MANIPAL INSTITUTE OF TECHNOLOGY
Manipal – 576 104

DEPARTMENT OF INFORMATION & COMMUNICATION TECHNOLOGY

CERTIFICATE

This is to certify that Ms./Mr. ……...……Reg.No. …..……Section: ……Roll No……has satisfactorily completed the lab exercises prescribed for **Mobile Application Development Lab** [**ICT 3268**] of Third Year B. Tech. CCE Degree at MIT, Manipal, in the academic year 2025

Date: …….....

Signature of the faculty

CONTENTS

Course Objectives

- To design and develop mobile applications using android.
- To familiarize with mobile UI design.
- To learn Database Connectivity to mobile applications.
- To be able to develop an application using advanced android concepts

Course Outcomes

At the end of this course, students will be able to

- Design and develop mobile applications for real world scenerio.
- Design and Implement mobile UI design.
- Demonstrate database connectivity for mobile applications.
- Implement wireless functionalities such as Bluetooth, etc in android applications.

Evaluation plan

Split up of 60 marks for Regular Lab Evaluation
Record :8 marks 2 regular evaluations will be carried . Each evaluation is for 12 marks.:2X12=24 marks Midterm:20 Marks Internal Project Demo: 8 marks Total Internal Marks: 8+24+20+8 =60 Marks
End Semester Lab evaluation: 20 marks (Duration 2 hrs)
Program write up: 6 Marks Program execution: 14 Marks Total: 6+14 =20 Marks
Mini project evaluation: 20 marks

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the Lab Manual Book and the required stationery to every lab session
2. Be in time and follow the institution dress code
3. Must sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum

In- Lab Session Instructions

- Follow the instructions on the allotted exercises
- Show the program and results to the instructors on completion of experiments
- On receiving approval from the instructor, copy the program and results in the lab record
- Prescribed textbooks and class notes can be kept ready for reference if required

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:
    - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
    - Comments should be used to give the statement of the problem.
    - Statements within the program should be properly indented.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.
- In case a student misses a lab, he/ she must ensure that the experiment is completed before the next evaluation with the permission of the faculty concerned.
- Students missing out lab on genuine reasons like conference, sports or activities assigned by the Department or Institute will have to take **prior permission** from the HOD to attend **additional lab**(with other batch) and complete it **before** the student goes on leave. The student could be awarded marks for the write up for that day provided he submits it during the **immediate** next lab.

- Students who fall sick should get permission from the HOD for evaluating the lab records. However attendance will not be given for that lab.
- Students will be evaluated only by the faculty with whom they are registered even though they carry out additional experiments in other batch.
- Presence of the student during the lab end semester exams is mandatory even if the student assumes he has scored enough to pass the examination
- Minimum attendance of 75% is mandatory to write the final exam.
- If the student loses his book, he/she will have to rewrite all the lab details in the lab record.
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

LAB NO: 1 **Date:**

BASICS OF ANDROID MOBILE APPLICATION DEVELOPMENT TOOL

Objectives

- To familiarize with mobile application development tool.
- To gain knowledge about how to develop simple mobile application using android features.

What is Android?

Android is an open source and Linux-based Operating System for mobile devices such as smartphones and tablet computers. Android was developed by the *Open Handset Alliance*, led by Google, and other companies.

Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.

The first beta version of the Android Software Development Kit SDK was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.

On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 Jelly Bean. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance. The source code for Android is available under free and open source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version 2.

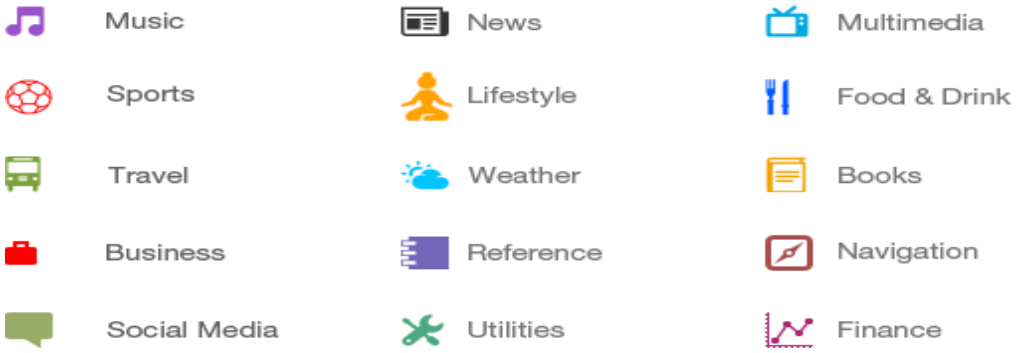
Android Applications

Android applications are usually developed in the Java language using the Android Software Development Kit. Once developed, Android applications can be packaged easily and sold out either through a store such as Google Play, SlideME, Opera Mobile Store, Mobango, F-droid and the Amazon Appstore.

Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It's the largest installed base of any mobile platform and growing fast. Every day more than 1 million new Android devices are activated worldwide.

Categories of Android applications

There are many android applications in the market. The top categories are −



What is API level?

API Level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform.

Android Architecture

Android architecture contains a different number of components to support any Android device's needs. Android software contains an open-source Linux Kernel having a collection of a number of C/C++ libraries which are exposed through application framework services. Among all the components Linux Kernel provides the main functionality of operating system functions to smartphones and Dalvik Virtual Machine (DVM) provide a platform for running an Android application. linux kernel.

Components of Android Architecture

The main components of Android architecture are the following:-

- Applications
- Application Framework
- Android Runtime
- Platform Libraries
- Linux Kernel

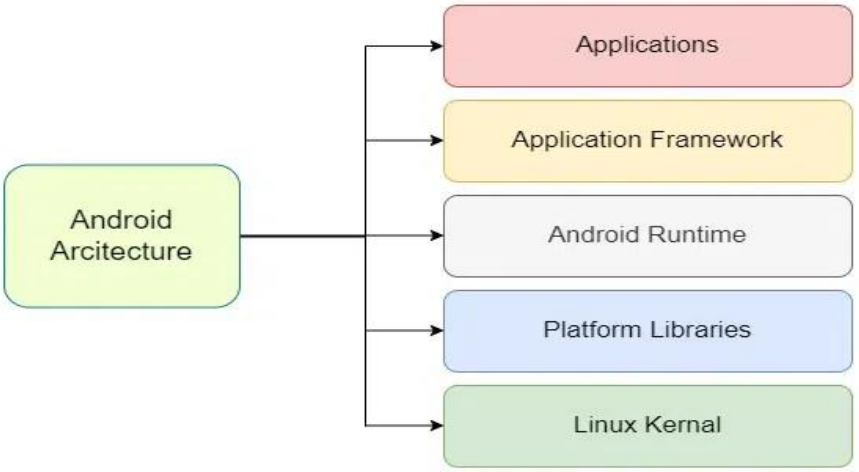
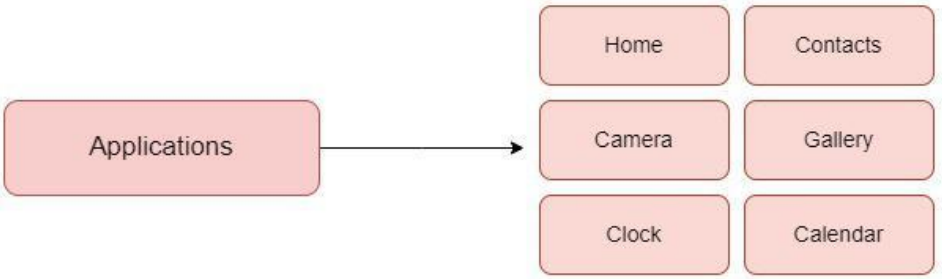


Figure 1: Pictorial representation of Android architecture with several main components and their sub-components(source : Android developers).

Understanding Android's architecture is essential for building efficient applications.

1. Applications

Applications is the top layer of android architecture. The pre-installed applications like home, contacts, camera, gallery etc and third party applications downloaded from the play store like chat applications, games etc. will be installed on this layer only. It runs within the Android run time with the help of the classes and services provided by the application framework.



