CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | |

First Internal Test

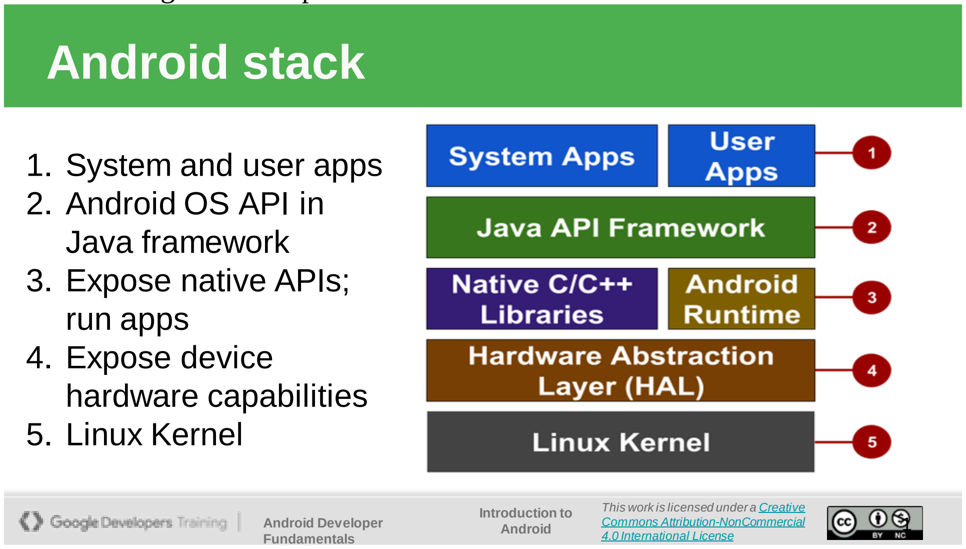
Sub:	MOBILE APPLICATION DEVELOPMENT							Code:	**15CS661**
Date:	14 / 03 / 2018	Duration:	90 mins	Max Marks:	50	Sem:	VI	Branch:	**ISE,CSE**
Answer Any **FIVE FULL** Questions									

		Marks	OBE	
			CO	RBT
1 (a)	What is Android? Explain Android architecture with block diagram? .	[10]	CO1	L2
2 (a)	What do you mean by Layout? Explain different Types of it?	[10]	CO1	L4
3 (a)	Explain about different Resources available in android? How can you use this in app development?	[10]	CO1	L4
4 (a)	What are Intents? Differentiate between Explicit Intent and Implicit Intent with a suitable note?	[10]	CO2	L3
5 (a)	Explain Activity lifecycle states and its callback methods?	[10]	CO1	L2
6 (a)	What are different basic components of an Android application?	[05]	CO1	L4
(b)	Explain the role of AndroiManifest.xml file in an Android application?	[05]	CO1	L3
7 (a)	How can you create options menu and floating contextual menu in android? Explain it with the help of the code snippet.	[10]	CO2	L4

1. **What is Android? Explain android Stack?**
   Explanation+block diagram-5+5

   Android is an open source Linux based operating system which is a product of Google Inc.(after acquiring from Android Inc.). Initially it was developed for touch screen mobile phones, but now it is also available for tablets, smart watch and Android Auto too. It is written in C, C++ and Java.

For developing the Android applications Android SDK is available with all supportive tools. You can also state that, android is a system that includes an open source operating system, an open source development platform and devices that run the operating system and applications created for it. It has become the multi-billion dollar industry since its inception, so there is lots of space for earning for developers and users.



Android stack

1. System and user apps
2. Android OS API in Java framework
3. Expose native APIs; run apps
4. Expose device hardware capabilities
5. Linux Kernel

1. **Apps:** Your apps live at this level, along with core system apps for email, SMS messaging, calendars, Internet browsing, or contacts.

2. **Java API Framework:** All features of Android are available to developers through application programming interfaces (APIs) written in the Java language. You don't need to know the details of all of the APIs to learn how to develop Android apps, but you can learn more about the following APIs, which are useful for creating apps:

View System used to build an app's UI, including lists, buttons, and menus.

