Android Development

W04 Practical - Activity Life Cycle

# App Overview

This is a pretty straightforward practical.

I’ve included, in this week’s resources, a starting code base that extends the TwoActivities App you built last week. It contains new functionality to return a String from the SecondActivity as a result following a call to startActivityForResult() in the MainActivity.

In this session you are going to implement some of the lifecycle callbacks that we’ve just discussed, and you will explore them by adding logging statements and observing the logs during runtime.

The changes you make will not affect the observed user behaviour.

# Explore the Project

The project should look familiar to you but look at a number of significant differences (additions)

## Linking the Button Views to onClick handlers is done in code

This is another way to implement UI callbacks that we discussed in last week’s sessions.

* In the code below from MainActivity the onClick() listener not a class method (linked from the XML) but a class instance variable containing that method.

**// Declare an OnClickListener for our button  
private OnClickListener buttonListener = new OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Log.d(LOG\_TAG, "Button clicked!");  
 Intent intent = new Intent(getApplicationContext(), SecondActivity.class);  
 String message = mMessageEditText.getText().toString();  
 intent.putExtra(EXTRA\_MESSAGE, message);  
 startActivityForResult(intent, TEXT\_REQUEST);  
 }  
};**

* The View is linked to the callback in the Activity onCreate() method as follows:

**@Override  
protected void onCreate(Bundle savedInstanceState) {  
  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
  
 // Initialize all the view variables.  
 mMessageEditText = findViewById(R.id.editText\_main);  
 mReplyHeadTextView = findViewById(R.id.text\_header\_reply);  
 mReplyTextView = findViewById(R.id.text\_message\_reply);  
 mSendButton = (Button) findViewById(R.id.button\_main);  
  
 // Set the 'onClick' listener for the button  
 mSendButton.setOnClickListener(buttonListener);  
  
}**

## MainActivity invokes SecondActivity using startActivityForResult

* The startActivityForResult call takes an intent but additionally an int request id argument so that we can check in the callback that we’re being told about the request we sent.

**// Unique tag required for the intent extra  
public static final String EXTRA\_MESSAGE = "uk.ac.hope.twohopeactivities.extra.MESSAGE";  
// Unique tag for the intent reply  
public static final int TEXT\_REQUEST = 1;**

**< CODE >v**

**Intent intent = new Intent(getApplicationContext(), SecondActivity.class);**

**intent.putExtra("EXTRA\_MESSAGE", messageTextView.getText().toString());**

**startActivityForResult(intent, *SEND\_MESSAGE\_REQUEST\_ID*);**

## MainActivity overrides (implements) onActivityResult

* The main activity implements the onActivityResult callback in order to handle the result of implementing SecondActivity.

***// onActivityResult implementation*@Override  
public void onActivityResult(int requestCode, int resultCode, Intent intent) {  
  
 super.onActivityResult(requestCode, resultCode, intent);  
 *// Identify activity* if (requestCode == *SEND\_MESSAGE\_REQUEST\_ID*) {  
 *// Activity succeeded* if (resultCode == *RESULT\_OK*) {  
 String reply = intent.getStringExtra(SecondActivity.*EXTRA\_REPLY*);  
 replyTextView.setText("Reply: " + reply);  
 }  
 }  
}**

# Add lifecycle callbacks

## Implement callbacks in MainActivity

1. Open the TwoActivities project in Android Studio, and open **MainActivity** in the **Project > Android** pane.
2. In the onCreate() method, add the following log statements:

**Log.d(LOG\_TAG, "-------");  
Log.d(LOG\_TAG, "onCreate");**

1. Add an override for the onStart() callback, with a statement to the log for that event:

**@Override**

**public void onStart(){**

**super.onStart();**

**Log.d(LOG\_TAG, "onStart");**

**}**

For a shortcut, select **Code > Override Methods** in Android Studio. A dialog appears with all of the possible methods you can override in your class. Choosing one or more callback methods from the list inserts a complete template for those methods, including the required call to the superclass.

1. Use the onStart() method as a template to implement the onPause(), onRestart(), onResume(), onStop(), and onDestroy() lifecycle callbacks.

All the callback methods have the same signatures (except for the name). If you **Copy** and **Paste**onStart() to create these other callback methods, don't forget to update the contents to call the right method in the superclass, and to log the correct method.

1. Run your app.
2. Click the **Logcat** tab at the bottom of Android Studio to show the **Logcat** pane. You should see three log messages showing the three lifecycle states the Activity has transitioned through as it started:

## Implement lifecycle callbacks in SecondActivity

1. Open SecondActivity.
2. At the top of the class, add a constant for the LOG\_TAG variable:

**private static final String LOG\_TAG = SecondActivity.class.getSimpleName();**

1. Add the lifecycle callbacks and log statements to the second Activity. (You can Copy and Paste the callback methods from MainActivity.)
2. Add a log statement to the returnReply() method just before the finish() method:

**Log.d(LOG\_TAG, "End SecondActivity");**

## Observe the log as the App runs

Experiment using your app and note that the lifecycle events that occur in response to different actions. In particular, try these things:

* Use the app normally (send a message, reply with another message).
* Use the Back button to go back from the second Activity to the main Activity.
* Use the Up arrow in the app bar to go back from the second Activity to the main Activity.
* Rotate the device on both the main and second Activity at different times in your app and observe what happens in the log and on the screen.
* Press the overview button (the square button to the right of Home) and close the app (tap the **X**).
* Return to the home screen and restart your app.