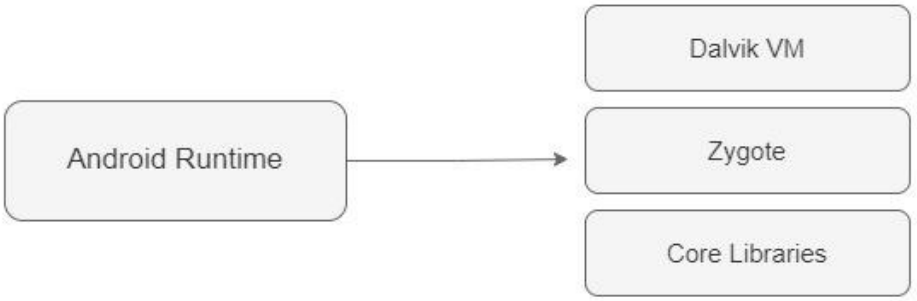
2. **Application framework**

Application Framework provides several important classes which are used to create an Android application. It provides a generic abstraction for hardware access and also helps in managing the user interface with application resources. Generally, it provides the services with the help of which we can create a particular class and make that class helpful for the Applications creation. It includes different types of services activity manager, notification manager, view system, package manager etc. which are helpful for the development of our application according to the prerequisite.

3. **Application runtime**

Android Runtime environment is one of the most important part of Android. It contains components like core libraries and the Dalvik virtual machine(DVM). Mainly, it provides the base for the application framework and powers our application with the help of the core libraries. Like Java Virtual Machine (JVM), Dalvik Virtual Machine (DVM) is a register-based virtual machine and specially designed and optimized for android to

ensure that a device can run multiple instances efficiently. It depends on the layer Linux kernel for threading and low-level memory management. The core libraries enable us to implement android applications using the standard JAVA or Kotlin programming languages.

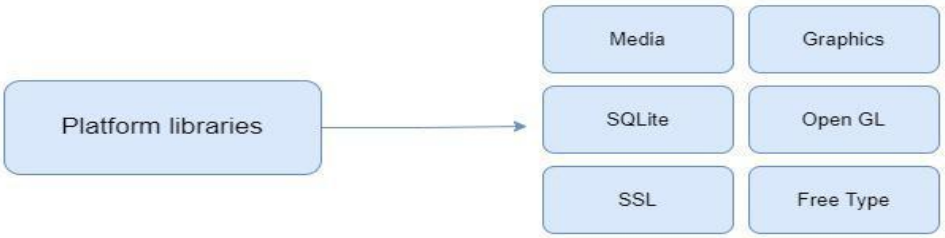


Note: Now, starting from Android 5.0 and above, we use ART (Android Runtime) to compile bytecode into native code to leverage ahead-of-time compilation.

4. **Platform libraries**

The Platform Libraries includes various C/C++ core libraries and Java based libraries such as Media, Graphics, Surface Manager, OpenGL etc. to provide a support for android development.

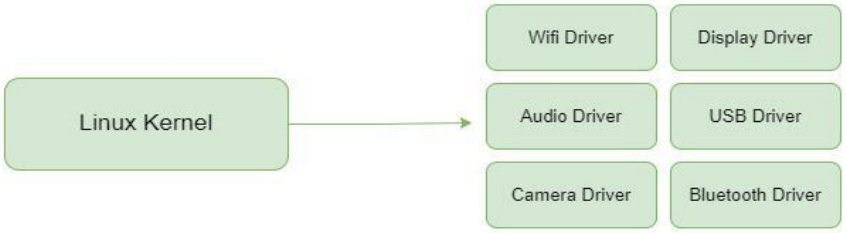
- Media library provides support to play and record an audio and video formats.
- Surface manager responsible for managing access to the display subsystem.
- SGL and OpenGL both cross-language, cross-platform application program interface (API) are used for 2D and 3D computer graphics.
- SQLite provides database support and FreeType provides font support.
- Web-Kit This open source web browser engine provides all the functionality to display web content and to simplify page loading.
- SSL (Secure Sockets Layer) is security technology to establish an encrypted link between a web server and a web browser.



5. **Linux Kernel**

Linux Kernel is heart of the android architecture. It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc. which are required during the runtime. The Linux Kernel will provide an abstraction layer between the device hardware and the other components of android architecture. It is responsible for management of memory, power, devices etc. The features of Linux kernel are:

- Security: The Linux kernel handles the security between the application and the system.
- Memory Management: It efficiently handles the memory management thereby providing the freedom to develop our apps.
- Process Management: It manages the process well, allocates resources to processes whenever they need them.
- Network Stack: It effectively handles the network communication.
- Driver Model: It ensures that the application works properly on the device and hardware manufacturers responsible for building their drivers into the Linux build.



Project structure:

Each project in Android Studio contains one or more modules with source code files and resource files. The types of modules include:

- Android app modules
- Library modules
- Google App Engine modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 1. This view is organized by modules to provide quick access to your project's key source files. All the build files are visible at the top level, under Gradle Scripts.

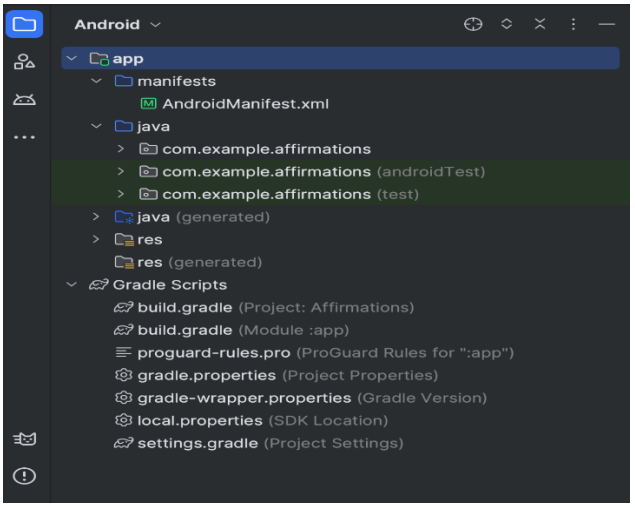


Figure 2. Project files in Android project view.

Each app module contains the following folders:

- manifests: Contains the AndroidManifest.xml file.
- java: Contains the Kotlin and Java source code files, including JUnit test code.
- res: Contains all non-code resources such as UI strings and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select Project instead of Android from the Project menu.

Gradle build system

Android Studio uses Gradle as the foundation of the build system, with more Android-specific capabilities provided by the Android Gradle plugin. This build system runs as an integrated tool from the Android Studio menu and independently from the command line. You can use the features of the build system to do the following:

- Customize, configure, and extend the build process.
- Create multiple APKs for your app with different features, using the same project and modules.
- Reuse code and resources across source sets.

By employing the flexibility of Gradle, you can achieve all of this without modifying your app's core source files.

Android Studio build files are named build.gradle.kts if you use Kotlin (recommended) or build.gradle if you use Groovy. They are plain text files that use the Kotlin or Groovy syntax to configure the build with elements provided by the Android Gradle plugin. Each project has one top-level build file for the entire project and separate module-level build files for each module. When you import an existing project, Android Studio automatically generates the necessary build files.

Manage dependencies

Dependencies for your project are specified by name in the module-level build script. Gradle finds dependencies and makes them available in your build. You can declare module dependencies, remote binary dependencies, and local binary dependencies in your build.gradle.kts file.

Android Studio configures projects to use the Maven Central Repository by default. This configuration is included in the top-level build file for the project.

Debug and profile tools

Android Studio helps you debug and improve the performance of your code, including inline debugging and performance analysis tools.

Inline debugging

Use inline debugging to enhance your code walkthroughs in the debugger view with inline verification of references, expressions, and variable values.

Inline debug information includes:

- Inline variable values
- Objects that reference a selected object
- Method return values
- Lambda and operator expressions
- Tooltip values

To enable inline debugging, in the Debug window, click Settings and select Show Variable Values in Editor.

Performance profilers

Android Studio provides performance profilers so you can easily track your app's memory and CPU usage, find deallocated objects, locate memory leaks, optimize graphics performance, and analyze network requests.

To use performance profilers, with your app running on a device or emulator, open the Android Profiler by selecting View > Tool Windows > Profiler.

Data file access

The Android SDK tools, such as Systrace and Logcat, generate performance and debugging data for detailed app analysis.

