android application. This file is where you define the behavior and functionality of the initial screen or activity of your app.
2. By default, when you create a new Android project in Android Studio, it generates a main activity file for you. The main activity file usually follows a naming convention like MainActivity.java or MainActivity.kt, depending on whether you're using Java or Kotlin for development.
3. It is responsible for initializing the user interface (UI) and handling user interactions.

Here's a basic example of what a main activity file might look like in Java:

```
package com.example.helloandroid;

import androidx.appcompat.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);
    }
}
```

Activitymain.xml

The activity_main.xml is a layout file available in res/layout directory, that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application. For your "Hello World!" application, this file will have following content related to default layout.

```
<?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">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

Running the application on Emulator(AVD)

To run the app from Android studio,open Helloworld project's activity file and click Run Icon from the tool bar.The below Figure shows the application running on the Android Emulator.

USING ACTIVITIES, FRAGMENTS AND INTENTS IN ANDROID

1. An activity is an application component that allows a user to perform an operation.Generally, a user performs various operations,such as making a call,capturing a photo, sending a message or viewing a map.For each such operation,an activity is beign defined.
2. An application is a collection of these activities.The main objective of an activity is to interact with user,therefore each activity provides a window in which the user interface is being created.
3. An important concept related to activities is fragments.A fragment represents a behaviour or a portion of the user interface in an activity.It is always embedded in an activity an its lifecycle is directly affected by its host activity's life cycle.
4. Another important concept is Intent that allows the different activities of different applications to work together in such a manner that they all belong to a single application.

WORKING WITH ACTIVITIES

Activities play a crucial role in an android application because they are used to make up the screens for users.Therefore,it is very essential to learn how to start,stop,suspend and resume an android activity.

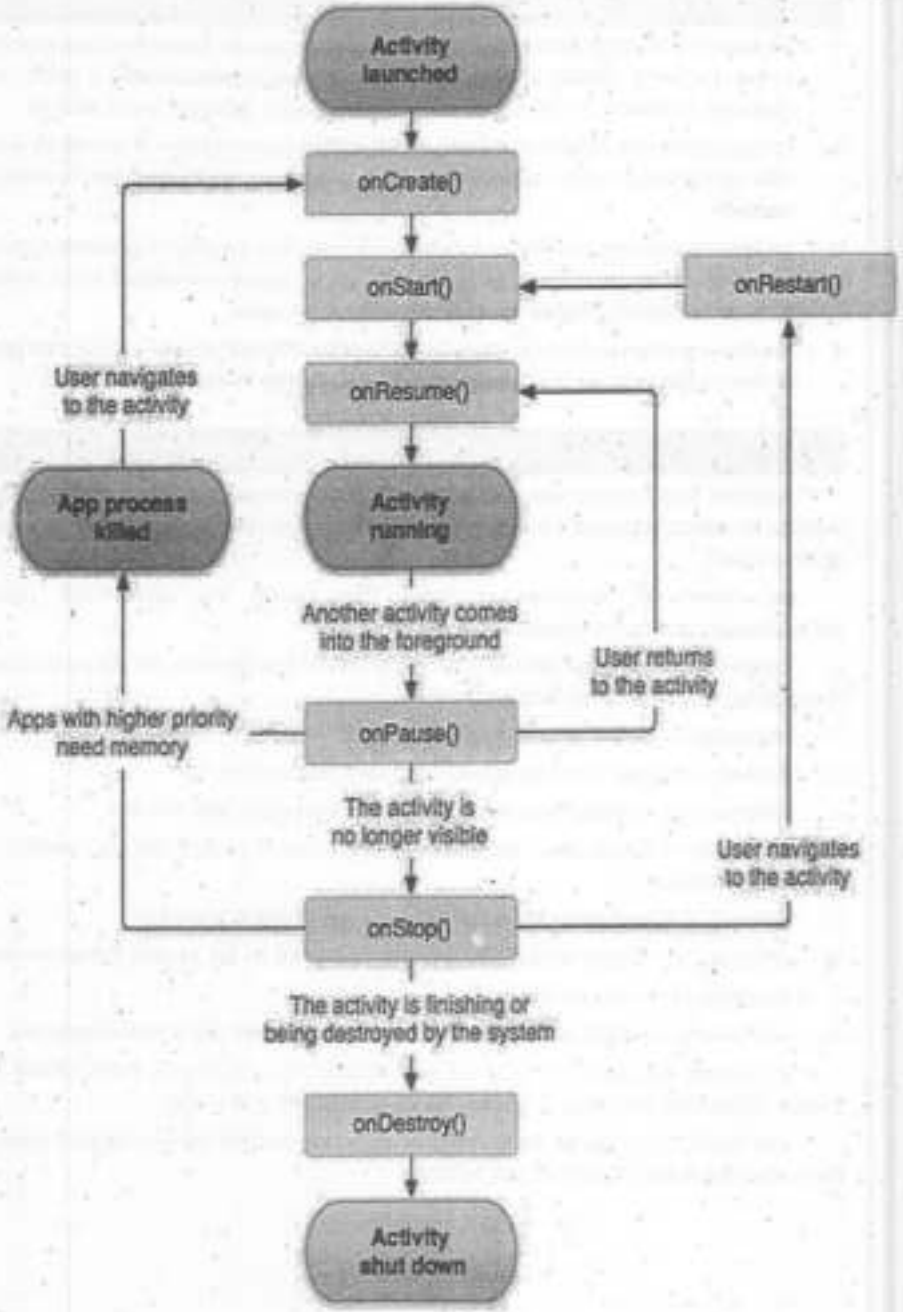
An activity is composed of views that handle the layout and provide text elements,button,forms and graphics to a device screen.

The Activity base class defines a series of events that governs the life cycle of an activity. The Activity class defines the following events:

1. onCreate() — Called when the activity is first created
2. onStart() — Called when the activity becomes visible to the user
3. onResume() — Called when the activity starts interacting with the user
4. onPause() — Called when the current activity is being paused and the previous activity is being resumed
5. onStop() — Called when the activity is no longer visible to the user
6. onDestroy() — Called before the activity is destroyed by the system (either manually or by
7. the system to conserve memory)
8. onRestart() — Called when the activity has been stopped and is restarting again.

By default, the activity created for you contains the onCreate() event. Within this event handler is the code that helps to display the UI elements of your screen.

The Figure below shows the life cycle of an activity and the various stages it goes through - from when the activity is started until it ends.



UNDERSTANDING THE LIFE CYCLE OF AN ACTIVITY

Let's create the ManageActivityCycle application that would display the appropriate message according to the event being raised.Perform the following steps to create and run the ManageActivityCycle application:

1. Create the ManageActivityCycle application in the android studio IDE under the com.developers.dripdownlist package.
2. In the MainActivity.java file, add the following statements:

```
package com.developers.dripdownlist;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
public class MainActivity extends Activity {
String tag = "Events";
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
Log.d(tag, "In the onCreate() event");
}
public void onStart()
{
        super.onStart();
        Log.d(tag, " onStart()");
}
public void onRestart()
{
        super.onRestart();
        Log.d(tag, "onRestart() ");
}
public void onResume()
{
        super.onResume();
        Log.d(tag, "onResume() ");
}
public void onPause()
```

```
{
        super.onPause();
        Log.d(tag, " onPause() ");
 }
public void onStop()
{
         super.onStop();
        Log.d(tag, " onStop() ");
   }
   public void onDestroy()
    {
            super.onDestroy();
           Log.d(tag, "onDestroy() ");
         }
}
```

Save the changes in the ManageActivityCycle application and click the Run button from the toolbar of android studio IDE.The application is executed as shown below:

Once the application starts, the onPause( ) ,onStop( ),onDestroy( ),onCreate( ),onStart( ) and onResume( ) events are raised.You can view the message in the LogCat window of the Debug perspective,as shown below:

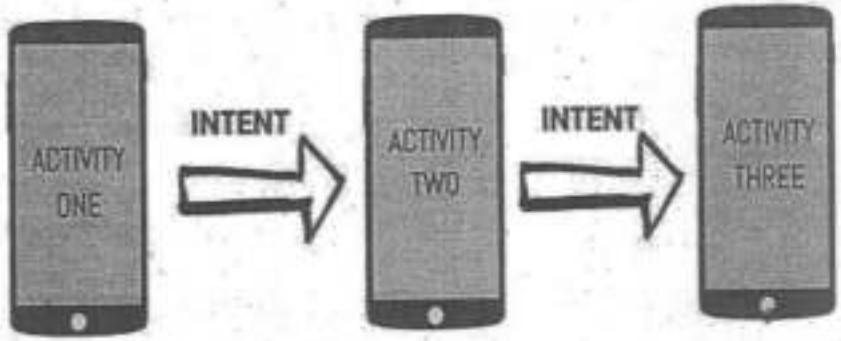
```
:10:47.305 17280-17280/com.developers.dripdowntest E/Activity 1:  onPause
:10:48.793 17280-17280/com.developers.dripdowntest E/Activity 1:  onStop
:10:48.813 17280-17280/com.developers.dripdowntest E/Activity 1:  onDestroy
:10:52.482 17280-17280/com.developers.dripdowntest E/Activity 1:  onCreate
:10:52.495 17280-17280/com.developers.dripdowntest E/Activity 1:  onStart
:10:52.498 17280-17280/com.developers.dripdowntest E/Activity 1:  onResume
```

LINKING ACTIVITIES USING INTENTS

Intents:

An android Intent is a messaging object used to switch between different android components.An intent's most common use is to launch a new activity from the current activity.

Intent facilitates communication between different components.



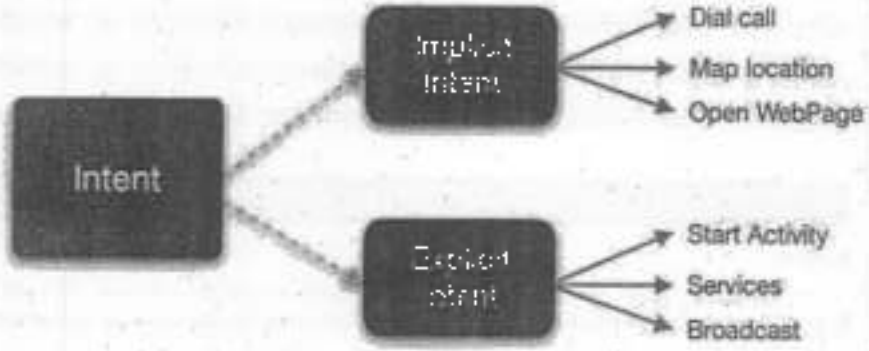
Common use cases for Intents include:

1. *Starting Activities:* Use intents to launch new activities within your application or to launch activities from other applications.
2. *Broadcasting Messages:* Use Intents to send broadcast messages within your application or to other applications, allowing them to receive and respond to events.
3. *Invoking Services:* Use intents to start services that perform background tasks or handle long-running operations.

4. ***Passing Data:*** Intents can carry data (extra information) as key-value pairs, allowing components to exchange information, such as passing data between activities or between different parts of your application.

Intent are of two types

- Explicit Intent
- Implicit Intent



a) Explicit Intent

1. Explicit Intent is used to invoke a specific target component.It is used to switch from one activity to another in the same application. It is also used to pass data by invoking the external class.
2. Explicit intents specify the target component by providing the exact class name of the component to be invoked.
3. Explicit Intent example: we can use an explicit Intent to start a new activity when the user invokes an action or plays music in the background or On click button go to another Activity.

Example:

```
                    =                  (, TargetActivity.class);
startActivity(explicitIntent);
```

2. Implicit Intent

1. Implicit Intents do not specify the target component by class name but instead declare an action to be performed.
2. They are used when you want the system to determine the appropriate component to handle the Intent based on the action and data provided.

3. Implicit Intent example1: A button on click of which you will redirect to a web page. If your Device has multiple browsers then the options popup (bottom sheet) will open and show them all.
4. Implicit Intent example2 : If the user wants to see a location on a map,we can use an implicit intent to switch to another app that displays a specific location on a map.

Example:

```
                    =                  (Intent.ACTION_VIEW);
implicitIntent.setData(Uri.parse(                                  ));
startActivity(implicitIntent);
```

FRAGMENT IN ANDROID

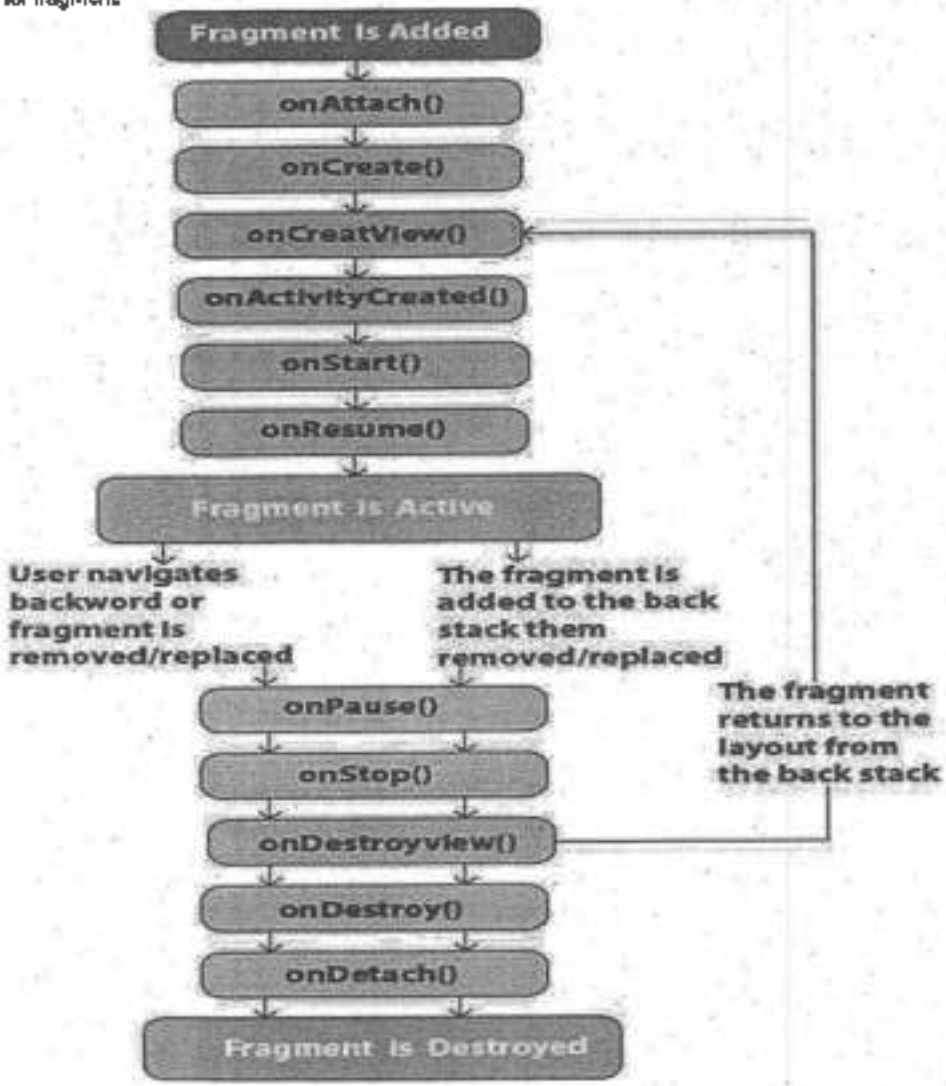
1. In these days, we can see a variety of android devices available in market,which have different resolution and screen density.And it is a challenge for android application developers to develop applications that support all types of devices.In such a case,fragments come into picture.
2. In Android, the fragment is the part of Activity which represents a portion of User Interface(UI) on the screen. It is the modular section of the android activity that is very helpful in creating UI designs that are flexible in nature and auto-adjustable based on the device screen size.
3. Fragments are always embedded in Activities i.e., they are added to the layout of activity in which they reside. Multiple fragments can be added to one activity. This task can be carried out in 2 ways:

a) Statically: Explicitly mention the fragment in the XML file of the activity. This type of fragment can not be replaced during the run time.

b) Dynamically: FragmentManager is used to embed fragments with activities that enable the addition, deletion, or replacement of fragments at run time.

ANDROID FRAGMENT LIFECYCLE

The lifecycle of android fragment is like the activity lifecycle. There are 12 lifecycle methods for fragment.



ANDROID FRAGMENT LIFECYCLE METHODS

Method	Description
1. onAttach(Activity)	1. It is called only once when it is attached with activity.
2. onCreate(Bundle)	2. It is used to initialize the fragment.
3. onCreateView(LayoutInflater, ViewGroup, Bundle)	3. creates and returns view hierarchy.
4. onActivityCreated(Bundle)	4. It is invoked after the completion of onCreate() method.
5. onViewStateRestored(Bundle)	5. It provides information to the fragment that all the saved state of fragment view hierarchy has been restored.
6. onStart()	6. makes the fragment visible.
7. onResume()	7. makes the fragment interactive.
8. onPause()	8. is called when fragment is no longer interactive.
9. onStop()	9. is called when fragment is no longer visible.
10. onDestroyView()	10. allows the fragment to clean up resources.
11. onDestroy()	11. allows the fragment to do final clean up of fragment state.
12. onDetach()	12. It is called immediately prior to the fragment no longer being associated with its activity.

ANDROID FRAGMENT EXAMPLE

Let's have a look at the simple example of android fragment.

File: activity_main.xml

```
<?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:layout_width="fill_parent"
    android:layout_height="fill_parent"
    tools:context="example.demo.com.fragmentexample.MainActivity">
    <fragment
        android:id="@+id/fragment1"
        android:name="example.demo.com.fragmentexample.Fragment1"
        android:layout_width="0px"
```

```
        android:layout_height="match_parent"
        android:layout_weight="1"
        />
    <fragment
        android:id="@+id/fragment2"
        android:name="example.demo.com.fragmentexample.Fragment2"
        android:layout_width="0px"
        android:layout_height="match_parent"
        android:layout_weight="1"
        />
</LinearLayout>
```

File: fragment_fragment1.xml

```
<FrameLayout 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:background="#F5F5DC"
    tools:context="example.demo.com.fragmentexample.Fragment1">
    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/hello_blank_fragment" />
</FrameLayout>
```

File: fragment_fragment2.xml

```
<FrameLayout 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:background="#F0FFFF"
    tools:context="example.demo.com.fragmentexample.Fragment2">
    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
```



```
        android:text="@string/hello_blank_fragment" />
</FrameLayout>
```

File: MainActivity.java

```
package example.demo.com.fragmentexample;
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);
    }
}
```

File: Fragment1.java

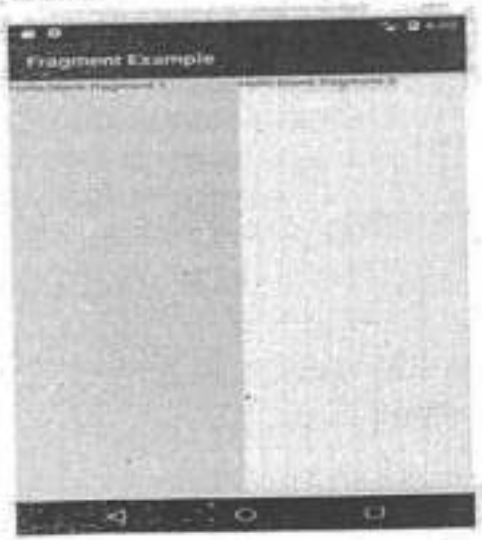
```
package example.demo.com.fragmentexample;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class Fragment1 extends Fragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_fragment1, container, false);
    }
}
```

File: Fragment2.java

```
package example.demo.com.fragmentexample;
import android.os.Bundle;
```

```
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class Fragment2 extends Fragment {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
   }
   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {
      // Inflate the layout for this fragment
      return inflater.inflate(R.layout.fragment_fragment2, container, false);
   }
}
```

Output:



USING THE INTENT OBJECT TO INVOKE BUILT-IN APPLICATION

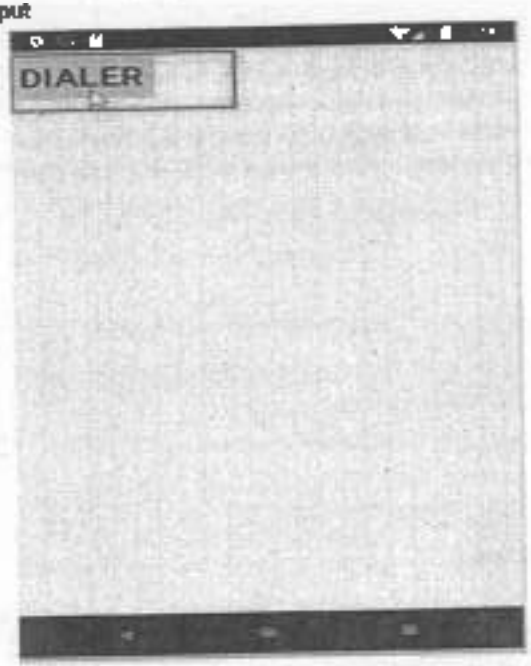
Intent object can also be used to call build-in applications, such as contacts, messaging and phone call.For example, Imagine your application needs to send a message to a particular person whose details have been saved in the contacts application. In this example ,we need not to create an application to completely load all your contacts. Instead, we can simply select the person by calling the contacts application.

Example program:

File: MainActivity.java

```
package com.example.mydemo;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
public class MainActivity extends Activity {
Button b1;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
b1=(Button) findViewById(R.id.b1);
b1.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent i1=new Intent();
i1.setAction(Intent.ACTION_DIAL);
startActivity(i1);
}
});
}
}
```

Output



REVIEW QUESTIONS

1. Describe the history of mobile technologies.
2. Explain different types of mobile technologies.
3. Briefly explain the key mobile application services.
4. List the features of android operating system.
5. Explore the steps to install and configure android studio and SDK.
6. Write the steps to develop an android application?
7. Explain the life cycle of an android activity.
8. Explain different android application components.
9. Briefly discuss about fragments and intents.
10. Brief about calling Built-in applications using intents.

Working with the User Interface Using Views

Highlights

UNIT-2

WORKING WITH THE USER INTERFACE USING VIEWS

Understanding the Components of a Screen

1. In Android development, a screen or user interface (UI) is typically built using various components to create a visually appealing and interactive experience for the user.
2. In Android, the primary building block for a screen is an activity.An activity contains Views and ViewGroups.

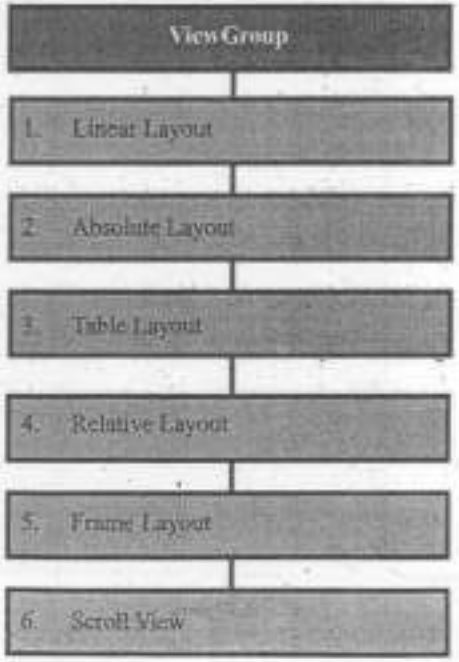
VIEW

A view is a widget that has an appearance on screen.Examples of views are buttons, labels,radio buttons,check boxes and text boxes.

VIEWGROUP

One or more views can be grouped together 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.

Android supports the following ViewGroups:

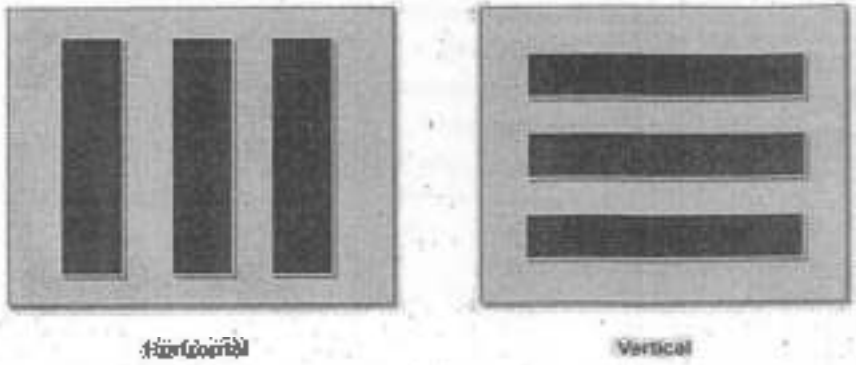


1. LINEAR LAYOUT

a) In Android development, LinearLayout is a fundamental ViewGroup component used to arrange views in a single direction, either horizontally or vertically based on the orientation property.

b) In android, we can specify the linear layout orientation using android:orientation attribute.

c) Following is the pictorial representation of linear layout in android applications.



IMPORTANT ATTRIBUTES OF LINEARLAYOUT

	Attributes	*Description*
1.	android:layout_weight	1. It is defined individually to the child's views to specify how LinearLayout divides the remaining space amongst the views it contains
2.	android:weightSum	2. Defines the total weight sum
3.	android:orientation	3. How the elements should be arranged in the layout. It can be horizontal or vertical.
4.	android:gravity	4. It specifies how an object should position its content on its X and Y axes. Possible values are – center_vertical, fill, center, bottom, end, etc.
5.	android:layout_gravity	5. Sets the gravity of the View or Layout relative to its parent. Possible values are – center_vertical, fill, center, bottom, end, etc.
6.	android:baselineAligned	6. This must be a boolean value, either "true" or "false" and prevents the layout from aligning its children's baselines.
7.	android:id	7. This gives a unique id to the layout.

Android LinearLayout Example

open an activity_main.xml file from \res\layout path and write the code like as shown below

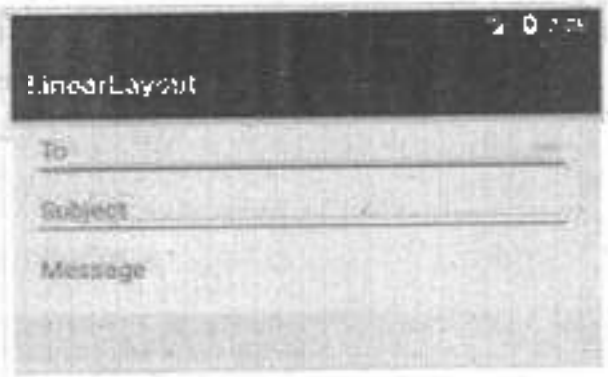
File type : activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="20dp"
    android:paddingRight="20dp"
    android:orientation="vertical" >
    <EditText
        android:id="@+id/txtTo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="To"/>
    <EditText
        android:id="@+id/txtSub"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Subject"/>
    <EditText
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="Message"/>
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="Send"/>
</LinearLayout>
```


File type : MainActivity.java

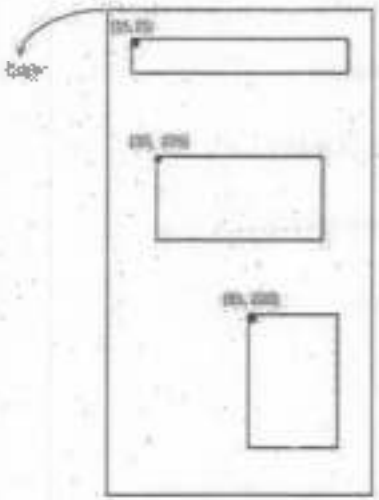
```
package com.demo.linearlayout;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Output of Android LinearLayout Example.



2. ABSOLUTE LAYOUT

An Absolute Layout allows you to specify the exact location .i.e., X and Y coordinates of its children with respect to the origin at the top left corner of the layout. The absolute layout is less flexible and harder to maintain for varying sizes of screens that's why it is not recommended.



The Syntax for Absolute Layout

```
<AbsoluteLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
<!--add child views-->
</AbsoluteLayout>
```

3. TABLE LAYOUT

a) In android, TableLayout is a ViewGroup subclass that is used to display the child View elements in rows and columns.

b) In android, TableLayout will position its children elements into rows and columns and it won't display any border lines for rows, columns or cells.

c) The TableLayout in android will work same as the HTML table and the table will have as many columns as the row with the most cells. The TableLayout can be explained as <table> and TableRow is like <tr> element.

Android TableLayout Example

File type : activity_main.xml

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



```
android:layout_height="match_parent"
android:layout_marginTop="100dp"
android:paddingLeft="10dp"
android:paddingRight="10dp" >
<TableRow android:background="#0079D6" android:padding="5dp">
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="UserId" />
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="User Name" />
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="Location" />
</TableRow>
<TableRow android:background="#DAE8FC" android:padding="5dp">
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="1" />
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="Suresh Dasari" />
   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
```

```
        android:layout_weight="1"
        android:text="Hyderabad" />
    </TableRow>
    <TableRow android:background="#DAE8FC" android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="2" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Rohini Alavala" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Guntur" />
    </TableRow>
    <TableRow android:background="#DAE8FC" android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="3" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Trishika Dasari" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
```



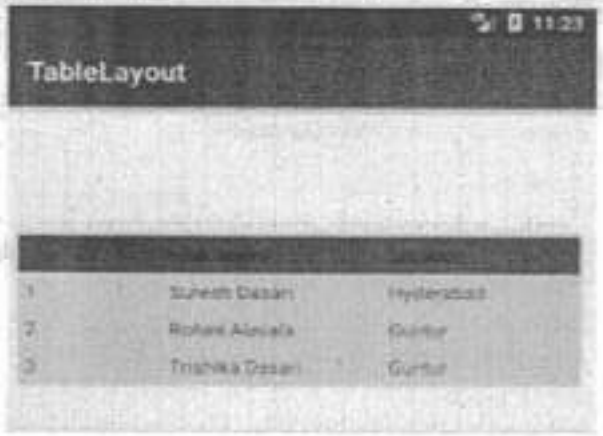
```
            android:text="Guntur" />
    </TableRow>
</TableLayout>
```

File type : MainActivity.java

```
package com.demo.linearlayout;
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);
    }
}
```

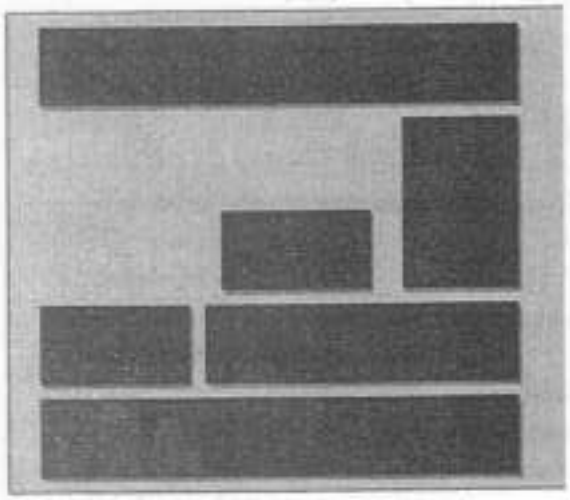
Output of Android TableLayout Example



4. RELATIVE LAYOUT

a) In android, RelativeLayout is a ViewGroup which is used to specify the position of child View instances relative to each other (Child A to the left of Child B) or relative to the parent (Aligned to the top of parent).

b) Following is the pictorial representation of relative layout in android applications.



Android RelativeLayout Example

File type : activity_main.xml

```
<?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 Button"
   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 Button"
   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 Button"
   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 Button"
   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 Button"
   android:layout_centerVertical="true"
   android:layout_centerHorizontal="true"/>
</RelativeLayout>
```

File type : MainActivity.java

```
package com.example.example1;
import androidx.appcompat.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);
   }
}
```

Output

5. FRAME LAYOUT

a) In android, Framelayout is a ViewGroup subclass that is used to specify the position of View instances it contains on the top of each other to display only single View inside the FrameLayout.

b) In simple manner, we can say FrameLayout is designed to block out an area on the screen to display a single item.

c) In android, FrameLayout will act as a placeholder on the screen and it is used to hold a single child view.

d) In FrameLayout, the child views are added in a stack and the most recently added child will show on the top. We can add multiple children views to FrameLayout and control their position by using gravity attributes in FrameLayout.

e) Following is the pictorial representation of frame layout in android applications.

Frame layout Example

File type : activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <ImageView
      android:id="@+id/imgvw1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:scaleType="centerCrop"
      android:src="@drawable/flimg" />
   <TextView
      android:id="@+id/txtvw1"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_marginTop="40dp"
      android:background="#4C374A"
      android:padding="10dp"
      android:text="Grand Palace, Bangkok"
      android:textColor="#FFFFFF"
      android:textSize="20sp" />
   <TextView
      android:id="@+id/txtvw2"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right|bottom"
        androidbackground="#AA000000"
        androidpadding="10dp"
        android:text="21/Aug/2017"
        android:textColor="#FFFFFF"
        android:textSize="18sp" />
</FrameLayout>
```

File type : MainActivity.java

```
package com.demo.linearlayout;
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);
    }
}
```

Output of Android FrameLayout Example

6. SCROLLVIEW IN ANDROID

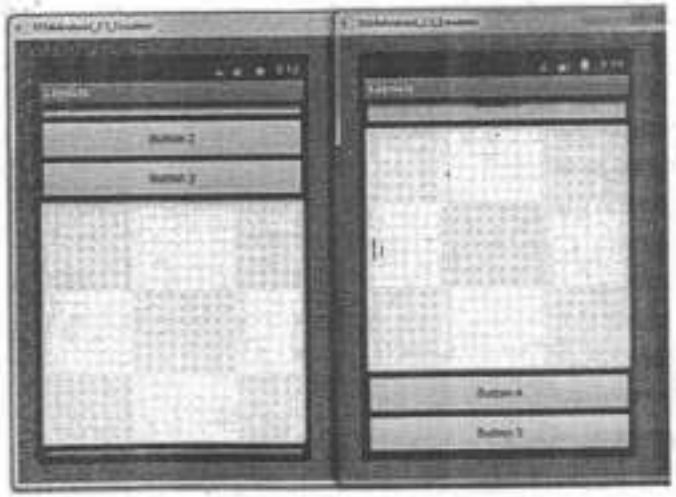
- In Android, a ScrollView is a view group that is used to make vertically scrollable views. A scroll view contains a single direct child only.The android.widget.ScrollView class provides the functionality of scroll view.
- ScrollView is used to scroll the child elements of palette inside ScrollView. Android supports vertical scroll view as default scroll view. Vertical ScrollView scrolls elements vertically.Android uses HorizontalScrollView for horizontal ScrollView.

