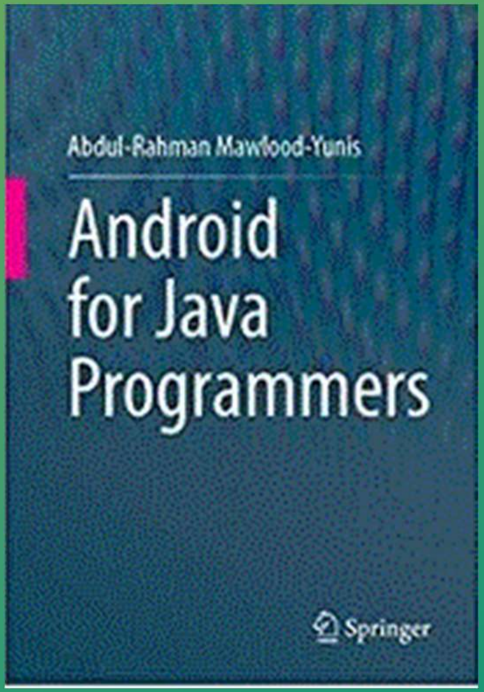
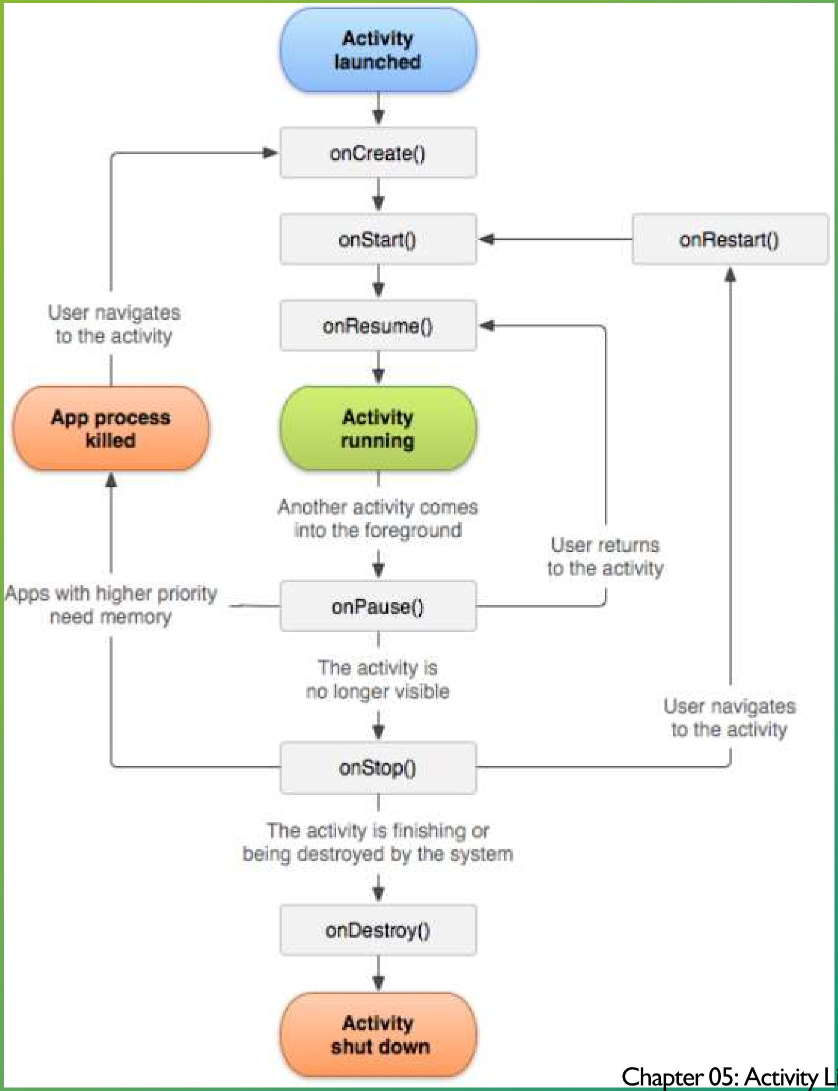
CHAPTER 05: ACTIVITY LIFECYCLE AND PASSING OBJECTS BETWEEN SCREENS USING PARCELABLE INTERFACE



Activity states and the callback method

- The Activity states and the callback method order diagram provided by Google are shown below



Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

What you will learn in this chapter

Use	the Manifest file for multiple Activities
Create and use	the Launcher Activity
Understand	Activity lifecycle, the creation, running, and destruction of Activities
Implement	the Activity lifecycle callback methods
Implement	the Parcelable interface and pass objects between Activities
Learn	how to pass objects between Activities

Check out the demo project

- How to run the code: unzip the code in a folder of your choice, then in Android Studio click File->import->Existing Android code into the workspace

5.1 Introduction

In this chapter, we use the demo app developed specifically for this chapter to explain Activity lifecycles and callback methods, as well as the Parcelable interface

The app has four buttons on the main page

The first button is for demoing the **onCreate** method creation and usage, which is an important method of the Android Activity class and Activity lifecycle

The second button is for demoing **the Activities' lifecycle**.

The third button is for passing objects between Activities using the **Parcellable interface**.

The last button is for **restores the Activity states**

Part 1: Activity States

- In this part, we describe states and Activity lifecycles and why it is important to understand these concepts for app development

5.2 Activity and States

It is important to understand **what the *state* is** to understand the Activity lifecycle.

Activities, like Java objects, can be in different states, where a state is a snapshot of an Activity at a given time.

When instance variables of an Activity are holding values and/or its operations are executing, the Activity is in a state. **The state can be *starting*, *resuming*,**

The set of states that an Activity can go through during its lifetime is called the ***Activity lifecycle***.

All the states an Activity has, along with transitions from one state to another, form a directed graph.

5.2 Activity and States

While users navigate in and out of the application or navigate through the Activities (screens) of your app, Activity states change.

While this happens, the Android system calls various lifecycle *callback methods*.

You can control how your application behaves as the user interacts with your application and Activity states change.

The control is implemented through callback methods.

So far, you have only seen the onCreate() callback method.

5.2 Activity and States

Each Android app is made of one or more Activities (screens)

Because a phone has a limited size display, the app designer cannot present all the app's views at once on the device's screen.

Mobile app views, or screens, overlay one another in the **back stack** as the user navigates through the app, and this leads to Activities being in different states.

5.2 Activity and States

- The developer can implement, what kind of **actions** take place when an Activity transits from one state to another. For example:

- The developer can ***save data*** to a database, a file, or another Activity before the Activity is killed by the system so that the next time the app runs, it starts from where it left off.

- The developer can **bring into focus the view** that was previously running in the foreground.

