08027 Lab 2 Activities and Intents

Goal

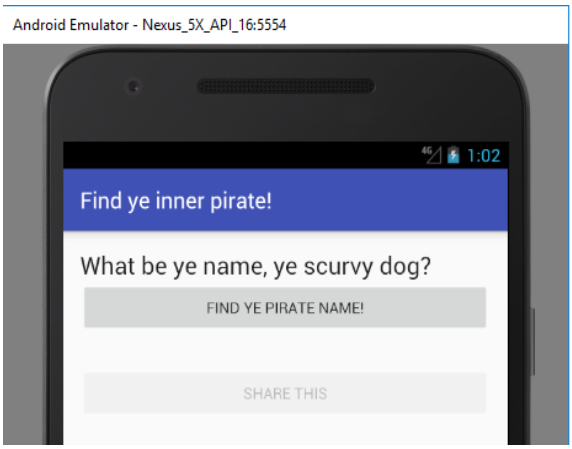
Towards the end of the last lab we learnt about how to view log messages and at the same time we took a brief look at the activity lifecycle. In this lab we learn how to programmatically start new activities from existing ones using intents. We will learn how to pass data to new activities we launch and how to retrieve a result from another activity that we started, and we will look at how we can start other activities that are not part of our application using implicit intents. Finally, you will have an opportunity to exercise what you learnt last week about resources to tidy up some messy code. To do this we're going to create an application to help you find your inner pirate! Arrgh!

Check out the folder **<your repository>**\Week2\InnerPirate to C:\Temp and open this application in Android Studio. When you've finished remember to commit your changes and delete your work from the local drive.

This ~~tedious~~ fun application helps the user discover their inner pirate by giving them a pirate nickname and asking them to join a ship's crew. All the activities have been written for you. Your job is just to join activities up by launching new activities, and understand the different ways we can use intents. Passing data to new activities and getting results back from activities. You will have the opportunity to share your new found vocation with friends. Finally, we will tidy up some of the code using lessons we learnt in the last lab.

Explicit Intents

Intents are used to send a message to the Android OS request that it start a new activity. This activity is usually added to the task back stack of the current task. Run the code in an AVD and you should see this window.

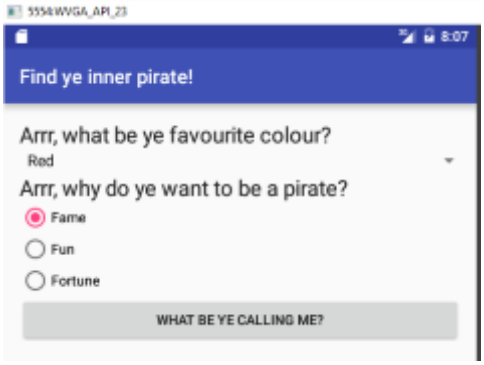


We can see some text, a button, a gap that we'll later fill with our pirate name, and a greyed out button to share our pirate name with the world. When the user clicks the "Find ye pirate name!" button we want to launch a new activity called findNameActivity. As we know explicitly which activity we want to launch we can achieve this using an explicit intent.

In the MainActivity java file find the findNameButtonOnClick method. In that method create and intent. Pass that intent a Context (in most cases the current activity is the context, so we can just use this), and the activity class that you want to launch. Then pass the new intent into the startActivity method to launch the intent. You will also need to import android.content.Intent at the top of the class.

```
public void findNameButtonOnClick(View pView){
    Intent intent = new Intent(this, findNameActivity.class);
    startActivity(intent);
}
```

Run the code and click the button. You should be taken to a window that asks for your favourite colour, and what motivation you have to become a pirate. Once the user has entered that information they can click "What be ye calling me?" to calculate their pirate name using a highly scientific algorithm.



At this point you might notice a bug that arises with a specific combination of colour and motive. What might you do to fix it?

Commit with an appropriate log message like *"Completed lab 2.1 task 1 launched the findNameActivity using an explicit intent"*

So far we've launched one activity from another using an explicit intent.

Once the pirate name has been determined, we want to offer the user to select a ship and crew, or to set sail alone. If the user chooses to set sail alone they should be returned to the previous screen. We'll deal with that later. First let's find out how to find a ship and crew. To do that we want to launch the chooseShipActivity.

Before we can use the button we need to enable it at the of the getNameButtonOnClick method. Do this by adding the lines after the code that finds your pirate name in the findNameActivity class.