To view the available generated data files:

1. Open the Captures tool window.

2. In the list of the generated files, double-click a file to view the data.
3. Right-click any HPROF files to convert them to the standard.
4. Investigate your RAM usage file format.

Lab exercises

Develop an Android application for the following program

1. Create an Android application to show the demo of displaying text with justifying elements, changing text colors ,fonts etc.
2. Find the “hello word” text in the XML document and modify the text.

LAB NO: 2 **Date:**

INTRODUCTION TO ACTIVITY AND LAYOUTS IN ANDROID

Objectives

- To apply the concepts of layouts to enrich the user interface:
- Understanding layout types and how to use them effectively improves the UI design, enhancing the user experience.
- To understand different activity and fragments to use in android application.

Activity

An Activity in Android represents a single screen with a user interface. It acts as the entry point for interacting with the user and is a core component of an Android application. Activities are declared in the AndroidManifest.xml file and can be linked together to create a seamless user experience.

Key Components of an Activity:

Lifecycle: Activities have a lifecycle managed by the Android system, which includes methods such as:

- onCreate(): Initializes the activity and sets up the UI.
- onStart(), onResume(), onPause(), and onStop(): Handle the transition states of the activity.
- onDestroy(): Final cleanup before the activity is destroyed.

Intent Handling: Activities can start other activities or services using intents.

Layouts

Layouts define the structure and appearance of the user interface. They are declared in XML files and referenced in Java code. Layouts determine how widgets (e.g., buttons, text views) are arranged on the screen.

Common Layout Types:

LinearLayout: Aligns children in a single row or column.
RelativeLayout: Positions elements relative to each other or the parent.
ConstraintLayout: Provides more flexibility with constraints between elements.
FrameLayout: Used for a single child view, often for overlays.
TableLayout: Arranges children into rows and columns.

Activities and fragments allow modular and dynamic UI creation. Fragments can be embedded within activities for reusability and better device compatibility (e.g., tablets and phones).

Example Code: Setting up an Activity with a LinearLayout in Java

XML Layout (res/layout/activity_main.xml):

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <TextView
      android:id="@+id/textView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello, World!" />

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

Java Code (MainActivity.java):

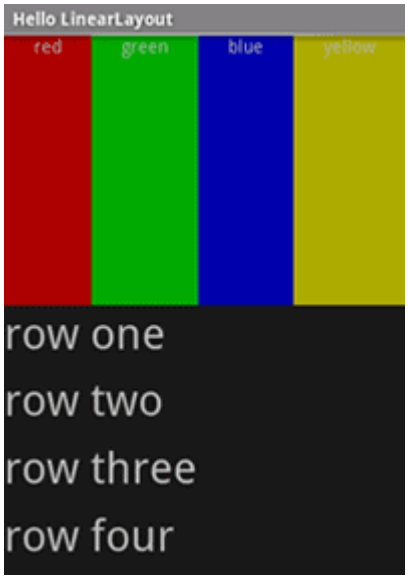
```
package com.example.myapp;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

      TextView textView = findViewById(R.id.textView);
      Button button = findViewById(R.id.button);
      button.setOnClickListener(view -> textView.setText("Button Clicked!"));
   }
}
```

Lab exercises

1. Create an app that illustrates that the activity lifecycle method being triggered by various action. Understand when onCreate(),onStart(),onResume event occur.

2. Create a Calculator app that does the function of multiplication, addition, division, subtraction but displays the result in the format of:-Num1 operator num2 = result. Back button on the next activity should get back to the calculator activity again.

3. Create the following given scenario using linear and relative layout concept.



4. Create an app such that when the user click on the given URL typed by the user, it visits the corresponding page.

LAB NO: 3 **Date:**

ACTIVITY AND LAYOUT CONTINUED

Objectives

- To acquire knowledge on list view, grid view, table view in android application.
- To develop interactive mobile application with multiple pages.

1. TextView:

TextView is a simple view for displaying text on the screen.

Xml:

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Hello, World!" />
```

2. ImageView:

ImageView is used to display images.

```
<ImageView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:src="@drawable/sample_image" />
```

3. Button:

Button is an interactive view that performs an action when clicked.

Xml:

```
<Button
   android:id="@+id/button"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Click Me" />
```

Java code (MainActivity.java):

```
Button button = findViewById(R.id.button);
button.setOnClickListener(view -> {
   // Perform an action
});
```

4. Building Layouts with an Adapter:

An **Adapter** in Android serves as a mediator between a data source and UI components like ListView, GridView, RecyclerView, or TabLayout. Its main role is to bind data to these views and

manage the rendering of each item in the list, grid, or tab. Adapters simplify data binding and UI updates, making them essential for dynamic and interactive mobile application development. They optimize memory and performance by recycling views for displaying data dynamically.

Key Features of the Adapter Class

- **Data Management:** Fetches data from sources such as arrays, lists, or databases.
- **View Creation:** Inflates layouts and populates them with data for individual UI components.
- **Efficiency:** Reuses views for improved performance and memory optimization.

A. Adapter with ListView:

The ListView uses an adapter to display data in a vertically scrollable list. The adapter manages the rows in the list by binding data to them.

Using ArrayAdapter with ListView.:

```
ListView listView = findViewById(R.id.listView);
String[] items = {"Item 1", "Item 2", "Item 3"};
ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, items);
listView.setAdapter(adapter);
```

B. Adapter with GridView:

The GridView displays data in a grid format. An adapter provides the data for each cell in the grid.

Using ArrayAdapter with GridView:

```
GridView gridView = findViewById(R.id.gridView);
String[] items = {"A", "B", "C", "D", "E"};
ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, items);
gridView.setAdapter(adapter);
```

5. TabLayout

TabLayout is an Android UI component that provides a tabbed navigation interface. It is commonly used to switch between different views or fragments within an app. Each tab can contain text, an icon, or both, and clicking on a tab typically updates the displayed content.

Key features:

- **Tabbed Navigation**: Simplifies switching between different sections or views.
- **Integration with ViewPager**: Works seamlessly with ViewPager or ViewPager2 to display content for each tab.
- **Customizable Tabs**: Allows adding text, icons, or custom views to tabs.

Attributes

- **app:tabMode:** Determines the tab layout mode:
  - scrollable: Tabs are scrollable.
  - fixed: Tabs are evenly distributed and fixed.
- **app:tabGravity:** Controls the tab placement.
  - fill: Tabs fill the width of the screen.
  - center: Tabs are centered.
- **app:tabTextColor & app:tabSelectedTextColor:** Customizes tab text color.

TabLayout with ViewPager(using FragmentPagerAdapter class)

TabLayout is often used with a ViewPager or ViewPager2 to display and navigate content dynamically.

Example code:

XML Layout:

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

   <com.google.android.material.tabs.TabLayout
      android:id="@+id/tabLayout"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      app:tabMode="fixed"
      app:tabGravity="fill" />

   <androidx.viewpager.widget.ViewPager
      android:id="@+id/viewPager"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />
</LinearLayout>
```

Java Code:

```
package com.example.tablayoutdemo;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentPagerAdapter;
import androidx.viewpager.widget.ViewPager;
import com.google.android.material.tabs.TabLayout;

public class MainActivity extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
```

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

        TabLayout tabLayout = findViewById(R.id.tabLayout);
        ViewPager viewPager = findViewById(R.id.viewPager);

            viewPager.setAdapter(new
       FragmentPagerAdapter(getSupportFragmentManager())  {
           @Override
           public Fragment getItem(int position) {
              switch (position) {
                 case 0:
                    return new TabFragment("Tab 1 Content");
                 case 1:
                    return new TabFragment("Tab 2 Content");
                 default:
                    return new TabFragment("Tab 3 Content");
              }
           }

           @Override
           public int getCount() {
              return 3; // Number of tabs
           }