- These decisions need to be made at the application design stage

5.3 Transition Between States

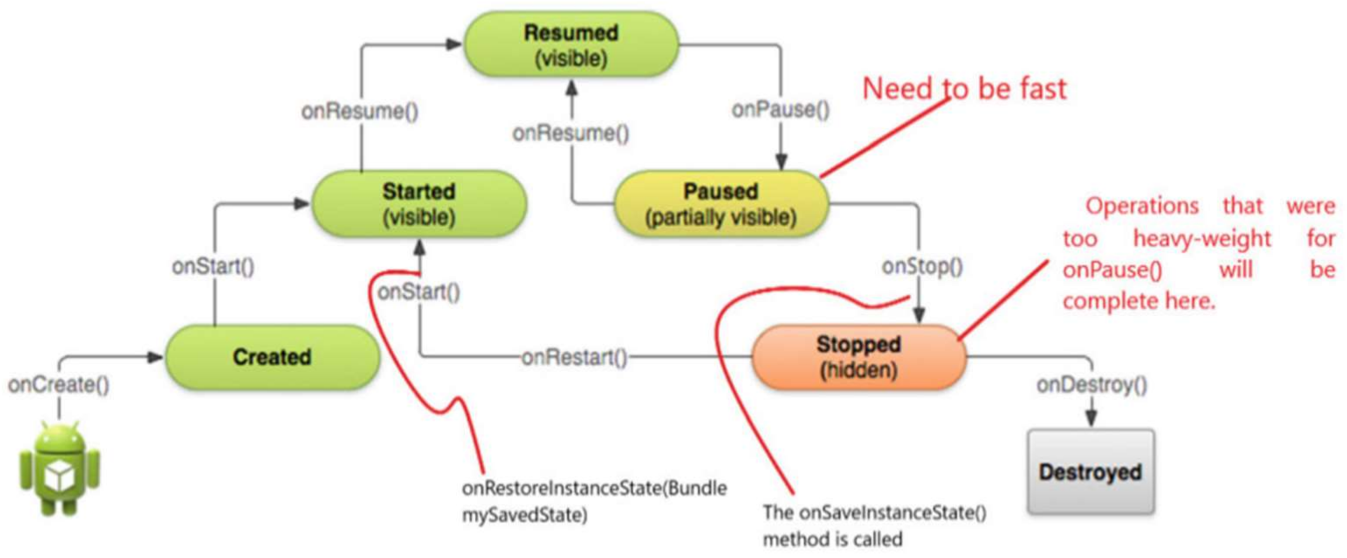
At any given time, an Activity can be in one of these three states:

Running , the activity has the focus and is at the top of the activity stack

Paused, the device goes to sleep; the activity is partially hidden

Stopped, the activity is obscured by another activity

5.3 Transition Between States


Resumed (visible)
onResume()
onResume()
onPause()
Need to be fast
Started (visible)
Paused (partially visible)
onStart()
onStart()
onStop()
Operations that were too heavy-weight for onPause() will be complete here.
Created
onRestart()
Stopped (hidden)
onCreate()
onDestroy()
Destroyed
onRestoreInstanceState(Bundle mySavedState)
The onSaveInstanceState() method is called

At any given time, an Activity can be in one of these three states:

5.3 Transition Between States

Activity states are of two types, static and transient states.

When Activities exist or remain for a **long period** in a state, the state is called a **static state**.

If the Activities exist in a state for a very **short time**, the state is called a **transient state**.

5.3 Activity state examples

After the onCreate() method, the onStart() method is called followed by the onResume() method.

When these callback methods are called, the Activity would be in a *transient state*. That is, it will be in these states for a very short time.

In these cases, the transition between the states does not require user intervention, the changes happen automatically, i.e., the method calls are initiated by the Android system.

When the app is running, e.g., you are browsing the internet on your device, the Activity is running and is in a resumed *static state*.

5.4 The Launcher Activity

So far, we have worked with apps that have one or two Activities.

However, almost all Android applications will have several Activities.

One of these Activities is the Launcher Activity.

Each application has only one *Launcher Activity*. It is equivalent to the Java class with the *main* method that starts an application.

The Launcher Activity is executed when the user clicks the application icon.

The programmer must decide which Activity is the Launcher Activity and declare it in the Android Manifest.xml file.

5.4 The Launcher Activity

In the Manifest XML file, you will have an element called *application*, where all Activities of your app are children of this element, including the Launcher Activity.

Inside the application element and the Activity that you would like to declare as the Launcher Activity, you include a subelement called *intent-filter*.

The intent-filter element has two properties, action, and category.

For your activity to be the Launcher activity, its **intent-filter action name** must be **MAIN**, and its **category** must be LAUNCHER.

Next slide shows an example of the AndroidManifest.xml file in which the MainActivity has been defined as the LAUNCHER activity and DisplayMessageActivity is the second Activity.

5.4 The Launcher Activity

```
• <?xml version="1.0" encoding="utf-8"?>
• <manifest xmlns:android="http://schemas.android.com/apk/res/android"
•     package="code.android.abdulrahman.com.LifeCycleWithParcellable">
•     <application
•         android:allowBackup="true"
•         android:icon="@mipmap/ic_launcher"
•         android:label="@string/app_name"
•         android:roundIcon="@mipmap/ic_launcher_round"
•         android:supportsRtl="true"
•         android:theme="@style/AppTheme">
•         <activity android:name=".MainActivity">
•             <intent-filter>
•                 <action android:name="android.intent.action.MAIN" />
•                 <category  android:name=
•                                    "android.intent.category.LAUNCHER" />
•             </intent-filter>
•         </activity>
•         <activity android:name=".DisplayMessageActivity" />
•     </application>
• </manifest>
```

5.5 Implementing onCreate () method

- To develop an Android app, you need to implement the onCreate() method for each Activity
- It is the only callback method required to be implemented
- The onCreate() code executes once for the entire lifetime of the Activity
- In the onCreate() method for the Launcher Activity, you implement basic application startup logic, set up an interface, initialize class scope variables, instantiate widgets, etc
- The onCreate() method acts as the constructor method in Java classes

5.5 Implementing onCreate

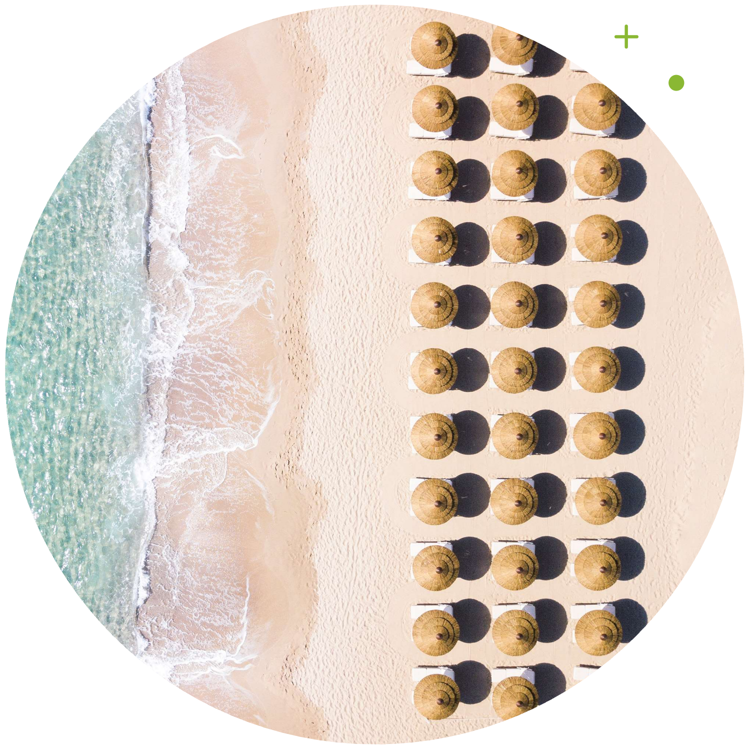
The onCreate() callback method is called when the system creates the Activity for the first time

The method has only one parameter. The type of the parameter is ***Bundle*** and its value is null when the app launches

The parameter value changes to store the Activity's previous state if the Activity is recreated for any reason. For example, when the device is rotated, or the app's language is changed

The method signature for the onCreate method is as follows:

protected void onCreate(Bundle savedInstanceState);



5.5 Implementing onCreate

Two actions that are done inside the onCreate method for almost all Activities are:

- super.onCreate(savedInstanceState) method call is made to the base class onCreate () method
- the activity layout is set using the setContentView(Layout) method call.

