

**NAME: Abdullah Habib**

**REG NO: L1F22BSCS0346**

**SECTION: E11**

**Class Participation: 4**

**COURSE NAME: Mobile Application Development**

**COURSE INSTRUCTOR: Sir Muneeb Ali Muzaffar**

**DATE OF SUBMISSION: 8th December, 2024**

**CP 04**

**Deadline (8-12-2024) Time 11:50 pm**

**Read and apply these concepts in one App using multiple activities (no need to attach activities**

**with each other – just learn how to use these concepts and use them in our app)**

**• What is an animation**

**• Creating an animation in android**

**• Loading an animation**

**• Camera: How to access Camera and taking pictures with camera**

**• Introduction to ListView in android**

**• Setting up data on a list view**

**• Using custom layout for listview**

**• Customizing adapter for a ListView**

**• Introduction ScrollView and TabView**

**What is an Animation?**

Animation in Android refers to the visual movement or transformation of UI elements to make the app more interactive and engaging. It helps improve user experience by providing visual feedback or enhancing aesthetics.

**Creating an Animation in Android**

**1. Types of Animations**

* **Property Animations:** Modify properties of objects over time (e.g., position, alpha, scale).
* **Drawable Animations:** Play frame-by-frame animations defined in a drawable resource.
* **View Animations:** Perform simple transformations like translate, rotate, scale, or fade.
* **Motion Layout Animations:** Advanced animations using ConstraintLayout.

**2. Steps to Create an Animation**

**Step 1: Define Animation in**

Animations are usually defined in the res/anim folder using .

* **Example:** A fade-in animation (fade\_in. ):

<alpha ns:android="http://schemas.android.com/apk/res/android"

android:duration="1000"

android:fromAlpha="0.0"

android:toAlpha="1.0" />

* **Example:** A slide animation (slide\_in. ):

<translate ns:android="http://schemas.android.com/apk/res/android"

android:duration="1000"

android:fromXDelta="-100%"

android:toXDelta="0%" />

**Step 2: Load and Apply the Animation**

* Use the Animation class to load and apply the animation to a view.

**Code:**

Animation fadeIn = AnimationUtils.loadAnimation(this, R.anim.fade\_in);

TextView textView = findViewById(R.id.textView);

textView.startAnimation(fadeIn);

**Loading an Animation**

**1. Loading View Animations**

* Use AnimationUtils.loadAnimation() to load -defined animations.

**Code:**

Animation slideIn = AnimationUtils.loadAnimation(this, R.anim.slide\_in);

View view = findViewById(R.id.myView);

view.startAnimation(slideIn);

**2. Loading Property Animations**

* Use the ObjectAnimator or AnimatorSet for property-based animations.

**Code:**

ObjectAnimator animator = ObjectAnimator.ofFloat(myView, "translationX", 0f, 100f);

animator.setDuration(1000);

animator.start();

**3. Frame-by-Frame Drawable Animation**

* Define an animation-list in (res/drawable/animation\_list. ):

<animation-list ns:android="http://schemas.android.com/apk/res/android">

<item android:drawable="@drawable/frame1" android:duration="100" />

<item android:drawable="@drawable/frame2" android:duration="100" />

<item android:drawable="@drawable/frame3" android:duration="100" />

</animation-list>

ImageView imageView = findViewById(R.id.imageView);

imageView.setBackgroundResource(R.drawable.animation\_list);

AnimationDrawable animationDrawable = (AnimationDrawable) imageView.getBackground();

animationDrawable.start();

**Sample Activity with Animations**

import android.os.Bundle;

import android.view.animation.Animation;

import android.view.animation.AnimationUtils;

import android.widget.ImageView;

import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

public class AnimationActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_animation);

// Load and apply fade-in animation

TextView textView = findViewById(R.id.textView);

Animation fadeIn = AnimationUtils.loadAnimation(this, R.anim.fade\_in);

textView.startAnimation(fadeIn);

// Load and apply slide-in animation

ImageView imageView = findViewById(R.id.imageView);

Animation slideIn = AnimationUtils.loadAnimation(this, R.anim.slide\_in);

imageView.startAnimation(slideIn);

}

}

**Testing**

1. Create and save animation files in the res/anim folder.
2. Add UI elements (e.g., TextView, ImageView) in your layout.
3. Use AnimationUtils or ObjectAnimator to load and start animations.

**1. Using the Camera Intent**

**Step 1: Add Permissions**

To access the camera, include the following permission in your AndroidManifest. :

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

<uses-feature android:name="android.hardware.camera" android:required="true" />

**Step 2: Start Camera Intent**

Launch the camera app using an Intent to capture a picture and retrieve the image.

