Mobile Application Development.

Model Paper-01

→ 2 marks:

01. Briefly describe the evolution of mobile technologies

A:- Evolution of mobile technologies:

1. SMS
   - It stands for Short Message Service.
   - It is now the most widely used and oldest text messaging service.
2. MMS
   - It stands for Multimedia Messaging Service.
   - It is standard method of delivering multimedia material, including messages.
3. 1G [First Generation]
4. 2G [Second Generation]
5. 3G [Third Generation]
6. 4G [Fourth Generation]
7. GSM [Global System for Mobile technology]
8. CDMA [Code Division Multiple Access]
9. Wi-Fi [Wireless Fidelity]

02. What are the key components of an Android application?

A:- Key components of an Android application are:-

- Activities
- Services
- Broadcast Receivers
- Content providers

03. How do you launch your first Android application in the IDE?

A:- 1. Create a New Project

2. Create an Activity

3. Create a layout

4. Implement an event handler

04. Define an Intent in Android and its primary use.

A:- An Intent provides a facility for performing late runtime binding between the code in different applications.

(OR)

An Intent facilitates communication between different components within an app or even across different app

- primaryly used to navigate user to another activity or application to achieve a specific task.

05. Explain the role of the action bar in an android application.

A:- The role of the action bar is.

- Navigation
- Actions
- Branding
- Consistency
- Overflow menu

06. What is the purpose of using fragments in Android?

A:- the purpose of using fragment are:

- modularity
- Reusable UI components
- Flexible UI Design
- Lifecycle management
- Dynamic UI changes
- back stack management.

→5 marks:

07. Describe the steps involved in debugging an android application.

A:- Debugging

It is the process of finding and fixing errors or bugs in the source code of any software

The steps involved in debugging are:-

1. Enable Debugging
   - enable debugging on your device
   - Run a debuggable build variant
2. Start debugging
   - Set breakpoints in your app's code
   - In the toolbar, select a device to debug your app on from the target device menu. If you don't have any devices configured, then you need to create an AVD to use the Android Emulator
3. In the toolbar, click Debug
4. If the Debug window isn't open, select View>

Tool windows > Debug or click Debug in the tool window bar

Q5. How can you manage changes to screen Orientation in an Android application?

A:- Managing changes to screen orientation in an android application involves handling configuration changes and ensuring the app's UI & state adapt properly.

1. Handling configuration changes manually.

By default, activity restarts when the screen orientation changes, leading to a full recreation of the activity. To handle configuration changes manually without restarting the activity.

- Declare in your Android Manifest.xml that your activity will handle configuration changes:

xml

```
<activity
    android:name=".YourActivity"
    android:configChanges="orientation|screenSize">
</activity>
```

- Override onConfigurationChanged in your activity:

java

```
@Override
public void onConfigurationChanged(Configuration
                                   newConfig) {
    super.onConfigurationChanged(newConfig);
    if (newConfig.orientation == Configuration.
        ORIENTATION_LANDSCAPE) {
```

```
}
else if(newConfig.orientation == Configuration.
                ORIENTATION_PORTRAIT) {

}
}
```

2. Saving and Restoring State

If you let the activity restart on orientation changes, ensure you save the UI state and data properly

- Override onSaveInstanceState to save necessary state data

```
Java
@Override
protected void onSaveInstanceState(Bundle outState)
{
  super.onSaveInstanceState(outState);
  outState.putString("key", value);
}
```

- Restore the state in onCreate or onRestoreInst-anceState:

```
Java
@Override
protected void onCreate(Bundle savedInstanceState)
{
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   if(savedInstanceState != null) {
      String value = savedInstanceState.getString("key");
   }
}
```

```
@Override
protected void onRestoreInstanceState(Bundle
        savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
}
```

09. Explain the process of creating and using an SQLite database in an Android application.

A:- The process of creating and using an SQLite database

1. Create an SQLite Database:

SQLite is a lightweight, embedded database that comes ~~in A~~ with Android.

- Define a contract class: create a contract class that defines the database schema, including table names, column names and data types.
- Create a Helper class: Implement an SQLiteOpenHelper subclass. This class manages database creation, version management and provides methods for creating and upgrading tables.
- Override onCreate() Method: In the helper class, override the onCreate() method. this method is called when the database is created for the first time. Inside it, create your tables using SQL statements
- Override onUpgrade() method: If you need to modify the database schema, Override the onUpgrade() method.