           @Override
           public CharSequence getPageTitle(int position) {
              return "Tab " + (position + 1);
           }
        });

        tabLayout.setupWithViewPager(viewPager);
     }
  }
```

6. TableLayout

TableLayout is a layout manager in Android that arranges its child views into rows and columns. Each row is defined as a TableRow, and views are added inside these rows. TableLayout is useful for displaying structured or tabular data such as forms, timetables, or statistics.

Key Features

- R**ow and Column Structure:** Organizes views in a tabular format.
- **Flexible Layout:** Allows cells to span multiple columns.
- **Customizable Alignment:** Each cell can have its own alignment, padding, and margins.
- **No Borders:** By default, TableLayout does not draw grid lines, but you can add them manually with styling or custom views.

Example Code:

XML:

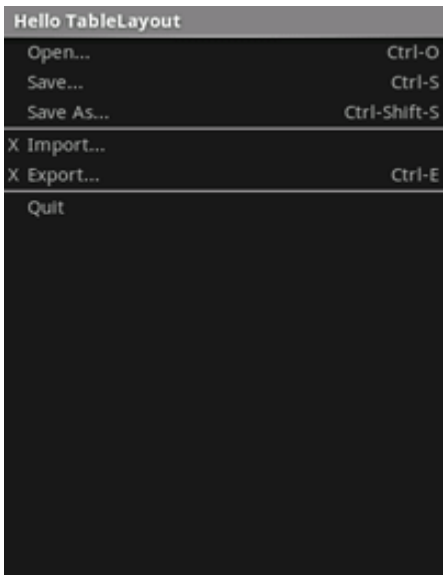
```
<TableLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:stretchColumns="1"> <!-- Stretches the second column -->

   <!-- Header Row -->
   <TableRow>
      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Header 1"
         android:textStyle="bold" />

      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Header 2"
         android:textStyle="bold" />

      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Header 3"
         android:textStyle="bold" />
   </TableRow>

   <!-- Data Row -->
   <TableRow>
      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Data 1" />

      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Data 2" />

      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Data 3" />
   </TableRow>
</TableLayout>
```

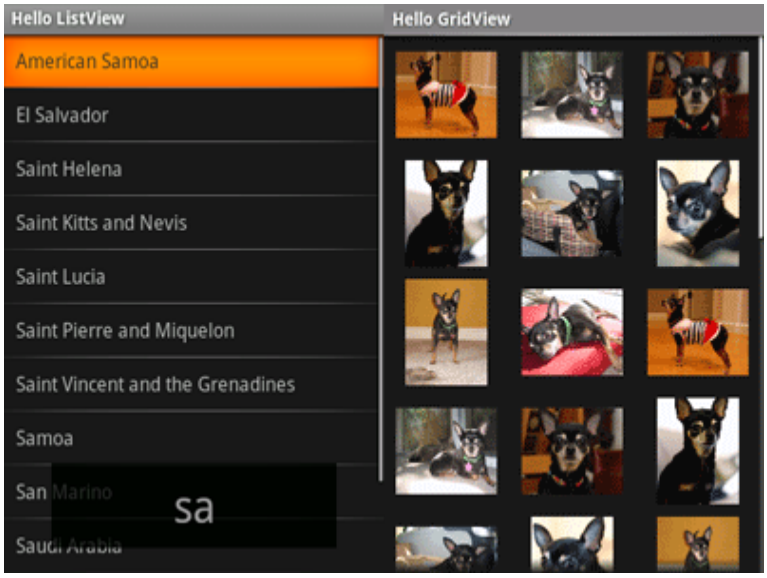
Key Attributes:

Attribute	Description
android:stretchColumns	Specifies which columns should stretch to fill the available space.
android:shrinkColumns	Specifies which columns should shrink to fit the content.
android:collapseColumns	Hides specific columns from view without removing them from the layout.
android:layout_width	Width of the TableLayout (e.g., match_parent, wrap_content).
android:layout_height	Height of the TableLayout (e.g., match_parent, wrap_content).

Lab Exercise:

1. Using given scenario develop an application to perform following layout operations
   - List view
   - Grid view
   - Tab layout
   - Table layout





2. Write a program to list different sports, and when a sport is selected, display a message showing the selected sport.
3. Design an news application and Implement the navigation between sections like Top Stories, Sports, and Entertainment using tab layout.

LAB NO: 4 **Date:**

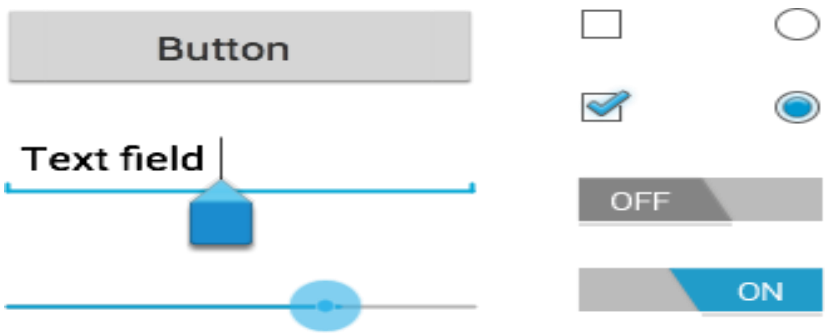
INPUT CONTROLS IN ANDROID

Objectives

- To learn the usage of interactive components in application.
- To learn the interactive mobile application development using variety of control inputs.

Input controls

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



Each of these UI controls serves specific purposes in Android app development:

- Buttons for actions.
- Text fields for data input.
- SeekBar for range selection.
- Checkboxes and radio buttons for choices.
- ToggleButton and Switch for binary states.

These controls can be customized with attributes like size, color, and event handlers to suit the application's needs.

1. **Button**
   A Button is a clickable UI element used to perform actions when tapped.

```
<Button
   android:id="@+id/button"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Click Me" />
```

A. Styling Your Button

The appearance of your button (background image and font) may vary from one device to another, because devices by different manufacturers often have different default styles for input controls.

You can control exactly how your controls are styled using a theme that you apply to your entire application. For instance, to ensure that all devices running Android 4.0 and higher use the Holo theme in your app, declare android:theme="@android:style/Theme.Holo" in your manifest's <application> element. Also read the blog post, Holo Everywhere for information about using the Holo theme while supporting older devices.

To customize individual buttons with a different background, specify the android:background attribute with a drawable or color resource. Alternatively, you can apply a style for the button, which works in a manner similar to HTML styles to define multiple style properties such as the background, font, size, and others. For more information about applying styles, see Styles and Themes.

B. Implementing OnClickListener in Java

The OnClickListener interface is used to detect click events on UI components like buttons, checkboxes, etc.

Steps:

- Implement View.OnClickListener in your activity or fragment.
- Override the onClick(View v) method.
- Attach the listener to the UI control using setOnClickListener.

Code snippet for Using OnClickListener with a Button

```
Button button = findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      Toast.makeText(MainActivity.this, "Button Clicked!",
Toast.LENGTH_SHORT).show();
   }
});
```

C. Toast message:

A **Toast** is a small message that pops up on the screen to provide feedback or information to the user. It is non-intrusive, meaning it doesn't interrupt the user's interaction with the app and automatically disappears after a brief duration. **Toast** messages are used to display short notifications or simple alerts, such as "Data saved" or "Operation successful."

Key Features of Toast:

- **Short Duration:** The message appears for a brief moment and disappears automatically.
- **Non-Intrusive:** Does not require any user interaction to dismiss.
- **Customizable:** You can customize the position, duration, and text appearance.

Syntax:

Toast.makeText(context, "Message to show", Toast.LENGTH_SHORT).show();
where

- context: The current context, usually an activity or application context.
- "Message to show": The message text you want to display.
- Toast.LENGTH_SHORT: Duration for the toast to be visible (short duration).
- Toast.LENGTH_LONG: Alternative to display the toast for a longer duration.
- .show(): This is used to display the Toast on the screen.

2. **SeekBar**

A SeekBar is a slider control that allows users to select a value from a range.

XML code snippet:

```
<SeekBar
   android:id="@+id/seekBar"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:max="100" />
```

3. **CheckBox**

A CheckBox is a toggleable UI element that allows users to select multiple options from a set.

XML code snippet:

```
<CheckBox
   android:id="@+id/checkBox"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Option 1" />
```

4. **ZoomButton**

A ZoomButton is a special type of button for zooming in and out.

XML code snippet:

```
<ZoomButton
   android:id="@+id/zoomButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Zoom" />
```

5. **ToggleButton**

   A ToggleButton is a switch-like control that toggles between two states: ON and OFF.

   XML code snippet:

```
<ToggleButton
  android:id="@+id/toggleButton"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textOn="ON"
  android:textOff="OFF" />
```

6. **RadioButton and RadioGroup**
   a. RadioButton: Allows users to select a single option from a group.
   b. RadioGroup: Groups multiple RadioButton controls for exclusive selection.

   RadioGroup with RadioButton

```
<RadioGroup
  android:id="@+id/radioGroup"
  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:text="Option 1" />

  <RadioButton
    android:id="@+id/radioButton2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Option 2" />
</RadioGroup>
```

7. **Switch:**
   A Switch is similar to a ToggleButton but has a more modern UI for ON/OFF states.