# Save and Restore the Activity instance state

Depending on system resources and user behaviour, each Activity in your app may be destroyed and reconstructed far more frequently than you might think.

You may have noticed this behaviour in the last section when you rotated the device or emulator. Rotating the device is one example of a device configuration change. Although rotation is the most common one, all configuration changes result in the current Activity being destroyed and recreated as if it were new. If you don't account for this behaviour in your code, when a configuration change occurs, your Activity layout may revert to its default appearance and initial values, and your users may lose their place, their data, or the state of their progress in your app.

The state of each Activity is stored as a set of key/value pairs in a Bundle object called the Activity instance state. The system saves default state information to instance state Bundle just before the Activity is stopped and passes that Bundle to the new Activity instance to restore.

To keep from losing data in an Activity when it is unexpectedly destroyed and recreated, you need to implement the onSaveInstanceState() method. The system calls this method on your Activity (between onPause() and onStop()) when there is a possibility the Activity may be destroyed and recreated.

The data you save in the instance state is specific to only this instance of this specific Activity during the current app session. When you stop and restart a new app session, the Activity instance state is lost and the Activity reverts to its default appearance. If you need to save user data between app sessions, use shared preferences or a database.

## **Save the Activity instance state with onSaveInstanceState()**

You may have noticed that rotating the device does not affect the state of the second Activity at all. This is because the second Activity layout and state are generated from the layout and the Intent that activated it. Even if the Activity is recreated, the Intent is still there and the data in that Intent is still used each time the onCreate() method in the second Activity is called.

In addition, you may notice that in each Activity, any text you typed into message or reply  EditText elements is retained even when the device is rotated. This is because the state information of some of the View elements in your layout are automatically saved across configuration changes, and the current value of an EditText is one of those cases.

So, the only Activity state you're interested in are the TextView elements for the reply header and the reply text in the main Activity. Both TextView elements are invisible by default; they only appear once you send a message back to the main Activity from the second Activity.

In this task you add code to preserve the instance state of these two TextView elements using **onSaveInstanceState**().

1. Open MainActivity.
2. Add this skeleton implementation of onSaveInstanceState() to the Activity, or use Code > Override Methods to insert a skeleton override.

**@Override**

**public void onSaveInstanceState(Bundle outState) {**

**super.onSaveInstanceState(outState);**

**}**

1. Check to see if the header is currently visible, and if so put that visibility state into the state Bundle with the putBoolean() method and the key "reply\_visible".

**if (mReplyHeadTextView.getVisibility() == View.VISIBLE) {**

**outState.putBoolean("reply\_visible", true);**

**}**

Note that the reply header and text are marked invisible until there is a reply from the second Activity. If the header is visible, then there is reply data that needs to be saved. Note that we're only interested in that visibility state — the actual text of the header doesn't need to be saved, because that text never changes.

1. Inside that same check, add the reply text into the Bundle.

**outState.putString("reply\_text",mReplyTextView.getText().toString());**

If the header is visible you can assume that the reply message itself is also visible. You don't need to test for or save the current visibility state of the reply message. Only the actual text of the message goes into the state Bundle with the key "reply\_text".

You save the state of only those View elements that might change after the Activity is created. The other View elements in your app (the EditText, the Button) can be recreated from the default layout at any time.

Note that the system will save the state of some View elements, such as the contents of the EditText.

## **Restore the Activity instance state in onCreate()**

Once you've saved the Activity instance state, you also need to restore it when the Activity is recreated. You can do this either in onCreate(), or by implementing the onRestoreInstanceState() callback, which is called after onStart() after the Activity is created.

Most of the time the better place to restore the Activity state is in onCreate(), to ensure that the UI, including the state, is available as soon as possible. It is sometimes convenient to do it in onRestoreInstanceState() after all of the initialization has been done, or to allow subclasses to decide whether to use your default implementation.

1. In the onCreate() method, after the View variables are initialized with findViewById(), add a test to make sure that savedInstanceState is not null.

**// Initialize all the view variables.**

**mMessageEditText = findViewById(R.id.editText\_main);**

**mReplyHeadTextView = findViewById(R.id.text\_header\_reply);**

**mReplyTextView = findViewById(R.id.text\_message\_reply);**

**// Restore the state.**

**if (savedInstanceState != null) {**

**}**

When your Activity is created, the system passes the state Bundle to onCreate() as its only argument. The first time onCreate() is called and your app starts, the Bundle is null—there's no existing state the first time your app starts. Subsequent calls to onCreate() have a bundle populated with the data you stored in onSaveInstanceState().

1. Inside that check, get the current visibility (true or false) out of the Bundle with the key "reply\_visible".

**if (savedInstanceState != null) {**

**boolean isVisible = savedInstanceState.getBoolean("reply\_visible");**

**}**

1. Add a test below that previous line for the isVisible variable.

**if (isVisible) {**

**}**

If there's a reply\_visible key in the state Bundle (and isVisible is therefore true), you will need to restore the state.

1. Inside the isVisible test, make the header visible.

**mReplyHeadTextView.setVisibility(View.VISIBLE);**

1. Get the text reply message from the Bundle with the key "reply\_text” and set the reply TextView to show that string.

**mReplyTextView.setText(savedInstanceState.getString("reply\_text"));**

1. Make the reply TextView visible as well:

**mReplyTextView.setVisibility(View.VISIBLE);**

1. Run the app. Try rotating the device or the emulator to ensure that the reply message (if there is one) remains on the screen after the Activity is recreated.