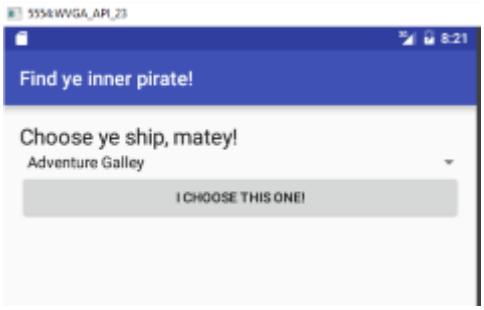
```
Button findShipButton = (Button)findViewById(R.id.findShipButton);
findShipButton.setEnabled(true);
```

Android studio will complain that it doesn't understand what "Button" is. To remedy this select the error and press alt and enter together, which should add the correct import statement for you. Remember this trick. You will have to do this at various points throughout the labs.

Run the code to verify that now once your pirate name has been determined the Find A Ship and Crew button is activated.

Commit with an appropriate log message like *"Completed lab 2.1 task 2 enabled the findShipButton when the pirate name is derived"*

Find the findShipButtonOnClick method in the findNameActivity java file. When this method is called we want to launch the chooseShipActivity in the same way that we did for the findNameActivity. Just as we did in the previous section add the code to create the intent and use it to start the new activity. Run your code to verify that this has worked.



Commit with an appropriate log message like *"Completed lab 2.1 task 3 launched the chooseShipActivity using an explicit intent"*

This "Choose ye ship, matey!" message seems a bit impersonal now that we know the pirate name. It would be better if we could personalise the message. To do this we need a way to put extra data into our intent. Luckily, there is a method to do just that.

The first step is to create a private String datamember called m_PirateName in the findNameActivity class. Then when we work out what the pirate name is in the getNameButtonOnClick method we need to store that name in our new data member. That way we can access the pirate name when we launch our new activity. Once we've done all that, before we start the chooseShipActivity we can put some extra data into our intent with the putExtra method.

```
intent.putExtra("pirateName", m_PirateName);
```

The extra data is stored in a way similar to key value pairs. In this case pirateName is the key, and whatever was stored in m_PirateName is the value.

In the onCreate method of the chooseShipActivity we should now be able to get hold of the pirate name and change the welcome message to be more personal. You can do this with the following code.

```
String pirateName = getIntent().getStringExtra("pirateName");
TextView greeting = (TextView)findViewById(R.id.greetingTextView);
greeting.setText("Choose ye ship, " + pirateName + "!");
```

Check that this is working as expected, then if it is commit with an appropriate log message like *"Completed lab 2.1 task 4 put extra data into the intent used to start the chooseShipActivity and extracted that data to personalise the greeting"*

Once the user has chosen a ship they should click the button, and then we need to get back to the first activity. To do this we need to destroy the current activity. This will cause it to come off the task back stack and we will drop down to the previous name finding activity. To end the current activity we need to call the finish method. Do that in the chooseShipButtonOnClick method.

Check the code works. You should notice that we are able to drop down to the previous activity, but then we're kind of stuck.

Commit your work with an appropriate log message like *"Completed lab 2.1 task 5 finished the chooseShipActivity when the chooseShipButton is clicked"*

In the previous exercise we need to pass information on to an activity we were launching using putExtra. This time though, we want to get some information – the name of the ship the user chose – back from the activity we chose. To do this when we launch the activity instead of passing the intent object to the startActivity method we need to pass it to a startActivityForResult method. Start activity for result requires an additional parameter that is used to determine what request is being responded to, and which result should be expected. Add a static final integer to your findNameActivity class called CHOOSE_SHIP_REQUEST and set it to have the value 1.

```
static final int CHOOSE_SHIP_REQUEST = 1;
```

Then pass this, along with our intent to the startActivityForResult method.

```
startActivityForResult(intent, CHOOSE_SHIP_REQUEST);
```

The purpose of this parameter is so that we know what our intent was when the result is returned. In this case it is trivial, but if a single activity can launch more than one activity that requires a result this becomes very important.

Next, we need to pay some attention to the chooseShipButtonOnClick method in the chooseShipActivity class. In this method we need to set the result of the activity to OK, to show that the activity completed correctly. We also need to add a key value pair for the chosen ship to an intent to be passed back to the previous activity.

```
public void chooseShipButtonOnClick(View pView) {
    Intent data = new Intent();
    Spinner shipSpinner = (Spinner)findViewById(R.id.shipSpinner);
    String ship = (String)shipSpinner.getSelectedItem();
    data.putExtra("ship", ship);
    setResult(RESULT_OK, data);
    finish();
}
```

Because we used the startActivityForResult method to launch the new activity, when that new activity finishes the onActivityResult method in findNameActivity will be called. To pick up that call we should override that method. We need to test that the request code is the same as the request code that we're expecting, and that the new activity finished as we expect, and not for some other reason, such as the user clicking the back button. Once we've verified that we can extract the information we need from the Intent object that has been passed back. After this method is called we're simply going to pass the same data, along with the pirate name we stored earlier back to the MainActivity object.

