

Start another activity

After completing the [previous lesson](#)

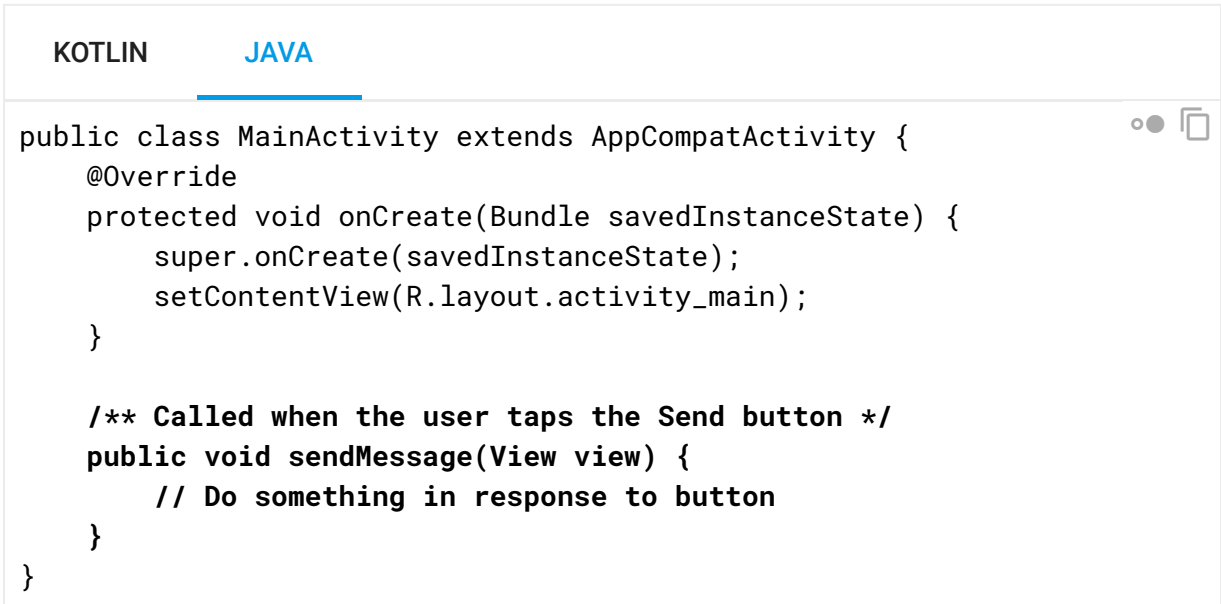
(<https://developer.android.com/training/basics/firstapp/building-ui.html>), you have an app that shows an activity (a single screen) with a text field and a button. In this lesson, you'll add some code to **MainActivity** that starts a new activity to display the message when the user taps **Send**.

Note: This lesson expects that you are using Android Studio 3.0 or higher.

Respond to the send button

Add a method to the **MainActivity** class that's called by the button as follows:

1. In the file **app > java > com.example.myfirstapp > MainActivity**, add the **sendMessage()** method stub as shown below:

A screenshot of the Android Studio code editor showing the MainActivity.java file. The editor has tabs for KOTLIN and JAVA, with JAVA selected. The code is in Java and shows the MainActivity class extending AppCompatActivity. It includes an onCreate method and a new sendMessage method stub. The sendMessage method has a comment indicating it's called when the user taps the Send button. The code is as follows:

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
  
    /** Called when the user taps the Send button */  
    public void sendMessage(View view) {  
        // Do something in response to button  
    }  
}
```

You may see an error because Android Studio cannot resolve the **View** class used as the method argument. So click to place your cursor on the **View** declaration, and then perform a Quick Fix by pressing **Alt + Enter** (or **Option + Enter** on Mac). (If a menu appears, select **Import class**.)

2. Now return to the **activity_main.xml** file to call this method from the button:
 - a. Click to select the button in the Layout Editor.

- b. In the **Attributes** window, locate the **onClick** property and select **sendMessage [MainActivity]** from the drop-down list.

Now when the button is tapped, the system calls the `sendMessage()` method.

Take note of the details in this method that are required in order for the system to recognize it as compatible with the `android:onClick`

(https://developer.android.com/reference/android/view/View.html#attr_android:onClick) attribute.

Specifically, the method has the following characteristics:

- **Public access**
- **A void** or, in Kotlin, an implicit unit (<https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-unit/index.html>) return value
- **A View** (<https://developer.android.com/reference/android/view/View.html>) as the only parameter (it is the View (<https://developer.android.com/reference/android/view/View.html>) object that was clicked)

Next, you'll fill in this method to read the contents of the text field and deliver that text to another activity.

Build an Intent

An Intent (<https://developer.android.com/reference/android/content/Intent.html>) is an object that provides runtime binding between separate components, such as two activities. The Intent (<https://developer.android.com/reference/android/content/Intent.html>) represents an app's "intent to do something." You can use intents for a wide variety of tasks, but in this lesson, your intent starts another activity.

In MainActivity, add the `EXTRA_MESSAGE` constant and the `sendMessage()` code, as shown here:

KOTLIN

JAVA

```
public class MainActivity extends AppCompatActivity {  
    public static final String EXTRA_MESSAGE = "com.example.myfirstapp.MESSA  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
  
    /** Called when the user taps the Send button */
```

```

public void sendMessage(View view) {
    Intent intent = new Intent(this, DisplayMessageActivity.class);
    EditText editText = (EditText) findViewById(R.id.editText);
    String message = editText.getText().toString();
    intent.putExtra(EXTRA_MESSAGE, message);
    startActivity(intent);
}
}

```

Android Studio again encounters **Cannot resolve symbol** errors, so press Alt + Enter (or Option + Return on Mac). Your imports should end up as the following:

KOTLIN

JAVA

```

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

```

An error remains for `DisplayMessageActivity`, but that's okay; you'll fix that in the next section.

Here's what's going on in `sendMessage()`:

- The [Intent](https://developer.android.com/reference/android/content/Intent.html) (<https://developer.android.com/reference/android/content/Intent.html>) constructor takes two parameters:
 - A [Context](https://developer.android.com/reference/android/content/Context.html) (<https://developer.android.com/reference/android/content/Context.html>) as its first parameter (**this is used because the [Activity](https://developer.android.com/reference/android/app/Activity.html)** (<https://developer.android.com/reference/android/app/Activity.html>) class is a subclass of [Context](https://developer.android.com/reference/android/content/Context.html) (<https://developer.android.com/reference/android/content/Context.html>))
 - The [Class](https://developer.android.com/reference/java/lang/Class.html) (<https://developer.android.com/reference/java/lang/Class.html>) of the app component to which the system should deliver the [Intent](https://developer.android.com/reference/android/content/Intent.html) (<https://developer.android.com/reference/android/content/Intent.html>) (in this case, the activity that should be started).
- The [putExtra\(\)](https://developer.android.com/reference/android/content/Intent.html#putExtra(java.lang.String,java.lang.String)) ([https://developer.android.com/reference/android/content/Intent.html#putExtra\(java.lang.String,java.lang.String\)](https://developer.android.com/reference/android/content/Intent.html#putExtra(java.lang.String,java.lang.String))) method adds the `EditText`'s value to the intent. An `Intent` can carry data types as key-value pairs called *extras*. Your key is a public constant `EXTRA_MESSAGE` because the next activity uses the key to retrieve the text value. It's a good practice to define

keys for intent extras using your app's package name as a prefix. This ensures the keys are unique, in case your app interacts with other apps.

- The **`startActivity()`** ([https://developer.android.com/reference/android/app/Activity.html#startActivity\(android.content.Intent\)](https://developer.android.com/reference/android/app/Activity.html#startActivity(android.content.Intent))) method starts an instance of the **`DisplayMessageActivity`** specified by the **`Intent`** (<https://developer.android.com/reference/android/content/Intent.html>). Now you need to create that class.

The Navigation Architecture Component, currently in alpha, allows you to use the Navigation Editor to associate one activity with another. Once the relationship is made, you can use the API to start the second activity when the user triggers the associated action (i.e. clicking a button). To learn more, see [The Navigation Architecture Component](https://developer.android.com/topic/libraries/architecture/navigation/) (<https://developer.android.com/topic/libraries/architecture/navigation/>).

Create the second activity

1. In the **Project** window, right-click the **app** folder and select **New > Activity > Empty Activity**.
2. In the **Configure Activity** window, enter "DisplayMessageActivity" for **Activity Name** and click **Finish** (leave all other properties set to the defaults).

Android Studio automatically does three things:

- Creates the **`DisplayMessageActivity`** file.
- Creates the corresponding **`activity_display_message.xml`** layout file.
- Adds the required **`<activity>`** (<https://developer.android.com/guide/topics/manifest/activity-element.html>) element in **`AndroidManifest.xml`**.

If you run the app and tap the button on the first activity, the second activity starts but is empty. This is because the second activity uses the empty layout provided by the template.

Add a text view

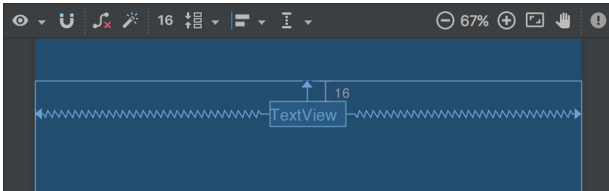



Figure 1. The text view centered at the top of the layout

The new activity includes a blank layout file, so now you'll add a text view where the message will appear.

1. Open the file **app > res > layout > activity_display_message.xml**.
2. Click **Turn On Autoconnect**  in the toolbar (it should then be enabled, as shown in figure 1).
3. In the **Palette** window, click **Text** and then drag a **TextView** into the layout—drop it near the the top of the layout, near the center so it snaps to the vertical line that appears. Autoconnect adds left and right constraints to place the view in the horizontal center.
4. Create one more constraint from the top of the text view to the top of the layout, so it appears as shown in figure 1.

Optionally, make some adjustments to the text style by expanding **textAppearance** in the **Attributes** window and change attributes such as **textSize** and **textColor**.

Display the message

Now you will modify the second activity to display the message that was passed by the first activity.

1. In **DisplayMessageActivity**, add the following code to the **onCreate()** method:

```
KOTLIN    JAVA
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_display_message);

    // Get the Intent that started this activity and extract the string
    Intent intent = getIntent();
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
```

```
// Capture the layout's TextView and set the string as its text
TextView textView = findViewById(R.id.textview);
textView.setText(message);
}
```

2. Press Alt + Enter (or Option + Return on Mac) to import missing classes. Your imports should end up as the following:

```
KOTLIN  JAVA
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
```

Add up navigation

Each screen in your app that is not the main entry point (all screens that are not the "home" screen) should provide navigation so the user can return to the logical parent screen in the app hierarchy by tapping the Up button in the app bar (<https://developer.android.com/training/appbar/index.html>).

All you need to do is declare which activity is the logical parent in the **AndroidManifest.xml** (<https://developer.android.com/guide/topics/manifest/manifest-intro.html>) file. So open the file at **app > manifests > AndroidManifest.xml**, locate the `<activity>` tag for `DisplayMessageActivity` and replace it with the following:

```
<activity android:name=".DisplayMessageActivity"
    android:parentActivityName=".MainActivity">
    <!-- The meta-data tag is required if you support API level 15 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

The Android system now automatically adds the Up button in the app bar.

Run the app

Now run the app again by clicking **Apply Changes** ⚡ in the toolbar. When it opens, type a message in the text field, and tap **Send** to see the message appear in the second activity.



Figure 2. Screenshots of both activities

That's it, you've built your first Android app!

To continue learning the basics about Android app development, follow the other links provided on [this tutorial's front page](https://developer.android.com/training/basics/firstapp/) (<https://developer.android.com/training/basics/firstapp/>).

[Previous](#)



[Build a simple user interface](#)

(<https://developer.android.com/training/basics/firstapp/building-ui>)

Content and code samples on this page are subject to the licenses described in the [Content License](#) (/license).
Java is a registered trademark of Oracle and/or its affiliates.

Last updated June 15, 2018.



Twitter

Follow @AndroidDev on
Twitter



Google+

Follow Android Developers on
Google+



YouTube

Check out Android Developers
on YouTube