Resource Manager used to access to non-code resources such as localized strings, graphics, and layout files.

Notification Manager used to display custom alerts in the status bar.

Activity Manager that manages the lifecycle of apps.

Content Providers that enable apps to access data from other apps.

All framework APIs that Android system apps use.

3. **Libraries and Android Runtime:** Each app runs in its own process and with its own instance of the Android Runtime, which enables multiple virtual machines on low-memory devices. Android also includes a set of core runtime libraries that provide most of the functionality of the Java programming language, including some Java 8 language features that the Java API framework uses. Many core Android

system components and services are built from native code that requires native libraries written in C and C++. These native libraries are available to apps through the Java API framework.

4. **Hardware Abstraction Layer (HAL):** This layer provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework. The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or bluetooth module.

5. **Linux Kernel:** The foundation of the Android platform is the Linux kernel. The above layers rely on the Linux kernel for underlying functionalities such as threading and low-level memory management. Using a Linux kernel enables Android to take advantage of key security features and allows device manufacturers to develop hardware drivers for a well-known kernel.

2.Explain about Layouts ?Explain different Types of it?

Layout Explanation+atleast 5 categories explanation-5+5

Layouts

- are specific types of view groups
- are subclasses of ViewGroup
- contain child views
- can be in a row, column, grid, table, absolute

LinearLayout: A group of child views positioned and aligned horizontally or vertically.

RelativeLayout: A group of child views in which each view is positioned and aligned relative to other views within the view group. In other words, the positions of the child views can be described in relation to each other or to the parent view group.

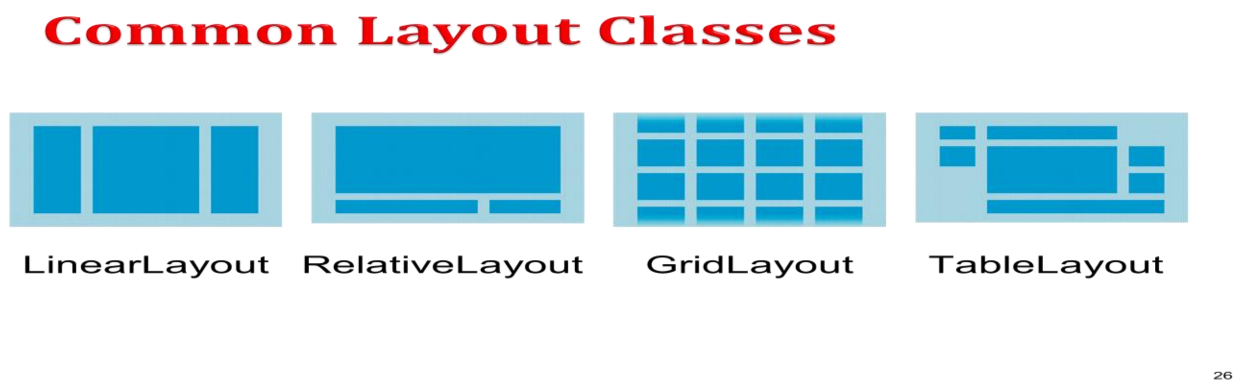
ConstraintLayout: A group of child views using anchor points, edges, and guidelines to control how views are positioned relative to other elements in the layout. ConstraintLayout was designed to make it easy to drag and drop views in the layout editor.

TableLayout: A group of child views arranged into rows and columns.

AbsoluteLayout: A group that lets you specify exact locations (x/y coordinates) of its child views. Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.

FrameLayout: A group of child views in a stack. FrameLayout is designed to block out an area on the screen to display one view. Child views are drawn in a stack, with the most recently added child on top. The size of the FrameLayout is the size of its largest child view.

GridLayout: A group that places its child screens in a rectangular grid that can be scrolled.



3. **Explain about different Resources available in android ?how can you handle this?**
Explanation+acess-5+5

Resource files

Resource files are a way of separating static values from code so that you don't have to change the code itself to change the values. You can store all the strings, layouts, dimensions, colors, styles, and menu text separately in resource files.Resource files are stored in folders located in the **res** folder, including:

drawable: For images and icons

layout: For layout resource files

menu: For menu items

mipmap: For pre-calculated, optimized collections of app icons used by the Launcher

values: For colors, dimensions, strings, and styles (theme attributes).

