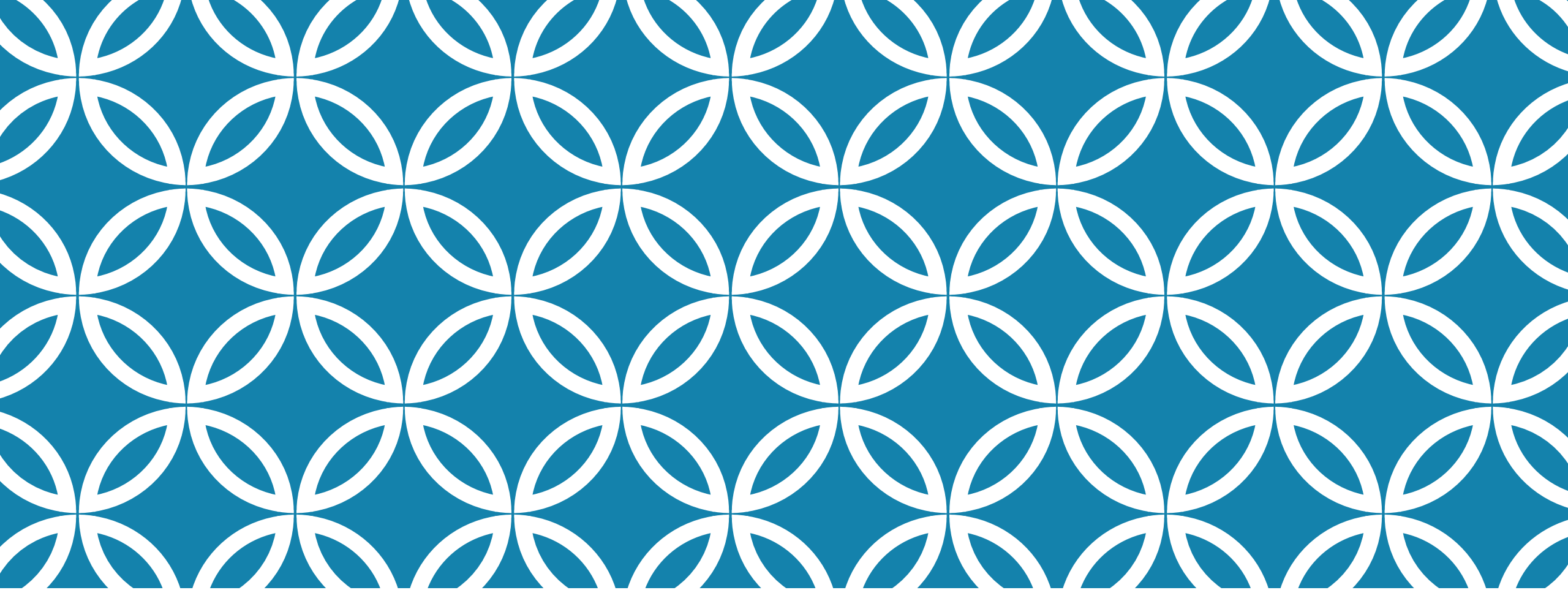


ANDROID ACTIVITIES

CS 175 Mobile Software Dev

by Dr. Angus Yeung

ACTIVITIES

Any typical Android app comprises multiple screens. In Android, each single screen is implemented as an **Activity**.

The screen is contained in a window of user interface. The window may fill the entire screen or may be smaller than the screen. In some cases, the window may float on top of other windows.

ACTIVITIES

Even though multiple activities may be grouped together to provide a coherent and seamless experience to users, each activity is independent of the others and operates in its own process.

All applications are composed of at least one Activity class and there is one Activity class that serves as the main activity.

ACTIVITIES

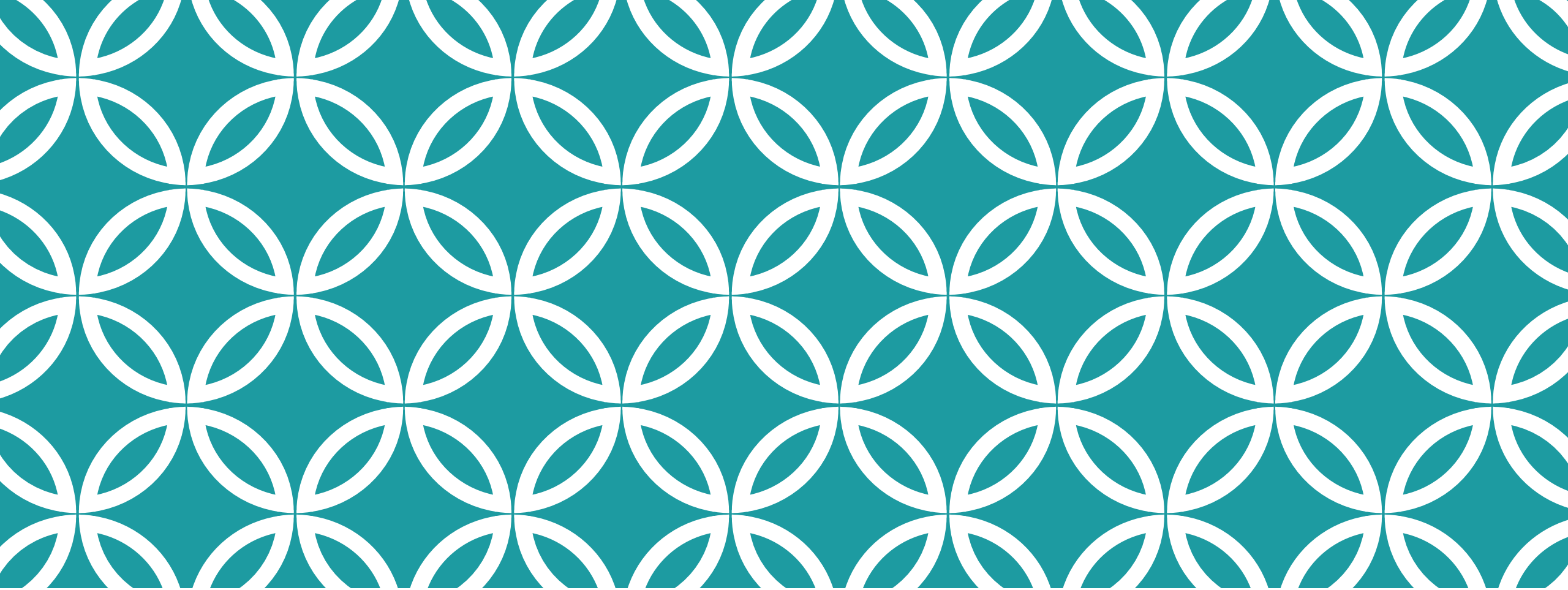
When an application is launched, one activity specified as the "main" activity will start. The activity serving as a starting point for Android system is specified in the manifest file using the <activity> element.

```
<manifest ... >
  <application ... >
      <activity android:name=".MyCarActivity" />
      ...
  </application ... >
  ...
</manifest >
```

ACTIVITIES

From the main activity, one can start one or more activity instances.

Each time a new activity starts, the previous activity is paused and its status is preserved by the Android system.



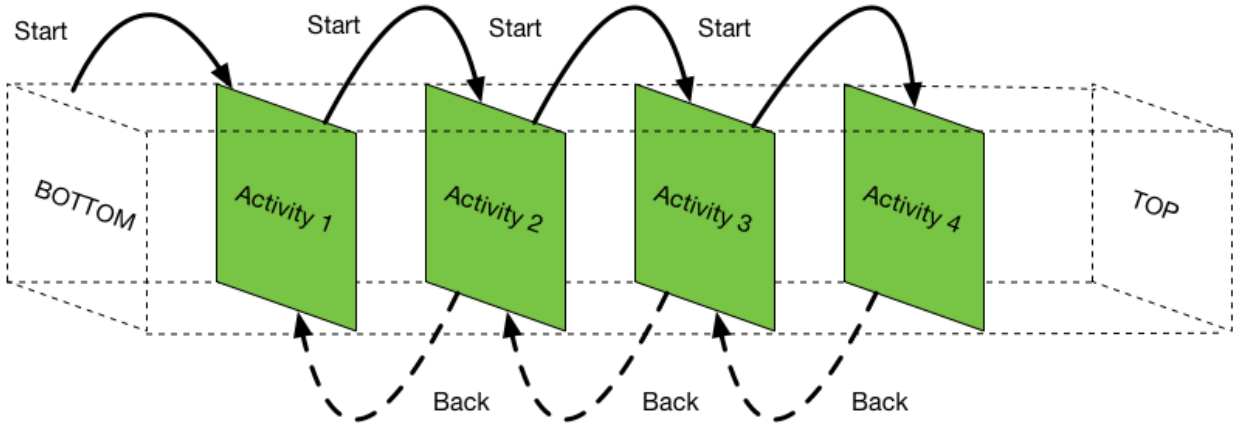
MANAGING ACTIVITIES

CS 175 Mobile Software Dev

by Dr. Angus Yeung

MANAGING ACTIVITIES

The interaction with main activity may trigger the start of another activity to perform different actions. The **switching** between different activities is accomplished through a kind of **LIFO stack** managed by Android system. So when the user is done with current activity and press the back button, the current activity will be popped from the stack and the previous activity will be brought back to the foreground.