Override the onActivityResult method like this:

```
@Override
protected void onActivityResult(int pRequestCode, int pResultCode, Intent pData) {
    if(pResultCode == RESULT_OK && pRequestCode == CHOOSE_SHIP_REQUEST) {
        if(pData.hasExtra("ship")) {
            Intent data = new Intent();
            data.putExtra("ship", pData.getStringExtra("ship"));
            data.putExtra("name", m_PirateName);
            setResult(RESULT_OK, data);
            finish();
        }
    }
}
```

We want to get our pirate name and ship all the way back out to the first activity. That means that we need to change the intent in the first activity to also return a result. You can do this by changing call to the startActivity method in the MainActivity class. You'll also have to add an appropriate result code, and before the call to finish in onActivityResult you'll have to populate an intent with the data you want to return and set the result of the activity. You've done all this in the previous step, so refer to that, and if you get stuck as for some assistance.

The final step is to override the onActivityResult method in the MainActivity class. Use that to report the information that has been returned in the pirateNameTextView.

```
@Override
protected void onActivityResult(int pRequestCode, int pResultCode, Intent pData) {
    if(pResultCode == RESULT_OK && pRequestCode == CHOOSE_NAME_AND_SHIP_REQUEST) {
        if(pData.hasExtra("name") && pData.hasExtra("ship"))
        {
            TextView pirateName = (TextView)findViewById(R.id.pirateNameTextView);
            String result = "Arr, 'tis " + pData.getStringExtra("name") +
                    " of the " + pData.getStringExtra("ship");
            pirateName.setText(result);
        }
    }
}
```

Check this all works as expected. If it does, you should be able to start the application, get a pirate name, choose a ship, and then return to the start page where your choice will be displayed to you.

If it's working as expected then commit your code with an appropriate log message like *"Completed lab 2.1 task 6 passed results back from the chooseShipActivity, through the findNameActivity all the way back to the mainActivity"*

Take a moment to consider what is happening. The mainActivity now calls the findNameActivity using startActivityForResult which means when that activity finishes the mainActivity class's onActivityResult method will be called. In the findNameActivity we generate a name and store that in a datamember in that class. We also start a new chooseShipActiviy, again using startActivityForResult. We also put extra data into the intent that starts the chooseShipActivity so that we can personalise that activity with the newly generated pirate name. Once a ship is chosen we build some data to be returned and finish the activity. Because that activity was started with startActivityForResult in the findNameActivity class as the chooseShipActivity finishes the onActivityResult method in findNameActivity is called. This method effectively adds the pirate name we stored previously, sets up some data to be passed back and then finishes that activity. This causes the onActivityResult method to be called in the MainActivity class, which can then unpack the data and use it to populate the textView in MainActivity.

Implicit Intents

Now we have a pirate name and a pirate ship it's time to share that information with the world. To do that we're going to use an implicit intent. An implicit intent is used to communicate to the operating system what you want to do without explicitly specifying what activity you want to do it. This means that your application can make use of other activities available on your device.

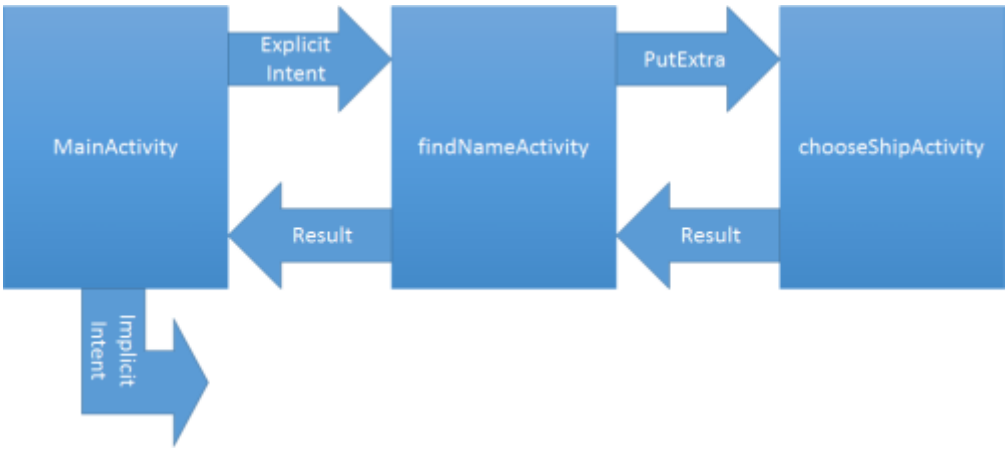
In the onActivityResult method in the MainActivity, after you set the text of the pirateNameTextView enable the shareButton using the following code:

```
Button shareButton = (Button)findViewById(R.id.shareButton);
shareButton.setEnabled(true);
```

