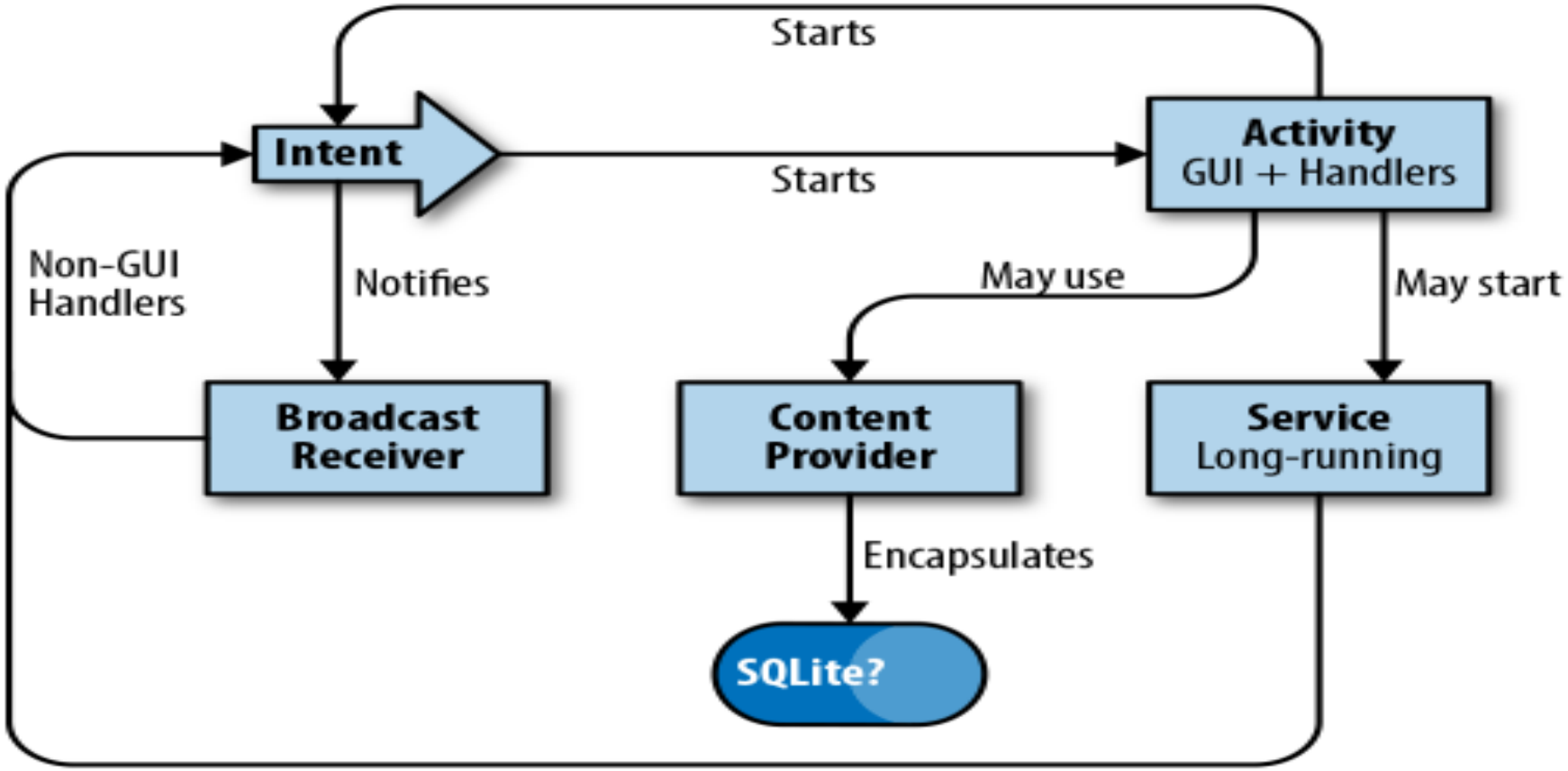
Chapter three
ANDROID Applications Component

- Application components are the essential building blocks of an Android application.
- There are following four main components that can be used within an Android application:

Components	Description
Activities	They dictate the UI and handle the user interaction to the smartphone screen
Services	They handle background processing associated with an application.
Broadcast Receivers	They handle communication between Android OS and applications.
Content Providers	They handle data and database management issues.