MANAGING ACTIVITIES

Android uses an Intent as a messenger to facilitate communication between asynchronous processes.

An intent object is a passive data structure for the description of an operation to be performed.

For example, it can be used with startActivity to launch another activity, broadcastIntent to send it to any interested broadcast receivers, or startService/bindService to communicate with a background service.

MANAGING ACTIVITIES

You can start another activity from the current activity using the application context to call startActivity() method. The method takes a single parameter, an Intent.

```
startActivity(new Intent(getApplicationContext(),
MyFavoriteCarActivity.class));
```

MANAGING ACTIVITIES

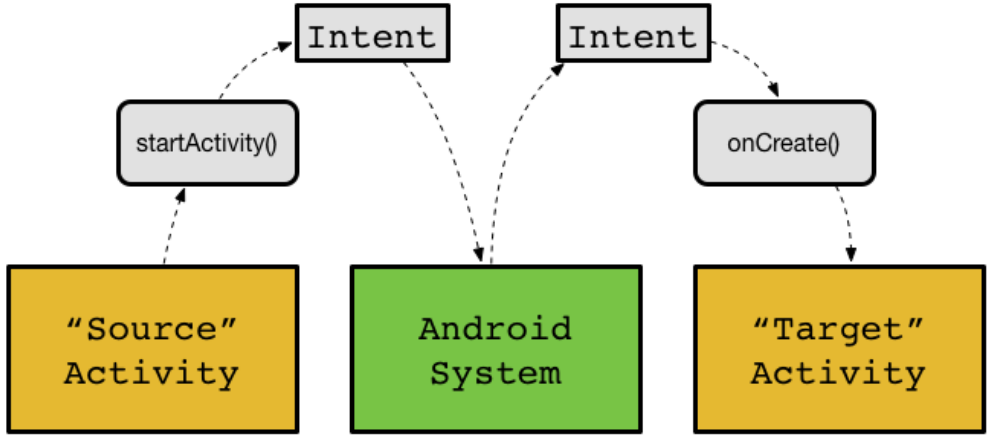
The **explicit intent** is created to launch a class by name. There is another way to launch an activity *implicitly* using **intent filter**. **Implicit intents** do not require a specific component by name, but instead declare a general action to perform.

An intent filter is registered in the AndroidManifest.xml file and it is used by activities, services, and broadcast receivers to specify the kind of intents they want to receive.

MANAGING ACTIVITIES

The illustration below shows how implicit intents work.

The “source” activity creates an Intent with an action description and passes it to startActivity(). The Android system searches for all components in apps that have declared an intent filter that matches with the Intent. When a match is found, Android system starts the “target” activity by invoking its onCreate() method and passing it the Intent received from “source” activity.



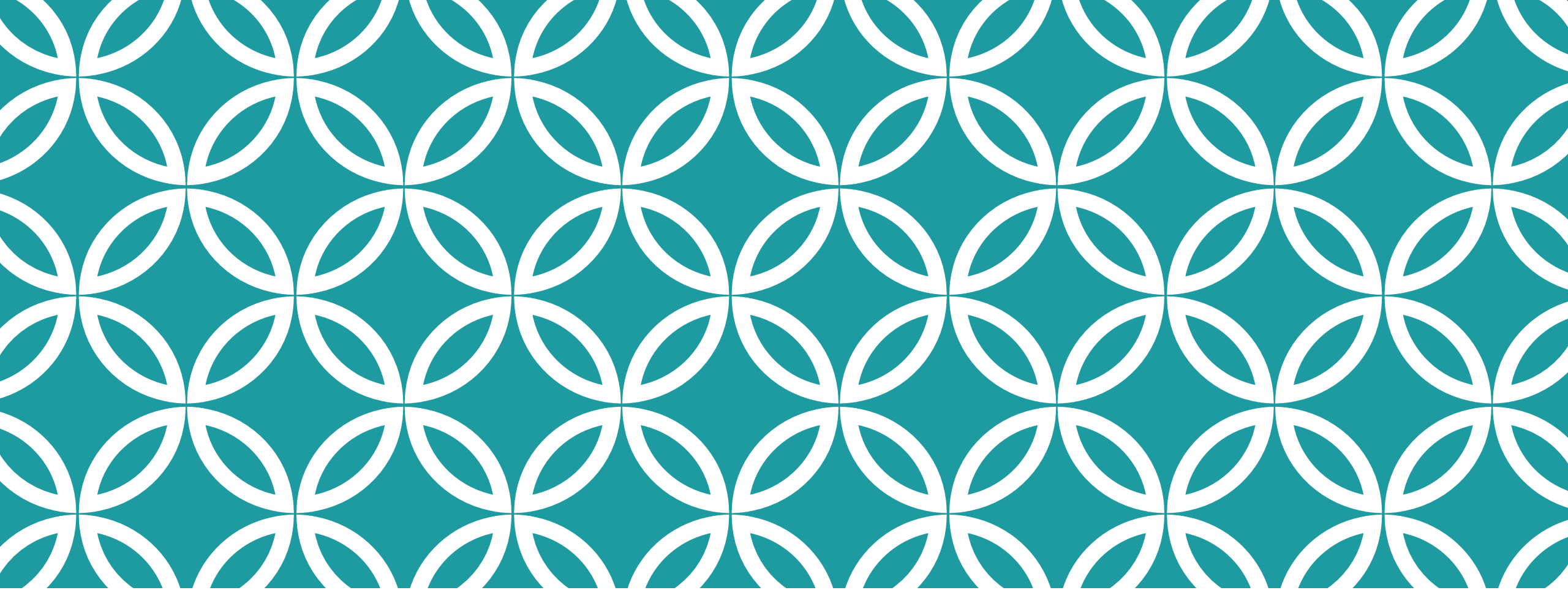
DEEP DIVE: ANDROID INTENT

API Documentation
https://developer.android.com/reference/android/content/Intent.html

https://developer.android.com/guide/components/intents-filters.html

- Use Cases:
  - Starting an activity: `startActivity()`
  - Starting a service: `startService()`
  - Delivering a broadcast: `sendBroadcast()`
- Attributes: action, data, extras,…
- Explicit Intents versus Implicit Intents
- Intent Filter (for intent matching)
- Main Entry of an App

- Constructors for Intent:
  - `Intent(String action)`
  - `Intent(String action, Uri uri)`
  - `Intent(Context packageContext, Class<?> cls)`
- Useful member functions:
  - `Intent putExtra(String name, …)`
  - `Bundle getExtras()`
  - `int getIntExtra(String name, int defaultValue)`
- Pending Intent (for foreign app)



LIFECYCLE OF ACTIVITIES

CS 175 Mobile Software Dev
by Dr. Angus Yeung

UNDERSTANDING ACTIVITY LIFECYCLE

An activity has four states: running, paused, stopped, and killed.

