RV College of Engineering®
Mysore Road, RV Vidyaniketan Post,
Bengaluru - 560059, Karnataka, India

Department of Master of Computer Applications (MCA)

Mobile Application Development (MCA221IA)

Hand Notes

Unit - 1	**Topic: Activity, Layout File**
USN: 1RV24MC101 **NAME:** Shreya P Negur	

List of Questions

1. **Question 11 (02 Marks)**
1. **Question 05 (04 Marks)**
1. **Question 05 (06 or 08 Marks)**
1. **Question 03 (10 Marks)**

02 Marks Questions:

Q1: What are the main lifecycle callback methods invoked when an Android activity is first launched?
Answer: When an activity is first launched, the system calls the following lifecycle methods in order: onCreate(), onStart(), and onResume().

Q2: What is the purpose of the onCreate() method in the Android activity lifecycle?
Answer: The onCreate() method is called when an activity is first created. It's used to initialize the activity, set the user interface layout using setContentView(), and perform startup logic like initializing variables, setting up listeners, or restoring saved state.

Q3: When is onRestart() called in the activity lifecycle?
Answer: onRestart() is called after the activity has been stopped, just before it is started again. It's used to perform any logic required to resume the activity after it was previously stopped.

Q4: List any two callback methods that indicate the activity is no longer in the foreground.
Answer: onPause() and onStop() indicate the activity is no longer in the foreground. onPause() is triggered when another activity is partially visible, while onStop() is triggered when it is fully obscured.

Q5: Which method is invoked when an activity is about to be destroyed?
Answer: onDestroy() is invoked when the activity is about to be destroyed, either because the user finishes the activity or the system is temporarily destroying it to save space.

Q6: Explain the significance of onStop() and what kind of operations should be handled in this method.
Answer: onStop() is triggered when the activity is no longer visible. It's the place to stop ongoing

tasks, unregister listeners, and release resources like receivers or sensors. Unlike onPause(), it indicates full obscurity, so heavier cleanup can occur here.

Q7: What is the sequence of lifecycle methods when a user presses the back button to exit the activity?
Answer: The sequence is: onPause(), followed by onStop(), and then onDestroy(). This represents the activity going from active to background to being removed from memory.

Q8: What is the purpose of a layout file in Android?
Answer: A layout file in Android defines the structure of the user interface (UI) of an app using XML. It specifies how UI elements such as buttons, text views, and images are arranged on the screen.

Q9: Name the types of layout managers in Android.
Answer: Linear Layout
Relative Layout
Constraint Layout
Table Layout
Frame Layout
Grid Layout

Q10: How is a layout file associated with an activity in Android?
Answer: A layout file is linked to an activity using the setContentView(R.layout.layout_name) method in the onCreate() method of the activity.

Q11: Write the syntax to reference a layout file in setContentView().
Answer: setContentView(R.layout.activity_main);

04 Marks Questions:

Q1: Define the onResume() and onPause() in the lifecycle of an android activity.
Answer: In the Android activity lifecycle, onResume() and onPause() play crucial roles in managing an activity's visibility and interaction with the user.

- onResume() is called when the activity enters the foreground and becomes interactive. This means the user can now see and interact with the activity. It is typically used to restart any processes that were paused in onPause(), such as resuming animations, refreshing UI elements, restarting a camera preview, or acquiring exclusive resources like sensors or GPS.

- onPause() is called when the system is about to place the activity into the background, but it is still partially visible (e.g., a new activity appears on top as a dialog or with transparency). It is used to pause ongoing operations that shouldn't continue when the activity isn't in full focus. Common tasks include pausing animations, stopping media playback, or saving unsaved data such as form input or temporary states.

Q2: What happens when the Android system kills an activity in the background?
Answer: When the Android system is low on memory or resources, it may terminate background activities to free up space for active processes. This often happens when the activity is no longer visible to the user and considered to be in the background. When such a termination occurs, the activity instance is completely removed from memory, and any unsaved transient state such as user input, temporary UI changes, or variables stored in memory will be lost.To handle such unexpected

termination, Android provides the onSaveInstanceState() callback. This method allows developers to save essential UI-related data, such as scroll positions, form inputs, or other temporary states, into a Bundle. This data is then made available during the next activity creation via onCreate() or onRestoreInstanceState(), enabling the app to restore the previous state and offer a seamless user experience.

Q3: Differentiate between Linear Layout and Relative Layout.
Answer:

Linear Layout	Relative Layout
We can adjust the views and widgets linearly i,e. horizontally or vertically.	It allows positioning of elements relative to each other or to the parent layout.
Views are stacked one after another in a linear form.	Views can aligned using attributes like: android:layout_alignParentTop, ParentLeft etc.
Simple and efficient for straightforward UI designs.	More flexible for complex UI designs.
Example: A row buttons or a vertical list of text fields.	Example: A button placed to the right of a text field, or an image centered inside a parent.
Syntax: <LinearLayout android: layout_width=”match_parent” android: layout_height=”match_parent” android:orientation=”vertical or horizontal”> </LinearLayout>	Syntax: <RelativeLayout android: layout_width=”match_parent” android: layout_height=”match_parent”> </RelativeLayout>

Q4: Explain how match_parent and wrap_content work in layout files.
Answer: match_parent – It makes views expand to fill the entire space of its parent.
Wrap_content – It adjust views size to fit its content.
Example: <Button
```
        android:layout_width=”match-parent”
        android:layout_height=”wrap_content”
        android:text=”Click me”/>
```

Q5: What are the advantages of using constraints layout over traditional layout?

- Reduces the need for nested layouts.
- Improves the performance by flattening the view hierarchy.
- Offers flexible positioning using constraints.
- Supports complex UI’s with fewer lines of XML.

06 or 08 Marks Questions:

Q1: Explain the role of onSaveInstanceState() and onRestoreInstanceState() in the Android lifecycle.
Answer: 1. onSaveInstanceState(Bundle outState)
This method is called before your activity gets paused or destroyed by the system. It is used to save the current state of your activity, especially any temporary UI-related data that the user was working

- For example:
  - If the user typed something in an input field (EditText), you can save that text.
  - If the user scrolled down in a list or page, you can save that scroll position.

    - If the user selected an option in a drop-down or clicked on something, that state can be saved.

You use a Bundle object (which is like a key-value storage) to store this information. The system keeps this bundle safe, and it gives it back to you when the activity is restarted.
So basically, onSaveInstanceState() lets your app "remember" what the user was doing before the activity was paused or destroyed.

2. onRestoreInstanceState(Bundle savedInstanceState)
This method is called after the activity has been restarted, and only if there is saved state available from onSaveInstanceState().
In this method, you can get back the data that you stored earlier in the bundle and use it to restore the activity to the same state it was in before.

- For example:
    - You can set the saved text back into the input field.
    - You can scroll the page back to where the user left off.
    - You can highlight or show the same selection or screen as before.

Q2: Describe the Android Activity Lifecycle in detail. Discuss the significance of each lifecycle callback method. Also explain how state can be saved and restored during lifecycle changes.
Answer: The Android Activity Lifecycle manages how activities are created, run, paused, stopped, and destroyed. It includes the following callback methods:

1. onCreate():
    - Called when the activity is first created.
    - UI components are initialized here.
    - If the activity is being re-initialized after being previously shut down, the savedInstanceState contains the data it most recently supplied.
2. onStart():
    - Makes the activity visible to the user.
    - Not yet in the foreground, but the user can see it.
3. onResume():
    - The activity is now in the foreground and the user can interact with it.
    - This is where the activity starts responding to user input.
4. onPause():
    - The system calls this when the activity is partially obscured.
    - Good place to pause animations or commit unsaved changes.
5. onStop():
    - The activity is no longer visible.
    - Release resources or save state here.
6. onRestart():
    - Called when the activity is coming back to the foreground from a stopped state.
7. onDestroy():
    - Called before the activity is destroyed.
    - Final cleanup is performed here.

Q3: Explain the Android Activity Lifecycle.
Answer: Here are the key lifecycle methods:

1. onCreate(): Called when the activity is first created. Initialization of UI and variables is done

here.
2. onStart(): Called when the activity becomes visible to the user.
3. onResume(): Called when the activity starts interacting with the user.
4. onPause(): Called when the system is about to resume another activity. The activity is still visible but partially obscured.
5. onStop(): Called when the activity is no longer visible.
6. onRestart(): Called when the activity is coming back to the foreground from the stopped state.
7. onDestroy(): Called before the activity is destroyed either due to user finishing it or system destroying it to reclaim memory.

Q4: What happens when the Android system kills an activity in the background? Explain in detail.
Answer: When the Android system is low on memory or resources, it may decide to kill activites that are in the background to free up space for foreground tasks or other hight priority processes.
This process is entirely managed by the android operating system when an activity is not visible to the user (i,e it is in the background) the system kills the background activity, all data that is not saved in persistent storage such as database or a file is lost.This includes any temporary data in memory like the current scroll position inputs in txt fields or the state of the user interface. To manage this, android provides the onSaveInstanceState() method which is called before an activity is killed. This method can be used to store essential UI related data, such as form the inputs, navigations positions or UI selections. Later, when the activity is created (such as the user returns to it) this data can be retrieved either in the onCreate() or onRestoreInstanceState() methods.

Q5: Explain the structure and components of an android XML layout file with an example**.**
Answer: An android layout file is an XML files that defines the UI. The root tag is usually is a view group like linear layout or constraint layout. Inside it there are view elements like TextView, Button etc.

1.Root Element:
The layout XML file begins with a root element which acts a container for all other UI elements.
Common roots include:
Linear Layout: Arranges children horizontally or vertically.
Constraint Layout: Allows flexible positioning of children with constraints.

2.View Elements:
Inside the root element, you define various UI elements like:
TextView: Display text.
Button: Create a clickable button.
EditText: Allows user to enter text.

3.Attributes:
Each view elements has attributes that control its appearance and behaviour such as:
Android:layout_width, android:layout_height:- Specifies the width and height of the view.
Android:text :- Sets the text displayed by a TexView.
Android:layout_alignParentStart etc.

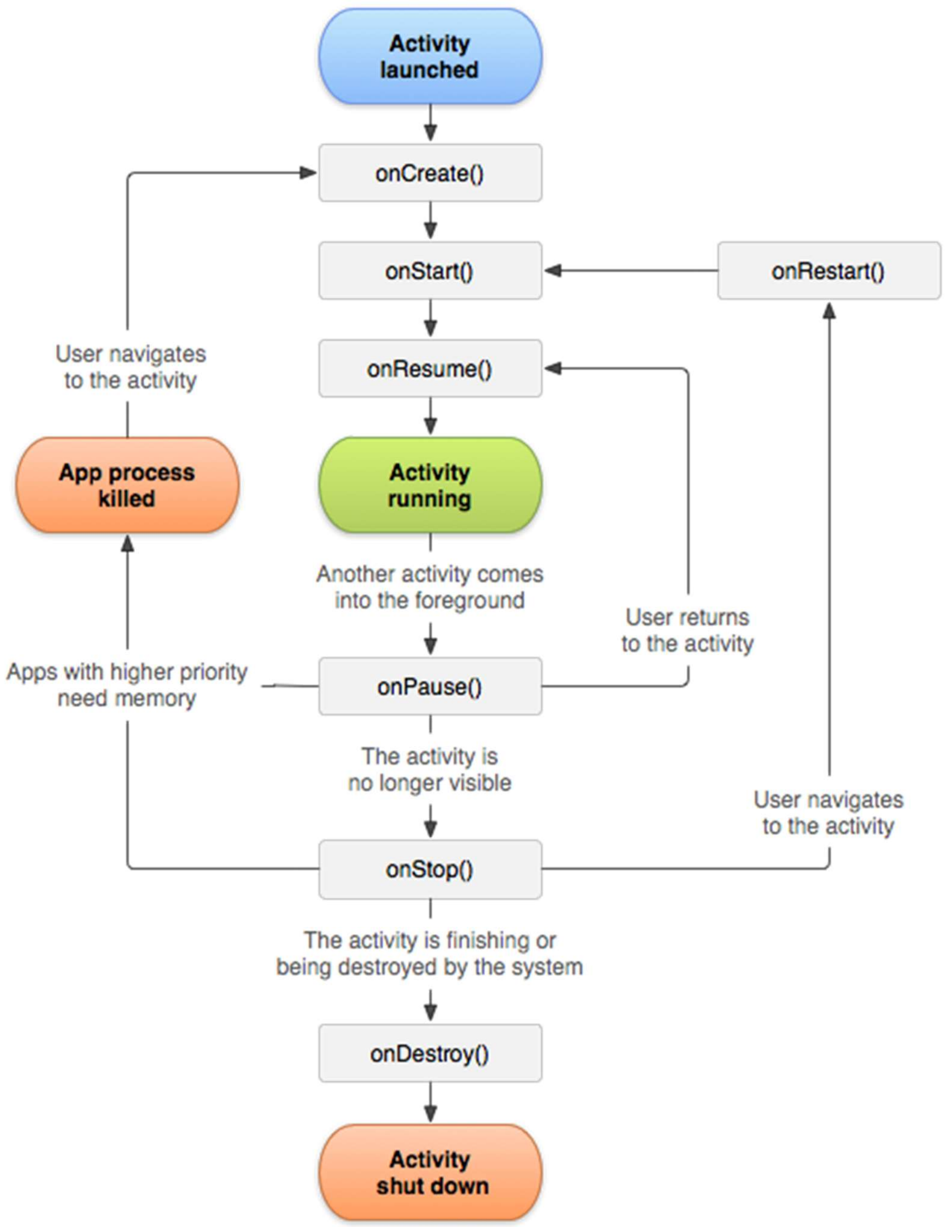
Example:
```
<LinearLayout xmlns:android=”https://schemas.android.com/apk/res/android”
 android:layout_width=”match_parent”
 android:layout_height="match_parent"
 android:orientation="vertical"
 android:padding="16dp">

<TextView
  android:id ="@+id/textview1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Hello world"
  android:textsize="20sp"/>

<Button
   android:id="@+id/button1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text= "Click me"/>
</LinearLayout>
```

10 Marks Questions

Q1: Explain the Android Activity Lifecycle with the help of a diagram.
Answer: The Android Activity Lifecycle defines the various states an activity goes through from creation to destruction. Here are the key lifecycle methods:

1. **onCreate()**: Called when the activity is first created. Initialization of UI and variables is done here.
2. **onStart()**: Called when the activity becomes visible to the user.
3. **onResume()**: Called when the activity starts interacting with the user.
4. **onPause()**: Called when the system is about to resume another activity. The activity is still visible but partially obscured.
5. **onStop()**: Called when the activity is no longer visible.
6. **onRestart()**: Called when the activity is coming back to the foreground from the stopped state.
7. **onDestroy()**: Called before the activity is destroyed either due to user finishing it or system destroying it to reclaim memory.

The activity lifecycle diagram is represented as below shown:


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

Q2: Describe a scenario where all lifecycle methods from onCreate() to onDestroy() would be called in one session.

Answer: Suppose a user opens your app and navigates to a screen(an activity) looks at it for a few seconds, and then presses the Back button to exit the screen. Since pressing back means the user wants to completely close that screen. Android removes the activity from memory triggering the full lifecycle.

1 .**onCreate() :** This is called when the activity is first created. Android sets up the user interface and prepares everything needed.

2. **onStart() :** It is called right after onCreate(). This means the activity is now visible to the user.

3 .**onResume() :** The activity comes to the foreground and is now ready for user interaction. The user can how see and interact with the screen.

4: **onPause() :** It is called when the activity is about to go out of the foreground.

5: **onStop() :** The activity is no longer visible to the user.
6: **onRestart() :** It is called when the activity is coming back to the foreground from the stopped state.
7: **onDestroy()** : It is called when the activity is about to be completely removed from memory. All resources are cleaned up.

Q3: Explain the different types of layout in Android and explain with example.
Answer: In Android development, layouts are used to define how UI elements (views) are arranged on the screen. Each layout type provides a different way of positioning views.
1.Linear Layout :- LinearLayout arranges its children in a single row or column, either vertically or horizontally.
Key Features:

- Set direction using: android:orientation="vertical" or "horizontal".
- Views appear one after another.
- Supports weight distribution with layout_weight.

Example:
```
        <LinearLayout
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

  <TextView android:text="Username" ... />
  <EditText android:id="@+id/username" ... />
  <Button android:text="Submit" ... />
</LinearLayout>
```

2. Constraint Layout :- ConstraintLayout is a flexible and powerful layout. It allows you to position elements relative to each other and to the parent layout using constraints.
Key Features:

- Reduces nesting of views.
- Each view must have at least one horizontal and one vertical constraint.
- Offers drag-and-drop UI designing in Android Studio.
- Good for complex UIs.

Example:
```
        <androidx.constraintlayout.widget.ConstraintLayout ... >

  <Button
    android:id="@+id/button1"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    android:text="Click Me" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

3. Relative Layout :- RelativeLayout arranges views in relation to other views or the parent.
Key Features:

- You can align views to the left/right/above/below of other views.

- Reduces nesting compared to LinearLayout.
- Can become confusing with many views.

Example:

```
        <RelativeLayout ... >

   <TextView
      android:id="@+id/label"
      android:text="Name:" />

   <EditText
      android:layout_below="@id/label"
      android:layout_alignStart="@id/label"
      android:id="@+id/nameInput" />
</RelativeLayout>
```

4. Table Layout :- TableLayout arranges its children into rows and columns, similar to a table in HTML.

Key Features:

- Uses TableRow as a container for horizontal row of views.
- Columns can stretch to fit.
- Good for tabular data or forms.

Example:

```
        <TableLayout ... >

   <TableRow>
      <TextView android:text="Name" />
      <EditText android:id="@+id/name" />
   </TableRow>

   <TableRow>
      <TextView android:text="Email" />
      <EditText android:id="@+id/email" />
   </TableRow>
</TableLayout>
```

5. Frame Layout :- FrameLayout is the simplest layout, designed to hold one child view, but it can actually contain multiple views.

Key Features:

- All views are stacked on top of each other, like layers.
- The first view is at the bottom, and the next is drawn above it.

Example:

```
        <FrameLayout ... >

   <ImageView android:src="@drawable/background" />
   <TextView android:text="Welcome" />
</FrameLayout>
```

6. Grid Layout :- GridLayout arranges views in a grid format using rows and columns. It's more flexible than TableLayout.

Key Features:

- Allows control over row/column spanning.
- Works well for keyboard layouts, game UIs, etc.

Example:

```
        <GridLayout
   android:columnCount="2"
   android:layout_width="match_parent"
   android:layout_height="wrap_content">

   <Button android:text="1" />
   <Button android:text="2" />
   <Button android:text="3" />
   <Button android:text="4" />
</GridLayout>
```