Labsheet 3 – Thursday 9th September 2021

Multiple Activities and Intents

Software Development for Portable Devices

Information about HD TA:

AASHITA DUTTA <h20201030130@hyderabad.bits-pilani.ac.in>

Breakout room link: https://meet.google.com/erq-uxqi-egv

Information about FD TA:

ARUNEEMA DESHMUKH <f20180568@hyderabad.bits-pilani.ac.in>

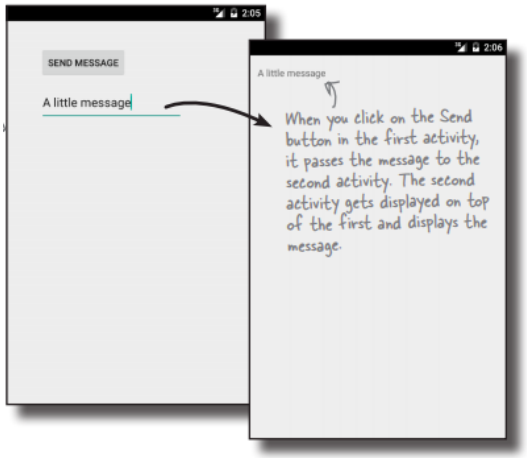
Breakout room link: https://meet.google.com/pvo-osiz-cuh

In this practical, you learn how to build apps with multiple activities. Further we will use activities within other existing apps as well.

1. Create a basic app with a single activity and layout.
   - Adding an onSendMessage() method to the activity which will be called by the SEND Button.
2. Add a second activity and layout.
   - Take a look at AndroidManifest.xml
3. Get the first activity to call the second activity.
   - Create and pass an Intent
4. Get the first activity to pass data to the second activity

App Description:

The app will contain two activities. The first activity will allow you to type a message. When you click on a button in the first activity (CreateMessageActivity), it will launch the second activity and pass it the message. The second activity (ReceiveMessageActivity) will then display the message.



Task 1: Create a new project: Messenger

1. Select **Empty Activity** as Project Template. After clicking next you can name this activity (**CreateMessageActivity**). [In case you do not get this option, you can rename the activity later, follow Renaming an Activity]

2. In the layout of this activity (activity_create_message.xml) add a **Button** and an **EditText.** The **EditText** element defines an editable text field for entering text. It inherits from the same Android View class.
    I. Let the Button have a text value of **@string/send.** Add a string resource in strings.xml called "send".Give it a value, this value is the text that will appear on the Button.
    ```
    <string name="send">SEND MESSAGE</string>
    ```
    II. Add a method called **"onSendMessage"** to the Button in the layout and in the CreateMessageActivity. This method will get called when the button is clicked.
    ```
    public void onSendMessage(View view) {}
    ```
    III. In the EditText add an **android:ems** attribute and set it to 10. [What is ems?]

activity_create_message.xml

```
<?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="16dp"
   android:paddingLeft="16dp"
   android:paddingRight="16dp"
   android:paddingTop="16dp"
   tools:context=".CreateMessageActivity" >
   <Button
       android:id="@+id/send"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_alignParentLeft="true"
       android:layout_alignParentTop="true"
       android:layout_marginLeft="36dp"
       android:layout_marginTop="21dp"
       android:onClick="onSendMessage"
       android:text="@string/send" />
   <EditText
       android:id="@+id/message"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_alignLeft="@+id/send"
       android:layout_below="@+id/send"
       android:layout_marginTop="18dp"
       android:ems="10" />
</RelativeLayout>
```

Questions:

1. Find out other attributes that can be used in place of ems?
2. How to limit edit text entry to single line?
3. What is the difference between an activity and a task?

