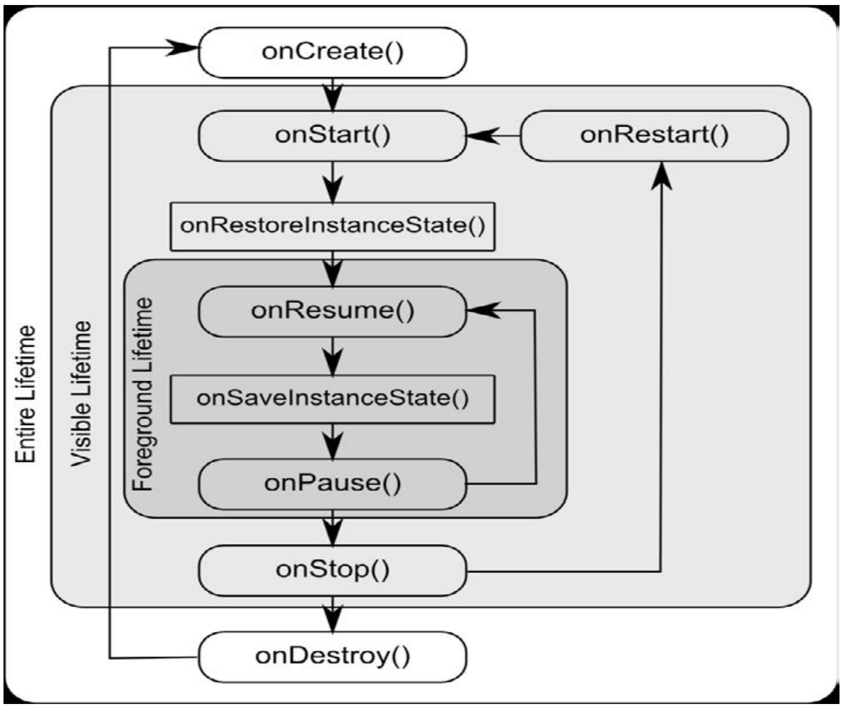
LAB 2 – Understanding Android Application and Activity Lifecycles

At the end of this lab, students should be able to:

- Know about activity lifecycle.
- Create an example Android application for the purpose of demonstrating different lifecycles states.
- Save and restore an activity's dynamic state.

Activities Lifecycle

If you have worked with C, C++ or Java programming language then you must have seen that your program starts from main() function. Very similar way, Android system initiates its program within an Activity starting with a call on onCreate() callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity lifecycle diagram:



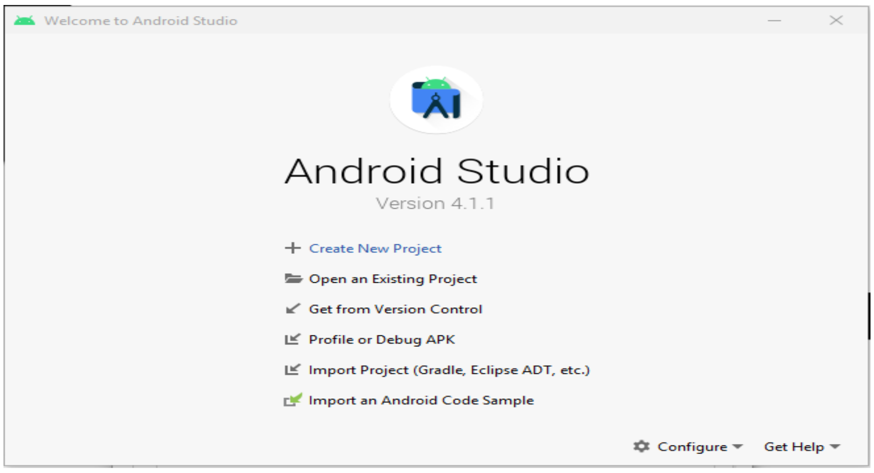
Callback Methods	Description
onCreate()	This is the first callback and called when the activity is first created.
onRestart()	This callback is called when the activity restarts after stopping it.
onStart()	This callback is called when the activity becomes visible to the user.
onResume()	Indicates that the activity is now at the top of the activity stack and is the activity with which the user is currently interacting.
onPause()	Indicates that a previous activity is about to become the foreground activity. This call will be followed by a call to