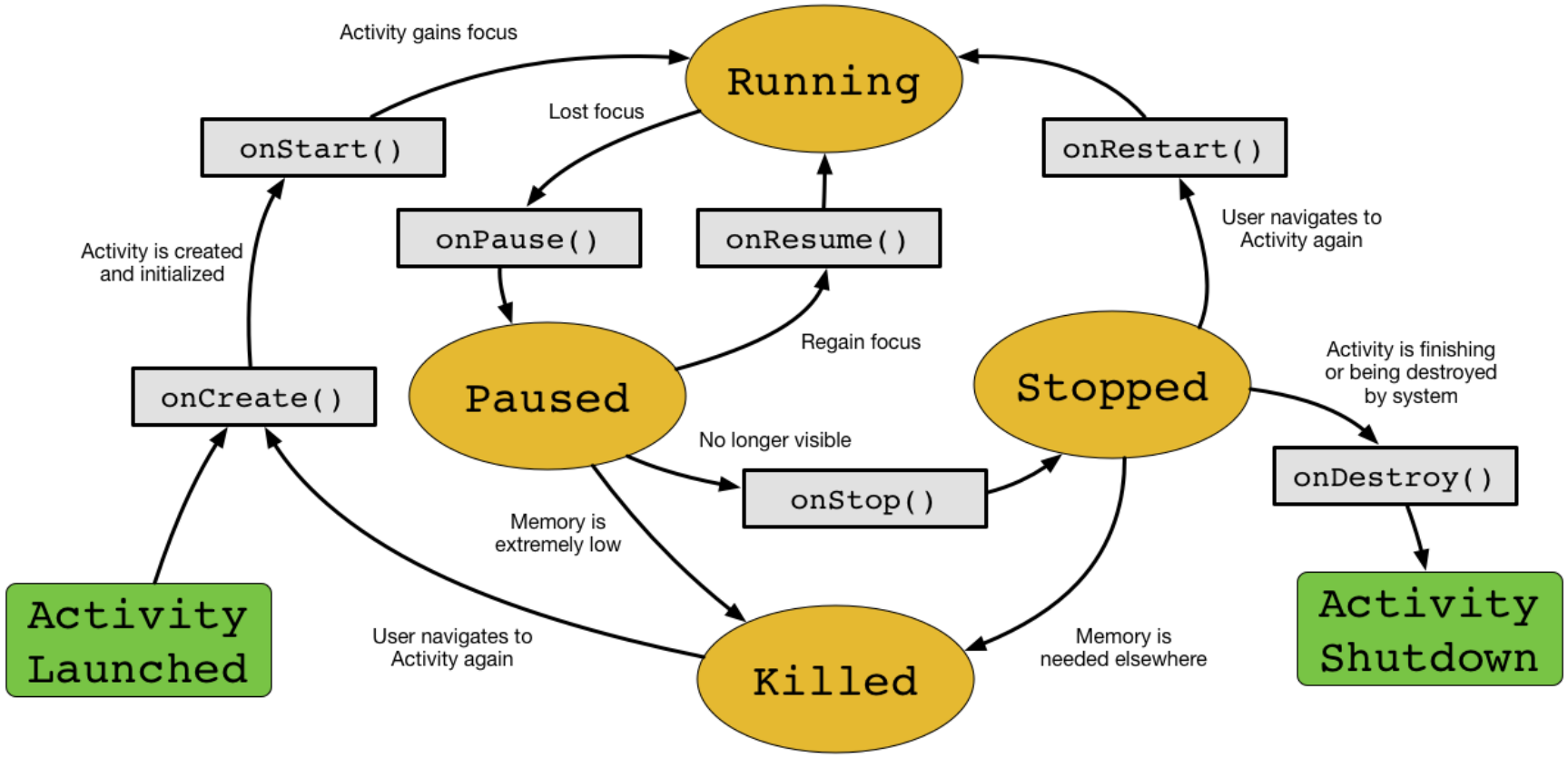
If an activity is actively shown in the foreground of the screen, it is ***running***. If the activity has lost focus but is still visible, it becomes ***paused*** but still alive.

A paused activity maintains all state and member information but can be killed by the system when the system memory becomes extremely low.

If the activity is no longer visible, it is ***stopped***. Like paused state, a stopped activity maintains all state and member information. A stopped activity will often be killed by the system when memory is needed elsewhere. When an activity is already in either paused or stopped state, the system may ask the process to finish or decide to kill the process.

If the activity is ***killed***, all information related to the activity will be removed from the memory. The activity needs to be created again when it is displayed in next time.

UNDERSTANDING ACTIVITY LIFECYCLE



UNDERSTANDING ACTIVITY LIFECYCLE

When the state of activity changes, different callback methods are invoked. All of these methods are hooks for you to override and add appropriate actions so you can define an activity's behavior when the activity changes state.

```
public class MyCarActivity extends ApplicationContext {
    protected void onCreate(Bundle savedInstanceState);
    protected void onStart();
    protected void onRestart();
    protected void onResume();
    protected void onPause();
    protected void onStop();
    protected void onDestroy();
}
```

UNDERSTANDING ACTIVITY LIFECYCLE

The activity receives dynamic instance state in Bundle when the onCreate() method is called.

To save the persistent data of activity, you can save them in the onPause() method.

As you can see from the lifecycle diagram, the callback method onPause() will be invoked before the activity goes to either stopped or killed state.

By putting your persistent data code in onPause(), you can make sure that these persistent data can be retrieved when the activity is re-created next time, that is when onCreate() is invoked again.

MONITORING ACTIVITY STATES

Since you can override the methods for an activity's lifecycle, it becomes very easy to monitor the activity's lifecycle states in a simple Hello World program.

<1> CREATE A NEW PROJECT

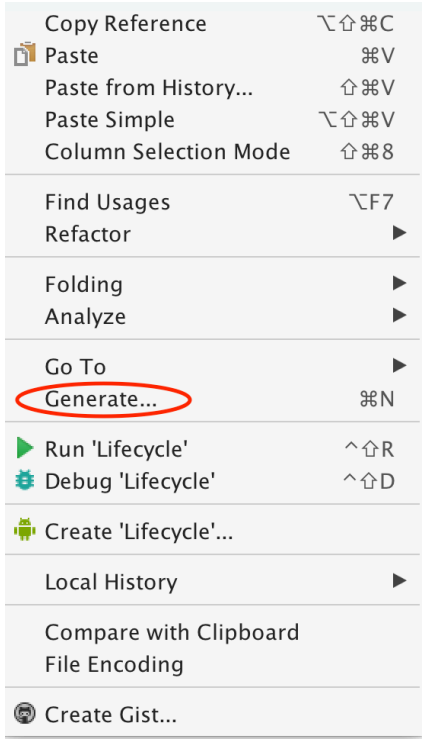
In Android Studio, create a new project with blank activity. Open the main Java file, you will see the onCreate() method generated for you already [LINE 10].

Add a Toast.makeText().show() call to pop up a toast message when onCreate() is invoked.

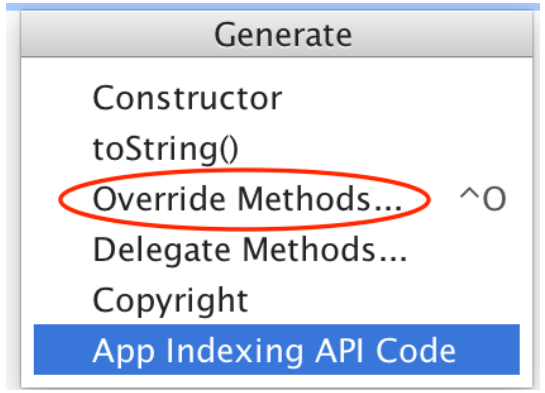
```
8
9       @Override
10      protected void onCreate(Bundle savedInstanceState) {
11          super.onCreate(savedInstanceState);
12          setContentView(R.layout.activity_lifecycle);
13          Toast.makeText(this, "onCreate()", Toast.LENGTH_SHORT).show();
14      }
15
```

<2> OVERRIDE LIFECYCLE METHODS

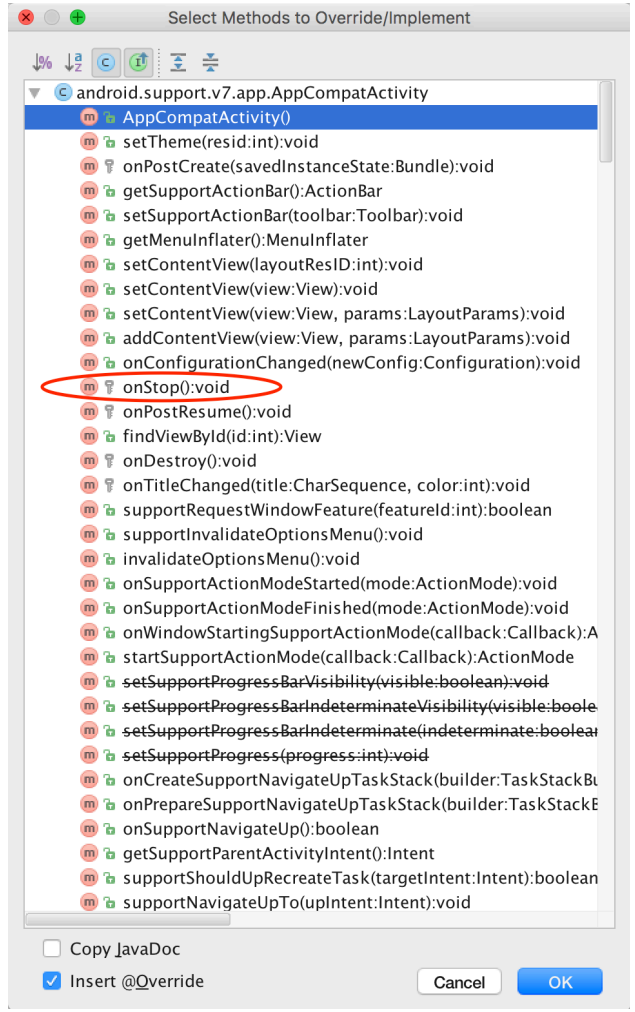
2.1 Use the context menu, by right-clicking your mouse, to generate all other methods:



2.2 Select **Override Methods...** in the **Generate** menu:



2.3 Select one of the lifecycle methods: `onDestroy()`, `onStart()`, `onStop()`, `onResume()`, `onPause()`, and `onRestart()`.



