Android Activity

- An Android activity is **one screen**of the Android app's user interface.

  or

- An activity **provides the window**in which the app draws its UI.
-
- An Android app may contain **one or more activities**, meaning one or more screens.
- The Android app **starts by showing the main activity**, and from there the app may make it possible to open additional activities.
- ***main activity,***which is the first screen to appear when the user launches the app
- Each activity can **then start another activity**in order to perform different actions.
- One activity implements one screen in an app.
- Implement an activity as a subclass of the **Activity class.**
- To use activities in your app, you must **register information**about them in the app's **manifest**, and you must **manage activity lifecycles**appropriately.
- **Creating Activities**
  - o Extend Activity to create a new Activity class

```
packagecom.paad.activities;
importandroid.app.Activity;
importandroid.os.Bundle;
public class MyActivity extends Activity
{
 /** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
        }
 }
```

- The base Activity class presents an empty screen. So, create the UI with
    - Fragments,
    - layouts, and
    - Views.
- **Views**- UI controls that display data and provide user interaction.
- **Layout classes** (View Groups) - contain multiple Views to help to layout UIs.
- **Fragments**- to encapsulate segments of your UI (for dynamic interfaces that can be rearranged to optimize your layouts for different screen sizes and orientations)
- To assign a UI to an Activity, call setContentView from the onCreate method of your Activity.

```
@Override
public void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
}
```

- passing a resource ID for a layout defi ned in an external resource
- **Activity Registration in Manifest file**
    - Add a new activity tag **within the application node of the manifest**;
    - The **activity tag includes attributes**for metadata, such as the label, icon, required permissions, and themes used by the Activity.

    ```
    <activity android:label=”@string/app_name”
    android:name=”.MyActivity”>
    </activity>
    ```

- **Intent - Filter (for communication within the app and with other apps)**
    - Each Intent Filter defines one or more **actions and categories** that your Activity supports.
    - If an Activity to be available from the application launcher, it must include an Intent Filter listening for the MAIN action and the LAUNCHER category

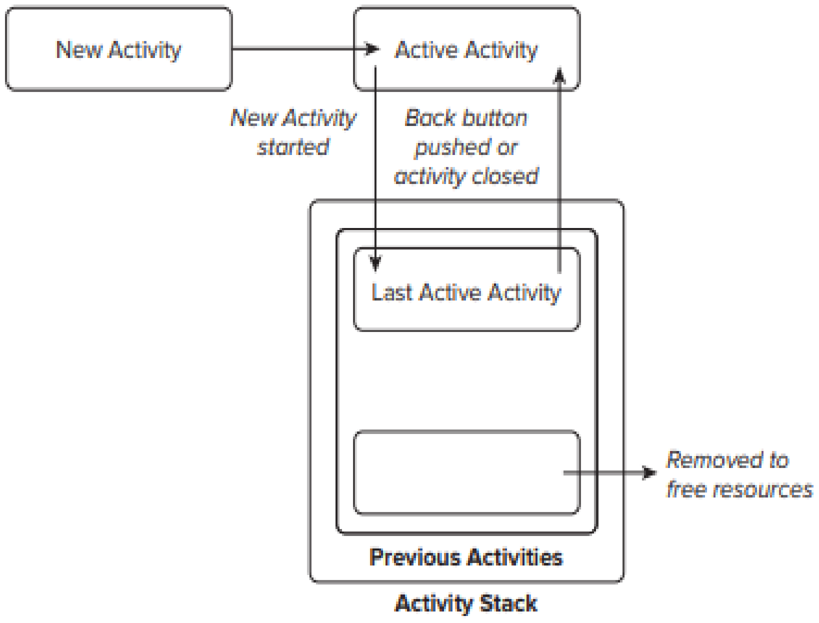
```
<activity android:label=”@string/app_name”
android:name=”.MyActivity”>

<intent-filter><action android:name=”android.intent.action.MAIN” />

<category android:name=”android.intent.category.LAUNCHER” />

</intent-filter>
</activity>
```

