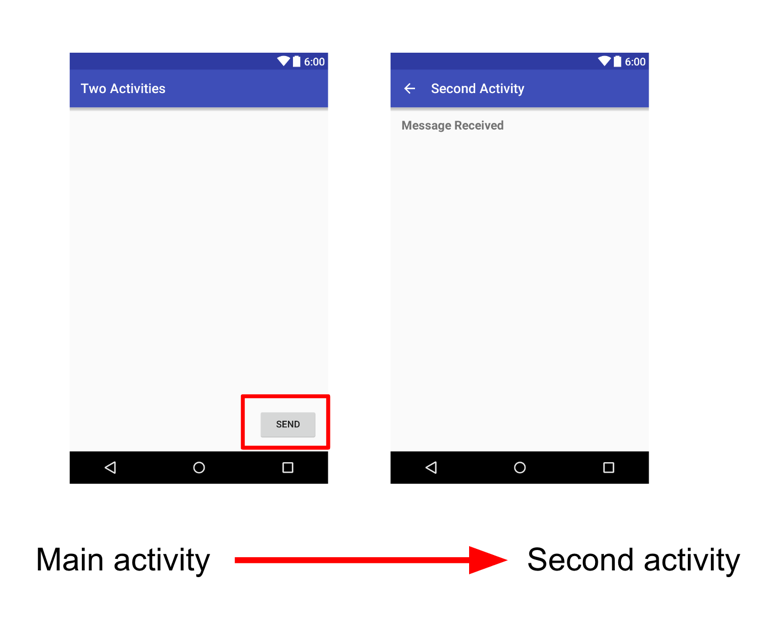
Android Development

05 Practical - Activities

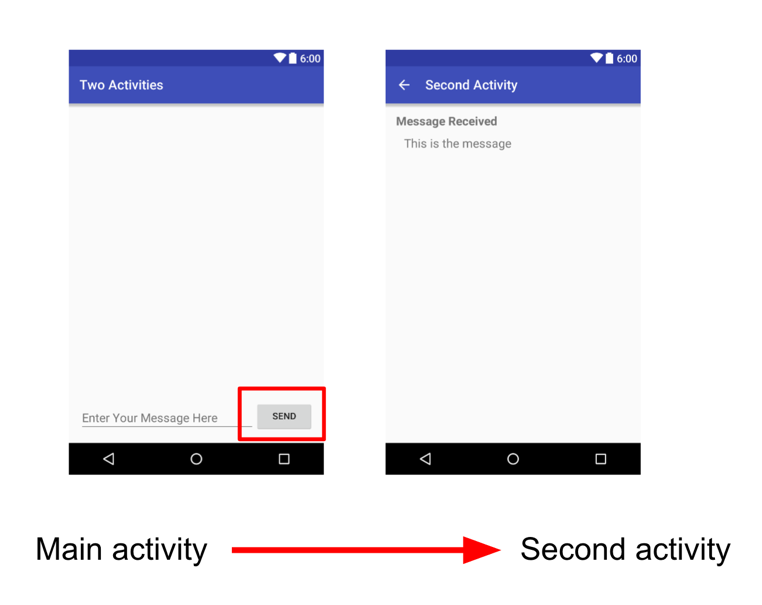
# App Overview

You will create and build an app called Two Activities that, unsurprisingly, contains two Activity implementations. You build the app in two stages.

In the first stage, you create an app whose main activity contains one button, **Send**. When the user clicks this button, your main activity uses an intent to start the second activity.



In the second stage, you add an EditText view to the main activity. The user enters a message and clicks **Send**. The main activity uses an intent to start the second activity and send the user's message to the second activity. The second activity displays the message it received.



# Create the TwoActivities project

Set up the initial project with a main Activity, define the layout, and define a skeleton method for the onClick button event

## Create the TwoActivities project

1. Start Android Studio and create a new Android Studio project. Name your app **Two Activities** and choose the same **Phone and Tablet** settings that you used in previous practicals. The project folder is automatically named TwoActivities, and the app name that appears in the app bar will be "Two Activities".
2. Choose **Empty Views Activity** for the Activity template. Click **Next.**
3. Change the Name package name if you wish but leave the language as Java and the default SDK as API 24 Android 7.0
4. Click **Finish**.

## Define the layout for the main Activity

1. Open res > layout > activity\_main.xml in the Project > Android pane. The layout editor appears.
2. Click the Design tab if it is not already selected and delete the TextView (the one that says "Hello World") in the Component Tree pane.
3. With Autoconnect turned on (the default setting), drag a Button from the Palette pane to the lower right corner of the layout. Autoconnect creates constraints for the Button.
4. In the Attributes pane, set the ID to button\_main, the layout\_width and layout\_height to wrap\_content, and enter Send for the Text field. The layout should now look like this:



1. Click the **Text** tab to edit the XML code. Add the following attribute to the Button:

android:onClick="launchSecondActivity"

The attribute value is underlined in red because the launchSecondActivity() method has not yet been created. Ignore this error for now; you fix it in the next task.