<3> ADD TOAST MESSAGES

Similar to what you have done in onCreate(), add a toast message call to each of the generated methods.

```
19          Toast.makeText(this, "onDestroy()", Toast.LENGTH_SHORT).show();
26          Toast.makeText(this, "onStart()", Toast.LENGTH_SHORT).show();
32          Toast.makeText(this, "onStop()", Toast.LENGTH_SHORT).show();
38          Toast.makeText(this, "onPause()", Toast.LENGTH_SHORT).show();
44          Toast.makeText(this, "onResume()", Toast.LENGTH_SHORT).show();
50          Toast.makeText(this, "onRestart()", Toast.LENGTH_SHORT).show();
```

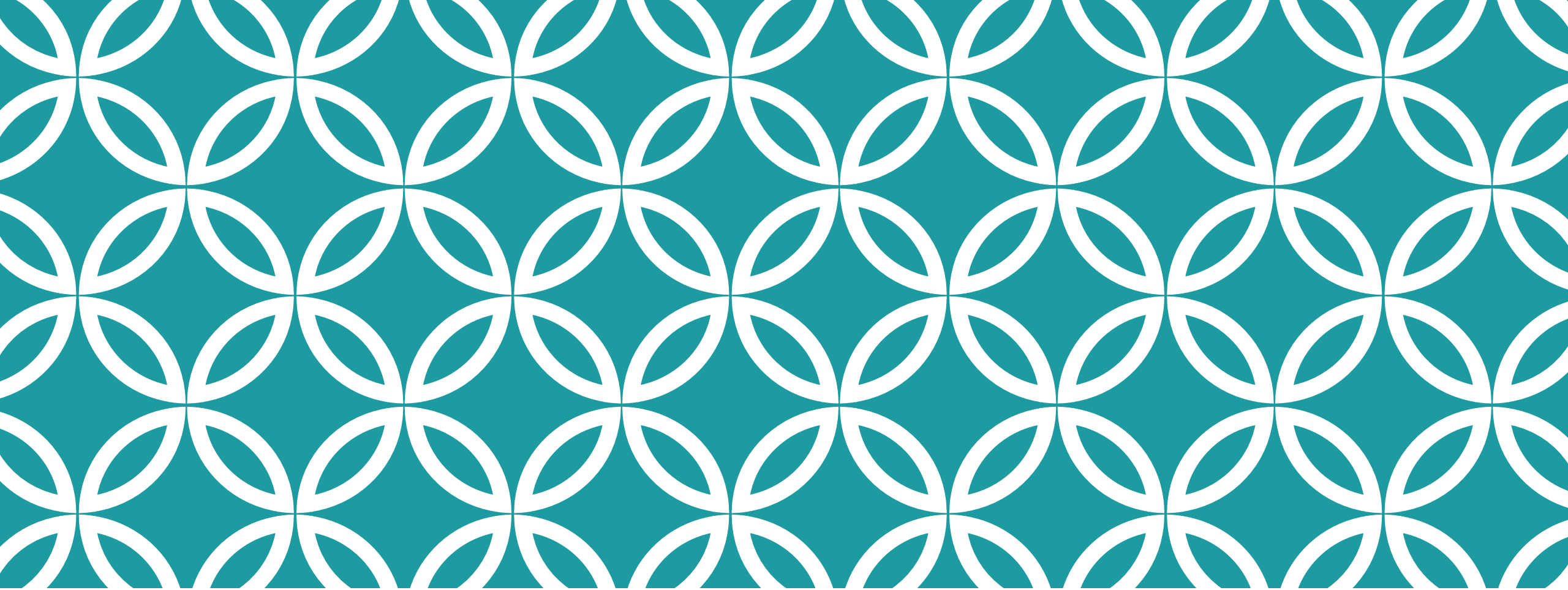
<4> RUN YOUR APPLICATION

Run your application and make observation of the toast messages. You should see the sequential toast messages: onCreate(), onStart(), and onResume().

Press the back button on your Android device and the current activity will exit. You should see the sequential toast messages: onPause(), onStop(), and onDestroy().

LISTING OF LIFECYCLE.JAVA

```
1   package com.wearable.myactivitylifecycle;
2
3   import android.support.v7.app.AppCompatActivity;
4   import android.os.Bundle;
5   import android.widget.Toast;
6
7   public class Lifecycle extends AppCompatActivity {
8
9       @Override
10      protected void onCreate(Bundle savedInstanceState) {
11          super.onCreate(savedInstanceState);
12          setContentView(R.layout.activity_lifecycle);
13          Toast.makeText(this, "onCreate()", Toast.LENGTH_SHORT).show();
14      }
15
16      @Override
17      protected void onDestroy() {
18          super.onDestroy();
19          Toast.makeText(this, "onDestroy()", Toast.LENGTH_SHORT).show();
20      }
21
22
23      @Override
24      protected void onStart() {
25          super.onStart();
26          Toast.makeText(this, "onStart()", Toast.LENGTH_SHORT).show();
27      }
28
29      @Override
30      protected void onStop() {
31          super.onStop();
32          Toast.makeText(this, "onStop()", Toast.LENGTH_SHORT).show();
33      }
34
35      @Override
36      protected void onPause() {
37          super.onPause();
38          Toast.makeText(this, "onPause()", Toast.LENGTH_SHORT).show();
39      }
40
41      @Override
42      protected void onResume() {
43          super.onResume();
44          Toast.makeText(this, "onResume()", Toast.LENGTH_SHORT).show();
45      }
46
47      @Override
48      protected void onRestart() {
49          super.onRestart();
50          Toast.makeText(this, "onRestart()", Toast.LENGTH_SHORT).show();
51      }
52
53  }
54
```



SAVING AND RESTORING STATES

CS 175 Mobile Software Dev
by Dr. Angus Yeung

RETAINING THE STATES OF AN ACTIVITY

When an activity is paused or stopped, the state of the activity is retained

When the system destroys an activity in order to recover memory, the memory for that activity object is also destroyed.

USE (KEY, VALUE) TO STORE THE STATE

```
1  static final String CHESS_LEVEL = "playerLevel";
2  private int mPlayerLevel;
3  ...
4  @Override
5  public void onSaveInstanceState(Bundle savedInstanceState) {
6      savedInstanceState.putInt(STATE_LEVEL, mPlayerLevel);
7      super.onSaveInstanceState(savedInstanceState);
8  }
```

A BUNDLE

A Bundle is a container for the activity state information that can be saved. The following is an incomplete list of these methods:

putChar()

putString()

putBoolean()

putByte()

putFloat()

putLong()

putShort()

putParcelable()

RECOVERING THE SAVED STATE

Recover the saved state from the Bundle that the system passes to the activity.

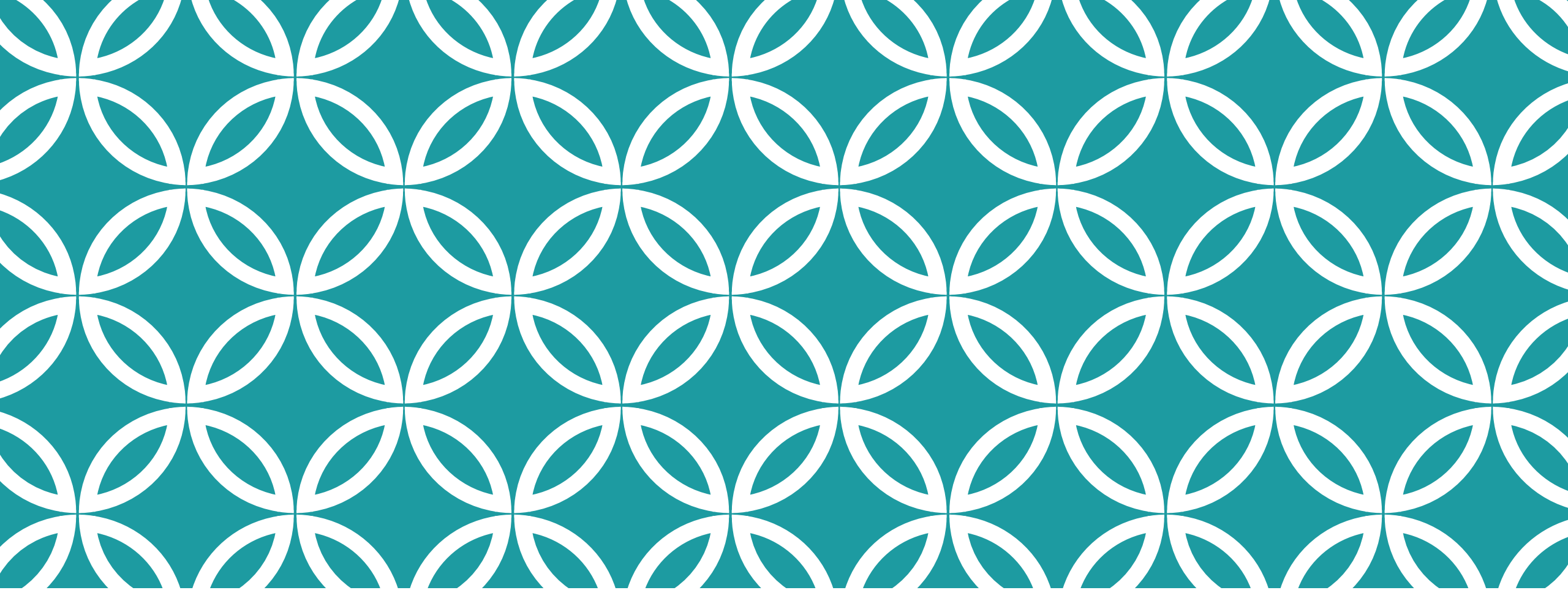
`onCreate()` and `onRestoreInstanceState()` callback methods receive the same Bundle object that contains the instance state information.

