Mobile Application Development

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

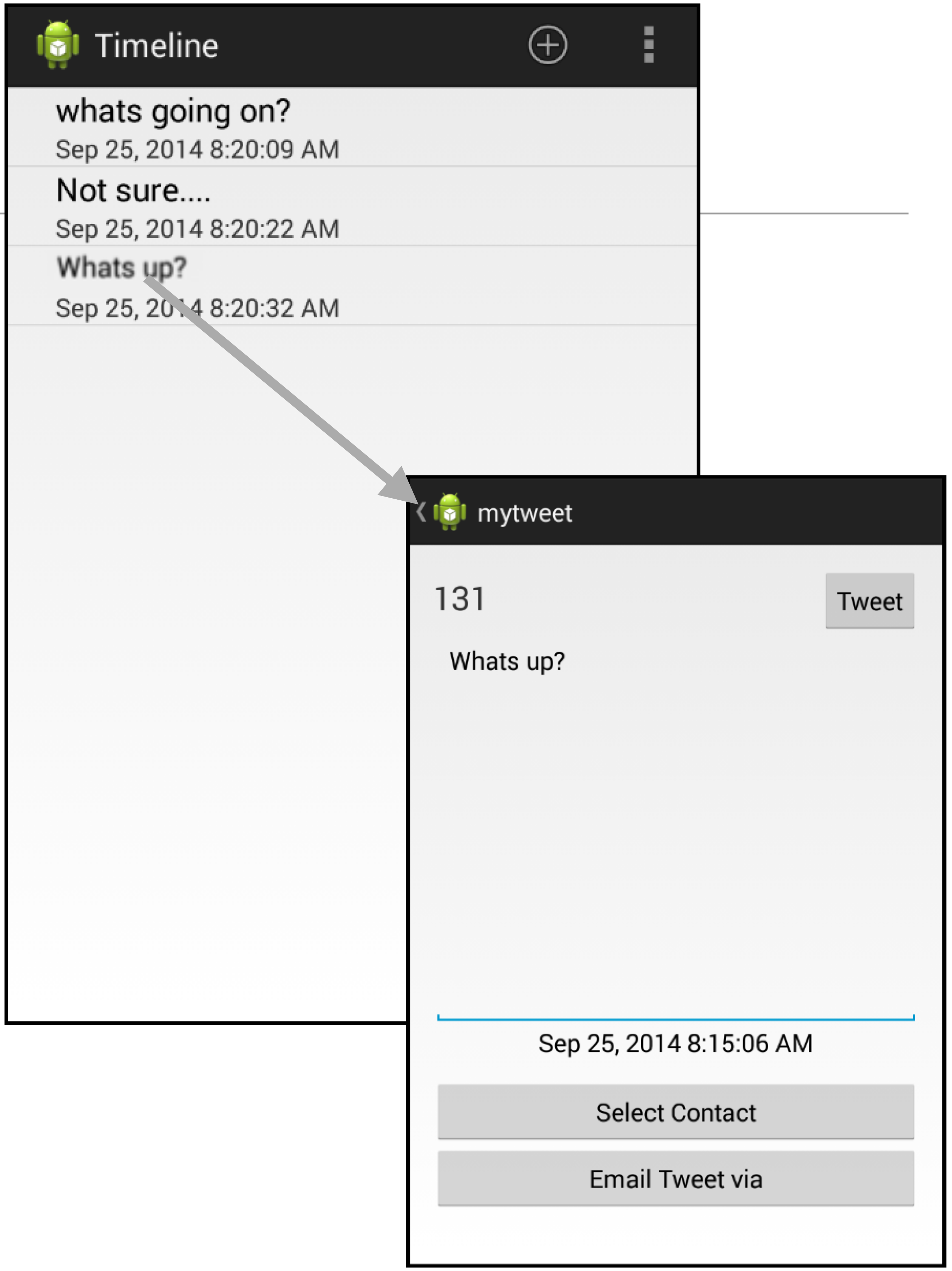
http://www.wit.ie

http://elearning.wit.ie

Activities

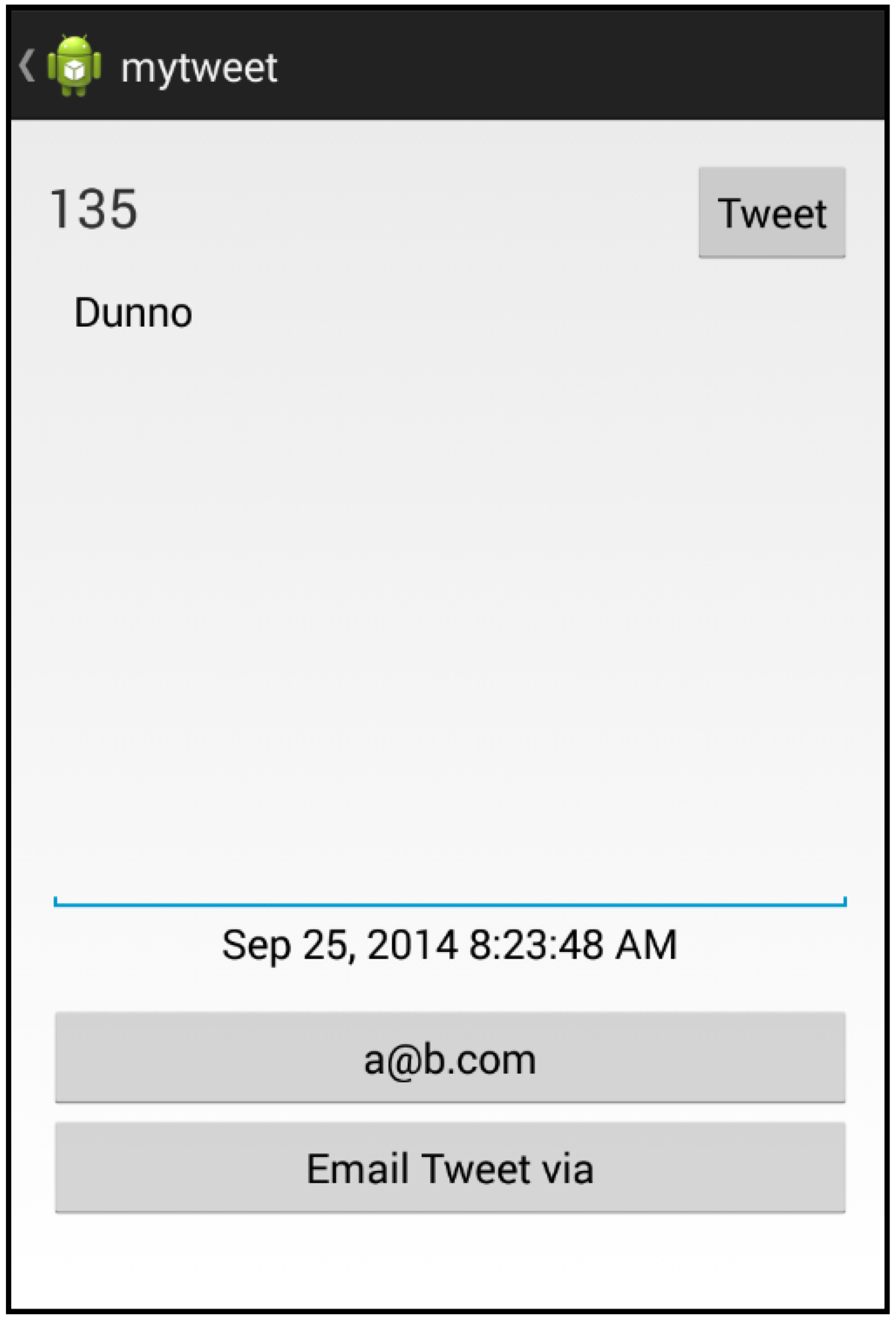
Activities

- Purpose of an Activity
- Activity Stack
- Activity Lifecycle
- Creating an Activity