Android application components

Introduction to Activity

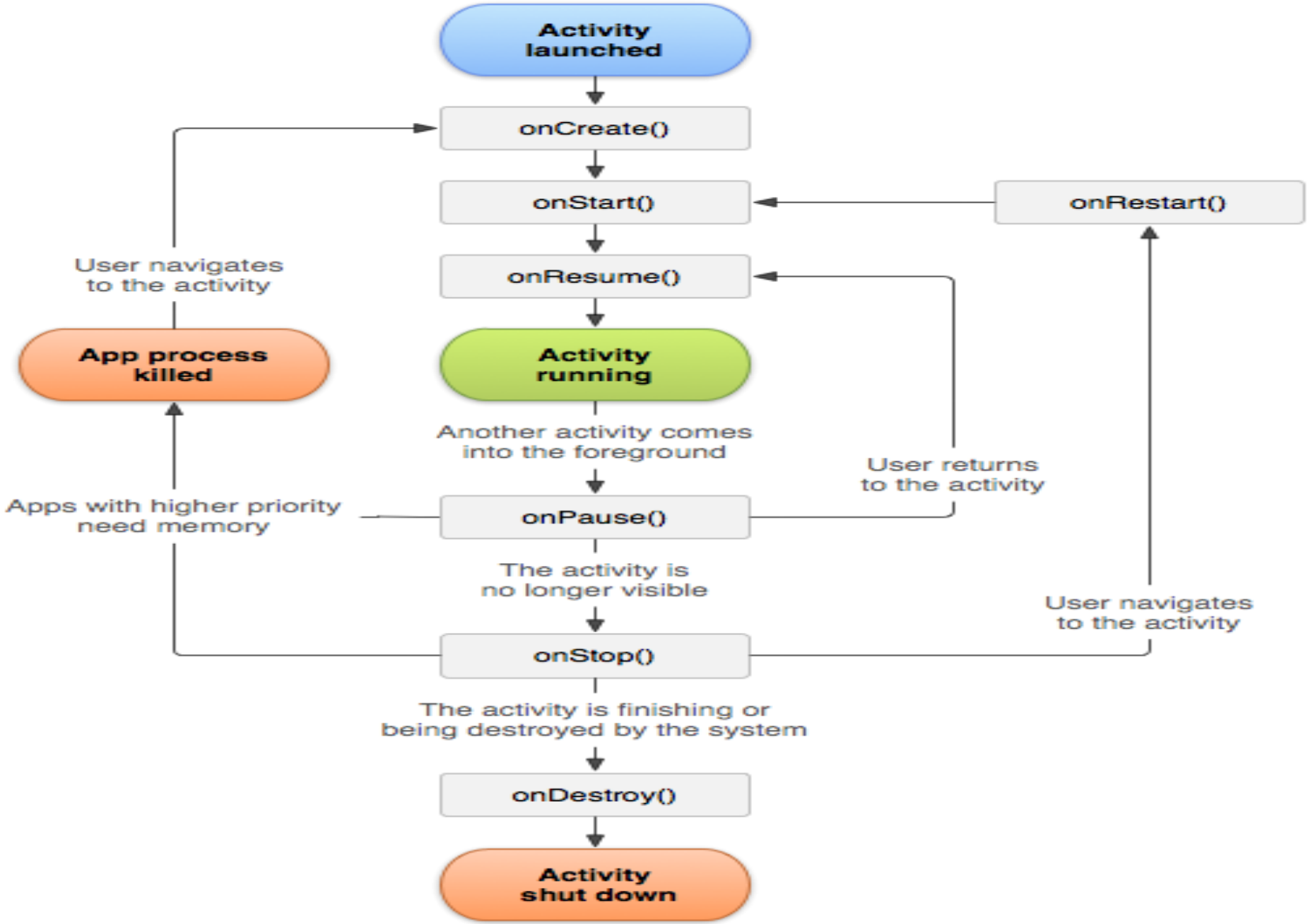
- The Activity class is a crucial component of an Android app.
- the way activities are launched and put together is a fundamental part of the platform's application model.
- An activity **provides the window in which the app draws its UI**. This window typically fills the screen, but may be smaller than the screen and float on top of other windows. Generally, one activity implements one screen in an app.