Task 2: Create the second activity and layout

1. To create the new activity, choose **File** → **New** → **Activity**, and choose the option for Empty Activity. You will be presented with a new screen where you can choose options for your new activity. Give the new activity a name of **"ReceiveMessageActivity"** and the layout a name of "activity_receive_message". Do not select Launcher Activity as the first activity is set as Launcher Activity.
2. After clicking Finish a new file called ReceiveMessageActivity.java has been created in the app/ src/main/java folder, and a file called activity_receive_message.xml has been created under app/src/main/res/layout. This layout will have a **TextView** by default, no need to change it for now. If not add one.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

3. Android Studio has also made a configuration change to the app in a file called **AndroidManifest.xml**.

AndroidManifest.xml

Every Android app must include a file called AndroidManifest.xml.You can find it in the app/src/main folder of your project. The AndroidManifest.xml file contains essential information about your app, such as what activities it contains, required libraries, and other declarations. Android creates the file for you when you create the app.

All activities need to be declared in AndroidManifest.xml. If an activity is not declared in the file, the system will not know it exists. If you add extra activities manually (without the IDE), you'll need to edit AndroidManifest.xml yourself. The activity declaration may include other properties too, such as security permissions, and whether it can be used by activities in other apps.

Here's what our app's AndroidManifest.xml should include

```
<activity android:name=".ReceiveMessageActivity"></activity>
<activity android:name=".CreateMessageActivity">
   <intent-filter>
       <action android:name="android.intent.action.MAIN" />
       <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

Questions:

1. Why .(dot) is put before activity names?
2. Why are we creating new Activity and not a new custom Java class as we did in the last lab?

3. Identify other attributes inside the activity tag in the android manifest file.

Task 3: Call Second Activity

Next, we need to get CreateMessageActivity to call ReceiveMessageActivity when the user clicks the Send Message button. Whenever we want an activity to start a second activity, we use an intent.

Intent

An intent is a type of message. It's a type of message that allows you to bind separate objects (such as activities) together at runtime. If one activity wants to start a second activity, it does it by sending an intent to Android. Android will start the second activity and pass it the intent.

To create an Intent [**EXPLICIT**]:

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

The first parameter tells Android which object the intent is from, and you can use the word this to refer to the current activity. The second parameter is the class name of the activity that needs to receive the intent.This intent is an **explicit intent**; you explicitly tell Android which class you want it to run.

To pass this Intent (This tells Android to start the activity specified by the intent):

```
startActivity(intent);
```

If there's an action you want to be performed but you let the user choose which app's activity does it, you can create an **implicit intent**.More about implicit intent will be discussed later.

Once Android receives the intent, it checks everything's OK and tells the activity to start. If it can't find the activity, it throws an ActivityNotFoundException.

To know more refer android.content.Intent

1. We want to start the ReceiveMessageActivity when the Send Message Button is Clicked. So in onSendMessage() method, create a new intent and use the intent in the startActivity() method. Import android.content.Intent;

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

Run the app. Press the button to change the Activity

Questions:

1. What are the uses of intents apart from launching an activity from existing activity?
2. Can you list down the attributes contained in Intent?
3. What is the need of android:parentActivityName=".MainActivity" in the android Manifest file?

Task 4: Pass the Data

Next, we'll get CreateMessageActivity to pass text to ReceiveMessageActivity so that ReceiveMessageActivity can display it. In order to accomplish this, we'll do three things:

1. First, update the layout activity_receive_message.xml. Give the TextView element an ID of "message". Delete the android:text attribute.
2. Update CreateMessageActivity, so that it gets the text the user inputs.
   Try it Yourself: Use getText() to get the Text from EditText and toString(); to store it in a string called messageText.
3. CreateMessageActivity then needs to add the text to the intent before it sends it. We can do so with the **putExtra()** method. putExtra() lets you put extra information in the intent you're sending.
   Eg: `intent.putExtra("message", value);`
   Here *message* is a String name for the value that is passing in, and *value* is the value. The putExtra() method is overloaded so *value* has many possible types. As an example, it can be a primitive such as a boolean or int, an array of primitives, or a String. putExtra() can be used repeatedly to add numerous extra data to the intent, however each one should be given a unique name.

   Try it Yourself: Add the text from the EditText to the intent, give it a name of "message".