- Specifying the UI
- Manifest
- Starting Activities

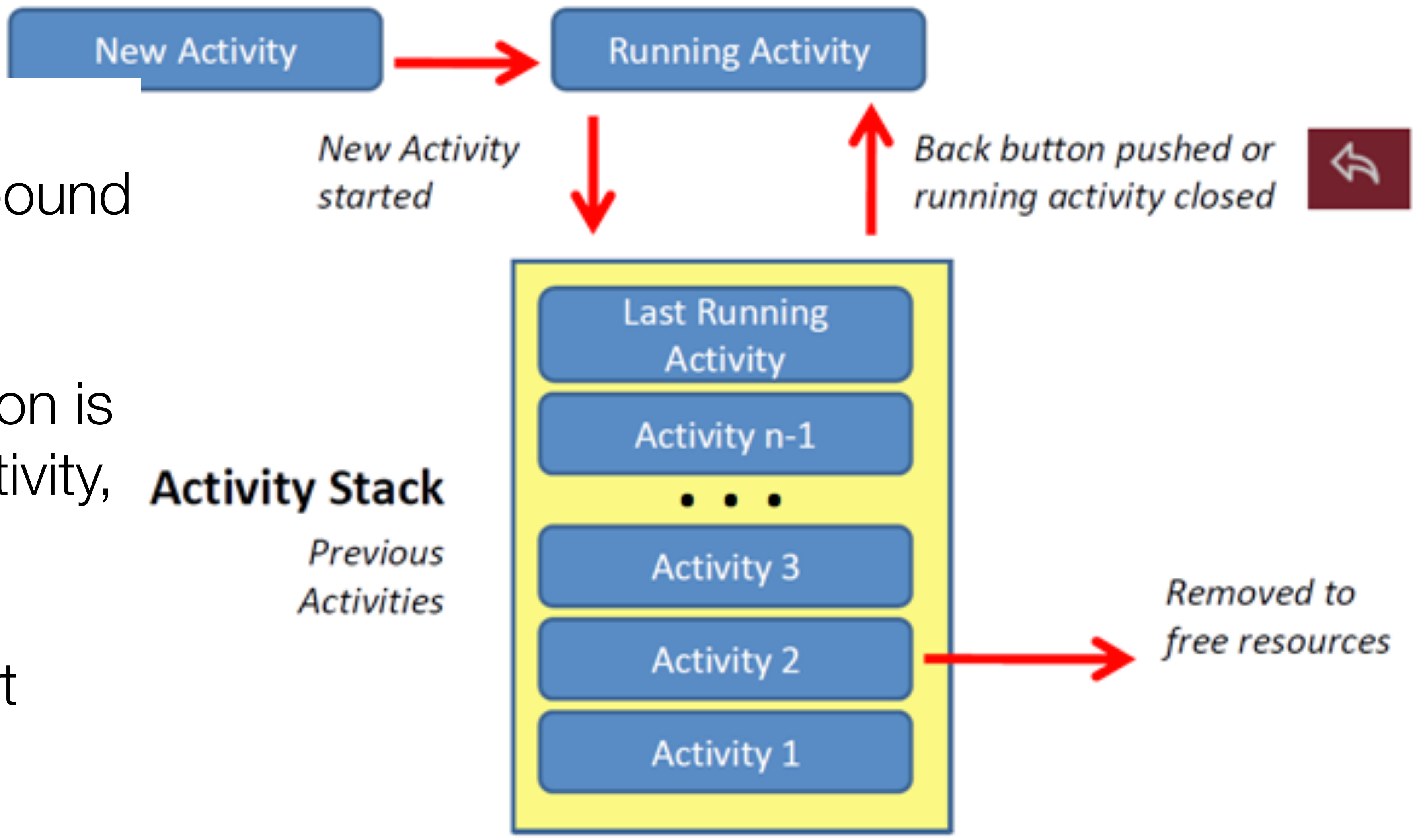


Activities

- An application component that provides a screen with which users can interact in order to do something
- Each activity is given a window in which to draw its user interface.
- The window typically fills the screen.