- An activity represents a single screen with a user interface.
- For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and one for reading emails.
- If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

- For your app to be able to use activities, you must declare the activities, and certain of their attributes, in the manifest.
- The only required attribute for this element is android:name, which specifies the class name of the activity. You can also add attributes that define activity characteristics such as label, icon, or UI theme.

Understanding the Android Activity Life Cycle

- Android apps do not have a “main” method; you need to understand how they get started and how they stop or get stopped.
- The class android.app.Activity provides a number of well-defined life-cycle methods
  that are called when an application is started, suspended, restarted, and so on, as well as a method you can call to mark an Activity as finished.


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
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

onCreate()

- The onCreate() callback is compulsory in all Android applications.
- It is the first method called when we launch an activity from the home screen or intent.
- In other words, it is a default callback that is automatically created when you create a new activity.
- protected void onCreate (**Bundle** savedInstanceState)
- Called when the activity is starting. This is where most initialization should go: calling setContentView(int) to inflate the activity's UI, using findViewById(int) to programmatically interact with widgets in the UI to retrieve cursors for data being displayed, etc.

- You can call finish() from within this function, in which case onDestroy() will be immediately called after onCreate(Bundle) without any of the rest of the activity lifecycle (onStart(), onResume(), onPause(), etc) executing.

```
private static final String TAG = "MainActivity";
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toast.makeText(this, "onCreate MainActivity",
Toast.LENGTH_SHORT).show();
    Log.d(TAG, "onCreate MainActivity");
}
```

onStart()

- Called after onCreate(Bundle) — or after onRestart() when the activity had been stopped, but is now again being displayed to the user.
- protected void onStart ()
- It will usually be followed by onResume(). This is a good place to begin drawing visual elements, running animations, etc.
- You can call finish() from within this function, in which case onStop() will be immediately called after onStart() without the lifecycle transitions in-between (onResume(), onPause(), etc) executing.

Here is how onStart() is implemented.

```
@Override
protected void onStart() {
    Toast.makeText(this, "onStart MainActivity",
Toast.LENGTH_SHORT).show();
        Log.d(TAG, "onStart MainActivity");
    super.onStart();
      }
```

onResume()

- Once onStart() is called, onResume() is immediately invoked.
- Every component associated with this activity is brought to the foreground state. The activity is now considered interactive.
- protected void onResume ()
- Called after onRestoreInstanceState(Bundle), onRestart(), or onPause().
- This is usually a hint for your activity to start interacting with the user, which is a good indicator that the activity became active and ready to receive input.

onPause()

- onPause() is called when the user switches to another activity or a multi-window mode application.
- At this point, the activity has lost focus and is running in the background.
- protected void onPause ()
- Called as part of the activity lifecycle when the user no longer actively interacts with the activity,
- but it is still visible on screen. The counterpart to onResume().
- When onPause() is called, you might release some resources from memory. However, make sure that you initialize them again during the onResume() callback.

onStop()

- At this point, most of the activity processes have been stopped. However, the activity is still running in the background.
- This life-cycle usually occurs after the onPause() method is executed due to the user switching to other activities or pressing the home button.
- protected void onStop ()
- Called when you are no longer visible to the user. You will next receive either onRestart(), onDestroy(), or nothing, depending on later user activity. This is a good place to stop refreshing UI, running animations and other visual things.

- When a user opens it again, the application will not reload all instances. Instead, it will retrieve them from memory. This includes UI components such as the TextViews.

onRestart()

- Since the activity’s states still exist, the onRestart() method can be called when the user restarts the activity. This means the activity will go back to the main screen and the user can resume interacting with its components.
- when the onRestart() method is executed, the activity will resume by executing the onStart() then onResume(). Since the onCreate() function is only called once in an activity’s life-cycle.

onDestroy()

- This is the final callback that the activity will receive when it is stopped.
- The method is called when there is a change in the configuration states such as screen rotation or language settings.
- The Android system will destroy the activity, then recreate it with the set configurations.

Android Intent

- **Android Intent** is the *message* that is passed between components such as activities, content providers, broadcast receivers, services etc.
- It is generally used with startActivity() method to invoke activity, broadcast receivers etc.

Android intents are mainly used to:

- Start the service
- Launch an activity
- Display a web page
- Display a list of contacts
- Broadcast a message
- Dial a phone call etc.

Types of Android Intents

- There are two types of intents in android: implicit and explicit.

1) Implicit Intent

Implicit Intent doesn't specifiy the component. In such case, intent provides information of available components provided by the system that is to be invoked.