2. Perform CRUD Operations:

CRUD stands for Create, Read, Update, Delete

- Insert Data : Use SQLiteDatabase.insert() or SQLiteDatabase.execSQL() to add data to your tables.
- Query Data : Retrieve data using SQLiteDatabase.query() or raw SQL queries.
- Update Data: modify existing data using SQLiteDatabase.update()
- Delete Data : Remove data using SQLiteDatabase.delete()

3. Access the Database

To access your database, create an instance of your helper class

java

```
MyDatabaseHelper dbHelper = new MyDatabaseHelper(
                                context);
SQLiteDatabase db = dbHelper.getWritableDatabase();
getReadableDatabase()
```

4. Perform Transactions

Use transactions to group multiple database operations into a single unit of work.

java

```
db.beginTransaction();
try {
  db.setTransactionSuccessful();
}
finally {
  db.endTransaction();
}
```

5. close the Database

Always close the database when you're done using it

```
db.close();
```

10. What are the benefits of using WebView in Android applications?

A:- Benefits of using web view are:-

- cost - Effective Implementation
  - It allows you to display web content within your app.
  - we can leverage existing web pages or web based services without building a separate native UI.
- Easier styling and Dynamic Content
  - Styling and layout changes can be applied to the web content without requiring a new app release.
- Cross-platform consistency
  - It allows you to share the same web based UI across platforms
  - It ensures an user experience on different devices.
- Integration of Online information
  - It enables seamless integration of online information into your native Android app.

11. Outline the steps to save and load user preferences in Android.

A:- Step 1:- Create a shared preferences file

- create a shared preferences file for your app.
- to get the shared preferences, call the getSharedPreferences() method
- SharedPreferences sharedpreferences = getSharedPref-erences("com.example.myapp-preferences", context.MODE_PRIVATE);

Step 2:- Write Data to Shared Preferences

- To write data to the shared preferences, create a SharedPreferences.Editor by calling edit() on your sharedpreferences object.
- SharedPreferences.Editor editor = sharedpreferences.edit();
- use methods like putInt(), putString() to set the key-value pair

Step 3:- Apply changes

- After setting the values, applying the changes using apply() or commit()
- editor.apply();
- editor.commit();

Step 4:- Read Data from shared preferences

- To retrieve data, simply use the same SharedPreferences object.
- int HighScore = sharedPreferences.getInt("highscore", 0);

Step 5:- Default values

- You can set default values for preferences using XML files
- To load default values, use PreferenceManager.setDefaultValues() in your onCreate() method

12. Discuss the importance of using alternate resources in Android development.

A:- Importance of using alternate resources are:-

1. Localization: It allow developers to provide different versions of strings, layouts and other resources for different locals.
2. ~~Custo~~ Screen Size and Density variations: developers can create different layouts and drawable resources optimized for small, medium, large & extra large screens as well as different pixel densities.
3. Configuration changes: It help handle these changes seamlessly by providing specific resources for each configuration.
4. Theming & Styling: It allow developers to define different themes & style for different parts of the app or for different types of devices.
5. Accessibility: It ensures better support for accessibility features.

-> 8 marks:

13. Discuss the process of publishing an Android application to the Google Play store.

A:- To publish your finished application on the Google Play store, you must generate a signed APK (Android Application package)

- The APK is the compiled, executable version of your application.
- Signing it is much like signing your name to a document. The signature identifies the app's developer to Google and the users who install your application.
- Unless your Android studio is in developer mode, unsigned applications will not run.

Step1:- Generate a signed APK from your code by selecting Build -> Generate Signed APK from the menu bar to bring up the Generate Signed APK window.

Step2:- Assuming you have never published an application from Android studio, you need to create a new key store. Click the create new button to display the New key Store window.

Step 3:- Fill out all of the information on this form because it pertains to your entity and application. Notice that there are two places for a password. These are the passwords for your key store & your key, respectively. Because a key store can hold multiple keys, it requires a separate password than that of the key for a specific app.

Step 4:- Click OK to return to the Generate Signed APK window.

Step 5: In the Generate Signed APK windows, click Next to review and finish the process.