The syntax to reference a resource in an XML layout is as follows:

@package_name:resource_type/resource_name

The *package_name* is the name of the package in which the resource is located. This is not required when referencing resources from the same package — that is, stored in the **res** folder of your project.

resource_type is the R subclass for the resource type. See Resource Types for more information about each resource type and how to reference them.

resource_name is either the resource filename without the extension, or the android:name attribute value in the XML element.

For example, the following XML layout statement sets the android:text attribute to a string resource:

android:text="@string/button_label_toast"

The *resource_type* is string .

The resource_name is button_label_toast.

Values resource files

Keeping values such as strings and colors in separate resource files makes it easier to manage them, especially if you use them more than once in your layouts.

For example, it is essential to keep strings in a separate resource file for translating and localizing your app, so that you can create a string resource file for each language without changing your code. Resource files for images, colors, dimensions, and other attributes are handy for developing an app for different device screen sizes and orientations.

Strings

String resources are located in the **strings.xml** file in the **values** folder inside the **res** folder when using the Project:Android view. You can edit this file directly by opening it:

```
<resources>
<string name="app_name">Hello Toast</string>
<string name="button_label_count">Count</string>
<string name="button_label_toast">Toast</string>
<string name="count_initial_value">0</string>
</resources>
```

The name (for example, button_label_count) is the resource name you use in your XML code, as in the following attribute:

android:text="@string/button_label_count"

The string value of this name is the word (Count) enclosed within the <string></string> tags (you don't use quotation marks unless the quotation marks should be part of the string value.)

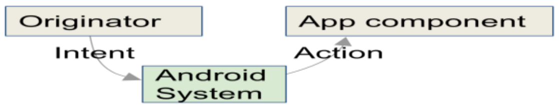
4.What are Intents? Differentiate between Explicit Intent and Implicit Intent with a suitable note.?

Explanation+block diagram-5+5

WHAT IS AN INTENT?

An intent is a description of an operation to be performed.

An Intent is an object used to request an action from another app component via the Android system.



WHAT CAN INTENTS DO?

- Start activities
  - A button click starts a new activity for text entry
  - Clicking Share opens an app that allows you to post a photo
- Start services
  - Initiate downloading a file in the background
- Deliver broadcasts
  - The system informs everybody that the phone is now charging

EXPLICIT AND IMPLICIT INTENTS

Explicit Intent

- Starts a specific activity
  - Request tea with milk delivered by Nikita
  - Main activity starts the ViewShoppingCart activity

Implicit Intent

- Asks system to find an activity that can handle this request
  - Find an open store that sells green tea
  - Clicking Share opens a chooser with a list of apps

Start an Activity with an explicit intent

To start a specific activity, use an explicit intent

1. Create an intent
   - `Intent intent = new Intent(this, ActivityName.class);`
2. Use the intent to start the activity
   - `startActivity(intent);`

Start an Activity with implicit intent

To ask Android to find an Activity to handle your request, use an implicit intent

1. Create an intent
   - `Intent intent = new Intent(action, uri);`
2. Use the intent to start the activity
   - `startActivity(intent);`