ScrollView Example

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView android:layout_width="match_parent"
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_height="match_parent">
</ScrollView>
<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="300px"
/>
<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>
```

Output:



ADAPTING TO DISPLAY ORIENTATION

Android supports two screen orientations: portrait and landscape. By default,when you change the display orientation of your Android device, the current activity that is displayedwill automatically redraw its content in the new orientation. This is because the onCreate() event of the activity is fired whenever there is a change in display orientation.

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

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.

3. Anchoring Views

a) Anchoring views in Android refers to techniques you can use to position views relative to other views, the parent layout, or the edges of the screen. This ensures your UI adapts to different screen sizes and resolutions, providing a consistent and usable experience for users.

b) It can easily be achieved by using RelativeLayout.

File type : main.xml file

```
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
tools:context=".MainActivity">
   <Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Position - Top Left"
android:id="@+id/buttondisplay1"
android:layout_alignParentLeft="true"
android:layout_alignParentTop="true"
/>
   <Button
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Position - Top Right"
android:id="@+id/buttondisplay2"
android:layout_alignParentTop="true"
android:layout_alignParentRight="true"
/>
   <Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Position - Bottom Left"
android:id="@+id/buttondisplay3"
android:layout_alignParentLeft="true"
android:layout_alignParentBottom="true"
/>
   <Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Position - Bottom Right"
android:id="@+id/buttondisplay4"
android:layout_alignParentRight="true"
android:layout_alignParentBottom="true"
/>
   <Button
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Position - Middle"
android:id="@+id/buttondisplay5"
android:layout_centerVertical="true"
android:layout_centerHorizontal="true"
/>
</RelativeLayout>
```


Output:

In the above image, there are various views of the Button element.

a) Layout_alignParentLeft – This property aligns the view to the left of the parent view.

b) Layout_alignParentRight – This property aligns the view to the right of the parent view.

c) Layout_alignParentTop – This property aligns the view to the top of the parent view.

d) Layout_alignParentBottom – This property aligns the view to the bottom of the parent view.

e) Layout_centerVertical – This property aligns the view at the center vertically within its parent view.

f) Layout_centerHorizontal – This property aligns the view at the center horizontally within its parent view.

If the screen orientation is changed from portrait 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.

RESIZING AND REPOSITIONING

a) 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.

b) To support landscape mode, you can create a new folder in the res folder and name it as layout-land (representing landscape).

c) The main.xml file contained within the layout folder defines the UI for the activity in portrait mode, whereas the main.xml file in the layout-land folder defines the UI in landscape mode.

Now add below code in the main.xml file exists in res/layout/land folder.

```
<Button
   android:layout_width="120px"
   android:layout_height="wrap_content"
   android:text="Position - Top Middle"
   android:id="@+id/buttondisplay6"
   android:layout_centerVertical="true"
   android:layout_centerHorizontal="true"
   android:layout_alignParentTop="true"
   />
<Button
   android:layout_width="120px"
   android:layout_height="wrap_content"
   android:text="Position - Bottom Middle"
   android:id="@+id/buttondisplay7"
   android:layout_centerVertical="true"
   android:layout_centerHorizontal="true"
```



```
   android:layout_alignParentBottom="true"
   />
```

Run the app and see the difference at run time, when the activity is loaded in portrait mode it will display five buttons but if the activity is loaded in landscape mode, seven buttons are displayed.

It shows that different XML files are loaded when the device is in different orientations.

MANAGING CHANGES TO SCREEN ORIENTATION

let's explore what happens to an activity's state when the device changes orientation.

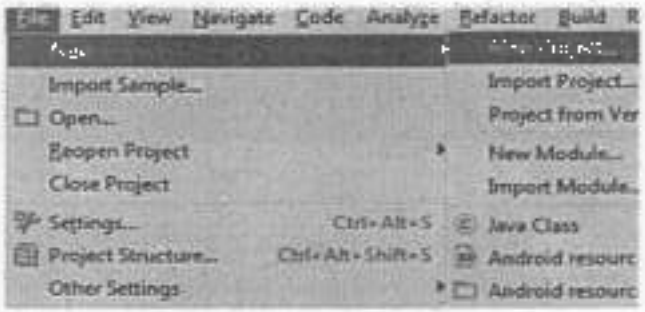
Understanding Activity Behavior When Orientation Changes.

a) Activity Behavior Changes When the Screen Orientation Changes in Android,So when the device orientation changes, first the Activity will disappear for a millisecond when the onPause(), OnStop, and onDestroy() methods are called.

b) After a few milliseconds, the activity will be restarted when the onCreate(), onStart() and onResume() methods are called.

Implementation

Create a new project by selecting the File->New->New Project.



Add 2 EditText element in the activity_main.xml file. Change the layout as LinearLayout.

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical"
tools:context=".MainActivity">
   <EditText
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:id="@+id/txt1"
/>
   <EditText
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:id="@+id/txt2"
/>
</LinearLayout>
```

Add the code, given below, in the MainActivity.java file:

```
package com.example.administrator.orientationapp;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
```



```
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
public class MainActivity extends ActionBarActivity {
  @Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d("ChangeStateInfo","onCreate");
  }
  @Override
  public void onStart(){
      Log.d("ChangeStateInfo", "onStart");
      super.onStart();
  }
  @Override
  public void onResume(){
      Log.d("ChangeStateInfo","onResume");
      super.onResume();
  }
  @Override
  public void onPause(){
    Log.d("ChangeStateInfo","onResume");
      super.onPause();
  }
  @Override
  public void onStop(){
      Log.d("ChangeStateInfo","onStop");
      super.onStop();
  }
  @Override
  public void onDestroy(){
      Log.d("ChangeStateInfo","onDestroy");
      super.onDestroy();
  }
```

```
@Override
public void onRestart(){
    Log.d("ChangeStateInfo","onRestart");
    super.onRestart();
}
@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);
}
}
```

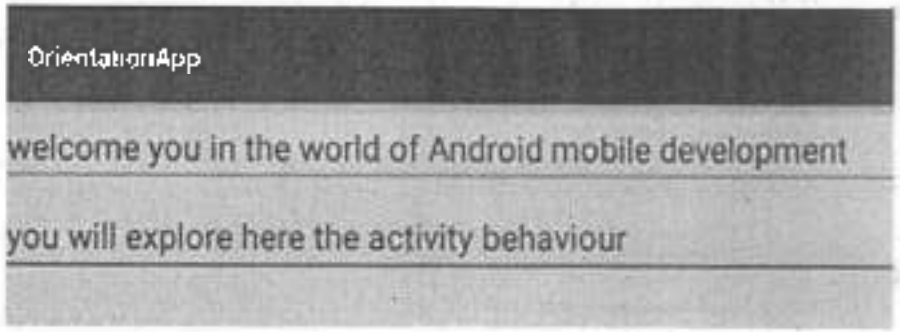
Explanation

Run the Application by clicking F11 on either Android Emulator or device. When the views are in portrait mode, it looks like the image, given below,

The state sequence is in the following order when the views are in the portrait state initially,

```
16219-16219/com.example.administrator.orientationapp E/art: Failed sending reply to debu
16219-16219/com.example.administrator.orientationapp I/art: Debugger is no longer active
16219-16219/com.example.administrator.orientationapp D/ChangeStateInfo: onCreate
16219-16219/com.example.administrator.orientationapp D/ChangeStateInfo: onStart
16219-16219/com.example.administrator.orientationapp D/ChangeStateInfo: onResume
```

As soon as the state is changed from portrait to landscape, it looks like the image, given below,

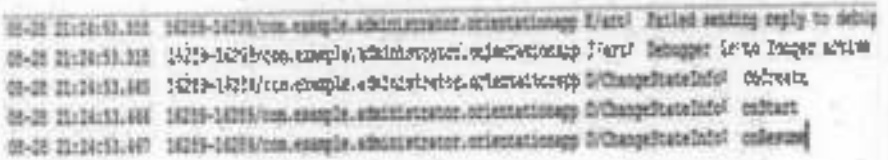


The state sequence is in the following order,

```
08-28 21:30:28.004 16219-16219/com.example.administrator.orientationapp D/ChangeStateInfo: onResume
08-28 21:30:28.005 16219-16219/com.example.administrator.orientationapp D/ChangeStateInfo: onStop
08-28 21:30:28.006 16219-16219/com.example.administrator.orientationapp D/ChangeStateInfo: onDestroy
08-28 21:30:28.030 16219-16219/com.example.administrator.orientationapp D/ChangeStateInfo: onCreate
08-28 21:30:28.032 16219-16219/com.example.administrator.orientationapp D/ChangeStateInfo: onStart
08-28 21:30:28.033 16219-16219/com.example.administrator.orientationapp D/ChangeStateInfo: onResume
```

From the above output, we see that the activity is destroyed or resumed when the device changes the orientations from landscape to portrait or vice versa.

Afterward, it again recreates.



Thus, it is very important to understand the behavior because you need to ensure that you take the necessary steps to preserve the state of your activity before it changes orientation. Suppose, your activity may have the variables that contain the values, required for some calculations in the activity. For any activity, you will have to save whatever state you need to save in the onResume() method, which is fired every time the activity changes an orientation.

UTILIZING THE ACTION BAR

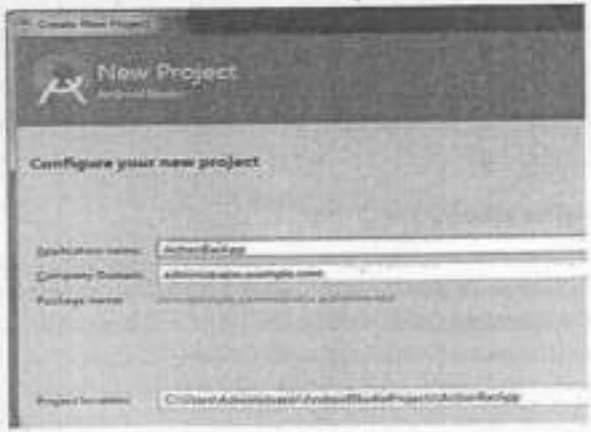
a) In Android, the Action Bar (also known as the App Bar) is a component that provides users with quick access to common actions or navigation options within an app. It typically appears at the top of an activity's layout and may contain a title, navigation buttons, action buttons, or overflow menus.

b) The action bar displays the application icon together with the activity title. On the right side of the action bar, there are action items.

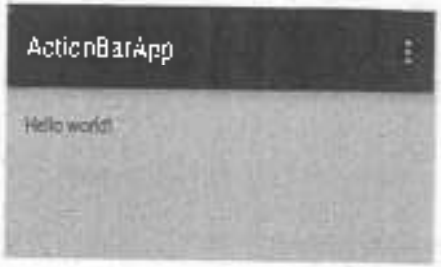
Implementation

Create a new project by selecting :

File->New->New Project.



Now, debug the application by pressing F11 on the Android emulator. Action bar located at the top of the screen will appear, which actually contains the application icon and the application name "ActionBarApp".



CREATING THE USER INTERFACE PROGRAMMATICALLY

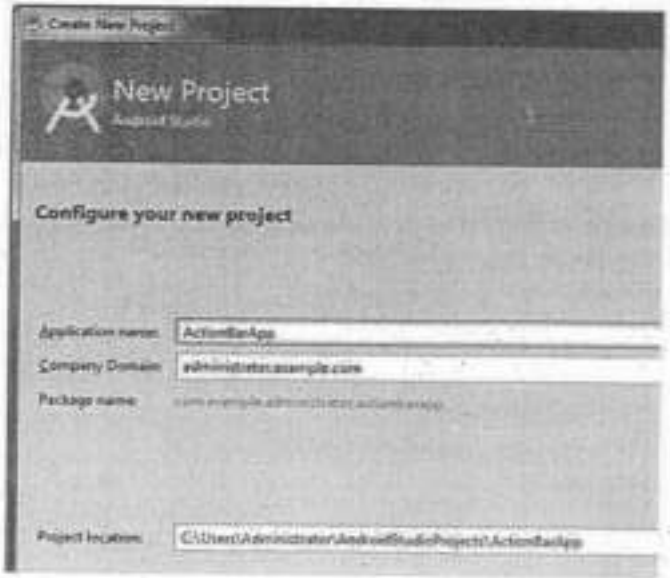
a) In Android, you can create the user interface (UI) programmatically by instantiating and configuring UI components in your Java or Kotlin code rather than using XML layout files.

b) So far, we have seen all the UIs, which you have created, using XML file but we can create the user interface programmatically. This is useful, when UI needs to be dynamically generated during runtime.

c) For example: If you are creating an app for the air ticket reservation system and your app is supposed to display the seats for each way's travel, using the buttons. In this case, you will have to dynamically generate the UI code, which is based on the air travel selected by the user.

Implementation

Create a new project by selecting File->New->New Project.



Now, add some elements which are used to create UI in Android, as shown below.

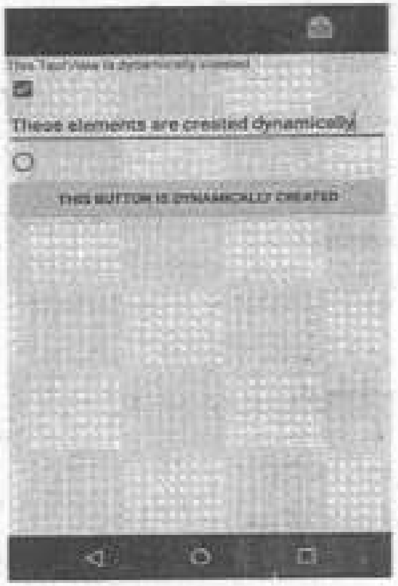
```
package com.example.administrator.actionbarapp;
import android.app.ActionBar;
import android.app.Activity;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.TextView;
import android.widget.Toast;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //getSupportActionBar().setDisplayHomeAsUpEnabled(false);
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
        //Create params for views————————
        LinearLayout.LayoutParams params = new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.FILL_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT);
        //Create a layout————————
        LinearLayout linearLayout = new LinearLayout(this);
        linearLayout.setOrientation(LinearLayout.VERTICAL);
        //—Create a TextView——
        TextView textView = new TextView(this);
        textView.setText("This TextView is dynamically created");
        textView.setLayoutParams(params);
        //—Create A EditText————————
```



```
        EditText editText = new EditText(this);
        editText.setLayoutParams(params);
        //—Create a CheckBox.————————
        CheckBox checkBox = new CheckBox(this);
        checkBox.setLayoutParams(params);
        //— Create a RadioGroup————————
        RadioGroup radioGroup = new RadioGroup(this);
        radioGroup.setLayoutParams(params);
        //——Create a RadioButton————————
        RadioButton radioButton = new RadioButton(this);
        radioButton.setLayoutParams(params);
        //—Create a Button.————
        Button button = new Button(this);
        button.setText("This Button is dynamically created");
        button.setLayoutParams(params);
        //—Add all elements to the layout
        linearLayout.addView(textView);
        linearLayout.addView(checkBox);
        linearLayout.addView(editText);
        linearLayout.addView(radioGroup);
        linearLayout.addView(radioButton);
        linearLayout.addView(button);
        //—Create a layout param for the layout————————
    LinearLayout.LayoutParams layoutParams = new
LinearLayout.LayoutParams(ActionBar.LayoutParams.FILL_PARENT,
ActionBar.LayoutParams.WRAP_CONTENT);
    this.addContentView(linearLayout, layoutParams);
    }
}
```

Explanation

Press F11 to debug the Application on the mobile device or Android emulator. The activity created looks as shown below.



LISTENING FOR UI NOTIFICATIONS

There are two levels of Android user interface with which users interact and they are as follows:

1. Activity level
2. View level

1. Activity Level

At activity level, there are certain methods in Activity class which we can override. Some of the genuine methods are as follow:

a) onKeyUp(): This is called when a key was released. This is not handled by any of the views inside the activity

b) onKeyDown(): This is called when a key was pressed. This is not handled by any of the views inside the activity.

c) onMenuItemSelected(): This is called when any item of the menu panel is pressed by user.

d) onMenuOpened(): This method is called when user opens the panel's menu.

e) The following code snippet is an instance of such an implementation:


```
boolean [illegible] ( [illegible], [illegible] event) {
    [illegible] {
    [illegible]
    [illegible]
        [illegible]);

    [illegible]
    [illegible],
        [illegible]);
    }
}
```

2. View Level

When any user interacts with a view, the corresponding view fires event. When a user touches a button or an image button or any such view we have to service the related service so that appropriate action can be performed. For this, events need to be registered.

For a button we will have code like this:

```
OnClickListener SaveListener =     OnClickListener() {
@Override
        onClick(View v) {

    Toast.makeText(getBaseContext(),
        Toast.LENGTH_SHORT).show();
}
```

DESIGNING THE USER INTERFACE USING VIEWS

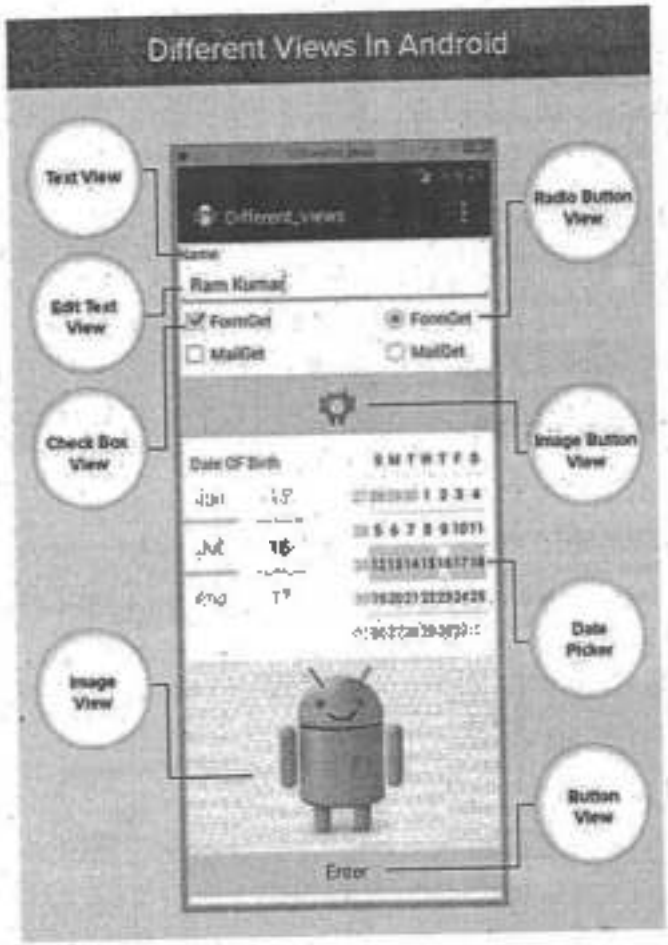
In an android application, an activity contains various views,such as textview,button,radiobutton and checkbox,which are used to design an UI. Each view represents a visual element that users can interact with.

Generally,there are 3 types of views,namely:

a) ***Basic views*** - Commonly used views such as the TextView, EditText, and Button views.

b) ***Picker views*** -Views that enable users to select from a list, such as the TimePicker and DatePicker views.

c) ***List views*** - Views that display a long list of items, such as the ListView and the SpinnerView views.



BASIC VIEWS

To get started, let's explore some of the basic views that can be used to design the UI of the android applications:-

a) TextView
b) EditText
c) Button
d) ImageButton
e) CheckBox
f) ToggleButton
g) RadioButton
h) RadioGroup

i) TextView

This is a view that displays text. It can be used to show a single line or multi-line text. It's one of the most basic and frequently used views in Android.

Android TextView Attributes:

The following are some of the commonly used attributes related to TextView control in android applications.

	Attribute	Description
1.	android: id	1. It is used to uniquely identify the control
2.	android:autoLink	2. It will automatically found and convert URLs and email addresses as clickable links.
3.	android:hint	3. It is used to display the hint text when text is empty
4.	android:text	4. It is used to display the text.
5.	android:textColor	5. It is used to change the color of the text.
6.	android:gravity	6. It is used to specify how to align the text by the view's x and y axis.
7.	android:textSize	7. It is used to specify the size of the text.

Android TextView Example

```
<TextView
    android:id="@+id/textView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Hello !!BCA Students" />
```

Output:

Hello!! BCA Students

1. Edit Text:
2. This class makes text to be editable in Android application. It helps in building the data interface taken from any user, also contains certain features through which we can hide the data which are confidential.

Android EditText Attributes

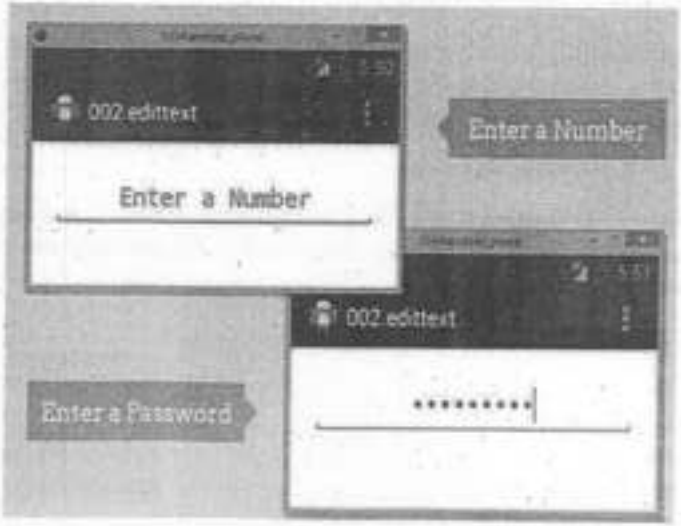
The following are some of the commonly used attributes related to EditText control in android applications.

	Attribute		Description
1.	android:id	1.	It is used to uniquely identify the control
2.	android:gravity	2.	It is used to specify how to align the text like left, right, center, top, etc.
3.	android:hint	3.	It is used to display the hint text when text is empty
4.	android:textColorHint	4.	It is used to change the text color of hint text.
5.	android:editable	5.	If we set false, EditText won't allow us to enter or modify the text
6.	android:textAllCaps	6.	It is used to present the text in all CAPS

Android EditText Example

```
<EditText
        android:id="@+id/myEdittext"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="Enter a Number"
        android:singleLine="true"
        android:inputType="textPassword" />
```

Output:



Button

1. In android, Button is a user interface control that is used to perform an action whenever the user clicks or tap on it.
2. Generally, Buttons in android will contain a text or an icon or both and perform an action when the user touches it.
3. In android, we have a different type of buttons available to use based on our requirements, those are ImageButton, ToggleButton, RadioButton.
4. In android, we can create a Button control in two ways either in the XML layout file or create it in the Activity file programmatically.
6. Following is the pictorial representation of using Buttons in android applications.

Android Button Control Attributes

Attribute	Description
1. android:id	1. It is used to uniquely identify the control
2. android:padding	2. It is used to set the padding from left, right, top and bottom.
3. android:drawableBottom	3. It's drawable to be drawn to the below of text.
4. android:background	4. It is used to set the background color for button control.
5. android:drawableRight	5. It's drawable to be drawn to the right of text.

Android Button Control Example

```
<Button
        android:id="@+id/buttonExample"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Click Here!" />
```

Output:

1. ImageButton
2. In a control of type ImageButton we can define an image to be displayed instead of a text, for which we must assign the android property: src. Normally we will assign this property with the descriptor of some resource that we have included in the / res / drawable folders this time I choose a default image from the android project.

Android ImageButton Control Attributes:

Attribute	Description
1. android:id	1. It is used to uniquely identify the control
2. android:src	2. It is used to specify the source file of an image
3. android:background	3. It is used to set the background color for an image button control.
4. android:padding	4. It is used to set the padding from left, right, top and bottom of the image button.
5. android:baseline	5. It is used to set the offset of the baseline within the view.

Android ImageButton Control Example

```
<ImageButton
    android:id="@+id/imageButton1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:src="@drawable/ic_launcher" />
```

Output:

CheckBox

a) Android CheckBox is a type of two state button either checked or unchecked.

b) Generally, we can use multiple CheckBox controls in android application to allow users to select one or more options from the set of values.

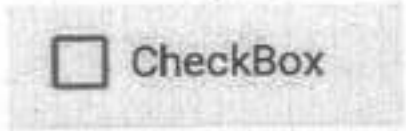
c) By default, the android CheckBox will be in the OFF (Unchecked) state. We can change the default state of CheckBox by using android:checked attribute.

Android CheckBox Control Attributes

Attribute	Description
1. android:id	1. It is used to uniquely identify the control
2. android:checked	2. It is used to specify the current state of checkbox
3. android:visibility	3. It is used to control the visibility of control.
4. android:padding	4. It is used to set the padding from left, right, top and bottom.

Android CheckBox Control Example

```
<CheckBox
    android:id="@+id/checkBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="CheckBox"
    android:checked="false"
```

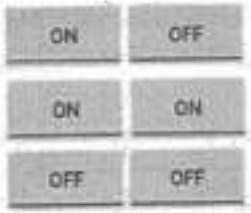


ToggleButton

a) In android, Toggle Button is a user interface control that is used to display ON (Checked) or OFF (Unchecked) states as a button with a light indicator.

b) The ToggleButton is useful for the users to change the settings between two states either ON or OFF. We can add a ToggleButton to our application layout by using the ToggleButton object.

c) Following is the pictorial representation of using ToggleButton in android applications.



Android ToggleButton Control Attributes

Attribute	Description
1. android:id	1. It is used to uniquely identify the control
2. android:checked	2. It is used to specify the current state of toggle button
3. android:textOn	3. It is used to set the text when the toggle button is in the ON / Checked state.
4. android:textOff	4. It is used to set the text when the toggle button is in the OFF / Unchecked state.

Android ToggleButton Control Example

```
<ToggleButton
    android:id="@+id/toggle1"
   android:layout_width="wrap_content"
    android:layout_height="wrap_content"
   android:layout_marginLeft="100dp"
    android:layout_marginTop="120dp"
    android:checked="true"
    android:textOff="OFF"
    android:textOn="ON"
```

Output

RadioButton

a) Like the checkbox controls, a radio button can be marked or unmarked, but in this case they are usually used within a group of options where one, and only one, of them must be marked.

Android RadioButton Control Attributes

Attribute	*Description*
1. android:id	1. It is used to uniquely identify the control
2. android:checked	2. It is used to specify the current state of radio button
3. android:onClick	3. It's the name of the method to invoke when the radio button clicked.
4. android:visibility	4. It is used to control the visibility of control.

RadioGroup

a) A RadioGroup is a ViewGroup in Android that manages a set of RadioButton views. It ensures that only one RadioButton within the group can be selected at a time.

b) It's commonly used for situations where users need to choose one option from a list of mutually exclusive choices.

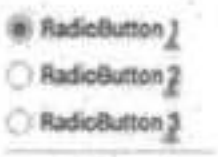
Android Radio and RadioGroup Example

```
<RadioGroup
        android:id="@+id/radioGroupExample"
```



```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >
    <RadioButton
        android:id="@+id/radioButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:text="RadioButton 1" />
<RadioButton
        android:id="@+id/radioButton2"
        android:layout_width="wrap_content"
       android:layout_height="wrap_content"
        android:text="RadioButton 2" />
    <RadioButton
        android:id="@+id/radioButton3"
        android:layout_width="wrap_content"
       android:layout_height="wrap_content"
        android:text="RadioButton 3" />
</RadioGroup>
```

Output:



Picker views

a) Picker views in Android are specialized UI components that allow users to select a value from a predefined set of options. They are commonly used for choosing dates, times, numbers, or other pre-defined values in a visually appealing and efficient way.

b) There are 2 types of picker views commonly used in Android:

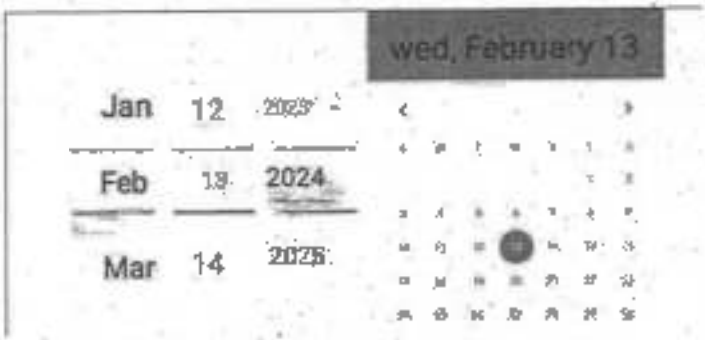
(i) Date Picker view

(ii) Time Picker view

DatePicker view

a) In android, DatePicker is a control that will allow users to select the date by a day, month and year in our application user interface.

b) Following is the pictorial representation of using a datepicker control in android applications.



Create Android DatePicker in XML Layout File

In android, we can create a DatePicker in XML layout file using <DatePicker> element with different attributes like as shown below

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

In anroid, the DatePicker supports a two types of modes, those are Calendar and Spinner to show the date details in our application.

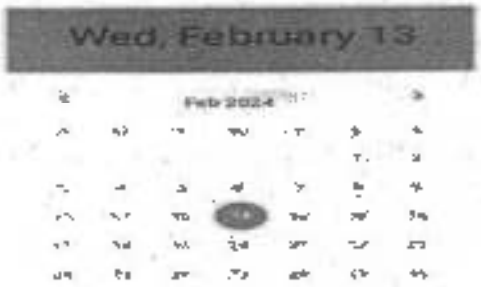
Android DatePicker with Calendar Mode

We can define android DatePicker to show only a calendar view by using DatePicker android:datePickerMode attribute.

Following is the example of showing the DatePicker in Calendar mode.

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

The above code snippet will return the DatePicker in android like as shown below



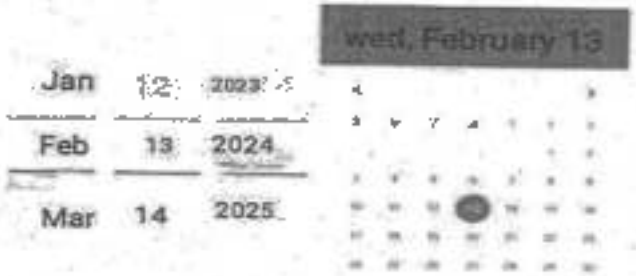
Android DatePicker with Spinner Mode

If we want to show the DatePicker in spinner format like showing day, month and year separately to select the date, then by using DatePicker android:datePickerMode attribute we can achieve this.

Following is the example of showing the DatePicker in Spinner mode.

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

The above code snippet will return the DatePicker in android like as shown below



Time Picker view

a) In android, TimePicker is a widget for selecting the time of day, in either 24-hour or AM/PM mode.

b) Generally, in android TimePicker available in two modes, one is to show the time in clock mode and another one is to show the time in spinner mode.

Create Android DatePicker in XML Layout File

In android, we can create a TimePicker in XML layout file using <TimePicker> element with different attributes like as shown below

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

In anroid, the TimePicker supports a two types of modes, those are Clock and Spinner to show the date details in our application.

Android TimePicker with Clock Mode

We can define android TimePicker to show time in clock format by using TimePicker android:timePickerMode attribute.

Following is the example of showing the TimePicker in Clock mode.

```
<TimePicker androidid="@+id/timePicker1"
        androidlayout_width="wrap_content"
        androidlayout_height="wrap_content"
        androidtimePickerMode="clock" />
```

The above code will return the TimePicker like as shown below.

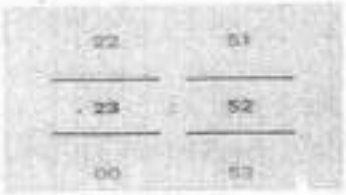
Android TimePicker with Spinner Mode

If we want to show the TimePicker in spinner format like showing hours and minutes separately to select the time, then by using TimePicker android:timePickedMode attribute we can achieve this.

Following is the example of showing the TimePicker in spinner mode.

```
<TimePicker
  androidid="@+id/datePicker1"
  androidlayout_width="wrap_content"
  androidlayout_height="wrap_content"
  android:timePickerMode="spinner"/>
```

The above code will return the TimePicker like as shown below



list view

a) Android ListView is a view which contains the group of items and displays in a scrollable list. ListView is implemented by importing android.widget.ListView class. ListView is a default scrollable which does not use other scroll view.

b) Generally, the adapter pulls data from sources such as an array or database and converts each item into a result view and that's placed into the list.

c) Following is the pictorial representation of listview in android applications.

Android ListView Example

- Following is the example of creating a ListView using arrayadapter in android application.
- Now open an activity_main.xml file from \res\layout 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"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  androidlayout_height="match_parent"
  android:orientation="vertical">
  <ListView
    android:id="@+id/userlist"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
  </ListView>
</LinearLayout>
```

Once we are done with creation of layout, now we will bind data to our ListView using ArrayAdapter, for that open main activity file MainActivity.java from \java\com.demo.listview path and write the code like as shown below.

MainActivity.java

```
package com.tutlane.listview;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
public class MainActivity extends AppCompatActivity {
  private ListView mListView;
  private ArrayAdapter aAdapter;
  private String[] users = { "Hema", "Ramya", "Nirmala", "Indira", "Meena"};
  @Override
  protected void onCreate(Bundle savedInstanceState) {
```

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

The above code will return the TimePicker like as shown below:

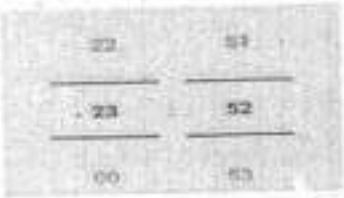
Android TimePicker with Spinner Mode

If we want to show the TimePicker in spinner format like showing hours and minutes separately to select the time, then by using TimePicker android:timePickerMode attribute we can achieve this.

Following is the example of showing the TimePicker in spinner mode.

```
<TimePicker
  android:id="@+id/datePicker1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:timePickerMode="spinner"/>
```

The above code will return the TimePicker like as shown below



list view

a) Android ListView is a view which contains the group of items and displays in a scrollable list. ListView is implemented by importing android.widget.ListView class. ListView is a default scrollable which does not use other scroll view.

b) Generally, the adapter pulls data from sources such as an array or database and converts each item into a result view and that's placed into the list.

c) Following is the pictorial representation of listview in android applications.

Android ListView Example

- Following is the example of creating a ListView using arrayadapter in android application.
- Now open an activity_main.xml file from \res\layout 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"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">
  <ListView
    android:id="@+id/userlist"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
  </ListView>
</LinearLayout>
```

Once we are done with creation of layout, now we will bind data to our ListView using ArrayAdapter, for that open main activity file MainActivity.java from \java\com.demo.listview path and write the code like as shown below.