```
<Switch
  android:id="@+id/switch"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Switch" />
```

8. **Intents:**
   An Intent in Android is a messaging object that facilitates communication between different components of an app (such as activities, services, and broadcast receivers) and between different apps. Intents are used to perform actions like opening a new activity, sending data to another component, starting a service, broadcasting a message, etc.

Intents come in two types:

Explicit Intents: Used when you want to start a specific activity or service within the same app.
Example: Opening a specific activity in your app.

Implicit Intents: Used when you want to perform an action that is not limited to a specific app or activity, such as sharing data, opening a URL, etc. The system finds the appropriate component to handle the request (e.g., opening a browser, sending an email).
Example: Opening a web page in a browser or sending an email.
Syntax:
Explicit Intent Syntax
To navigate between activities or start a service within the same app:

```
Intent intent = new Intent(CurrentActivity.this, TargetActivity.class);
intent.putExtra("key", "value");  // Optional: Pass data using key-value pairs
startActivity(intent);
```

Implicit Intent Syntax
To invoke components of other apps or services:

```
// Implicit intent to open a webpage in the browser
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse ("http:// www.example.com" ) );
startActivity(intent);

// Implicit intent to send an email
Intent intent = new Intent(Intent.ACTION_SENDTO,
Uri.parse("mailto:someone@example.com"));
intent.putExtra(Intent.EXTRA_SUBJECT, "Subject Text");
intent.putExtra(Intent.EXTRA_TEXT, "Email body text");
startActivity(intent);
```

Lab exercises:

1. Develop a "Test App" that includes a layout with a **Button** and a **ToggleButton**. When each button is clicked, a custom **Toast message** should be displayed with different images as their content.

2. Create an app that contains a view with **multiple buttons**, each labeled with different Android versions. When a button is clicked, a **Toast message** should appear, displaying the corresponding Android version's name along with its associated icon.

3. Develop a view with a **ToggleButton** labeled "Current Mode" that has two states: "Wi-Fi" and "Mobile Data." Based on the state of the toggle button, an image corresponding to the selected mode should appear, and a **Toast message** should display the current mode. Additionally, when the user clicks the "Change Mode" button, the app should switch to the corresponding mode and update the image accordingly.

4. Create a “Food Ordering App” which **lists** food items with **check boxes**. Once the user checks /unchecks the item and click on the submit button display the items ordered along with cost of each item and total cost in a new activity. Once the user clicks on the submit button he/she should not be allowed to change the order i.e. he should not be allowed to change the state of the items checked.

LAB NO.: 5 **Date:**

INPUT CONTROLS-SPINNERS, PICKERS

Objectives

- To introduce the concept of spinners and pickers.
- To develop an interactive mobile UI for mobile phones.

Spinners and **Pickers** are common UI components in Android Studio used to allow users to select an option from a predefined list. They are user-friendly and essential in many applications.

Spinners

Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.

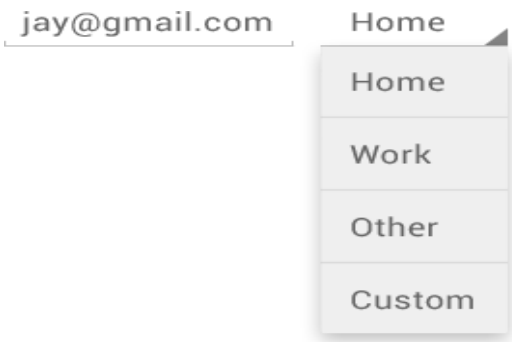


Figure 1: Spinners (source credits: internet)

How to Use a Spinner:

Add Spinner to Layout

Add a Spinner to your XML layout file:

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

Populate Spinner

Use an ArrayAdapter to populate the spinner with data:

```
Spinner spinner = findViewById(R.id.spinner);
ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_item, data);
```

```
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_ite
m);
spinner.setAdapter(adapter);
```

Handle Item Selection

Implement OnItemSelectedListener to handle user selection:

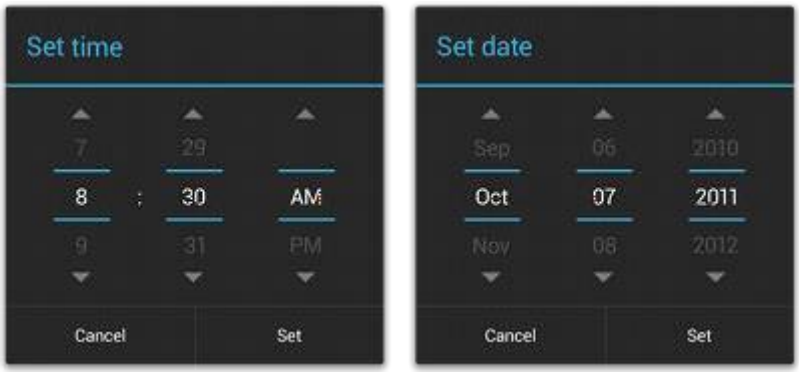
```
spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
   @Override
   public void onItemSelected(AdapterView<?> parent, View view, int position, long
id) {
      String selectedItem = parent.getItemAtPosition(position).toString();
      Toast.makeText(getApplicationContext(), "Selected: " + selectedItem,
Toast.LENGTH_SHORT).show();
   }
   @Override
   public void onNothingSelected(AdapterView<?> parent) {
      // Handle the case where no item is selected
   }
});
```

2. Pickers

Android provides controls for the user to pick a time or pick a date as ready-to-user dialogs. Each picker provides controls for selecting each part of the time (hour, minute, AM/PM) or date (month, day, year). Pickers allow users to select a specific value, such as a date, time, or number.

Types of Pickers:

- **DatePicker**: Allows users to select a date.
- **TimePicker**: Allows users to select a time.



Example: DatePickerDialog

```
DatePickerDialog datePickerDialog = new DatePickerDialog(this, (view, year,
month, dayOfMonth) -> {
   String date = dayOfMonth + "/" + (month + 1) + "/" + year;
   Toast.makeText(this, "Selected Date: " + date,
Toast.LENGTH_SHORT).show();
}, year, month, day);
```

```
datePickerDialog.show();
```

Example: TimePickerDialog

```
TimePickerDialog timePickerDialog = new TimePickerDialog(this, (view, hourOfDay, minute) -> {
   String time = hourOfDay + ":" + minute;
   Toast.makeText(this, "Selected Time: " + time, Toast.LENGTH_SHORT).show();
}, hour, minute, true);
timePickerDialog.show();
```

Advanced Methods for Spinners and Pickers

1. Customizing Spinners:

Custom Layout for Spinner Items:

Create a custom XML layout for spinner items to modify their appearance (e.g., fonts, colors, etc.):

```
<TextView
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:textColor="#FF0000"
   android:padding="10dp" />
```

Use Custom Adapters:

Create a custom adapter by extending ArrayAdapter or BaseAdapter to handle complex datasets or designs.

Adding images in Spinners can enhance the user interface, especially when dealing with options that are better represented visually (e.g., flags for countries or product images).

Adding images in Spinners can enhance the user interface, especially when dealing with options that are better represented visually (e.g., flags for countries or product images).

Steps to Add Images in Spinners

1. **Create a Custom Layout for Spinner Items** Design a custom XML layout for each item in the Spinner. This layout can include an ImageView alongside a TextView.

Example: spinner_item.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:orientation="horizontal"
   android:padding="8dp">
```

```
    <ImageView
        android:id="@+id/image"
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:layout_marginEnd="8dp" />

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:textColor="#000" />
</LinearLayout>
```

2. **Prepare the Data** Create a data class to hold the text and image resource for each item.
   Example: Data Class

```
public class SpinnerItem {
    private String text;
    private int imageResId;

    public SpinnerItem(String text, int imageResId) {
        this.text = text;
        this.imageResId = imageResId;
    }
    public String getText() {
        return text;
    }
    public int getImageResId() {
        return imageResId;
    }
}
```