5.5 Implementing onCreate

The code for the onCreate() method below shows the following:

- The use of the Bundle object
- A call to the Base class method using the keyword *super*
- The Activity's layout is set using the setContentView method

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

5.6 Bundle Class

- Bundle is a simple class that acts as a data structure for storing data
- It is like a map class in Java that uses a key/value pair to store values
- The Bundle object provides the means to save and retrieve the activity's state data
- When the first instance of an Activity is created, the Bundle object is null
- The Bundle object is used with the activity methods:

onCreate(Bundle savedInstanceState)

onSaveInstanceState(Bundle outState)

onRestoreInstanceState (Bundle outState)

5.6 Bundle Class

The Bundle class has several methods. Here are a few of them:

clear(), removes all elements from the Bundle.

clone(), clones the current Bundle

deepCopy(), makes a deep copy of the given bundle.

describeContents(), reports the nature of the Parcelable's contents

5.6 Bundle Class

Another import method of the Bundle class is getBundle(String key)

returns the value associated with the given key, or null if the key doesn’t exist.

Bundles have "get" and "put" methods for all the primitive types, Parcelables, and Serializable objects.

getInt(), putInt(), getDouble(), putDouble(), etc.

5.6.1 Using Bundle Object with Intent

- You can use the Bundle object with Intent in a few different ways:
- Use getExtras() to get a Bundle object from the Intent object and use the put methods with the key/values to insert values into the Bundle.
- The code snippet below shows how to use getExtras() to retrieve the Bundle object from the Intent object and put a value into it using a key.

```
Intent intent = new Intent (this, secondActivity.class);
   Bundle extras = intent.getExtras();
   extras.putString("key", "value");
```

- Note, in the case above, if the Intent object is created without the Bundle object added to it, calling getExtras() on the Intent object will return null.

Intent methods that make use of Bundle

- getExtras(): Returns a Bundle containing any additional data that was included with the Intent. This can be used to pass extra information between components.
- getStringExtra(String key): Retrieves a String extra from the Intent's Bundle.
- getIntExtra(String key, int defaultValue): Retrieves an integer extra from the Intent's Bundle with a default value if the extra is not found.
- getBooleanExtra(String key, boolean defaultValue): Retrieves a boolean extra from the Intent's Bundle with a default value if the extra is not found.
- getFloatExtra(String key, float defaultValue): Retrieves a float extra from the Intent's Bundle with a default value if the extra is not found.
- getDoubleExtra(String key, double defaultValue): Retrieves a double extra from the Intent's Bundle with a default value if the extra is not found.
- getSerializableExtra(String key): Retrieves a Serializable extra from the Intent's Bundle.
- getParcelableExtra(String key): Retrieves a Parcelable extra from the Intent's Bundle.
- getAction() and setData(Uri uri): Gets or sets the action and data URI associated with the Intent, respectively.

5.6.1 Using Bundle Object with Intent

Instantiate the Bundle object and use the putExtras() method to add the Bundle object into the Intent object.

The Bundle object can be used with the StartActivity as well to pass data to the second activity;

```
 Bundle bundle = new Bundle();
 bundle.putString("key", "value");
Intent newIntent = new Intent(this,secondActivity.class);
newIntent.putExtras(bundle);
```

```
startActivity (Intent intent, Bundle bundle);
```

5.6.1 Using Bundle Object with Intent

The Bundle object provides the means to save and retrieve the Activity's state data.

When the first instance of an Activity is created, the Bundle object is null.

The Bundle object is used with the Activity methods to save and restore Activity states.

- *onCreate(Bundle savedInstanceState)*,
- *onSaveInstanceState(Bundle outState)*
- and *onRestoreInstanceState* (*Bundle outState*)

5.6.1 Using Bundle Object with Intent

The Android system temporarily destroys the running activity when the user

- rotates their device
- changes the device mode to split-screen, or
- changes the language settings,
- changes the configuration of the device

The method onSaveInstanceState(Bundle outState) is then invoked for saving the Activity's state for recreation

Using Bundle with **onSaveInstanceState**

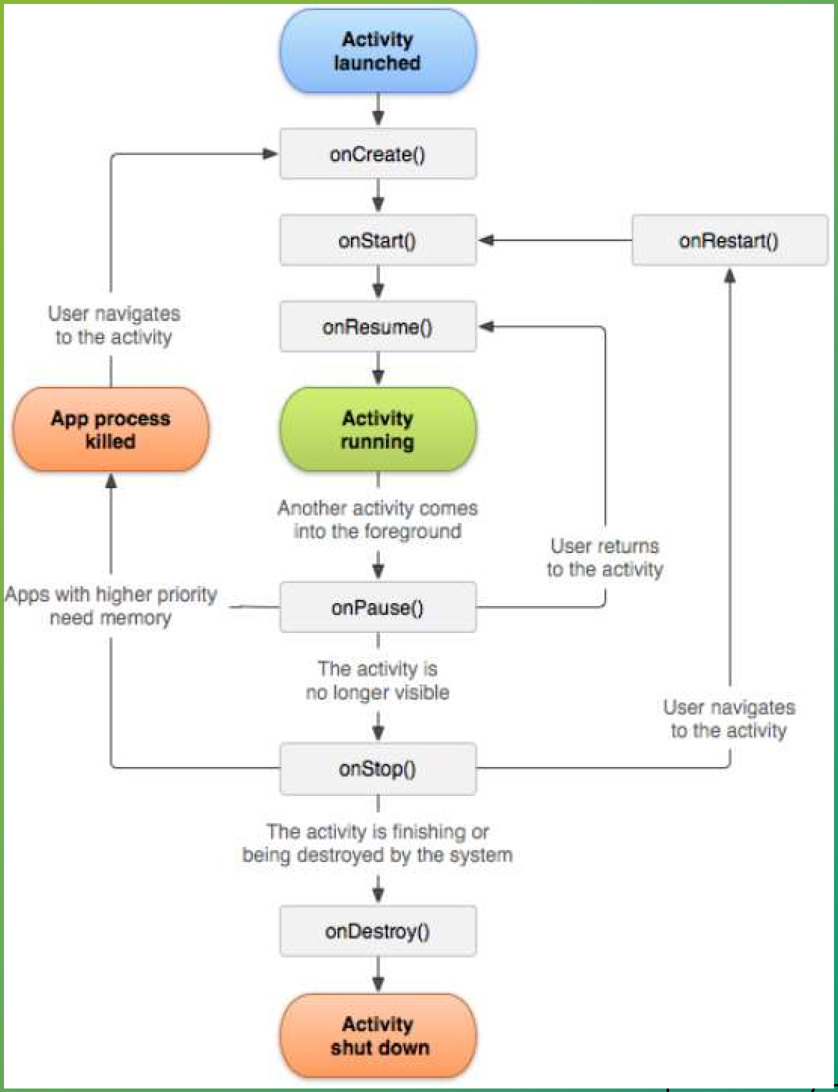
```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);

    String nameValue = "Abdul-Rahman";
    int accountNumber = 12345;

    outState.putString("name", nameValue);
    outState.putInt("accountNumber", accountNumber);
}
```

Using **onRestoreInstanceState** to save and restore the Activity’s state

```
 @Override
protected void onRestoreInstanceState(@NonNull Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);

    String r_name = savedInstanceState.getString("name");

    int r_accountNumber = savedInstanceState.getInt("accountNumber");

    textView.setText("\n" + "Restoring state \n" + r_name + " " + r_accountNumber);
 }
```