```
public void onSendMessage(View view) {
    EditText messageView = (EditText) findViewById(R.id.message);
    String messageText = messageView.getText().toString();
    Intent intent = new Intent(this, ReceiveMessageActivity.class);
    intent.putExtra("message", messageText);
    startActivity(intent);
}
```

   Now change the name to **ReceiveMessageActivity.EXTRA_MESSAGE**
   `intent.putExtra(ReceiveMessageActivity.EXTRA_MESSAGE, messageText);`
   We will use a constant for the name of the extra information so that we know CreateMessageActivity and ReceiveMessageActivity are using the same String. We'll add this to ReceiveMessageActivity in the next step.
   You can also see what fix the IDE is suggesting when there is an error, in case the symbol EXTRA_MESSAGE is not found.
4. Add the constant `public static final String EXTRA_MESSAGE = "message" ;`
   in ReceiveMessageActivity. As the activity's *onCreate()* method gets called as soon as the activity is created, we'll add the rest of the code to this method. We will retrieve the information and set it to the TextView.
5. To retrieve this information from the intent, in ReceiveMessageActivity.We will need to use

   `Intent intent = getIntent();`

   getIntent() returns the intent that started the activity, and you can use this to retrieve any extra information that was sent along with it. How you do this depends on the type of information that was sent.
   Eg:

```
String string = intent.getStringExtra("message");
int intNum = intent.getIntExtra("name", default_value);
```

   Try it Yourself: get the intent using the getIntent() method, then get the value of the message using getStringExtra(EXTRA_MESSAGE). Display the text using setText().

```
package com.example.messenger;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class ReceiveMessageActivity extends AppCompatActivity {


   public static final String EXTRA_MESSAGE =  "message" ;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_receive_message);
       Intent intent = getIntent();
       String messageText = intent.getStringExtra(EXTRA_MESSAGE);
       TextView messageView = (TextView)findViewById(R.id.message);
       messageView.setText(messageText);
   }
}
```

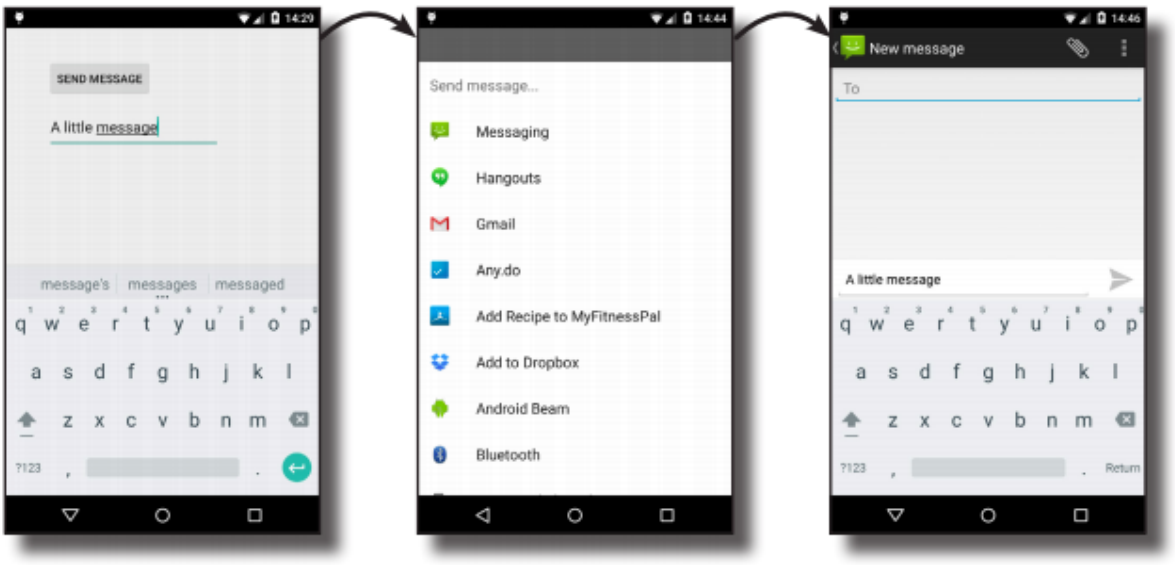
Run the app.

Questions:

1. What are other extended data retrieved by intent other than getStringExtra or getIntExtra ?
2. Difference between putExtra() and putExtras() or getExtra() and getExtras().

Now we will modify the app to use implicit intent. We can send the message to other people by integrating the app with other message sending apps already on the device. The steps to follow are

1. Create an intent that specifies an action.
   - Learn more about Actions
   - Understand how Intent filters work
2. Allow the user to choose which app to use



Task 1: Using Implicit Intents

1. Instead of creating an intent that's explicitly for ReceiveMessageActivity, we're creating an intent that uses a send action. The next activity should be able to handle data with a MIME data-type of "text/plain".We will send the message text using putExtra(). If some extra information for eg: subject, isn't relevant to a particular app, it will just ignore this information. Any apps that know how to use it will do so.

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, messageText);
startActivity(intent);
```