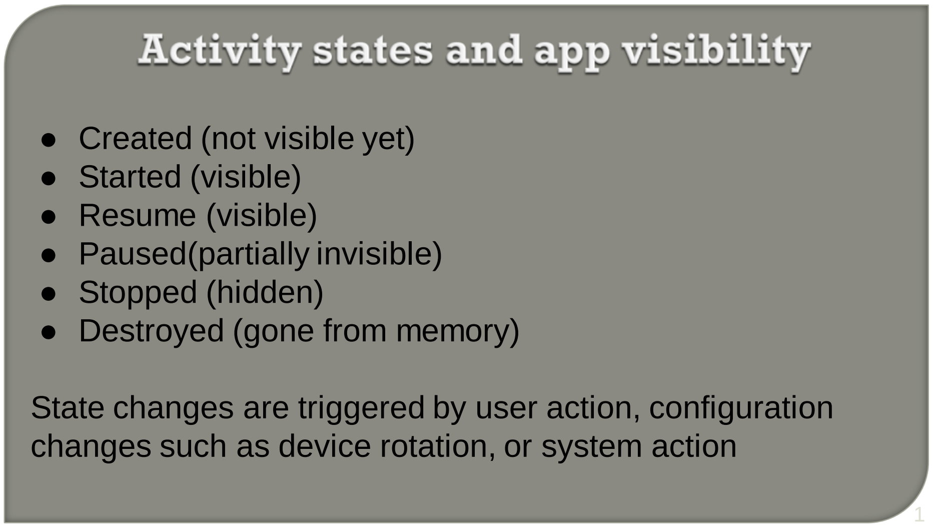
Eg: **Show a web page**

```
Uri uri = Uri.parse("http://www.google.com");
Intent it = new Intent(Intent.ACTION_VIEW,uri);
startActivity(it);
```

5. Activity states and lifecycle callback methods

Explanation+ diagram-5+5

When an activity transitions into and out of the different lifecycle states as it runs, the Android system calls several lifecycle callback methods at each stage. All of the callback methods are hooks that you can override in each of your Activity classes to define how that activity behaves when the user leaves and re-enters the activity. Keep in mind that the lifecycle states (and callbacks) are per activity, not per app, and you may implement different behavior at different points in the lifecycle for different activities in your app.

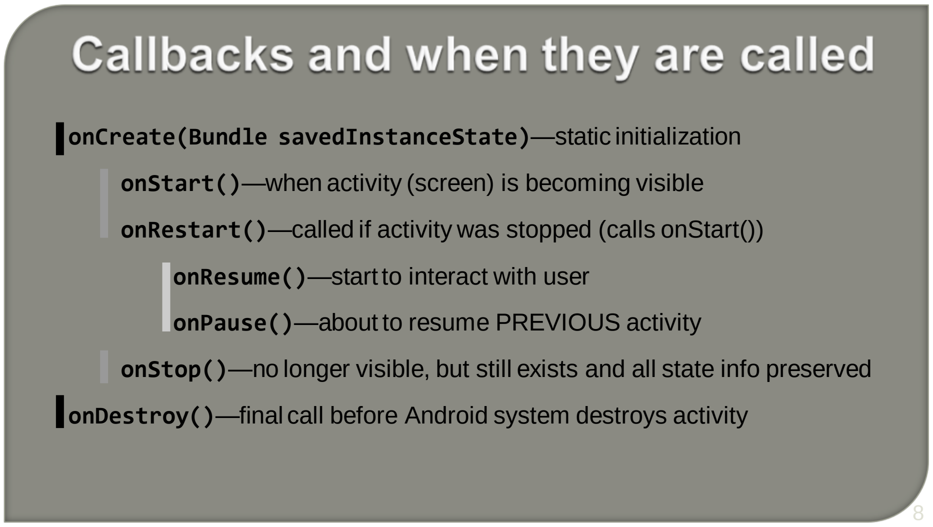


Activity states and app visibility

- Created (not visible yet)
- Started (visible)
- Resume (visible)
- Paused(partially invisible)
- Stopped (hidden)
- Destroyed (gone from memory)

State changes are triggered by user action, configuration changes such as device rotation, or system action

1



Callbacks and when they are called

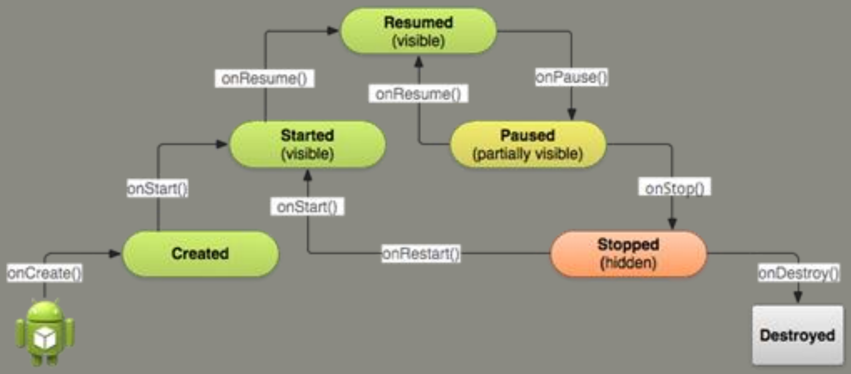
`onCreate(Bundle savedInstanceState)`—static initialization

