جامـــعـة المـــســتـقـبـل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قســـــــم الأنظمة الطبية الذكية
Intelligent Medical Systems Department

Subject: Start Activity, Pass massages, Activity lifecycle

Class: 3rd

Lecturer: Asst.Lect Mustafa Ameer Awadh

Lecture: (8)

Study Year: 2024-2025

1.1 Introduction:

An activity represents a single screen in your app with which your user can perform a single, focused task such as dial the phone, take a photo, send an email, or view a map. Activities are usually presented to the user as full-screen windows. An app usually consists of multiple activities that are loosely bound to each other. Typically, one activity in an application is specified as the "main" activity, which is presented to the user when the app is launched. Each activity can then start other activities in order to perform different actions. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack"). When a new activity starts, that new activity is pushed onto the back stack and takes user focus. The back stack abides to the basic "last in, first out" stack mechanism, so, when the user is done with the current activity and presses the Back button, that current activity is popped from the stack (and destroyed) and the previous activity resumes. Android activities are started or activated with an intent. Intents are asynchronous messages that you can use in your activity to request an action from another activity (or other app component). You use intents to start one activity from another and to pass data between activities. There are two kinds of intents: explicit and implicit. An explicit intent is one in which you know the target of that intent. An implicit intent is one in which you do not have the name of the target component but have a general action to perform. In this practical you'll learn about explicit intents.

2.1 **Create the Two Activities project:**

Task .1: Create the Two Activities

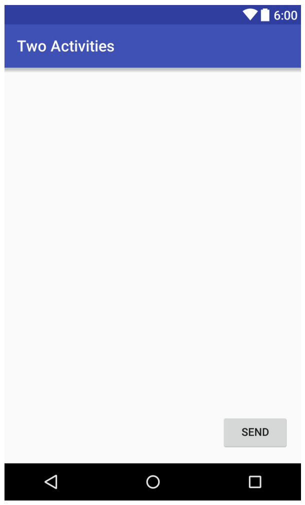
A) Start Android Studio and create a new Android Studio project.

B) Call your application "Two Activities" and change the company domain to "android.example.com." Choose the same Minimum SDK that you used in the previous projects.

A) Choose Empty Activity for the project template. Click Next.
B) Accept the default activity name (MainActivity). Make sure the Generate Layout file box is checked. Click Finish.
C) Click the Design tab delete the TextView that says "Hello World."
D) Add a Button to the layout in any position.
E) Switch to the XML Editor and modify these attributes in the Button:
F) android:id "@+id/button_main“
G) android:onClick "launchSecondActivity“
android:text "Send"

Solution code: Depending on your version of Android Studio, your code will look something like the following.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.example.android.twoactivities.MainActivity">
    <Button
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/button_main"
       android:id="@+id/button_main"
       android:layout_alignParentBottom="true"
       android:layout_alignParentRight="true"
       android:layout_alignParentEnd="true"
       android:onClick="launchSecondActivity"/>
</RelativeLayout>
```



2.2 Define the button action:

- In the XML Editor, place the cursor on the word "launchSecondActivity" .
- Press Alt-Enter (Option-Enter on the Mac) and select Create 'launchSecondActivity(View)' in 'MainActivity.
- The MainActivity.java files opens, and Android Studio generates a skeleton method for the onClick handler.
- Inside launchSecondActivity , add a log statement that says "Button Clicked!"

```
Log.d(LOG_TAG, "Button clicked!");
```

LOG_TAG will show as red. The definitions for that variable will be added in a later step.

Place the cursor on the word "Log" and press Alt-Enter (Option-Enter on the Mac). Android Studio adds an import statement for android.util.Log.

- At the top of the class, add a constant for the LOG_TAG variable:

```
private static final String LOG_TAG =MainActivity.class.getSimpleName();
```

This constant uses the name of the class itself as the tag.

- Run your app. When you click the "Send" button you will see the "Button Clicked!" message in the Android Monitor

```
package com.example.android.twoactivities;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;

public class MainActivity extends AppCompatActivity {
   private static final String LOG_TAG = MainActivity.class.getSimpleName();

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

   public void launchSecondActivity(View view) {
       Log.d(LOG_TAG, "Button clicked!");
   }
}
```

2.3 Create and launch the second activity

- Each new activity you added to your project has its own layout and Java files, separate from those of the main activity. They also have their own <activity> elements in the Android manifest. As with the main activity, new activities you create in
- Android Studio also extend from the AppCompatActivity class.
- All the activities in your app are only loosely connected with each other. However, you can define an activity as a parent of another activity in the AndroidManifest.xml file. This parent-child relationship enables Android to add navigation hints such as left-facing arrows in the title bar for each activity.

- Activities communicate with each other (both in the same app and across different apps) with intents.
- In this task you'll add a second activity to our app, with its own layout. You'll modify the Android manifest to define the main
- activity as the parent of the second activity. Then you'll modify the onClick event method in the main activity to include an intent that launches the second activity when you click the button.

2.4 Modify the Android manifest

- Open manifests/AndroidManifest.xml .
- Find the <activity> element that Android Studio created for the second activity.
  <activity android:name=".SecondActivity"></activity>
- Add these attributes to the <activity> element:
  android:label "Second Activity"
  android:parentActivityName ".MainActivity“
- Add a <meta-data> element inside the <activity> element for the second activity. Use these attributes:
  android:name "android.support.PARENT_ACTIVITY"
  android:value "com.example.android.twoactivities.MainActivity“

2.5 Solution code:

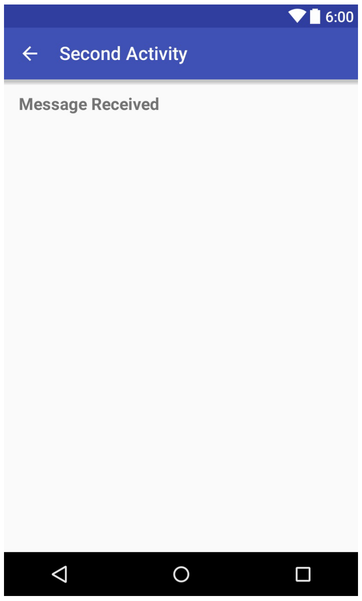
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.android.twoactivities">

   <application
       android:allowBackup="true"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:supportsRtl="true"
       android:theme="@style/AppTheme">
       <activity android:name=".MainActivity">
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
       </activity>
       <activity android:name=".SecondActivity"
           android:label="@string/activity2_name"
           android:parentActivityName=".MainActivity">
           <meta-data
               android:name="android.support.PARENT_ACTIVITY"
               android:value="com.example.android.twoactivities.MainActivity" />
       </activity>
   </application>
</manifest>
```