3. **Create a Custom Adapter** Extend ArrayAdapter to bind the custom layout and data to the Spinner.
   Example: CustomAdapter

```
public class CustomSpinnerAdapter extends ArrayAdapter<SpinnerItem> {
    public CustomSpinnerAdapter(Context context, List<SpinnerItem> items) {
        super(context, 0, items);
    }
    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull
ViewGroup parent) {
```

```
        return createView(position, convertView, parent);
    }
    @Override
    public View getDropDownView(int position, @Nullable View convertView,
@NonNull ViewGroup parent) {
        return createView(position, convertView, parent);
    }
    private View createView(int position, View convertView, ViewGroup parent)
{
        if (convertView == null) {
            convertView                                                    =
LayoutInflater.from(getContext()).inflate(R.layout.spinner_item, parent, false);
        }
        SpinnerItem currentItem = getItem(position);
        ImageView imageView = convertView.findViewById(R.id.image);
        TextView textView = convertView.findViewById(R.id.text);

        if (currentItem != null) {
            imageView.setImageResource(currentItem.getImageResId());
            textView.setText(currentItem.getText());
        }
        return convertView;
    }
}
```

4. **Bind the Adapter to the Spinner** Populate the Spinner with a list of SpinnerItem objects and attach the custom adapter.

Example: MainActivity

```
Spinner spinner = findViewById(R.id.spinner);
List<SpinnerItem> items = new ArrayList<>();
items.add(new SpinnerItem("Apple", R.drawable.apple));
items.add(new SpinnerItem("Banana", R.drawable.banana));
items.add(new SpinnerItem("Cherry", R.drawable.cherry));

CustomSpinnerAdapter adapter = new CustomSpinnerAdapter(this, items);
spinner.setAdapter(adapter);

spinner.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position,
long id) {
```

```
        SpinnerItem selectedItem = (SpinnerItem) parent.getItemAtPosition(position);
        Toast.makeText(getApplicationContext(), "Selected: " + selectedItem.getText(), Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        // Handle the case where no item is selected
    }
});
```

Lab exercise:

1. Develop an application named "Vehicle Parking Registration" where the user can register their vehicle for parking. The app should include a spinner that allows users to select the type of vehicle (e.g., car, bike, etc.) and text fields for entering the vehicle number and RC number. Upon clicking the submit button, the entered details should be displayed in a separate view, providing the user with options to either confirm the details or edit them. Once the user confirms the information, a toast message should appear showing a unique serial number to confirm the parking allotment.
2. Design and develop a "Travel Ticket Booking" app where users can book tickets by selecting the source and destination from dropdown lists (spinners) and choosing the date of travel using a date picker. Include a toggle button to let users specify whether they want a one-way ticket or a round-trip ticket. The app should have "Submit" and "Reset" buttons. When the "Submit" button is clicked, display the entered details on a new screen in a structured format. The "Reset" button should clear all input fields and set the date picker to the current system date and time.
3. Design and develop a "Movie Ticket Booking" app where users can book tickets by selecting the movie and theatre from dropdown menus (spinners), the date of the show using a date picker, and the preferred showtime using a time picker. Include a toggle button to let users choose between a standard ticket or a premium ticket. If the "premium" option is selected, ensure the "Submit" button becomes clickable only after 12:00 PM. The app should have "Book Now" and "Reset" buttons. When the user clicks "Book Now," display the entered details along with available seats for the selected show in a new screen in a well-organized format. The "Reset" button should clear all fields and reset the date picker to the current system date. Validate all inputs to ensure correct data entry.

LAB NO. 6 **Date:**

INTRODUCTION TO MENU

Objectives

- To understand how menu works with android and how to make it runnable in web browser.
- To learn how to create options menu and an app bar using android.
- To learn how to create context menu using android.
- To learn how to create a pop up menu in android

What is a Menu?

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

1. **Option menu:**
   The Options Menu displays global actions (e.g., Search, Settings) relevant to the current activity. It appears as a three-dot overflow menu or action buttons in the App Bar.

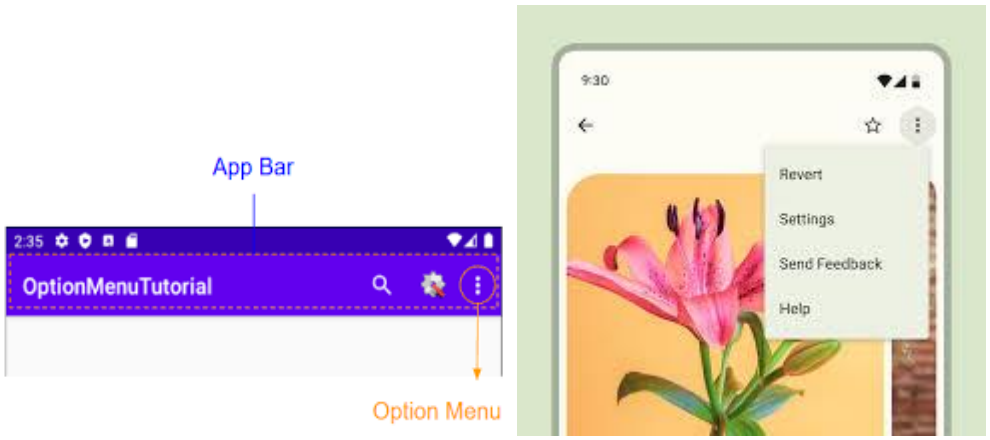


Figure 1: Option Menu example(source : Android developers)

A. Define the Menu in XML

Create a menu resource file in the res/menu/ directory.

Example code: res/menu/menu_main.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <!-- Menu item displayed as an icon in the app bar -->
  <item
```

```
        android:id="@+id/action_search"
        android:title="Revert"
        android:icon="@drawable/ic_search"
        android:showAsAction="ifRoom" />

    <!-- Menu item displayed only in the overflow menu -->
    <item
        android:id="@+id/action_settings"
        android:title="Settings"
        android:showAsAction="never" />
</menu>
```

where

android:id: Unique ID for the menu item.
android:icon: Icon for the menu item.
android:title: Text displayed for the menu item.
android:showAsAction: Determines when to display as an action (e.g., ifRoom, always, never).

B. Inflate the Menu in the Activity

Inflate the menu resource and handle item clicks.

Example: MainActivity.java

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the app bar.
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle menu item clicks
    switch (item.getItemId()) {
        case R.id.action_revert:
            // Perform revert action
            return true;
        case R.id.action_settings:
            // Open settings
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

2. App Bar:

The App Bar (formerly known as the Action Bar) provides a consistent space at the top of the screen for navigation and key actions. It integrates with the options menu to display important actions as icons or text.

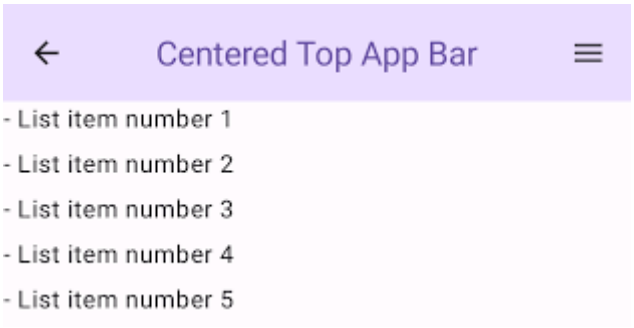


Figure 2: Centered Top App Bar(source : Android developers)

A. Add an App Bar (Toolbar)

Use the Toolbar widget in your activity layout.

Example: res/layout/activity_main.xml

```
<androidx.appcompat.widget.Toolbar
   android:id="@+id/toolbar"
   android:layout_width="match_parent"
   android:layout_height="?attr/actionBarSize"
   android:background="?attr/colorPrimary"
   android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
   android:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

Set the Toolbar as the App Bar in your activity:

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

   // Set the toolbar as the app bar
   Toolbar toolbar = findViewById(R.id.toolbar);
   setSupportActionBar(toolbar);
}
```

Lab exercises:

1. Design a home page for the "XYZ Fitness Center" using an Options Menu with the following Requirements.
    a. Create a simple option menus where in once you click the “menu options”, the option items should get displayed which are "Workout Plans," "Trainers," "Membership". "Workout Plans" should display the list of workout programs (e.g., Weight Loss, Cardio). "Trainers" should display the names and specializations of trainers with their photos. "Membership" should show the membership packages with pricing details.
    b. This option menu uses images/icons instead of textual content. This textual content should represent “Contact US”, “About Us”,” Homepage”. Once the user clicks on the corresponding icons it should display corresponding content.

LAB NO. 7 **Date:**

CREATING CONTEXTUAL AND POP-UP MENUS

Objectives

- To demonstrate significant experience with the contextual menu's
- To create suitable application using pop menus.

1. Implementation of Contextual Actions in Android

A. Floating Context Menu

A menu appears as a floating list of menu items (similar to a dialog) when the user performs a long-click (press and hold) on a view that declares support for a context menu. Users can perform a contextual action on one item at a time.

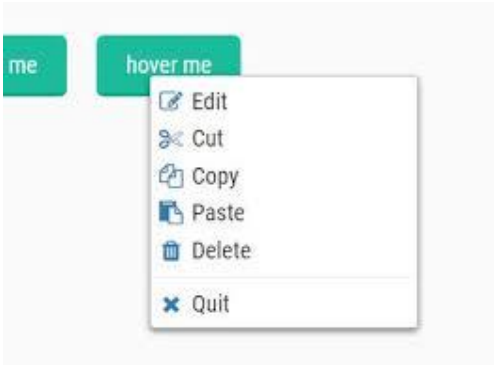


Figure 1: **Floating Context Menu**

Steps to Implement:

1. Register the view for the context menu.
2. Override methods to create and handle the menu.

Java Code Example:

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

   // Register the view for the context menu
   View view = findViewById(R.id.my_view);
   registerForContextMenu(view);
}

// Create the context menu
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo) {
```

```
    super.onCreateContextMenu(menu, v, menuInfo);
    getMenuInflater().inflate(R.menu.context_menu, menu);
}

// Handle context menu item clicks
@Override
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_edit:
            // Handle edit action
            return true;
        case R.id.action_delete:
            // Handle delete action
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

Changes Needed: Define a context_menu.xml in the res/menu/ directory and Register the view using registerForContextMenu().

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/action_edit" android:title="Edit" />
    <item android:id="@+id/action_delete" android:title="Delete" />
</menu>
```

B. Contextual Action Mode

This mode is a system implementation of ActionMode that displays a contextual action bar at the top of the screen with action items that affect the selected item(s). When this mode is active, users can perform an action on multiple items at once (if your app allows it). This provides a contextual action bar at the top for multiple item actions.

Figure 3: **Contextual Action Mode**

Steps to Implement:

- Start ActionMode when an item is selected.
- Inflate and handle the menu in ActionMode.Callback.

Java code Example:

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

   // Example ListView setup
   ListView listView = findViewById(R.id.my_list);
   listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);

   // Set MultiChoiceModeListener for the ListView
   listView.setMultiChoiceModeListener(new
AbsListView.MultiChoiceModeListener() {
      @Override
      public void onCreateActionMode(ActionMode mode, Menu menu) {
         mode.getMenuInflater().inflate(R.menu.contextual_action_menu,
menu);
      }

      @Override
      public void onItemCheckedStateChanged(ActionMode mode, int
position, long id, boolean checked) {
         // Update the number of selected items, if needed
      }

      @Override
      public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
         return false;
      }

      @Override
      public boolean onActionItemClicked(ActionMode mode, MenuItem item)
{
         switch (item.getItemId()) {
            case R.id.action_delete:
               // Perform delete action
               mode.finish(); // Close the action mode
               return true;
            default:
```

```
            return false;
        }
    }

    @Override
    public void onDestroyActionMode(ActionMode mode) {
        // Cleanup when the action mode is closed
    }
  });
}
```

Main Changes Needed: Define a contextual_action_menu.xml in the res/menu/ directory:

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/action_delete"
      android:title="Delete"
      android:icon="@drawable/ic_delete"
      android:showAsAction="ifRoom" />
</menu>
```

2. Creating a Popup Menu

A PopupMenu is a modal menu anchored to a View. It appears below the anchor view if there is room, or above the view otherwise. It's useful for:

- Providing an overflow-style menu for actions that relate to specific content (such as Gmail's email headers, shown in figure 4).
- Providing a second part of a command sentence (such as a button marked "Add" that produces a popup menu with different "Add" options).
- Providing a drop-down similar to Spinner that does not retain a persistent selection.

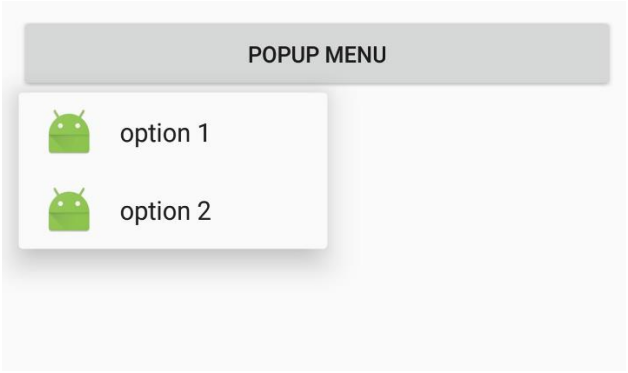


Figure 4: Pop-Up Menu

If you define your menu in XML then:

1. Instantiate a PopupMenu with its constructor, which takes the current application Context and the View to which the menu should be anchored.
2. Use MenuInflater to inflate your menu resource into the Menu object returned by PopupMenu.getMenu().
3. Call PopupMenu.show().

Java Code Example:

```
package com.example.popupmenudemo;
import android.os.Bundle;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.Toast;
import android.widget.Button;
import android.widget.PopupMenu;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

      // Reference to the Button that triggers the PopupMenu
      Button popupButton = findViewById(R.id.popupButton);

      // Set a click listener to show the PopupMenu
      popupButton.setOnClickListener(view -> {
         // Step 1: Instantiate PopupMenu with Context and the anchor View
         PopupMenu popupMenu = new PopupMenu(MainActivity.this, view);

         // Step 2: Inflate the menu resource into the PopupMenu
         MenuInflater inflater = popupMenu.getMenuInflater();
         inflater.inflate(R.menu.popup_menu, popupMenu.getMenu());

         // Step 3: Set click listener for menu item selection
         popupMenu.setOnMenuItemClickListener(item -> {
            switch (item.getItemId()) {
               case R.id.option_one:
                  Toast.makeText(MainActivity.this, "Option One Selected",
Toast.LENGTH_SHORT).show();
                  return true;
               case R.id.option_two:
                  Toast.makeText(MainActivity.this, "Option Two Selected",
Toast.LENGTH_SHORT).show();
                  return true;
               case R.id.option_three:
```

```
                Toast.makeText(MainActivity.this, "Option Three Selected",
        Toast.LENGTH_SHORT).show();
                return true;
            default:
                return false;
        }
      });

      // Step 4: Display the PopupMenu
      popupMenu.show();
    });
  }
}
```

XML Files: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="match_parent"
  android:layout_height="match_parent"
  android:padding="16dp">

  <Button
    android:id="@+id/popupButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Show Popup Menu"
    android:layout_centerInParent="true" />
</RelativeLayout>
```

res/menu/popup_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item
    android:id="@+id/option_one"
    android:title="Option One" />
  <item
    android:id="@+id/option_two"
    android:title="Option Two" />
  <item
    android:id="@+id/option_three"
    android:title="Option Three" />
</menu>
```

Lab Questions:

1. Design and develop an application that displays a list of installed applications on your device. On long press of any application, display the following options:
   a) Show whether the application is a system app or user-installed.
   b) Provide options to open the app, uninstall it, or view its details (e.g., version, storage usage).
   c) Indicate whether the app has special permissions enabled, like location or camera access.
   When an option like **"View Details"** is selected, navigate to a detailed view showing the app's permissions, size, and version. For the **"Uninstall"** option, prompt the user for confirmation before proceeding.
2. Create a View with the name “My Menu ” which contains an image as an icon.On click of that icon it shows a submenu as ”Image -1”,”Image -2” .On click of each item that particular image should be displayed with the corresponding content in the Toast.