**Code Example:**

import android.content.Intent;

import android.net.Uri;

import android.os.Bundle;

import android.provider.MediaStore;

import android.widget.Button;

import androidx.annotation.Nullable;

import androidx.appcompat.app.AppCompatActivity;

import .io.File;

import .io.IOException;

public class CameraIntentActivity extends AppCompatActivity {

private static final int REQUEST\_IMAGE\_CAPTURE = 1;

private Uri photoUri;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_camera\_intent);

Button captureButton = findViewById(R.id.captureButton);

captureButton.setOnClickListener(v -> openCamera());

}

private void openCamera() {

Intent intent = new Intent(MediaStore.ACTION\_IMAGE\_CAPTURE);

// Create a file for the photo

File photoFile = null;

try {

photoFile = File.createTempFile("temp\_image", ".jpg", getExternalFilesDir(null));

photoUri = Uri.fromFile(photoFile);

} catch (IOException e) {

e.printStackTrace();

}

if (photoFile != null) {

intent.putExtra(MediaStore.EXTRA\_OUTPUT, photoUri);

startActivityForResult(intent, REQUEST\_IMAGE\_CAPTURE);

}

}

@Override

protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {

super.onActivityResult(requestCode, resultCode, data);

if (requestCode == REQUEST\_IMAGE\_CAPTURE && resultCode == RESULT\_OK) {

// Display or process the captured image

// e.g., set the photoUri to an ImageView

}

}

}

**2. Using CameraX (Recommended for Advanced Features)**

**CameraX** provides a modern, easy-to-use API for integrating the camera.

**Step 1: Add Dependencies**

Add the CameraX dependencies in your build.gradle:

gradle

dependencies {

def camerax\_version = "1.3.0"

implementation "androidx.camera:camera-core:$camerax\_version"

implementation "androidx.camera:camera-camera2:$camerax\_version"

implementation "androidx.camera:camera-lifecycle:$camerax\_version"

implementation "androidx.camera:camera-view:$camerax\_version"

}

**Step 2: Add Permissions**

Include the camera permission in the AndroidManifest. :

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

**Step 3: Setup Camera Preview and Capture**

* Create a PreviewView in your layout:

<androidx.camera.view.PreviewView

android:id="@+id/previewView"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

* Initialize CameraX and capture pictures:

import android.os.Bundle;

import android.widget.Button;

import androidx.appcompat.app.AppCompatActivity;

import androidx.camera.core.ImageCapture;

import androidx.camera.core.ImageCaptureException;

import androidx.camera.lifecycle.ProcessCameraProvider;

import androidx.core.content.ContextCompat;

import .io.File;

import .util.concurrent.ExecutorService;

import .util.concurrent.Executors;

public class CameraXActivity extends AppCompatActivity {

private ImageCapture imageCapture;

private ExecutorService cameraExecutor;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_camerax);

cameraExecutor = Executors.newSingleThreadExecutor();

startCamera();

Button captureButton = findViewById(R.id.captureButton);

captureButton.setOnClickListener(v -> takePhoto());

}

private void startCamera() {

ProcessCameraProvider cameraProviderFuture =

ProcessCameraProvider.getInstance(this);

cameraProviderFuture.addListener(() -> {

try {

ProcessCameraProvider cameraProvider = cameraProviderFuture.get();

PreviewView previewView = findViewById(R.id.previewView);

// Setup Preview Use Case

androidx.camera.core.Preview preview = new androidx.camera.core.Preview.Builder().build();

preview.setSurfaceProvider(previewView.getSurfaceProvider());

// Setup Image Capture Use Case

imageCapture = new ImageCapture.Builder().build();

// Bind Use Cases to Lifecycle

cameraProvider.bindToLifecycle(

this,

androidx.camera.core.CameraSelector.DEFAULT\_BACK\_CAMERA,

preview,

imageCapture

);

} catch (Exception e) {

e.printStackTrace();

}

}, ContextCompat.getMainExecutor(this));

}

private void takePhoto() {

File photoFile = new File(getExternalFilesDir(null), "photo.jpg");

ImageCapture.OutputFileOptions outputOptions =

new ImageCapture.OutputFileOptions.Builder(photoFile).build();

imageCapture.takePicture(outputOptions, ContextCompat.getMainExecutor(this),

new ImageCapture.OnImageSavedCallback() {

@Override

public void onImageSaved(ImageCapture.OutputFileResults outputFileResults) {

// Photo saved successfully

}

@Override

public void onError(ImageCaptureException exception) {

exception.printStackTrace();

}

});

}

@Override

protected void onDestroy() {

super.onDestroy();

cameraExecutor.shutdown();

}

}

**3. Best Practices**

* Use CameraX for modern and feature-rich camera applications.
* Use Camera Intent for simple image capture needs.
* Always handle permissions gracefully, especially starting from Android 6.0 (runtime permissions).

**Introduction to ListView in Android**

A **ListView** in Android is a UI component that displays a vertically scrolling list of items. It is commonly used to show a list of data, such as contacts, messages, or settings.

**Setting Up Data on a ListView**

**1. Basic ListView Setup**

* Add a ListView in your layout :

<ListView

android:id="@+id/listView"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

**2. Populate the ListView**

* Use an ArrayAdapter to bind a list of data (e.g., an array of strings) to the ListView.

**Code Example:**

import android.os.Bundle;

import android.widget.ArrayAdapter;

import android.widget.ListView;

import androidx.appcompat.app.AppCompatActivity;

public class BasicListViewActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_basic\_listview);

// Sample data

String[] items = {"Apple", "Banana", "Cherry", "Date", "Fig"};

// Find ListView and set adapter

ListView listView = findViewById(R.id.listView);

ArrayAdapter<String> adapter = new ArrayAdapter<>(

this,

android.R.layout.simple\_list\_item\_1, // Built-in layout

items

);

listView.setAdapter(adapter);

}

}

**Using Custom Layout for ListView**

To customize the appearance of each list item, use a custom layout.

**1. Create a Custom Layout**

* Create an file for individual list items (res/layout/list\_item. ):

<LinearLayout

ns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="horizontal"

android:padding="10dp">

<ImageView

android:id="@+id/itemImage"

android:layout\_width="40dp"

android:layout\_height="40dp"

android:src="@drawable/ic\_launcher\_foreground"

android:layout\_marginRight="10dp" />

<TextView

android:id="@+id/itemText"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Item Name"

android:textSize="16sp" />

</LinearLayout>

**2. Bind Data Using ArrayAdapter**

* Create an adapter that binds the custom layout to the ListView.

**Code Example:**

import android.os.Bundle;

import android.widget.ListView;

import android.widget.ArrayAdapter;

import androidx.appcompat.app.AppCompatActivity;

public class CustomLayoutListViewActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_custom\_listview);

// Sample data

String[] items = {"Apple", "Banana", "Cherry", "Date", "Fig"};

// Custom adapter

ListView listView = findViewById(R.id.listView);

ArrayAdapter<String> adapter = new ArrayAdapter<>(

this,

R.layout.list\_item, // Custom layout

R.id.itemText, // TextView in custom layout

items

);

listView.setAdapter(adapter);

}

}

**Customizing Adapter for a ListView**

To fully control how data is displayed, create a custom adapter by extending BaseAdapter or ArrayAdapter.

**1. Define a Data Model**

public class Fruit {

String name;

int imageResId;

public Fruit(String name, int imageResId) {

this.name = name;

this.imageResId = imageResId;

}

}

**2. Create a Custom Adapter**

import android.content.Context;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.BaseAdapter;

import android.widget.ImageView;

import android.widget.TextView;

import .util.List;

public class CustomAdapter extends BaseAdapter {

private final Context context;

private final List<Fruit> fruitList;

public CustomAdapter(Context context, List<Fruit> fruitList) {

this.context = context;

this.fruitList = fruitList;

}

@Override

public int getCount() {

return fruitList.size();

}

@Override

public Object getItem(int position) {

return fruitList.get(position);

}

@Override

public long getItemId(int position) {

return position;

}

@Override

public View getView(int position, View convertView, ViewGroup parent) {

if (convertView == null) {

convertView = LayoutInflater.from(context).inflate(R.layout.list\_item, parent, false);

}

// Bind data

Fruit fruit = fruitList.get(position);

ImageView imageView = convertView.findViewById(R.id.itemImage);

TextView textView = convertView.findViewById(R.id.itemText);

imageView.setImageResource(fruit.imageResId);

textView.setText(fruit.name);

return convertView;

}

}

**3. Use the Custom Adapter**

import android.os.Bundle;

import android.widget.ListView;

import androidx.appcompat.app.AppCompatActivity;

import .util.ArrayList;

import .util.List;

public class CustomAdapterListViewActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_custom\_listview);

// Sample data

List<Fruit> fruits = new ArrayList<>();

fruits.add(new Fruit("Apple", R.drawable.apple\_icon));

fruits.add(new Fruit("Banana", R.drawable.banana\_icon));

fruits.add(new Fruit("Cherry", R.drawable.cherry\_icon));

// Set adapter

ListView listView = findViewById(R.id.listView);

CustomAdapter adapter = new CustomAdapter(this, fruits);

listView.setAdapter(adapter);

}

}

**Key Points**

1. **Default Layouts:** Use ArrayAdapter with built-in layouts for simple lists.
2. **Custom Layouts:** Define a custom layout and use it with ArrayAdapter or BaseAdapter.
3. **Custom Adapter:** Extend BaseAdapter for complete control over list rendering.
4. **Performance:** Use ViewHolder pattern inside custom adapters for better performance in large datasets.