	either the onResume() or onStop() method depending on whether the activity moves back to the foreground or becomes invisible to the user.
onStop()	The activity is now no longer visible to the user.
onDestroy()	This callback is called before the activity is destroyed by the system.
onRestoreInstanceState(Bundle savedInstanceState)	This method is called immediately after a call to the onStart() method in the event that the activity is restarting from a previous invocation in which state was saved. As with onCreate(), this method is passed a Bundle object containing the previous state data. This method is typically used in situations where it makes more sense to restore a previous state after the initialization of the activity has been performed in onCreate() and onStart().
onSaveInstanceState(Bundle outState)	Called before an activity is destroyed so that the current dynamic state (usually relating to the user interface) can be saved. The method is passed the Bundle object into which the state should be saved and which is subsequently passed through to the onCreate() and onRestoreInstanceState() methods when the activity is restarted.

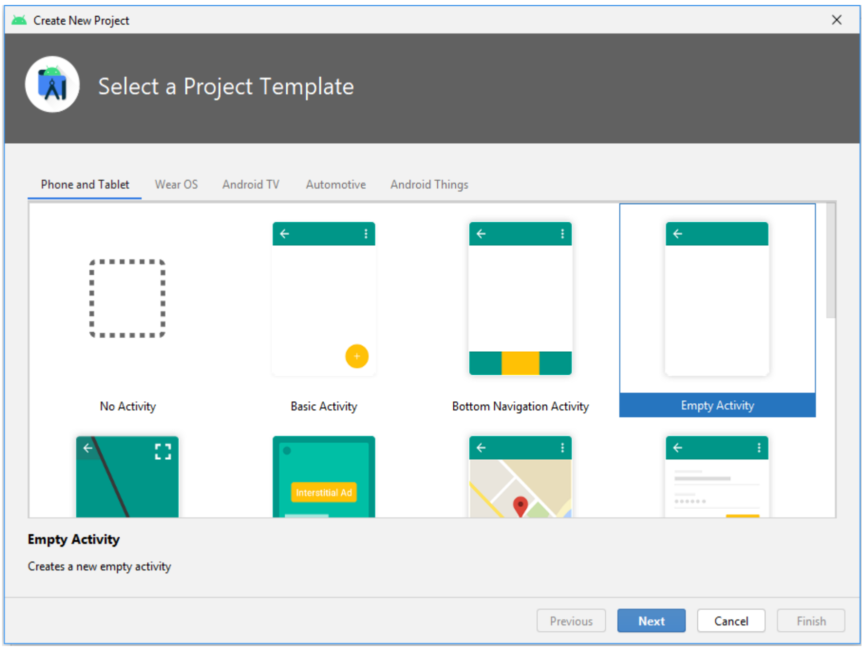
Create application to demonstrate the different android's activity state