For example, you may write the following code to view the webpage.

```
Intent intent=new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.javatpoint.com"));
startActivity(intent);
```

2) Explicit Intent

- **Explicit Intent** specifies the component. In such case, intent provides the external class to be invoked.

```
Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
startActivity(i);
```

- Android Explicit intent specifies the component to be invoked from activity. In other words, we can call another activity in android by explicit intent.
- We can also pass the information from one activity to another using explicit intent.
- Here, we are going to see an example to call one activity from another and vice-versa.

Android calling one activity from another activity

```
Intent i = new Intent(getApplicationContext(), SecondActivity.class);
      i.putExtra("Value1", "Android By Javatpoint");
      i.putExtra("Value2", "Simple Tutorial");
            startActivity(i);
```

The second activity access the data using

```
Bundle extras = getIntent().getExtras();
String value1 = extras.getString("Value1");
String value2 = extras.getString("Value2");
```

Bundles are used with intent and values are sent and retrieved in the same fashion, as it is done in the case of Intent. It depends on the user what type of values the user wants to pass, but bundles can hold all types of values (int, String, boolean, char) and pass them to the new activity

- Android uses **ACTION_SEND** event of **android.content.Intent** class to send data from one activity to another and from current activity to outside the application. Intent class needs to specify the data and its type which is to be share.

Android Fragments

- **Android Fragment** is the part of activity, it is also known as sub-activity.
- There can be more than one fragment in an activity. Fragments represent multiple screen inside one activity.
- Android fragment lifecycle is affected by activity lifecycle because fragments are included in activity.

Android service

- **Android service** is a component that is *used to perform operations on the background* such as playing music, handle network transactions, interacting content providers etc. It doesn't has any UI (user interface).
- The service runs in the background indefinitely even if application is destroyed.
- Moreover, service can be bounded by a component to perform interactivity and inter process communication (IPC).
- The android.app.Service is subclass of ContextWrapper class.

- Life Cycle of Android Service
- There can be two forms of a service. The lifecycle of service can follow two different paths: started or bound.

1.Started

2.Bound

- 1) Started Service
- A service is started when component (like activity) calls **startService()** method, now it runs in the background indefinitely. It is stopped by **stopService()** method. The service can stop itself by calling the **stopSelf()** method.
- 2) Bound Service
- A service is bound when another component (e.g. client) calls **bindService()** method. The client can unbind the service by calling the **unbindService()** method.
- The service cannot be stopped until all clients unbind the service.

Broadcast Receivers

- Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or Intents.

Creating the Broadcast Receiver

- A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and overriding the onReceive() method where each message is received as a **Intent** object parameter.

```
public class MyReceiver extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      Toast.makeText(context, "Intent Detected.",
Toast.LENGTH_LONG).show();
   }
}
```

Registering Broadcast Receiver

- An application listens for specific broadcast intents by registering a broadcast receiver in *AndroidManifest.xml* file.
- Consider we are going to register *MyReceiver* for system generated event ACTION_BOOT_COMPLETED which is fired by the system once the Android system has completed the boot process.

```
<receiver android:name="MyReceiver">
        <intent-filter>
        <action
android:name="android.intent.action.BOOT_COMPLETED">
        </action>
    </intent-filter>
    </receiver>
```

Sr.No	Event Constant & Description
1	android.intent.action.BATTERY_CHANGED Sticky broadcast containing the charging state, level, and other information about the battery.
2	android.intent.action.BATTERY_LOW Indicates low battery condition on the device.
3	android.intent.action.BATTERY_OKAY Indicates the battery is now okay after being low.
6	android.intent.action.CALL Perform a call to someone specified by the data.
7	android.intent.action.CALL_BUTTON The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.
8	android.intent.action.DATE_CHANGED The date has changed.
9	android.intent.action.REBOOT Have the device reboot.

Android - Content Providers

- A content provider component supplies data from one application to others on request.
- Such requests are handled by the methods of the ContentResolver class.
- A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.

ContentProvider

- **onCreate()** This method is called when the provider is started.
- **query()** This method receives a request from a client. The result is returned as a Cursor object.
- **insert()**This method inserts a new record into the content provider.
- **delete()** This method deletes an existing record from the content provider.
- **update()** This method updates an existing record from the content provider.
- **getType()** This method returns the MIME type of the data at the given URI.

Thank you!!