Andhra Pradesh State Skill Development Corporation

ANDROID APPLICATION DEVELOPMENT

INTENTS

Activities and intents

Introduction

In this, you learn about the Activity class, the major building block of your app's user interface (UI). You also learn about using an Intent to communicate from one activity to another.

About activities

- An activity represents a single screen in your app with an interface the user can interact with. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading individual messages. Your app is probably a collection of activities that you create yourself, or that you reuse from other apps.
- Although the activities in your app work with each other to form a cohesive user experience, each activity is independent of the others. This enables your app to start an activity in another app, and it enables other apps to start activities in your app (if your app allows this). For example, a messaging app could start an activity in a camera app to take a picture, then start an activity in an email app to let the user share the picture in email.

Creating an Activity

To implement an Activity in your app, do the following:

Create an Activity Java class.

- Implement a basic UI for the Activity in an XML layout file.
- Declare the new Activity in the AndroidManifest.xml file.
- When you create a new project for your app, or add a new Activity to your app by choosing File > New > Activity, the template automatically performs the steps listed above.

Create the Activity

- When you create a new project in Android Studio and choose the Backwards Compatibility (AppCompat) option, the MainActivity is, by default, a subclass of the AppCompatActivity class. The AppCompatActivity class lets you use up-to-date Android app features such as the app bar and Material Design, while still enabling your app to be compatible with devices running older versions of Android.
- Here is a skeleton subclass of AppCompatActivity:

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

- The first task for you in your Activity subclass is to implement the standard Activity lifecycle callback methods (such as onCreate()) to handle the state changes for your Activity. These state changes include things such as when the Activity is created, stopped, resumed, or destroyed. You learn more about the Activity lifecycle and lifecycle callbacks in a different chapter.

- The one required callback your app must implement is the onCreate() method. The system calls this method when it creates your Activity, and all the essential components of your Activity should be initialized here. Most importantly, the onCreate() method calls setContentView() to create the primary layout for the Activity.
- You typically define the UI for your Activity in one or more XML layout files. When the setContentView() method is called with the path to a layout file, the system creates all the initial views from the specified layout and adds them to your Activity. This is often referred to as inflating the layout.
- You may often also want to implement the onPause() method in your Activity. The system calls this method as the first indication that the user is leaving your Activity (though it does not always mean that the Activity is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back). You learn more about onPause() and all the other lifecycle callbacks in a later chapter.
- In addition to lifecycle callbacks, you may also implement methods in your Activity to handle other behavior such as user input or button clicks.

Implement the activity's UI

- The UI for an activity is provided by a hierarchy of View elements, which controls a particular space within the activity window and can respond to user interaction.
- The most common way to define a UI using View elements is with an XML layout file stored as part of your app's resources. Defining your layout in XML enables you to maintain the design of your UI separately from the source code that defines the activity behavior.
- You can also create new View elements directly in your activity code by inserting new View objects into a ViewGroup, and then passing the root ViewGroup to setContentView(). After your layout has been inflated—regardless of its source—you can add more View elements anywhere in the View hierarchy.

Declare the Activity in AndroidManifest.xml

Each Activity in your app must be declared in the AndroidManifest.xml file with the element, inside the section. When you create a new project or add a new Activity to your project in Android Studio, the AndroidManifest.xml file is created or updated to include skeleton declarations for each Activity. Here's the declaration for MainActivity:

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

- The element includes a number of attributes to define properties of the Activity such as its label, icon, or theme. The only required attribute is android:name, which specifies the class name for the Activity (such as MainActivity). See the element reference for more information on Activity declarations.
- The element can also include declarations for Intent filters. The Intent filters specify the kind of Intent your Activity will accept.

```
<intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

- Intent filters must include at least one element, and can also include a and optional . The MainActivity for your app needs an Intent filter that defines the "main" action and the "launcher" category so that the system can launch your app. Android Studio creates this Intent filter for the MainActivity in your project.
- The element specifies that this is the "main" entry point to the app. The element specifies that this Activity should be listed in the system's app launcher (to allow users to launch this Activity).
- Each Activity in your app can also declare Intent filters, but only your MainActivity should include the "main" action. You learn more about how to use an implicit Intent and Intent filters in a later section.

Add another Activity to your project

- The MainActivity for your app and its associated layout file is supplied by an Activity template in Android Studio such as Empty Activity or Basic Activity. You can add a new Activity to your project by choosing File > New > Activity. Choose the Activity template you want to use, or open the Gallery to see all the available templates.
- When you choose an Activity template, you see the same set of screens for creating the new activity that you did when you created the project. Android Studio provides three things for each new activity in your app:
- A Java file for the new Activity with a skeleton class definition and onCreate() method. The new Activity, like MainActivity, is a subclass of AppCompatActivity.
- An XML file containing the layout for the new activity. Note that the setContentView() method in the Activity class inflates this new layout.
- An additional element in the AndroidManifest.xml file that specifies the new activity. The second Activity definition does not include any Intent filters. If you plan to use this activity only within your app (and not enable that activity to be started by any other app), you do not need to add filters

What are intents?

Android application components can connect to other Android applications. This connection is based on a task description represented by an Intent object.

- Intents are asynchronous messages which allow application components to request functionality from other Android components. Intents allow you to interact with components from the same applications as well as with components contributed by other applications. For example, an activity can start an external activity for taking a picture.
- Intents are objects of the android.content.Intent type. Your code can send them to the Android system defining the components you are targeting. For example, via the startActivity() method you can define that the intent should be used to start an activity.

About intents

- Each activity is started or activated with an Intent, which is a message object that makes a request to the Android runtime to start an activity or other app component in your app or in some other app.
- When your app is first started from the device home screen, the Android runtime sends an Intent to your app to start your app's main activity (the one defined with the MAIN action and the LAUNCHER category in the AndroidManifest.xml file). To start another activity in your app, or to request that some other activity available on the device perform an action, you build your own intent and call the startActivity() method to send the intent.
- In addition to starting an activity, an intent can also be used to pass data between one activity and another. When you create an intent to start a new activity, you can include information about the data you want that new activity to operate on. So, for example, an email Activity that displays a list of messages can send an Intent to the Activity that displays that message. The display activity needs data about the message to display, and you can include that data in the intent.
- In this chapter you learn about using intents with activities, but intents can also be used to start services or broadcast receivers. You learn how to use those app components in another practical.

Intent types

Intents can be explicit or implicit:

Explicit intent: You specify the receiving activity (or other component) using the activity's fully qualified class name. You use explicit intents to start components in your own app (for example, to move between screens in the UI), because you already know the package and class name of that component.
Implicit intent: You do not specify a specific activity or other component to receive the intent. Instead, you declare a general action to perform, and the Android system matches your request to an activity or other component that can handle the requested action. You learn more about using implicit intents in another practical.

- Intent objects and fields
- For an explicit Intent, the key fields include the following:
- The Activity class (for an explicit Intent). This is the class name of the Activity or other component that should receive the Intent; for example, com.example.SampleActivity.class. Use the Intent constructor or the setComponent(), setComponentName(), or setClassName() methods to specify the class.

- The Intent data. The Intent data field contains a reference to the data you want the receiving Activity to operate on as a Uri object.
- Intent extras. These are key-value pairs that carry information the receiving Activity requires to accomplish the requested action.
- Intent flags. These are additional bits of metadata, defined by the Intent class. The flags may instruct the Android system how to launch an Activity or how to treat it after it's launched.
- For an implicit Intent, you may need to also define the Intent action and category. You learn more about Intent actions and categories in another chapter.

Starting an Activity with an explicit Intent

- To start a specific Activity from another Activity, use an explicit Intent and the startActivity() method. An explicit Intent includes the fully qualified class name for the Activity or other component in the Intent object. All the other Intent fields are optional, and null by default.
- For example, if you want to start the ShowMessageActivity to show a specific message in an email app, use code like this:

  Intent messageIntent = new Intent(this, ShowMessageActivity.class);
  startActivity(messageIntent);
  The intent constructor takes two arguments for an explicit Intent:

- An application context. In this example, the Activity class provides the context (this).
- The specific component to start (ShowMessageActivity.class).
- Use the startActivity() method with the new Intent object as the only argument. The startActivity() method sends the Intent to the Android system, which launches the ShowMessageActivity class on behalf of your app. The new Activity appears on the screen, and the originating Activity is paused.

For Implementing The Activities And Intents Example Follow the code :

Explicit Intent And Implicit Intent

At screen 1: Click the **OPEN SECOND SCREEN** utton it will navigates SecondActivity Screen Like below Screen 2
At Screen 2: Shows the Up navigaton also when click thab, then it will open it's Parent Acitivity(i.e. MainActivity)
At Screen 3: Now ,enter the web site name at second EditText field ..then click web button shows like below Sceen 4
At Screen 4: Here, system os asiking to choose the destination ..here choosen crome ..it will open below 4(a) screen
Data Passing Between Two activitys
At Screen 5: Enter the some text at first EditText then click the OPEN SECOND SCREEN button that text will display on SecondActivity like Screen 6
A Screen 6: Here Displays the text where you entered at first screen
acivity_main.xml

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

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Explicit Intent Example"
    android:textSize="25sp"
    android:gravity="center"
    android:textColor="@color/colorPrimary"
    android:layout_margin="10dp"
    />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/et_user_input"
        android:hint="Enter the data"
        android:layout_margin="10dp"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="open Second screen"
        android:onClick="show_secondActivity"/>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:orientation="vertical">
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="30sp"
            android:gravity="center"
            android:textColor="@color/colorPrimary"
            android:text="Implicit Intent "/>
        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/et_web_input"
            android:hint="Enter the url "/>
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="web"
            android:id="@+id/webButton"
            />
    </LinearLayout>
</LinearLayout>
```

MainAcivity.java:

```
package com.muneiah.intentstypetest;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {
EditText et,et2;
Button btn;
public static final String KEY="apssdc.in";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        et=findViewById(R.id.et_user_input);
        et2=findViewById(R.id.et_web_input);
        btn=findViewById(R.id.webButton);
       btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String input=et2.getText().toString();
                Uri uri=Uri.parse("https://www."+input+".com");
                Intent myintent=new Intent(Intent.ACTION_VIEW,uri);
                startActivity(myintent);
            }
        });
    }

    public void show_secondActivity(View view) {
        String data=et.getText().toString();
        Intent int_obj=new Intent(this,SecondActivity.class);
        int_obj.putExtra(KEY,data);
        startActivity(int_obj);
    }
}
```

AndroidManifest.xml

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

```
    package="com.muneiah.intentstypetest">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".SecondActivity"
            android:parentActivityName=".MainActivity"
            ></activity>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

activity_secon.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondActivity"
    android:gravity="center"
    android:background="@drawable/aplogos">
    <!--for displaying the Data design TEXTVIEW -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/tv_result"
    android:text="here displays Data"
    android:textSize="40sp"

    android:layout_margin="20sp"
    android:layout_gravity="center"
    />
</LinearLayout>
```

SecondActivity.java

```
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class SecondActivity extends AppCompatActivity {
TextView label;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        label=findViewById(R.id.tv_result);
        /*Getting from activity useing getIntent()*/
        Intent i=getIntent();
       String d= i.getStringExtra(MainActivity.KEY);
       label.setText(d);
    }
}
```