a) Creating the State Change Example Project

i. Go to File -> New Project



ii. Choose empty activity template, then click Next

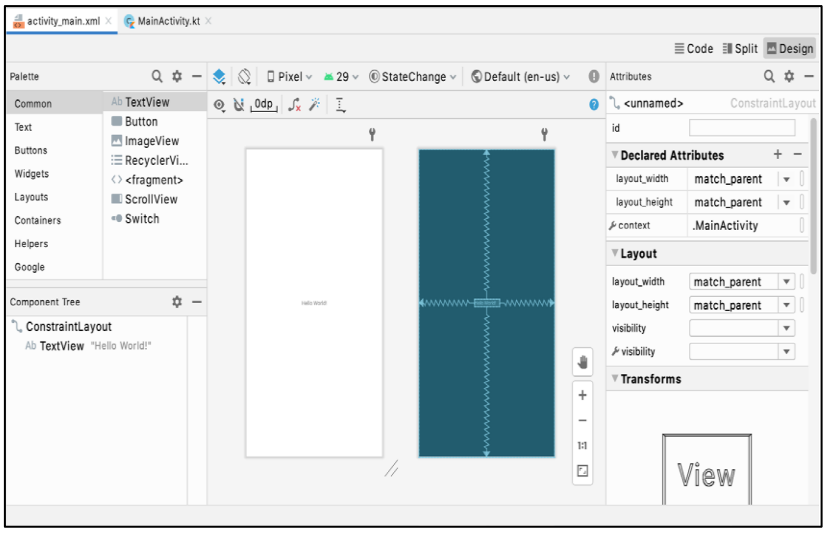


Empty Activity

Creates a new empty activity

iii. Enter StateChange into the Name field and specify com.ebookfrenzy.statechange as the package name.
Change the Minimum API level setting to API26: Android 8.0(Oreo) and the Language menu to Java.
Click Finish.

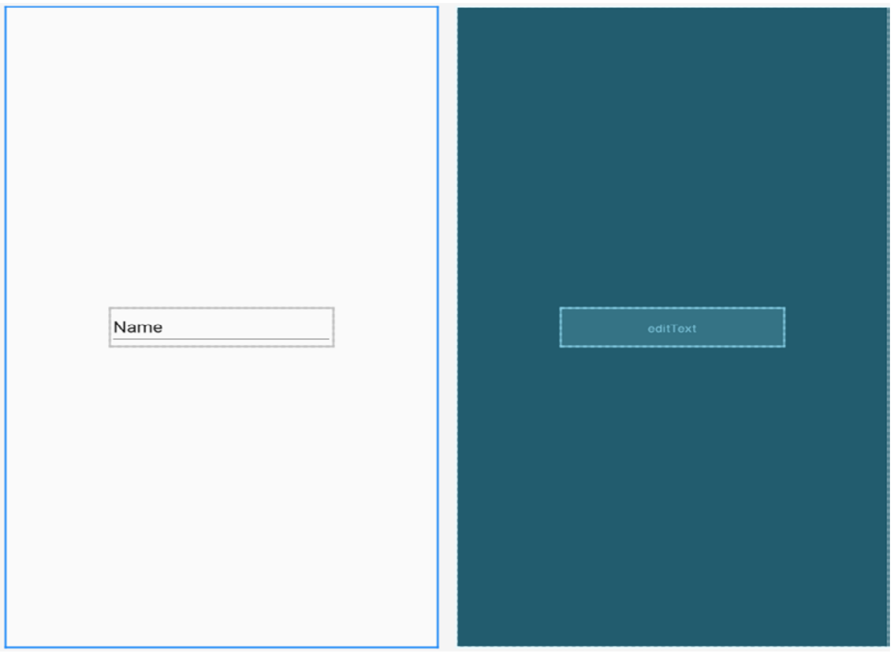
iv. Once the project has been created, open activity_main.xml.



b) Designing the User Interface

i. Instead of the “Hello World!” TextView currently present in the user interface design, the activity actually requires an EditText view. Select the TextView object in the Layout Editor canvas and press the Delete key on the keyboard to remove it from the design.

ii. From the Palette located on the left side of the Layout Editor, select the Text category and, from the list of text components, click and drag a Plain Text component over to the visual representation of the device screen. Move the component to the center of the display so that the center guidelines appear and drop it into place so that the layout resembles below figure.



iii. Select the EditText widget in the layout and locate the inputType entry within the Attributes tool window.
iv. Click on the flag icon to the left of the current setting to open the list of options and, within the list, switch off textPersonName and enable text before clicking on the Apply button. Remaining in the Attributes tool window, change the id of the view to editText and click on the Refactor button in the resulting dialog.
v. By default the EditText is displaying text which reads “Name”. Remaining within the Attributes panel, delete this from the text property field so that the view is blank within the layout.

c) Overriding the Activity Lifecycle Methods

i. Open MainActivity.java. So far the only lifecycle method overridden by the activity is the onCreate() method.

ii. Now we will add on another method(as highlighted) so that it outputs a diagnostic message in the Android Studio Logcat panel each time it executes. For this, we will use the Log class, which requires that we import android.util.Log and declare a tag that will enable us to filter these messages in the log output:

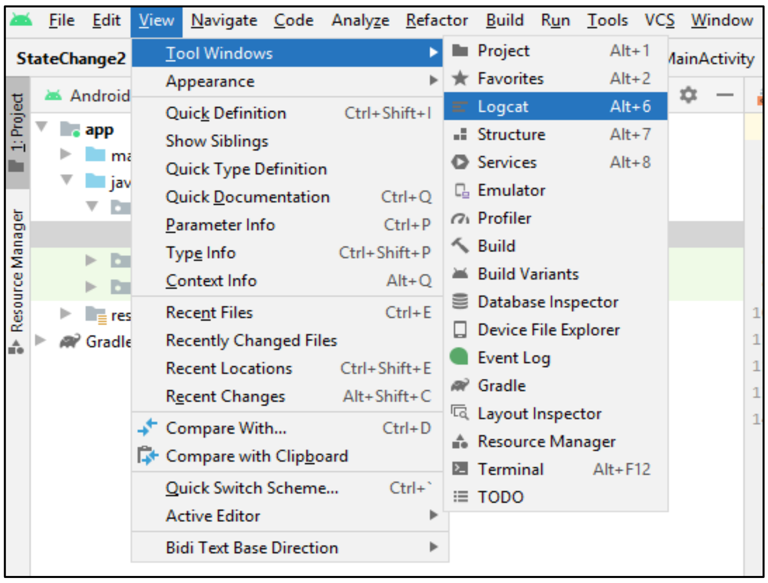
```
package com.ebookfrenzy.statechange;
.
.
import android.util.Log;
import androidx.annotation.NonNull;

public class MainActivity extends AppCompatActivity {

private static final String TAG = "StateChange";

@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      Log.i(TAG, "onCreate");
   }
}
```

iii. The next task is to override some more methods, with each one containing a corresponding log call.

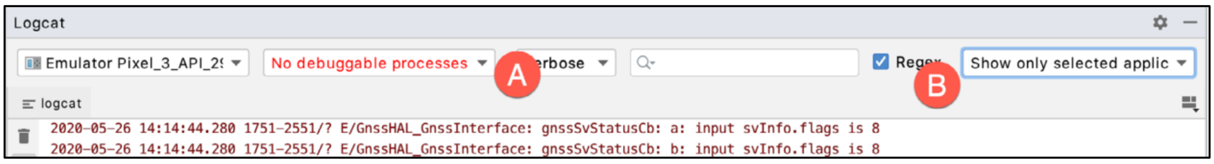
```
@Override
protected void onStart() {
super.onStart();
Log.i(TAG, "onStart");
}

@Override
protected void onResume() {
super.onResume();
Log.i(TAG, "onResume");
}
@Override
protected void onPause() {
super.onPause();
Log.i(TAG, "onPause");
}

@Override
protected void onStop() {
super.onStop();
Log.i(TAG, "onStop");
}

@Override
protected void onRestart() {
super.onRestart();
Log.i(TAG, "onRestart");
}

@Override
protected void onDestroy() {
super.onDestroy();
Log.i(TAG, "onDestroy");
}

@Override
protected void onSaveInstanceState(@NonNull Bundle outState)
{
super.onSaveInstanceState(outState);
Log.i(TAG, "onSaveInstanceState");  }

@Override
protected void onRestoreInstanceState(@NonNull Bundle
savedInstanceState) {
super.onRestoreInstanceState(savedInstanceState);
Log.i(TAG, "onRestoreInstanceState");                    }
```

d) Filtering the Logcat Panel

i. The purpose of the code added to the overridden methods in MainActivity.java is to output logging information to the Logcat tool window. So open the Logcat window by referring to the below figure.

