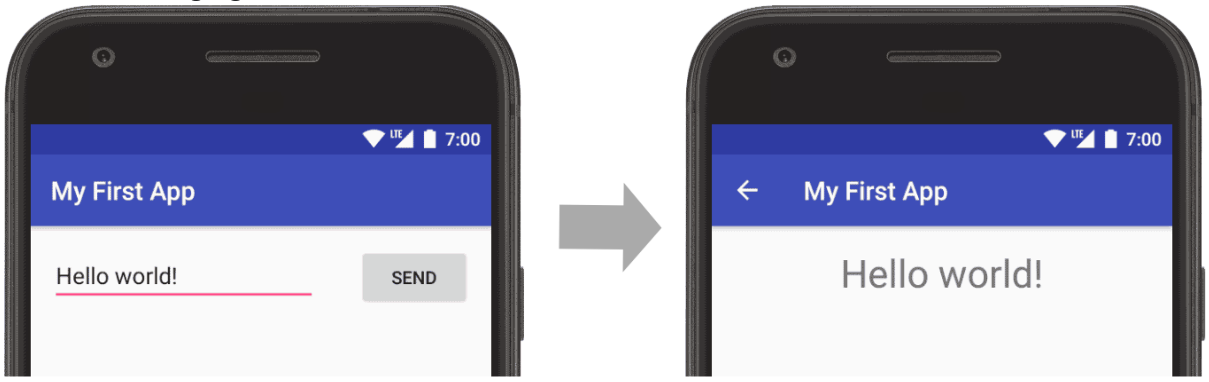
Mobile Programming – Bina Nusantara

Exercise: Multiple Activities

In this exercise, you add some code to the `MainActivity` that starts a new activity to display a message when the user taps the **Send** button. This is similar to our textbook Headfirst Android Programming. Please check Binusmaya to get details.

Expected Outcome:
When you complete this exercise, you'll be able to connect from one activity to another, like in the following figure:



Respond to the Send button

Follow these steps to add a method to the `MainActivity` class that's called when the **Send** button is tapped:

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

```
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 might see an error because Android Studio cannot resolve the `View` class used as the method argument. To clear the error, click the `View` declaration, place your cursor on it, and then press `Alt+Enter`, or `Option+Enter` on a Mac, to perform a Quick Fix. If a menu appears, select **Import class**.

2. Return to the **activity_main.xml** file to call the method from the button:

a. Select the button in the Layout Editor.

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

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

Take note of the details in this method. They're required for the system to recognize the method as compatible with the `android:onClick` attribute. Specifically, the method has the following characteristics:

- Public access.
- A void.
- A `View` as the only parameter. This is the `View` object you clicked at the end of Step 1.

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

Build an intent

An `Intent` is an object that provides runtime binding between separate components, such as two activities. The `Intent` 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:

KOTLINJAVA

```
public class MainActivity extends AppCompatActivity {
    public static final String EXTRA_MESSAGE =
"com.example.myfirstapp.MESSAGE";
    @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);
    }
}
```

Expect Android Studio to encounter **Cannot resolve symbol** errors again. To clear the errors, press `Alt+Enter`, or `Option+Return` on a Mac. Your should end up with the following imports:

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
```

An error still remains for `DisplayMessageActivity`, but that's okay. You fix it in the next section.

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

- The `Intent` constructor takes two parameters, a `Context` and a `Class`.

  The `Context` parameter is used first because the `Activity` class is a subclass of `Context`.

  The `Class` parameter of the app component, to which the system delivers the `Intent`, is, in this case, the activity to start.
- The `putExtra()` method adds the value of `EditText` 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 with your app's package name as a prefix. This ensures that the keys are unique, in case your app interacts with other apps.

- The `startActivity()` method starts an instance of the `DisplayMessageActivity` that's specified by the `Intent`. Next, you need to create that class.

Note: The Navigation Architecture Component 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, such as when the user clicks a button.

Create the second activity

To create the second activity, follow these steps:

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**. Leave all other properties set to their defaults and click **Finish**.