MainActivity.java

```
package com.tutlane.listview;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
public class MainActivity extends AppCompatActivity {
  private ListView mListView;
  private ArrayAdapter aAdapter;
  private String[] users = { "Hema", "Ramya", "Nirmala", "Indira", "Meena"};
  @Override
  protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mListView = (ListView) findViewById(R.id.userlist);
        aAdapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1, users);
        mListView.setAdapter(aAdapter);
    }
}
```

Output of Android ListView Example

Spinner view (Dropdown List)

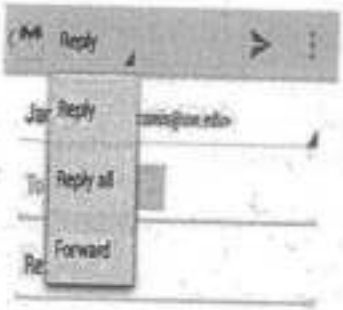
a) In Android development, a Spinner is a view that provides a drop-down menu with a set of options. Users can select one option from the menu by tapping on it, and the selected option is displayed in the Spinner's text area. Spinners are commonly used for selecting items from a predefined list of choices.

b) Create Android Spinner in XML Layout File

In android, we can create Spinner in XML layout file using <Spinner> element with different attributes like as shown below.

```
<Spinner android:id="@+id/spinner1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
```

Following is the pictorial representation of using a spinner view control in android applications.



Example

Following is the content of the main activity file src/com.example.spinner/AndroidSpinnerExampleActivity.java.

```
package com.example.spinner;
import java.util.ArrayList;
import java.util.List;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;
import android.widget.AdapterView.OnItemSelectedListener;
class AndroidSpinnerExampleActivity extends Activity implements OnItemSelectedListener{
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // Spinner element
    Spinner spinner = (Spinner) findViewById(R.id.spinner);
    // Spinner click listener
    spinner.setOnItemSelectedListener(this);
    // Spinner Drop down elements
    List<String> categories = new ArrayList<String>();
    categories.add("Automobile");
```

```
        categories.add("Business Services");
        categories.add("Computers");
        categories.add("Education");
        categories.add("Personal");
        categories.add("Travel");
        // Creating adapter for spinner
            ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, categories);
        // Drop down layout style - list view with radio button
        dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        // attaching data adapter to spinner
        spinner.setAdapter(dataAdapter);
    }
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
      // On selecting a spinner item
      String item = parent.getItemAtPosition(position).toString();
      // Showing selected spinner item
      Toast.makeText(parent.getContext(), "Selected: " + item, Toast.LENGTH_LONG).show();
    }
    public void onNothingSelected(AdapterView<?> arg0) {
      // TODO Auto-generated method stub
    }
}
```

Modify the content of res/layout/activity_main.xml to the following

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:padding="10dip"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dip"
    android:text="Category:"
    android:layout_marginBottom="5dp"/>
  <Spinner
    android:id="@+id/spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:prompt="@string/spinner_title"/>
</LinearLayout>
```

Output:

If you click on spinner button, it will a drop down menu as shown below

Specialized Fragments:

a) Specialized fragments in Android refer to fragments that are designed to serve specific purposes or functionalities within an application.

b) Fragments are modular components of an Android activity that can be combined or reused in different layouts or activities.

c) some examples of specialized fragments commonly used in Android development are :
   (i) List fragment
   (ii) Dialog fragment
   (iii) Preference fragment.

d) The basic implementation of list fragment is for creating list of items in fragments.

Let's have a look at the simple example of android ListFragment.

File type : activity_main.xml

```
<?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"
    tools:context="com.example.lenovo.listfragmentapp.MainActivity">

    <FrameLayout
        android:id="@+id/frame_layout"
        class="com.example.lenovo.listfragmentapp.FragmentA"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        />

</RelativeLayout>
```


File type : fragment1.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"
    tools:context="com.example.lenovo.listfragmentapp.FragmentA">


    <ListView
        android:id="@id/android:list"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    </ListView>

</LinearLayout>
```

```
package com.example.lenovo.listfragmentapp;

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);

    FragmentA fragment1 = new FragmentA();
    getSupportFragmentManager().beginTransaction().replace(R.id.frame_layout, fragment1).commit();
  }

}
```

File type : Fragment1.Java

```
package com.example.lenovo.listfragmentapp;
import android.os.Bundle;
import android.support.v4.app.ListFragment;
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[] code_name = {"Froyo", "GingerBread", "HoneyComb", "IceCream Sandwich",
"JellyBean", "Kitkat", "Lollipop", "Marshmallow", "Nougat", "Oreo"};
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment1, container, false);
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(inflater.getContext(),
android.R.layout.simple_list_item_1, code_name);
        setListAdapter(adapter);
        return view;
    }
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
    Toast.makeText(getActivity().getBaseContext(),                code_name[position],
Toast.LENGTH_SHORT).show();
    }
}
```

When you run the app it will look like this:

Dialog fragment

a) Dialogs are a way to show more concise information or accessibility options without creating a new activity for users.

b) DialogFragment is a utility class of android development that is used to show a Dialog window, Floating on top of an activity window in an android application.

Implementation Steps:

a) ***Create a class:*** Extend DialogFragment in your Java/Kotlin file.

b) ***Define layout:*** Design the dialog's visual layout using an XML file.

c) ***Override onCreateDialog:*** Initialize the dialog view (inflate layout, find views, set listeners).

d) ***Show the dialog:*** Call show() on the FragmentManager from your activity/fragment.

e) ***Handle user interactions:*** Implement click listeners or other event handlers for buttons, text views, etc.

f) ***Dismiss the dialog:*** Call dismiss() on the DialogFragment when appropriate (e.g., button click).

Preference fragment

a) A PreferenceFragment in Android is a specialized fragment used to display app settings or preferences in a hierarchical list. It's designed to simplify the process of creating a settings screen for your Android application.

b) PreferenceFragment provides a convenient way to manage and present preferences to the user, including simple preferences like checkboxes and text inputs, as well as more complex preferences like list preferences and multi-select preferences.

Implementation Steps:

a) ***Extend PreferenceFragment:*** Create a class that extends PreferenceFragment in your Java/Kotlin file.

b) ***Define preferences in XML:*** Create an XML file with the <PreferenceScreen> element and define child preference elements like <CheckBoxPreference>, <EditTextPreference>, etc.

c) ***Load preferences:*** In your fragment's onCreatePreferences method, inflate the XML file to load the preference structure.

d) ***Handle interactions (optional):*** Implement event listeners like onPreferenceChangeListener if you need to respond to specific preference changes programmatically.

USING IMAGE VIEWS TO DISPLAY PICTURES

Image view

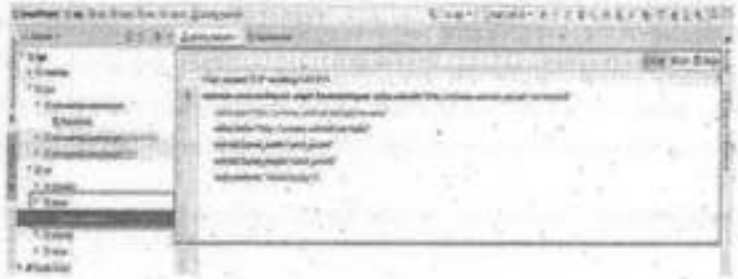
a) To display an image in an Android application, you typically use an ImageView widget.

b) An ImageView in Android Studio is a UI element that displays an image within a layout.

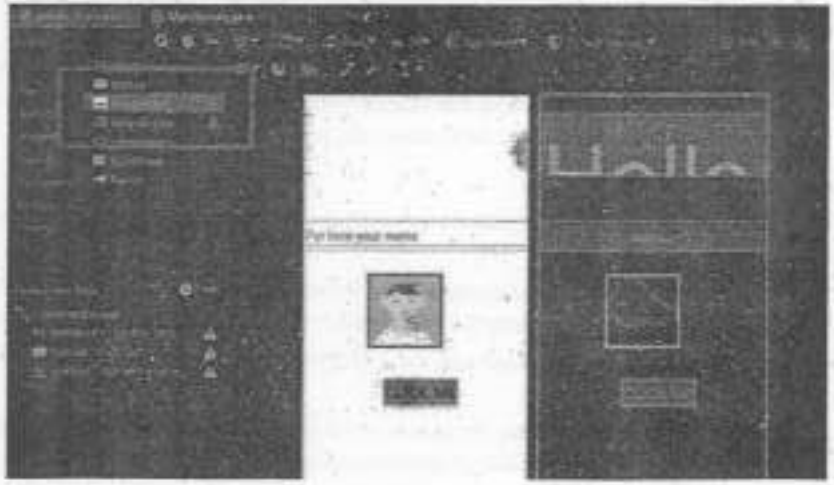
Adding an ImageView to an Activity

a) Whenever ImageView is added to an activity, it means there is a requirement for an image resource. Thus it is oblivious to provide an image file to that ImageView class. It can be done by adding an image file that is present in the Android Studio itself or we can add our own image file.

b) Open the activity_main.xml File in which the Image is to be Added



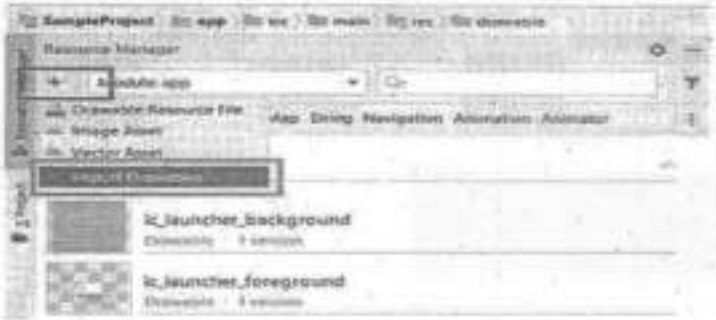
c) Switch from the Code View to the Design View of the activity_main.xml File



d) For adding an Image from Android Studio, Drag the ImageView widget to the activity area of the application, a pop-up dialogue box will open choose from the wide range of drawable resources and click "OK".

For Adding an Image File other than Android Studio Drawable Resources:

- Click on the "Resource Manager" tab on the leftmost panel and select the "Import Drawables" option OR We can also add our resources by clicking the + icon located at the corner.



Add Resource Images

Expand the res folder and add the following images to the drawable folder

Add ImageView to the XML Layout (UI):Open the xml layout file and add a ImageView from the palette.

activity_main.xml

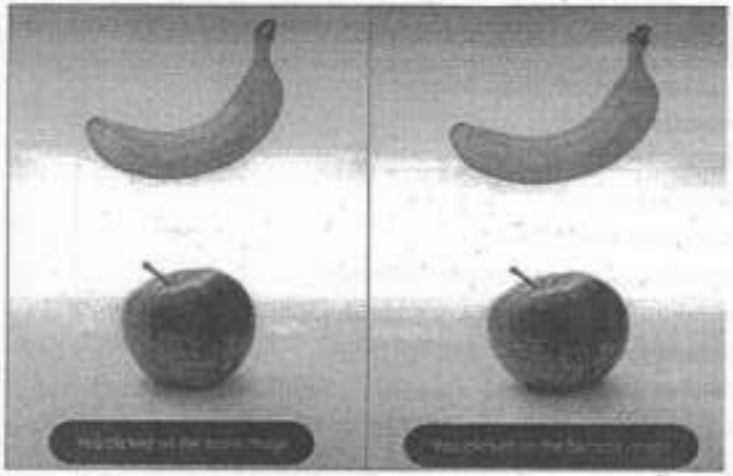
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
tools:context=".MainActivity">
<TextView
android:id="@+id/textView"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Hello Androidchunk!"
android:textSize="22sp" />
<ImageView
android:id="@+id/bananaImageView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="15dp"
android:src="@drawable/banana" />
<ImageView
android:id="@+id/appleImageView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginTop="15dp"
android:src="@drawable/apple" />
</LinearLayout>
```


Update Activity File:Open the activity file and add a click listener to the ImageView. We will display a toast when the imageview is clicked.

MainActivity.java

```
package com.androidchunk.helloimageview;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import com.androidchunk.helloimageview.databinding.ActivityMainBinding;
class MainActivity extends AppCompatActivity {
  //view binding
  private ActivityMainBinding binding;
  @Override
  protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding = ActivityMainBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());
    //banana image click listener
    binding.bananaImageView.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
          Toast.makeText(MainActivity.this, "You clicked on the banana image",
Toast.LENGTH_SHORT).show();
      }
    });
    //apple image click listener
    binding.appleImageView.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view){
          Toast.makeText(MainActivity.this, "You clicked on the apple image",
Toast.LENGTH_SHORT).show();
      }
    });
  }
}
```

Output:



ImageSwitcher

ImageSwitcher is a specialized view switcher that will provide a smooth transition animation effect to the images while switching from one image to another.

To use ImageSwitcher in our application, we need to define an XML component .xml like as shown below.

```
<ImageSwitcher android:id="@+id/imgSw"
        android:layout_width="match_parent"
        android:layout_height="250dp">
</ImageSwitcher>
```

After that we need to create an instance of ImageSwitcher and use setFactory() method to implement the ViewFactory interface to return the ImageView. We need to add the required animation effects to ImageSwitcher based on our requirements.

Android ImageSwitcher Example

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"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="10dp"
    android:paddingRight="10dp">
    <ImageSwitcher android:id="@+id/imgSw"
        android:layout_width="match_parent"
        android:layout_height="250dp"/>
    <Button
        android:id="@+id/btnPrevious"
        android:layout_below="@+id/imgSw"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Previous" android:layout_marginLeft="100dp" />
    <Button
        android:id="@+id/btnNext"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/btnPrevious"
        android:layout_toRightOf="@+id/btnPrevious"
        android:text="Next" />
</RelativeLayout>
```

File type :MainActivity.java

```
package com.tutlane.imageswitcherexample;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageSwitcher;
import android.widget.ImageView;
import android.widget.ViewSwitcher;
public class MainActivity extends AppCompatActivity {
  private Button previousbtn, nextbtn;
  private ImageSwitcher imgsw;
  private int[] images = {R.drawable.bangkok,R.drawable.bangkok2};
  private int position = 0;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    previousbtn = (Button)findViewById(R.id.btnPrevious);
    nextbtn = (Button)findViewById(R.id.btNNext);
    imgsw = (ImageSwitcher) findViewById(R.id.imgSw);
    imgsw.setFactory(new ViewSwitcher.ViewFactory() {
      @Override
      public View makeView() {
        ImageView imgVw= new ImageView(MainActivity.this);
        imgVw.setImageResource(images[position]);
        return imgVw;
      }
    });
    imgsw.setInAnimation(this, android.R.anim.slide_in_left);
    imgsw.setOutAnimation(this, android.R.anim.slide_out_right);
    previousbtn.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        if(position>0)
        position--;
        else if(position<0)
            position = 0;
        imgsw.setImageResource(images[position]);
      }
    });
    nextbtn.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        if(position<images.length)
            position++;
        if(position>=images.length)
            position = images.length-1;
```



```
        imgsw.setImageResource(images[position]);
      }
    });
  }
}
```

Output of Android ImageSwitcher Example

USING MENUS WITH VIEWS

Menus are useful for displaying additional options that are not directly visible on the main UI of an application. There are two main types of menus in Android:

a) Option menu

b) Context menu

c) Option menu

Options Menu is a primary collection of menu items for an activity and it is useful to implement actions that have a global impact on the app, such as Settings, Search, etc.

By using Options Menu, we can combine multiple actions and other options that are relevant to our current activity. We can define items for the options menu from either our Activity or Fragment class.

Options Menu Example

File type :options_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/search_item"
        android:title="Search" />
    <item android:id="@+id/upload_item"
        android:title="Upload" />
    <item android:id="@+id/copy_item"
        android:title="Copy" />
    <item android:id="@+id/print_item"
        android:title="Print" />
    <item android:id="@+id/share_item"
        android:title="Share" />
    <item android:id="@+id/bookmark_item"
        android:title="BookMark" />
</menu>
```

File type :MainActivity.java

```
package com.demo.optionsmenu;
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) {
        getMenuInflater().inflate(R.menu.options_menu, menu);
        return true;
    }
```



```
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
Toast.makeText(this, "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
        switch (item.getItemId()) {
            case R.id.search_item:
                return true;
            case R.id.upload_item:
                return true;
            case R.id.copy_item:
                return true;
            case R.id.print_item:
                return true;
            case R.id.share_item:
                return true;
            case R.id.bookmark_item:
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

Output of Android Options Menu Example

Context menu

Context Menu is like a floating menu and that appears when the user performs a long press or click on an element and it is useful to implement actions that affect the selected content or context frame.

Context Menu Example

File type : activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:id="@+id/btnShow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="long press me"
        android:layout_marginTop="200dp" android:layout_marginLeft="100dp"/>
</LinearLayout>
```

File type : MainActivity.java

```
package com.demo.contextmenuexample;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ContextMenu;
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.btnShow);
        registerForContextMenu(btn);
    }
```



```
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);
        menu.setHeaderTitle("Context Menu");
        menu.add(0, v.getId(), 0, "Upload");
        menu.add(0, v.getId(), 0, "Search");
        menu.add(0, v.getId(), 0, "Share");
        menu.add(0, v.getId(), 0, "Bookmark");
    }
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        Toast.makeText(this, "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
        return true;
    }
}
```

Output of Android Context Menu Example

WEBVIEW

1. The WebView enables you to embed a web browser in your activity. This is very useful if your application needs to embed some web content, such as maps from some other providers and so on.
2. Android WebView is used to display web page in android. The web page can be loaded from same application or URL. It is used to display online content in android activity.

3. WebView uses webkit engine to display web page.The loadUrl() and loadData() methods of Android WebView class are used to load and display web page.

WebView Example

File type : Layout XML file (activity_main.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
  <WebView
    android:id="@+id/webView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
</RelativeLayout>
```

File type : Activity Java file (MainActivity.java)

```
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import android.webkit.WebView;
import android.webkit.WebViewClient;
public class MainActivity extends AppCompatActivity {
  private WebView webView;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    webView = findViewById(R.id.webView);
    webView.setWebViewClient(new WebViewClient()); // Required to open links within the
WebView
    // Load a web page
    webView.loadUrl("https://www.google.com");
    // Alternatively, you can load HTML content directly
      // webView.loadData("<html><body><h1>Hello, World!</h1></body></html>", "text/
html", "UTF-8");
  }
  @Override
  public void onBackPressed() {
    // Go back in WebView history if possible
    if (webView.canGoBack()) {
      webView.goBack();
    } else {
      super.onBackPressed();
    }
  }
}
```

Make sure to add the necessary permissions in the AndroidManifest.xml if you're loading web content from the internet. For example:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Output:

SAVING AND LOADING USER PREFERENCES

1. One of the most Interesting Data Storage options Android provides its users is Shared Preferences. Shared Preferences is the way in which one can store and retrieve small amounts of primitive data as key/value pairs to a file on the device storage such as String, int, float, Boolean that make up your preferences in an XML file inside the app on the device storage.

2. Shared Preferences are suitable for different situations. For example, when the user's settings need to be saved or to store data that can be used in different activities within the app. As you know, onPause() will always be called before your activity is placed in the background or destroyed. So for the data to be saved persistently, it's preferred to save it in onPause(), which could be restored in onCreate() of the activity. The data stored using shared preferences are kept private within the scope of the application.
3. In order to use shared preferences, you have to call a method getSharedPreferences() that returns a SharedPreference instance pointing to the file that contains the values of preferences.

public abstract SharedPreferences getSharedPreferences (String name, int mode)

This method takes two arguments, the first being the name of the SharedPreference(SP) file and the other is the context mode that we want to store our file in.

MODE_PUBLIC will make the file public which could be accessible by other applications on the device

MODE_PRIVATE keeps the files private and secures the user's data.

MODE_APPEND is used while reading the data from the SP file.

Example to Demonstrate the use of Shared Preferences

In this example,there are two EditTexts, which save and retain the data entered earlier in them. Using Shared Preferences, the user will not have to fill in details again and again. Invoke the following code inside the activity_main.xml file to implement the UI:

```
<?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"
    tools:context="..MainActivity"
    tools:ignore="HardcodedText">
    <TextView
        android:id="@+id/textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```



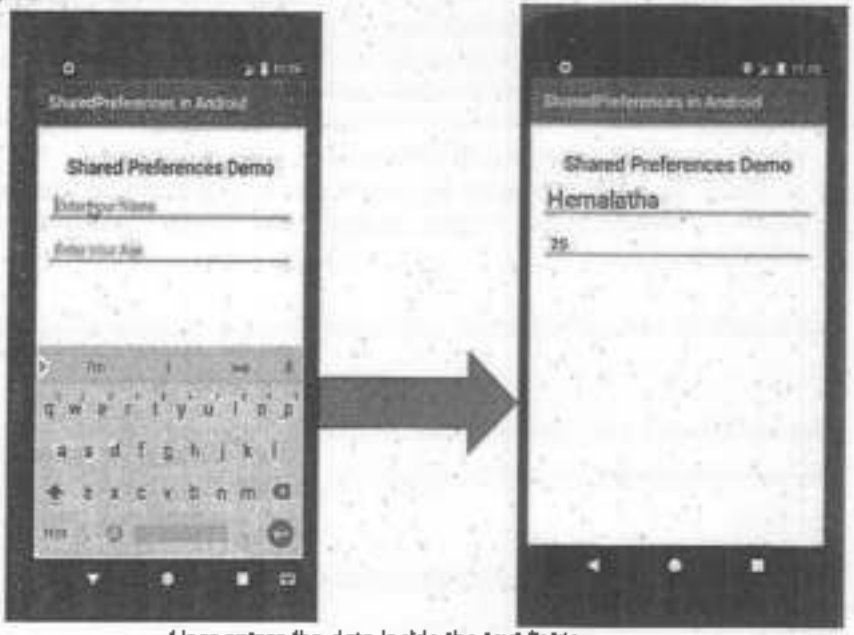
```
        android:layout_centerHorizontal="true"
        android:layout_marginTop="32dp"
        android:text="Shared Preferences Demo"
        android:textColor="@android:color/black"
        android:textSize="24sp" />
    <!--EditText to take the data from the user and save the data in SharedPreferences-->
    <EditText
        android:id="@+id/edit1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textview"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="16dp"
        android:hint="Enter your Name"
        android:padding="10dp" />
    <!--EditText to take the data from the user and save the data in SharedPreferences-->
    <EditText
        android:id="@+id/edit2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/edit1"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="16dp"
        android:hint="Enter your Age"
        android:inputType="number"
        android:padding="10dp" />
</RelativeLayout>
```

Working with the MainActivity file to handle the two of the EditText to save the data entered by the user inside the SharedPreferences.Below is the code for the MainActivity file.

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.widget.EditText;
public class MainActivity extends AppCompatActivity {
```

```
    private EditText name, age;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        name = findViewById(R.id.edit1);
        age = findViewById(R.id.edit2);
    }
    // Fetch the stored data in onResume() Because this is what will be called when the app
opens again
    @Override
    protected void onResume() {
        super.onResume();
        // Fetching the stored data from the SharedPreference
        SharedPreferences sh = getSharedPreferences("MySharedPref", MODE_PRIVATE);
        String s1 = sh.getString("name", "");
        int a = sh.getInt("age", 0);
        // Setting the fetched data in the EditTexts
        name.setText(s1);
        age.setText(String.valueOf(a));
    }
    // Store the data in the SharedPreference in the onPause() method
    // When the user closes the application onPause() will be called and data will be stored
    @Override
    protected void onPause() {
        super.onPause();
        // Creating a shared pref object with a file name "MySharedPref" in private mode
    SharedPreferences sharedPreferences = getSharedPreferences("MySharedPref",
MODE_PRIVATE);
        SharedPreferences.Editor myEdit = sharedPreferences.edit();
        // write all the data entered by the user in SharedPreference and apply
        myEdit.putString("name", name.getText().toString());
        myEdit.putInt("age", Integer.parseInt(age.getText().toString()));
        myEdit.apply();
    }
}
```


Output:



User enters the data inside the text fields

If the user closes the application completely.The application is completely destroyed.

If the user again open the application,the data entered previously,will be set for the textfields through SharedPreferences.

PERSISTING DATA TO FILES

In Android, you can persist data to files using various methods. One common approach is to use internal or external storage to save files.

Saving to Internal Storage

Saving and loading data on the internal storage is private for an application that can not be accessed by other applications. When the app is uninstalled the data stored in the internal by that app is removed.

To read and write in the android internal storage we have two methods: -

Writing file

In order to use internal storage to write some data in the file, call the openFileOutput() method with the name of the file and the mode. The mode could be private , public e.t.c.

Its syntax is given below –

```
FileOutputStream fOut = openFileOutput("file name here",MODE_WORLD_READABLE);
```

Reading file

In order to read from the file you just created , call the openFileInput() method with the name of the file. It returns an instance of FileInputStream.

Its syntax is given below –

```
FileInputStream fin = openFileInput(file);
```

Example

Following is the content of the main activity file src/MainActivity.java.

```
package com.example.demo.myapplication;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import java.io.FileInputStream;
import java.io.FileOutputStream;
public class MainActivity extends Activity {
  Button b1,b2;
  TextView tv;
  EditText ed1;
```



```
String data;
private String file = "mydata";
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  b1=(Button)findViewById(R.id.button);
  b2=(Button)findViewById(R.id.button2);
  ed1=(EditText)findViewById(R.id.editText);
  tv=(TextView)findViewById(R.id.textView2);
  b1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      data=ed1.getText().toString();
      try {
        FileOutputStream fOut = openFileOutput(file,MODE_WORLD_READABLE);
        fOut.write(data.getBytes());
        fOut.close();
Toast.makeText(getBaseContext(),"file saved",Toast.LENGTH_SHORT).show();
      }
      catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }
    }
  });
  b2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      try {
        FileInputStream fin = openFileInput(file);
        int c;
        String temp="";
        while( (c = fin.read()) != -1){
          temp = temp + Character.toString((char)c);
```

```
                }
                tv.setText(temp);
                Toast.makeText(getBaseContext(),"file read",Toast.LENGTH_SHORT).show();
            }
            catch(Exception e){
            }
        }
    });
  }
}
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
                android:layout_height="match_parent"  android:paddingLeft="@dimen/
activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  androidtpaddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">
  <TextView android:text="Internal storage" android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textview"
    android:textSize="35dp"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Example Program"
    android:id="@+id/textView"
    android:layout_below="@+id/textview"
    android:layout_centerHorizontal="true"
    android:textColor="#ff7aff24"
    android:textSize="35dp" />
  <Button
```



```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Save"
    android:id="@+id/button"
    android:layout_alignParentBottom="true"
    android:layout_alignLeft="@+id/textView"
    android:layout_alignStart="@+id/textView" />
  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:hint="Enter Text"
    android:focusable="true"
    android:textColorHighlight="#ff7eff15"
    android:textColorHint="#ffff2Se6"
    android:layout_below="@+id/imageView"
    android:layout_alignRight="@+id/textView"
    android:layout_alignEnd="@+id/textView"
    android:layout_marginTop="42dp"
    android:layout_alignLeft="@+id/imageView"
    android:layout_alignStart="@+id/imageView" />
  <ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView"
    android:src="@drawable/abc"
    android:layout_below="@+id/textView"
    android:layout_centerHorizontal="true" />
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="load"
    android:id="@+id/button2"
    android:layout_alignTop="@+id/button"
    android:layout_alignRight="@+id/editText"
```

```
    android:layout_alignEnd="@+id/editText" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Read"
    android:id="@+id/textView2"
    android:layout_below="@+id/editText"
    android:layout_toLeftOf="@+id/button2"
    android:layout_toStartOf="@+id/button2"
    android:textColor="#ff5bff1f"
    android:textSize="25dp" />
</RelativeLayout>
```

Following is the content of the res/values/string.xml.

```
<resources>
  <string name="app_name">My Application</string>
</resources>
```

Following is the content of AndroidManifest.xml file.

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


Output:

Enter some text. Press the save button. The following notification would appear on the output screen-

Now when you press the load button, the application will read the file , and display the data. In case of our, following data would be returned –

Note: We can actually view this file by switching to DDMS tab. In DDMS , select file explorer and navigate this path.

tools>android>android device Monitor

SAVING TO EXTERNAL STORAGE (SD CARD)

In Android, external storage refers to a storage location that can be accessed by the user and other applications. It includes the device's physical external storage (such as an SD card) and any other storage that can be mounted as external storage.

Example to save files on external storage:

File type: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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">
  <!-- on below line creating the heading of our application -->
  <TextView
    android:id="@+id/idTVHeading"
```



```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    android:layout_marginTop="90dp"
    android:padding="8dp"
    android:text="Save File in External Storage in Android"
    android:textAlignment="center"
    android:textColor="@color/black"
    android:textSize="20sp"
    android:textStyle="bold" />
  <!-- on below line creating an edit text for storing the data in the file -->
  <EditText
    android:id="@+id/idEdtMessage"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/idTVHeading"
    android:layout_margin="10dp"
    android:hint="Enter the data to be saved in the file.."
    android:lines="4" />
  <!-- on below line creating a button to save this image view in external storage-->
  <Button
    android:id="@+id/idBtnSave"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/idEdtMessage"
    android:layout_margin="10dp"
    android:text="Save File To External Storage"
    android:textAllCaps="false" />
  <!-- on below line creating a text view for displaying the file data -->
  <TextView
    android:id="@+id/idTVFileData"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/idBtnSave"
    android:layout_centerHorizontal="true"
```

```
        android:layout_margin="10dp"
        android:padding="4dp"
        android:text="File Data will appear here"
        android:textAlignment="center"
        android:textColor="@color/black"
        android:textSize="18sp" />
    <!-- on below line creating a button to view the file stored in external storage-->
    <Button
        android:id="@+id/idBtnView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/idTVFileData"
        android:layout_margin="10dp"
        android:text="View File Data"
        android:textAllCaps="false" />
</RelativeLayout>
```

Adding permissions in AndroidManifest.xml file

Navigate to app>AndroidManifest.xml file and add below permissions to it in manifest tag to read sms.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

File type :MainActivity.java

```
package com.example.java_test_application;
import android.app.PendingIntent;
import android.content.ContentResolver;
import android.content.Intent;
import android.content.IntentSender;
import android.database.Cursor;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.drawable.BitmapDrawable;
import android.media.MediaScannerConnection;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
```



```
import android.provider.Telephony;
import android.text.TextUtils;
import android.util.Log;
import android.util.Patterns;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import com.google.android.gms.auth.api.Auth;
import com.google.android.gms.auth.api.credentials.Credential;
import com.google.android.gms.auth.api.credentials.CredentialPickerConfig;
import com.google.android.gms.auth.api.credentials.CredentialRequest;
import com.google.android.gms.auth.api.credentials.CredentialRequestResult;
import com.google.android.gms.auth.api.credentials.Credentials;
import com.google.android.gms.auth.api.credentials.CredentialsClient;
import com.google.android.gms.auth.api.credentials.CredentialsOptions;
import com.google.android.gms.auth.api.credentials.HintRequest;
import com.google.android.gms.auth.api.credentials.IdentityProviders;
import com.google.android.gms.auth.api.signin.GoogleSignIn;
import com.google.android.gms.auth.api.signin.GoogleSignInAccount;
import com.google.android.gms.auth.api.signin.GoogleSignInClient;
import com.google.android.gms.auth.api.signin.GoogleSignInOptions;
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.api.CommonStatusCodes;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.common.api.ResolvableApiException;
import com.google.android.gms.common.api.ResultCallback;
import com.google.android.gms.common.api.Status;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
```

```
import org.w3c.dom.Text;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Date;
import java.util.Random;
public class MainActivity extends AppCompatActivity {
  // creating variable for edit text on below line.
  private EditText msgEdt;
  // creating variables for button on below line.
  private Button saveFileBtn, viewFileBtn;
  // creating variable for text view on below line to display the data from the file.
  private TextView dataTV;
  File externalFile;
  private String filename = "file.txt";
  private String filepath = "files";
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // initializing variables for text view on below line.
    saveFileBtn = findViewById(R.id.idBtnSave);
    viewFileBtn = findViewById(R.id.idBtnView);
    msgEdt = findViewById(R.id.idEditMessage);
    dataTV = findViewById(R.id.idTVFileData);
    // on below line we are checking if the external storage is available on the device and the
external storage is not read only.
    if (!isExternalStorageAvailable() || isExternalStorageReadOnly()) {
    // if the external storage is not available then we are displaying the toast message on
below line.
      Toast.makeText(this, "External storage not available on the device..",
Toast.LENGTH_SHORT).show();
```



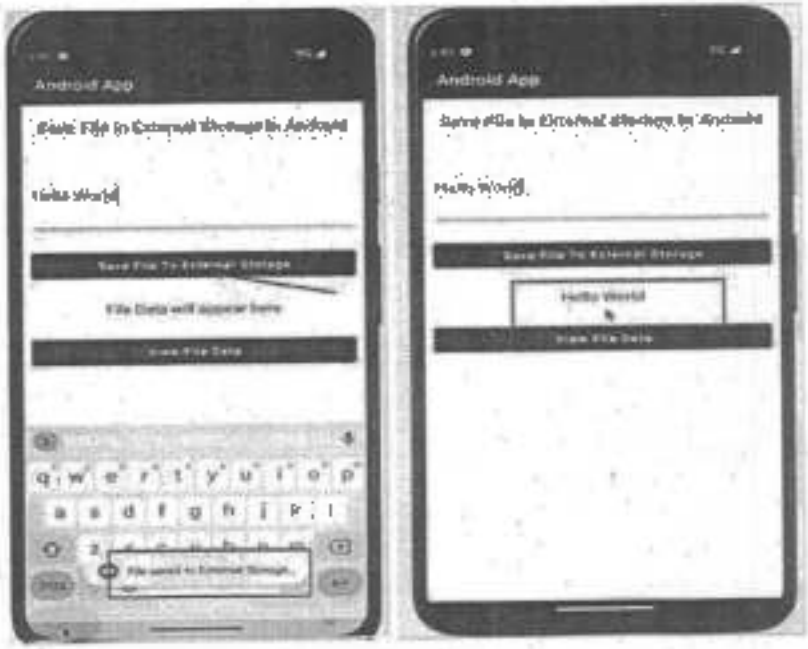
```
    } else {
      // on below line we are initializing external file variable and specifying the file path
along with file name.
      externalFile = new File(getExternalFilesDir(filepath), filename);
    }
    // on below line adding click listener for save file button.
    saveFileBtn.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        // on below line we are checking if the edit text is empty or not.
        if (msgEdt.getText().toString().isEmpty()) {
          // if the edit text is empty we are displaying a toast message.
          Toast.makeText(MainActivity.this, "Please enter your message..", Toast.LENGTH_
SHORT).show();
        } else {
          // on below line checking for the file output stream.
          try {
            // on below line creating a file output stream.
            FileOutputStream fos = new FileOutputStream(externalFile);
            // on below line writing the data from edit text in file output stream.
            fos.write(msgEdt.getText().toString().getBytes());
            // on below line closing the file output stream.
            fos.close();
            // on below line displaying toast message as file has been saved.
            Toast.makeText(MainActivity.this, "File saved to External Storage..",
Toast.LENGTH_SHORT).show();
            // on below line handling the exception.
          } catch (IOException e) {
            e.printStackTrace();
          }
        }
      }
    });
    // on below line adding click listener for view file button.
    viewFileBtn.setOnClickListener(new View.OnClickListener() {
      @Override
```