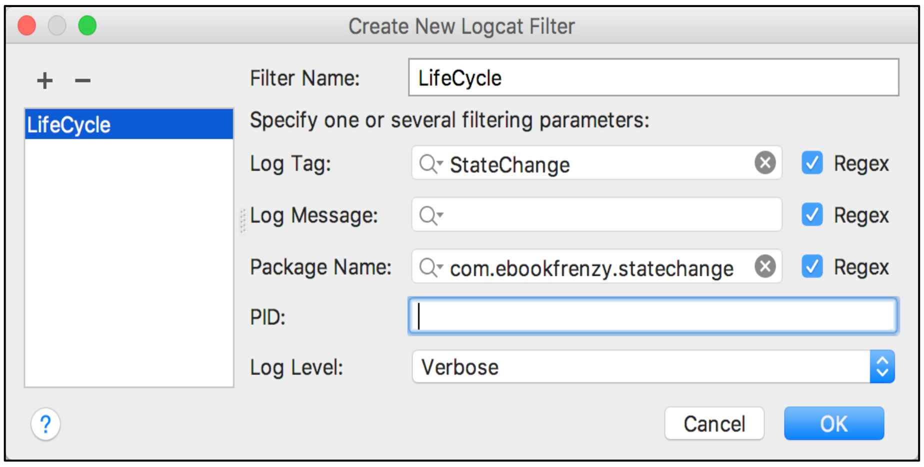


ii. From the filter menu(B), select the Edit Filter Configuration menu option. In the Create New Logcat Filter dialog name the filter Lifecycle and, in the Log Tag field, enter the Tag value declared in MainActivity.java
(in the previous code we created before this was StateChange).



```
2020-05-26 14:14:44.280 1751-2551/? E/GnssHAL_GnssInterface: gnssSvStatusCb: a: input svInfo.flags is 8
2020-05-26 14:14:44.280 1751-2551/? E/GnssHAL_GnssInterface: gnssSvStatusCb: b: input svInfo.flags is 8
```

iii. Enter the package identifier in the Package Name field and, when the changes are complete, click on the OK button to create the filter.



Filter Name: LifeCycle

Specify one or several filtering parameters:

Log Tag: StateChange ✓ Regex

Log Message: ✓ Regex

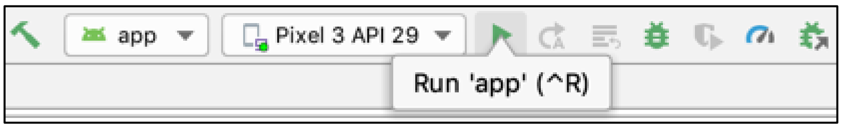
Package Name: com.ebookfrenzy.statechange ✓ Regex

PID:

Log Level: Verbose

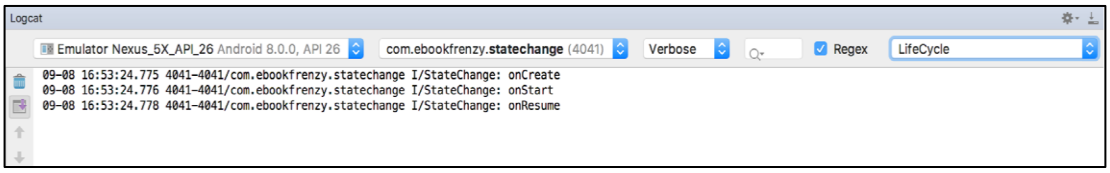
e) Running the Application

i. In order to run the project, make sure you already launch the emulator
ii. Click on the run button.



iii. After Android Studio has built the application and installed it on the device it should start up and be running in the foreground.

iv. A review of the Logcat panel should indicate which methods have so far been triggered.


Logcat
Emulator Nexus_5X_API_26 Android 8.0.0, API 26
com.ebookfrenzy.statechange (4041)
Verbose
Regex
LifeCycle

```
09-08 16:53:24.775 4041-4041/com.ebookfrenzy.statechange I/StateChange: onCreate
09-08 16:53:24.776 4041-4041/com.ebookfrenzy.statechange I/StateChange: onStart
09-08 16:53:24.778 4041-4041/com.ebookfrenzy.statechange I/StateChange: onResume
```

f) Experimenting with the Activity

i. With the diagnostics working, it is now time to exercise the application with a view to gaining an understanding of the activity lifecycle state changes. To begin with, consider the initial sequence of log events in the Logcat panel:

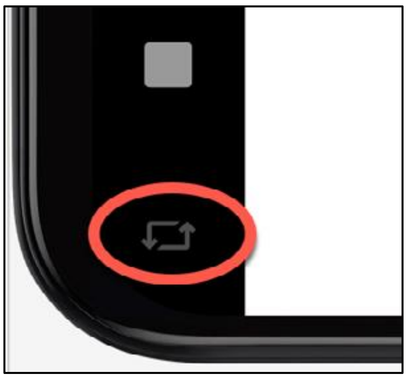
```
onCreate
onStart
onResume
```

ii. Tap on the Home icon in the bottom status bar on the device display and note the sequence of method calls reported in the log as follows:

```
onPause
onStop
onSaveInstanceState
```

iii. The destruction and recreation of an activity can be triggered by making a configuration change to the device, such as rotating from portrait to landscape. To see this in action, simply rotate the device while the StateChange application is in the foreground.

iv. When using the emulator, device rotation may be simulated using the rotation button located in the emulator toolbar. To complete the rotation, it will also be necessary to tap on the rotation button which appears in the toolbar of the device or emulator screen as shown in figure below:



v. The resulting sequence of method calls in the log should read as follows:

```
onPause
onStop
onSaveInstanceState
onDestroy
onCreate
onStart
onRestoreInstanceState
onResume
```

Clearly, the runtime system has given the activity an opportunity to save state before being destroyed and restarted.

g) Saving Dynamic State

An activity, as we have already learned, is given the opportunity to save dynamic state information via a call from the runtime system to the activity's implementation of the onSaveInstanceState() method. Passed through as an argument to the method is a reference to a Bundle object into which the method will need to store any dynamic data that needs to be saved. The Bundle object is then stored by the runtime system on behalf of the activity and subsequently passed through as an argument to the activity's onCreate() and onRestoreInstanceState() methods if and when they are called. The data can then be retrieved from the Bundle object within these methods and used to restore the state of the activity.

i. First, disable the the automatic saving of state for a user interface view in the XML layout file by setting the *android:saveEnabled* property to *false*.

ii. Edit the activity_main.xml file so that the entry for the view reads as follows:

```
<EditText
android:id="@+id/editText"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:ems="10"
android:inputType="text"
android:saveEnabled="false"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent" />
```

iii. After making the change, run the application, enter text and rotate the device to verify that the text is not saved and restored before proceeding.

h) Saving the State

i. Open MainActivity.java, modified *onSaveInstanceState()* method in the MainActivity.java file as follows(noting also the additional import directive for android.widget.EditText):

```
package com.ebookfrenzy.statechange;
..
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {
...
protected void onSaveInstanceState(@NonNull Bundle outState)
{
super.onSaveInstanceState(outState);
Log.i(TAG, "onSaveInstanceState");
final EditText editText =
findViewById(R.id.editText);
CharSequence userText = editText.getText();
outState.putCharSequence("savedText", userText);
}
```

i) Restoring the State

i. Still in MainActivity.java, add code to the onRestoreInstanceState() method to extract the saved state from the Bundle using the "savedText" key. We can then display the text on the editText component using the object's setText() method:

```
@Override
protected void onRestoreInstanceState(@NonNull Bundle
savedInstanceState) {
super.onRestoreInstanceState(savedInstanceState);
Log.i(TAG, "onRestoreInstanceState");
final EditText editText =findViewById(R.id.editText);
CharSequence userText =
savedInstanceState.getCharSequence("savedText");
editText.setText(userText);
}
```

j) Testing the Application

ii. Once again to build and run the StateChange application.

iii. Once running and in the foreground, touch the EditText component and enter some text before rotating the device to another orientation.

iv. Whereas the text changes were previously lost, the new text is retained within the editText component.

Exercise:

Comment out the *super.onSaveInstanceState()* and *super. onRestoreInstanceState()* calls from the two methods, re-launch the app, and see whether the text is still preserved after a device rotation or not.