The Activity Stack



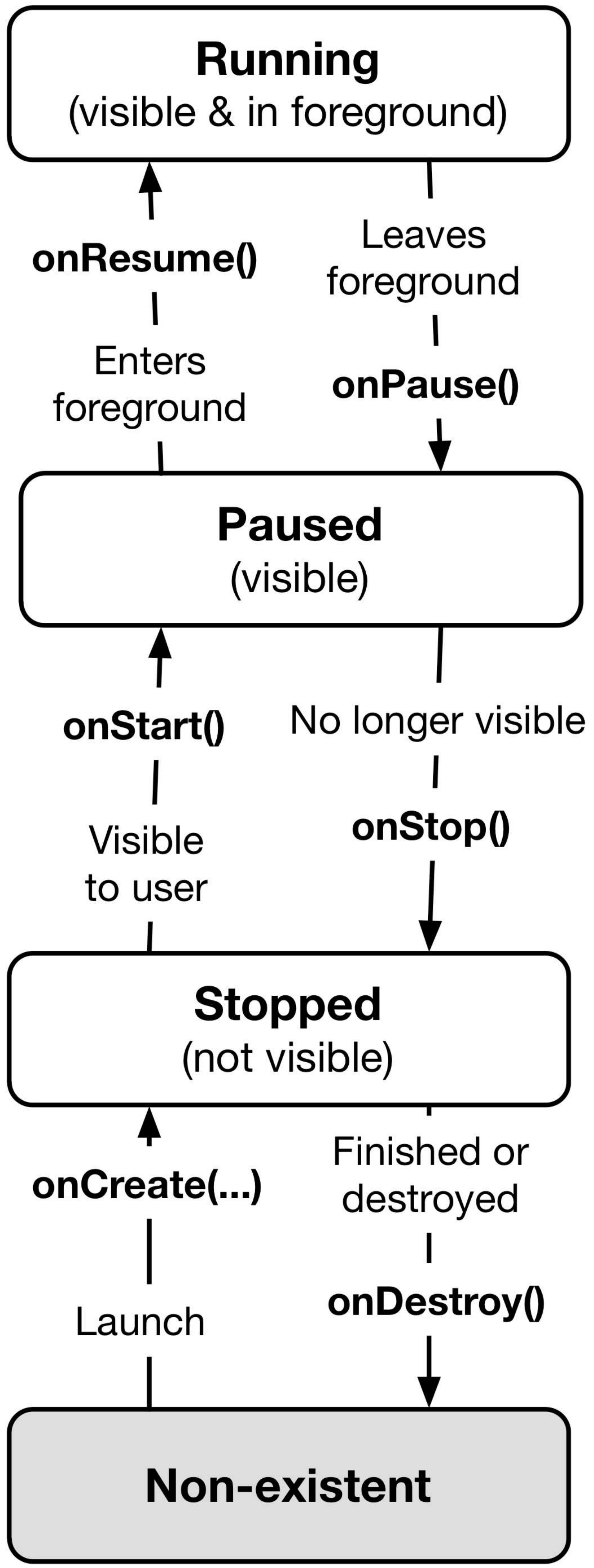
- An application consists of multiple activities loosely bound to each other.
- One activity in an application is specified as the "main" activity, presented on first launch
- Each activity can then start another activity to perform different actions.
- When an activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack"). .
- The back stack abides to the basic "last in, first out" stack mechanism, - when Back button pressed, it is popped from the stack that previous activity resumes.

Activity Lifecycle

- When an activity is stopped it is notified of this change in state through the activity's lifecycle callback methods:
  - Create,
  - Stop
  - Resume
  - Destroy
- These callback provide the opportunity to perform specific work that's appropriate to that state change.

Activity Lifecycle

- When stopped, your activity should release any large objects, such as network or database connections.
- When the activity resumes reacquire the necessary resources and resume actions that were interrupted.
- These state transitions are all part of the activity lifecycle.