- **`onStart()`**—when activity (screen) is becoming visible
- **`onRestart()`**—called if activity was stopped (calls onStart())
  - **`onResume()`**—start to interact with user
  - **`onPause()`**—about to resume PREVIOUS activity
- **`onStop()`**—no longer visible, but still exists and all state info preserved

`onDestroy()`—final call before Android system destroys activity

8

Activity states and callbacks graph



onCreate() –> Created

- Called when the activity is first created, for example when user taps launcher icon
- Does all static setup: create views, bind data to lists …
- Only called once during an activity's lifetime
- Created state is always followed by onStart()

onCreate(Bundle savedInstanceState)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // The activity is being created.
}
```

onRestart() –> Started

- Called after activity has been stopped, immediately before it is started again
- Always followed by `onStart()`

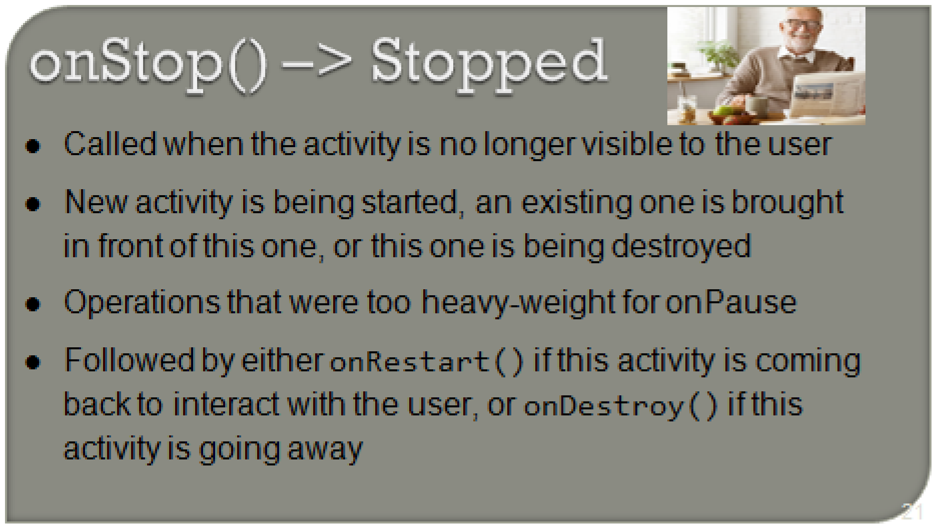
onRestart() –> Started

- Called after activity has been stopped, immediately before it is started again
- Always followed by `onStart()`

15

onPause() –> Paused

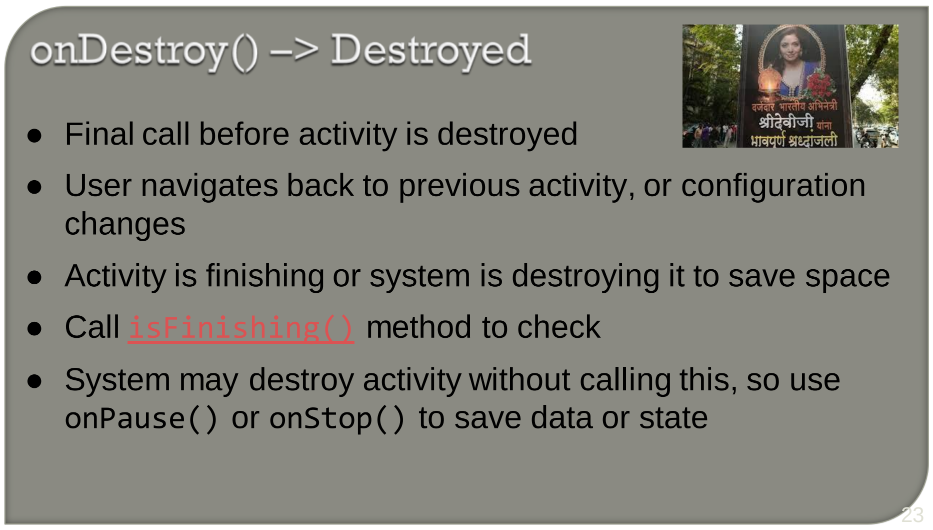
- Called when system is about to resume a previous activity
- The activity is partly visible but user is leaving the activity
- Typically used to commit unsaved changes to persistent data, stop animations and anything that consumes resources
- Implementations must be fast because the next activity is not resumed until this method returns
- Followed by either `onResume()` if the activity returns back to the front, or `onStop()` if it becomes invisible to the user



onStop() –> Stopped

- Called when the activity is no longer visible to the user
- New activity is being started, an existing one is brought in front of this one, or this one is being destroyed
- Operations that were too heavy-weight for onPause
- Followed by either `onRestart()` if this activity is coming back to interact with the user, or `onDestroy()` if this activity is going away

21


दर्जेदार भारतीय अभिनेत्री
श्रीदेवीजी यांना
भावपूर्ण श्रद्धांजली

onDestroy() –> Destroyed

- Final call before activity is destroyed
- User navigates back to previous activity, or configuration changes
- Activity is finishing or system is destroying it to save space
- Call `isFinishing()` method to check
- System may destroy activity without calling this, so use `onPause()` or `onStop()` to save data or state

23

6.a. What are different basic components of an Android application?

Here are following four main components that can be used within an Android application –

Sr.No	Components & Description
1	**Activities** They dictate the UI and handle the user interaction to the smart phone

	screen.
2	**Services** They handle background processing associated with an application.
3	**Broadcast Receivers** They handle communication between Android OS and applications.
4	**Content Providers** They handle data and database management issues.

Activities

An activity represents a single screen with a user interface,in-short Activity performs actions on the screen. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity 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.

An activity is implemented as a subclass of **Activity** class as follows −