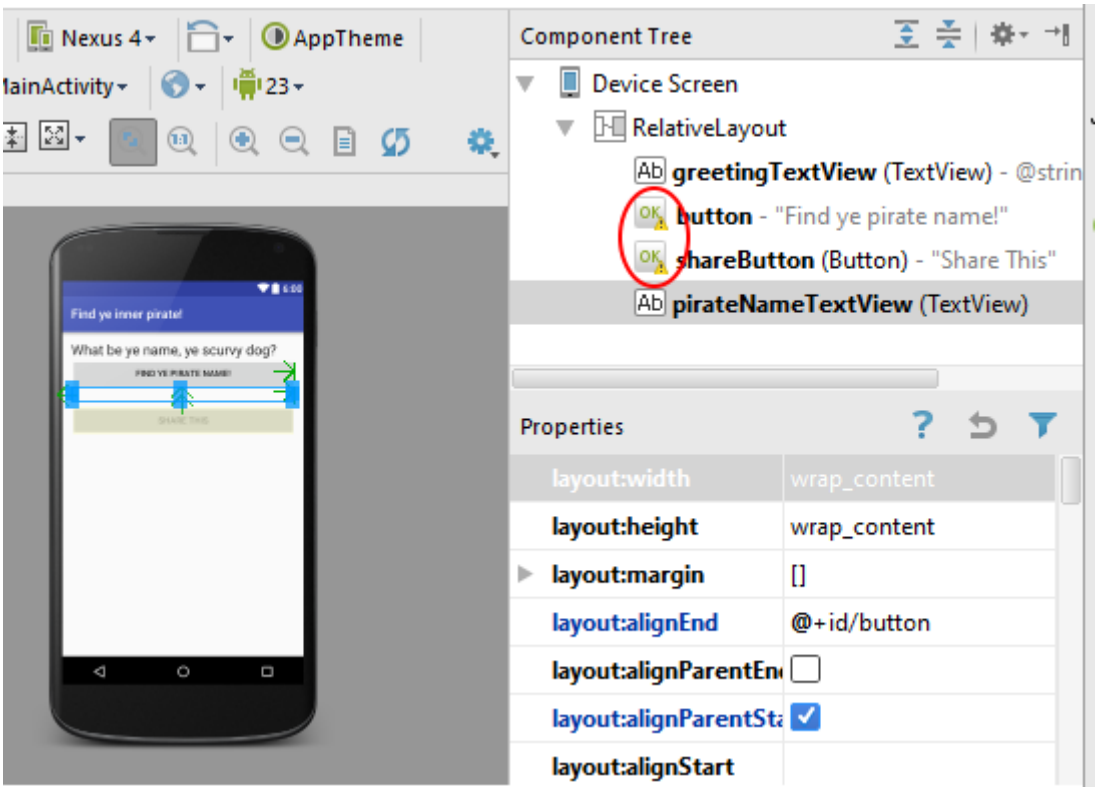
Next, create a String data member in MainActivity called m_Result. Use it to store the string we built in onActivityResult so that we can access is in the shareButtonOnClick method. Then in the shareButtonOnClick method create an implicit intent to send a text message from your phone.

```
public void shareButtonOnClick(View pView) {
    Intent intent = new Intent(Intent.ACTION_SENDTO, Uri.parse("smsto:555555"));
    intent.putExtra("sms_body", m_Result);
    startActivity(intent);
}
```

Run this code, and then see if you can send to sms. If you can the commit your code with an appropriate message like *"Completed lab 2.1 task 7 used an Implicit Intent to send the result of our application via SMS"*.



Finally, open each of the layout files. Note that some of the components are displaying yellow warning icons. These indicate that we have some hardcoded strings in our UI.



Last week you learnt how to store strings like these as resources. See if you can tidy up all of the warning icons from all of the layouts. If you double click the yellow icons android will provide a quick and easy way to edit your resource files.

Once you've sorted out all the hardcoded strings in all three activities commit your code with an appropriate message like *"Completed lab 2.1 task 8 tidied up the hardcoded strings"*

Summary

In this lab you've learnt how to use intents to launch new activities. You've used explicit intents when we know exactly what activity you want to launch. You've passed extra data to activities, and learnt how to retrieve data from activities when they have finished. Finally, you've learnt how to use activities outside of your application using implicit intents.