Actions

Actions are a way of informing Android what standard operations activities can perform. As an example, Android knows that all activities registered for a send action are capable of sending messages. Some examples of actions are Intent.ACTION_DIAL to dial a number, Intent. ACTION_WEB_SEARCH to perform a web search, and Intent.ACTION_SEND to send a message.

While creating an IMPLICIT Intent, we can specify what kind of action we want the next activity to perform. To create such an intent use

`Intent intent = new Intent(action);`

2. If just one activity is able to receive the intent, Android tells the activity to start and passes it the intent. If more than one activity is able to receive the intent, Android displays an activity chooser dialog and asks the user which one to use. When Android is given an intent, it has to figure out which activity, or activities, are able to handle it. This process is known as **intent resolution.** When an implicit intent is used, Android uses the information in the intent to figure out which components are able to receive it by checking the intent filters in every app's copy of AndroidManifest.xml.

Intent Filters
An intent filter specifies what types of intent each component can receive.

As an example, here's the entry for an activity that can handle an **action** of ACTION_SEND. The activity is able to accept **data** with MIME types of text/plain or image:

```
<activity android:name=".ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
        <data android:mimeType="image/*"/>
    </intent-filter>
</activity>
```

The intent filter also specifies a category. The category supplies extra information about the activity such as whether it can be started by a web browser, or if it's the main entry point of the app. An intent filter must include a **category** of android.intent.category.DEFAULT if it's to receive implicit intents. If an activity has no intent filter, or it doesn't include a category name of android.intent.category. DEFAULT, it means that the activity can't be started with an implicit intent. It can only be started with an explicit intent using the fully qualified component name.

Run the app at this point.

Questions:

1. What are the possible data types for intent.setType(type)?
2. What are different kinds of actions that an activity can handle other than SEND?
3. Can we have multiple "category" attributes inside <intent-filter>?

Task 2: Create a chooser

One can create a chooser that asks the user to pick an activity without asking if you always want to use it. The **Intent.createChooser()** method takes the intent that is already created, and wraps it in a chooser dialog. createChooser() method allows the developer to specify a title for the chooser dialog, and doesn't give the user the option of selecting an activity to use by default. It also lets the user know if there are no matching activities by displaying a message. For example

```
Intent chosenIntent = Intent.createChooser(intent, "Send message...");
```

The method takes two parameters: an intent and an optional String title for the chooser dialog window. The Intent parameter needs to describe the types of activity you want the chooser to display. The createChooser() method returns a brand-new Intent. This is a new explicit intent that's targeted at the activity chosen by the user. It includes any extra information supplied by the original intent, including any text. To start the activity the user chose, you need to call

startActivity(chosenIntent);

1. Add a string resource to strings.xml for the title of the Chooser.
   `<string name="chooser">Send message...</string>`
2. Update the onSendMessage() method.
   Try it yourself: Retrieve the value of the chooser string resource in strings.xml using `getString(R.string.chooser);` call the createChooser() method, and then start the activity the user chooses

   Run the app.