```
public class MainActivity extends Activity {

}
```

Services

A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

A service is implemented as a subclass of **Service** class as follows −

```
public class MyService extends Service {

}
```

Broadcast Receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

A broadcast receiver is implemented as a subclass of **BroadcastReceiver**class and each message is broadcaster as an **Intent** object.

```
public class MyReceiver  extends  BroadcastReceiver {
   public void onReceive(context,intent){}
}
```

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. The data may be stored in the file system, the database or somewhere else entirely.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends  ContentProvider {
   public void onCreate(){}
}
```

6.b Explain the role of AndroiManifest.xml file in an Android application?

Structure+content details-2.5+2.5

The **AndroidManifest.xml file** *contains information of your package*, including components of the application such as activities, services, broadcast receivers, content providers etc.

It performs some other tasks also:

- It is **responsible to protect the application** to access any protected parts by providing the permissions.
- It also **declares the android api** that the application is going to use.

- It **lists the instrumentation classes**. The instrumentation classes provides profiling and other informations. These informations are removed just before the application is published etc.

This is the required xml file for all the android application and located inside the root directory.

Elements of the AndroidManifest.xml file

The elements used in the above xml file are described below.

<manifest>

manifest is the root element of the AndroidManifest.xml file. It has **package** attribute that describes the package name of the activity class.

<application>

application is the subelement of the manifest. It includes the namespace declaration. This element contains several subelements that declares the application component such as activity etc.

The commonly used attributes are of this element are **icon**, **label**, **theme** etc.

android:icon represents the icon for all the android application components.

android:label works as the default label for all the application components.

android:theme represents a common theme for all the android activities.

<activity>

activity is the subelement of application and represents an activity that must be defined in the AndroidManifest.xml file. It has many attributes such as label, name, theme, launchMode etc.

android:label represents a label i.e. displayed on the screen.

android:name represents a name for the activity class. It is required attribute.