3. Create a view that displays textual content providing a description of "Digital Transformation". Add a filter option with submenus:
   a. "Search Keywords" - Allow the user to input keywords that can be searched within the content.
   b. "Highlight" - Highlight specific words or phrases provided by the user in the document.
   c. "Sort" - Provide an option to sort the entire content either alphabetically or by relevance to the search keywords.

LAB NO. 8 **Date:**

ANDROID SQLITE & SHARED PREFERENCES

Objectives

- To apply and manipulate several core SQL Queries through SQLite in Android
- To learn and understand SQLite Browser.
- To apply Shared Preference for storing data of an application.

Android 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 a lightweight, open-source relational database engine that is built into Android. It is used for storing structured data locally on the device. Android provides a set of APIs to work with SQLite databases, enabling developers to create, query, update, and manage data for their applications. It provides SQLiteOpenHelper to simplify database management.
Here, we are going to see the example of sqlite to store and fetch the data. Data is displayed in the logcat.

1. Creating a Database and Table

SQLite databases in Android are created and managed using the SQLiteOpenHelper class.
Example:

```
public class MyDatabaseHelper extends SQLiteOpenHelper {
private static final String DATABASE_NAME = "MyDatabase.db";
private static final int DATABASE_VERSION = 1;

// SQL command to create a table
private static final String CREATE_TABLE =
   "CREATE TABLE Users (" +
   "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
   "name TEXT NOT NULL, " +
   "email TEXT UNIQUE);";

public MyDatabaseHelper(Context context) {
   super(context, DATABASE_NAME, null, DATABASE_VERSION);
}

@Override
public void onCreate(SQLiteDatabase db) {
   db.execSQL(CREATE_TABLE); // Create the table
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
   db.execSQL("DROP TABLE IF EXISTS Users"); // Drop old table
```

```
        onCreate(db); // Recreate the table
    }
}
```

2. Inserting Data

To insert data into the database, use the insert() method or raw SQL.

Using ContentValues:

```
SQLiteDatabase db = dbHelper.getWritableDatabase();
ContentValues values = new ContentValues();
values.put("name", "John Doe");
values.put("email", "john.doe@example.com");
long newRowId = db.insert("Users", null, values); // Returns row ID
```

Using Raw SQL:

```
db.execSQL("INSERT INTO Users (name, email) VALUES ('Jane Doe', 'jane.doe@example.com');");
```

3. Querying Data

To retrieve data, use the query() method or raw SQL with rawQuery().

Using query():

```
Cursor cursor = db.query(
    "Users",                // Table name
    new String[]{"id", "name"}, // Columns to retrieve
    "email = ?",            // WHERE clause
    new String[]{"john.doe@example.com"}, // Arguments
    null, null, null        // GroupBy, Having, OrderBy
);

while (cursor.moveToNext()) {
    int id = cursor.getInt(cursor.getColumnIndexOrThrow("id"));
    String name = cursor.getString(cursor.getColumnIndexOrThrow("name"));
    // Use retrieved data
}
cursor.close();
```

Using Raw SQL:

```
Cursor cursor = db.rawQuery("SELECT * FROM Users WHERE email = ?",
        new String[]{"john.doe@example.com"});
```

4. **Updating Data**

To update data, use the update() method or raw SQL.

Using update():

```
ContentValues values = new ContentValues();
values.put("name", "John Smith");
int rowsAffected = db.update("Users", values, "email = ?", new
String[]{"john.doe@example.com"});
```

Using Raw SQL:

```
db.execSQL("UPDATE Users SET name = 'John Smith' WHERE email =
'john.doe@example.com';");
```

5. **Deleting Data**

To delete data, use the delete() method or raw SQL.

Using delete():

```
int rowsDeleted = db.delete("Users", "email = ?", new String[]{"john.doe@example.com"});
```

Using Raw SQL:

```
db.execSQL("DELETE FROM Users WHERE email = 'john.doe@example.com';");
```

6. **Closing the Database**

Always close the database when you are done to release resources.

```
db.close();
```

To view the SQLite database in an Android application

Use Android Device Monitor (for older Android Studio versions)

Steps:

1. Open Android Studio.
2. Run your application on an emulator or a physical device.
3. Go to Tools > Device File Explorer.
4. Navigate to the database location:
   - **Path**: /data/data/<your.package.name>/databases/
5. Locate your database file (e.g., DemoDB).
6. Right-click the database file and select **Save As** to download it to your computer.
7. Use an SQLite viewer (e.g., DB Browser for SQLite) to open and view the database.

Use Device File Explorer (for newer Android Studio versions)

Steps:

1. Open Android Studio.
2. Open View > Tool Windows > Device File Explorer.
3. In the Device File Explorer, navigate to:
   - /data/data/<your.package.name>/databases/
4. Find your database file.
5. Right-click the database and choose **Save As** to export it to your computer.
6. Use an external SQLite viewer to inspect the database.

Shared Preferences in Android

What Are Shared Preferences?

Shared Preferences in Android provide a simple way to store small amounts of data as key-value pairs. This data is persistent, meaning it remains available even after the app is closed or the device is restarted.

Shared Preferences are typically used for saving application preferences, user settings, or lightweight data that does not require a full database.

When to Use Shared Preferences

- **User Preferences:** Storing user-selected settings like themes (dark/light mode), notification preferences, or language selection.
- **Login State:** Maintaining login sessions, such as storing a boolean flag to check if the user is logged in.
- **Configuration Flags:** Saving configuration flags like the first-time launch state (isFirstLaunch).
- **Caching Small Data:** Temporarily storing small data that does not require a database.

Working with Shared Preferences

1. **File Storage:**
    - Shared Preferences are stored as an XML file in the app's private directory (/data/data/<package_name>/shared_prefs).
2. **Data Access:**
    - Data is accessed using a unique key, and updates are managed through SharedPreferences.Editor.

Syntax and Usage:

1. Initialize Shared Preferences

```
SharedPreferences sharedPreferences = getSharedPreferences("MyPrefs",
Context.MODE_PRIVATE);
```

2. Save Data

```
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString("username", "JohnDoe");
editor.putBoolean("isLoggedIn", true);
editor.putInt("age", 30);
editor.apply(); // Save changes asynchronously
```

3. Retrieve Data

```
String username = sharedPreferences.getString("username", "DefaultUser");
boolean isLoggedIn = sharedPreferences.getBoolean("isLoggedIn", false);
int age = sharedPreferences.getInt("age", 0);
```

4. Remove Data

```
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.remove("username"); // Remove a specific key
editor.apply();
```

5. Clear All Data

```
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.clear(); // Clear all stored preferences
editor.apply();
```

When Not to Use Shared Preferences

- For complex or relational data.
- For storing large files, such as media or JSON.
- For sharing data between different applications.

Lab Exercises:

1. Write a program to create a "Task Manager" application with the following features:

   a) A form to create and save tasks with fields like task name, due date, and priority level (High, Medium, Low).

   b) A list view to display all the saved tasks with their details.

   c) An option to edit or delete any task from the list.
2. Write a program to add grocery items along with its cost into the database and display the total cost of all the item selected by the user. Items has to be displayed through spinner.
3. Create an application “Movie Review” to create a movie review to do the following:
   a) User can write a movie review by stating movie name, year,giving points ranging from 1-5 and save details in a database.
   b) User can view the movie review for which review are already defined .Movie names has to be displayed using listview,and the selected movie’s details should be displayed in a table.
4. Make use of SQLite browser to view the database ,tables created.Modify the tables and upload the same into your project and view the modified data through your app.
5. Demonstrates the use of the Shared Preferences through an android application which displays a screen with some text fields. Save the value when the application is closed and brought back when it is opened again.

LAB NO. 9,10,11 **Date:**

PROJECT DESIGN AND IMPLEMENTATION

Objectives

- To design and develop a mobile application based on real world scenario by considering components like bluetooth, cameras, broadcast message or any other sensors in the phone.

REFERENCES

1. https://developer.android.com
2. Android Programming: The Big Nerd Ranch Guide,4th Edition. Bill Phillips, Chris Stewart, and Kristin Marsicano,2019
3. Professional Android, 4th Edition. Reito Meier and Ian Lake,2018
4. Zheng, Pei, and Lionel Ni. Smart phone and next generation mobile computing. Morgan Kaufmann, 2010.