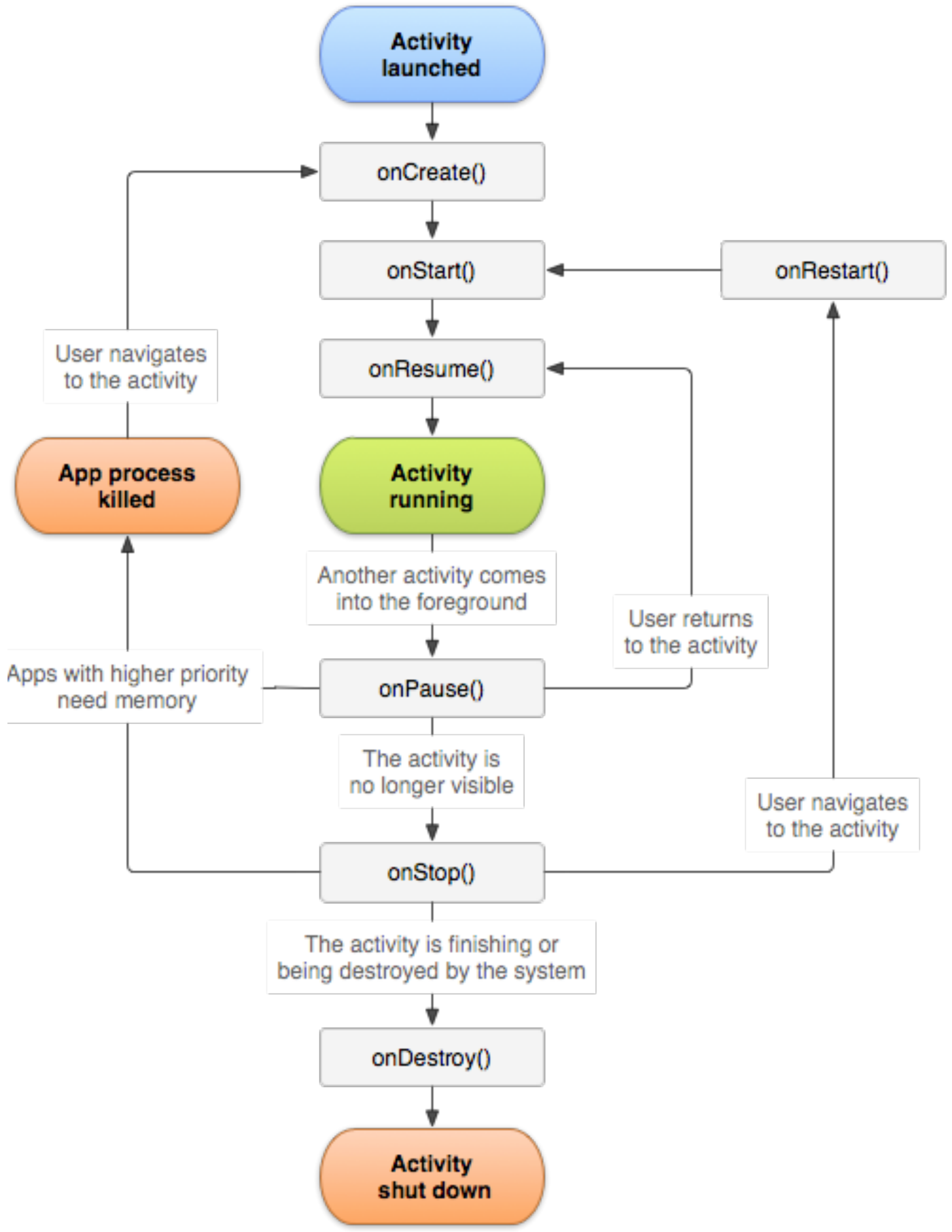
Creating an Activity

- Create a subclass of Activity and implement callback methods that the system calls when the activity transitions between various states of its lifecycle

- The two primary call-backs to implement are:

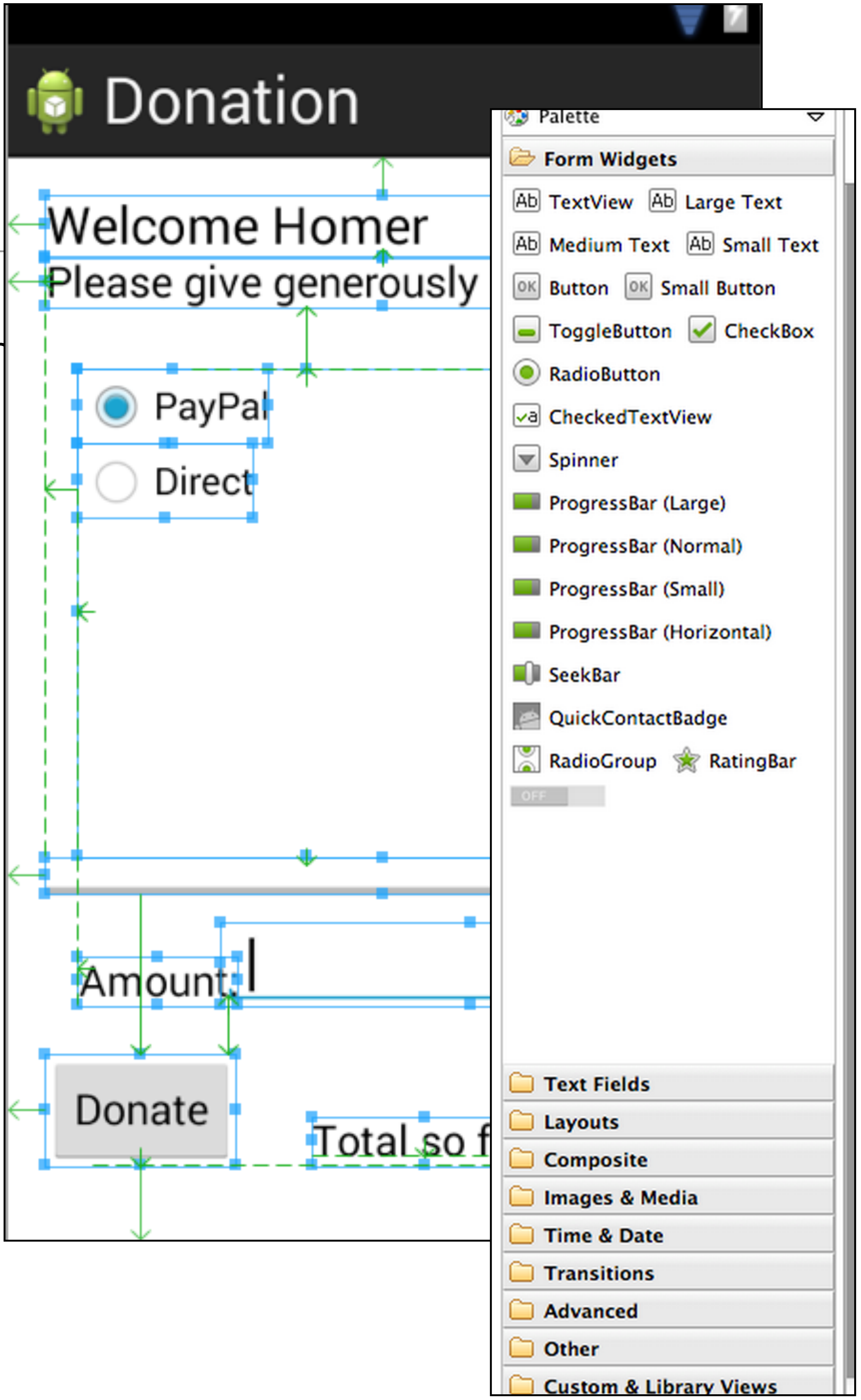
  - **onCreate()** : The system calls this when creating your activity. Initialize the essential components of your activity, usually including a call call setContentView() to define the layout for the user interface.