```
        public void onClick(View v) {
            // on below line creating a string variable named as file data.
            String fileData = "";
            try {
            // on below line creating a variable for file input stream and passing the file path to it.
                FileInputStream fis = new FileInputStream(externalFile);
                // on below line creating and initializing variable for data input stream.
                DataInputStream in = new DataInputStream(fis);
                // on below line creating and initializing variable for buffer reader.
                BufferedReader br = new BufferedReader(new InputStreamReader(in));
            // on below line reading the data from file and appending it to the file data variable.
                String strLine;
                while ((strLine = br.readLine()) != null) {
                    fileData = fileData + strLine;
                }
                // on below line setting data from variable to our text view.
                dataTV.setText(fileData);
                in.close();
                // on below line handling the exception.
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    });
}
// on below line creating a method to checking weather external storage is read only.
private static boolean isExternalStorageReadOnly() {
    // on below line getting external storage and checking if it is media mounted read only.
    String extStorageState = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(extStorageState)) {
        return true;
    }
    return false;
}
// on below line creating a method to check weather external storage is available or not.
```



```
private static boolean isExternalStorageAvailable() {
    // on below line checking external storage weather it is available or not.
    String extStorageState = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(extStorageState)) {
        return true;
    }
    return false;
}
}
```

Output:



In the above output,after entering text once the user press save file to external storage button,data will get saved in external storage by stating confirmation message "file saved to external storage".

REVIEW QUESTIONS

1. Define ViewGroup.Explain different types of viewgroupe supported by android.
2. Explain about adapting to display orientation.
3. a.Discuss about anchoring views.
   b.Brief about resize and repositioning.
4. Discuss about managing changes to screen orientation.
5. Explain creating the user interface programmatically.
6. Discuss about listening for UI notification.
7. Explain different types of basic views.
8. Write a note on
   a. TimePicker
   b. Date Picker
9. Explain list view and spinner view.
10. Discuss about list, dialog and preference fragment.
11. Explain imageview and imageswitcher with an example.
12. Write a note on
    a. Option menu
    b. Context menu
13. Explain about webview with an example.
14. Briefly discuss saving and loading user preferences.
15. Write a note on
    a. Saving data to internal storage
    b. Saving data to SD card.

*****

Designing User Interface

Highlights

UNIT-3

- Introduction
- Designing User Interface
- Designing by Declaration
- Creating the Opening Screen
- Using Alternate Resources
- Implementing an about Box
- Applying a Theme
- Adding a Menu
- Adding Settings
- Debugging with log Messages
- Debugging with Debugger

INTRODUCTION

In the previous chapter, you learned about the various layouts that you can use to position your views in an activity, also learned about the techniques we can use to adapt to different screen resolutions and sizes. In this chapter,will take a look at the various views that we can use to design the user interface for the different applications.

User Interface Design

User interface (UI) design or user interface engineering is the design of user interfaces for machines and software, such as computers, home appliances, mobile devices, and other electronic devices, with the focus on maximizing usability and the user experience.

In particular, we will learn about the following view groups

1. ***Basic views*** - Commonly used views such as the TextView, EditText, and Button views
2. ***Picker views*** - Views that enable users to select from a list, such as the TimePicker and DatePicker views
3. ***List views*** - Views that display a long list of items, such as the ListView and the SpinnerView views Subsequent chapters will cover the other views not covered in this chapter, such as the date and time picker views and other views for displaying graphics, etc.

UI SCREEN COMPONENTS

A typical user interface of an android application consists of action bar and the application content area.

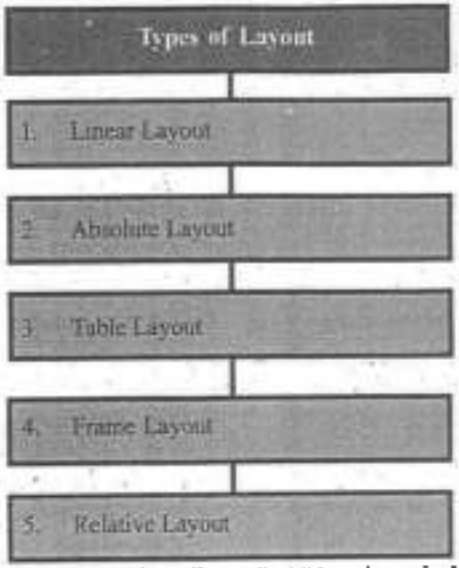
Understanding Screen Components

The basic unit of android application is the activity. A UI is defined in an xml file. During compilation, each element in the XML is compiled into equivalent Android GUI class with attributes represented by methods.

View and Viewgroups

An activity is consist of views. A view is just a widget that appears on the screen. It could be button e.t.c. One or more views can eb grouped together into one GroupView. Example of ViewGroup includes layouts

TYPES OF LAYOUT



1. ***Linear Layout:*** Is a View Group that aligns all children in a single direction, vertically or horizontally
2. ***Absolute Layout:*** allows us to specify the exact location of the child views and widgets
3. ***Table Layout:*** Is a view that groups its child views into rows and columns
4. ***Frame Layout:*** is a placeholder on screen that is used to display a single view
5. ***Realtive Layout:*** Is a ViewGroup that displays child views in relative positions

Apart form these attributes, there are other attributes that are common in all views and ViewGroups. They are listed below.

Sl. No	*View & description*
1. LAYOUT_WIDTH	Specifies the width of the View or View Group
2. LAYOUT_HEIGHT	Specifies the height of the View or View Group
3. LAYOUT_MARGIN	Top Specifies extra space on the top side of the View or View Group
4. LAYOUT_MARGIN	Bottom Specifies extra space on the bottom side of the View or ViewGroup

5. LAYOUT_MARGIN	Left Specifies extra space on the left side of the View or ViewGroup
6. LAYOUT_MARGIN	Right 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

UNITS OF MEASUREMENT

When you are specifying the size of an element on an Android UI, you should remember the following units of measurement.

Sl. No	*Unit & Description*
1. Dp	Density-independent pixel. 1 dp is equivalent to one pixel on a 160 dpi screen.
2. sp	Scale-Independent pixel. This is similar to dp and is recommended for specifying font sizes
3. pt	Point. A point is defined to be 1/72 of an inch, based on the physical screen size
4. px	Pixel. Corresponds to actual pixels on the screen

DESIGNING BY DECLARATION

Designing user interfaces by declaration involves specifying the layout, style, and behavior of UI elements using a declarative syntax or language, rather than writing procedural code to manipulate them. This approach abstracts away low-level details, making it easier to create and manage complex UIs. Here's an overview of how it works:

1. ***Declarative Syntax:*** Define UI components and their properties using a declarative syntax. This syntax could be JSON, XML, YAML, or a domain-specific language (DSL) designed for UI declaration.
2. ***Component Hierarchy:*** Describe the hierarchical structure of UI components, including their relationships and nesting. This hierarchy determines how components are organized and displayed on the screen.
3. ***Properties and Styles:*** Specify the properties and styles of UI components within the declaration. Properties may include attributes like size, position, text content, and event handlers, while styles define visual aspects such as colors, fonts, and spacing.
4. ***Reactivity and Interactivity:*** Declare how UI components react to user interactions or changes in state. This could involve defining event handlers, animations, transitions, or conditional rendering based on data or user input.
5. ***Layout Constraints:*** Define layout constraints to specify how components are arranged within their parent containers. These constraints ensure that UI elements adapt to different screen sizes and orientations.

6. ***Data Binding:*** Enable data binding to connect UI elements with underlying data models or sources. Declarative syntax can facilitate this by allowing you to specify bindings directly within the UI declaration.
7. ***Component Composition:*** Encourage component-based design by enabling the composition of complex UIs from reusable building blocks. Declarative syntax makes it easier to define and reuse components across multiple parts of an application.
8. ***Platform Agnostic:*** Ideally, the declarative UI framework should be platform-agnostic, allowing you to target multiple platforms (e.g., web, mobile, desktop) using the same declaration with minimal modifications.
9. ***Tooling Support:*** Provide tools and editors that offer syntax highlighting, code completion, and visual previews to streamline the process of designing UIs by declaration.
10. ***Integration with Frameworks:*** Integrate the declarative UI syntax with popular front-end frameworks or libraries to leverage their capabilities and ecosystem while still benefiting from the declarative approach.

Popular examples of declarative UI frameworks include React with JSX (JavaScript XML), Flutter with Dart's UI declarative syntax, SwiftUI for iOS/macOS apps, and XML-based layouts in Android development. Each of these frameworks simplifies UI development by allowing developers to describe the desired UI in a declarative manner.

CREATING THE OPENING SCREEN

Creating an open screen, also known as a splash screen, in an Android application provides users with an initial visual experience while the app loads.

The simple steps involved in creating a splash screen is as follows,

1. ***Design your splash screen layout:*** Start by designing splash screen layout. This is usually a simple layout

containing different app's logo or branding. We can use XML /java to define the layout.

The javacode to to define the layout.

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity; import android.widget.ImageView;
import android.widget.RelativeLayout;
public class SplashActivity extends AppCompatActivity { @Override
protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
// Creating the RelativeLayout
RelativeLayout relativeLayout = new RelativeLayout(this); RelativeLayout.LayoutParams layoutParams = new RelativeLayout.LayoutParams( RelativeLayout.LayoutParama.MATCH_PARENT, RelativeLayout.LayoutParams.MATCH_PARENT); relativeLayout.setLayoutParams(layoutParams);
```

```
relativeLayout.setBackgroundColor(getResources().getColor(R.color.splash_background));
// Creating the ImageView
ImageView imageView = new ImageView(this);
RelativeLayout.LayoutParams imageParams = new RelativeLayout.LayoutParams(
RelativeLayout.LayoutParams.WRAP_CONTENT,
RelativeLayout.LayoutParams.WRAP_CONTENT);
imageParams.addRule(RelativeLayout.CENTER_IN_PARENT, RelativeLayout.TRUE);
imageView.setLayoutParams(imageParams);
imageView.setImageResource(R.drawable.app_logo);
// Adding ImageView to RelativeLayout relativeLayout.addView(imageView);
// Setting the RelativeLayout as the content view of the activity
setContentView(relativeLayout);
}
}
```

Next, create a new activity for your splash screen. This activity will be displayed when the app is launched

Open the

activity as the launcher activity.

:

file and declare your splash screen activity. Additionally, you may want to set this

The XML code to Android Manifest define the layout.

```
import androidx.app.Activity; import android.content.Intent; import android.os.Bundle;
public class SplashScreenActivity extends Activity { @Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// Set the theme without action bar setTheme(R.style.AppTheme_NoActionBar);
// Set the content view for your splash screen activity setContentView(R. style);
// Assuming your actual layout file is named activity_splash_screen.xml
// You can replace with the correct layout resource ID
// Here you can add any additional setup for your splash screen activity
// For example, you might want to start another activity after a certain delay
// Intent intent = new Intent(this, NextActivity.class);
// startActivity(intent);
// finish();
}
}
```


Make sure you have defined

in the

file

We may want to customize the theme for the splash screen activity to remove the action bar or use a different style.

After completing the above steps, we can run the application to see the splash screen in action.

USING ALTERNAIVE RESOURCES

Most apps provide alternative resources to support specific device configurations. For example, the app should include alternative drawable resources for different screen densities, and alternative string resources for different languages. At runtime, Android detects the current device configuration and loads the appropriate resources.

If no resources are available for the device's specific configuration, Android uses the default resources that can include in the app the default drawables, which are in the res/drawable/ folder, the default text strings, which are in the res/values/strings.xml file, and so on.

Like default resources, alternative resources are kept in folders inside res/. Alternative-resource folders use the following naming convention:

<resource_name>-<config_qualifier>

<resource_name> is the folder name for this type of resource. For example, "drawable" or "values".

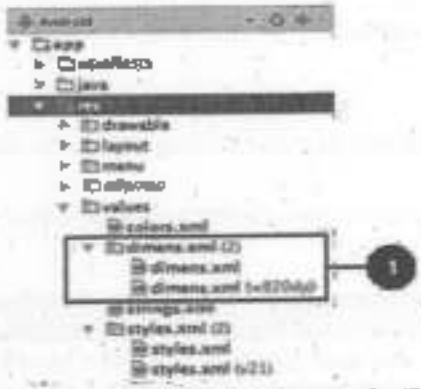
<config_qualifier> specifies a device configuration for which these resources are used.

To add multiple qualifiers to one folder name, separate the qualifiers with a dash. If you use multiple qualifiers for a resource folder, you must list them in the order

Example with multiple qualifiers:

1. Layout resources for a right-to-left layout running in "night" mode would be in a res/layout-ldrtl-night/ folder.
2. In the "Android" view in Android Studio, the qualifier is not appended to the end of the folder. Instead, the qualifier is shown as a label on the right side of the file in parentheses. For example, in the "Android" view shown below, the res/values/dimens.xml/ folder shows two files:
3. The dimens.xml file, which includes default dimension resources.

The dimens.xml (w820dp) file, which includes dimension resources for devices that have a minimum available screen width of 820dp.

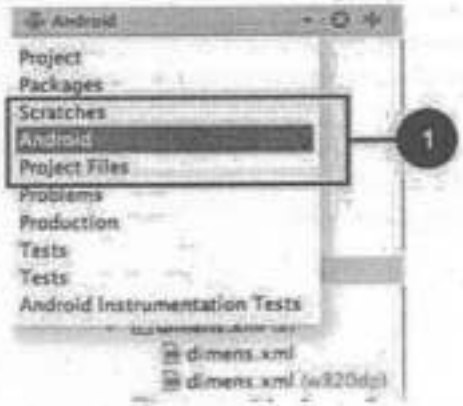


1. In the "Android" view in Android Studio, default resources for dimensions are shown in the same folder as alternative resources for dimensions.
2. In the "Project" view in Android Studio, the same information is presented differently, as shown in the screenshot below.

In the "Project" view in Android Studio, default resources for dimensions are shown in the res/values folder. Alternative resources for dimensions are shown in res/values-<qualifier> folders.

CREATING ALTERNATIVE RESOURCES

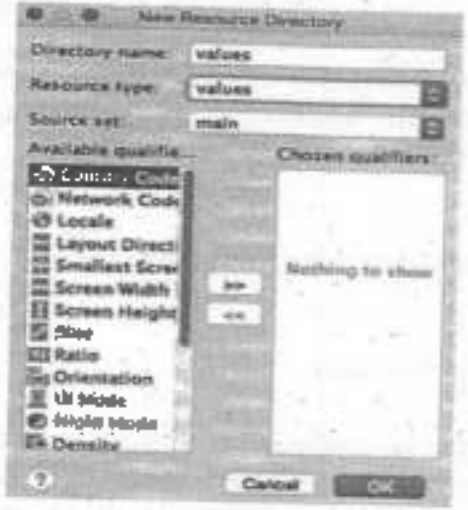
To create alternative resource folders most easily in Android Studio, use the "Android" view in the Project tool window.



1. Selecting the "Android" view in Android Studio. If you don't see these options, make sure the Project tool window is visible by selecting View > Tool Windows > Project.

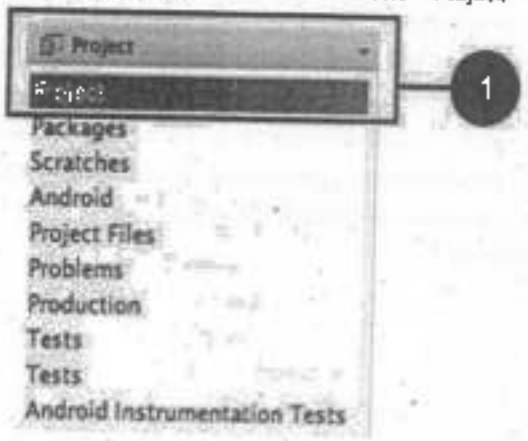
To use Android Studio to create a new configuration-specific alternative resource folder in res/ : 1.Be sure we are using the "Android" view, as shown above.

2. Right-click on the res/ folder and select New > Android resource directory. The New Resource Directory dialog box appears.



1. Select he type of resource and the qualifiers that apply to this set of alternative resources.
2. Click OK.

If we can't see the new folder in the Project tool window in Android Studio, switch to the "Project" view, as shown in the screenshot below. If we don't see these options, make sure the Project tool window is visible by selecting View > Tool Windows > Project.



COMMON ALTERNATIVE-RESOURCE QUALIFIERS

1. Screen Orientation

The screen-orientation qualifier has two possible values:

1. *port:* The device is in portrait mode (vertical). For example, res/layout-port/ would contain layout files to use when the device is in portrait mode.
2. *land:* The device is in landscape mode (horizontal). For example, res/layout-land/ would contain layout files to use when the device is in landscape mode.

If the user rotates the screen while your app is running, and if alternative resources are available, Android automatically reloads your app with alternative resources that match the new device configuration. For information about controlling how your app behaves during a configuration change, see Handling Runtime Changes.

To create variants of your layout XML file for landscape orientation and larger displays, use the layout editor. To use the layout editor:In Android Studio, open the XML file. The layout editor appears.

From the drop-down menu in the Layout Variants menu, choose an option such as Create Landscape Variant. The Layout Variants menu, which is visible when an XML file is open in Android Studio, is highlighted in the screenshot below.

A layout for a different landscape orientation appears, and a new XML file is created . For example, We might now have a file named "activity_main.xml (land)" along with your original

"activity_main.xml" file. We can use the editor to change the new layout without changing the original layout.

2. Smallest Width

The smallest-width qualifier specifies the minimum width of the device. It is the shortest of the screen's available height and width, the "smallest possible width" for the screen. The smallest width is a fixed screen-size characteristic of the device, and it does not change when the screen's orientation changes.

Specify smallest width in dp units, using the following format:

1. where <N> is the minimum width. For example, resources in a file named res/values-sw320dp/styles.xml are used if the device's screen is always at least 320dp wide.
2. It can use this qualifier to ensure that a certain layout won't be used unless it has at least <N> dps of width available to it, regardless of the screen's current orientation.

Some values for common screen sizes:

1. 320, for devices with screen configurations such as
2. 240x320 ldpi (QVGA handset)
3. 320x480 mdpi (handset)
4. 480x800 hdpi (high-density handset)
5. 480, for screens such as 480x800 mdpi (tablet/handset)
6. 600, for screens such as 600x1024 mdpi (7" tablet)
7. 720, for screens such as 720x1280 mdpi (10" tablet)

When the application provides multiple resource folders with different values for the smallest-width qualifier, the system uses the one closest to (without exceeding) the device's smallest width.

Example:

res/values-sw600dp/dimens.xml contains dimensions for images. When the app runs on a device with a smallest width of 600dp or higher (such as a tablet), Android uses the images in this folder.

3. Platform Version

The platform-version qualifier specifies the minimum API level supported by the device. For example, use v11 for API level 11 (devices with Android 3.0 or higher). See the Android API levels document for more information about these values.

Use the platform-version qualifier when the use resources for functionality that's unavailable in prior versions of Android.

a) Put default versions of the images in a res/drawable folder. These images must use an image format that's supported for all API levels, for example PNG.

b) Put WebP versions of the images in a res/drawable-v17 folder. If the device uses API level 17 or greater, Android will select these resources at runtime.

4. Localization

The localization qualifier specifies a language and, optionally, a region. This qualifier is a two-letter ISO 639-1 language code, optionally followed by a two-letter ISO 3166-1-alpha-2 region code (preceded by lowercase r).

a) We can specify a language alone, but not a region alone. Examples: res/values-fr-rFR/strings.xml

b) Strings in this file are used on devices that are configured for the French language and have their region set to France.

res/mipmap-fr-rCA/

c) Icons in this folder are used on devices that are configured for the French language and have their region set to Canada.

res/layout-ja/content_main.xml

d) This layout is used on devices that are configured for the Japanese language.

e) If the user changes the language or region in the device's system settings while app is running, and if alternative resources are available, Android automatically reloads your app with alternative resources that match the new device configuration.

PROVIDING DEFAULT RESOURCES

Default Resources:

Default resources specifies the default design and content for the application. when the app runs in a locale for which we have not provided locale-specific text, Android loads the default strings from res/values/strings.xml. If this default file is absent, or if it is missing even one string that your application needs, then your app doesn't run and shows an error.Default resources have standard resource folder names (values, for example) without any qualifiers in the folder name or in parentheses after the file names.

Tip: Always provide default resources, because the app might run on a device configuration that you don't anticipate.

APPLYING A THEME

What is Theme?

A theme is a collection of attributes that's applied to an entire app, activity, or view hierarchy not just an individual view.

When we apply a theme, every view in the app or activity applies each of the theme's attributes that it supports. A theme is a collection of attributes that's applied to an entire app, activity, or view hierarchy not just an individual view. When you apply a theme, every view in the

app or activity applies each of the theme's attributes that it supports. Themes can also apply styles to non-view elements, such as the status bar and window background.

themes and Styles are declared in a style resource file in res/values/, usually named

THEMES VERSUS STYLES

Themes and styles have many similarities, but they are used for different purposes. Themes and styles have the same basic structure a key-value pair that maps attributes to resources.

A theme defines a collection of named resources that can be referenced by styles, layouts, widgets, and so on. Themes assign semantic names, like colorPrimary, to Android resources.

HOW TO APPLY A STYLE AS A THEME

We can create a theme the same way you create styles. The difference is how we apply it: Instead of applying a style

with the

attribute on a view, we apply a theme with the

attribute on either the

tag or

an tag in the file.

For example, here's how to apply the Android Support Library's Material Design "dark" theme to the whole app:

And here's how to apply the "light" theme to just one activity:

Every view in the app or activity applies the styles that it supports from those defined in the given theme. If a view supports only some of the attributes declared in the style, then it applies only those attributes and ignores the ones it doesn't support.

Customize the Default Theme

While create a project with Android Studio, it applies a Material Design theme to app by default, as defined in the

project's file.

This style extends a theme from the Support Library and includes overrides for color attributes that are used

by key UI elements, such as the app bar and the floating action button, if used. So, we can quickly customize the app's color design by updating the provided colors.

For example, file looks similar to this:

The style values are actually references to other color resources, defined in the project's file.

That's the file we edit to change the colors. See the Material Design Color Overview to improve the user experience with dynamic color and additional custom colors.

The style values are actually references to other color resources, defined in the project's file.

That's the file edit to change the colors. See the Material Design Color Overview to improve the user experience with dynamic color and additional custom colors.Once we know the colors, update the values in

ADDING A MENU

What is Menu:

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

With the help of menu, users can experience a smooth and consistent experience throughout the application. In order to use menu, we should define it in a separate XML file and use that file in our application based on our requirementswe can use menu APIs to represent user actions and other options in our android application activities.

HOW TO DEFINE MENU IN XML FILE?

Android Studio provides a standard XML format for the type of menus to define menu items. We can simply define the menu and all its items in XML menu resource instead of building the menu in the code and also load menu resource as menu object in the activity or fragment used in our android application. Here, we should create a new folder menu inside of our project directory (res/menu) to define the menu and also add a new XML file to build the menu.

Way to create menu directory and menu resource file:

1. right-click on res folder
2. and navigate to res->New->Android Resource Directory.
3. Give resource directory name as menu and resource type also menu. one directory will be created under res folder.

To create xml resource file:

1. simply right-click on menu folder
2. and navigate to New->Menu Resource File.
3. Give name of file as menu_example. One menu_example.xml file will be created under menu folder.

XML code

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http:// schemas.android.com/apk/res/android">
<item android:id="@+id/mail" android:icon="@drawable/ic_mail" android:title="@string/mail"
/>
<item android:id="@+id/upload" android:icon="@drawable/ic_upload" android:title="@string/
upload" android:showAsAction="ifRoom" />
<item android:id="@+id/share" android:icon="@drawable/ic_share" android:title="@string/
share" />
</menu>
```


<menu> It is the root element that helps in defining Menu in XML file and it also holds multiple elements.

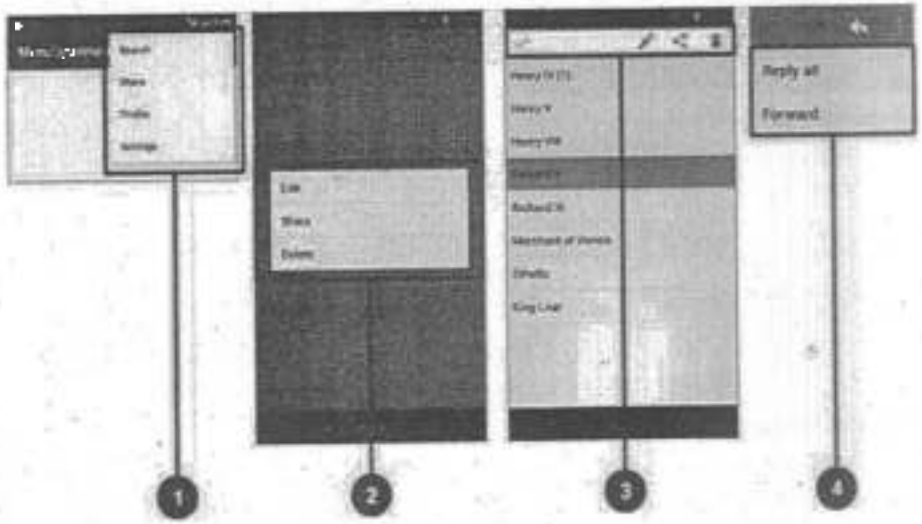
<item> It is used to create a single item in the menu. It also contains nested <menu> element in order to create a submenu.

<group> It is optional and invisible for <item> elements to categorize the menu items so they can share properties like active state, and visibility.

DIFFERENT TYPES OF MENUS

Four types of Menus available in Android are as follows

1. Options Menu
2. Context Menu
3. Popup Menu
4. Contextual action bar



Options Menu

Appears in the app bar and provides the primary options that affect use of the app itself. Examples of menu options: Search to perform a search, Share to share a link, and Settings to navigate to a Settings Activity.

Contextual Menu

Appears as a floating list of choices when the user performs a long tap on an element on The screen. Examples of menu options: Edit to edit the element, Delete to delete it, and Share to share it over

Social media

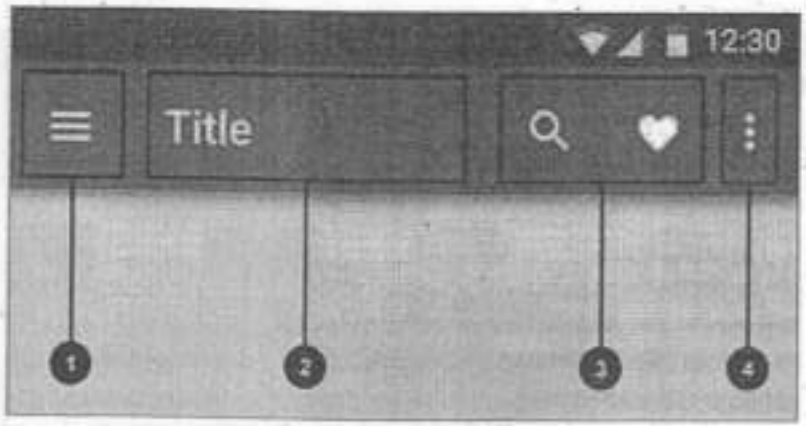
Contextual Action Bar: Appears at the top of the screen overlaying the app bar, with action items that affect the selected element or elements. Examples of menu options: Edit, Share, and Delete for one or more selected elements.

Popup Menu

Appears anchored to a View such as an ImageButton, and provides an overflow of actions or the second part of a two-part command. Example of a popup menu: the Gmail app anchors a popup menu to the app bar for the message view with Reply, Reply All, and Forward.

Options Menu

The options menu in the app bar usually provides navigation to other screens in the app, or options that affect using the app itself.The options menu appears in the right corner of the app bar. The app bar is split into four functional areas that apply to most apps, as shown in the figure below.



In the above figure:

1. *Navigation button or Up button:* Use a navigation button in this space to open a navigation drawer, or use an Up button for navigating up through your app's screen hierarchy to the parent activity. Both are described in the next chapter.
2. *Title:* The title in the app bar is the app title, or the name defined in AndroidManifest.xml by the android:label attribute for the activity.

3. *Action icons for the options menu:* Each action icon appears in the app bar and represents one of the options menu's most frequently used items. Less frequently used options menu items appear in the overflow options menu.
4. *Overflow options menu:* The overflow icon opens a popup with option menu items that are not shown as icons in the app bar.

ADDING THE OPTIONS MENU

Android provides a standard XML format to define options menu items. Instead of building the menu in your Activity code, we can define the menu and all its items in an XML menu resource. A menu resource defines an application menu (options menu, context menu, or popup menu) that can be inflated with MenuInflater, which loads the resource as a Menu object in your Activity.

If we start an app project using the Basic Activity template, the template adds the menu resource for you and inflates the options menu with MenuInflater, so you can skip this step and go right to "Defining how menu items appear".

If we are not using the Basic Activity template, use the resource-inflate design pattern, which makes it easy to create an options menu. Follow these steps (refer to the figure below):

XML menu resource. Create an XML menu resource file for the menu items, and assign appearance and position tributes as described in the next section.

Inflating the menu. Override the onCreateOptionsMenu() method in your Activity to inflate the menu.

Handling menu-item clicks. Menu items are View elements, so you can use the android:onClick attribute for each menu item. However, the onOptionsItemSelected() method can handle all the menu-item clicks in one place and determine which menu item the user clicked, which makes your code easier to understand.

Performing actions. Create a method to perform an action for each options menu item.

Heating an Xml Resource For The Menu

Steps to follow to add menu items to an XML menu resource:

Select the res folder in the Project > Android pane and choose File > New > Android resource directory. Choose menu in the Resource type drop-down menu, and click OK.

Select the new menu folder, and choose File > New > Menu resource file.

Enter the name, such as menu_main, and click OK. The new menu_main.xml file now resides within the menu folder. Open menu_main.xml and click the Text tab to show the XML code.

Add menu items using the <item ... /> tag. In this example, the item is Settings:

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

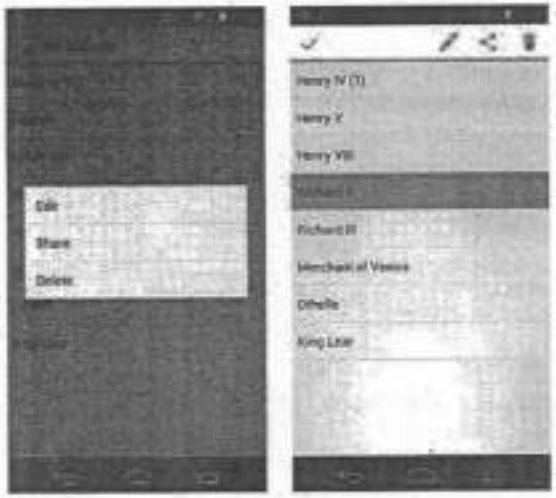
CONTEXTUAL MENUS

Use a contextual menu to allow users to take an action on a selected View. Contextual menus are most often used for items in a RecyclerView, GridView, or other view collection in which the user can perform direct actions on each item.

Android provides two kinds of contextual menus:

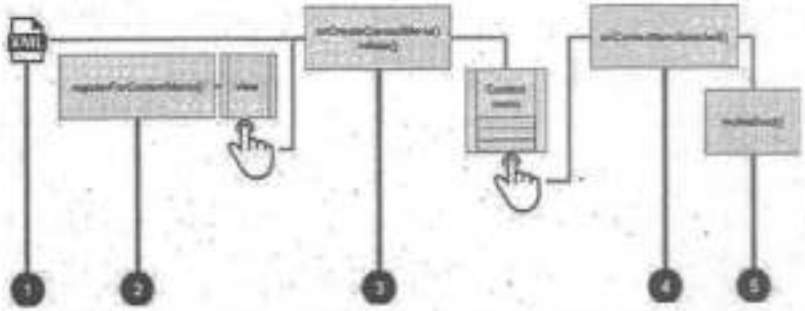
I) A context menu, shown on the left side in the figure below, appears as a floating list of menu items when the user performs a long tap on a View. It is typically used to modify the View or use it in some fashion. For example, a context menu might include Edit to edit the contents of a View, Delete to delete a View, and Share to share a View over social media. Users can perform a contextual action on one selected View at a time.

II) A contextual action bar, shown on the right side of the figure below, appears at the top of the screen in place of the app bar or underneath the app bar, with action items that affect one or more selected View elements. Users can perform an action on multiple View elements at once, if your app allows it.



STEPS FOR CREATING AND USING CONETXT MENU

The familiar resource-inflate design pattern is used to create a context menu, modified to include registering (associating) the context menu with a View. The pattern consists of the steps shown in the figure below.



Create an XML menu resource file for the menu items. Assign appearance and position attributes as described in the evious section for the options menu.

Register a View to the context menu using the registerForContextMenu() method of the Activity class. Implement the onCreateContextMenu() method in your Activity to inflate the menu.

Implement the onContextItemSelected() method in your Activity to handle menu-item clicks. Create a method to perform an action for each context menu item.

CREATING THE XML RESOURCE FILE

To create the XML menu resource directory and file, follow the steps in the previous section for the options menu. However, use a different name for the file, such as menu_context. Add the context menu items within <item ... /> tags.

For example, the following code defines the Edit menu item:

CONTEXTUAL ACTION BAR

A contextual action bar appears at the top of the screen to present actions the user can perform on a View after long- clicking the View, as shown in the figure below.

In the above figure:

1. *Contextual action bar.* The bar offers three actions on the right side (Edit, Share, and Delete) and the Done button (left arrow icon) on the left side.

   View. View on which a long-click triggers the contextual action bar.

The contextual action bar appears only when contextual action mode, a system implementation of ActionMode, occurs as a result of the user performing a long-click on one or more selected View elements.

ActionMode represents UI mode for providing alternative interaction, replacing parts of the normal UI until finished. For example, text selection is implemented as an ActionMode, as are contextual actions that work on a selected item on the screen. Selecting a section of text or long-clicking a view triggers ActionMode.While this mode is enabled, the user can select multiple items, if your app allows it. The user can also deselect items, and continue to navigate within the activity. When ActionMode is disabled, the contextual action bar disappears.

ActionMode is disabled when one of the following things occur:

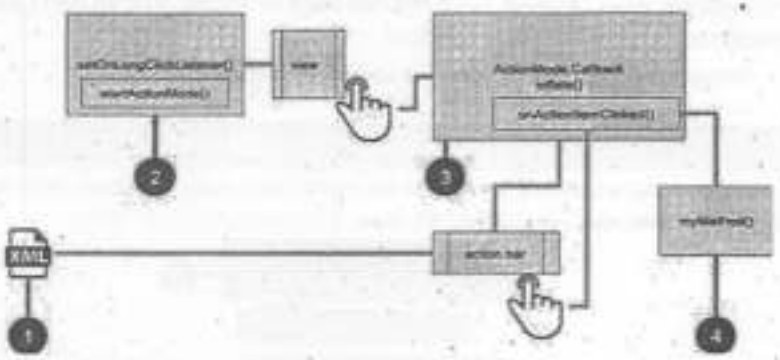
The user deselects all items.

The user presses the Back button.

The user taps Done (the left-arrow icon) on the left side of the action bar.

steps to create a contextual action bar

1. Create an XML menu resource file for the menu items, and



   assign an icon to each one (as described in a previous section).

2. Set the long-click listener using setOnLongClickListener() to the View that should trigger the contextual action bar. Call startActionMode() within the setOnLongClickListener() method when the user performs a long tap on the View.

Implement the ActionMode.Callback interface to handle the ActionMode lifecycle. Include in this interface the action for responding to a menu-item click in the onActionItemClicked() callback method.

Create a method to perform an action for each context menu item.

CREATING THE XML RESOURCE FILE

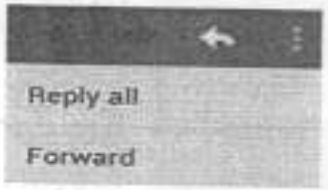
Create the XML menu resource directory and file by following the steps in the previous section on the options menu. Use a suitable name for the file, such as menu_context. Add icons for the context menu items. For example, the Edit menu item would have these attributes:

```
<item
android:id="@+id/context_edit"
android:orderInCategory="10"
android:icon="@drawable/ic_action_edit_white"
android:title="Edit" />
```

DEFINATION OF POPUP MENU

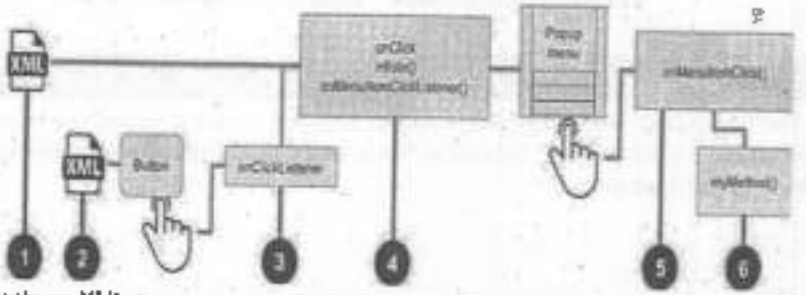
A PopupMenu is a vertical list of items anchored to a View. It appears below the anchor View if there is room, or above the View otherwise.

content in your Activity. Unlike a context menu, a popup menu is anchored to a Button, is always available, and its actions generally do not affect the content of the View.The popup menu items Reply, Reply All, and Forward are related to the email message, but don't affect or act on the message. Actions in a popup menu should not directly affect the corresponding content (use a contextual menu to directly affect selected content). As shown below, a popup can be anchored to theoverflow action button in the app bar.



HOW TO CREATE A POPUP MENU

Steps to create a popup menu (refer to figure below):



Create an XML menu resource file for the popup menu items, and assign appearance and position attributes (as described in a previous section).

Add an ImageButton for the popup menu icon in the XML activity layout file.

Assign onClickListener() to the ImageButton.

Override the onClick() method to inflate the popup menu and register it with PopupMenu.OnMenuItemClickListener.

Implement the onMenuItemClick() method.

Create a method to perform an action for each popup menu item.

Creating the Xml Resource File

Create the XML menu resource directory and file by following the steps in a previous section. Use a suitable name for the file, such as menu_popup.

ADDING SETTINGS

Typically refer to a section within the app where users can customize various preferences and configurations to tailor the app to their preferences or needs.

settings in an Android app provide users with a way to tailor their app experience to best suit their preferences and needs. It's essential for developers to design settings interfaces that are intuitive and easy to navigate, ensuring users can efficiently find and adjust the options they're looking for.

Settings can include options such as:

1. *General Settings:* These may include options for language preferences, notifications, and user interface customizations.
2. *Account Settings:* Users can manage their account information, such as profile picture, username, password, etc.
3. *Privacy and Security Setting:* Users can adjust privacy settings, permissions, and security preferences to control data access and ensure their information remains secure.
4. *Notification Settings:* Users can manage how they receive notifications from the app, including sound, vibration, and notification content.
5. *Appearance Settings:* Users can personalize the app's appearance, such as theme selection, font size, and color schemes.

6. *Advanced Settings:* These might include options for power users or advanced functionality, such as debug settings or experimental features.
7. *Data Management Settings:* Users can manage their data usage, clear cache, or control automatic data syncing.

BUILDING THE SETTINGS

Build an app's settings using various subclasses of the Preference class rather than using View elements. Preference provides the View to be displayed for each setting, and associates with it a SharedPreferences interface to store and retrieve the preference data.Each Preference subclass can be declared with an XML element that matches the class name, such as <CheckBoxPreference>.Each Preference appears as an item in a list. Direct subclasses provide containers for layouts involving multiple settings. example:

1. *PreferenceGroup:* Represents a group of settings (Preference objects).
2. *PreferenceCategory:* Provides a disabled title above a group as a section divider.
3. *PreferenceScreen:* Represents a top-level Preference that is the root of a Preference hierarchy. Use a PreferenceScreen in a layout at the top of each screen of settings.

Other Preference subclasses for settings provide the appropriate UI for users to change the setting. For example:

1. *CheckBoxPreference:* Creates a list item that shows a checkbox for a setting that is either enabled or disabled. The saved value is a boolean (true if it's checked).
2. *ListPreference:* Creates an item that opens a dialog with a list of radio buttons.
3. *SwitchPreference:* Creates a two-state option that can be toggled (such as on/off or true/false).
4. *EditTextPreference:* Creates an item that opens a dialog with an EditText element. The saved value is a String.
5. *RingtonePreference:* Lets the user choose a ringtone from those available on the device.

XML ATTRIBUTES FOR SETTINGS

XML attributes contains three different screen settings.

1. SwitchPreference toggle switch (at the top of the screen on the left side).
2. EditTextPreference text entry field (center).
3. ListPreference list of radio buttons (right).

The following example from the Settings Activity template defines a screen with three settings as shown in the figure below.

Displaying the settings

Use a specialized Activity or Fragment subclass to display a list of settings.

1. For an app that supports Android 3.0 and newer versions, the best practice for settings is to use a Settings Activity and a Fragment for each preference XML file. Add a Settings Activity class that extends Activity and hosts a fragment that extends PreferenceFragment.
2. To remain compatible with the v7 appcompat library, extend the Settings Activity with AppCompatActivity, and extend the Fragment with PreferenceFragmentCompat.
3. If your app must support versions of Android older than 3.0 (API level 10 and lower), build a special settings Activity as an extension of the PreferenceActivity class.

A Fragment like PreferenceFragment provides a more flexible architecture for the app, compared to using an Activity alone. A Fragment is like a modular section of an Activity—it has its own lifecycle and receives its own input events, and we can add or remove a Fragment while the Activity is running. Use PreferenceFragment to control the display of your settings instead of PreferenceActivity whenever possible.However, to create a two-pane layout for large screens when you have multiple groups of settings, we can use an Activity that extends PreferenceActivity and also use PreferenceFragment to display each list of settings.

Calling the settings Activity

1. If we implement the options menu with the Settings item, use the following Intent to call the Settings Activity from with the onOptionsItemSelected() method when the user taps Settings (using action_settings for the Settings menu resource id):

```
import android.content.Intent; import android.view.MenuItem;
public class YourActivity extends AppCompatActivity { @Override
public boolean onOptionsItemSelected(MenuItem item) {
```



```
// Get the ID of the selected menu item int id = item.getItemId();
// Check if the selected menu item is the settings option if (id == R.id.action_settings) {
// If it is, create an intent to start the SettingsActivity Intent intent = new Intent(this,
SettingsActivity.class); startActivity(intent); // Start the SettingsActivity
return true; // Return true to indicate that the event has been consumed
}
// If the selected menu item is not the settings option,
// delegate handling to the superclass return super.onOptionsItemSelected(item);
}
}
```

This Java code overrides the method to handle item selection in the options menu. If the selected item's ID matches the ID of the settings action (R.id.action_settings), it creates an Intent to start the

and returns superclass method.

to indicate that the event has been handled. Otherwise, it delegates the handling to the

2. If we implement a navigation drawer with the Settings item; use the following Intent to call the Settings Activity from with the onNavigationItemSelected() method when the user taps Settings (using action_settings for the Settings menu resource id):

```
@Override
public boolean onNavigationItemSelected(MenuItem item) { int id = item.getItemId();
if (id == R.id.action_settings) {
Intent intent = new Intent(this, SettingsActivity.class); startActivity(intent);
} else {
// Handle other navigation drawer items here.
}
return true; }
```

This code show that we have imported the necessary classes like MenuItem, Intent, and SettingsActivity. Also, remember to handle other navigation drawer items within the block as indicated by the comment.

READING THE SETTINGS VALUES

Each Preference you add has a corresponding key/value pair that the system uses to save the setting in a default SharedPreferences file for your app's settings. When the user changes a setting, the system updates the corresponding value in the SharedPreferences file for you. The only time you should directly interact with the associated SharedPreferences file is when you need to read the value in order to determine your app's behavior based on the user's setting.

All of an app's preferences are saved by default to a file that is accessible from anywhere within the app by calling the static method PreferenceManager.getDefaultSharedPreferences(). This method takes the context and returns the SharedPreferences object containing all the key/value pairs that are associated with the Preference objects.

DEBUGGING WITH LOG MESSAGES

Debugging: Debugging is the process of finding and fixing errors (bugs) or unexpected behavior in your code. All code has bugs, from incorrect behavior in your app, to behavior that excessively consumes memory or network resources, to actual app freezing or crashing.

Bugs can result for many reasons:

1. Errors in your design or implementation.
2. Android framework limitations (or bugs).
3. Missing requirements or assumptions for how the app should work.
4. Device limitations (or bugs).

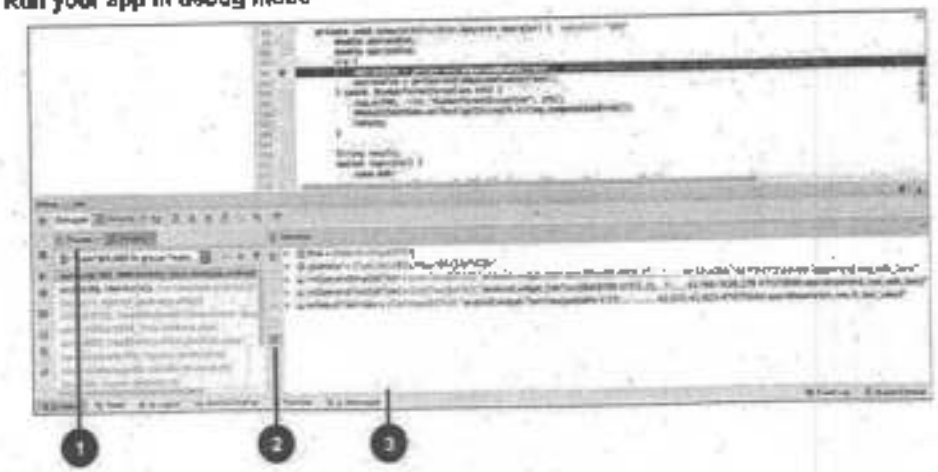
Using the debugging, testing, and profiling capabilities in Android Studio to help to reproduce, find, and resolve all of these problems. Those capabilities include:

1. The Logcat pane for log messages.
2. The Debugger pane for viewing frames, threads, and variables.
3. Debug mode for running apps with breakpoints.
4. Test frameworks such as JUnit or Espresso.
5. Dalvik Debug Monitor Server (DDMS), to track resource usage.

RUNNING THE DEBUGGER

Running an app in debug mode is similar to running the app. You can either run an app in debug mode, or attach the debugger to an already-running app.

Run your app in debug mode:



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

FEATURES OF DEBUGGER

1. ***Frames tab:*** Click to show the Frames pane with the current execution stack frames for a given thread. The execution stack shows each class and method that have been called in your app and in the Android runtime, with the most recent method at the top.
2. Click the Threads tab to replace the **Frames** pane with the **Threads** pane.
3. ***Watches button:*** Click to show the Watches pane within the Variables pane, which shows the values for any variable watches you have set. Watches allow you to keep track of a specific variable in your program, and see how that variable changes as your program runs.
4. ***Variables pane:*** Shows the variables in the current scope and their values. Each variable in this pane has an expand icon to expand the list of object properties for the variable. Try expanding a variable to explore its properties.

DEBUG A RUNNING APP

If the app is already running on a device or emulator, start debugging that app with these steps:

1. Select **Run > Attach debugger to Android process** or click the **Attach** icon in the toolbar.
2. In the **Choose Process** dialog, select the process to which you want to attach the debugger.
3. By default, the debugger shows the device and app process for the current project, as well as any connected hardware devices or virtual devices on your computer. Check the **Show all processes** option to show all processes on all devices.
4. Click OK. The Debug pane appears as before.

Resume or Stop Debugging

1. To resume executing an app after debugging it, select **Run > Resume Program** or click the **Resume** icon.
2. To stop debugging your app, select **Run > Stop** or click the **Stop** icon in the toolbar.

USING BREAKPOINTS

Android Studio supports several types of breakpoints that trigger different debugging actions. The most common type is a breakpoint that pauses the execution of your app at a specified line of code. While paused, you can examine variables, evaluate expressions, then continue execution line by line to determine the causes of runtime errors.

To add a breakpoint to a line in the code, the following steps are used:

1. Locate the line of code where you want to pause execution.
2. Click in the left gutter of the editor pane at that line, next to the line numbers. A red dot appears at that line, indicating a breakpoint. The red dot includes a check mark if the app is already running in debug mode.

3. As an alternative, you can choose Run > Toggle Line Breakpoint or press Control-F8 (Command-F8 on a Mac) to set or clear a breakpoint at a line.
4. If your app is already running, you don't need to update it to add the breakpoint.
5. If you click a breakpoint by mistake, you can undo it by clicking the breakpoint. If you clicked a line of code that is not executable, the red dot includes an "x" and a warning appears that the line of code is not executable.

When the code execution reaches the breakpoint, Android Studio pauses execution of your app. You can then use the tools in the Debug pane to view the state of the app and debug that app as it runs.

More Tools For Debugging

Android Studio and the Android SDK include a number of other tools to help you find and correct issues in your code.

System Log (Logcat Pane)

As you've learned in another chapter, you can use the Log class to send messages to the Android system log, and view those messages in Android Studio in the Logcat pane.

To write log messages in your code, use the Log class. Log messages help you understand the execution flow by collecting the system debug output while you interact with your app. Log messages can tell you what part of your app failed. For more information about logging, see Reading and Writing Logs.

Tracing and Logging

Analyzing traces allows you to see how much time is spent in certain methods, and which ones are taking the longest times.

To create the trace files, include the Debug class and call one of the startMethodTracing() methods. In the call, you specify a base name for the trace files that the system generates.

To stop tracing, call stopMethodTracing(). These methods start and stop method tracing across the entire virtual machine. For example, you could call startMethodTracing() in the onCreate() method of your Activity, and call stopMethodTracing() in the onDestroy() method of that Activity.

OTHER TOOLS

1. The Android Debug Bridge (ADB) is a command-line tool that lets to communicate with an emulator instance or connected Android-powered device.
2. The Android Profiler provides real-time data for your app's CPU, memory, and network activity. We can perform sample-based method tracing to time your code execution, capture heap dumps, view memory allocations, and inspect the details of network-transmitted files.
3. The CPU Profiler helps you inspect your app's CPU usage and thread activity in real-time, and record method traces, so we can optimize and debug your app's code.

4. The Network Profiler displays real-time network activity on a timeline, showing data sent and received, as well as the current number of connections. This lets you examine how and when your app transfers data, and optimize the underlying code appropriately.

Implementing An About Box

ABOUT BOX: typically refers to a dialog box or a screen within the application that provides users with information about the application itself. This information may include.

1. Application name
2. Version number
3. Copyright information
4. Credits for developers or contributors
5. Contact information or support links
6. Legal disclaimers or terms of use

The About Box serves as a way for users to quickly access important information about the application, its creators, and its purpose. It's often found in the settings or help section of the application's user interface. Developers include an About Box to enhance transparency and provide users with the necessary details they may need while using the application

How to Build a Reusable About box

With an Android

and its methods an About Box can be put together quickly. It can be enhanced using the class and a custom layout. With Linkify the About text any URLs (such as app help pages on the web)

and email addresses (useful for a support email link) can be made clickable.

Start by creating a new custom layout. Highlight the app or the res/layout folder in the Project explorer. Use the context menu (usually right click), or the File menu, and select New then XML and Layout XML File. For Layout File Name enter aboutbox, use ScrollView for the Root Tag and click Finish.

With aboutbox.xml open in Design mode set the ScrollView ID to aboutView, using Properties. Drop a horizontal LinearLayout on to the screen from the Layouts options in the Palette. Set the LinearLayout's layout_width and layout_height to match_parent, the ID to aboutLayout, and padding to 5dp (use

the View all properties option to change padding). Add also the LinearLayout from the Widgets Palette. (It

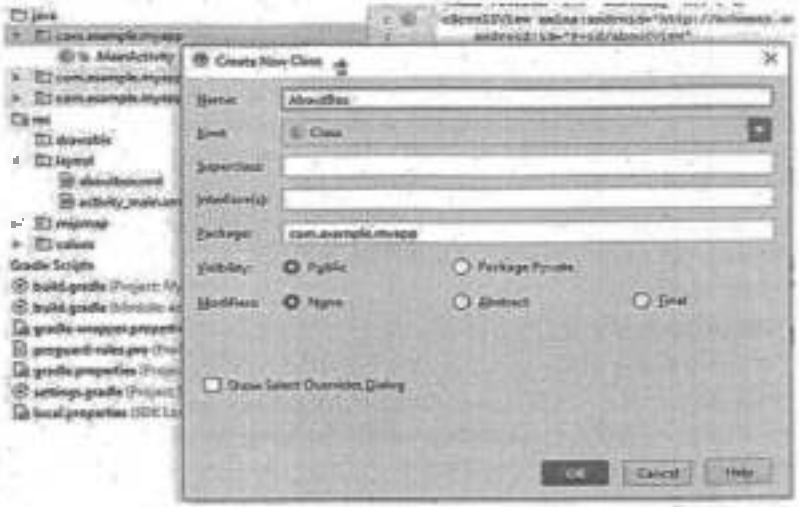
may be easier to drop the TextView on to the LinearLayout using the Component Tree.) For the TextView set the layout_width to wrap_content and layout_height to match_parent, the ID to aboutText, and the textColor to @android:color/white. The XML code for the layout should resemble the following (use this code if required):

```
import android.content.Context;import android.graphics.Color; import android.view.View
Group;
import android.widget.LinearLayout; import android.widget.ScrollView; Import
android.widget.TextView;
public class MyLayout extends ScrollView { public MyLayout(Context context) {
super(context);
// Create ScrollView
ScrollView scrollView = new ScrollView(context); scrollView.setLayoutParams(new
ViewGroup.LayoutParams( ViewGroup.LayoutParams.MATCH_PARENT, ViewGroup.
LayoutParams.MATCH_PARENT)); scrollView.setPadding(5, 5, 5, 5);
// Create LinearLayout
LinearLayout linearLayout = new LinearLayout(context); linearLayout.setLayoutParams(new
ViewGroup.LayoutParams( ViewGroup.LayoutParams.MATCH_PARENT, ViewGroup.Layout
Params.MATCH_PARENT)); linearLayout.setOrientation(LinearLayout. HORIZONTAL);linear
Layout.setBackgroundColor(Color.BLACK);
// Create TextView
TextView textView = new TextView(context);
textView.setLayoutParams(new ViewGroup.LayoutParams( ViewGroup.LayoutParams.
WRAP_CONTENT, ViewGroup.LayoutParams.MATCH_PARENT)); textView.setText("TextView");
textView.setTextColor(Color.WHITE); textView.setBackgroundColor(Color.BLACK); textView.set
Padding(5, 5, 5, 5);
// Add TextView to LinearLayout linearLayout.addView(textView);
// Add LinearLayout to ScrollView scrollView.addView(linearLayout);
// Set ScrollView as the content view of the layout this.addView(scrollView);
}
}
```

A is required for when the About text is long and the screens are small. Another advantage of the

custom layout for the About Box text is that the look of the text can be modified (here set to white with a little padding).

In the Project explorer select the same folder as MainActivity (or whatever name you used). Use the context or File menu to select New then Java Class. In the Name field enter AboutBox and click OK.



```
package com.example.myapp; import android.app.Activity; import android.content.Context;
import android.content.pm.PackageManager; import android.support.v7.app.AlertDialog;
import android.text.SpannableString;
import android.text.util.Linkify; import android.view.InflateException; import
android.view.LayoutInflater; import android.view.View;
import android.view.ViewGroup; import android.widget.TextView; public class AboutBox
{
static String VersionName(Context context)
{
try
{return context.getPackageManager().getPackageInfo(context.getPackageName(),0).version
Name;
}
catch (PackageManager.NameNotFoundException e)
{
return "Unknown";
}}
public static void Show(Activity callingActivity)
{ //Use a Spannable to allow for links highlighting
SpannableString aboutText =
newSpannableString("Version"+VersionName(callingActivity)+"\n\n" +
callingActivity.getString(R.string.about));
//Generate views to pass to AlertDialog.Builder and to set the text View about: TextView
tvAbout;
```

```
try {
//Inflate the custom view
LayoutInflater inflater = callingActivity.getLayoutInflater();
about = inflater.inflate(R.layout.aboutbox, (ViewGroup) callingActivity.findViewById
(R.id.aboutView)); tvAbout = (TextView) about.findViewById (R.id.aboutText);
catch(InflateException e) {
//Inflater can throw exception, unlikely but default to TextView if it occurs
new
//Set the about text
// Now Linkify the text
//Build and show the dialog
```

Notice that the app's icon can be shown in the About Box title using setIcon(R.mipmap.ic_launcher). String resources for the app's name and About Text are required in the usual res/values/strings.xml, like this:

```
packag import
import
import
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
new
```

To reuse this About Box drop the aboutbox.xml into a project's res/layout folder, add a new class called AboutBox, replace the class code with the AboutBox class code above. Then just call AboutBox.Show() from a button or menu click passing in a Context.

REVIEW QUESTIONS

1. What do you mean by ui?
2. Mention the different view groups.
3. Mention the ui screen componenets.
4. Mention the different types of layouts in user interface
5. What are the steps to be followed in creating alternative resources?
6. Mention the possible values in screen qualifier.
7. What is localization?
8. What is theme and how to aplly theme?
9. What is menu and how to create a menu
10. What are the different types of menu , explain them?
11. How to debug a running app in user interface
12. What is about box.how to implement the about box?

*****

Creating Your Own Content

UNIT-4

Highlights

- Introduction
- Creating Your Own Content Providers
- Using the Content Provider
- SMS Messaging
- Sending Email
- Displaying Maps
- Getting Location Data
- Monitoring a Location
- Putting SQL to Work Introducing SQLite
- In and Out of SQLite
- Hello Database
- Data Binding
- Using Cursor Provider
- Implementing Content Provider
- Reading/writing Local data
- Accessing the Internal File System
- Accessing the SD Card
- Preparing App for Publishing
- Deploying APK Files
- Uploading in Market
- Consuming Web Services Using HTTP Consuming JSON Services
- Creating Your Own Services Binding Activities to Services
- Understanding Threading

INTRODUCTION

Content providers are primarily used by other applications, which access the provider using a provider client object.Implementing a content provider has many advantages. Most importantly, you can configure a content provider to let other applications securely access and modify your app data.Content Providers are used to store and retrieve data such as contacts, images, videos, music, and other media. For example, the Android Contacts application uses a content provider to store and retrieve contacts data.

Android system allows the content provider to store the application data in several ways. Users can manage to store the application data like Images, audio, videos, and personal contact information by storing them in SQLite Database, in files, or even on a network. In order to share the data, content providers have certain permissions that are used to grant or restrict the rights to other applications to interfere with the data.

DEFINATION

Content providers are the standard interface that connects data in one process with code running in another process. A content provider manages access to a central repository of data.

A provider is part of an Android application, which often provides its own UI for working with the data., However, content providers are primarily used by other applications, which access the provider using a provider client object.

Example: Other platforms like Yahoo, Facebook, Twitter and Instagram can all be considered content providers because they are the websites that the consumer interacts with.

CONTENT URI

Content URI(Uniform Resource Identifier) is the key concept of Content providers. To access the data from a content provider, URI is used as a query string.

Structure of a Content URI: content://authority/optionalPath/optionalID

Different parts of Content URI

i. *content://* – Mandatory part of the URI as it represents that the given URI is a Content URI.

ii. *authority* – Signifies the name of the content provider like contacts, browser, etc. This part must be unique for every content provider.

iii. *optionalPath* – Specifies the type of data provided by the content provider. It is essential as this part helps content providers to support different types of data that are not related to each other like audio and video files.

iv. *optionalID* – It is a numeric value that is used when there is a need to access a particular record.

Operations in Content Provider

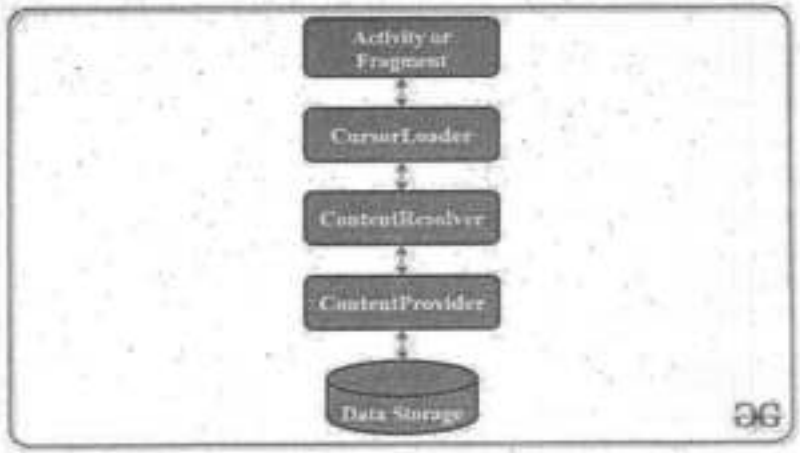
Four fundamental operations are possible in Content Provider namely Create, Read, Update, and Delete. These operations are often termed as CRUD operations.

1. *Create:* Operation to create data in a content provider.
2. *Read:* Used to fetch data from a content provider.
3. *Update:* To modify existing data.
4. *Delete:* To remove existing data from the storage.

Working of the Content Provider

UI components of android applications like Activity and Fragments use an object CursorLoader to send query requests to ContentResolver. The ContentResolver object sends requests (like create, read, update, and delete) to the ContentProvider as a client. After receiving a request, ContentProvider process it and returns the desired result. Below is a diagram to represent these processes in pictorial form.

CREATING A CONTENT PROVIDER



Steps to create a Content Provider:

1. Create a class in the same directory where the that MainActivity file resides and this class must extend the ContentProvider base class.
2. To access the content, define a content provider URI address.
3. Create a database to store the application data.
4. Implement the six abstract methods of ContentProvider class.
5. Register the content provider in AndroidManifest.xml file using <provider> tag.

Following are the six abstract methods and their description which are essential to override as the part of ContentProvider class:

Abstract Method	Description
query()	A method that accepts arguments and fetches the data from The desired table. Data is retired as a cursor object.
insert()	To Insert a new row in the database of the content provider. It returns the content URI of the inserted row.
update()	This method is used to update the fields of an existing row. It returns the number of rows updated.
delete()	This method returns the Multipurpose Internet Mail Extension(MIME) type of data to the given Content URI.
getType()	This method returns the Multipurpose Internet Mail Extension(MIME) type of data to the given Content URI.
onCreate()	As the content provider is created, the android system calls this method immediately to initialize the provider.

CREATING A CONTENT PROVIDER

Step 1: Create a new project

1. Click on File, then New => New Project.
2. Select language as Java/Kotlin.
3. Choose empty activity as a template
4. Select the minimum SDK as per your need.

Step 2: Modify the strings.xml file

All the strings used in the activity are stored here.

Step 3: Creating the Content Provider class

1. Click on File, then New => Other => ContentProvider.
2. Name the ContentProvider
3. Define authority (It can be anything for example "com.demo.user.provider")
4. Select Exported and Enabled option
5. Choose the language as Java/Kotlin

This class extends the ContentProvider base class and override the six abstract methods. Below is the complete code to define a content provider.

Step 4: Design the activity_main.xml layout

One Textview, EditText field, two Buttons, and a Textview to display the stored data will be added in the activity .

Step 5: Modify the MainActivity file

Button functionalities will be defined in this file. Moreover, the query to be performed while inserting and fetching the data.

Step 6: Modify the AndroidManifest file

The AndroidManifest file must contain the content provider name, authorities, and permissions which enable the content provider to be accessed by other applications. SMS Messaging:

This chapter demonstrate how to send messages programmatically from with an android application. Can also find out how to invoke the mail application within an android Applications to send an email messages. SMS messaging is one of the main killer applications on a mobile phone today for some users as necessary as the phone itself. Any mobile phone you buy today should have at least 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 own Android applications.

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

SENDING SMS MESSAGES PROGRAMMATICALLY

You will first learn 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

To send a text SMS over the phone using the SMSManager class in an Android application. For this, a basic knowledge of the fundamentals of android app development, such as

1. creating a new project,
2. running an android app,
3. Views, and
4. handling of click event buttons is required.

SMSManager class manages operations like sending a text message, data message, and multimedia messages (MMS). For sending a text message method sendTextMessage() is used likewise for multimedia messagesendMultimediaMessage() and for data message sendDataMessage() method is used.

Function	Description
sendTextMessage() String	sendTextMessage(String destinationAddress, scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent, long messageId)

sendDataMessage()	sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)
sendMultimediaMessage()	sendMultimediaMessage(Context context, Uri contentUri, String locationUrl, Bundle configOverrides, PendingIntent sentIntent

Steps Involved in Sending a Message

Step 1: Create a new Android Application.

Step 2: Go to AndroidManifest.xml.

Step 3: In AndroidManifest.xml add the permission to send SMS. It will permit an android application to send SMS.

<uses-permission android:name=" android.permission.SEND_SMS " />

Step 4: Open activity_main.xml from the following address and add the below code. Here, in a Linear Layout, two edittext for taking phone number and a text message and a button for sending the message is added.

app->res->layout->activitymain.xml

Step 5: Open MainActivity.java

app->java->com.example.gfg->MainActivity

Create objects for views used i.e., editTexts and button. In the onCreate method find all the views using the findViewById() method.

Viewtype object =(ViewType)findViewById(R.id.idOfView);

Since the Send button is for sending the message so onClickListener is added with the button.

Create two string variables and store the value of editText phone number and message into them using method getText() (before assigning convert them to a string using toString() method).

In try block create an instance of SMSManager class and get the SMSManager associated with the default subscription id. Now call method sendTextMessage() to send message.

SmsManager smsManager=SmsManager.getDefault();

smsManager.sendTextMessage(number,null,msg,null,null);

Then display the toast message as "Message sent" and close the try block. At last in catch block display a toast message because the message was not sent if the compiler executes this block of the code.

HOW IT WORKS

To listen for incoming SMS messages, you create a BroadcastReceiver class. The BroadcastReceiver class enables the application to receive Intents sent by other applications using the sendBroadcast() method.

Essentially, it enables the application to handle events raised by other applications. When an Intent is received, the onReceive() method is called; hence, we need to override this. When an incoming SMS message is received, the onReceive() method is fired.

The SMS message is contained in the Intent object (intent; the second parameter in the onReceive() method) via a Bundle object. The messages are stored in an Object array in the PDU format.

To extract each message, you use the static createFromPdu() method from the SmsMessage class. The SMS message is then displayed using the Toast class. The phone number of the sender is obtained via the getOriginatingAddress() method, so if you need to send an autoreply to the sender, this is the method to obtain the sender's phone number.

One interesting characteristic of the BroadcastReceiver is that you can continue to listen for incoming SMS messages even if the application is not running; as long as the application is installed on the device, any incoming SMS messages will be received by the application.

SENDING E- MAIL

Sending Email is Like SMS messaging, Android also supports e-mail. The Gmail/Email application on Android enables you to configure an e-mail account using POP3 or IMAP. Besides sending and receiving e-mails using the Gmail/Email application, you can also send e-mail messages programmatically from within your Android application.

We can send the mail with the help of intent with action as ACTION_SEND with extra fields:

- email Id to which you want to send mail,
- the subject of the email and
- body of the email.

What is Intent?

Intent is a simple message object that is used to communicate between android components such as activities, content providers, broadcast receivers, and services, here use to send the email.

STEPS INVOLVED IN SENDING AN E-MAIL

Step 1: Create a New Project in Android Studio

To create a new project in Android Studio please refer to Need to Create/Start a New Project in Android Studio. The code can be written in both Java and Kotlin Programming Language for Android.

Step 2: Working with the XML Files

Go to the activity_main.xml file, which represents the UI of the project. This file contains a Relative Layout which contains three Edit texts for receiver mail id, another for the subject of the mail, and last one for the body of the email, and three TextViews for the label and a button for starting Intent or sending mail.

The XML code to create activity_main.xml file;

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Relative Layout -->
<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" tools:context=".MainActivity">
<!-- Edit text for email id -->
<EditText android:id="@+id/editText1" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:layout_alignParentTop="true"
android:layout_alignParentRight="true" android:layout_marginTop="18dp"
android:layout_marginRight="22dp" />
<!-- Edit text for email subject -->
<EditText android:id="@+id/editText2" android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@+id/editText1" android:layout_alignLeft="@+id/editText1"
android:layout_marginTop="20dp" />
<!-- Edit text for email body -->
<EditText android:id="@+id/editText3" android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@+id/editText2" android:layout_alignLeft="@+id/editText2"
android:layout_marginTop="30dp" />
<!-- text Views for label -->
<TextView android:id="@+id/textView1" android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignBaseline="@+id/editText1" android:layout_alignBottom="@+id/editText1"
android:layout_alignParentLeft="true" android:text="Send To:" android:textColor="#0F9D58" />
```



```
<TextView android:id="@+id/textView2" android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignBaseline="@+id/editText2" android:layout_alignBottom="@+id/editText2"
android:layout_alignParentLeft="true" android:text="Email Subject:" android:textColor="#0F9D58"
/>
<TextView android:id="@+id/textView3" android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignBaseline="@+id/editText3" android:layout_alignBottom="@+id/editText3"
android:text="Email Body:" android:textColor="#0F9D58" />
<!-- Button to send email -->
<Button android:id="@+id/button"
android:layout_width="wrap_content" android:layout_height="wrap_content"
android:layout_below="@+id/editText3" android:layout_alignLeft="@+id/editText3"
android:layout_marginLeft="76dp" android:layout_marginTop="20dp" android:text="Send email!" /
>
</RelativeLayout>
```

Step 3: Working with the MainActivity File

Go to the MainActivity File and in MainActivity Intent object is created and its action is defined to ACTION_SEND to send an email, with Intent three extra fields are also added using the putExtra function. These fields are:

- Email of receiver
- Subject of email
- Body of email

setOnClickListener is attached to a button with the intent object in it to make Intent with action as ACTION_SEND to send email and Intent type as shown in code.