**Introduction to ScrollView and TabView in Android**

**1. ScrollView**

**What is ScrollView?**

* A ScrollView is a layout container that allows you to create a scrollable view when the content inside it is larger than the screen.
* It supports vertical scrolling by default.

**Key Features**

* It can contain only one direct child (use a LinearLayout or other layout to manage multiple views).
* It automatically enables scrolling when content overflows the visible area.

**Basic Usage**

**Layout Example:**

<ScrollView

ns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical">

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="This is a ScrollView Example!"

android:textSize="20sp"

android:padding="10dp" />

<!-- Add more content to make it scrollable -->

<ImageView

android:layout\_width="match\_parent"

android:layout\_height="200dp"

android:src="@drawable/example\_image" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Click Me" />

<!-- Add more views here -->

</LinearLayout>

</ScrollView>

**Code (Optional):**

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;

public class ScrollViewActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_scrollview);

}

}

**2. TabView**

**What is TabView?**

* A TabView (using TabLayout) allows you to organize content into tabs, enabling users to switch between different views or fragments.
* It is commonly used with ViewPager or ViewPager2 to manage swiping between tabs.

**Key Features**

* Provides a clean and intuitive way to navigate between multiple sections.
* Works seamlessly with fragments for dynamic content.

**Basic Usage**

**Step 1: Add Dependencies** For using TabLayout with ViewPager2, ensure the following dependency is in your build.gradle file:

gradle

implementation 'com.google.android.material:material:1.9.0'

**Step 2: Layout**

<com.google.android.material.tabs.TabLayout

android:id="@+id/tabLayout"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:background="?attr/colorPrimary" />

<androidx.viewpager2.widget.ViewPager2

android:id="@+id/viewPager"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

**Step 3: Set Up Tabs in Code**

import android.os.Bundle;

import androidx.annotation.NonNull;

import androidx.appcompat.app.AppCompatActivity;

import androidx.fragment.app.Fragment;

import androidx.fragment.app.FragmentActivity;

import androidx.viewpager2.adapter.FragmentStateAdapter;

import androidx.viewpager2.widget.ViewPager2;

import com.google.android.material.tabs.TabLayout;

import com.google.android.material.tabs.TabLayoutMediator;

import .util.ArrayList;

import .util.List;

public class TabViewActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_tabview);

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

ViewPager2 viewPager = findViewById(R.id.viewPager);

// Set up the adapter

TabAdapter adapter = new TabAdapter(this);

adapter.addFragment(new TabFragment("Tab 1"), "Tab 1");

adapter.addFragment(new TabFragment("Tab 2"), "Tab 2");

adapter.addFragment(new TabFragment("Tab 3"), "Tab 3");

viewPager.setAdapter(adapter);

// Attach TabLayout with ViewPager2

new TabLayoutMediator(tabLayout, viewPager,

(tab, position) -> tab.setText(adapter.getTabTitle(position))

).attach();

}

private static class TabAdapter extends FragmentStateAdapter {

private final List<Fragment> fragments = new ArrayList<>();

private final List<String> tabTitles = new ArrayList<>();

public TabAdapter(@NonNull FragmentActivity fragmentActivity) {

super(fragmentActivity);

}

public void addFragment(Fragment fragment, String title) {

fragments.add(fragment);

tabTitles.add(title);

}

@NonNull

@Override

public Fragment createFragment(int position) {

return fragments.get(position);

}

@Override

public int getItemCount() {

return fragments.size();

}

public String getTabTitle(int position) {

return tabTitles.get(position);

}

}

}

**Step 4: Create a Fragment for Tabs**

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.TextView;

import androidx.annotation.NonNull;

import androidx.annotation.Nullable;

import androidx.fragment.app.Fragment;

public class TabFragment extends Fragment {

private final String tabName;

public TabFragment(String tabName) {

this.tabName = tabName;

}

@Nullable

@Override

public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.fragment\_tab, container, false);

TextView textView = view.findViewById(R.id.tabText);

textView.setText(tabName);

return view;

}

}

**Tab Fragment Layout (fragment\_tab.xml ):**

<LinearLayout

ns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:gravity="center">

<TextView

android:id="@+id/tabText"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Tab Content"

android:textSize="18sp"

android:padding="16dp" />

</LinearLayout>

**Comparison**

| **Feature** | **ScrollView** | **TabView** |
| --- | --- | --- |
| Purpose | Scrollable content | Navigating between multiple sections |
| Typical Use | Long forms, lists, articles | Switching between fragments |
| Layout Example | Single-child scrolling | Tabs with ViewPager2 |

Both ScrollView and TabView enhance usability and are essential in designing user-friendly Android applications.