- **The Activity Lifecycle**
- Activity Stacks

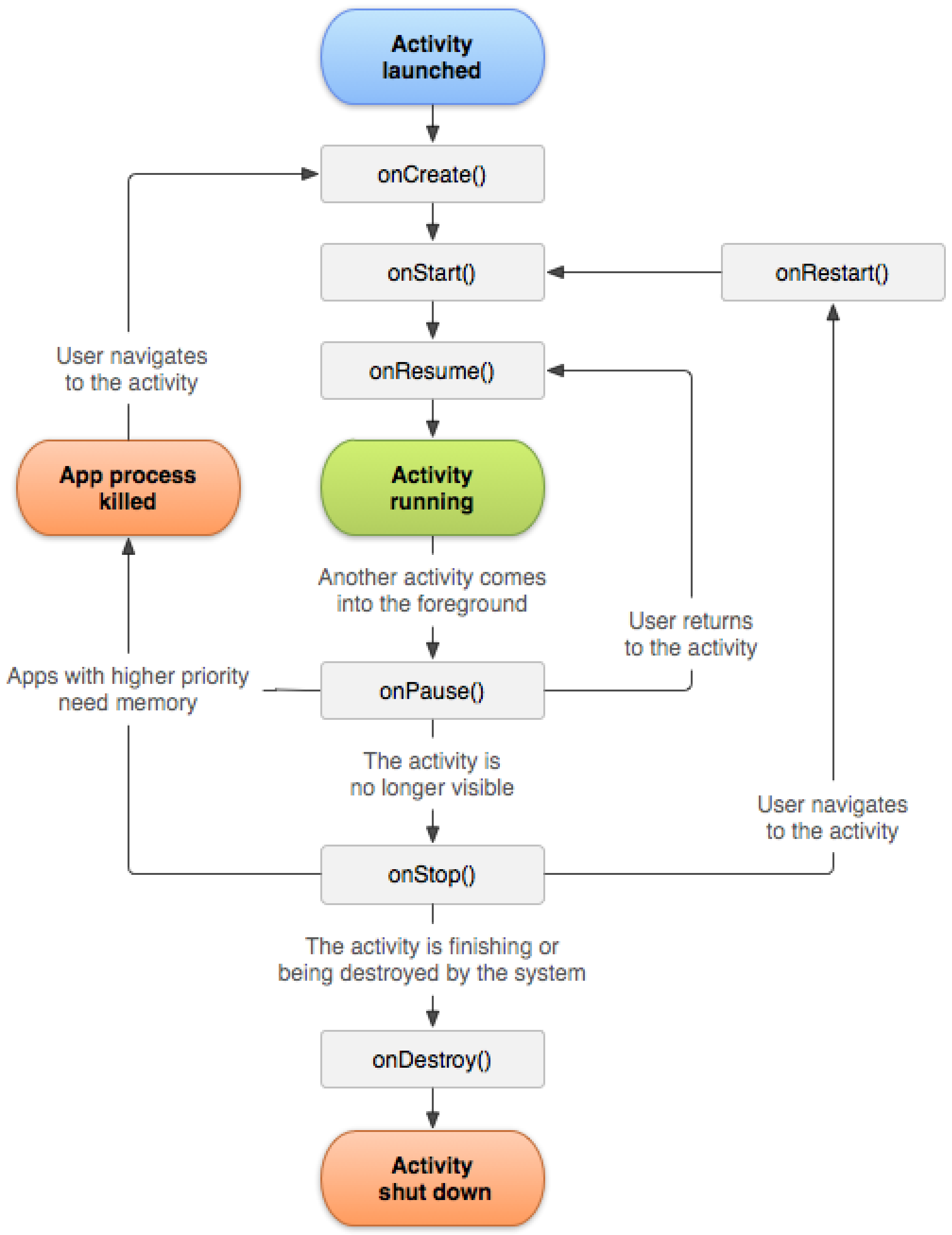


- o a last-in–fi rst-out colection of all the currently running Activities
- o When a new Activity starts, it becomes active and is moved to the top of the stack. If the user navigates back using the Back button, or the foreground Activity is otherwise closed, the next Activity down on the stack moves up and becomes active.

Activity States

- **Active**
  - **top of the stack** it is the **visible, focused**, foreground Activity that is **receiving user input.**
  - When another Activity becomes active, this one will be paused.
- **Paused**
  - **visible but will not have focus;**
  - occurs when a transparent or non-full-screen Activity is active in front
  - it doesn't receive user input events.
  - In extreme cases Android will kill a paused Activity to recover resources for the active Activity
- **Stopped**
  - When an Activity **isn't visible,** it "stops."
  - will remain in memory, retaining all state information
- **Inactive**
  - After an Activity has been killed, and before it's been launched, it's inactive.
  - Inactive Activities have been removed from the Activity stack

- **Managing Android Life Cycle**
    - **7 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

In its lifetime, an activity goes through a number of states. These are series of callbacks to handle transitions between states.

1. **onCreate() -** fires **when the system creates your activity**. Your implementation should **initialize the essential components** of your activity.
   - **Example:-** An app should create views and bind data to lists here. This is method where setContentView() is to be called.
   - Next callback is always onStart().

2. **onStart()**
   - As onCreate() exits, the activity enters the Started state, and the activity becomes **visible to the user**.
   - Consists - **preparations for coming to the foreground** and becoming interactive.

3. **onResume()**
   - The system invokes this callback **just before the activity starts interacting with the user**.
   - At this point, the activity is at the **top of the activity stack**, and captures all user input.
   - Most of an **app's core functionality is implemented** in the onResume() method.

4. **onPause()**
   - The system calls onPause() **when the activity loses focus** and enters a Paused state.
   - This state occurs when, for example, the **user taps the Back or Recents button.**
   - When the system calls onPause() for your activity, it technically means your activity is still **partially visible**, but most often is an indication that the user is leaving the activity, and the activity will soon enter the Stopped or Resumed state.
   - An activity in the Paused state **may continue to update the UI** if the user is expecting the UI to update. Examples of such an activity include one showing **a navigation map screen or a media player playing.** Even if such activities lose focus, the user expects their UI to continue updating.
   - **You should not use** onPause() to
     - save application or user data,
     - make network calls, or

        - execute database transactions.
    - Once onPause() finishes executing, the next callback is either onStop() or onResume(), depending on what happens after the activity enters the Paused state.

5. **onStop()**
    - The system calls onStop() when the activity is **no longer visible to the user.**
    - This may happen because
        - the activity is being destroyed,
        - a new activity is starting, or
        - an existing activity is entering a Resumed state and is covering the stopped activity.
    - The next callback that the system calls is either
        - onRestart(), if the activity is coming back to interact with the user, or
        - onDestroy() if this activity is completely terminating.

6. **onRestart()**
    - The system invokes this callback **when an activity in the Stopped state is about to restart.**
    - **restores the state of the activity** from the time that it was stopped.
    - This callback is always followed by onStart().

7. **onDestroy()**
    - The system invokes this callback **before an activity is destroyed**.
    - This callback is the final one that the activity receives.
    - onDestroy() is usually implemented to **ensure** that **all of an activity's resources are released** when the activity, or the **process** containing it, is **destroyed.**