The code to send email through intent from the android application:

```
import android.content.Intent; import android.os.Bundle; import android.widget.Button; import
android.widget.EditText;
import androidx.appcompat.app.AppCompatActivity; public class MainActivity extends
AppCompatActivity {
// define objects for edit text and button Button button;
EditText sendto, subject, body;
@Override
protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
```

```
// Getting instance of edittext and button sendto = findViewById(R.id.editText1); subject = findViewById(R.id.editText2); body = findViewById(R.id.editText3); button = findViewById(R.id.button);
// attach setOnClickListener to button with Intent object define in it button.setOnClickListener(view -> {
String emailsend = sendto.getText().toString(); String emailsubject = subject.getText().toString(); String emailbody = body.getText().toString();
// define Intent object with action attribute as ACTION_SEND Intent Intent = new Intent(Intent.ACTION_SEND);
// add three fields to intent using putExtra function intent.putExtra(Intent.EXTRA_EMAIL, new String[]{emailsend}); intent.putExtra(Intent.EXTRA_SUBJECT, emailsubject); intent.putExtra(Intent.EXTRA_TEXT, emailbody);
// set type of intent intent.setType("message/rfc822");
// startActivity with intent with chooser as Email client using createChooser function startActivity(Intent.createChooser(intent, "Choose an Email client :"));
});
}
}
```

DISPLAYING MAPS

Maps are of great use and it increases the productivity of an app. Google Maps API allows Android developers to integrate Google Maps in their app. In addition to simply using the Maps application, We 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 our 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
4. and vice versa). Add markers to Google Maps

TYPES OF GOOGLE MAPS

There are four different types of Google maps.

1. *Normal:* This type of map displays typical road map, natural features like river and some features build by humans.

   Syntax: googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);

1. *Hybrid:* This type of map displays satellite photograph data with typical road maps. It also displays road and feature labels.

   Syntax :googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);

2. *Satellite:* Satellite type displays satellite photograph data, but doesn't display road and feature labels.

   Syntax:googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);

4. *Terrain:* This type displays photographic data. This includes colors, contour lines and labels and perspective shading.

   Syntax: googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);

3. *None:* This type displays an empty grid with no tiles loaded.

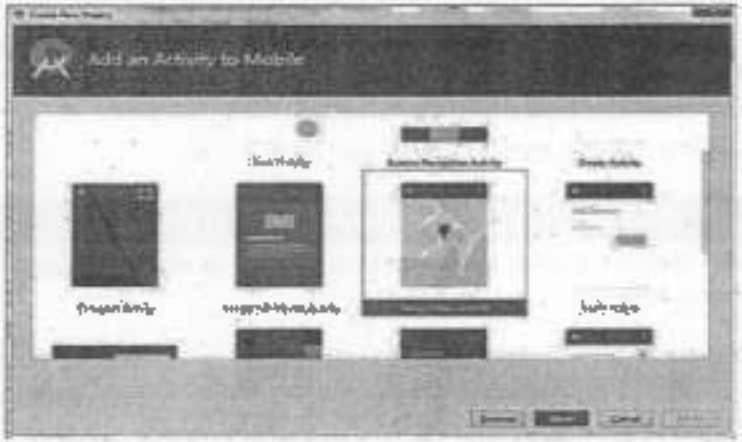
METHODS OF GOOGLE MAP

Google map API provides several methods that help to customize Google map. These methods are as following:

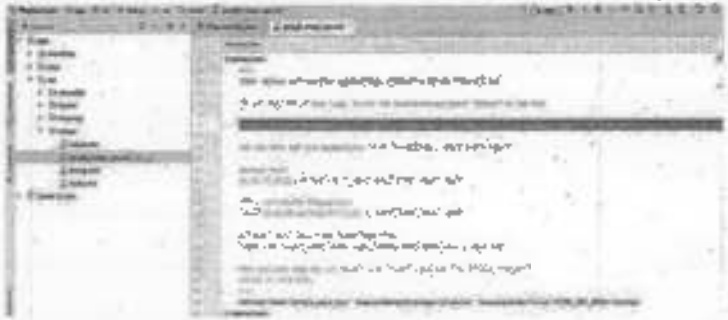
Methods	Description
addCircle(CircleOptions options)	This method add circle to map.
addPolygon(PolygonOptions options)	This method add polygon to map.
addTileOverlay(TileOverlayOptions options)	This method add tile overlay to the map.
animateCamera(CameraUpdate update)	This method moves the map according to the update with an animation
clear()	This method removes everything from the map.
getMyLocation()	This method returns the currently displayed user location.
moveCamera(CameraUpdate update)	This method reposition the camera according to the instructions defined in the update.
setTrafficEnabled(boolean enabled)	This method set the traffic layer on or off.
snapshot(GoogleMap.SnapshotReadyCallback callback)	This method takes a snapshot of the map.
stopAnimation()	This method stops the camera animation if there is any progress.

CREATING A GOOGLE MAP

1. To create maps we select google maps activity.

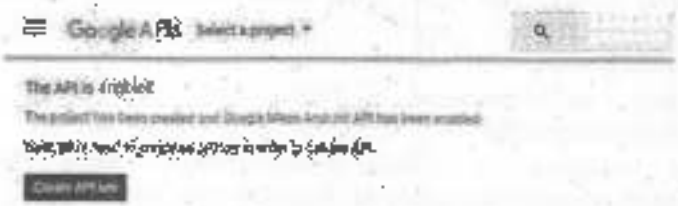


2. Copy the URL from google_map_api.xml file to generate Google map key.



3. Paste the copied URL at the browser. It will open the following page.

5. Click on Create API key to generate API key.



6. After clicking on Create API key, it will generate our API key displaying the following screen.

7. Copy this generated API key in our google_map_api.xml file

CODING FOR CREATING MAPS

<fragment xmlns:android="http://schemas.android.com/apk/res/android" xmlns: map="http://schemas.android.com/apk/res-auto" xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/map" android:name="com.google.android.gms.maps.SupportMapFragment" android:layout_width="match_parent" android:layout_height="match_parent" tools:context="example.com.mapexample.MapsActivity" />

MapsActivity.java

To get the GoogleMap object in our MapsActivity.java class we need to implement the OnMapReadyCallback interface and override the onMapReady() callback method.

```
package example.com.mapexample;
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 OnMapReady Callback{
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 use SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragment Manager()
.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)); } }
```

Required Permission

Add the following user-permission in AndroidManifest.xml file.

```
<usespermission android:name="android.permission.ACCESS_FINE_LOCATION" /
<usespermission android:name="android.permission.ACCESS_COARSE_LOCATION " />
<uses-permission android:name="android.permission.INTERNET" /> AndroidManifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="example.com.mapexample">
<!- 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. ->
<usespermission android:name="android.permission.ACCESS_FINE_LOCATION"
```



```
<usespermission android:name="android.permission.ACCESS_COARSE_LOCATIO N" />
<uses-permission android:name="android.permission.INTERNET" />
<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">
<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>
```

BUILD.GRADEL

Add the following dependencies in build.gradel file.

```
dependencies { implementation fileTree(dir: 'libs', include: ['*.jar']) implementation 'com.android.support:appcompat-v7:26.1.0' implementation 'com.google.android.gms:play-services-maps:11.8.0' testImplementation 'junit:junit:4.12' androidTestImplementation 'com.android.support.test:runner:1.0.1' androidTestImplementation 'com.android.support.test.espresso:espresso- core:3.0.1' }
```

Output

GETTING LOCATION DATA

Nowadays, mobile devices are commonly equipped with GPS receivers. Because of the many satellites orbiting the earth, we can use a GPS receiver to find your location easily. However, GPS 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 the 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.

Navigating the Map to a Specific Location Using the Location Manager Class-

1. Using the same project created in the previous section.
2. Press F11 to debug the application on the Android Emulator.
3. To simulate GPS data received by the Android Emulator, Can use the Location Controls tool located in the DDMS perspective
4. Ensure that we have first selected the emulator in the Devices tab. Then, in the Emulator Control tab, locate the Location Controls tool and select the Manual tab. Enter a latitude and longitude and click the Send button.
5. Observe that the map on the emulator now animates to another location. This proves that the application has received the GPS data.

HOW IT WORKS

In Android, location-based services are provided by the LocationManager class, located in the android .location package. Using the LocationManager class, the application can obtain periodic updates of the device's geographical locations, as well as fire an intent when it enters the proximity of a certain location.

In the MainActivity.java file,we first obtain a reference to the LocationManager class using the getSystemService() method. To be notified whenever there is a change in location, we need to register a request for location changes so that your program can be notified periodically. This is done via the requestLocationUpdates() method:

lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0,

0,

locationListener);

This method takes four parameters:

provider — The name of the provider with which you register. In this case, you are using GPS to obtain your geographical location data.

minTime — The minimum time interval for notifications, in milliseconds

minDistance — The minimum distance interval for notifications, in meters

listener — An object whose onLocationChanged() method will be called for each location update The MyLocationListener class implements the LocationListener abstract class. You need to override four methods in this implementation:

onLocationChanged(Location location) — Called when the locationhas changed

onProviderDisabled(String provider) — Called when the provider is disabled by the user

onProviderEnabled(String provider) — Called when the provider is enabled by the user

onStatusChanged(String provider, int status, Bundle extras) — Called when the provider status changes

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 so that if the user is within a five- meter radius from that location, the application will fire an intent to launch the web browser:

//—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: latitude, longitude, radius (in meters), expiration (time for the proximity alert to be valid, after which it will be deleted; -1 for no expiration), and the 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.

PUTTING SQL TO WORK

SQLite: 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.It is embedded in android by default. So, there is no need to perform any database setup or administration task. SQLiteOpenHelper class provides the functionality to use the SQLite database

SQLITE OPENHELPER CLASS

The android.database.sqlite.SQLiteOpenHelper class is used for database creation and version management. For performing any database operation, you have to provide the implementation of onCreate() and onUpgrade() methods of SQLiteOpenHelper class.

METHODS OF SQLITEOPENHELPER CLASS

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

Method	*Description*
public abstract void onCreate(SQLiteDatabase db)	called only once when database is created for the first time.
public abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)	called when database needs to be upgraded.
public synchronized void close ()	closes the database object.
public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)	called when database needs to be downgraded.

Sqlitedatabase Class:It contains methods to be performed on sqlite database such as create, update, delete, select etc.

METHODS OF SQLITEDATABASE CLASS

Method	*Description*
void execSQL(String sql)	executes the sql query not select query.
long Insert(String table, String null ColumnHack, ContentValues values)	Inserts a record on the database. The table specifies the table name, nullColumnHack doesn't allow completely null values. If second argument is null, android will store null values if values are empty. The third argument specifies the values to be stored.
Int update(String table, ContentValues values, String whereClause, String[] whereArgs)	updates a row.
Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)	returns a cursor over the resultset.

HOW TO VIEW THE DATA STORED IN SQLITE IN ANDROID STUDIO?

Following steps to view the database and its data stored in android sqlite

1. Open File Explorer.
2. Go to data directory inside data directory.
3. Search for your application package name.
4. Inside your application package go to databases where you will found your database (contactsManager).

5. Save your database (contactsManager) anywhere you like.
6. Download any SQLite browser plugins or tool (in my case DB Browser for SQLite).
7. Launch DB Browser for SQLite and open your database (contactsManager).
8. Go to Browse Data -> select your table (contacts) you will see the data stored.

Hello database: SQLite is a opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.SQLite supports all the relational database features. In order to access this database, We don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

Database – package: The main package is android.database.sqlite that contains the classes to manage our own databases

Database – Creation: In order to create a database need to call this method openOrCreateDatabase with the database name and mode as a parameter. It returns an instance of SQLite database which we have to receive in our own object.

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

FUNCTIONS AVAILABLE IN THE DATABASE PACKAGE

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

DATABASE – INSERTION

To can create a table or insert data into table using execSQL method defined in SQLiteDatabase class.

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

How to insert the values into the table database:

Sl_No	Method & Description
1	execSQL(String sql, Object[] bindArgs) This method not only insert data', but also used to update or modify already existing data in database using bind arguments

DATABASE - FETCHING

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

Syntax: Cursor resultSet = mydatbase.rawQuery("Select * from TutorialsPoint",null);

resultSet.moveToFirst();

String username = resultSet.getString(0); String password = resultSet.getString(1);

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

1 getColumnIndex(String columnName)

This method returns the index number of a column by specifying the name of the column

2 getColumnName(int columnIndex)

This method returns the name of the column by specifying the index of the column

3 getColumnNames()

This method returns the array of all the column names of the table.

4 getCount()

This method returns the total number of rows in the cursor

5 getPosition()

This method returns the current position of the cursor in the table

6 isClosed()

This method returns true if the cursor is closed and return false otherwise

DATABASE - HELPER CLASS

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

Syntax: public class DBHelper extends SQLiteOpenHelper { public DBHelper(){ super(context,DATABASE_NAME,null,1);

}

public void onCreate(SQLiteDatabase db) {}

public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}

}

Example: TO demonstrate the use of SQLite Database with steps.

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

SQLITE DATABASE STORAGE CLASSES

Storage classes refer to how stuff is stored within the database. SQLite databases store values in one of five possible storage classes:

1. NULL- For NULL values.
2. INTEGER- For integers containing as much as 8 bytes (thats from byte to long)
3. REAL- for float poiNTS.
4. TEXT- Text strings, stored using the database encoding (UTF-8 or UTF-16)
5. BLOB- Binary data, stored exactly as input.

DATA BINDING

Before we begin with data binding let us know what is data binding library.

Data binding library: The Data Binding Library is a support library that allows you to bind UI components in your layouts to data sources in your app using a declarative format rather than programmatically.

DEFINATION OF DATA BINDING

A binding class is generated for each layout file. By default, the name of the class is based on the name of the layout file, converted to Pascal case, with the Binding suffix added to it.

EXAMPLE: If the layout filename is activity_main.xml, so the corresponding generated binding class is ActivityMainBinding.This class holds all the bindings from the layout properties for example, the user variable to the layout's views and knows how to assign values for the binding expressions.

LAYOUTS AND BINDING EXPRESSIONS

Data binding layout files are slightly different and start with a root tag of layout, followed by a data element and a view root element. This view element is what your root is in a non-binding layout file.

The following code shows a sample layout file:

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
<data>
<variable name="user" type="com.example.User"/>
</data>
<LinearLayout android:orientation="vertical" android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView android:layout_width="wrap_content" android:layout_height="wrap_content"
android:text="@{user.firstName}"/>
<TextView android:layout_width="wrap_content" android:layout_height="wrap_content"
android:text="@{user.lastName}"/>
</LinearLayout>
</layout>
```

The user variable within data describes a property that can be used within this layout:

```
<variable name="user" type="com.example.User" />
```

Expressions within the layout are written in the attribute properties using the @{} syntax. In the following example, the TextView text is set to the firstName property of the user variable:

```
<TextView android:layout_width="wrap_content" android:layout_height="wrap_content"
android:text="@{user.firstName}" />
```

DATA OBJECTS

This type of object has data that never changes. It's common in apps to have data that is read once and never changes thereafter. It's also possible to use an object that follows a set of conventions, such as using accessor methods in the Java programming language

EXPRESSION LANGUAGE

Common features

The expression language looks a lot like expressions found in managed code. we can use the following operators and keywords in the expression language:

Mathematical: + - / * %

String concatenation: +

Logical: && ||

Binary: & | ^

Unary: + - ! ~

Shift: >> >>> <<

Comparison: == > < >= <= (the < needs to be escaped as <)

Instanceof

Grouping: ()

Literals, such as character, String, numeric, null

Cast

Method calls

Field access

Array access: []

Ternary operator: ?:

Missing operations

The following operations are missing from the expression syntax that you can use in managed code:

this

super

new

Explicit generic invocation

Null coalescing operator

The null coalescing operator (??) chooses the left operand if it isn't null or the right if the former is null:

Property references

An expression can reference a property in a class by using the following format, which is the same for fields, getters, and ObservableField objects:

Collections

We can access common collections, such as arrays, lists, sparse lists, and maps, using the [] operator for convenience.

STRING LITERALS

You can use single quotes to surround the attribute value, which lets you use double quotes in the expression.

Example: android:text='@{map["firstName"]}'

You can also use double quotes to surround the attribute value. When doing so, string literals must be surrounded with backticks `.

Example: android:text="@{map['firstName']}"

RESOURCES:An expression can reference app resources with the following syntax:

android:padding="@{large? @dimen/largePadding : @dimen/smallPadding}"

Some resources require explicit type evaluation, as shown in the following table:

Type	*Normal reference*	*Expression reference*
String[]	@array	@stringArray
int[]	@array	@intArray
TypedArray	@array	@typedArray
Animator	@animator	@animator
StateListAnimator	@animator	@stateListAnimator
color int	@color	@color
ColorStateList	@color	@colorStateList

EVENT HANDLING

Data binding lets you write expression handling events that are dispatched from the views—for example, the onClick() method. Event attribute names are determined by the name of the listener method, with a few exceptions. For example, View.OnClickListener has a method onClick(), so the attribute for this event is android:onClick.

There are some specialized event handlers for the click event that need an attribute other than android:onClick to avoid a conflict. You can use the following attributes to avoid these type of conflicts:

Class	*Listener setter*	*Attribute*
SearchView	setOnSearchClickListener(View.OnClickListener)	android:onSearchClick
ZoomControls	setOnZoomInClickListener(View.OnClickListener)	android:onZoomIn
ZoomControls	setOnZoomOutClickListener(View.OnClickListener)	android:onZoomOut

We can use these two mechanisms, described in detail in the sections that follow, to handle an event:

1. ***Method references:*** in your expressions, you can reference methods that conform to the signature of the listener method. When an expression evaluates to a method reference, data binding wraps the method reference and owner object in a listener and sets that listener on the target view. If the expression evaluates to null, data binding doesn't create a listener and sets a null listener instead.
2. ***Listener bindings:*** these are lambda expressions that are evaluated when the event happens. Data binding always creates a listener, which it sets on the view. When the event is dispatched, the listener evaluates the lambda expression.

Method references

We can bind events to handler methods directly, similar to the way you can assign android:onClick to a method in an activity. One advantage compared to the View onClick attribute is that the expression is processed at compile time. So, if the method doesn't exist or its signature is incorrect, you receive a compile time error.

The major difference between method references and listener bindings is that the actual listener implementation is created when the data is bound, not when the event is triggered. If you prefer to evaluate the expression when the event happens, use listener bindings.

LISTENER BINDINGS

Listener bindings are binding expressions that run when an event happens. They are similar to method references, but they let you run arbitrary data binding expressions. This feature is available with Android Gradle Plugin for Gradle version 2.0 and later.

In method references, the parameters of the method must match the parameters of the event listener. In listener bindings, only your return value must match the expected return value of the listener, unless it is expecting void.When a callback is used in an expression, data binding automatically creates the necessary listener and registers it for the event. When the view fires the event, data binding evaluates the given expression. As with regular binding expressions, you get the null and thread safety of data binding while these listener expressions are being evaluated.

IMPORTS, VARIABLES, AND INCLUDES

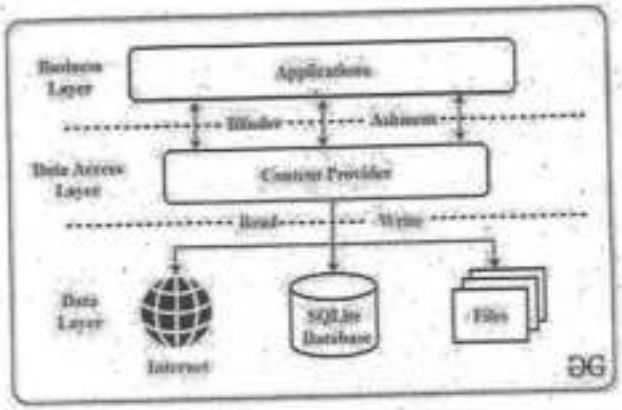
The Data Binding Library provides features such as imports, variables, and includes. Imports make easy-to-reference classes inside your layout files. Variables let you describe a property that can be used in binding expressions. Includes let you reuse complex layouts across your app.

1. ***Imports:*** Imports let you reference classes inside your layout file, like in managed code. You can use zero or more import elements inside the data element.
2. ***Type aliases:*** When there are class name conflicts, you can rename one of the classes to an alias.
3. ***Import other classes:*** can use imported types as type references in variables and expressions.
4. ***Variables:*** can use multiple variable elements inside the data element. Each variable element describes a property that can be set on the layout to be used in binding expressions within the layout file.
5. ***Includes:*** You can pass variables into an included layout's binding from the containing layout by using the app namespace and the variable name in an attribute.

USING A CONTENT PROVIDER

The best way to understand content providers is to actually use one. In Android, Content Providers are a very important component that serves the purpose of a relational database to store the data of applications. The role of the content provider in the android system is like a central

repository in which data of the applications are stored, and it facilitates other applications to securely access and modifies that data based on the user requirements. Android system allows the content provider to store the application data in several ways. Users can manage to store the application data like images, audio, videos, and personal contact information by storing them in SQLite Database, in files, or even on a network. In order to share the data, content providers have certain permissions that are used to grant or restrict the rights to other applications to interfere with the data.



PREDEFINED QUERY STRING CONSTANTS

Besides using the query URI, you can use a list of predefined query string constants in Android to specify the URI for the different data types.

For example, besides using the query content://contacts/people, we can rewrite the following statement :

Uri allContacts = Uri.parse("content://contacts/people"); using one of the predefined constants in Android, as

Uri allContacts = ContactsContract.Contacts.CONTENT_URI;

Some examples of predefined query string constants are as follows:

Browser.BOOKMARKS_URI

Browser.SEARCHES_URI

CallLog.CONTENT_URI

MediaStore.Images.Media.INTERNAL_CONTENT_URI

MediaStore.Images.Media.EXTERNAL_CONTENT_URI

Settings.CONTENT_URI

Projections

The second parameter of the managedQuery() method controls how many columns are returned by the query; this parameter is known as the projection. Earlier, you specified null:

Filtering

Cursor c = managedQuery(allContacts, null, null, null, null);

The third and fourth parameters of the managedQuery() method enable you to specify a SQL WHERE clause to filter the result of the query.

For example, the following statement retrieves only the people whose name ends with "Lee":

Cursor c = managedQuery(allContacts, projection,
ContactsContract.Contacts.DISPLAY_NAME + "LIKE '%Lee'",
null, null)

Sorting

The fifth parameter of the managedQuery() method enables you to specify a SQL ORDER BY clause to sort the result of the query. For example, the following statement sorts the contact names in ascending order.

Cursor c = managedQuery(allContacts,
projection,
ContactsContract.Contacts.DISPLAY_NAME + " LIKE ?", new String[] {"%"},
ContactsContract.Contacts.DISPLAY_NAME + " ASC");

To sort in descending order, use the DESC keyword:

Cursor c = managedQuery(allContacts, projection,
ContactsContract.Contacts.DISPLAY_NAME + " LIKE ?", new String[] {"%"},
ContactsContract.Contacts.DISPLAY_NAME + " DESC");

CREATING YOUR OWN CONTENT PROVIDERS

Steps involved in Creating Content Provider

Step 1: Create a new project

1. Click on File, then New => New Project.
2. Select language as Java/Kotlin.
3. Choose empty activity as a template
4. Select the minimum SDK as per your need.

Step 2: Modify the strings.xml file

All the strings used in the activity are stored here.

Step 3: Creating the Content Provider class

1. Click on File, then New => Other => ContentProvider.
2. Name the ContentProvider
3. Define authority (it can be anything for example "com.demo.user.provider")

4. Select Exported and Enabled option
5. Choose the language as Java/Kotlin

Step 4: Design the activity_main.xml layout

One Textview, EditText field, two Buttons, and another Textview to display the stored data are used to design.

Step 5: Modify the MainActivity file

Button functionalities will be defined in this file. Moreover, the query to be performed while inserting and fetching the data is mentioned here.

IMPLEMENTING THE CONTENT PROVIDER

When a content provider is called it can be implemented using different methods. content providers are created by subclassing the android.content.ContentProvider class. all content providers must have associated with them an authority and a content uri. In practice, the authority is typically the full package name of the content provider class itself, in this case com.example.database.provider.ContentProvider.

DIFFERENT METHODS OF IMPLEMENTING CONTENT PROVIDER

Implementing URI Matching in the Content Provider

When the methods of the content provider are called, they will be passed as an argument a URI indicating the data on which the operation is to be performed. This URI may take the form of a reference to a specific row in a specific table. It is also possible that the URI will be more general, for example specifying only the database table. It is the responsibility of each method to identify the Uri type and to act accordingly. This task can be eased considerably by making use of a UriMatcher instance. Once a UriMatcher instance has been created, it can be configured to return a specific integer value corresponding to the type of URI it detects when asked to do so. For the purposes of this tutorial, we will be configuring our UriMatcher instance to return a value of 1 when the URI references the entire products table, and a value of 2 when the URI references the ID of a specific row in the products table. Before working on creating the URIMatcher instance, we will first create two integer variables to represent the two URI types:

Implementing the Content Provider onCreate() Method

When the content provider class is created and initialized, a call will be made to the onCreate() method of the class. It is within this method that any initialization tasks for the class need to be performed. For the purposes of this example, all that needs to be performed is for an instance of the MyDBHandler class implemented in An Android SQLite Database Tutorial to be created. Once this instance has been created, it will need to be accessible from the other methods in the class; so a declaration for the database handler also needs to be declared, resulting in the following code changes to the ContentProvider.java file:

Implementing the Content Provider insert() Method

When a client application or activity requests that data be inserted into the underlying database, the insert() method of the content provider class will be called. At this point, however, all that exists in the ContentProvider.java file of the project is a stub method, which reads as follows:

Passed as arguments to the method are a URI specifying the destination of the insertion and a ContentValues object containing the data to be inserted.

This method now needs to be modified to perform the following tasks:

1. Use the sUriMatcher to identify the URI type.
2. Throw an exception if the URI is not valid.
3. Obtain a reference to a writable instance of the underlying SQLite database.
4. Perform a SQL insert operation to insert the data into the database table.
5. Notify the corresponding content resolver that the database has been modified.
6. Return the URI of the newly added table row.

Bringing these requirements together results in a modified insert() method, which reads as follows (note also that the argument names have been changed from arg0 and arg1 to names that are more self-explanatory):

Implementing the Content Provider query() Method

When a content provider is called upon to return data, the query() method of the provider class will be called. When called, this method is passed some or all of the following arguments:

1. *URI* – The URI specifying the data source on which the query is to be performed. This can take the form of a general query with multiple results, or a specific query targeting the ID of a single table row.
2. *Projection* – A row within a database table can comprise multiple columns of data. In the case of this application, for example, these correspond to the ID, product name and product quantity. The projection argument is simply a String array containing the name for each of the columns that is to be returned in the result data set.
3. *Selection* – The "where" element of the selection to be performed as part of the query. This argument controls which rows are selected from the specified database. For example, if the query was required to select only products named "Cat Food" then the selection string passed to the query() method would read productname = "Cat Food".
4. *Selection Args* – Any additional arguments that need to be passed to the SQL query operation to perform the selection.
5. *Sort Order* – The sort order for the selected rows.

When called, the query() method is required to perform the following operations:

1. Use the sUriMatcher to identify the Uri type.
2. Throw an exception if the URI is not valid.

3. Construct a SQL query based on the criteria passed to the method. For convenience, the SQLiteQueryBuilder class can be used in construction of the query.
4. Execute the query operation on the database.
5. Notify the content resolver of the operation.
6. Return a Cursor object containing the results of the query.

Implementing the Content Provider update() Method

The update() method of the content provider is called when changes are being requested to existing database table rows. The method is passed a URI, the new values in the form of a ContentValues object and the usual selection argument strings.

When called, the update() method would typically perform the following steps:

1. Use the sUriMatcher to identify the URI type.
2. Throw an exception if the URI is not valid.
3. Obtain a reference to a writable instance of the underlying SQLite database.
4. Perform the appropriate update operation on the database depending on the selection criteria and the URI type.
5. Notify the content resolver of the database change.
6. Return a count of the number of rows that were changed as a result of the update operation.

Implementing the Content Provider update() Method

The update() method of the content provider is called when changes are being requested to existing database table rows. The method is passed a URI, the new values in the form of a ContentValues object and the usual selection argument strings.

When called, the update() method would typically perform the following steps:

1. Use the sUriMatcher to identify the URI type.
2. Throw an exception if the URI is not valid.
3. Obtain a reference to a writable instance of the underlying SQLite database.
4. Perform the appropriate update operation on the database depending on the selection criteria and the URI type.
5. Notify the content resolver of the database change.
6. Return a count of the number of rows that were changed as a result of the update operation.

Implementing the Content Provider delete() Method

In common with a number of other content provider methods, the delete() method is passed a URI, a selection string and an optional set of selection arguments. A typical delete() method will also perform the following, and by now largely familiar, tasks when called:

1. Use the sUriMatcher to identify the URI type.
2. Throw an exception if the URI is not valid.

3. Obtain a reference to a writable instance of the underlying SQLite database.
4. Perform the appropriate delete operation on the database depending on the selection criteria and the Uri type.
5. Notify the content resolver of the database change.
6. Return the number of rows deleted as a result of the operation.

The next step is to make sure that the content provider is declared in the project manifest file so that it is visible to any content resolvers seeking access to it.

DECLARING THE CONTENT PROVIDER IN THE MANIFEST FILE

Unless a content provider is declared in the manifest file of the application to which it belongs, it will not be possible for a content resolver to locate and access it. As outlined, content providers are declared using the <provider> tag and the manifest entry must correctly reference the content provider authority and content URI.

All that remains before testing the application is to modify the database handler class to use the content provider instead of directly accessing the database.

READING/WRITING LOCAL DATA

Before talking about how your app reads from and writes to Realtime Database, let's introduce a set of tools you can use to prototype and test Realtime Database functionality: Firebase Local Emulator Suite. If you're trying out different data models, optimizing your security rules, or working to find the most cost-effective way to interact with the back-end, being able to work locally without deploying live services can be a great idea. A Realtime Database emulator is part of the Local Emulator Suite, which enables your app to interact with your emulated database content and config, as well as optionally your emulated project resources (functions, other databases, and security rules).

Using the Realtime Database emulator involves just a few steps:

1. Adding a line of code to your app's test config to connect to the emulator.
2. From the root of your local project directory, running firebase emulators:start.
3. Making calls from your app's prototype code using a Realtime Database platform SDK as usual, or using the Realtime Database REST API.

WRITE DATA

Basic write operations

For basic write operations, you can use setValue() to save data to a specified reference, replacing any existing data at that path. You can use this method to:

- Pass types that correspond to the available JSON types as follows:
- String

- Long
- Double
- Boolean
- Map<String, Object>
- List<Object>
- Pass a custom Java object, if the class that defines it has a default constructor that takes no arguments and has public getters for the properties to be assigned.