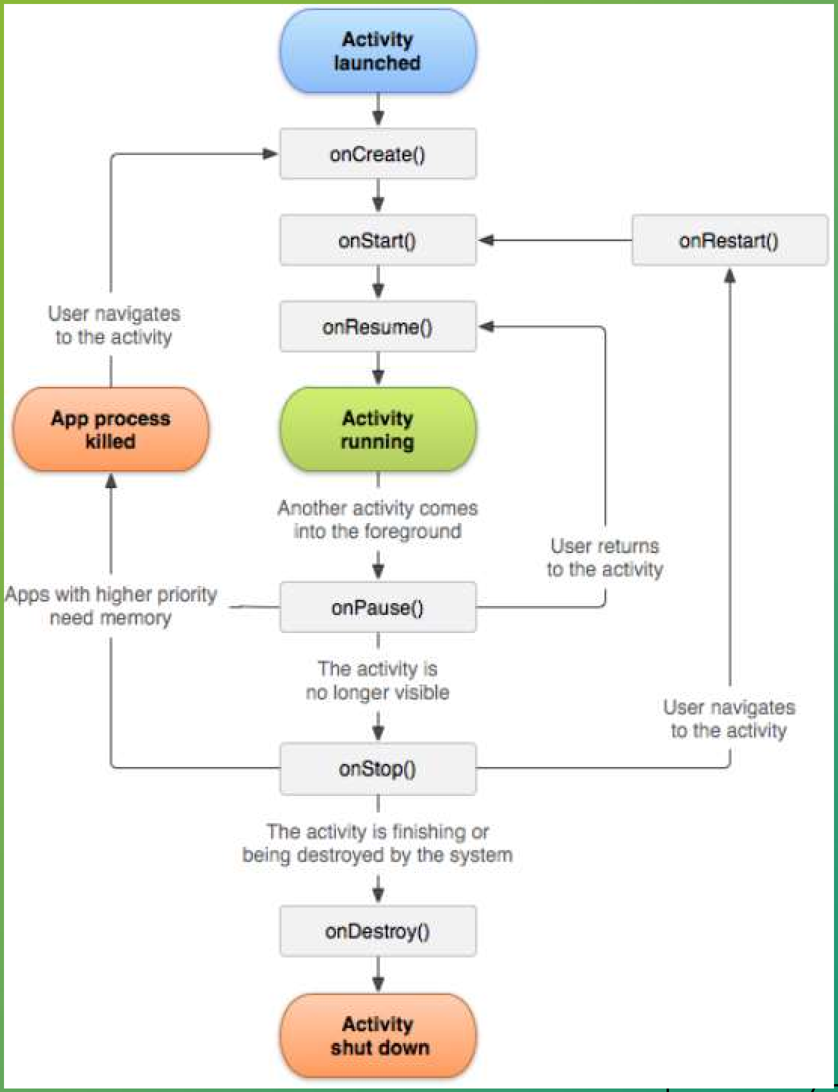
PART 2: UNDERSTANDING ACTIVITY LIFECYCLE

In this part, we continue to investigate the activity's lifecycle callback methods


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

5.7 Understanding the onDestroy Method

- The OnDestroy() method is called when the activity is about to get destroyed.
- It is usually used to release resources
- An Activity gets destroyed either because it is no longer needed or because of configuration changes


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

5.7 Understanding the onDestroy Method

- When an Activity gets destroyed because of configuration changes, a new Activity is created and the onCreate() method is called immediately
- You can use the **finish()** method to destroy an Activity
- and use the **isFinshing**() method to find out if the finish() method has been called

5.7 Understanding the onDestroy Method

In most cases, you do not need to implement the onDestroy method.

This is because most of the code/data cleanup is done using the onPause() or the onStop() method.

5.7 Understanding the onDestroy Method

To decide whether or not to implement the *onDestroy()* method in your code, be aware the following:

The onDestroy() method is the last lifecycle callback method. It is called when the application is removed from the system memory

If you call the *finish()* method inside the onCreate() method, the onDestroy method can be used for code/data cleanup This is because the *finish()* method triggers the OnDestroy method.

The onDestroy() method can be used to kill long-running processes or when a large amount of resources is released unwillingly

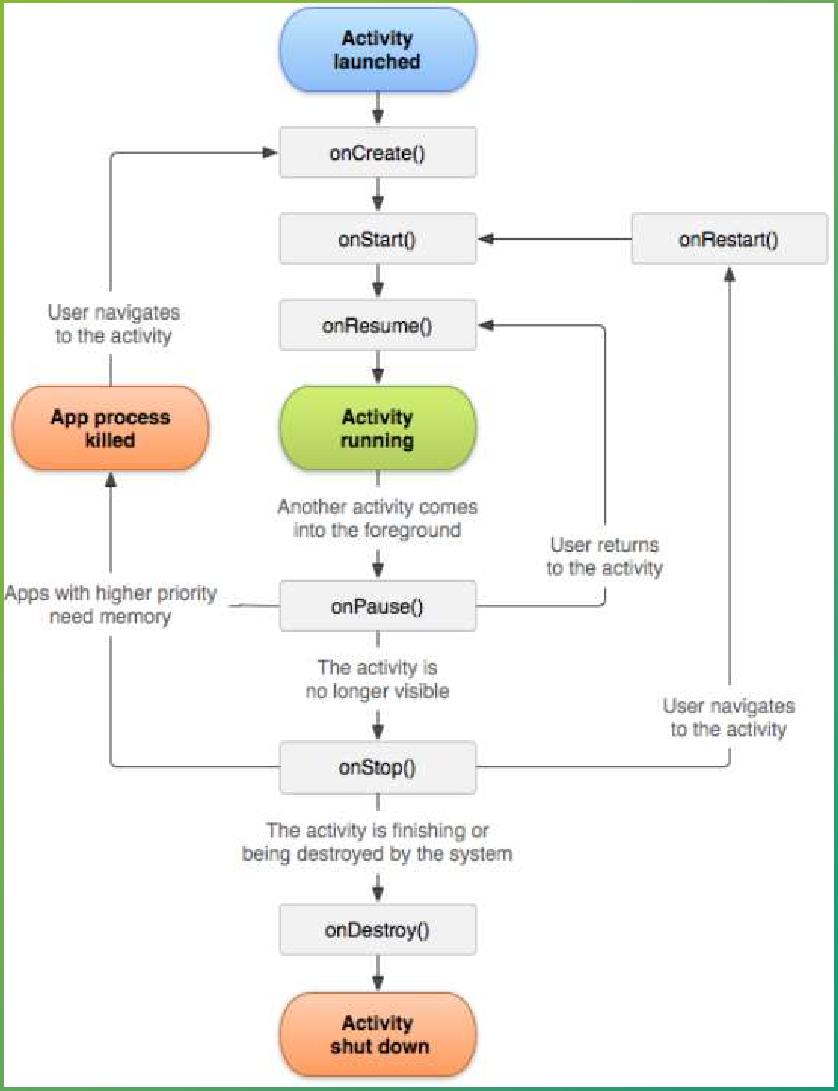
5.7 onDestroy and Finish Methods

What is unique about the Activity's finish() method

it calls the onDestroy() method, and if you come back to the Activity, the onCreate() method is called again

The onCreate() method is supposed to be called only once, the first time you create an Activity.

Now, because of the finish() method, if you return to the Activity, the onCreate() method is called again.


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

5.8 Pausing and Resuming an Activity

When a foreground Activity is partially obscured (because of another activity, for example), the system calls the **onPause()** method.

When the **onPause()** method is called, your application is still partially visible.

Once an activity is in the paused state, there are two states it can move to:

the *resume* or *stop* state

5.8 OnPause()

Your app should not consume resources while it is in the paused state

The *onPause()* implementations must not take a long time as well

The next Activity will not resume until the pausing method is completed

5.8 Pausing and Resuming an Activity

If applicable to your app, inside the onPause() method do the following:

Stop animations if you have any

Release system resources such as wifi locks, broadcast receivers, and close files

Release resources that consume battery life such as background services

Any other action similar to the ones mentioned should be stopped while your app is in the onPause() state.

5.8 Pausing and Resuming an Activity

When the user resumes activity from the paused state, the system calls onResume()

The onResume() method is called every time an application comes to the foreground, including the first time

If the *OnResume* method is called, you need to initialize resources released previously in the *onPause()* method

In the *onResume()* state, the user can interact with the Activity

The Activity has moved to the top of the Activity stack and accepts user input

5.8 Pausing and Resuming an Activity

Calling AlertDialog, Toast, Date/Time pickers, and similar objects will bring new windows on top of the current one, but they will not lead to executing the onPause() method.

Only launching a new Activity will push the current running Activity to the background and result in executing the onPause() callback method.

The fourth button on the demo app enables you to practice what we just described, obscuring the running window without calling the onPause() method.

5.9 Stopping and Restarting an Activity

An Activity might be stopped and restarted because:

- The user switch apps
- The user performs an action that starts a new Activity
- The user receives a phone call while using an app
- The app is doing a complex task, such as a database write
- The onStop() callback is called when an Activity becomes hidden

5.9 Stopping and Restarting an Activity

While stopped, the Activity instance still lives in the system memory

If the Android system runs out of memory space or experiences memory shortages, it might destroy the stopped Activity

For that reason, you often need to only implement the onPause() and onResume() methods and not the onStop() method

5.9 Stopping and Restarting an Activity

The onRestart() method is called when an Activity comes to the foreground from a stopped state.

The OnStart() method is not called when the application first starts.

The onStart() method is called in two cases:

- after onCreate() when an Activity becomes visible
- after onRestart() calls

Generally, you only need to handle the onStart() method.

5.10 Restoring Activity State

By default, the Android system only saves limited state information

These include Views with a unique id, scroll positions in a ListView object, etc

If you want to restore more than the default values when an Activity is recreated, then you need to take care of it

You can do that using either the onSaveInstanceState() or onRestoreInstanceState() callback methods.

5.10 Restoring Activity State

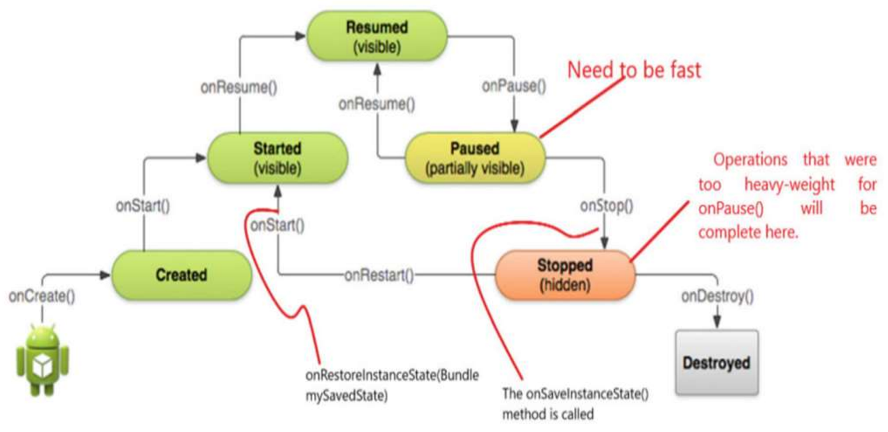
The *onSaveInstanceState()* method is called before an Activity begins to stop.

Here, you have a chance to save the Activity’s state information to the Bundle object using key-value pairs.

The saved information can be used for the Activity recreation after its destruction.

When you override the onSaveInstanceState() method, you must call the superclass in your coding. The call to the superclass saves the state of the View hierarchy, see the code snippet below:

```
@Override
  public void onSaveInstanceState(Bundle outState) {
     super.onSaveInstanceState(outState);
     outState.putString("name",
           String.valueOf(nameView.getText()));
  }
```


Resumed (visible)
onResume()
onResume()
onPause()
Need to be fast
Started (visible)
Paused (partially visible)
onStart()
onStart()
onStop()
Operations that were too heavy-weight for onPause() will be complete here.
Created
onRestart()
Stopped (hidden)
onCreate()
onDestroy()
Destroyed
onRestoreInstanceState(Bundle mySavedState)
The onSaveInstanceState() method is called

5.10 Restoring Activity State

There are two places where you can retrieve state information from the Bundle object.

- You can do it inside in the onCreate(Bundle mySavedState) method.

If the bundle is not null, you can retrieve state information.

- By implementing the onRestoreInstanceState(Bundle mySavedState) callback method,
- you can retrieve the saved state.

```
   @Override
   public void onRestoreInstanceState (Bundle mySavedState)
{
      super.onRestoreInstanceState(mySavedState);
      if (mySavedState != null) {
         String count = mySavedState.getString("name");
         if (count != null)
            nameView.setText(count);
      }
   }
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   mShowCount = findViewById(R.id.show_count);
   if (savedInstanceState != null) {
      String count = savedInstanceState.getString("count");
      if (mShowCount != null)
         myView.setText(count);
   }
}
```

PART 3: LIFECYCLE ILLUSTRATION APP

In our demo app we demonstrate the callback method calls and the order of the calls

The lifecycle callback methods implemented in the demo app are

protected void **onCreate**(Bundle savedInstanceState);

protected void **onStart**();

protected void **onRestart**();

protected void **onResume**();

protected void **onPause**();

protected void **onStop**();

protected void **onDestroy**();

protected void **onSaveInstanceState**(Bundle outState);

5.12 Lifecycle Callback Methods

- The demo app does the following:

- At each callback method, some text, or a string, is appended to a class variable.

- The added text indicates which method of the MainActivity has been called, which also indicates the Activity state.

- Once the user presses the send button, the values of the class variable are passed to the DisplayMessageActivity and are displayed on the screen showing the order of the method calls.

5.12 Lifecycle Callback Methods

- For example, the first time you press the send message button, the following messages are displayed on the second screen.

- from MainActivity, onCreate method invoked 0.
- from MainActivity, onStart method invoked 1.
- from MainActivity, onResume method invoked 2.

5.14 Callback Methods for theDisplayMessageActivity

Similar to the call order of the MainActivity callback methods, when DisplayMessageActivity starts, the onCreate → onStart → onResume methods are called

When the Back To Main Activity button is pressed, the texts for these three methods are put into the Intent object and passed to the MainActivity followed by a call to the onPause and OnStop methods

The onCreate → onStart → onResume method information for the DisplayMessageActivity are displayed on the MainActivity screen

Using the *submit* and *back to Main Activity* buttons, along with clear and close buttons, should enable you to test all the possible paths of the Activity lifecycle.

5.16 Callback Method Implementations

The code snippet in chapter 5, listing 5.5 shows the callback method implementations for the main activity where important steps are bolded for your attention

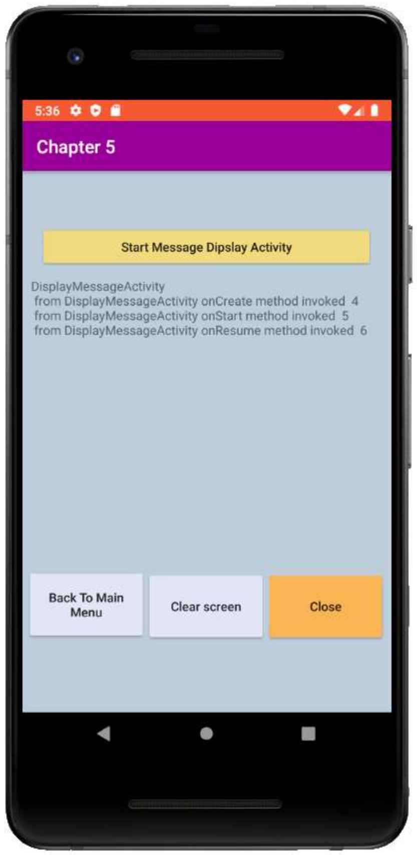
5.17 Trigger the onPause Method

We said earlier that Snackbar and Dialog box will not cause onPause method to be triggered even though they will come up on top of the activity and cover the main activity for some time