2.6 Define the layout for the second activity

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".SecondActivity">

   <TextView
       android:id="@+id/text_header"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_marginBottom="@dimen/activity_vertical_margin"
       android:text="@string/text_header"
       android:textAppearance="?android:attr/textAppearanceMedium"
       android:textStyle="bold" />
</RelativeLayout>
```



2.7 Add an intent to the main activity:

Create a new intent in the launchSecondActivity() method.
The Intent constructor takes two arguments for an explicit intent: an application Context and the specific component that will receive that intent.
Here you should use this as the context, and SecondActivity.class as the specific class.

```
Intent intent = new Intent(this, SecondActivity.class);
```

Place the cursor on Intent and press Alt-Enter to add an import for the Intent class.
Call the startActivity() method with the new intent as the argument.

```
startActivity(intent);
```

2.8 Send data from the main activity to the second activity:

Add an EditText to the main activity layout
Add a string to the main activity's intent extras

1. Open java/com.example.android.twoactivities/MainActivity .
2. Add a public constant at the top of the class to define the key for the intent extra:

```
public static final String EXTRA_MESSAGE =
"com.example.android.twoactivities.extra.MESSAGE";
```

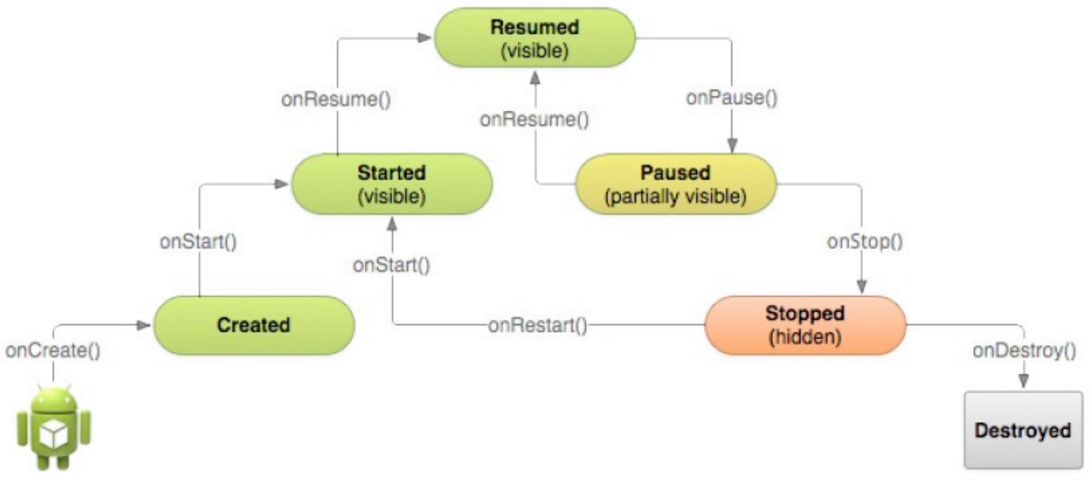
3. Add a private variable at the top of the class to hold the EditText object.

```
  private EditText mMessageEditText;
```

Add a TextView to the second activity for the message
Modify the second activity to get the extras and display the message

```
public class SecondActivity extends AppCompatActivity {
      @Override
      protected void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_second);
            Intent intent = getIntent();
             String message =
intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
                TextView textView = (TextView) findViewById(R.id.text_message);
                 textView.setText(message);
           }
 }
```

2.9 Activity Lifecycle and Instance State



3.1 Implement callbacks into MainActivity

- Open java/com.example.android.twoactivities/MainActivity. In the onCreate() method, add the following log statements:

```
        Log.d(LOG_TAG, "-------");
     Log.d(LOG_TAG, "onCreate");
```

- Add a new method for the onStart() callback, with a statement to the log for that event:

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

Use the onStart() method as a template to implement the other lifecycle callbacks:

onPause()
onRestart()
onResume()
onStop()
onDestroy()

All the callback methods have the same signatures (except for the name). If you copy and paste onStart() to create
these other callback methods, don't forget to update the contents to call the right method in the superclass, and to log the correct method. Build and run your app.

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

    Log.d(LOG_TAG, "-------");
    Log.d(LOG_TAG, "onCreate");

    mMessageEditText = (EditText) findViewById(R.id.editText_main);
    mReplyHeadTextView = (TextView) findViewById(R.id.text_header_reply);
    mReplyTextView = (TextView) findViewById(R.id.text_message_reply);
}

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

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

@Override
public void onResume() {
    super.onResume();
    Log.d(LOG_TAG, "onResume");
}

@Override
public void onPause() {
    super.onPause();
    Log.d(LOG_TAG, "onPause");
}

@Override
public void onStop() {
    super.onStop();
    Log.d(LOG_TAG, "onStop");
}

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