MOBILE APPLICATION DEVELOPMENT

MODEL PAPER - 1

SECTION - A

1. What is Mobile Application?

A mobile application, commonly referred to as a mobile app, is a software application designed to run on mobile devices such as smartphones and tablets. These applications are developed specifically for mobile platforms and are typically downloaded and installed from app stores or marketplaces like the Apple App Store for iOS devices and Google Play Store for Android devices.

Example: An example of a mobile application is Instagram, a popular app for sharing photos and videos with friends and followers. Users can upload images and videos, apply filters, and share them on their profile or with a selected audience

2. What is Android SDK?

An Android SDK (Software Development Kit) is a set of tools, libraries, and documentation provided by Google to help developers create applications for the Android platform. It includes everything developers need to start building Android apps, such as APIs for interacting with the device hardware, emulator for testing apps, and various other tools for debugging and profiling.

3. Define View in Android

A View in Android is a fundamental UI (User Interface) component that represents a rectangular area on the screen. It is responsible for drawing content and handling user interactions. Views are the building blocks for creating user interfaces in Android applications. Examples of views include buttons, text fields, images, and other interactive elements.

4 What is a WebView in Android?

"WebView' is a view that displays web pages within an Android application. It is based on the Wes engine and allows integration of web content seamlessly into the app. 'WebView' can be used to th static HTML content or to load web pages from the internet. It's particularly useful for displa content hosted online or for creating hybrid applications that combine web and native views.

5 What does "designing by declaration" refer to in Android development?

Designing by declaration refers to the approach of creating user interfaces by defining their structure and appearance using declarative markup languages such as XML, rather than writing procedural code. In this method, developers specify the layout, properties, and behavior of UI components through descriptive markup to focus on what should be displayed rather than how it should be implemented.

In the context of Android app development, designing by declaration involves using XML files to define the visual elements and attributes of the user interface. By separating the UI design from the application logic, developers can achieve a more modular and maintainable codebase. This approach allows for a clear separation of concerns, making it easier to manage and update the UI independently of the underlying functionality

6. What is a Content Provider in Android?

A content provider in Android is a component that manages access to a structured set of data. It serves as an intermediary between the data source and different applications that need to access or modify that data. The content provider provides a standardized and secure way for apps to share data with each other It encapsulates the data and exposes it through a consistent interface.

Section-B

7. Explain the Brief History of Mobile Techinologies

+ Early Beginnings

The journey of mobile technology began in the early 20th century with the invention of radio communication. The first real step towards mobile communication came with the development of mobile radio telephones in the 1940s. It was primarily used in vehicles with limited coverage and functionality

First Generation (16):

The introduction of the first-generation (1G) mobile networks in the 1980s marked the true beginning of mobile telephony 1G networks were analog and allowed for voice communication through mobile phones

The launch of the Motorola DynaTAC 8000X in 1983, the first commercially available mobile phone, was a landmark event despite its large size and limited battery life

Second Generation (2G)

The 1990s saw the advent of the second-generation (26) mobile networks, which were

digital and offered significant improvements in voice quality and capacity 2G networks introduced the Global System for Mobile Communications (GSM) and it

became the standard for mobile communication worldwide This era also saw the introduction of Short Message Service (SMS) by enabling text messaging between mobile devices

Third Generation (3G)

The early 2000s brought the third generation (3G) mobile networks to enable faster data

transmission and improved internet access on mobile devices. 3G networks supported multimedia messaging, video calls, and mobile internet browsing.

It has shown the way for the development of more advanced mobile applications and services

Technologies like Universal Mobile Telecommunications System (UMTS) and CDMA2000 expanded mobile capabilities beyond voice calls

+ Fourth Generation (4G) and the Rise of Smartphones:

The introduction of fourth-generation (4G) mobile networks in the late 2000s revolutionized mobile technology with high speed internet access to enable seamless streaming of high definition video, faster downloads, and enhanced mobile gaming experiences

Long Term Evolution (LTE) became the dominant 4G standard by offering significantly faster data speeds and lower latency

This period also witnessed the rise of smartphones, integrating powerful processors high resolution displays, and a wide array of sensors, transforming them into versatile tools for communication, entertainment, and productivity

+Fifth Generation (5G) and Beyond:

5G represents the latest advancement in mobile technology promising ultra fast speeds low latency, and massive connectivity to support emerging technologies like Internet of Things (IoT), augmented reality (AR), and autonomous vehicles

SG is set to revolutionize industries by enabling smart cities autonomous vehicles, and advanced healthcare solution

9. Explain DatePicker along with its attributes and features Give an example

DatePicker is a Ul component that allows users to select a specific date. It can display dates in various lunmats and allows navigation between days, months, and years.

Example Scenario: Scheduling an appointment, setting a birthday, or selecting a due date

Features of Date Picker

1. Date Selection: Allows users to select a specific date from a calendar or spinner view

2. Customizable Display: Can be displayed as a calendar or spinner, depending on the mode set (calendarViewShown and datePicker Mode attributes)

3. Min and Max Date: Supports setting minimum and maximum selectable dates to restrict user input to a specific range

4. OnDateChangedListener: Allows developers to handle date changes and perform actions when the selected date changes.

8. What are Intent Filters? Explain the components of Intent Filter

Intent filters are component of an Android application manifest file that specify the types of intent an activity service or broadcast receiver can respond to they define the capabilities of a component and allow other components to interact with it based on the specified criterio Intent filters of actions, categories, data, and MIME types that help Android system identify the appropria

component to handle a specific intent.

Components of an Intent Fiher

Iittent filters are composed of three main elements: action, category, and data. Each of these eleme plays a crucial rule in defining the scope and capability of the intent filter

1. Action: The action element represents the general action to be performed by the inte Actions are typically predefined constants in the 'Intent class. Examples of actions include

android.intent.action.VIEW: This action indicates that an activity can display data the user. It's commonly used for viewing web pages, images, and other types of content

"android.intent.action.SEND': This action is used to send data from one activity is another it's often used for sharing content such as text, images, or other data

android.intent.action DIAL' This action is used to dial a number opening the diale with the number populated

android intent.action.MAIN": This action is used to start the main activity of as application. It signifies the main entry point of the app and is often paired with the LAUNCHER category to display the app icon in the launcher.

android.intent.action EDIT: This action indicates that the activity can edit specified data. It's typically used for editing contacts or calendar events.

"android.intent.action.DELETE': This action is used to delete specified data, such a files or database entries.

2. Category: The category element provides additional information about the action being performed Categories help to refine the type of component that should handle the intem Some common categories include

"android.intent.category DEFAULT: This is the most commonly used category and indicates that the activity can be launched by default if no other category is specified

"android.intent.category BROWSABLE': This category allows the activity to be started by a web browser to display data referenced by a link. It enables activities to handle Ukli

android.intent.category. LAUNCHER': This category is used to indicate that the activity should be listed in the system's application launcher It designates the main entry poin for the application.

"android.intent.category HOME': This category is used for the home screen of the device

android.intent.category APP CONTACTS': This category is used for activities that desi with contacts.

"android.intent.category APP CALENDAR': This category is used for activities related to calendar events

Categories are used in combination with actions to provide a more specific contest for fi intent to ensure that the most appropriate component is chosen to handle it.

3. Data: The data element specifies the type of data that the intent is interested in. This can Include a URI scheme, a MIME type, or both. The data element allows components to declare their capability to handle specific types of data, such as:

URI Scheme: Specifies the protocol for accessing the data (eg. 'http', 'https', 'mailto'). This indicates the type of data source the component can handle.

Example: 'android scheme="http" indicates that the component can handle HTTP URLS

MIME Type: Specifies the format of the data the component can handle (eg. 'image/*". text/plain"). This allows components to specify the exact type of data they can process.

Example: 'android mimeType="image/* indicates that the component can handle any type of image file.

10. How does a Scroll View help in handling different screen sizes in Android? Explain with an example

11. How can an application pre fill email details like recipient, subject and body using an Intent? Provide an example

12. Explain the process of binding an activity to a service in Android

SECTION - C

13 a) What is an Android Studio Write its features

The Integrated Development Environment (IDE) is a software application that provides comprehensive facilities to programmers for software development. Android Studio is the official Integrated Development Environment (IDE) for Android app development. it is designed to streaming the entire app development process. It serves as a comprehensive platform for designing, coding testing, and debugging Android applications. It provides a range of tools and features to facilitate app development.

Key Features:

Code Editor: Android Studio provides a powerful code editur with features like youn highlighting, code completion, refactoring, and code navigation to enhance productivity and code quality

Layout Editor: The Layout Editor enables developers to create visually appealing interfaces with drag and drop functionality, real-time previews, and support for differen screen sizes and orientations.

SDK Manager: The Android SDK Manager is a tool within Android Studio that allows developers to download, install, and manage different versions of the Android SDK, as well as other essential tools and components.

Device Manager: The Device Manager is a tool within Android Studio that allows developers to create and manage virtual devices that simulate physical Android devices These virtual devices are used to test and debug applications on various configurations and Android versions without needing physical devices.

APK Analyzer: Helps analyze APK size, contents, and dependencies to optimize app performance and reduce file size

Built-in Emulator: The built-in Android Emulator allows developers to test their apps on virtual devices with various configurations, screen sizes, and Android versions for comprehensive testing

• Device Testing: Developers can test their apps on physical devices connected via USB for real- world testing scenarios, in addition to the Android Emulator, to ensure app compatibility and performance

Version Control Integration: Android Studio supports version control systems like Git, enabling developers to manage source code, track changes, and collaborate with team members efficiently within the IDE

Extensive Plugin Ecosystem: Android Studio offers a wide range of plugins and extensions from the Plugin Marketplace to extend functionality, integrate with external tools, and customize the development environment according to specific requirements.

Performance Profiling Tools: Android Studio offers performance profiling tools to analyze app performance, identify bottlenecks, and optimize CPU, memory, and network usage for better user experience.

Build Tools: Android Studio integrates the Gradle build system for automating build tasks, managing dependencies, and configuring build variants to streamline the app development proces

Integrated Debugger: An integrated debugger allows developers to identify and fix issues in their code by setting breakpoints, inspecting variables, and stepping through code execution

b) What is the use of Android Manifest File? Explain the components (or) structure of Android Manifest File

The AndroidManifest.xml file is a crucial component of an Android application that provides essential information about the app to the Android system. It contains metadata about the application, its components (such as activities, services, broadcast receivers), permissions required, hardware and software features used, and more. It is located in the project's root directory as AndroidManifest. xml, this file defines how different parts of the app interact with each other and with the Android system.

. Structure for) Components of AndroidManifest.xml

The AndroidManifest.xml file is structured in a hierarchical format with nested elements. Let us understand the components of AndroidManifest.xml

1. XML Declaration

The file starts with an XMI declaration specifying the version of XML being used.

chaml version "1.0" encoding="utf-8"?s

2. <manifest> Element:

The <manifest> element is the root element of the AndroidManifest.xml file and contains information about the application package and versioning.

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package com.example.myfirstapp

android:versionCode="1"

android:versionName-"1.0">

Attributes:

xmlns:android: The XML namespace declaration. This must be included in the roa element

package: The unique identifier for the application. This should match the package nams used in the Java or Kotlin code

android:versionCode: An integer value that represents the version code of the application

android:versionName: A string value that represents the release version of the application, shown to users.

3. <application> Element:

The <application element contains information about the application itself, such as its icos label, theme, and components

capplication

android:allowBackup="true"

android:icon="@mipmap/ic launcher"

android:label="@string/app_name"

android: roundIcon="@mipmap/ic launcher_round"

android:supportsRt1-"true"

android:theme="@style/AppTheme">

Attributes:

android:allowBackup: Specifies whether the application's data can be backed up an restored

android:icon: Specifies the icon for the application.

android:label: Specifies the default text label for the application.

android:roundicon: Specifies the round icon for the application.

android:supportsRtl: Indicates whether the application supports right-to-left (RTL)

layouts

android:theme: Specifies the default theme for the application.

4. <activity> Element:

Inside the <application>' element, <activity>' elements define the activities in the application. Each <activity> element represents a screen with a user interface.

cactivity android:name=".MainActivity">

cintent-filter>

<action android:name="android.intent.action.MAIN" />

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

</intent-filter>

</activity:

<activity android:name=". SecondActivity" />

Attributes:

android:name: Specifies the name of the activity class.

<intent-filter> Defines the intent filters for the activity, specifying how it can be launched. The example shows an activity that responds to the MAIN action and LAUNCHER category, making it the entry point of the application.

5. Services: <service>

Declares a service, which is a component that performs long-running operations in the background.

<service android:name=". MyService" />

Attributes:

android:name: Specifies the name of the service class.

6. Broadcast Receivers: <receiver>

Declares a broadcast receiver, which allows the application to listen for system-wide broadcast announcements.

<receiver android:name=".MyReceiver" />

Attributes:

android:name: Specifies the name of the receiver class

7. Content Providers: <provider>

Declares a content provider, which manages access to a structured set of data.

<provider android:name=". MyProvider

android:authorities="com.example.myapp.provider android:exported="false" />

Attributes:

android:name: Specifies the name of the provider class

android:authorities: Specifies the URI authority that identifies the provider.

android:exported: Specifies whether the provider is accessible by other applications.

8. Permissions:

The <uses-permission element is used to declare permissions required by the application to access certain features or resources.

ruses-permission android:name="android.permission.INTERNET" />

ruses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

Attributes:

android:name: Specifies the name of the required permission (e.g. android.permission

INTERNET)

9. Features: <uses-feature>

Declares the hardware and software features used or required by the application.

Kuses-feature android:name="android.hardware.camera" android:required="true" />

Attributes:

android:name: Specifies the name of the feature (eg, android.hardware.camera).

android-required: Indicates whether the feature is required to run the application.

14. a) How to create a style and theme in android? Explain with an example

The styles and themes are powerful tools to define and apply consistent visual styles across an application or specific components like activities. Understanding styles and themes is essential for creating a visually appealing user interface.

1. Styles:

Definition: A style in Android is a collection of properties that specify the look and format for a View or a window It can be used to share common attributes across multiple elements to ensure a consistent look and feel throughout the application

Usage: Styles can be applied to individual views in layout XML files or set programmatically in Java/Kotlin code

Purpose:

+ Reusability: Styles enable the reuse of a set of properties across different U components to promote a consistent design and reducing redundancy,

+ Customization: They allow for easy customization and updates Changing a style definition updates all components using that style.

Components:

+ Style Resource: Defined in an XML file under the res/values directory, typically in themes.xml

+ Attributes: Consist of key-value pairs that define various properties like layout width, layout height, textColor, background, etc.

2. Themes:

Definition: A theme in Android is a style applied to an entire Activity or application rather than an individual View Themes are used to define the look and feel of all elements within the scope they are applied to. Themes can include styles for various U components like buttons, text fields, dialogs, etc.

Purpose:

+ Global Consistency: Themes ensure a uniform appearance across the entire application or specific activities

+ Customization: They allow for the definition of global attributes such as color palettes, font styles, and dimensions, which can be easily changed in one place

• Components:

+ Theme Resource: The themes are defined in themes.xml file within the res values directory

+ Inheritance: Themes can inherit properties from other themes, allowing for hierarchical and layered customization.

Application Level vs. Activity Level: Themes can be applied at the application level in the AndroidManifest.xml file or at the activity level in layout XML files.

Customization: Themes can be customized to create unique visual experiences for different parts of the application.

Example

Using Activities, Fragments and Intents in Android

Applying a Style and Theme to an Activity

to define a simple style for a button and a theme for the s the layout, and the theme is set for Do nangle demonstrates hone to the button in applied application. The style the entire application in the manifest file. This

a consistent look and feel with minimal code Stop 1: Create Colors in colors.xml

3.13

Define the colors to be used in the style and theme in coloriamd file located under res/values holder

Code

<resources>

colors.xml

color name="colorPrimary">#6200EE</color>

<color name "colorFrimaryDark">#370003</color>

<color name="colorAccent">#03DACS</color>

</resources>

Sirp 2: Define a Simple Style and Theme in themes.xml file

Deline a single style and theme for demonstration purposes in themes.xml file located under res/

value/themes/themes.xml

Code

(resources>

themes.xml

Base application theme ->

< style name "AppTheme" parent"Theme. AppCompat.Light DarkActionBars citem name="colorPrimary">@color/colorPrimary</item>

citem name="colorPrimaryDark">@color/colorPrimaryDark</item> <item name="colorAccent">@color/colorAccent</item>

</style>

Custom Button style

style name "CustomButtonStyle">

<item name="android:layout_width">wrap_content</item> <item name android:layout height">wrap_content</items

<item name="android:background">@color/colorAccent</item>

<iten name="android:textColor">#FFFFFF</item>

<item name="android:padding">10dp</item>

</style>

</resources>

Step 3: Apply Styles to Layout in acivity main.xml

Apply the defined style to a button in acivity main.xml file located under res/layout

Code acivity main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<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"

tools context". MainActivity">
<Button

android:id="@+id/button"

style="@style/CustomButtonStyle"

android:text="Click Me"

android:layout_centerInParent="true"/>

</RelativeLayout>
```

Step 4: Apply the Theme in the Manifest

Set the theme for the entire application in the AndroidManifest.xml file.

Code

AndroidManifest.xml

manifest xmlns:android="http://schemas.android.com/apk/res/android" <

package com.example.myfirstapp">

<application

android:allowBackup="true"

android:icon="@mipmap/ic_launcher" android:label="@string/app_name"

android:roundIcon="@mipmap/ic_launcher_round"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=". MainActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

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

</intent-filter>

</activity>

</application>

</manifest>

Step 5: Implement MainActivity

Create the MainActivity java with basic setup.

Code

MainActivity.java

package com.example.myfirstapp;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);

)

b) Discuss the process of adding fragments dynamically in an Android application

1. Set Up the Project: Create a new Android project in Android Studio.

2. Create the Fragment Class: Define a fragment class that will be added dynamically

3. Create Fragment Layout: Define the layout XML file for the fragment.

4. Modify the Activity Layout: Add a container (FrameLayout) in the activity's layout file where the fragment will be dynamically added.

5. Add Fragment Dynamically in Activity: Use the FragmentManager and FragmentTransaction to add the fragment dynamically within the activity code.

15. List and briefly describe common views used in Android applications with their XML definitions

Some common views used in Android applications, along with their XML definitions are given below. These views are essential components in creating interactive and user-friendly interfaces in Android applications. Each view type has its own unique attributes and behaviors, allowing for flexible and dynamic Ul design.

1. TextView:

Description:

Displays text to the user.

Commonly used for labels, instructions, or any text content.

Code

<TextView

android:layout_width="wrap_content"

android:layout_height="wrap_content" android:text="Hello, World!"

android:textSize="20sp"

android:textColor="#000000"/>

Explanation

android:layout, width="wrap content"" The width of the TextView will be just enough to fit the content.

android:layout height="wrap_content"": The height of the TextView will be just enough to fit the content

"android text="Hello, World!" Sets the text displayed by the TestView

android:textSize="20sp Sets the size of the text to 20 scale-independent pixels (sp)

android textColor="#000000": Sets the color of the text to black.

2. Button:

Description:

A push-button that can be pressed or clicked by the user. Used to perform actions when clicked.
Code

<Button

android:layout width="wrap_content" android:layout height="wrap_content"

android:text="Click Me"/>

Explanation

android:layout width="wrap content"": The width of the Button will be just enough to fit the content

android layout height="wrap content" The height of the Button will be just enough to fit the content.

android:text="Click Me Sets the text displayed on the Button.

3. EditText:

Description:

A text field that allows users to input text.

Commonly used for forms and input fields.

Code

<EditText

android:layout_width="match_parent"

android:layout height="wrap_content"

android:hint="Enter text"/>

Explanation

android:layout, width="match parent"": The width of the EditText will expand to match the width of its parent container

android:layout height="wrap.content". The height of the Edir Text will be just enough to fit the content.

android hint="Enter text Sets a hint that is displayed when the EditText is empty, promptag the user for input

4. ImageView:

Description:

Displays an image resource.

Used for displaying images, icons, or any drawable resources

Code

<ImageView

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:src="@drawable/sample_image"/>

Explanation

'android layout, width="wrap content". The width of the Image View will be just enough to the content

android:layout height="wrap content": The height of the Image View will be just enough to the content.

android:src="@drawable/sample image: Sets the image resource to be displayed in the ImageView. The @drawable/sample image reference should point to an image resource the 'res/drawable' directory.

. CheckBox

• Description:

A checkbox that can be checked or unchecked.

• Used for binary choices or options in forms.

Code

CheckBox

android:layout width="wrap_content"

android:layout height="wrap_content"

android:text="Check me"/>

Explanation

android layout width="wrap content": The width of the CheckBox will be just enough to fit the content.

android layout height="wrap content" The height of the CheckBox will be just enough to fit the content

android:text="Check me". Sets the text displayed next to the CheckBox.

6. RadioButton:

Description:

A radio button, usually used in a group where only one option can be selected at a time

Commonly used in forms where a single choice must be made from multiple options.

Code

<RadioButton

android:layout width="wrap_content"

android:layout_height="wrap_content"

android:text="Option 1"/>

Explanation

android layout width="wrap content: The width of the RadioButton will be just enough to fit the content.

android layout height="wrap content"". The height of the RadioButton will be just enough to fit the content

android:text="Option 1: Sets the text displayed next to the RadioButton.

16. a) Explain Anchoring Views to Adapt Display Orientation

Anchoring waves is a layout design technique used to ensure that you are elements maintain their position relative to specific edges of the screen regardless of the device orientation this approach helps create consistent User experience by keeping important controls such as buttons and input fields in predictable locations

When anchoring views, we need to specify rules or constraints that dictate how a view should be Pontioned relative to its parent layout or other views. This technique is commonly used to by elements aligned to specific edges, corners, or positions on the screen to ensure a consistent layo across different screen sizes and orientations.

?f

What is Anchoring Views ?

Anchoring views in Android layout design refers to the technique of fixing the position of Ul elements relative to specific edges or positions on the screen. By anchoring views, we can ensure that certain elements remain in a consistent location regardless of changes in screen orientation to provide a stable and predictable layout for the user interface.

How It Works?

In Android XML layouts, anchoring views is typically achieved using a RelativeLayout, which allows to define the position of views relative to the parent layout or other sibling views. By setting the below attributes, we can anchor views to specific edges of the screen.

Anchoring Attributes:

1. 'android:layout alignParentBottom': This attribute positions the view at the bottom edge of its parent layout. When set to "true", the view will be anchored to the bottom of the parent layout to ensure it remains at the bottom regardless of other views or screen orientation changes

2. 'android:layout_alignParentLeft': Setting this attribute to "true" aligns the view with the lett edge of its parent layout. The view will be anchored to the left side of the parent, maintaining its position relative to the left edge even if other views are added or the screen orientation changes

3. 'android:layout alignParentRight': Similar to 'layout alignParentLeft', this attribute aligns the view with the right edge of its parent layout. By setting it to "true", the view will be anchored to the right side of the parent to ensure its position remains fixed relative to the right edge

4. 'android:layout alignParentTop': When this attribute is set to "true", the view is aligned with the top edge of its parent layout. Anchoring the view to the top ensures it stays at the top of the parent layout, maintaining its position regardless of other elements or orientation adjustments.

b) What Happens to an Activity's State When Orientation Changes?

. Activity Destruction and Recreation:

When the device orientation changes. Android destroys the current activity and creates a new instance of it. This process is essential to reload resources that match the new configuration, such as different layout files optimized for landscape or portrait mode

During this process, the 'onCreate() method is called again, setting up the new activity instance. This ensures that the activity is correctly configured for the new orientation, but it also means that the activity starts from scratch, losing any transient Ul state unless it's explicitly saved and restored

Example: Imagine an email application where a user is composing an email. If the user changes the device orientation while typing, the activity is destroyed and recreated: Without proper handling, the user might lose the text they were typing, which can be frustrating and disruptive

2. Loss of Transient State:

Transient Ul states, such as text entered in text fields, selected items in lists, and other temporary data, can be lost during the destruction process. This happens because the default behavior does not automatically save this transient state

Any unsaved data is discarded when the activity is destroyed. The new activity instance created after the orientation change will not have access to the previous state unless mechanisms are in place to save and restore in.

Example: Consider a shopping app where a user is filling out a checkout form. If the user rotates the device mid-process, all the information entered into the form fields could be lost, requiring the user to start over. This could lead to user frustration and potentially abandonment of the checkout process.

3. Persisting State:

To preserve the activity state across configuration changes. Android provides several mechanisms

onSaveInstanceState() Method: This method allows the activity to save its current Ul state into a Bundle before it is destroyed. When the activity is recreated, the saved state can be restored from the Bundle

onRestoreInstanceState() Method: This method can be used to restore the state from the Bundle that was saved in onSaveInstanceState(). It provides an opportunity to restore the Ul state after the onCreate() method has completed

Example: in a note-taking app, a user may be writing a note. If the orientation changes, the app should save the current note's text using the 'onSaveInstanceState() method. When the activity is recreated, the app retrieves the text from the 'Bundle' and restores it in the text field.

17. Explain the complete workflow of creating, reading, updating, and deleting data in a SQLite database with code examples.

18 Explain the Step by Step Implementation of Consuming XML Web Services.