  - **onPause()** : The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed). Commit any changes that should be persisted beyond the current user session (because the user might not come back).


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

Implementing the UI

- The user interface for an activity is provided by a hierarchy of views—objects derived from the View class.
- Each view controls a particular rectangular space within the activity's window and can respond to user interaction. They consist of:
  - **Layouts:** views derived from ViewGroup that provide a unique layout model for its child views (e.g. linear. grid or relative layout).
  - **Widgets:** standard views that provide a visual (and interactive) elements for the screen, such as a button, text field, checkbox, or just an image.



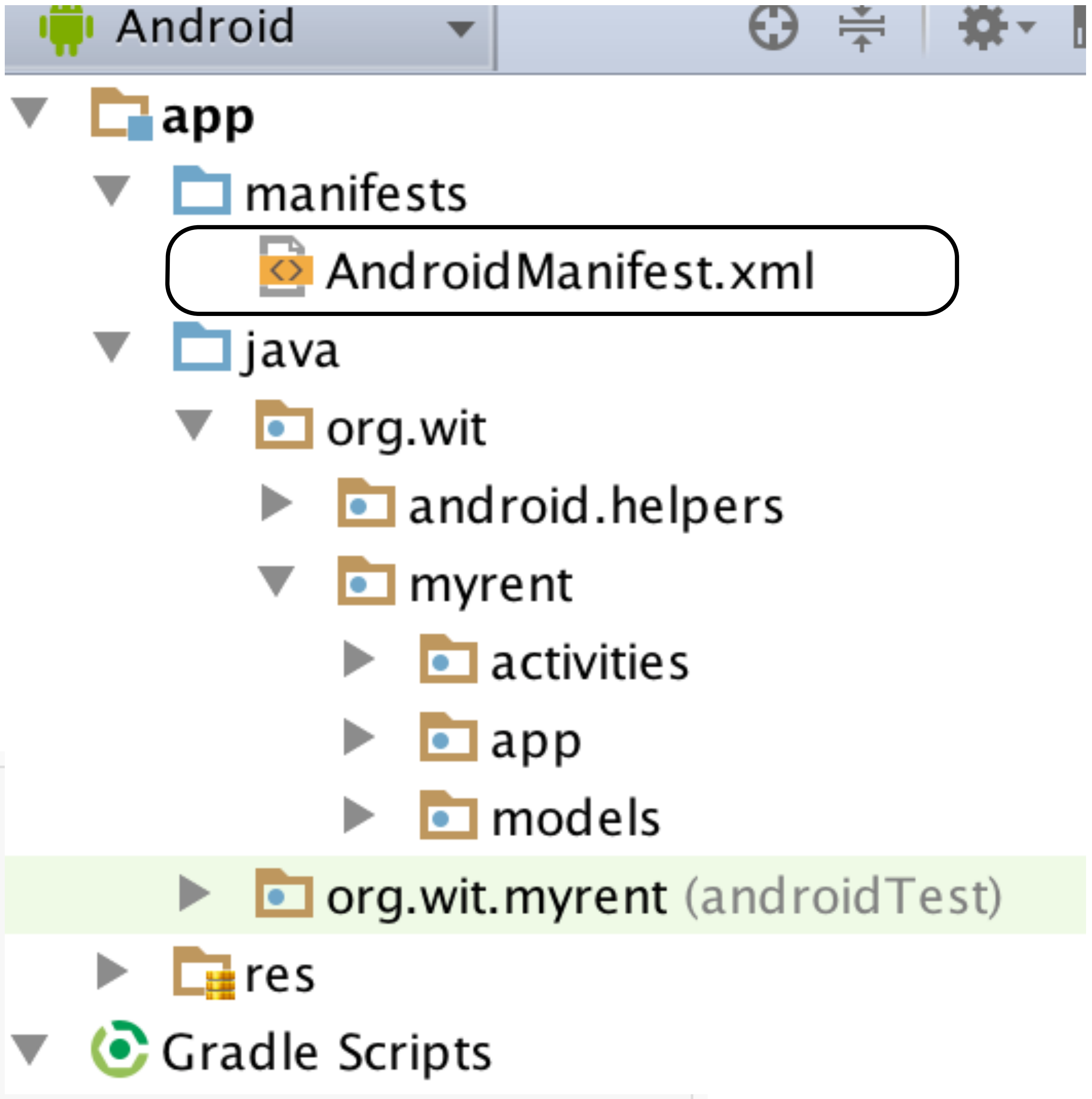
XML Layouts

- The most common way to define a layout using views is with an XML layout file saved in your application resources.
- Enables the design of your user interface separately from the source code that defines the activity's behavior.
- Set the layout as the UI for your activity with setContentView(), passing the resource ID for the layout.
- Or create new Views in your activity code and build a view hierarchy by inserting new Views into a ViewGroup, then use that layout by passing the root ViewGroup to setContentView().

```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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".Donate" >

    <TextView
        android:id="@+id/donateTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:text="@string/donateTitle"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <TextView
        android:id="@+id/donateSubtitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/donateTitle"
        android:text="@string/donateSubtitle"
        android:textAppearance="?android:attr/textAppearanceMedium" />

</RelativeLayout>
```

Declaring the activity in the manifest

- Declare your activity in the manifest file in order for it to be accessible to the system.