```
@IgnoreExtraProperties public class User { public String username;
public String email; public User() {
// Default constructor required for calls to DataSnapshot.getValue(User.class)
}
public User(String username, String email) { this.username = username;
this.email = email;
}}
```

We can add a user with setValue() as follows:

```
public void writeNewUser(String userId, String name, String email) { User user = new User(name, email);
mDatabase.child("users").child(userId).setValue(user);
}
```

Using setValue() in this way overwrites data at the specified location, including any child nodes. However, you can still update a child without rewriting the entire object. If you want to allow users to update their profiles you could update the username as follows:

```
mDatabase.child("users").child(userId).child("username").setValue(name);
```

READ DATA

Read data with persistent listeners

To read data at a path and listen for changes, use the addValueEventListener() method to add a ValueEventListener to a DatabaseReference. can use the onDataChange() method to read a static snapshot of the contents at a given path, as they existed at the time of the event. This method is triggered once when the listener is attached and again every time the data, including children, changes. The event callback is passed a snapshot containing all data at that location, including child data. If there is no data, the snapshot will return false when you call exists() and null when you call getValue() on it.

The listener receives a DataSnapshot that contains the data at the specified location in the database at the time of the event. Calling getValue() on a snapshot returns the Java object representation of the data. If no data exists at the location, calling getValue() returns null.

READ DATA ONCE

Read once using get()

The SDK is designed to manage interactions with database servers whether your app is online or offline. Generally, you should use the ValueEventListener techniques described above to read data to get notified of updates to the data from the backend. The listener techniques reduce your usage and billing, and are optimized to give your users the best experience as they go online and offline.

If we need the data only once, you can use get() to get a snapshot of the data from the database. If for any reason get() is unable to return the server value, the client will probe the local storage cache and return an error if the value is still not found. Unnecessary use of get() can increase use of bandwidth and lead to loss of performance, which can be prevented by using a realtime listener as shown above.

```
mDatabase.child("users").child(userId).get().addOnCompleteListener(new OnCompleteListener<DataSnapshot>() {
@Override
public void onComplete(@NonNull Task<DataSnapshot> task) { if (!task.isSuccessful()) {
Log.e("firebase", "Error getting data", task.getException());
}
else {
Log.d("firebase", String.valueOf(task.getResult().getValue()));
}
}
});
```

Read once using a listener

In some cases you may want the value from the local cache to be returned immediately, instead of checking for an updated value on the server. In those cases you can use addListenerForSingleValueEvent to get the data from the local disk cache immediately. This is useful for data that only needs to be loaded once and isn't expected to change frequently or require active listening. For instance, the blogging app in the previous examples uses this method to load a user's profile when they begin authoring a new post.

UPDATING OR DELETING DATA

Update specific fields

To simultaneously write to specific children of a node without overwriting other child nodes, use the updateChildren() method. When calling updateChildren(), you can update lower- level child values by specifying a path for the key. If data is stored in multiple locations to scale better, you can update all instances of that data using data fan-out. To create a post and simultaneously

update it to the recent activity feed and the posting user's activity feed. Using these paths, can perform simultaneous updates to multiple locations in the JSON tree with a single call to updateChildren(), such as how this example creates the new post in both locations. Simultaneous updates made this way are atomic: either all updates succeed or all updates fail.

INTERNAL STORAGE

In Android, the File System is a structured approach to storing, accessing, and managing data and files on the device. The File System consists of two primary storage options: Internal Storage and External Storage. Understanding the differences between these two storage types is crucial for developing efficient and user-friendly Android applications.

Understanding Internal Storage

Internal storage, as the name suggests, refers to the storage space that is private to the app itself. The data stored in the internal storage is only accessible to the app that created it. This makes it a secure option for storing sensitive user data that should not be accessible to other apps or users.

The internal storage of an app is a part of the device's built-in storage and is always available to the app, regardless of whether the device has external storage (SD card) or not. However, it's essential to note that the internal storage has limited space, and using it excessively might result in running out of storage, leading to potential crashes or data loss.

Accessing Internal Storage

To access the internal storage in an Android app, we use the Context object. The Context provides methods to open and create files or directories in the internal storage.

```
java// Get the context of the app
Context context = getApplicationContext();// Create or access a file in internal storageString filename = "example.txt";
File file = new File(context.getFilesDir(), filename);
```

Accessing the SD Card

Accessing an SD card in an Android Studio project involves interacting with external storage on an Android device. SD card access can be vital for various tasks such as reading and writing files, storing media, or accessing data from external sources. Here's a brief introduction to SD card access in Android Studio.

External Storage in Android: Android devices typically have two types of storage: internal storage and external storage. External storage can include SD cards or other removable media. While internal storage is private to each app and requires no special permissions, accessing external storage, including SD cards, requires explicit user permission.

To access external storage, your app needs appropriate permissions declared in the AndroidManifest.xml file. The two most common permissions for

external storage are and

WRITE_EXTERNAL_STORAGE. You request these permissions either at install

time or dynamically at runtime for devices running Android 6.0 (API level 23) or higher.

When your app needs to access external storage, it checks whether it has the required permissions. If not, it requests them from the user. This step is crucial for ensuring the user's consent and to comply with Android's security model.

use Android's storage

Once your app has the necessary permissions, you can APIs to access the SD card. The method provides the root directory of the primary external storage (which could be the SD card), and you can navigate the file system from there. Alternatively, you can use more specialized APIs like or for more advanced storage operations.

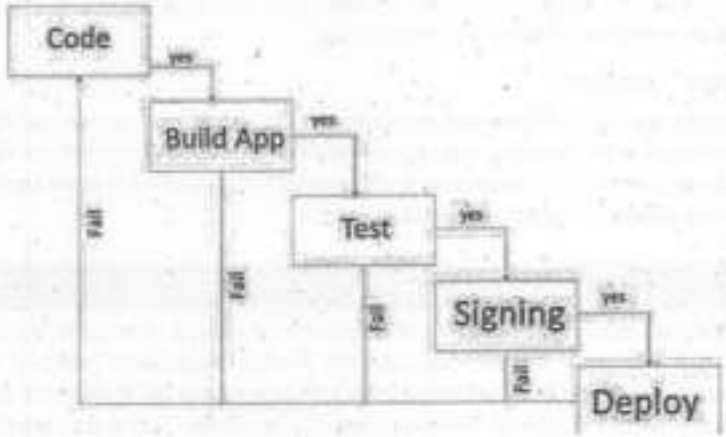
PREPARING APP FOR PUBLISHING

Publishing an app on Android refers to the process of making your application available for download and installation by users through the Google Play Store, which is the official marketplace for Android apps. When you publish your app on Android, it becomes accessible to millions of users worldwide who use Android-powered smartphones, tablets, and other devices.

key steps involved in publishing an app on Android

1. ***Preparation:*** Ensure the app is fully developed, tested, and ready for release. This includes finalizing features, fixing any bugs, optimizing performance, and making sure it complies with Google Play's policies and guidelines.
2. ***Creating a developer account:*** You need to sign up for a Google Play Developer account. There is a one-time registration fee, and this account allows you to publish and manage your apps on the Google Play Store.
3. ***Generating apk file:*** You need to build a signed APK file for your app. This file contains all the necessary code and resources required to install and run your application on Android devices.
4. ***Preparing Assets:*** You'll need to gather assets such as icons, screenshots, promotional images, and a feature graphic. These assets are used to showcase your app on the Google Play Store and help users understand what your app offers.
5. ***Setting up Store listing:*** Write a compelling app description, choose relevant categories, set pricing and distribution options if applicable, and provide other details such as contact information and a link to your privacy policy.
6. ***Submitting google play console:*** Log in to your Google Play Developer Console, upload your APK file, fill in the required information, and submit your app for review. 7.Review process: Google will review your app to ensure it meets their policies and quality standards. This process typically takes a few hours to a few days.
8. ***Launch:*** Once your app passes the review process, it will be published on the Google Play Store, and users will be able to find, download, and install it on their Android devices.

9. ***Promotions and maintenance:*** After launching your app, you'll need to promote it through various channels to attract users. You should also regularly monitor its performance, gather user feedback, and release updates to improve the app over time.



Preparing the app for release

Preparing the app for release is a multistep process involving the following tasks:

1. ***Configure the app for release:*** At a minimum, we need to make sure that logging is disabled and removed and that the release variant has debuggable false for Groovy or isDebuggable = false for Java script set. You should also set your app's version information.
2. ***Build and sign a release version of the app:*** we can use the Gradle build files with the release build type to build and sign a release version of the app. For more information, see Build and run the app.
3. ***Test the release version of the app:*** Before distributing the app, we should thoroughly test the release version on at least one target handset device and one target tablet device. Firebase Test Lab is useful for testing across a variety of devices and configurations.
4. ***Update app resources for release:*** Make sure that all app resources, such as multimedia files and graphics, are updated and included with your app or staged on the proper production servers.
5. ***Prepare remote servers and services that the app depends on:*** If your app depends on external servers or services, make sure they are secure and production ready.

RELEASING THE APP TO USERS

We can release the Android apps in several ways. Typically, we release apps through an app marketplace such as Google Play. we can also release apps on our own website or by sending an app directly to a user.

Release through an app marketplace

If we want to distribute our apps to the broadest possible audience, release them through an app marketplace.Google Play is the premier marketplace for Android apps and is particularly useful if we want to distribute our apps to a large global audience. However, we can distribute our apps through any app marketplace, and we can use multiple marketplaces.

Release your apps on Google Play

Google Play is a robust publishing platform that helps you publicize, sell, and distribute your Android apps to users around the world. When we release our apps through Google Play, we have access to a suite of developer tools that let you analyze your sales, identify market trends, and control who our apps are being distributed to.

Google Play also gives us access to several revenue-enhancing features such as in-app billing and app licensing. The rich array of tools and features, coupled with numerous end-user community features, makes Google Play the premier marketplace for selling and buying Android apps.

Releasing the app on Google Play is a simple process that involves three basic steps:

a. ***Prepare promotional materials:*** To fully leverage the marketing and publicity capabilities of Google Play, we need to create promotional materials for the app such as screenshots, videos, graphics, and promotional text.

b. ***Configure options and uploading assets:*** Google Play lets you target our app to a worldwide pool of users and devices. By configuring various Google Play settings, we can choose the countries we want to reach, the listing languages we want to use, and the price we want to charge in each country. can also configure listing details such as the app type, category, and content rating. When we are done configuring options, we can upload your promotional materials and our app as a draft app.

c. ***Publish the release version of your app:*** If we are satisfied that your publishing settings are correctly configured and our uploaded app is ready to be released to the public, click Publish. Once it has passed Google Play review, the app will be live and available for download around the world.

DEPLOYING APK FILES

What is an APK file (Android Package Kit file format)?

An APK file (Android Package Kit file format) is the file format for applications used on the Android operating system (OS). An APK file contains all the data an app needs, including all of the software program's code, assets and resources. All Android apps, including those downloaded from the Google Play Store or downloaded manually, use APK files.Developers who create applications for use on Android devices must compile their application into the APK format prior to uploading it to Google Play, the official marketplace for Android applications. An APK file is a type of archive format, similar to ZIP files, that contain multiple files and metadata in them. APK files are also a variant of Java Archive files.

CONTENTS OF AN ANDROID PACKAGE KIT FILE

APK files contain all contents needed to run the application, including the following:

1. AndroidManifest.xml. This is an additional Android manifest file that describes the name, version, access rights, library and other contents of the APK file.
2. assets/. These are application assets and resource files included with the app.
3. classes.dex. These are compiled Java classes in the DEX file format that are run on the device.
4. lib/. This folder contains platform-dependent compiled code and native libraries for device-specific architectures, such as x86 or x86_64.
5. META-INF/. This folder contains the application certificate, manifest file, signature and a list of resources.
6. res/. This is a directory that holds resources – for example, images that are not already compiled into resources.arsc.
7. resources.arsc. This is a file containing pre-compiled resources used by the app.

THE OPTIONS FOR DEVELOPING ANDRIOD APP

Android applications are packaged as ZIP files called Application Packages (APK) or Android App Bundles (AAB). You can install and run APK files on a device. You can upload AAB files to the Google Play store. To specify settings for application packages, select Projects > Build > Build Android APK > Details.

1. Packaging Applications

Because bundling applications as APK packages is not trivial, Qt has the androiddeployqt tool. When you deploy an application using a Qt for Android kit, Qt Creator runs the tool to create the necessary files and to bundle them into an APK. For more information, see Android Package Templates.

To view the packages that the androiddeployqt tool created, select the Open package location after build check box.

2. Specifying Deployment Settings

The Method field lists deployment settings. To add deployment methods for a project, select Add.

To rename the current deployment method, select Rename. To remove the current deployment method, select Remove.

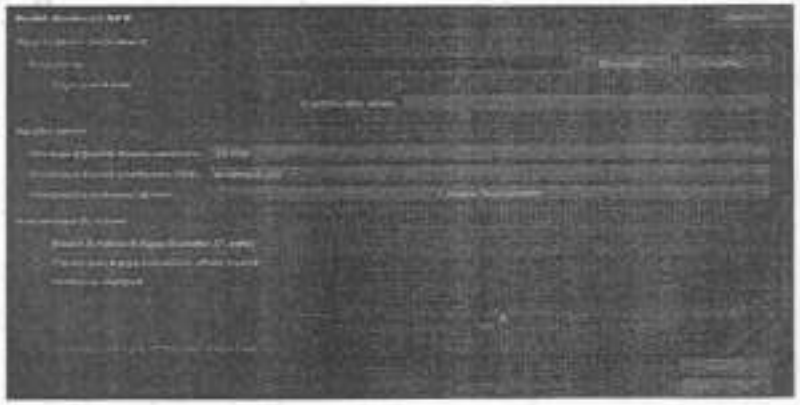
Qt Creator deploys the packages on the Android device that you select in the kit selector. To add devices, select Manage.

For more information about specifying additional start options for applications, see Specifying Run Settings for Android Devices.

To remove previously installed files from the device, select Uninstall the existing app before deployment.

To install a pre-built APK, such as a 3rd-party application to a device, select Install an APK File.

Specifying Settings for Packages



To specify settings for the androiddeployqt tool, select Projects > Build & Run > Build > Build Android APK > Details.

You can view information about what the androiddeployqt tool is doing in Compile Output. To view more information, select the Verbose output check box.

Select Add debug server to include the debug server binary into a package.

Selecting API Level:

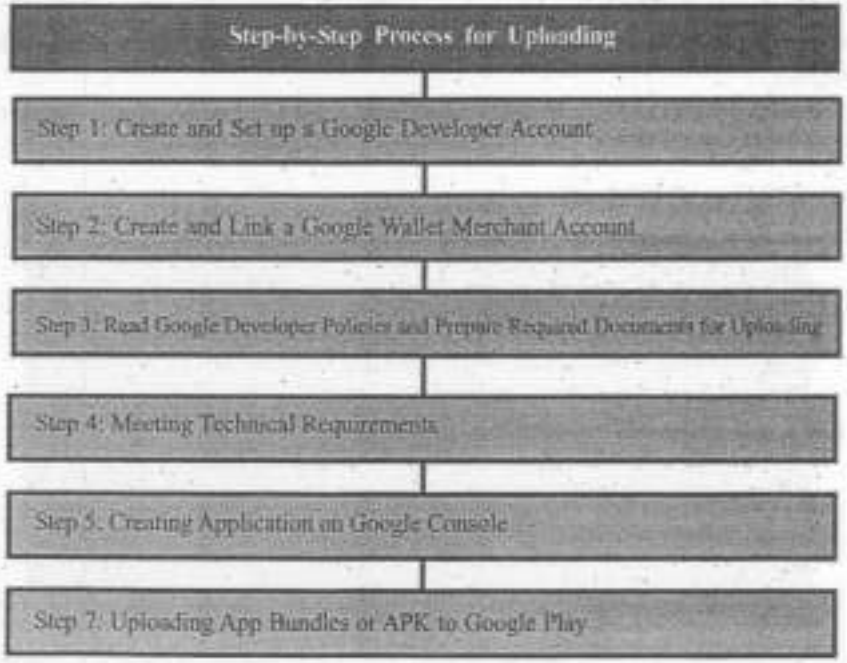
In the Android build platform SDK field, select the API level to use for building the application. Usually, you should select the highest API level available.

Usually, you should use the highest version of the Android SDK build-tools for building. If necessary, select another version in the Android build-tools version field.

UPLOADING IN MARKET

After spending countless hours crafting your Android app, pouring our creativity and expertise into every line of code, we stand at the important moment where our application will make its debut on the Google Play Store, the global stage for Android apps. The initial publication phase can be both exhilarating and demanding.

STEP-BY-STEP PROCESS FOR UPLOADING



Step 1: Create and Set up a Google Developer Account

Before outlining the steps for publishing an Android app, it's essential to cover the initial groundwork required before the publishing process begins. One of the initial tasks is setting up a Google Developer account, which we should do at the early stages of app development. Without a registered Google Developer Account, we won't be able to release our app on the Play Market. We have the flexibility to use an existing Google account or create a new one when signing up for a Google Developer Account. Whether it's a personal or corporate account, it doesn't matter. We can easily transfer your app to a different account in the future if needed.

Step 2: Create and Link a Google Wallet Merchant Account.

If we intend to sell paid apps or offer in-app purchases, it's necessary to establish a Google Merchant Account. This account enables us to oversee app sales, handle monthly payouts, and analyze sales reports. Upon completing the creation of your Merchant profile, our developer account will be seamlessly linked to it.

Step 3: Read Google Developer Policies and Prepare Required Documents for Uploading

Preparing paperwork, especially legal documents, can be quite a task. We recommend getting a head start by drafting essential documents like the End User License Agreement (EULA) and Privacy Policy. These can be created from similar app references or customized by a legal professional. The EULA outlines user rights, licensing fees, and intellectual property information.

Step 4: Meeting Technical Requirements

After navigating the intricate path of app development, thorough testing, and relentless bug fixing, the long-anticipated day of app release arrives. However, before proceeding with the upload process, several critical checks are in order.

Firstly, ensure that our app possesses a unique Bundle ID or package name. This identifier must be chosen thoughtfully, as it remains fixed throughout our application's lifecycle and cannot be modified after distribution. We can specify the package name in the app's manifest file.

It's imperative that our app release is signed with a valid Signing Certificate. This digital signature serves to identify the app's author and is irreplaceable once generated. Another essential aspect to verify is the app's size. Google imposes file size limits for uploads, allowing up to 100MB for Android 2.3 and higher (API level 9-10, 14, and above), and 50MB for earlier Android versions. If our app surpasses these limits, an alternative solution is to utilize APK Expansion Files.

consider the file format for our release. Google accepts two formats: the app bundle and .apk. The preferred format is .aab (app bundle). To utilize this format, enrollment in app signing by Google Play is necessary, offering an enhanced publishing experience.

Step 5: Creating Application on Google Console

With our app file ready to be uploaded, it's time to dive into the exciting part of the process: creating a new app within our Developer Account. Here's how to get started:

1. Go to the "All applications" tab located in the menu on Google Console.
2. Select "Create Application."
3. From the dropdown menu, pick the app's default language.
4. Add a concise app description (remember, we can always modify this later).
5. Finally, click on "Create." Step 6: Play Store Listing

Let's embark on the process of preparing the Store listing, a crucial step in ensuring our app's visibility and success in the digital marketplace, all while adhering to app store optimization (ASO) principles. Certain sections are mandatory and marked with an asterisk, and it's wise to start assembling the necessary materials beforehand.

Step 7: Uploading App Bundles or APK to Google Play

To upload our App bundle or APK files, along with the signed app release, start by navigating to the 'Release Management' section and then access the 'App Release' tab located in the menu. Here, we will be presented with four release options: internal test, closed test, production release, and open test. Carefully choose the release type that best suits our strategy, keeping in mind the goals and audience for our app. Once we made our selection, simply click on the 'Create Release' button to proceed.

After selecting our release type, we will be redirected to the 'New Release to Production' page. At this stage, an important decision awaits: whether or not to opt for Google Play app store signing. If we decide not to use this feature, we can easily select the 'OPT-OUT' option. Then, we need to choose 'Browse files' to initiate the APK upload process. Follow the on-screen instructions to complete the upload, including naming and describing our release. It's an opportunity to provide clear and enticing information about our app for potential users.

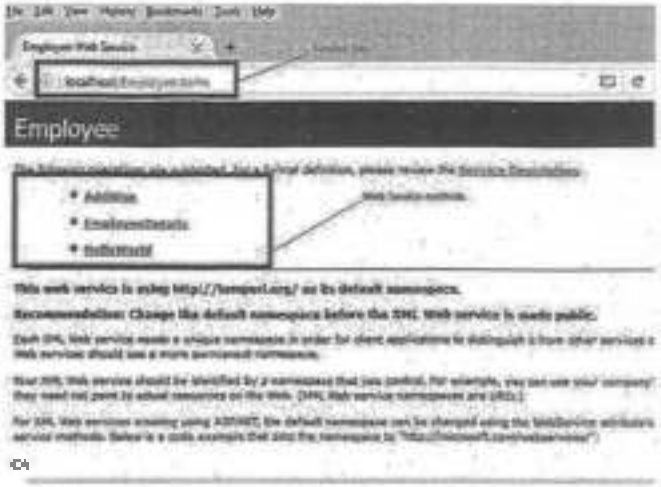
CONSUMING WEB SERVICES USING HTTP

A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language Java can talk with Perl; Windows applications can talk with Unix applications.

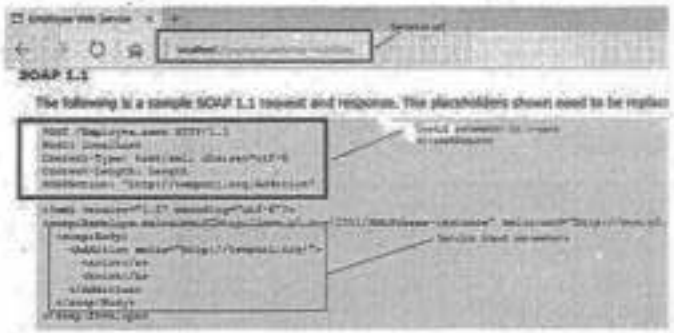
STEPS INVOLVED IN CONSUMING WE SERVICES

Step 1: Find the Web Service SOAP Request Body.

find out the SOAP request body which is used to pass input parameters and invoke the web service . I have one web service which is hosted in in my local IIS .So paste your web service url into the browser and wait till following response come which is shown in following image.



there are three web methods in the web service named Addition , EmployeeDetails and HelloWorld. Now click on one of the method which you wants to invoke , After clicking on web method it shows SOAP request and SOAP Response body , find out the SOAP request body which is shown in the following image.



In the above image , We are clearly observing that , It shows the all details how to make SOAP request and also showing parameter values which is require to invoke service using HttpWebRequest . So lets learn in brief about those parameters .

HTTP Method: It's nothing but tha what type of HTTP request you wants to make to the service such as GET , POST ,PUT or delete.

1. Host: It defines the url where is the service hosted , in our scenario our host is localhost , i.e http://localhost/Employee.asmx.
2. Content-Type: It defines what type of content type request you are making such as XML or json now in our scenario its 'text/xml'.
3. Content-Length: It defines the content length of request body.

4. SOAPAction: This is very important attribute to identify specific web method and call it from multiple web methods .
5. SOAP Body: It contains the request and response body.

Step 2: Create the Console application to call Web Service

1. "Start" - "All Programs" - "Microsoft Visual Studio 2015".
2. "File" – "New" - "Project..." then in the New Project window, console "C#" - ".
3. Give the project a name, such as "UsingSOAPRequest" or another as you wish and specify the location,
4. Let's create the method to invoke the web service using SOAP request bo
5. Now call above method from main method by writing following code as:
6. Now run the coinsole application ,then the console windows will be look like as follows and enter the input values as:

Now press enter button , it will give the response in XML format are as follows

In preceding console window you have seen web service response in the form of XML format which contains the output as 300 .

CONSUMING JSON SERVICES

Consuming JSON services refers to the process of retrieving data from web services that communicate using the JSON (JavaScript Object Notation) format. JSON has become a popular choice for data interchange due to its lightweight and human- readable nature. It is commonly used in web development, mobile applications, and various other software systems for transmitting structured data between a client and a server. When consuming JSON services, a client application typically sends HTTP requests to a server, which responds with JSON-formatted data. This data can then be parsed and processed by the client application to extract the relevant information. Consuming JSON services is a fundamental aspect of modern web development and is often utilized for fetching data from APIs (Application Programming Interfaces) provided by third-party services or internal systems.

Creating the Helper Method

To connect to a web service, your application needs to first of all connect to the server using HTTP. You need to also determine if you will be using HTTP GET or HTTP POST. Once that is determined, you will fetch the data from the server and get ready for the next step, which is parsing. For this article, the GeoNames Web Services that you will be using uses HTTP GET, and hence, you will first of all create the helper method readJSONFeed() in the MainActivity.java file:

The readJSONFeed() method takes in a string representing the URL of the web service and then connects to the server using HTTP GET. You make use of the HttpClient class to connect to the server, the HttpGet class to specify the URL of the server, and the HttpResponse class to get the connection status from the server. Once the connection is established successfully, you all use the BufferedReader and InputStreamReader classes to download the result (which is a JSON string in this example) from the server. The readJSONFeed() method then returns the JSON string.As we need Internet access for this project to work, remember to add the INTERNET permission in the AndroidManifest.xml file:

Getting Weather Information

Now that you can connect to the server to download the JSON result, it is now time to connect to the GeoNames Web Services to get weather information. In particular, to get the weather information of a particular location, you will use the following URL (replace the <lat> and <lng> with the actual latitude and longitude):If you observe, you have a primary key – weatherObservation. Its value is a collection of key/value pairs, such as clouds, weatherCondition, etc.In order to use the readJSONFeed() method, you need to call it asynchronously as beginning with Android 3.0 you can no longer call network operations from within your UI thread (such as within an activity). The easiest way to do this is to wrap it using an AsyncTask class. Hence, add the following statements to the MainActivity class:

CREATING YOUR OWN SERVICES BINDING ACTIVITIES TO SERVICES

Services are a great way to run operations in the background long after an application closes. However, while our app is open, we might want to directly access a service to call methods or influence how it runs. Unfortunately, the only way to interact with a started service is

to create a new intent and call startService() again. While this is great generic messaging and communication, it is not so great if you want to interact with the service directly. In order to interact with a service directly (like to control audio playing in a service), we need to bind to it.

How to Bind to a Service

If you recall in our services example, we had to override an onBind() method in our service and that method just returned null. If we want to bind to a service, we need to make this onBind() method return a Binder object that can be used to communicate with the service. A binder is an object that's used for accessing a service and its methods outside of the actual service class. Binders only work if the bound service is private to your application and exists within the same process. Since all of our services fit within those parameters, they're good candidates for using a binder.

Considerations with Bound Services

If we want to start a service, bind, and unbind without calling stop, the service would continue running until it was stopped. So to recap:

1. Unbinding will stop bound services that haven't been started.
2. If you start a service, you must call stop on it to stop it.
3. If you start a service and bind to it, you cannot stop that service until you have unbinded.

Lastly, if you unbind from a service, you still have access to the Binder and Service objects that were returned during binding. This is because there's no callback to say if a service unbinded successfully. Be sure that when you unbind from a service, we null out your binders and services so that you don't encounter any bugs from the service being destroyed behind the scenes, and also so we don't hold on to that memory and stop it from being cleaned up by the system properly.

Understanding Threading

Threading in mobile app development refers to the practice of creating and managing concurrent execution flows within an application. In simpler terms, threading allows an app to perform multiple tasks simultaneously, which is crucial for maintaining responsiveness and smooth user experience, especially in situations where the app needs to handle multiple operations concurrently. When an application is launched in Android, it creates the primary thread of execution, referred to as the "main" thread. Most thread is liable for dispatching events to the acceptable interface widgets also as communicating with components from the Android UI toolkit. To keep your application responsive, it's essential to avoid using the most thread to perform any operation which will find yourself keeping it blocked.

Threading in Android

In Android, you'll categorize all threading components into two basic categories:

1. Threads that are attached to an activity/fragment: These threads are tied to the lifecycle of the activity/fragment and are terminated as soon because the activity/fragment is destroyed.
2. Threads that aren't attached to any activity/fragment: These threads can still run beyond the lifetime of the activity/fragment (if any) from which they were spawned.

Type #1: Threading Components that Attach to an Activity/Fragment

1. Asynctask

AsyncTask, however, falls short if you would like your deferred task to run beyond the lifetime of the activity/fragment. The fact that even the slightest of screen rotation can cause the activity to be destroyed is simply awfull! So We Come to:

2. Loaders

Loaders are the answer to the awful nightmare mentioned above. Loaders are great at performing in that context and they automatically stop when the activity is destroyed, even more, the sweet fact is that they also restart themselves after the activity is recreated.

Type #2: Threading Components that Don't Attach to an Activity/Fragment

1. Service

Service could be thought of as a component that's useful for performing long (or potentially long) operations with no UI. Yes, you heard that right! Service's don't have any UI of their! Service runs within the main thread of its hosting process; the service doesn't create its own thread and doesn't run during a separate process unless you specify otherwise.

Thread Priority

As described in Processes and therefore the Application Lifecycle, the priority that your app's threads receive depends partly on where the app is within the app lifecycle. As you create and manage threads in your application, it's important to line their priority in order that the proper threads get the proper priorities at the proper times.

If the priority is set too high, then that thread might disrupt the UI Thread and even block it in some adverse cases and even the Render Thread, causing the app performance issues like dropped frames, lag, sluggish app UI, etc.

Every time you create a thread, you ought to call setThreadPriority(). The system's thread scheduler gives preference to threads with high priorities, balancing those priorities with the necessity to eventually get all the work done. Generally, threads within the foreground group get around 95% of the entire execution time from the device, while the background group gets roughly 5%.

REVIEW QUESTIONS

1. Define content providers?
2. Mention the different operations in content provider.
3. Explain the steps involved in sms messanging.
4. What is intent?
5. Mention the types of maps.
6. How to view the stored data in sqlite in android studi.
7. What is database helper class?
8. What is db fetching? Give the example.
9. What is data binding?
10. Mention the different event handlings.
11. Explain steps involved in creating content provider.
12. Mention the contents of an andriod pacakge kit files.
13. Mention the different ways to acess the sd card in an andriod

LAB

1. CREATING "HELLO WORLD" APPLICATION

File : Main Activity File

```
package com.example.helloandroid;
import androidx.appcompat.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);
  }
}
```

File :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">
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Output:

2. CREATING AN APPLICATION THAT DISPLAYS MESSAGE BASED ON THE SCREEN ORIENTATION

In this program, we will create two activities of different screen orientation. The first activity (MainActivity) will be as "portrait" orientation and second activity (SecondActivity) as "landscape" orientation type.

File: activity_main.xml

```
<?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="example.demo.com.screenorientation.MainActivity">
    <Button
        android:id="@+id/button1"
```



```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginTop="112dp"
        android:onClick="onClick"
        android:text="Launch next activity"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.512"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText1"
        app:layout_constraintVertical_bias="0.613" />
    <TextView
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="124dp"
        android:ems="10"
        android:textSize="22dp"
        android:text="This activity is portrait orientation"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.502"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

File : MainActivity.java

```
package example.demo.com.screenorientation;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
```

```
public class MainActivity extends AppCompatActivity {
    Button button1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button1=(Button)findViewById(R.id.button1);
    }
    public void onClick(View v) {
        Intent intent = new Intent(MainActivity.this,SecondActivity.class);
        startActivity(intent);
    }
}
```

File: activity_second.xml

```
<?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="example.demo.com.screenorientation.SecondActivity">
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="180dp"
        android:text="this is landscape orientation"
        android:textSize="22dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.502"
        app:layout_constraintStart_toStartOf="parent"
```



```
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

File: SecondActivity.java

```
package example.demo.com.screenorientation;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class SecondActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
    }
}
```

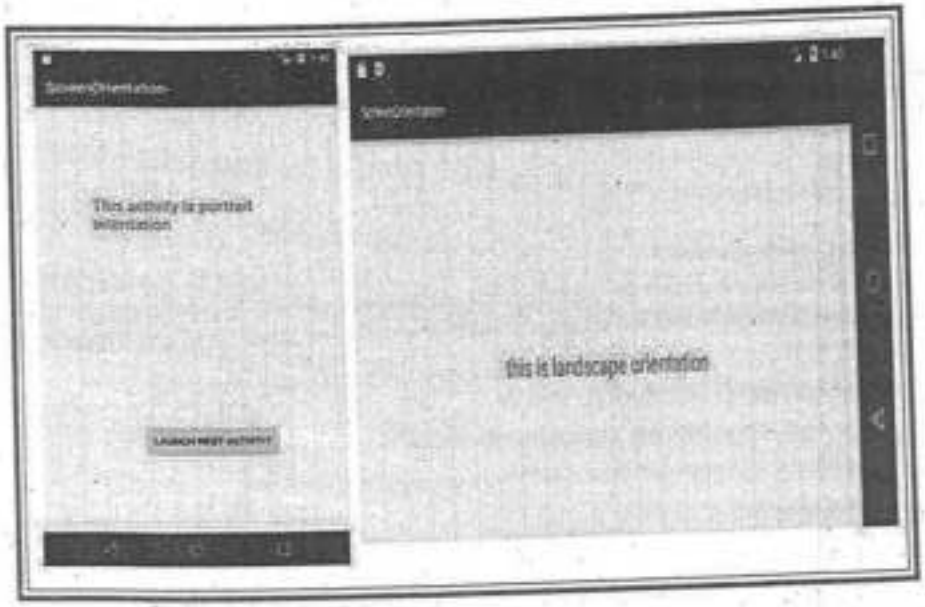
File: AndroidManifest.xml

In AndroidManifest.xml file add the screenOrientation attribute in activity and provides its orientation. In this example, we provide "portrait" orientation for MainActivity and "landscape" for SecondActivity.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="example.demo.com.screenorientation">
    <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="example.demo.com.screenorientation.MainActivity"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
```

```
        </activity>
        <activity android:name=".SecondActivity"
            android:screenOrientation="landscape">
        </activity>
    </application>
</manifest>
```

Output:



3. CREATE AN APPLICATION TO DEVELOP LOGIN WINDOW USING UI CONTROLS

File type: activity_login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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="com.rezoomist.simpleloginapplication.Login">
```



```
<TextView
    android:id="@+id/tv_login"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="11dp"
    android:layout_marginStart="11dp"
    android:layout_marginTop="13dp"
    android:text="Login"
    android:fontFamily="sans-serif-condensed"
    android:textSize="30sp"/>
<TextView
    android:id="@+id/tv_username"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/tv_login"
    android:layout_alignStart="@+id/tv_login"
    android:layout_below="@+id/tv_login"
    android:layout_marginTop="80dp"
    android:fontFamily="monospace"
    android:text="Username"
    android:textSize="25sp" />
<EditText
    android:id="@+id/et_username"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/tv_username"
    android:ems="17"
    android:layout_alignLeft="@+id/tv_username" />
<TextView
    android:id="@+id/tv_password"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
        android:layout_alignLeft="@+id/et_username"
        android:layout_alignStart="@+id/et_username"
        android:layout_below="@+id/et_username"
        android:layout_marginTop="33dp"
        android:fontFamily="monospace"
        android:text="Password"
        android:textSize="25sp" />
    <EditText
        android:id="@+id/et_password"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="textPassword"
        android:ems="17"
        android:layout_below="@+id/tv_password"
        android:layout_alignLeft="@+id/tv_password" />
    <Button
        android:id="@+id/btn_login"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_below="@id/et_password"
        android:layout_centerInParent="true"
        android:ems="12"
        android:layout_marginTop="30dp"
        android:text="Login"/>
</RelativeLayout>
```

File type : activity_user.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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="com.razorinist.simpleloginapplication.User">
    <TextView
        android:id="@+id/textView1"
```



```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Welcome"
        android:textSize="40sp"
        android:layout_marginTop="177dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:layout_marginTop="30dp"
        android:layout_centerInParent="true"
        android:layout_below="@+id/textView1"
        android:text="Administrator"/>
</RelativeLayout>
```

File: AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.razorinist.simpleloginapplication">
    <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=".Login"
            android:configChanges="orientation"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
```

```
        </activity>
        <activity android:name=".User"
          android:configChanges="orientation"
          android:screenOrientation="portrait">
          <intent-filter>
            <action android:name="com.razomlist.simpleloginapplication.User" />
            <category android:name="android.intent.category.DEFAULT" />
          </intent-filter>
        </activity>
      </application>
    </manifest>
```

File : Login.java

```
package com.razomlist.simpleloginapplication;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class Login extends AppCompatActivity {
  EditText et_username, et_password;
  Button btn_login;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);
    Login();
  }
  void Login(){
    et_username = (EditText)findViewById(R.id.et_username);
```



```
    et_password = (EditText)findViewById(R.id.et_password);
    btn_login = (Button)findViewById(R.id.btn_login);
    btn_login.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
                                    if(et_username.getText().toString().equals("admin") &&
et_password.getText().toString().equals("admin")){
        Toast.makeText(Login.this, "Username and Password is correct",
Toast.LENGTH_SHORT).show();
          Intent intent = new Intent(Login.this,User.class);
          startActivity(intent);
        }else{
        Toast.makeText(Login.this, "Username or Password is incorrect",
Toast.LENGTH_SHORT).show();
        }
      }
    });
  }
}
```

File : User.java

```
package com.razomlist.simpleloginapplication;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class User extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_user);
  }
}
```

Output:

username: admin

password: admin.

4. CREATE AN APPLICATION TO IMPLEMENT NEW ACTIVITY USING EXPLICIT INTENT, IMPLICIT INTENT

File : activity_main.xml

```
<?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:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity"
android:gravity="center">
<Button
android:layout_width="wrap_content"
```



```
android:layout_height="wrap_content"
android:layout_marginRight="16dp"
android:text="Implicit Intent"
android:onClick="onImplicitButtonClicked"/>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Explicit Intent
android:onClick="onExplicitButtonClicked"/>
</LinearLayout>
```

File : MainActivity.java

```
package com.ep2024.androidintent;
import ...
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
public void onImplicitButtonClicked(View view) {
Uri uri = Uri.parse("https://www.google.com");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
}
public void onExplicitButtonClicked(View view) {
Intent intent = new Intent( packageContext: MainActivity.this, SecondActivity.class);
startActivity(intent);
Intent intent = new Intent(MainActivity.this, SecondActivity.class);
}
}
```

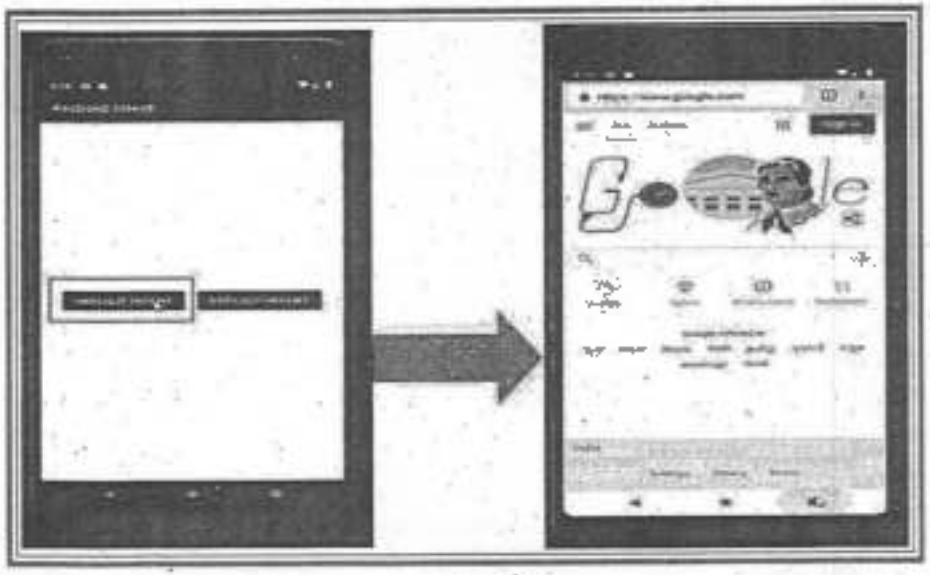
In order to create Second Activity:Go to file ? New? Activity?Empty Activity?Name it as Second Activity

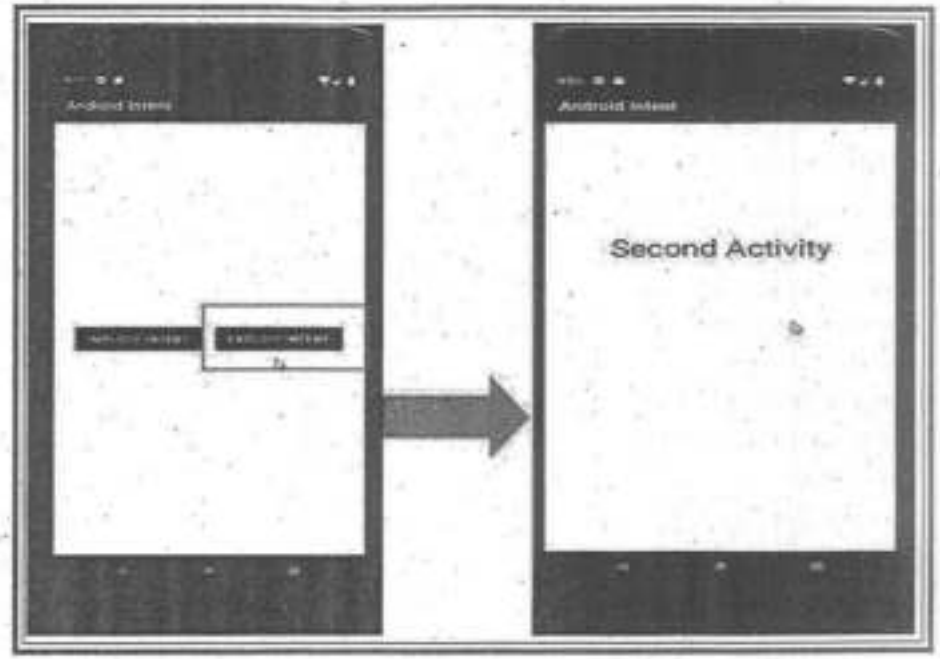
File :SecondActivity.xml

```
<?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=".SecondActivity">
  <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginEnd="8dp"
      android:layout_marginStart="8dp"
      android:layout_marginTop="8dp"
      android:text="Second Activity" />
</android.support.constraint.ConstraintLayout>
```

Output:





5. CREATE AN APPLICATION THAT DISPLAYS CUSTOM DESIGNED OPENING SCREEN

File : activity_main.xml

```
<?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:id="@+id/activity_main"
    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="android.com.splashscreen.MainActivity">
    <TextView
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Hello Android Developer:"
        android:textSize="20sp"
        android:layout_centerInParent="true"/>
</RelativeLayout>
```

File: Create a new XML file,name it as splashfile.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center"
android:background="@color/splashBackground">
<ImageView
  android:id="@+id/logo_id"
  android:layout_width="250dp"
  android:layout_height="250dp"
  android:layout_centerInParent="true"
  android:src="@drawable/android"/>
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/logo_id"
    android:layout_centerHorizontal="true"
    android:text="Splash Screen"
    android:textSize="30dp"
    android:textColor="@color/blue"/>
</RelativeLayout>
```

File : MainActivity.java

```
package android.com.splashscreen;
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);
  }
}
```

File : SplashActivity.java

```
package android.com.splashscreen;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
public class SplashActivity extends Activity {
  Handler handler;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.splashfile);
    handler=new Handler();
    handler.postDelayed(new Runnable() {
      @Override
      public void run() {
        Intent intent=new Intent(SplashActivity.this,MainActivity.class);
        startActivity(intent);
        finish();
      }
    },3000);
  }
}
```

File :AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="android.com.splashscreen">
<application
  android:allowBackup="true"
  android:icon="@drawable/android"
  android:label="@string/app_name"
  android:supportsRtl="true"
```

```
android:theme="@style/AppTheme">
<activity android:name="android.com.splashscreen.SplashActivity">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
<activity android:name="android.com.splashscreen.MainActivity"/>
</application>
</manifest>
```

Output:

6. CREATE AN UI WITH ALL VIEWS

File :Activitymain.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >
  <TextView
    android:id="@+id/textView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Hello Readers!!!" />
  <Button
    android:id="@+id/buttonExample"
    android:layout_width="match_parent"
```



```
    android:layout_height="wrap_content"
    android:text="Example Button" />
  <CheckBox
    android:id="@+id/checkBoxExample"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="CheckBox Example" />
  <ToggleButton
    android:id="@+id/toggleButtonExample"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="ToggleButton Example" />
  <RadioGroup
    android:id="@+id/radioGroupExample"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >
    <RadioButton
      android:id="@+id/radioButton1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:checked="true"
      android:text="RadioButton 1" />
    <RadioButton
      android:id="@+id/radioButton2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="RadioButton 2" />
    <RadioButton
      android:id="@+id/radioButton3"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="RadioButton 3" />
  </RadioGroup>
  <ImageButton
    android:id="@+id/imageButtonExample"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher" />
</LinearLayout>
```

File : Main Activity File

```
package com.android.tution.BasicViews;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.ImageButton;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.RadioGroup.OnCheckedChangeListener;
import android.widget.Toast;
import android.widget.ToggleButton;
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    setContentView(R.layout.activity_main);
    Button example = (Button) findViewById(R.id.buttonExample);
    CheckBox exampleCheckBox = (CheckBox) findViewById(R.id.checkBoxExample);
RadioGroup exampleRadioGroup = (RadioGroup) findViewById(R.id.radioGroupExample);
ToggleButton exampletoggleButton = (ToggleButton) findViewById(R.id.toggleButton
Example);
ImageButton exampleImageButton = (ImageButton) findViewById(R.id.imageButton
Example);
    example.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View arg0) {
        // TODO Auto-generated method stub
        // Toast.makeText(this, "Hey Button is pressed !!",
        // Toast.LENGTH_SHORT).show();
```



```
        ToastToDisplay("Hey Button is pressed!!");
      }
    });
    exampleCheckBox.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        // TODO Auto-generated method stub
        if (((CheckBox) v).isChecked()) {
          ToastToDisplay("Check Box is checked");
        } else {
          ToastToDisplay("Check box is unchecked");
        }
      }
    });
    exampleRadioGroup
        .setOnCheckedChangeListener(new OnCheckedChangeListener() {
          @Override
          public void onCheckedChanged(RadioGroup group, int checkedId) {
            // TODO Auto-generated method stub
            RadioButton rb1 = (RadioButton) findViewById(R.id.radioButton1);
            RadioButton rb2 = (RadioButton) findViewById(R.id.radioButton2);
            if (rb1.isChecked()) {
              ToastToDisplay("Radio Button 1 is checked");
            } else if (rb2.isChecked()) {
              ToastToDisplay("Radio Button 2 is checked");
            } else {
              ToastToDisplay("Radio Button 3 is checked");
            }
          }
        });
    exampletoggleButton.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        // TODO Auto-generated method stub
        if (((ToggleButton) v).isChecked()) {
```

```
                ToastToDisplay("Toggle button is ON");
              } else {
                ToastToDisplay("Toggle Button is OFF");
              }
            }
        });
        exampleImageButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                ToastToDisplay("Image Button is pressed");
            }
        });
    }
    private void ToastToDisplay(String args) {
      Toast.makeText(getBaseContext(), args, Toast.LENGTH_SHORT).show();
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
```

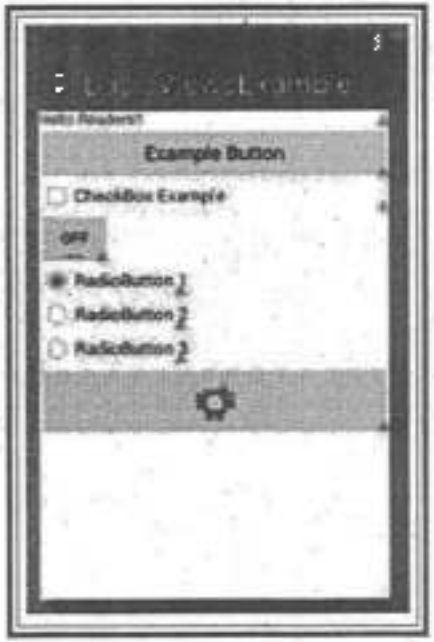
File: AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.tution.BasicViews"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="15" />
    <application
      android:allowBackup="true"
```



```
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
          android:name="com.android.tution.BasicViews.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>
```

Output:

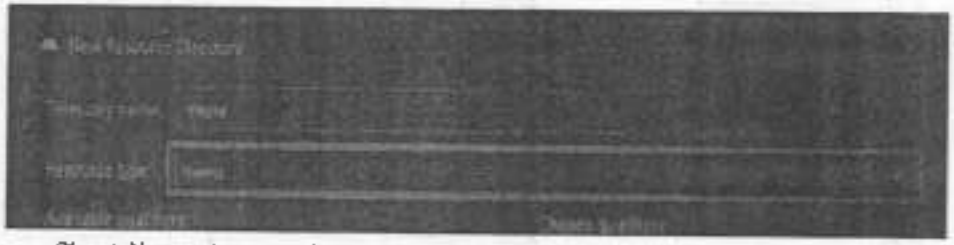


7. CREATE MENU IN APPLICATION.

Step 1: Open the project and go to the app > src > main > res as shown in the below image.

Step 2: Right-click on the res folder > New > Android Resource Directory as shown in the below image.

Step 3: Now you select the menu option from the values drop-down section.



Step 4: Now you can see the menu folder inside the res folder. To create a menu file select the menu folder and right-click on it and select Menu Resource File.Give a name to your menu file, for instance here we have given the name as my_menu.

File : my_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
<item android:id="@+id/search_item"
      android:title="Search" />
<item android:id="@+id/upload_item"
```



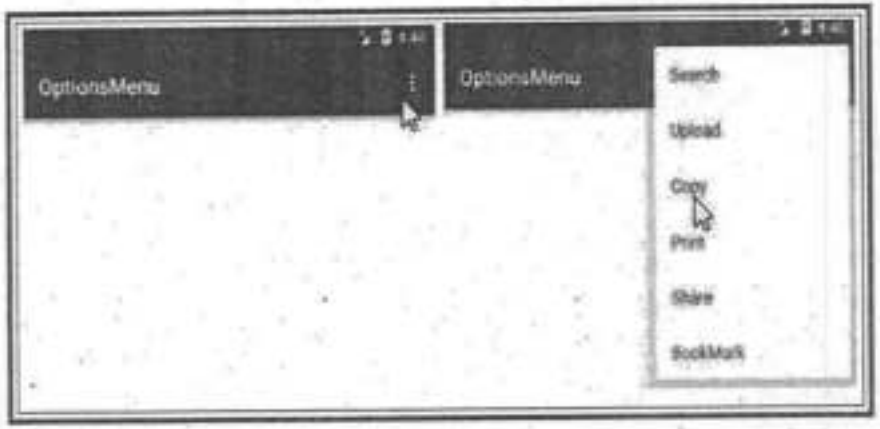
```
      android:title="Upload" />
<item android:id="@+id/copy_item"
      android:title="Copy" />
<item android:id="@+id/print_item"
      android:title="Print" />
<item android:id="@+id/share_item"
      android:title="Share" />
<item android:id="@+id/bookmark_item"
      android:title="BookMark" />
</menu>
```

File : MainActivity.java

```
package com.example.optionsmenu;
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) {
  getMenuInflater()
    .inflate(R.menu.options_menu, menu);
  return true;
 }
 @Override
 public boolean onOptionsItemSelected(MenuItem item) {
  Toast.makeText(this, "Selected Item: " + item.getTitle(), Toast.LENGTH_SHORT)
    .show();
  switch (item.getItemId()) {
  case R.id.search_item:
```

```
        return true;
      case R.id.upload_item:
        return true;
      case R.id.copy_item:
        return true;
      case R.id.print_item:
        return true;
      case R.id.share_item:
        return true;
      case R.id.bookmark_item:
        return true;
      default:
        return super.onOptionsItemSelected(item);
      }
    }
  }
```

Output:



8. READ/ WRITE THE LOCAL DATA

```
import android.content.Context;
import android.content.SharedPreferences;
public class MyPreferences {
    private static final String PREFS_NAME = "MyPrefs";
    public static void saveData(Context context, String key, String value) {
      SharedPreferences sharedPreferences = context.getSharedPreferences(PREFS_NAME,
Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.putString(key, value);
        editor.apply();
    }
}
import android.content.Context;
import android.content.SharedPreferences;
public class MyPreferences {
    private static final String PREFS_NAME = "MyPrefs";
    public static String getData(Context context, String key) {
      SharedPreferences sharedPreferences = context.getSharedPreferences(PREFS_NAME,
Context.MODE_PRIVATE);
        return sharedPreferences.getString(key, "");
    }
}
import android.os.Bundle;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Write data to SharedPreferences
        MyPreferences.saveData(MainActivity.this, "name", "John");
        // Read data from SharedPreferences
        String name = MyPreferences.getData(MainActivity.this, "name");
        // Display the data
```

```
        TextView textView = findViewById(R.id.text_view);
        textView.setText("Name: " + name);
    }
}
```

Sample output:

9. CREATE / READ / WRITE DATA WITH DATABASE (SQLITE).

Create a Database Helper Class:

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
public class DatabaseHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "mydatabase";
    private static final int DATABASE_VERSION = 1;
    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
```



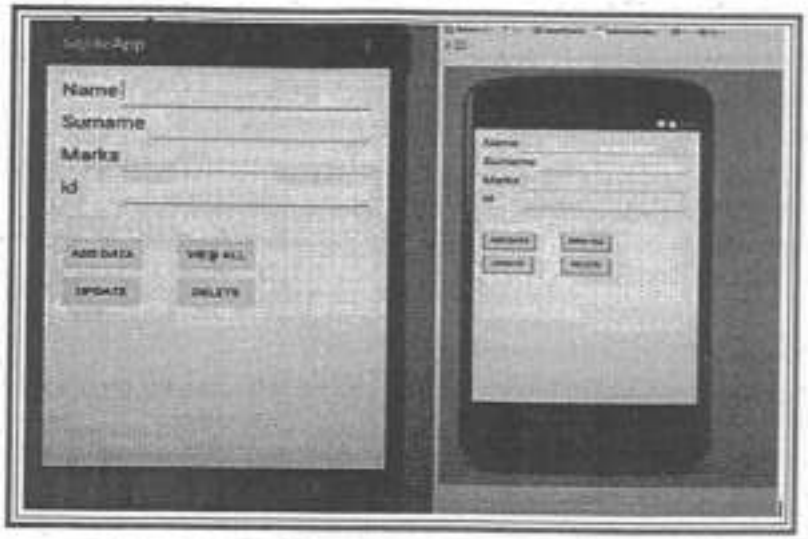
```
        String createTableQuery = "CREATE TABLE mytable (id INTEGER PRIMARY KEY,
name TEXT)";
        db.execSQL(createTableQuery);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS mytable");
        onCreate(db);
    }
}
```

Perform CRUD Operations in MainActivity

```
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity {
    private DatabaseHelper dbHelper;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        dbHelper = new DatabaseHelper(this);
        // Example: Inserting Data
        insertData("John");
        // Example: Reading Data
        String data = readData();
        TextView textView = findViewById(R.id.text_view);
        textView.setText(data);
        // Example: Updating Data
        updateData(1, "Alice");
        // Example: Deleting Data
        deleteData(1);
    }
```

```
    private void insertData(String name) {
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put("name", name);
        db.insert("mytable", null, values);
        db.close();
    }
    private String readData() {
        SQLiteDatabase db = dbHelper.getReadableDatabase();
        Cursor cursor = db.rawQuery("SELECT * FROM mytable", null);
        StringBuilder result = new StringBuilder();
        if (cursor.moveToFirst()) {
            do {
                int id = cursor.getInt(cursor.getColumnIndex("id"));
                String name = cursor.getString(cursor.getColumnIndex("name"));
                result.append("ID: ").append(id).append(", Name: ").append(name).append("\n");
            } while (cursor.moveToNext());
        }
        cursor.close();
        db.close();
        return result.toString();
    }
    private void updateData(int id, String newName) {
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put("name", newName);
        db.update("mytable", values, "id = ?", new String[]{String.valueOf(id)});
        db.close();
    }
    private void deleteData(int id) {
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        db.delete("mytable", "id = ?", new String[]{String.valueOf(id)});
        db.close();
    }
}
```


Sample output:



10. CREATE AN APPLICATION TO SEND SMS AND RECEIVE SMS

MainActivity.java

```
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.os.Build;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.telephony.SmsMessage;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
public class MainActivity extends AppCompatActivity {
    private static final int PERMISSION_REQUEST_SEND_SMS = 1;
    private static final int PERMISSION_REQUEST_RECEIVE_SMS = 2;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Request SEND_SMS permission
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.SEND_SMS) !=
PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.SEND_SMS},
PERMISSION_REQUEST_SEND_SMS);
        }
        // Request RECEIVE_SMS permission
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.RECEIVE_SMS) !=
PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.RECEIVE_SMS},
PERMISSION_REQUEST_RECEIVE_SMS);
        }
    }
    // Method to send SMS
    private void sendSMS(String phoneNumber, String message) {
        SmsManager smsManager = SmsManager.getDefault();
        smsManager.sendTextMessage(phoneNumber, null, message, null, null);
    }
    // Method to receive SMS
    private BroadcastReceiver smsReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Bundle bundle = intent.getExtras();
            if (bundle != null) {
                Object[] pdus = (Object[]) bundle.get("pdus");
                if (pdus != null) {
                    for (Object pdu : pdus) {
                        SmsMessage smsMessage;
                        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                            String format = bundle.getString("format");
                            smsMessage = SmsMessage.createFromPdu((byte[]) pdu, format);
                        } else {
```



```
                            smsMessage = SmsMessage.createFromPdu((byte[]) pdu);
                        }
                        String senderPhoneNumber = smsMessage.getDisplayOriginatingAddress();
                        String messageBody = smsMessage.getMessageBody();
                        // Handle received SMS
                        Toast.makeText(context, "SMS Received from: " + senderPhoneNumber +
"\nMessage: " + messageBody, Toast.LENGTH_SHORT).show();
                    }
                }
            }
        }
    };
    @Override
    protected void onResume() {
        super.onResume();
        // Register SMS receiver
    registerReceiver(smsReceiver, new IntentFilter("android.provider.Telephony.
    SMS_RECEIVED"));
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Unregister SMS receiver
        unregisterReceiver(smsReceiver);
    }
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        if (requestCode == PERMISSION_REQUEST_SEND_SMS) {
            if (grantResults.length > 0 && grantResults[0] == PackageManager.
PERMISSION_GRANTED) {
                // Permission granted to send SMS
                // You can now send SMS
            } else {
                // Permission denied
```

```
                // Handle accordingly
            }
        } else if (requestCode == PERMISSION_REQUEST_RECEIVE_SMS) {
                                .if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                // Permission granted to receive SMS
                // You can now receive SMS
            } else {
                // Permission denied
                // Handle accordingly
            }
        }
    }
```

AndroidManifest.xml file:

```
xml
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<activity android:name=".MainActivity">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
package com.example.sendsmsdemo;
import android.os.Bundle;
import android.app.Activity;
import android.telephony.SmsManager;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends Activity {
```



```
    Button sendBtn;
    EditText txtphoneNo;
    EditText txtMessage;
  @Override
    protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
  sendBtn = (Button) findViewById(R.id.btnSendSMS);
      txtphoneNo = (EditText) findViewById(R.id.editTextPhoneNo);
      txtMessage = (EditText) findViewById(R.id.editTextSMS);
  sendBtn.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
          sendSMSMessage();
        }
      });
    }
    protected void sendSMSMessage() {
      Log.i("Send SMS", "");
      String phoneNo = txtphoneNo.getText().toString();
      String message = txtMessage.getText().toString();
      try {
        SmsManager smsManager = SmsManager.getDefault();
        smsManager.sendTextMessage(phoneNo, null, message, null, null);
  Toast.makeText(getApplicationContext(), "SMS sent.", Toast.LENGTH_LONG).show();
      } catch (Exception e) {
                Toast.makeText(getApplicationContext(), "SMS failed, please try again.",
Toast.LENGTH_LONG).show();
        e.printStackTrace();
      }
    }
}}
  protected void sendSMSMessage() {
    Log.i("Send SMS", "");
    String phoneNo = txtphoneNo.getText().toString();
    String message = txtMessage.getText().toString();
```

```
        try {
            SmsManager smsManager = SmsManager.getDefault();
            smsManager.sendTextMessage(phoneNo, null, message, null, null);
            Toast.makeText(getApplicationContext(), "SMS sent.", Toast.LENGTH_LONG).show();
        } catch (Exception e) {
                    Toast.makeText(getApplicationContext(),"SMS failed, please try again.",
Toast.LENGTH_LONG).show();
            e.printStackTrace();
        }
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
    }
```

res/layout/activity_main.xml file:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/textViewPhoneNo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/phone_label" />
```

res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">SendSMSDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="phone_label">Enter Phone Number:</string>
    <string name="sms_label">Enter SMS Message:</string>
```



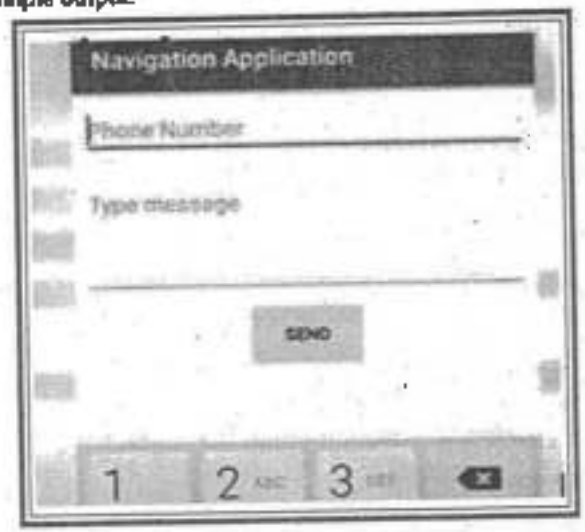
```
    <string name="send_sms_label">Send SMS</string>
</resources>
    <EditText
        android:id="@+id/editTextPhoneNo"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="phone" />
    <TextView
        android:id="@+id/textViewMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/sms_label" />
    <EditText
        android:id="@+id/editTextSMS"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="textMultiLine" />
    <Button
        android:id="@+id/btnSendSMS"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Send SMS" />
</LinearLayout>
android:text="@string/send_sms_label" />
</linearLayout>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sendsmsdemo"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.SEND_SMS" />
```

```
    <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme">
      <activity
        android:name="com.example.sendsmsdemo.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>
```

Sample output:



11. CREATE AN APPLICATION TO SEND AN E-MAIL.

```
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button sendEmailButton = findViewById(R.id.send_email_button);
    sendEmailButton.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        composeEmail();
      }
    });
  }
  private void composeEmail() {
    Intent intent = new Intent(Intent.ACTION_SENDTO);
    intent.setData(Uri.parse("mailto:")); // only email apps should handle this
    intent.putExtra(Intent.EXTRA_EMAIL, new String[]{"recipient@example.com"});
    intent.putExtra(Intent.EXTRA_SUBJECT, "Subject of the email");
    intent.putExtra(Intent.EXTRA_TEXT, "Body of the email");

    if (intent.resolveActivity(getPackageManager()) != null) {
      startActivity(intent);
    }
  }
}
```

AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
```

activity_main.xml

```
<Button
    android:id="@+id/send_email_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Send Email" />
```

Sample output:

12. DISPLAY MAP BASED ON THE CURRENT/GIVEN LOCATION.

```
package com.example.googlemaps;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
import android.os.Bundle;
```



```
import android.app.Activity;
public class MainActivity extends Activity {
    static final LatLng TutorialsPoint = new LatLng(21, 57);
    private GoogleMap googleMap;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        try {
            if (googleMap == null) {
                googleMap = ((MapFragment) getFragmentManager().findFragmentById
(R.id.map)).getMap();
            }
            googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
            Marker TP = googleMap.addMarker(new MarkerOptions().position
(TutorialsPoint).title("TutorialsPoint"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

res/layout/activity_main.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="fill_parent">
    <fragment
        android:id="@+id/map"
        android:name="com.google.android.gms.maps.MapFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</RelativeLayout>
```

AndroidManifest.xml file.

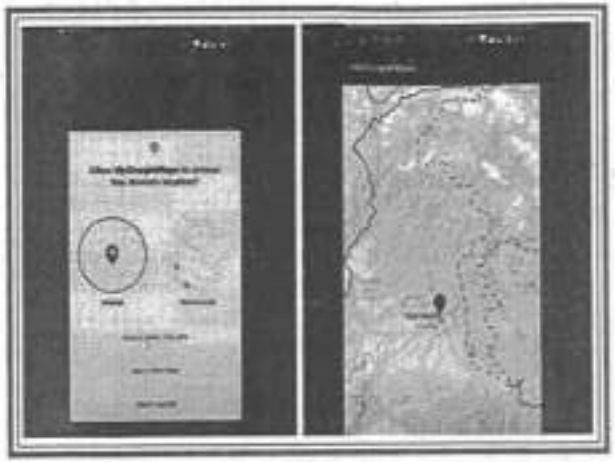
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.googlemaps"
```

```
        android:versionCode="1"
        android:versionName="1.0" >
    <uses-permission android:name="com.example.googlemaps.permission.MAPS_RECEIVE" /
>
        <uses-sdk
          android:minSdkVersion="12"
          android:targetSdkVersion="17" />
        <permission
          android:name="com.example.googlemaps.permission.MAPS_RECEIVE"
          android:protectionLevel="signature" />
        <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
        <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission          android:name="com.google.android.providers.gsf.permission.
READ_GSERVICES" />
        <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
        <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
        <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
        <uses-feature
          android:glEsVersion="0x00020000"
          android:required="true" />
        <application
          android:allowBackup="true"
          android:icon="@drawable/ic_launcher"
          android:label="@string/app_name"
          android:theme="@style/AppTheme" >
          <activity
            android:name="com.example.googlemaps.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>
          <meta-data
```



```
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyDKymeBXNeIFWY5jRUejv6zIpmr2MVyQ0" />
        </application>
    </manifest>
```

Sample output:



13. CREATE A SAMPLE APPLICATION WITH LOGIN MODULE(CHECK USER NAME AND PASSWORD) ON SUCCESSFUL LOGIN CHANGE TEXTVIEW "LOGIN SUCCESSFUL". ON LOGIN FAIL ALERT USING TOAST "LOGIN FAIL"

activity_main.xml

```
<TextView
    android:id="@+id/tvName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="21dp"
    android:layout_marginTop="49dp"
    android:text="User Name"
    android:textSize="18sp" />
<EditText
    android:id="@+id/etUsername"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
        android:layout_alignBaseline="@+id/tvName"
        android:layout_alignBottom="@+id/tvName"
        android:layout_alignParentEnd="true"
        android:layout_marginEnd="23dp"
        android:ems="10"
        android:inputType="textPersonName" />
    <TextView
        android:id="@+id/tvPass"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignEnd="@+id/tvName"
        android:layout_below="@+id/etUsername"
        android:layout_marginTop="32dp"
        android:text="Password"
        android:textSize="18sp" />
    <EditText
        android:id="@+id/etPassword"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/tvPass"
        android:layout_alignBottom="@+id/tvPass"
        android:layout_alignStart="@+id/etUsername"
        android:ems="10"
        android:inputType="textPassword" />
<Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/etPassword"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="38dp"
        android:text="LOGIN"
        />
    <TextView
        android:id="@+id/tvLoginStatus"
```



```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/button"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="100sp"
        />
</RelativeLayout>
package com.example.helloworld;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
    EditText etUsername, etPassword;
    Button btnStatus;
    TextView tvLoginStatus;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        etUsername = (EditText) findViewById(R.id.etUsername);
        etPassword = (EditText) findViewById(R.id.etPassword);
        btnStatus = (Button) findViewById(R.id.button);
        tvLoginStatus = (TextView) findViewById(R.id.tvLoginStatus);
        btnStatus.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                check();
            }
        });
    }
    public void check(){
```

```
    if(etUsername.getText().toString().equals("tonystark") && etPassword.getText().toString().
equals("loveyou3000")){
            tvLoginStatus.setText("Login successful");
        }else{
            Toast.makeText(this, "Login fail", Toast.LENGTH_LONG).show();
        }
    }
}
```

Sample Output:

14. LEARN TO DEPLOY ANDROID APPLICATIONS.

MainActivity.java

```
package com.example.myfirstapp;
import androidx.appcompat.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);
    }
}
```

activity_main.xml (layout file);

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, World!"
        android:textSize="24sp"
        android:layout_centerInParent="true"/>
</RelativeLayout>
```

AndroidManifest.xml (configuration file);

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

```
    </application>
</manifest>
```

strings.xml (resource file for storing strings):

```
<resources>
    <string name="app_name">MyFirstApp</string>
</resources>
```

This is a basic "Hello World" Android application. To deploy it

1. Set up android studio,
2. Create new project.
3. Copy the provided code into the respective files.
4. Buil and run the applications on emulator or physical Android testing

To deploy the application to production, you would typically need to sign the APK, configure release builds, and publish it on Google Play Store or distribute it through other channels. Make sure to follow the official Android Developer documentation for detailed instructions on deployment and distribution.