Check whether the activity's state (stored in the Bundle object) is null before you attempt to read it.

If it is null, then the system is creating a new instance of the Activity class, instead of restoring a previous one that was destroyed.

USE KEY TO RETRIEVE VALUE

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4
5      if (savedInstanceState != null) {
6          mPlayerLevel = savedInstanceState.getInt(CHESS_LEVEL);
7      }
8  }
```



FRAGMENTS

CS 175 Mobile Software Dev

by Dr. Angus Yeung

USING FRAGMENTS

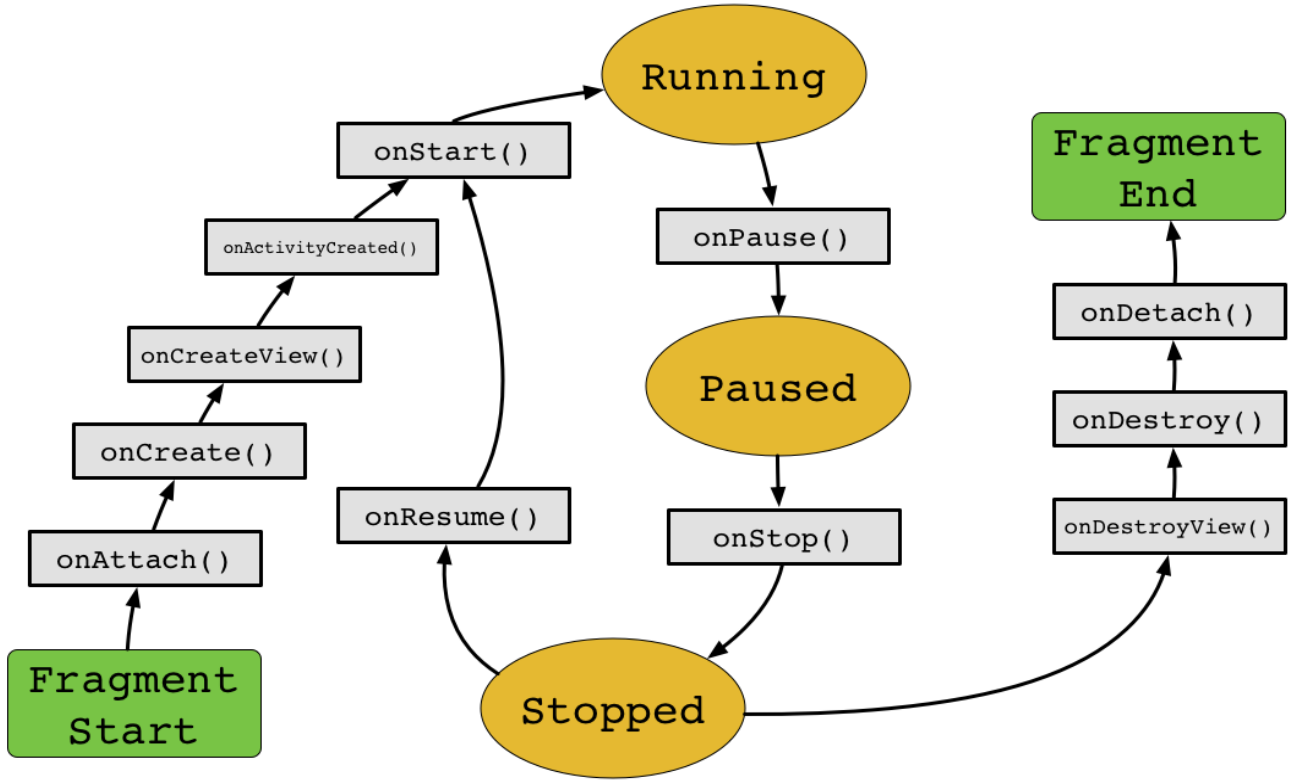
You can think of Fragment as a kind of sub-activity. Each fragment is a piece of user interface or behavior that is **contained** in an activity.

While an activity is running, you may add or remove fragments in an activity.

You can also combine many fragments in a single activity to create an user interface with multiple planes, or share a single fragment among multiple activities.

USING FRAGMENTS

A fragment has its own life cycle callbacks. Since fragment is contained in an activity, fragment life cycle is closely related to the life cycle of its host activity.



USING FRAGMENTS

Before you can use a fragment, it needs to be attached to an activity.

This is the time when **onAttach()** is called, followed by **onCreate()** when the fragment is created, and **onCreateView()** when the user interface for the fragment is drawn for the first time.

Now the fragment is ready for its host activity.

USING FRAGMENTS

You can use the callback method **OnActivityCreated()** to get a signal when the host activity is created.

The **onStart()** method is called once the fragment gets visible and onResume() is called when it becomes active.

When the user is leaving the fragment, **onPause()** is called. Like what we have recommended in Activity, the onPause() method is the best place for us to persist the changes the user has made.

USING FRAGMENTS

The **onStop()** method is called when the fragment is about to stop, followed by onDestroyView() when fragment view is destroyed, and onDestroy() when Android does the final clean up to the fragment object.

The **onDestroy()** method is not guaranteed to be called by Android, so normally you should not override this method in your fragment implementation.

CREATING FRAGMENTS

In this sample, you'll be shown how to implement two simple fragments.

If the Android device is in **landscape** mode, the landscape fragment will be used.

If you rotates the device by 90 degree, the **portrait** fragment will be used.

<1> CREATE A NEW PROJECT

In Android Studio, create a new blank activity. By default, the onCreate() method will be generated.