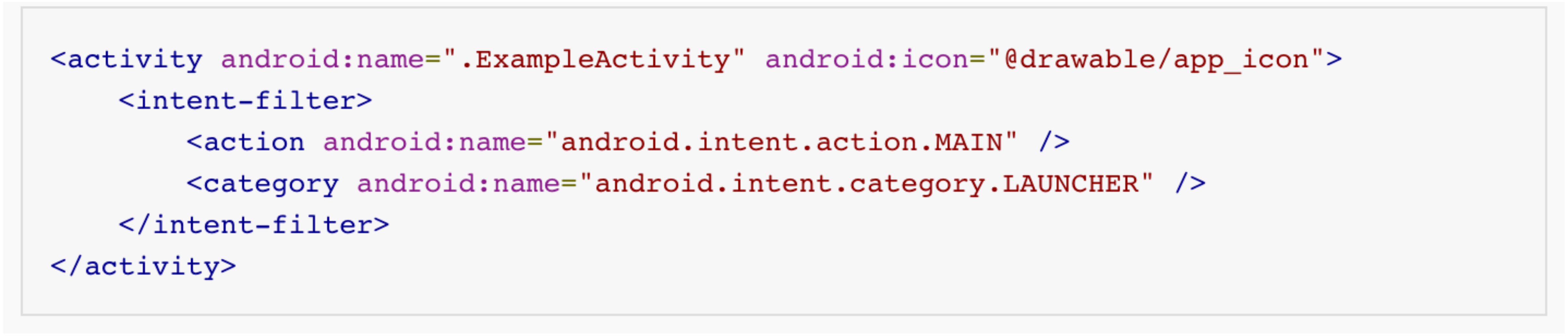


```
<manifest ... >
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application ... >
  ...
</manifest >
```

Using intent filters

- An <activity> element can also specify various intent filters in order to declare how other application components may activate it.

- The main activity for an will require an intent filter that declares the activity responds to the "main" action, and should be placed in the "launcher" category:



```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

- The <action> element specifies that this is the "main" entry point to the application.

- The <category> element specifies that this activity should be listed in the system's application launcher (to allow users to launch this activity).

Implicit Intents

- The Activity can respond to “implicit” intents that are delivered from other applications

- To do this, define additional intent filters for the activity in the manifest.

- Include an <intent-filter> that includes an <action> element and, optionally, a <category> element and/or a <data> element.

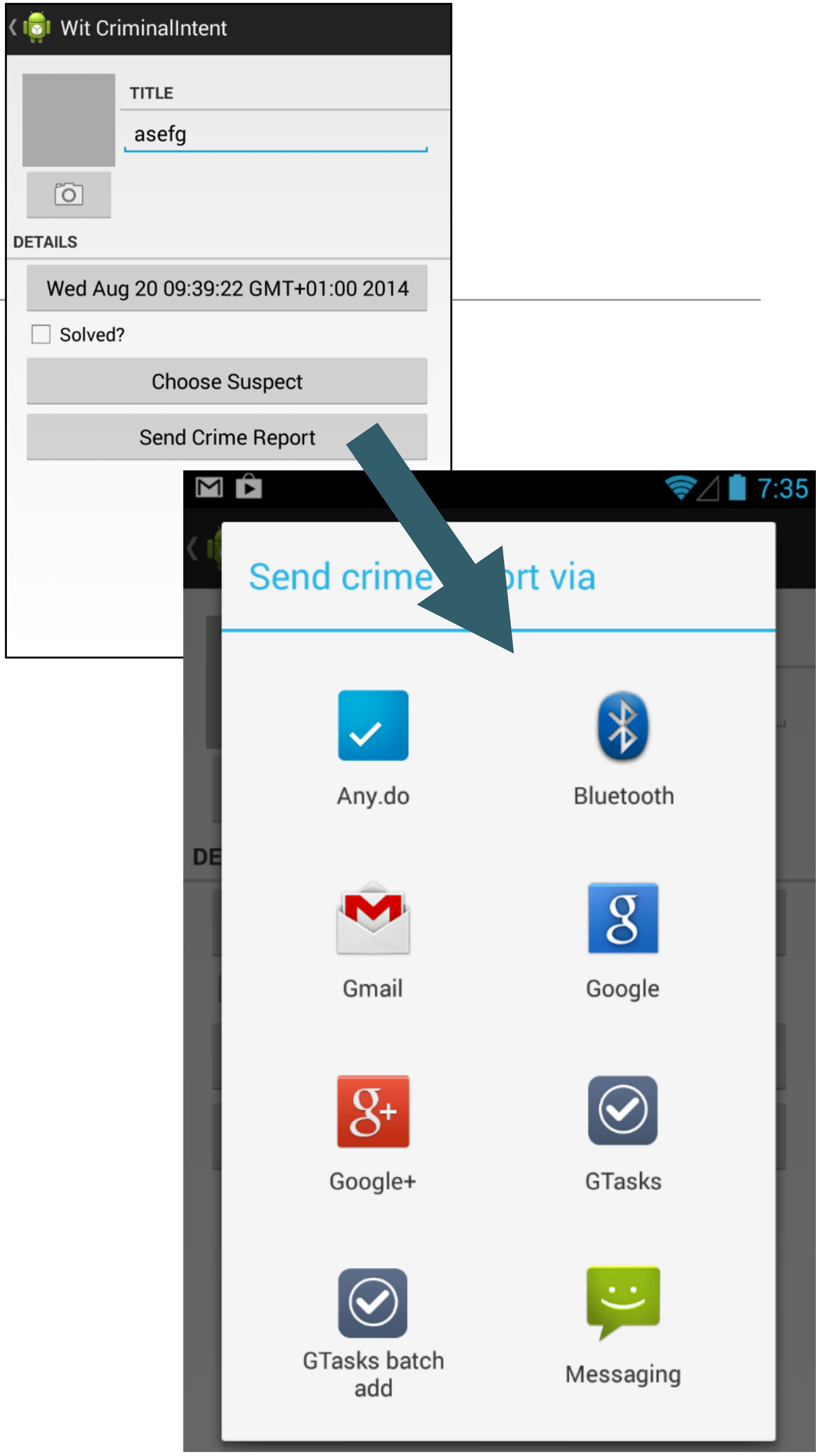
Starting an Activity

- Start another activity by calling startActivity(), passing it an Intent that describes the activity you want to start.
    - The intent specifies either the exact activity you want to start
    - Or describes the type of action you want to perform (and the system selects the appropriate activity for you, which can even be from a different application).
- An intent can also carry small amounts of data to be used by the activity that is started.