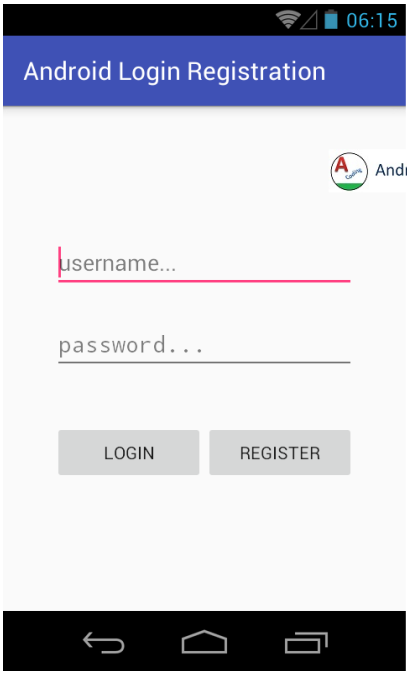
Questions:

1. Why would one use this type of chooser over the default Android chooser?
2. What would happen if no matching activities are found to perform action?
3. How android checks which activities are able to receive the intents?

Additional Exercises (To be completed as home assignment)- Based on Explicit Intent

Design Simple User Register/Login Activity.

1. Create App- **AndroidLoginRegistration**
2. Create Main Activity- **UserForm**, along with layout file having two Edit Text widgets as Username and Password along with placeholders as username.. and password.. and two buttons having labels as LOGIN and REGISTER as shown below:



3. Create two more activities and modify android manifest file accordingly:

- First Activity- **RegisterSuccess** Activity will display a message as shown below on click of REGISTER button. In place of User, pass your username from Main Activity to First Activity and print the following using the activity:

"Manik" Registered Successfully

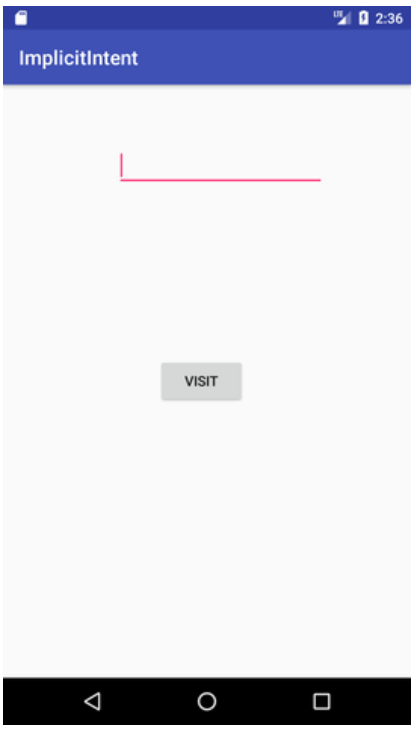
- Second Activity - **LoginSuccess** Activity will display a message as shown below on click of LOGIN button. No username/password check needs to be done for now since there is no database integration.

** Each of the activities will have their own layout file and design accordingly. Try various UI design layouts for UserForm by referring https://dribbble.com/tags/login_screen.

Additional Exercises (To be completed as home assignment)- Based on Implicit Intents, Actions and Intent Filters

Try out simple example of implicit intent that displays a web page

1. Create App- **WebPageView**
2. Create Main Activity: **VisitWebPage** along with layout file having one Edit Text widget and one button having label as VISIT as shown below:



3. Define Intent in VisitWebPage activity to open the browser on click on visit button after typing URL for say “https://developer.android.com/”. Android can find more than one matching app (like Chrome, Safari) to open, Choose one. Your browser will open the requested page.
4. Identify what type of action will be used defined by Intent and how to pass URL.
5. Try implementing intent.createChooser to choose any browser application.