   Android Studio automatically does three things:

- Creates the `DisplayMessageActivity` file.
- Creates the layout file `activity_display_message.xml`, which corresponds with the `DisplayMessageActivity` file.
- Adds the required `<activity>` 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

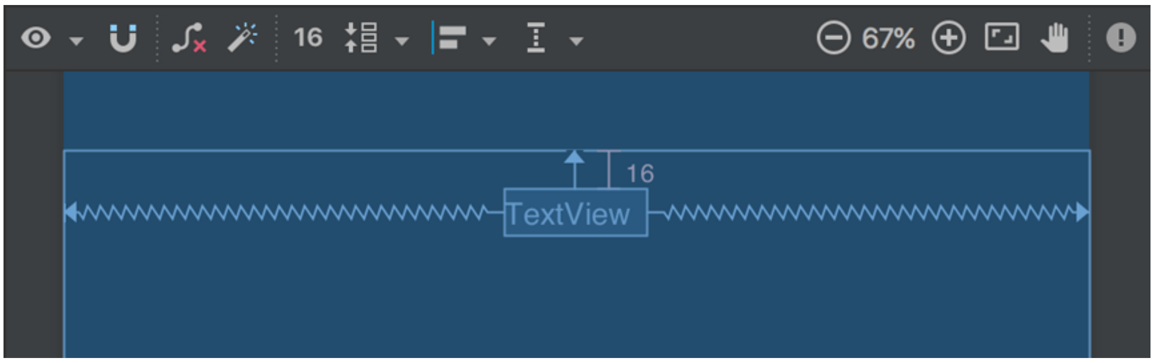


Figure 1. The text view centered at the top of the layout.

The new activity includes a blank layout file. Follow these steps to add a text view to where the message appears:

1. Open the file **app > res > layout > activity_display_message.xml**.
2. Click **Enable Autoconnection to Parent** in the toolbar. This enables Autoconnect. See figure 1.

3. In the **Palette** panel, click **Text**, drag a **TextView** into the layout, and drop it near the top-center of the layout so that it snaps to the vertical line that appears. Autoconnect adds left and right constraints in order 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 that it appears as shown in figure 1.

   Optionally, you can make some adjustments to the text style if you expand **textAppearance** in the **Common Attributes** panel of the **Attributes** window, and change attributes such as **textSize** and **textColor**.

Display the message

In this step, you 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:

```
@Override
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 a Mac, to import these other needed classes:

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;
```

Add upward navigation

Each screen in your app that's not the main entry point, which are all the screens that aren't the home screen, must provide navigation that directs the user to the logical parent screen in the app's hierarchy. To do this, add an **Up** button in the app bar.

To add an **Up** button, you need to declare which activity is the logical parent in the `AndroidManifest.xml` file. 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 to the app bar.

Run the app

Before running the app, make sure you have configured AVD (Android Virtual Device) in your workstation IDE.

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

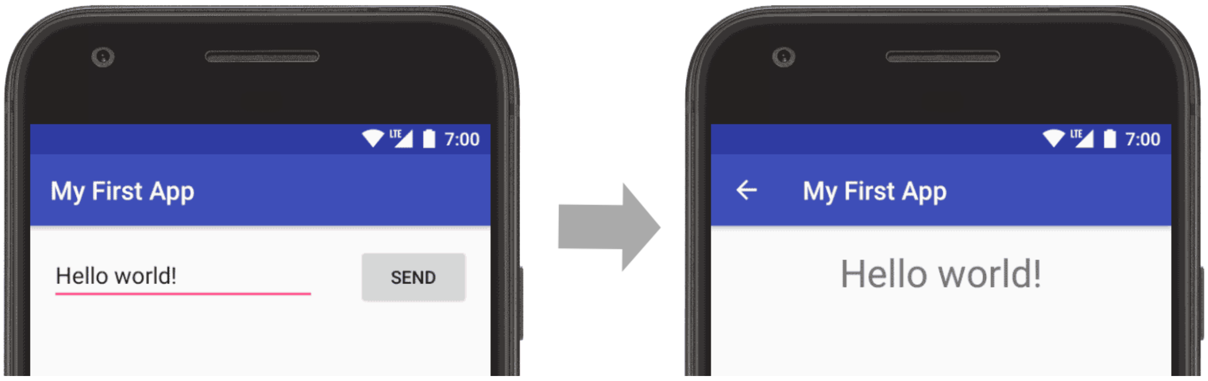


Figure 2. App opened, with text entered on the left screen and displayed on the right.

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