```
10   public class MainActivity extends AppCompatActivity {
11
12       @Override
13       protected void onCreate(Bundle savedInstanceState) {
14           super.onCreate(savedInstanceState);
15           setContentView(R.layout.activity_main);
16    . . .
30       }
31
32   }
```

<2.1> ADD NEW FRAGMENTS

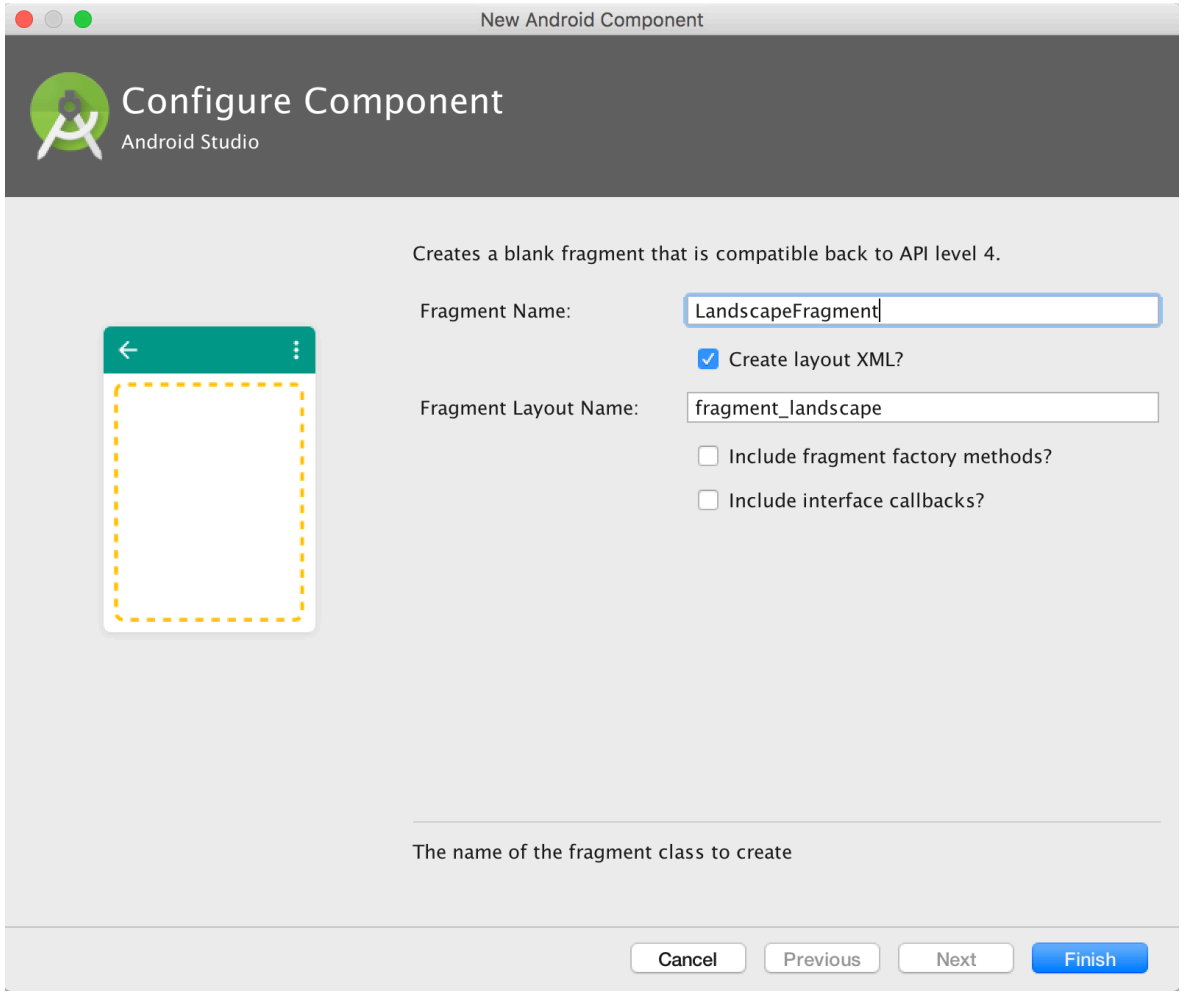
Use **File -> New -> Fragment -> Fragment (blank)** to create a new fragment.

A new dialog will pop up for you to configure the new fragment.

Put down the **Fragment Name** as LandscapeFragment.

Also uncheck the options "**Include fragment factory methods?**" and "**Include interface callbacks?**" if they are checked by default.

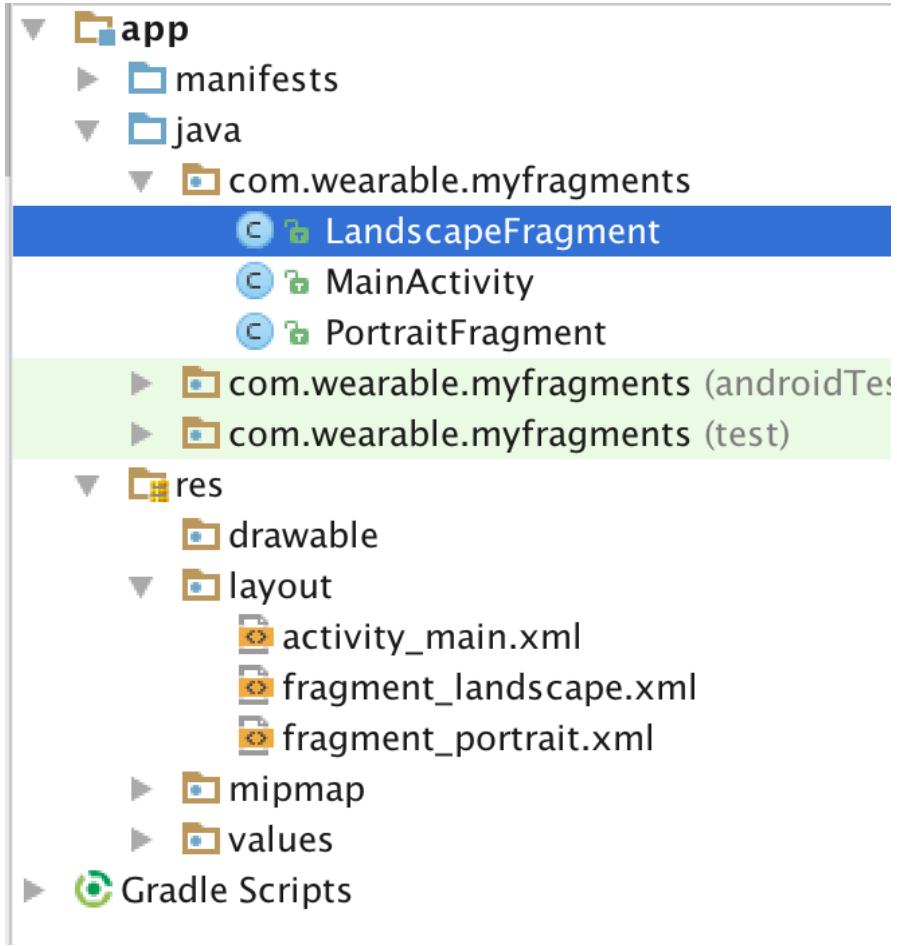
Click on **Finish.**



<2.2> ADD NEW FRAGMENTS

The new LandscapeFragment.java and layout file fragment_landscape.xml are generated.

Repeat the above steps for **Portrait Fragment.**



<3.1> MODIFY EACH FRAGMENT'S LAYOUT FILE

Modify the TextView to display the landscape_text string [LINE 11] with custom text size,30dp[LINE 12] and background color #FFBBBB [LINE 13].

```
1    <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
2                 xmlns:tools="http://schemas.android.com/tools"
3                 android:layout_width="match_parent"
4                 android:layout_height="match_parent"
5                 tools:context="com.wearable.myfragments.LandscapeFragment">
6
7        <!-- TODO: Update blank fragment layout -->
8        <TextView
9            android:layout_width="match_parent"
10           android:layout_height="match_parent"
11           android:text="@string/landscape_text"
12           android:textSize="30dp"
13           android:background="#FFBBBB"/>
14
15   </FrameLayout>
16
```

<3.2> MODIFY EACH FRAGMENT'S LAYOUT FILE

Repeat the same modification for the layout file for portrait fragment fragment_portrait.xml:

```
1    <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
2                 xmlns:tools="http://schemas.android.com/tools"
3                 android:layout_width="match_parent"
4                 android:layout_height="match_parent"
5                 tools:context="com.wearable.myfragments.PortraitFragment">
6
7        <!-- TODO: Update blank fragment layout -->
8        <TextView
9            android:layout_width="match_parent"
10           android:layout_height="match_parent"
11           android:text="@string/portrait_text"
12           android:textSize="30dp"
13           android:background="#BBBBFF"/>
14
15   </FrameLayout>
16
```

<3.3> MODIFY EACH FRAGMENT'S LAYOUT FILE

In the strings.xml resource file, add landscape_text [LINE 3] and portrait_text [LINE 4] strings.

```
1    <resources>
2        <string name="app_name">MyFragments</string>
3        <string name="landscape_text">This is the landscape mode.</string>
4        <string name="portrait_text">This is the portrait mode.</string>
5    </resources>
6
```

<4> MODIFY MAIN ACTIVITY'S LAYOUT

An activity can include each fragment using the <fragment> tag in its layout file.

In activity_main.xml, use two <fragment> tags to include both LandscapeFragment [LINE 9] and PortraitFragment [LINE 16].

You'll be able to configure how each fragment should be displayed in the main activity's Java source code.

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3                 xmlns:tools="http://schemas.android.com/tools"
4                 android:layout_width="fill_parent"
5                 android:layout_height="fill_parent"
6                 android:orientation="horizontal">
7
8       <fragment
9           android:name="com.wearable.myfragments.LandscapeFragment"
10          android:id="@+id/landscape_fragment"
11          android:layout_weight="1"
12          android:layout_width="0dp"
13          android:layout_height="match_parent" />
14
15      <fragment
16          android:name="com.wearable.myfragments.PortraitFragment"
17          android:id="@+id/portrait_fragment"
18          android:layout_weight="2"
19          android:layout_width="0dp"
20          android:layout_height="match_parent" />
21  </LinearLayout>
22
```

<5.1> ADD CODE TO DISPLAY FRAGMENTS

In the MainActivity::onCreate() method, get a handle to FragmentTransaction [LINE 19] using FragmentManager [LINE 18]. Call beginTransaction() [LINE 20] to start a series of edit operations on the Fragments associated with this FragmentManager.

You'll use the device's orientation configuration to decide which fragment to display. Retrieve the system's configuration [LINE 17] and check for the device's orientation.

<5.2> ADD CODE TO DISPLAY FRAGMENTS

If the configuration is in landscape orientation [LINE 22], use FragmentTransaction::replace() to invoke LandscapeFragment [LINE 24].

Otherwise, use FragmentTransaction::replace() to invoke PortraitFragment [LINE 27].

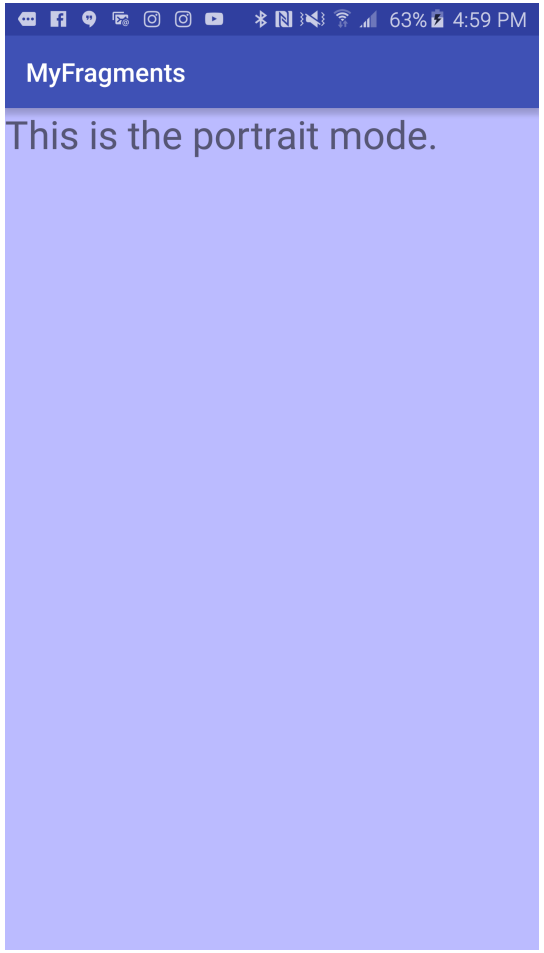
After all modifications are done, commit the changes using commit() [LINE 29].

<5.3> ADD CODE TO DISPLAY FRAGMENTS

```
12      @Override
13      protected void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          setContentView(R.layout.activity_main);
16
17          Configuration config = getResources().getConfiguration();
18          FragmentManager fragmentManager = getFragmentManager();
19          FragmentTransaction fragmentTransaction =
20                  fragmentManager.beginTransaction();
21
22          if (config.orientation == Configuration.ORIENTATION_LANDSCAPE) {
23              LandscapeFragment landscapeFragment = new LandscapeFragment();
24              fragmentTransaction.replace(android.R.id.content, landscapeFragment);
25          } else {
26              PortraitFragment portraitFragment = new PortraitFragment();
27              fragmentTransaction.replace(android.R.id.content, portraitFragment);
28          }
29          fragmentTransaction.commit();
30      }
```

<6.1> RUN THE MYFRAGMENTS APP

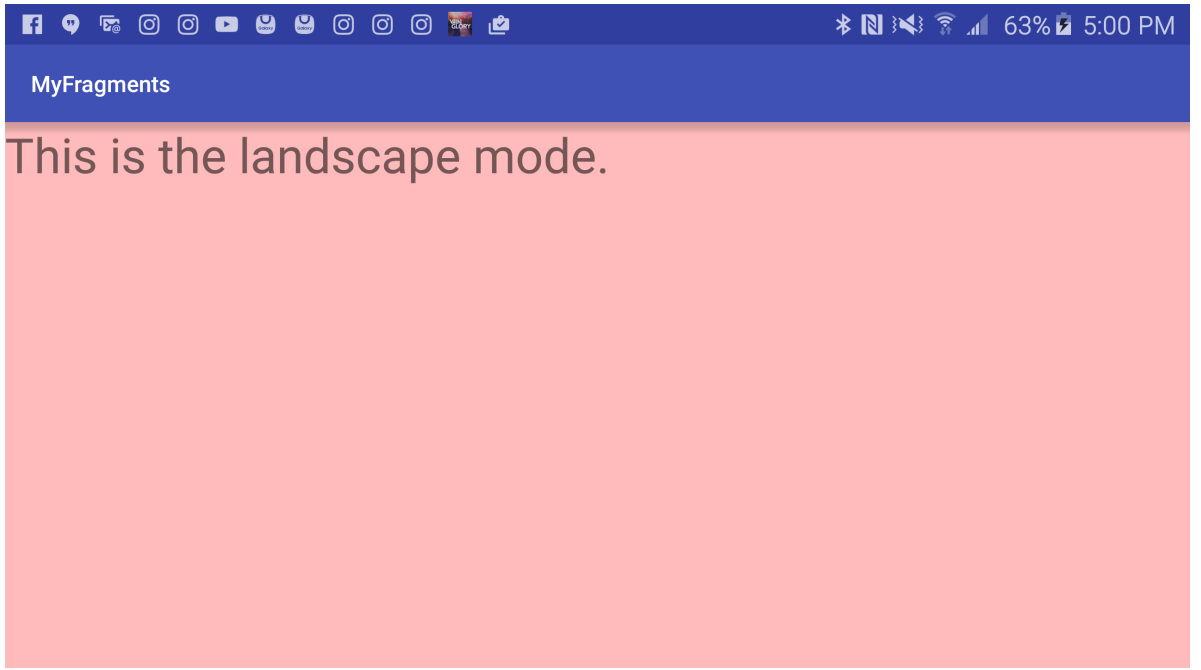
When you run your application in an Android device, you'll see the portrait fragment displayed in the screen.



<6.2> RUN THE MYFRAGMENTS APP

Rotate your device by 90 degree. If you are using AVD, use Left CTRL + F11 key combo to perform the rotation.

Now you'll see the landscape fragment displayed in the screen.



MAINACTIVITY.JAVA

```
1   package com.wearable.myfragments;
2
3   import android.app.Fragment;
4   import android.app.FragmentManager;
5   import android.app.FragmentTransaction;
6   import android.content.res.Configuration;
7   import android.support.v7.app.AppCompatActivity;
8   import android.os.Bundle;
9
10  public class MainActivity extends AppCompatActivity {
11
12      @Override
13      protected void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          setContentView(R.layout.activity_main);
16
17          Configuration config = getResources().getConfiguration();
18          FragmentManager fragmentManager = getFragmentManager();
19          FragmentTransaction fragmentTransaction =
20                  fragmentManager.beginTransaction();
21
22          if (config.orientation == Configuration.ORIENTATION_LANDSCAPE) {
23              LandscapeFragment landscapeFragment = new LandscapeFragment();
24              fragmentTransaction.replace(android.R.id.content, landscapeFragment);
25          } else {
26              PortraitFragment portraitFragment = new PortraitFragment();
27              fragmentTransaction.replace(android.R.id.content, portraitFragment);
28          }
29          fragmentTransaction.commit();
30      }
31
32  }
33
```

LANDSCAPEFRAGMENT.JAVA

```
1   package com.wearable.myfragments;
2
3
4   import android.os.Bundle;
5   import android.app.Fragment;
6   import android.view.LayoutInflater;
7   import android.view.View;
8   import android.view.ViewGroup;
9
10
11  /**
12   * A simple {@link Fragment} subclass.
13   */
14  public class LandscapeFragment extends Fragment {
15
16
17      public LandscapeFragment() {
18          // Required empty public constructor
19      }
20
21
22      @Override
23      public View onCreateView(LayoutInflater inflater, ViewGroup container,
24                               Bundle savedInstanceState) {
25          // Inflate the layout for this fragment
26          return inflater.inflate(R.layout.fragment_landscape, container, false);
27      }
28
29  }
30
```

PORTRAITFRAGMENT.JAVA

```
1   package com.wearable.myfragments;
2
3
4   import android.os.Bundle;
5   import android.app.Fragment;
6   import android.view.LayoutInflater;
7   import android.view.View;
8   import android.view.ViewGroup;
9
10
11  /**
12   * A simple {@link Fragment} subclass.
13   */
14  public class PortraitFragment extends Fragment {
15
16
17      public PortraitFragment() {
18          // Required empty public constructor
19      }
20
21
22      @Override
23      public View onCreateView(LayoutInflater inflater, ViewGroup container,
24                               Bundle savedInstanceState) {
25          // Inflate the layout for this fragment
26          return inflater.inflate(R.layout.fragment_portrait, container, false);
27      }
28
29  }
30
```

