**Lab Manual for Application for Mobile Devices**

**Lab No. 7**

**Activity Communication: Sending Data**

Objectives

The purpose of this lab is to familiarize with multiple activities communication

**LAB # 07**

**Activity Communication: Sending Data**

## **Introduction**

Up till now you have an app that shows an activity that consists of a single screen with a text field and a **Send** button which calls sendMessage method. In this lab, you add some code to the MainActivity that starts a new activity to display a message when the user taps the **Send** button.

**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. 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  
    }**  
}

**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:

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);**  
    }  
}

**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.

* Now at a TextView in the activity.

**Display the message**

In this step, you modify the second activity to display the message that was passed by the first activity. 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);**  
}

**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](https://developer.android.com/training/appbar).

To add an **Up** button, you need to declare which activity is the logical parent in the [AndroidManifest.xml](https://developer.android.com/guide/topics/manifest/manifest-intro) 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">  
        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**

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.

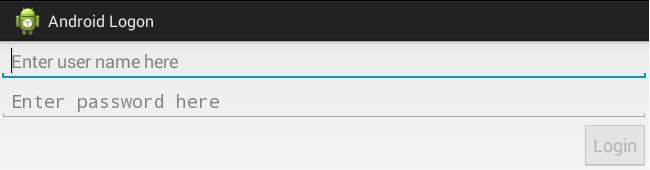


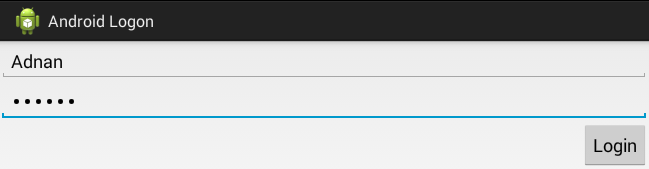
**Task:**

1. Create an application which can take user-id and password from user, and if the credentials are valid then it shows “Login successful”, otherwise it shows “login failure” message.

Your application must have following two activities:

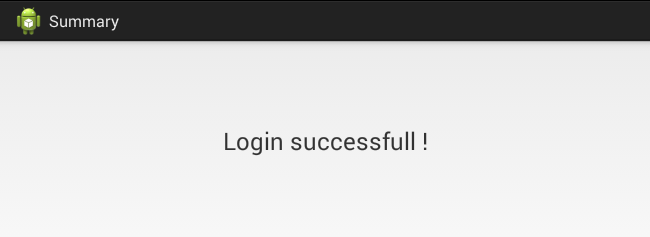
1. **LoginActivity**

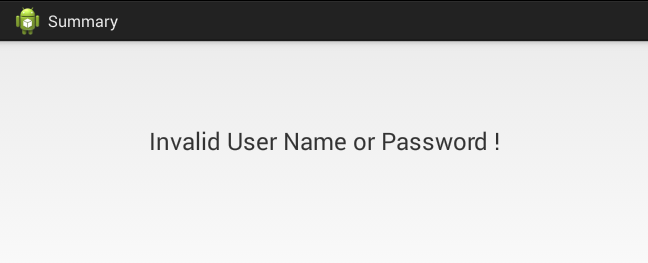




* Depending upon the user input, Login button must be enabled/disabled. You may use [*TextWatcher*](https://developer.android.com/reference/android/text/TextWatcher.html) interface with *addTextChangedListener()*
* When user pressed Login button user id and password must be passed to the ***SummaryActivity*** for authentication.

1. **SummaryActivity**





* Depending upon the credentials provided to this activity, appropriate message must be appeared.