```
Intent intent = new Intent(this, SignInActivity.class);
startActivity(intent);
```

Implicit Intents

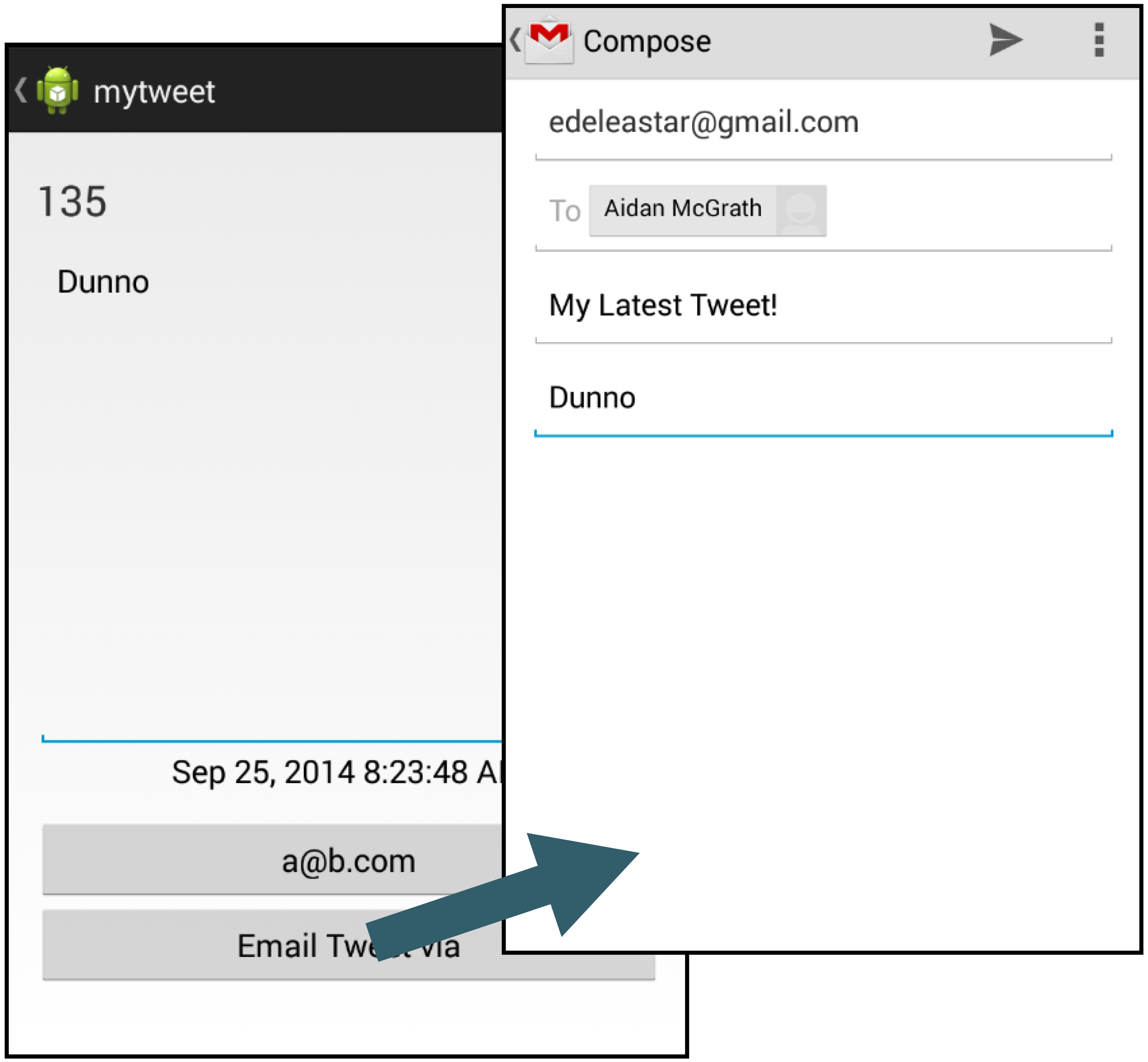
- An app may also want to perform some action, such as send an email, text message, or status update,.
- If the application does not have its own activities to perform such actions, leverage the activities provided by other applications on the device, which have declared (using intent filers) that can perform the actions.
- If there are multiple activities that can handle the intent, then the user can select which one to use.



TITLE
asefg
DETAILS
Wed Aug 20 09:39:22 GMT+01:00 2014

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

- The EXTRA_EMAIL extra added to the intent is a string array of email addresses to which the email should be sent.
- When an email application responds to this intent, it reads the string array provided in the extra and places them in the "to" field of the email composition form.
- In this situation, the email application's activity starts and when the user is done, your activity resumes.


mytweet

135

Dunno

Sep 25, 2014 8:23:48 A

a@b.com

Email Tweet via

Compose

edeleastar@gmail.com

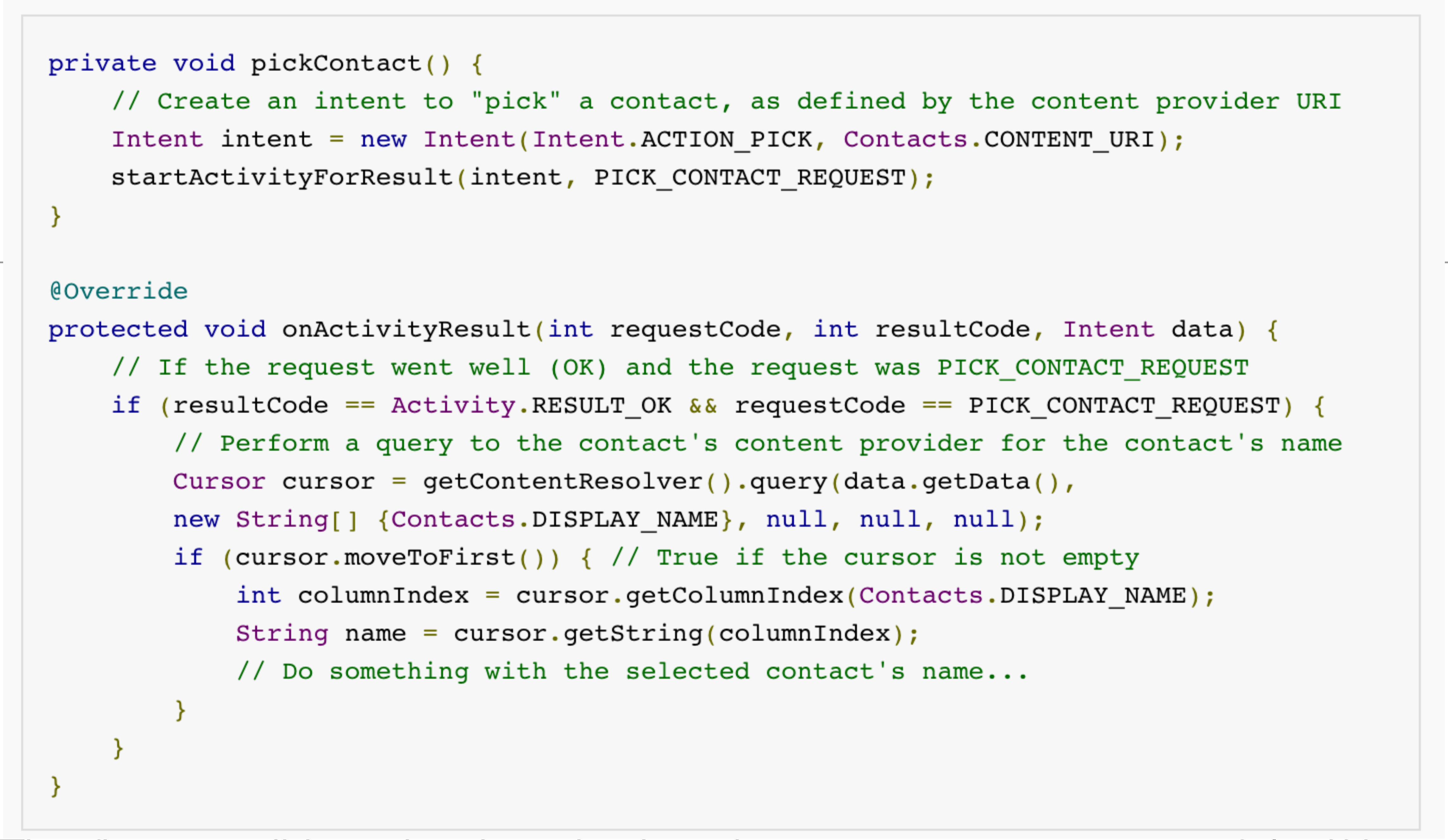
To Aidan McGrath

My Latest Tweet!

Dunno

Starting an activity for a result

- Sometimes, you might want to receive a result from the activity that you start.
- In that case, start the activity by calling startActivityForResult() (instead of startActivity()).
- To then receive the result from the subsequent activity, implement the onActivityResult() callback method.
- When the subsequent activity is done, it returns a result in an Intent to your onActivityResult() method.



```java
private void pickContact() {
    // Create an intent to "pick" a contact, as defined by the content provider URI
    Intent intent = new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);
    startActivityForResult(intent, PICK_CONTACT_REQUEST);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // If the request went well (OK) and the request was PICK_CONTACT_REQUEST
    if (resultCode == Activity.RESULT_OK && requestCode == PICK_CONTACT_REQUEST) {
        // Perform a query to the contact's content provider for the contact's name
        Cursor cursor = getContentResolver().query(data.getData(),
        new String[] {Contacts.DISPLAY_NAME}, null, null, null);
        if (cursor.moveToFirst()) { // True if the cursor is not empty
            int columnIndex = cursor.getColumnIndex(Contacts.DISPLAY_NAME);
            String name = cursor.getString(columnIndex);
            // Do something with the selected contact's name...
        }
    }
}
```

- The first condition checks whether the request was successful—if it was, then the resultCode will be RESULT_OK—and whether the request to which this result is responding is known—in this case, the requestCode matches the second parameter sent with startActivityForResult().

- From there, the code handles the activity result by querying the data returned in an Intent

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit