Activities

Amit Gulati, amit.gulati@gmail.com

1

Activity

- **android.app.Activity** class
  - An activity is represented by the **android.app.Activity** class.
  - **Activity** class is generally sub-classed or extended to create an Activity for Application

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

2

2

Activity

- **Activity** sub-classes must to the following
  - Implement the onCreate method and load a ContentView

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

3

3

Activity

- **Activity** must be registered with the Android System
  - Be Declared in the AndroidManifest.xml.

androidmanifest.xml

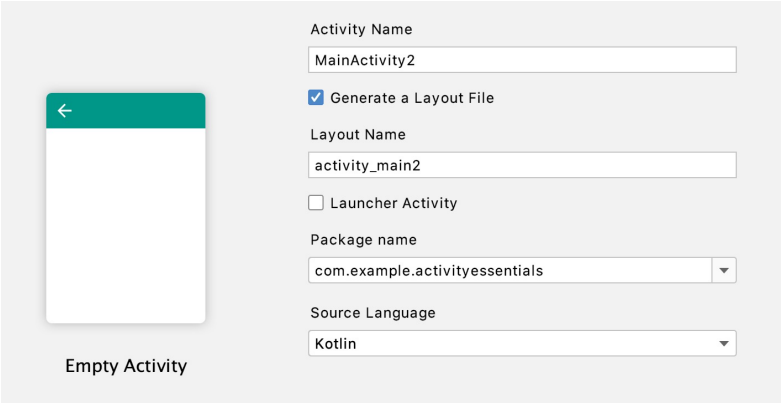
```
<activity
    android:name="com.example.testing.MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

4

4

Activity

- Creating a new Activity using Android Studio



5

5

Activity

- Starting a New Activity.
  - Method that is used to start an activity such that it will not return any information back to the parent activity.

```
public void startActivity (Intent intent)
```

intent, Intent object that will be used to create the activity.

- No apparent relationship between current activity and newly launched activity.

- Terminating an Activity

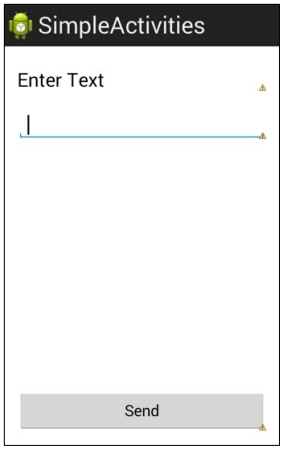
```
void finish ()
```

6

6

Activity and Layout

- Activity layout



res\layout\activity_main.xml

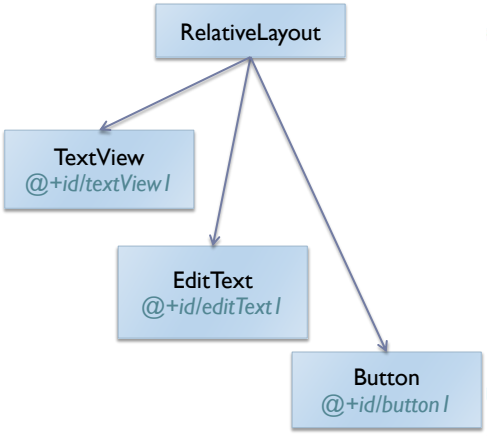
```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="Enter Text"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="21dp"
        android:ems="10" >

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignRight="@+id/editText1"
        android:text="Send" />

</RelativeLayout>
```

7

7

Activity and Layout

- Activity layout is a View Tree



res\layout\activity_main.xml

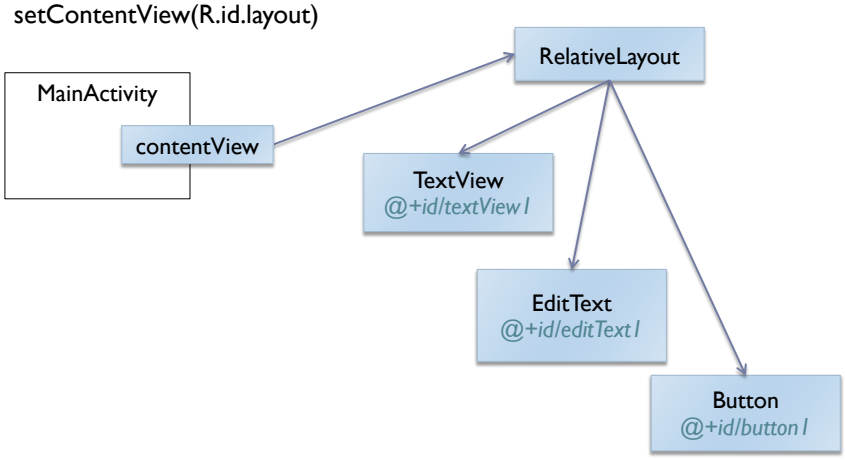
```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="Enter Text"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="21dp"
        android:ems="10" >

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignRight="@+id/editText1"
        android:text="Send" />

</RelativeLayout>
```

8

8

Activity and Layout

- Activity attaches a View to itself
  - setContentView(R.id.layout)

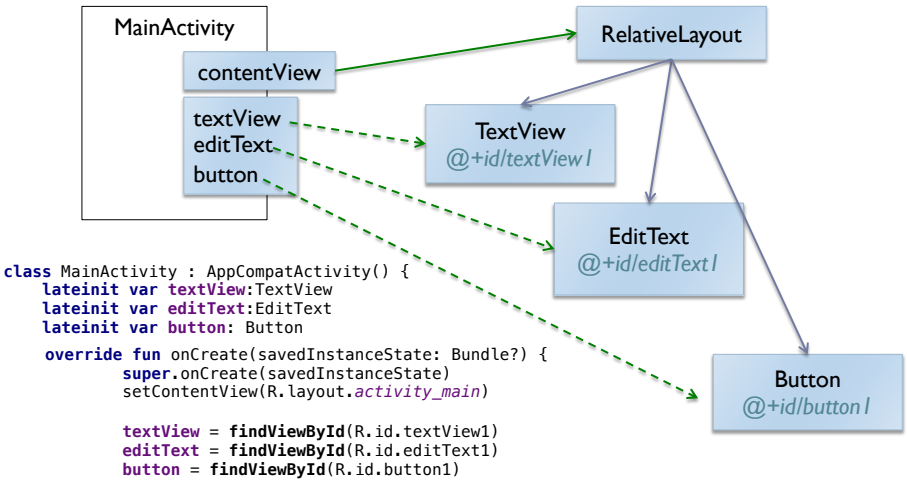


9

9

Activity and Widgets

- Activity and View References




```
class MainActivity : AppCompatActivity() {
    lateinit var textView:TextView
    lateinit var editText:EditText
    lateinit var button: Button

    override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            setContentView(R.layout.activity_main)

            textView = findViewById(R.id.textView1)
            editText = findViewById(R.id.editText1)
            button = findViewById(R.id.button1)
```

10

10

Activity and Widgets

- Activity and View References
  - To interact with Views in the content view hierarchy, we require references to those objects.
  - Method that allows us to get references to View objects in Activities content view

View **findViewById** (int id)

id, ID of the View for which we require a reference.

11

11

Activity and Widgets

- Button Event Handling
  - Button widget provides an attribute in XML "**android:onClick**", that's used for specifying name of the handler method in Activity.

activity_main.xml

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:layout_alignRight="@+id/editText1"
    android:text="Send"
    android:onClick="send" />
```

MainActivity.kt

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    fun send(view:View) { }
}
```

12

12

Activity and Widgets

- View Event Handling using Listener
  - Listener object is attached to the View on which events need to be handled.
    - Example

```
var button:Button
button = findViewById(R.id.mainButton)
button.setOnClickListener(View.OnClickListener {

})
```

13

13

Activity Life-Cycle

Amit Gulati, amit.gulati@gmail.com

14

Activity Life-Cycle

- Active/Running
  - Activity is in the foreground of the screen.
  - The user is interacting with the Activity.
  - Example:
    - **MainActivity** is fully visible, focused and touch events will go to this Activity.
    - MainActivity is Active

Activity Stack

MainActivity
Launcher

15

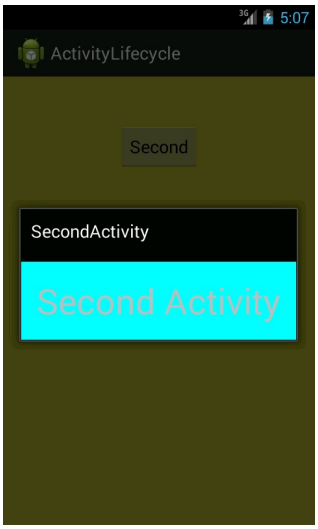
15

Activity Life-Cycle

- Paused/Visible
  - Activity is visible but does not have focus.
    - Obscured by another activity but not fully.
  - Paused activity does not receive user input events.
  - Example:
    - MainActivity is Paused.

Activity Stack

SecondActivity
MainActivity
Launcher



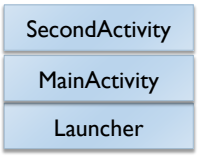
16

16

Activity Life-Cycle

- Stopped
  - Activity is fully obscured by another activity.
  - Activity in stop mode retains all state information and remains in memory.
  - Example:
    - MainActivity is now Stopped.

Activity Stack

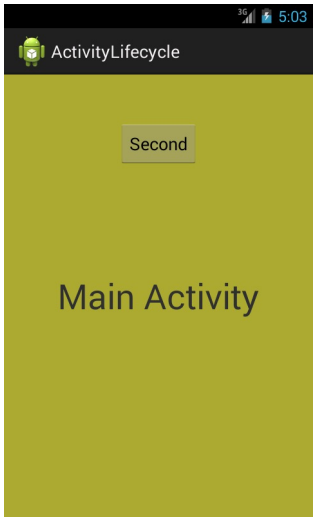


17

17

Activity Life-Cycle

- Terminated/Destroyed
  - Activity is terminated, if
    - Back button is pressed by user.
    - System requires memory and Activity is not visible to user
    - **finish**() method called in Activity
  - Example:
    - SecondActivity is terminated.



Activity Stack

18

18

Activity Life-Cycle

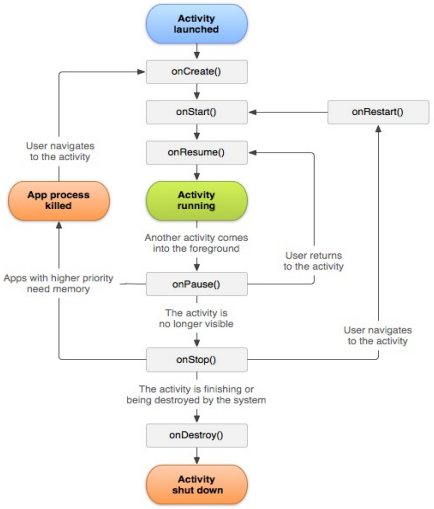
- Activity Lifecycle methods
  - Methods called by the Android runtime when activity state changes.
  - Following functions are called as a result of activity state transition

    void **onCreate**(Bundle savedInstanceState)
    void **onStart**()
    void **onRestart**()
    void **onResume**()
    void **onPause**()
    void **onStop**()
    void **onDestroy**()

19

19

Activity Life-Cycle Flowchart

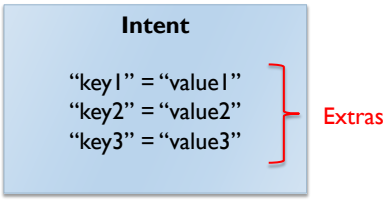


20

20

Activity and Data Flow

- Passing data from one activity to another
  - We can attach extra values to an Intent.
  - Any extra value attached to the Intent will be received by the receiver Activity.
  - Intent Extras are nothing by key, value pairs attached to Intent.
    - blble across process boundaries



21

21

Activity and Data Flow

- Passing data from one activity to another
  - Adding primitives as extras to Intent

```
Intent putExtra(String name, [int, float, double, String] value)
```

  - Adding Object as extra to Intent

```
Intent putExtra(String name, Parcelable p)
```

22

22

Activity and Data Flow

- Passing data from one activity to another
  - Adding values as extra to Intent

```
var intent = Intent(this, SecondActivity::class.java)
intent.putExtra("IntVal", 100)
intent.putExtra("StringVal", "message")
```

Intent
Extras
"IntVal" = 100
"StringVal" = "message"

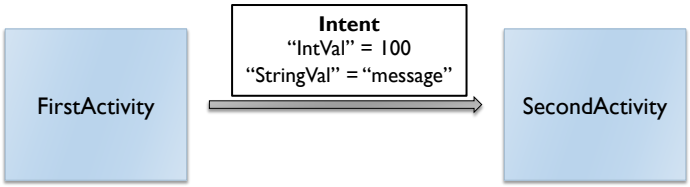
23

23

Activity and Data Flow

- Passing data from one activity to another
  - Launching Activity and passing Intent with Extras

startActivity(intent)



24

24

Activity and Data Flow

- Passing data from one activity to another
  - Getting access to Launching Intent

    Intent **getIntent** ()

    - method of Activity class
    - provides access to the Intent that was used for launching the Activity.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    //get access to intent that launched this Activity
    var launchingIntent = getIntent()
}
```

25

25

Activity and Data Flow

- Passing data from one activity to another
  - Getting access to Launching Intent and data attached to it

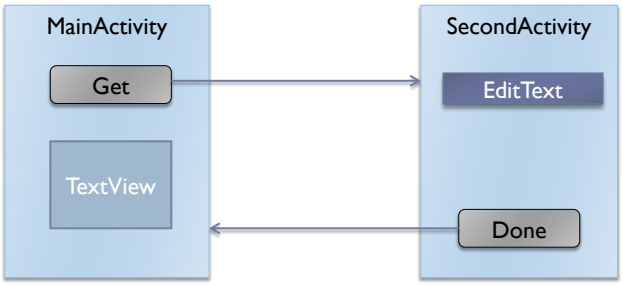
```
//get access to intent that launched this Activity
val launchingIntent = getIntent()
//reading attached data from the launching intent
if (launchingIntent.extras != null) {
    val intVal = launchingIntent.getIntArrayExtra("IntVal")
    val stringVal = launchingIntent.getStringExtra("StringVal")
} else {
    Log.i("MainActivity", "No Extras")
}
```

26

26

Activity and Data Flow

- Returning a Result back to previous Activity
  - Often we want an Activity to return a value back to the Previous Activity



27

27

Activity and Data Flow

- Returning a Result back to previous Activity
  - Start an activity that returns a result value

public void **startActivityForResult** (Intent *intent*, int *requestCode*)

requestCode,	An integer that uniquely identifies the result request made to a sub-activity.

- Method is used in-place of calling startActivity().
- Allows the Launched Activity to return a value back to the Launching Activity.

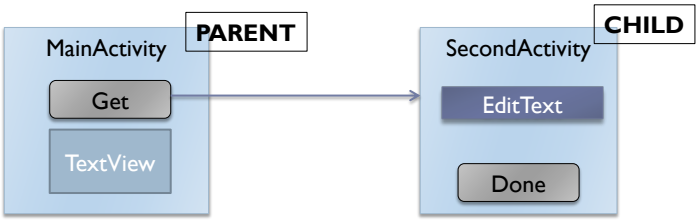
28

28

Activity and Data Flow

- Returning a Result back to previous Activity
  - Start an activity that returns a result value

```
private val SECOND_ACTIVITY_ID = 101
fun launchSecond(view: View) {
    val intent = Intent(this, SecondActivity::class.java)
    startActivityForResult(intent, SECOND_ACTIVITY_ID)
}
```



29

29

Activity and Data Flow

- Returning a Result back to previous Activity
  - Method that allows a child Activity to send an Intent back to the parent Activity.

void **setResult** (int *resultCode*, Intent *data*)

resultCode,	*RESULT_OK*,	User successfully selected/entered the data.
	RESULT_CANCELLED,	User decided not to select/ enter data.
data,	Intent object that contains data to be sent back (attached as extras to the Intent)	

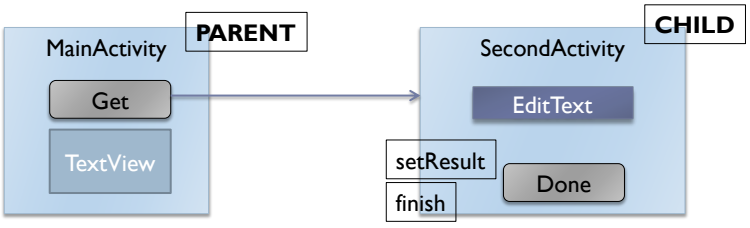
30

30

Activity and Data Flow

- Returning a Result back to previous Activity
  - Child Activity returning data back to Parent

```
val intent = Intent()
intent.putExtra("key", "value")
setIntent(intent)
finish()
```



31

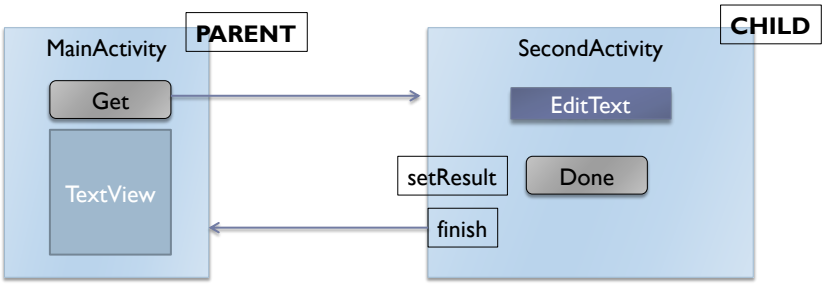
31

Activity and Data Flow

- Returning a Result back to previous Activity
  - Going back to previous Activity (back button).

void **finish**()

- This method terminates the current Activity.
- Results are propagated back to the parent Activity.



32

32

Activity and Data Flow

- Returning a Result back to previous Activity
  - Method that is executed in the Parent Activity when the Child calls **setResult**() and then **finish**()

protected void **onActivityResult** (int *requestCode*, int *resultCode*, Intent *data*)

requestCode,	paramater that was passed to the startActivityForResult, uniquely identify the request.
resultCode,	Result value as set by the sub-activity using the setResult() method. RESULT_OK, the operation ended successfully. RESULT_CANCELED, child activity failed.
data,	Intent object holding the data for parent.

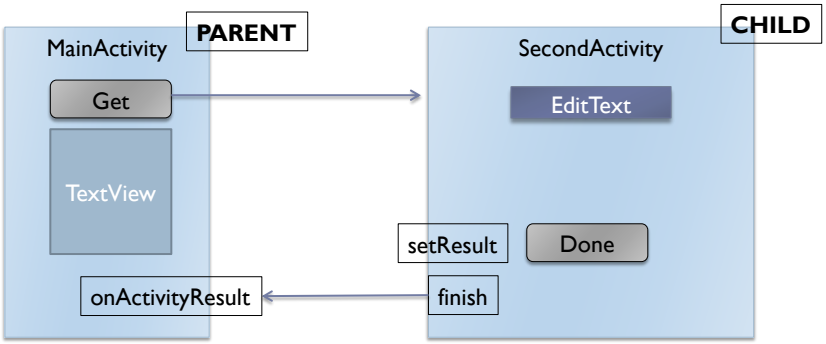
to

33

33

Activity and Data Flow

- Returning a Result back to previous Activity
  - Method that is executed in the Parent Activity when the Child calls **setResult**() and then **finish**()



34

34

Activity State

Amit Gulati, amit.gulati@gmail.com

35

Activity State

- Activity is terminated in the following Scenarios
  - System runs low on memory
    - As a result it may terminate Activities that are not visible to the user.
  - Configuration Change
    - Device Orientation Change
- Activity Termination Scenario must be handled by the programmer.
  - Preserve Activity instance data before Activity terminated
  - Android system provides a mechanism that allows the programmer to save the state of an Activity (before it is terminated), so that it can be later restored.

36

36

Activity State

- Preserve Activity data before Activity terminated
  - Methods that is called before an Activity is killed because of system killing a process.

void **onSaveInstanceState** (Bundle *outState*)

outState, Bundle object can be used to write key-value pair that will be made available to the activity in the onCreate() method, if it is restarted by the system.

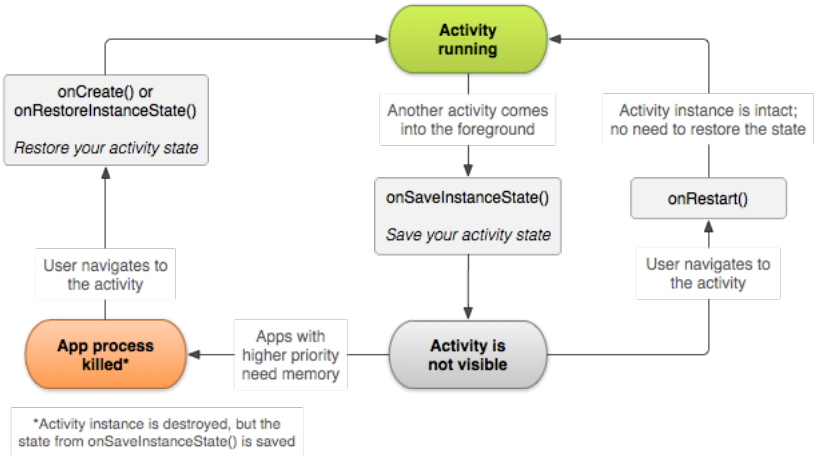
```
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    outState.putString("key", "value")
    outState.putInt("key", 0)
}
```

37

37

Activity State

- Activity Termination Scenario



38

38

Activity State

- Retrieve Activity state when Activity re-starts
  - When the system restarts the Activity that was killed, the bundle object is passed to the Activity in the **onCreate**() method.
  - Example

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    if (savedInstanceState != null) {
        val stringVal = savedInstanceState.getString("StringVal")
        var intVal = savedInstanceState.getInt("IntVal")
    }
```

39

39

Activity State

- Handling configuration changes yourself
  - In some cases you may not want the Activity to be terminated for a configuration change.
  - You can declare that your Activity will handle configuration changes itself.
  - In the manifest file add the following attribute to the <activity> registration

android:configChanges

"orientation"
"screenSize"
"keyboardHidden"
"layoutDirection"

40

40

Activity State

- Handling configuration changes yourself
  - Implement the onConfigurationChanged function

```
public void onConfigurationChanged(Configuration newConfig)
```

41

41

Activity State

- Preserving Objects
  - Retain an object during configuration change using Activity API
    - Override the method, and return reference to object you want to preserve

```
public Object onRetainNonConfigurationInstance()
```

    - Once new Activity created, get access to the retained object

```
public Object getLastNonConfigurationInstance()
```

42

42