<intent-filter>

intent-filter is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.

<action>

It adds an action for the intent-filter. The intent-filter must have at least one action element.

<category>

It adds a category name to an intent-filter.

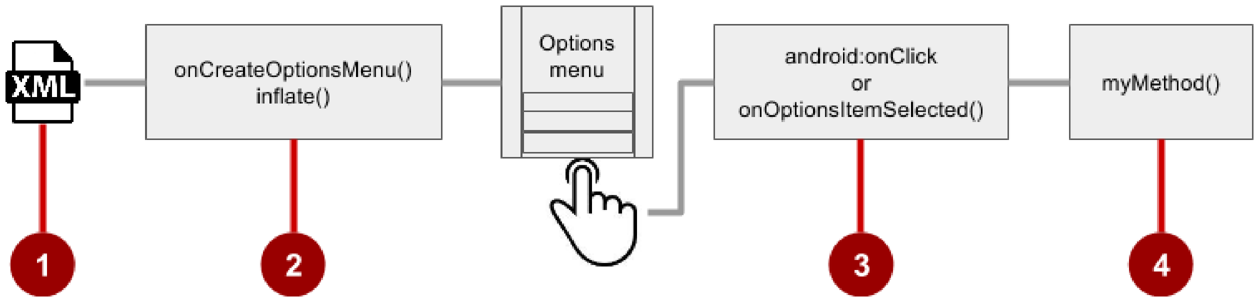
7.Explain options menu and contextual menu in android?

Explanation+ steps to make it(5+5)

Options menu and app bar

The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."

1. XML menu resource (menu_main.xml)
2. onCreateOptionsMenu() to inflate the menu
3. onClick attribute or onOptionsItemSelected()
4. Method to handle item click

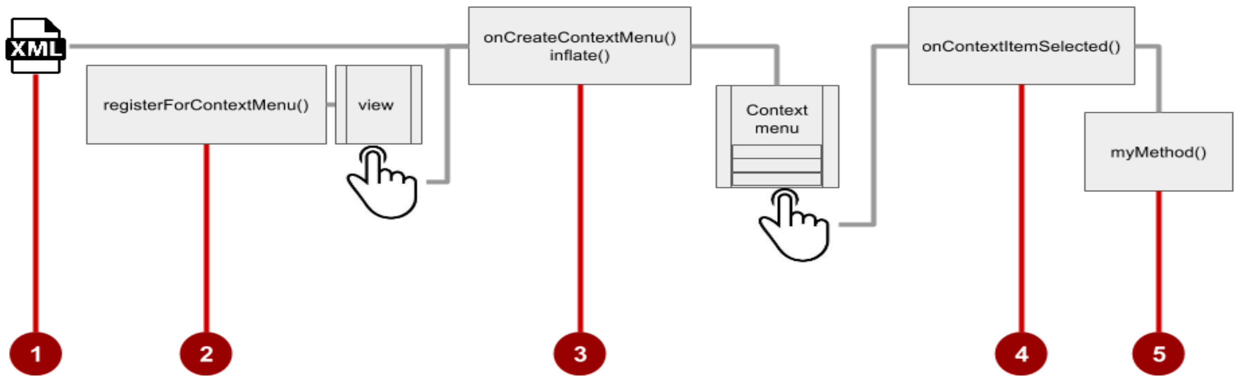


Context menu and contextual action mode

A context menu is a floating menu that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.

The contextual action mode displays action items that affect the selected content in a bar at the top of the screen and allows the user to select multiple items.

1. Create XML menu resource file and assign appearance and position attributes
2. Register view to use a context menu using registerForContextMenu()
3. Implement onCreateContextMenu() in the activity or fragment to inflate the menu
4. Implement onContextItemSelected() to handle menu item clicks
5. Create a method to perform an action for each context menu item


XML
registerForContextMenu()
view
onCreateContextMenu()
inflate()
Context
menu
onContextItemSelected()
myMethod()
1
2
3
4
5