ACTIVITY_MAIN.XML

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3                 xmlns:tools="http://schemas.android.com/tools"
4                 android:layout_width="fill_parent"
5                 android:layout_height="fill_parent"
6                 android:orientation="horizontal">
7
8       <fragment
9           android:name="com.wearable.myfragments.LandscapeFragment"
10          android:id="@+id/landscape_fragment"
11          android:layout_weight="1"
12          android:layout_width="0dp"
13          android:layout_height="match_parent" />
14
15      <fragment
16          android:name="com.wearable.myfragments.PortraitFragment"
17          android:id="@+id/portrait_fragment"
18          android:layout_weight="2"
19          android:layout_width="0dp"
20          android:layout_height="match_parent" />
21  </LinearLayout>
22
```

FRAGMENT_LANDSCAPE.XML

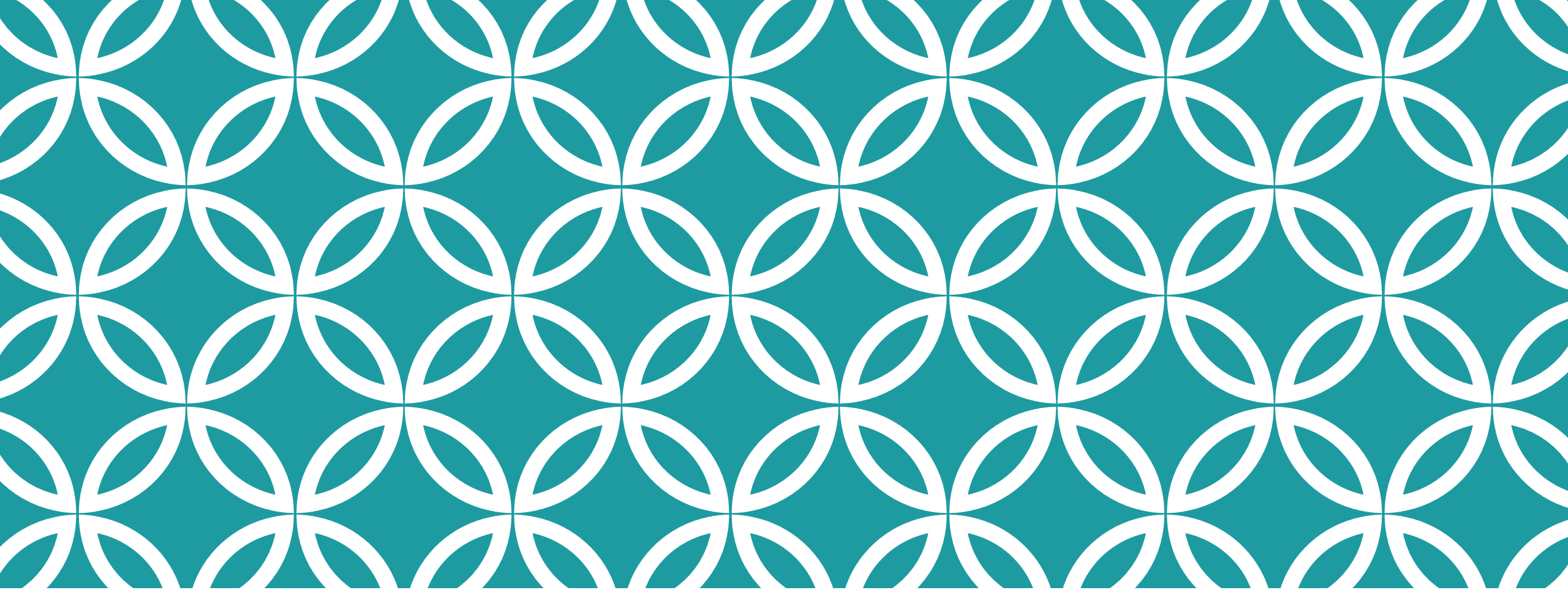
```
1    <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
2                 xmlns:tools="http://schemas.android.com/tools"
3                 android:layout_width="match_parent"
4                 android:layout_height="match_parent"
5                 tools:context="com.wearable.myfragments.LandscapeFragment">
6
7        <!-- TODO: Update blank fragment layout -->
8        <TextView
9            android:layout_width="match_parent"
10           android:layout_height="match_parent"
11           android:text="@string/landscape_text"
12           android:textSize="30dp"
13           android:background="#FFBBBB"/>
14
15   </FrameLayout>
16
```

FRAGMENT_PORTRAIT.XML

```
1    <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
2                 xmlns:tools="http://schemas.android.com/tools"
3                 android:layout_width="match_parent"
4                 android:layout_height="match_parent"
5                 tools:context="com.wearable.myfragments.PortraitFragment">
6
7        <!-- TODO: Update blank fragment layout -->
8        <TextView
9            android:layout_width="match_parent"
10           android:layout_height="match_parent"
11           android:text="@string/portrait_text"
12           android:textSize="30dp"
13           android:background="#BBBBFF"/>
14
15   </FrameLayout>
16
```

STRINGS.XML

```
1    <resources>
2        <string name="app_name">MyFragments</string>
3        <string name="landscape_text">This is the landscape mode.</string>
4        <string name="portrait_text">This is the portrait mode.</string>
5    </resources>
6
```



FRAGMENTS AND ACTION BAR

CS 175 Mobile Software Dev

by Dr. Angus Yeung

EXTRA READINGS: FRAGMENTS & ACTION BAR

FRAGMENTS AND ACTION BAR

activity_my.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_gravity="center"
    android:background="@color/black"
    android:orientation="vertical"
    tools:context=".MyActivity">


<LinearLayout
        android:id="@+id/fragment_container"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal">
</LinearLayout>

</LinearLayout>
```

FRAGMENTS AND ACTION BAR

fragment_breakfast.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:orientation="vertical"
6      android:background="@color/black">
7
8  <ImageView
9          android:id="@+id/imageView"
10         android:layout_width="fill_parent"
11         android:layout_height="fill_parent"
12         android:contentDescription="@string/ui_tabname_breakfast"
13         android:orientation="horizontal"
14         android:src="@drawable/breakfast" />
15
16 </LinearLayout>
```

FRAGMENTS AND ACTION BAR

BreakfastFragment.java

```
1  package com.cornez.actionbarexperiment;
2
3  import android.app.Fragment;
4  import android.os.Bundle;
5  import android.view.LayoutInflater;
6  import android.view.View;
7  import android.view.ViewGroup;
8
9  public class BreakfastFragment extends Fragment {
10         @Override
11         public View onCreateView(LayoutInflater inflater,
12                                  ViewGroup container,
13                                  Bundle savedInstanceState) {
14
15             // Inflate the layout for this fragment
16             return inflater.inflate(R.layout.fragment_breakfast,
17                                     container, false);
18     }
19 }
```

FRAGMENTS AND ACTION BAR

MyActivity.java

```
1  package com.cornez.actionbarexperiment;
2
3  import android.app.ActionBar;
4  import android.app.Activity;
5  import android.app.Fragment;
6  import android.app.FragmentTransaction;
7  import android.content.Context;
8  import android.os.Bundle;
9  import android.view.Menu;
10 import android.view.MenuItem;
11
12 public class MyActivity extends Activity {
13
14     private static final String TAB_KEY_INDEX = "tab_key";
15     private Fragment breakfastFragment;
16     private Fragment lunchFragment;
17     private Fragment snackFragment;
18     private Fragment dinnerFragment;
19
20     @Override
21     protected void onCreate(Bundle savedInstanceState) {
22         super.onCreate(savedInstanceState);
23         setContentView(R.layout.activity_my);
24
25         // SET THE ACTIONBAR
26         ActionBar actionBar = getActionBar();
27         actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
28         actionBar.setDisplayShowTitleEnabled(false);
29
```

```
30 // CREATE THE TABS AND BIND THEM TO THE ACTIONBAR
31         ActionBar.Tab breakfastTab = actionBar.newTab().setText(
32                 getString(R.string.ui_tabname_breakfast));
33         ActionBar.Tab lunchTab = actionBar.newTab().setText(
34                 getString(R.string.ui_tabname_lunch));
35         ActionBar.Tab snackTab = actionBar.newTab().setText(
36                 getString(R.string.ui_tabname_snack));
37         ActionBar.Tab dinnerTab = actionBar.newTab().setText(
38                 getString(R.string.ui_tabname_dinner));
39
40         // CREATE EACH FRAGMENT AND BIND THEM TO THE ACTIONBAR
41         breakfastFragment = new BreakfastFragment();
42 snackFragment = new SnackFragment();
43         lunchFragment = new LunchFragment();
44         dinnerFragment = new DinnerFragment();
```

FRAGMENTS AND ACTION BAR

```
45          // SET LISTENER EVENTS FOR EACH OF THE ACTIONBAR TABS
46          breakfastTab.setTabListener(new
47  MyTabsListener(breakfastFragment,
48                  getApplicationContext()));
49          snackTab.setTabListener(new
50  MyTabsListener(snackFragment,
51                  getApplicationContext()));
52
53          lunchTab.setTabListener(new
54  MyTabsListener(lunchFragment,
55                  getApplicationContext()));
56  dinnerTab.setTabListener(new
57  MyTabsListener(dinnerFragment,
58                  getApplicationContext()));
59
60          // ADD EACH OF THE TABS TO THE ACTIONBAR
61          actionBar.addTab(breakfastTab);
62          actionBar.addTab(lunchTab);
63          actionBar.addTab(snackTab);
64          actionBar.addTab(dinnerTab);
65
66
67          // RESTORE NAVIGATION
68          if (savedInstanceState != null) {
69              actionBar.setSelectedNavigationItem(
70                  savedInstanceState.getInt(TAB_KEY_INDEX, 0));
71          }
72      }
```

FRAGMENTS AND ACTION BAR

```
73  class MyTabsListener implements ActionBar.TabListener {
74          public Fragment fragment;
75
76          public MyTabsListener(Fragment f, Context context) {
77              fragment = f;
78          }
79
80          @Override
81          public void onTabReselected(ActionBar.Tab tab,
82  FragmentTransaction ft) {
83          }
84
85          @Override
86          public void onTabSelected(ActionBar.Tab tab,
87  FragmentTransaction ft) {
88              ft.replace(R.id.fragment_container, fragment);
89          }
90
91          @Override
92          public void onTabUnselected(ActionBar.Tab tab,
93  FragmentTransaction ft) {
94              ft.remove(fragment);
95          }
96      }
97
98
99      @Override
100     public boolean onCreateOptionsMenu(Menu menu) {
101         // Inflate the menu;
102         getMenuInflater().inflate(R.menu.my, menu);
103         return true;
104     }
105
106     @Override
107     public boolean onOptionsItemSelected(MenuItem item) {
108         // Handle action bar item clicks here. The action bar will
109         // automatically handle clicks on the Home/Up button, so long
110         // as you specify a parent activity in AndroidManifest.xml.
111         int id = item.getItemId();
112         if (id == R.id.action_settings) {
113             return true;
114         }
115         return super.onOptionsItemSelected(item);
116     }
117 }
```