1. Extract the string resource, as described in a previous practical, for "Send" and use the name button\_main for the resource.

The XML code for the Button should look like the following:

Button

android:id="@+id/button\_main"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginBottom="16dp"

android:layout\_marginRight="16dp"

android:text="@string/button\_main"

android:onClick="launchSecondActivity"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintRight\_toRightOf="parent" />

## Define the Button action

Now you implement the launchSecondActivity() method you referred to in the layout for the android:onClick attribute.

1. Click on "launchSecondActivity" in the activity\_main.xml XML code.
2. Press Alt+Enter (Option+Enter on a Mac) and select **Create 'launchSecondActivity(View)' in 'MainActivity.**

The MainActivity file opens, and Android Studio generates a skeleton method for the launchSecondActivity() handler.

1. Inside launchSecondActivity(), add a Log statement that says "Button Clicked!"

Log.d(LOG\_TAG, "Button clicked!");

1. At the top of the MainActivity 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.

1. Run your app. When you click the **Send** button you see the "Button Clicked!" message in the **Logcat** pane. If there's too much output in the monitor, type **MainActivity** into the search box, and the **Logcat** pane will only show lines that match that tag.

The code for MainActivity should look as follows:

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!");

}

}

# Create and launch a second activity

Each new activity you add 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 AndroidManifest.xml file. As with the main activity, new activity implementations that you create in Android Studio also extend from the AppCompatActivity class.

Each activity in your app is only loosely connected with other activities. 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.

## Create a second Activity

1. Click the **app** folder for your project and choose **File > New > Activity > Empty Activity**.
2. Name the new Activity **SecondActivity**. Make sure **Generate Layout File** and **Backwards Compatibility (AppCompat)** are checked. The layout name is filled in as activity\_second. Do *not* check the **Launcher Activity** option.
3. Click **Finish**. Android Studio adds both a new Activity layout (activity\_second.xml) and a new Java file (SecondActivity.java) to your project for the new Activity. It also updates the AndroidManifest.xml file to include the new Activity.

## Add the new Activity to the manifest

1. Open manifests > AndroidManifest.xml.
2. Find the <activity> element that Android Studio created for the second Activity.

<activity android:name=".SecondActivity"></activity>

1. Replace the entire <activity> element with the following:

<activity android:name=".SecondActivity"

android:label = "Second Activity"

android:parentActivityName=".MainActivity">

<meta-data

android:name="android.support.PARENT\_ACTIVITY"

android:value=

"com.example.android.twoactivities.MainActivity" />

</activity>

The label attribute adds the title of the Activity to the app bar.

With the parentActivityName attribute, you indicate that the main activity is the parent of the second activity. This relationship is used for Up navigation in your app: the app bar for the second activity will have a left-facing arrow so the user can navigate "upward" to the main activity.

With the <meta-data> element, you provide additional arbitrary information about the activity in the form of key-value pairs. In this case the metadata attributes do the same thing as the android:parentActivityName attribute—they define a relationship between two activities for upward navigation. These metadata attributes are required for older versions of Android, because the android:parentActivityName attribute is only available for API levels 16 and higher.

1. Extract a string resource for "Second Activity" in the code above and use activity2\_name as the resource name.

## Define the layout for the second Activity

1. Open **activity\_second.xml** and click the **Design** tab if it is not already selected.
2. Drag a **TextView** from the **Palette** pane to the top left corner of the layout and add constraints to the top and left sides of the layout. Set its attributes in the **Attributes** pane as follows:

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| id | text\_header |
| Top margin | 16 |
| Left margin | 8 |
| layout\_width | wrap\_content |
| layout\_height | wrap\_content |
| text | Message Received |
| textAppearance | AppCompat.Medium |
| textStyle | B (bold) |

The value of **textAppearance** is a special Android theme attribute that defines basic font styles. You learn more about themes in a later lesson.

The layout should now look like this:



1. Click the **Text** tab to edit the XML code and extract the "Message Received" string into a resource named text\_header.
2. Add the android:layout\_marginLeft="8dp" attribute to the TextView to complement the layout\_margin Start attribute for older versions of Android.

The XML code for activity\_second.xml should be as follows:

<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.example.android.twoactivities.SecondActivity">

<TextView

android:id="@+id/text\_header"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginStart="8dp"

android:layout\_marginLeft="8dp"

android:layout\_marginTop="16dp"

android:text="@string/text\_header"

android:textAppearance=

"@style/TextAppearance.AppCompat.Medium"

android:textStyle="bold"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>

## Add an Intent to the main Activity

Now add an explicit Intent to the main Activity. This Intent is used to activate the second Activity when the **Send** button is clicked.