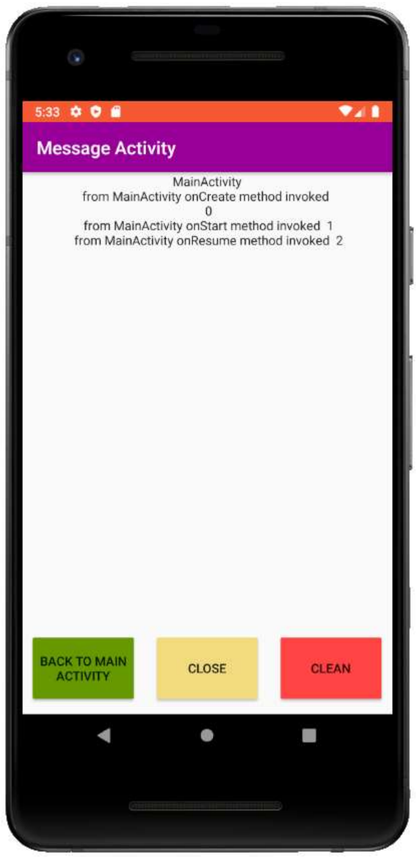
The same thing is true when you attach fragment views to the activity's layout

To demonstrate how you can trigger the **onPause** method and without going back and forth between activities, you can create an Activity as a Dialog box

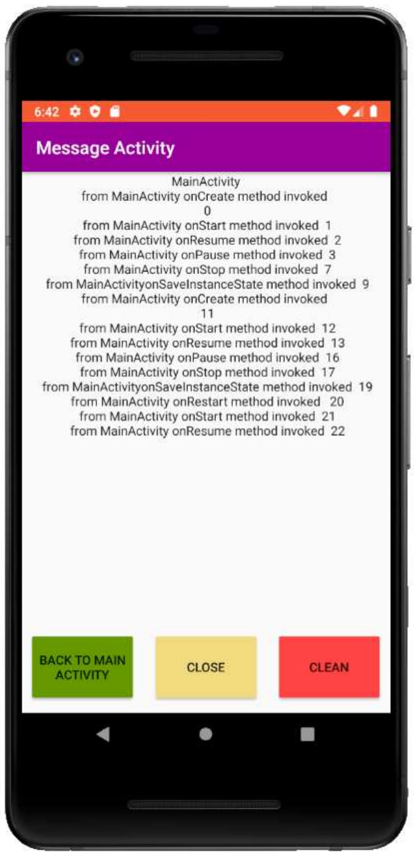
A complete code for triggering onPause method is shown in chapter 5, Listing 5.6.



```
DisplayMessageActivity
 from DisplayMessageActivity onCreate method invoked  4
 from DisplayMessageActivity onStart method invoked  5
 from DisplayMessageActivity onResume method invoked  6
```



```
MainActivity
from MainActivity onCreate method invoked
0
from MainActivity onStart method invoked  1
from MainActivity onResume method invoked  2
```



```
MainActivity
from MainActivity onCreate method invoked
0
from MainActivity onStart method invoked  1
from MainActivity onResume method invoked  2
from MainActivity onPause method invoked  3
from MainActivity onStop method invoked  7
from MainActivityonSaveInstanceState method invoked  9
from MainActivity onCreate method invoked
11
from MainActivity onStart method invoked  12
from MainActivity onResume method invoked  13
from MainActivity onPause method invoked  16
from MainActivity onStop method invoked  17
from MainActivityonSaveInstanceState method invoked  19
from MainActivity onRestart method invoked  20
from MainActivity onStart method invoked  21
from MainActivity onResume method invoked  22
```

Part 4: Creating and Using Parcelable Objects

Java is an object-oriented programming language, and except for primitive data types, almost everything is an object, hence, you need to be able to pass objects between activities

This is accomplished by putting key/value pairs in Intent objects using the putExtra method

In this part, we describe how you can pass user-defined objects between Activities using the Parcelable interface

5.18 Passing User-Defined Objects Between Activities

The Intent class puts restrictions on the type of values that can be passed between the Activities.

The value types are restricted to primitives, arrays, CharSequence, String, Parcelable or Serializable objects.

You have already seen one example of passing objects between Activities by inserting the Bundle object into the Intent object.

5.18 Passing User-Defined Objects Between Activities

- Android recommends using Parcelable rather than Serializable for performance.

- The method signatures for the putExtra methods from the Intent class that take objects are

    - public Intent putExtra (String name, Parcelable value);
    - public Intent putExtra (String name, Serializable value);

5.18 Passing User-Defined Objects Between Activities

Let’s assume you have a class called Grades: **Public class Grades {…}**.

If you want to create an Intent object and try to insert a Grades object into the Intent object to pass between Activities, it will not work.

This code snippet will not work

```
Intent intent = new Intent();
Grades grades = new Grades ();
String name = " a student";
intent.putExtra(name, grades.toString() );
```

5.18 Passing User-Defined Objects Between Activities

You need to change the definition of the Grades class and let it implement the Parcelable interface:

Public class Grades implements Parcelable {…} or

you cast the grades object to the Parcelable object:

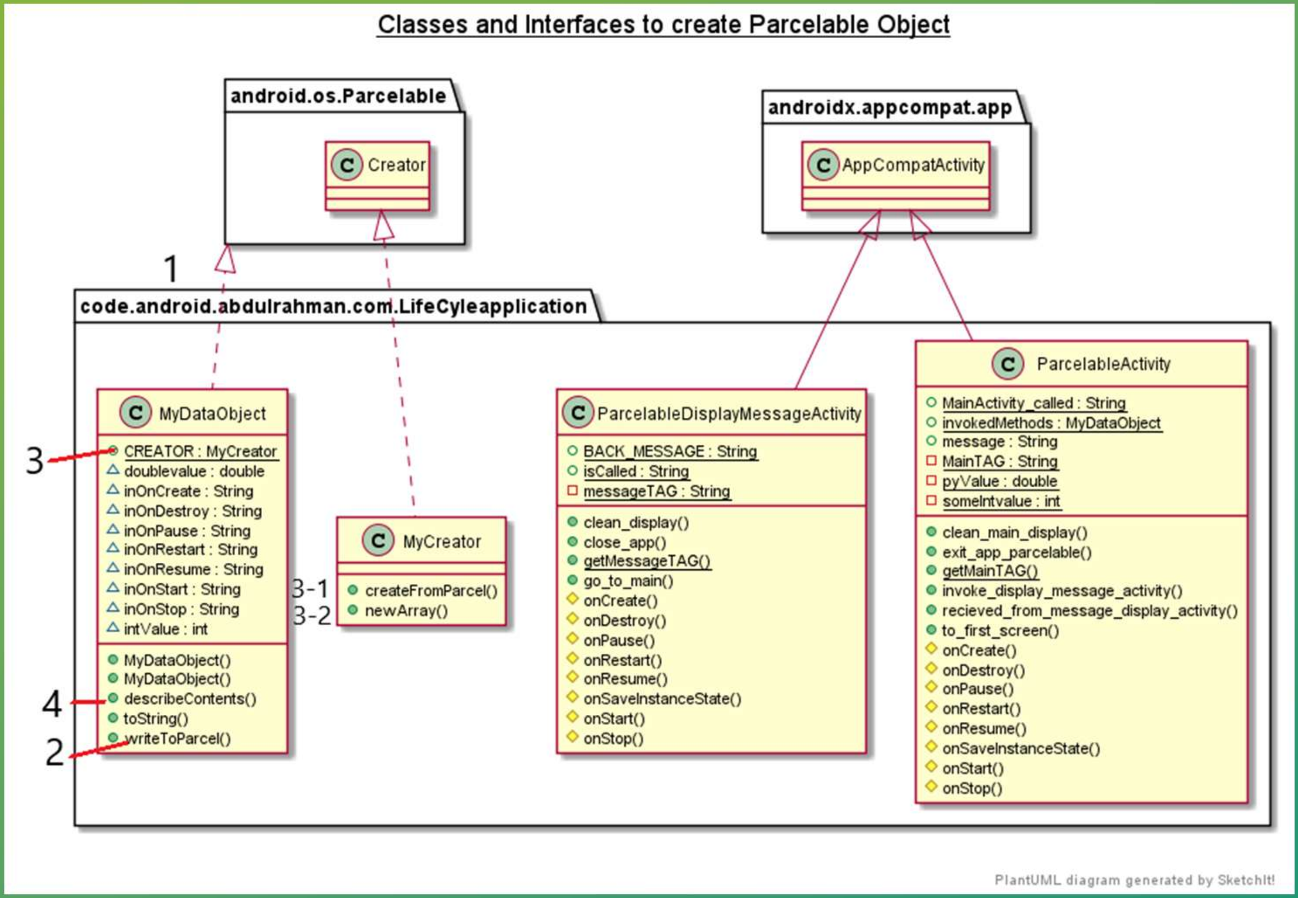
intent.putExtra(name, (Parcelable) grades);

In the first part, we described object serialization and the serializable interface.

5.19 LifeCycle with Parcelable Object

- ParcelableActivity and ParcelableDisplayMessageActivity to pass a Parcelable object between Activities.
- A new Java class called MyDataObject is created to represent the data object that goes back and forth between Activities.
- A new Java class to illustrate how you can create and use Parcelable objects.
- These classes are shown in the class diagram

5.19 LIFECYCLE WITH PARCELABLE OBJECT



5.18 Create Parcelable class

For an object to become Parcelable it needs to comply with certain requirements.

1. Implement the Parcelable interface

public class MyDataObject ***implements*** *Parcelable* { …}

2. Override writerToParcel and

DescribeContent methods from the Parcelable interface

5.18 Create Parcelable class

- Override the public void writeToParcel(**Parcel** destination, int flags) method to flatten your object into a parcel object, i.e., add all of the data in your class fields to a Parcel object

- Override the public int describeContents()

The method implementation can be as simple as returns 0, or any integer value that can be used as an id for some description of the objects contained in the Parcelable instances passed between Activities. returning

5.18 Create Parcelable class

3. Include a **public static final** field of type **Parcelable.Creator** in the class definition.

The Creator field is an Interface with two methods:

public MyDataObject **createFromParcel**(Parcel in){}

public MyDataObject[] **newArray**(int size) {}

5.18 Create Parcelable class

These methods can be implemented as follows:

```
public MyDataObject createFromParcel(Parcel in) {
        return new MyDataObject(in);
    }
public MyDataObject[] newArray(int size) {
        return new MyDataObject[size];
    }
```

5.18 Create Parcelable class

You use the *createFromParcel(Parcel source)* method to create a new Parcelable object and return it. The object is instantiated from the given Parcel class whose data had previously been written by the *writeToParcel* method.

The newArray(int size) creates a new array of the Parcelable class.

5.20 Parcelable Example

MyDataObject implements the constraints listed earlier

1. public class MyDataObject **implements Parcelable** {…}

2. MyDataObject class has **a special *field*** of type interface

public class MyDataObject implements Parcelable {

…

public static final *MyCreator* CREATOR = new   MyCreator();

…

}

- The CREATOR, is of an interface type that has two methods that need to be implemented.
- The interface implementation is done in a separate class called MyCreator.

```
import android.os.Parcel;
import android.os.Parcelable;

public class MyCreator implements Parcelable.Creator <MyDataObject> {

 @Override
   public MyDataObject createFromParcel(Parcel source) {
      return new MyDataObject(source);
   }
   @Override
   public MyDataObject[] newArray(int size) {
      return new MyDataObject[size];
   }
}
```

5.20 Parcelable Example

3 MyDataObject Implementing the Describe Contents method

```
@Override
  public int describeContents() {
     // TODO Auto-generated method stub
     return 0;
  }
```

5.20 Parcelable Example

✓ 4 MyDataObject implements the writeToParcel method

You put any value you need to be passed between Activities inside the Parcel object using methods like writeString, writeInt, WriteDouble, etc. pass

```
@Override
public void writeToParcel(Parcel destination, int flags) {
    destination.writeString(inOnCreate);
    destination.writeString(inOnStart);
    destination.writeString(inOnRestart);
    destination.writeString(inOnResume);
    destination.writeString(inOnPause);
    destination.writeString(inOnStop);
    destination.writeString(inOnDestroy);
}
```

5.20 Parcelable Example

So far, we have implemented all requirements to have a Parcelable object

The code snippet below shows how the ParcelableActivity uses a Parcelable object

5.20 Parcelable Example

```
public class ParcelableActivity extends AppCompatActivity {
    …
 public static MyDataObject MyDataObject = new MyDataObject() ;
…

intent.putExtra("MyDataObject", myDataObject);
        startActivity(intent);
…
}
```

5.20 Parcelable Example

- Note, both the *writeToParcel* method and the constructor of the MyDataObject take *Parcel* as an input parameter.
- This is because an instance of MyDataObject will be written to and restored from Parcel
- The MyDataObject constructor code is shown a Parcel object

```
@Override
public void writeToParcel(Parcel destination, int flags) {
    destination.writeString(inOnCreate);
    destination.writeString(inOnStart);
    destination.writeString(inOnRestart);
    destination.writeString(inOnResume);
    destination.writeString(inOnPause);
    destination.writeString(inOnStop);
    destination.writeString(inOnDestroy);
    destination.writeInt(intValue);
```

```
public MyDataObject(Parcel source) {
   this.inOnCreate = source.readString();
   this.inOnStart = source.readString();
   this.inOnRestart = source.readString();
   this.inOnResume = source.readString();
   this.inOnPause = source.readString();
   this.inOnStop = source.readString();
   this.inOnDestroy = source.readString();
   this.intValue = source.readInt() ;
   this.doublevalue = source.readDouble() ;
```

5.20 Parcelable Example

- Once the Parcel object is created, the createFromParcel method uses the Parcelable constructor to create a Pacelable object and return it.

```
@Override
     public MyDataObject createFromParcel(Parcel source) {
         return new MyDataObject(source); // returns a new Pacelable object
     }
```

5.20 Parcelable Example

- The sequence of the above steps can be summarized as follows:
- StartActivity → writeToParcel() → createFromParcel() → call to the Parcelable Constructor

5.20.6 Passing a Parcelable Object to Second Activity

- To pass a Parcelable object to an Activity via an Intent, you use a key/value pair with the putExtra() method.
- The key argument is a String and is used to get the object in the receiving Activity.
- The value argument is the Parcelable object that you want to pass, i.e., an instance of the MyDataObject class.

5.20.6 Passing a Parcelable Object to Second Activity

public class ParcelableActivity extends AppCompatActivity {

- public String message = "" ; *// "initialize intent data";*
- private static final String MainTAG = "ParcelableActivity";
- public static String MainActivity_called = MainTAG;
- public static **MyDataObject myDataObject =** new **MyDataObject() ;**
- …
- Intent intent = new Intent(this, ParcelableDisplayMessageActivity.class);
- **intent.putExtra(**"MyDataObject", **myDataObject);**

5.20.7 Receiving a Parcelable Object from an Activity

- You can get the Parcelable object in the receiving Activity using key with either

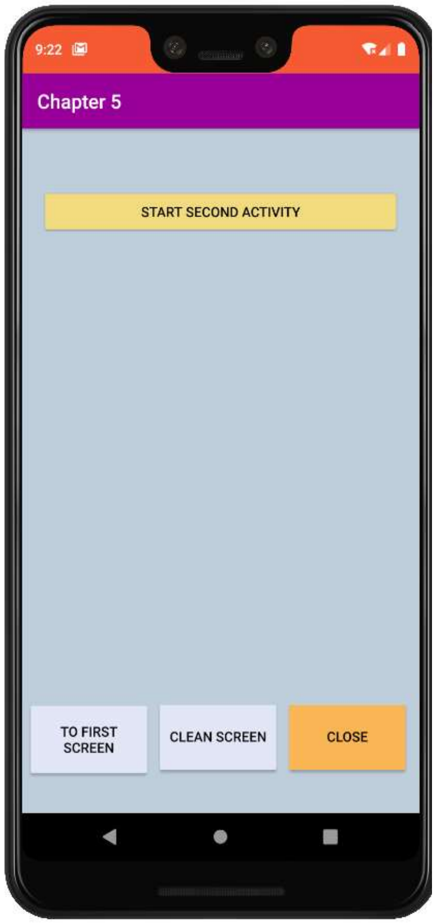
getIntent().getExtras().getParcelable() or
getIntent().getParcelableExtra()

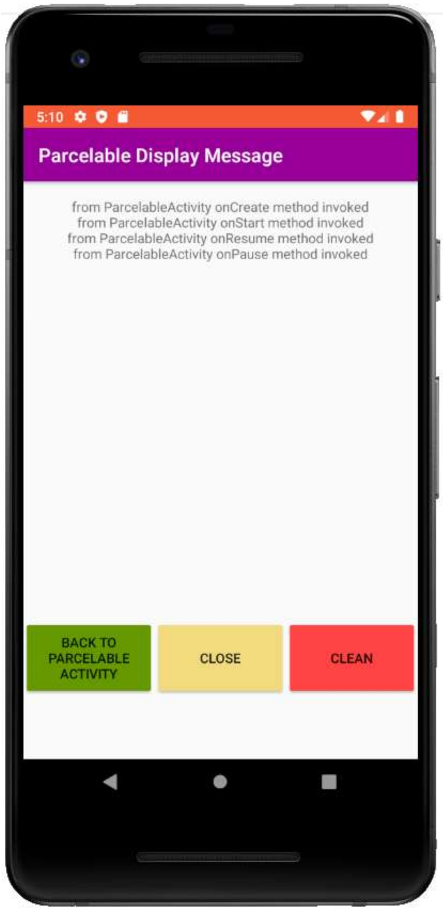
- You need to check that the key/value pair exists in the Bundle by using the hasExtra() method before trying to retrieve the Parcelable object from it

Code for retrieving Parcelable object from the receiver object.

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
                 setContentView(R.layout.activity_parcelable_display_message);
        Intent intent = getIntent();
        if (intent != null) {
            if (intent.hasExtra("myDataObject")) {
                 MyDataObject porecieved =  getIntent().getParcelableExtra("myDataObject");
      …
       TextView messageTextView = findViewById(R.id.TextMessage);
        messageTextView.setText(
               ParcelableActivity.myDataObject.toString());
}
```

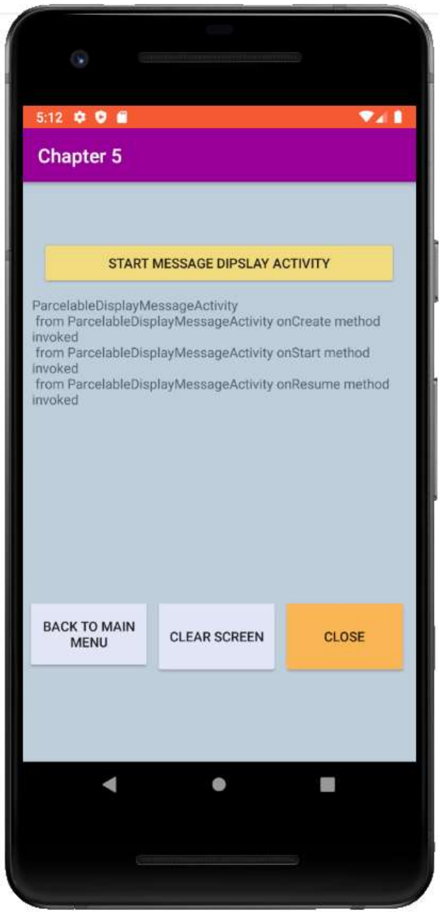
5.20.9 Testing LifecycleParcelable app



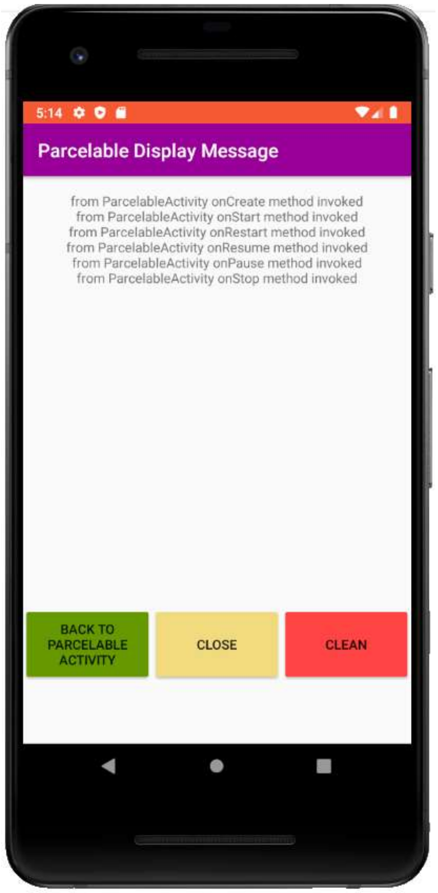


from ParcelableActivity onCreate method invoked
from ParcelableActivity onStart method invoked
from ParcelableActivity onResume method invoked
from ParcelableActivity onPause method invoked

5.20.9 Testing LifecycleParcelable app



```
ParcelableDisplayMessageActivity
 from ParcelableDisplayMessageActivity onCreate method
invoked
 from ParcelableDisplayMessageActivity onStart method
invoked
 from ParcelableDisplayMessageActivity onResume method
invoked
```



```
from ParcelableActivity onCreate method invoked
from ParcelableActivity onStart method invoked
from ParcelableActivity onRestart method invoked
from ParcelableActivity onResume method invoked
from ParcelableActivity onPause method invoked
from ParcelableActivity onStop method invoked
```

5.5 Chapter Summary

In this chapter, we covered two important topics:

- First, we studied the lifecycle of Android Activities, i.e., the states an Activity goes through during its lifetime before it gets destroyed

- We also studied how the lifecycle callback methods can be used to control and manage Android device resources to create a robust application.

- Second, we studied how to pass, not only the primitive data types between Activities but objects as well.

5.5 Chapter Summary

- Java is an object-oriented programming language, and except for primitive types, almost everything is an object. Hence, you need to be able to pass objects between Activities.
- Android allows Bundle objects to be passed between Activities. This is because the Bundle object is a Parcelable object.
- To be able to pass any user-defined object between Activities, your class needs to implement a Parcelable interface.
- We studied how to create a Parcelable object in detail.
- We also created a demo app to go with this chapter to help you learn how to code Parcelable objects and use lifecycle Activities.

Check Your Knowledge

To test your knowledge and your understanding of this chapter, you should be able to describe each of the below concepts in one or two sentences.

- Activity Lifecycle callback methods
- Activity states
- back-stack
- Bundle
- Finish
- isFinishing
- Launcher Activity
- Managing the Activity Lifecycle
- OnDestroy
- OnPause

Check Your Knowledge

To test your knowledge and your understanding of this chapter, you should be able to describe each of the below concepts in one or two sentences.

OnRestoreInstanceState	OnResume	OnSaveInstanceState	OnStart
OnStop	Parcel	Parcelable.Creator interface	Parcelable Interface

Further Reading

Activity: https://developer.android.com/reference/android/app/Activity#ActivityLifecycle

Parcel: https://developer.android.com/reference/android/os/Parcel

Parcelable: https://developer.android.com/reference/android/os/Parcelable