1. Open **MainActivity**.
2. 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);

1. Call the startActivity() method with the new Intent as the argument.
2. Run the app.

When you click the **Send** button, MainActivity sends the Intent and the Android system launches SecondActivity, which appears on the screen. To return to MainActivity, click the **Up** button (the left arrow in the app bar) or the Back button at the bottom of the screen.

# Send data from the main Activity to the second Activity

You will modify the explicit intent in MainActivity to include additional data (in this case, a user-entered string) in the intent extra Bundle. You then modify SecondActivity to get that data back out of the intent extra Bundle and display it on the screen.

## Add an EditText to the MainActivity layout

1. Open activity\_main.xml.
2. Drag a Plain Text (EditText) element from the Palette pane to the bottom of the layout and add constraints to the left side of the layout, the bottom of the layout, and the left side of the Send Button. Set its attributes in the Attributes pane as follows:

|  |  |
| --- | --- |
| Attribute | Value |
| id | editText\_main |
| Right margin | 8 |
| Left margin | 8 |
| Bottom margin | 16 |
| layout\_width | match\_constraint |
| layout\_height | wrap\_content |
| inputType | textLongMessage |
| hint | Enter Your Message Here |
| text | (Delete any text in this field) |

The new layout in activity\_main.xml looks like this:



## Add a string to the Intent extras

The Intent extras are key/value pairs in a [Bundle](https://developer.android.com/reference/android/os/Bundle.html). A Bundle is a collection of data, stored as key/value pairs. To pass information from one Activity to another, you put keys and values into the Intent extra Bundle from the sending Activity, and then get them back out again in the receiving Activity.

1. Open 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";

1. Add a private variable at the top of the class to hold the EditText:

private EditText mMessageEditText;

1. In the onCreate() method, use findViewById() to get a reference to the EditText and assign it to that private variable:

mMessageEditText = findViewById(R.id.editText\_main);

1. In the launchSecondActivity() method, just under the new Intent, get the text from the EditText as a string:

String message = mMessageEditText.getText().toString();

1. Add that string to the Intent as an extra with the EXTRA\_MESSAGE constant as the key and the string as the value:

intent.putExtra(EXTRA\_MESSAGE, message);

The onCreate() method in MainActivity should now look like the following:

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

mMessageEditText = findViewById(R.id.editText\_main);

}

The launchSecondActivity() method in MainActivity should now look like the following:

public void launchSecondActivity(View view) {

Log.d(LOG\_TAG, "Button clicked!");

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

String message = mMessageEditText.getText().toString();

intent.putExtra(EXTRA\_MESSAGE, message);

startActivity(intent);

}

## Add a TextView to SecondActivity for the message

1. Open activity\_second.xml.
2. Drag another TextView to the layout underneath the text\_header TextView and add constraints to the left side of the layout and to the bottom of text\_header.
3. Set the new TextView attributes in the Attributes pane as follows:

|  |  |
| --- | --- |
| **Attribute** | **Value** |
| id | text\_message |
| Top margin | 8 |
| Left margin | 8 |
| layout\_width | wrap\_content |
| layout\_height | wrap\_content |
| text | (Delete any text in this field) |
| textAppearance | AppCompat.Medium |

The new layout looks the same as it did in the previous task, because the new TextView does not (yet) contain any text, and thus does not appear on the screen.

The XML code for the activity\_second.xml layout should look something like the following:

android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.example.android.twoactivities.SecondActivity">

<TextView

android:id="@+id/text\_header"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginStart="8dp"

android:layout\_marginTop="16dp"

android:text="@string/text\_header"

android:textAppearance=

"@style/TextAppearance.AppCompat.Medium"

android:textStyle="bold"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

<TextView

android:id="@+id/text\_message"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginStart="8dp"

android:layout\_marginTop="8dp"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toBottomOf="@+id/text\_header" />

</android.support.constraint.ConstraintLayout>

## Modify SecondActivity to get the extras and display the message

1. Open **SecondActivity** to add code to the onCreate() method.
2. Get the Intent that activated this Activity:

Intent intent = getIntent();

1. Get the string containing the message from the Intent extras using the MainActivity.EXTRA\_MESSAGE static variable as the key:

String message = intent.getStringExtra(MainActivity.EXTRA\_MESSAGE);

1. Use findViewByID() to get a reference to the TextView for the message from the layout:

TextView textView = findViewById(R.id.text\_message);

1. Set the text of the TextView to the string from the Intent extra:

textView.setText(message);

1. Run the app. When you type a message in MainActivity and click **Send**, SecondActivity launches and displays the message.

The SecondActivity onCreate() method should look as follows:

@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 = findViewById(R.id.text\_message);

textView.setText(message);

}