14. Describe how to create a user interface programmatically in Android.

A:- • Create a new project by selecting
File -> New -> New Project
- Add some elements which are used to create UI in Android.
- Program

```
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import androidx.appcompat.app.AppCompatActivity;
public class MyActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        LinearLayout linearlayout = new LinearLayout(this);
        linearLayout.setOrientation(LinearLayout.VERTICAL);
        EditText editText=new EditText(this);
        editText.setHint("Enter text here");
        linearlayout.addView(editText);
        Button button = new Button(this);
        button.setText("Click me!");
        linearlayout.addview(button);
        setContentView(linearlayout);
    }
}
```

15. Explain the implementation of a content provider in Android and its uses.

A:- Content provider
- It server as a central repository for storing application data. They allow secure access & modification of data by other applications based on user requirements
- It can store data in various ways, including SQLite databases, files, or even on a network.
- It have permissions that grant or restrict access to data by other apps.
- Content URI is used as a query string to access data from a content provider.
- Syntax:-
  content://authority/optional Path/optional ID
  - content:// :- mandatory part indicating a content URI
  - authority :- Unique name of the content provider
  - Optional Path :- Specifies the type of data
  - Optional ID :- Numeric value for accessing specific records.

-> CRUD operations
- Create: Add data to a content provider.
- Read: Fetch data from a content provider.
- Update: modify existing data.
- Delete: Remove data from storage.

-> Working:

- UI components use cursor loader to send queries to the content Resolver.
- The content Resolver communicates with the content Provider, which processes requests & returns results.

-> creating a content provider:

1. Create a class extending the Content provider base class.
2. Define a content provider URI address to access the content.
3. Implement the six abstract methods of the content provider class:
   - query(): Retrieve data
   - insert(): Add new data
   - update(): Modify existing data
   - delete(): Remove data
   - getType(): Return MIME type
   - onCreate(): Initialize the provider.
4. Register the content provider in the Android Mainfest.xml using the <provider> tag

16. How can you consume JSON services in a Android application? Provide an example

A:- 1. create a New project:

Start by creating a new Android project in Android studio, selecting Java or kotlin as the programming language

2. UI setup:

In your activity_main.xml, create a listview to display the data.

Eg:- xml

```
<?xml version="1.0" encoding="utf-8"?>
<linearlayout
    xmlns:android="http://schemas.android.
                com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <listview
        android:id="@+id/user_list"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:divider height="1dp" />
</linear layout>
```

3. Create a layout Resource File

create another layout resource file to define how each item in the listview should look. Customize it according to your needs.

4. Fetching & parsing JSON Data

make an http request to a restful web service that returns JSON data

parse the JSON response using the JSONObject and JSONArray classes available in the org.json package.

5. Example

```
val apiURL = "https://api.example.com/users"
val listView = findViewById<ListView>(R.id.
                user_list)
val adapter = ArrayAdapter(this, android.R.
              layout.simple_list_item_1, userList)
ListView.adapter = adapter
```

17. Illustrate the steps involved in accessing the SD card for reading and writing local data.

A:- Step 1: Navigate to Settings

- Open the Notifications window by swiping down from the top of the screen.
- Tap on the Settings icon
- Alternatively, find the Settings app and tap that instead.
- ~~Select the SD card~~

Step 2: Access Storage Settings

- Scroll down (or use the search function) until you find Storage, then tap on it.

Step 3: Select the SD card

- Look for the SD card listed in the Portable Storage section
- Tap on the SD card to access its contents

Step 4: Mount or Unmount

- You can choose to mount the SD card (making it accessible) or unmount it (disconnecting it from the device)
- Formatting the SD card is also an option if needed.

18. Explain the concept of binding activities to services in Android and its significance

A:- Binding Activities

- It is a way for an activity to interact with a service so that it can send commands to the service, receive results and even perform inter-process communication (IPC) if the service is running in a different process.

=> Key concepts

- Service lifecycle :-

A started service is initiated by calling startService() and can run indefinitely, even if the component that started it is destroyed.

A bound service is bound to a client using bindService() and typically runs only as long as the client is bound to it.

- Binding to a Service :-

when an activity binds to a service, it uses the bindService() method. This method takes three parameters :

- Intent :- Specifies the service to bind to
- ServiceConnection :- A callback interface used to monitor the connection between the activity & the service
- Flags :- Options for the binding, such as BIND_AUTO_CREATE

- Service Connection Interface :

This interface has two key methods

- onServiceConnected(ComponentName, name, IBinder service) : called when a connection to the service has been established, providing the IBinder object to communicate with the service.
- onServiceDisconnected(ComponentName name) : Called when the connection to the service is unexpectedly disconnected.

- IBinder Interface :-

The IBinder interface is a core component in the binding mechanism. It provides a concrete interface for a bound service and allows the client to call methods on the service.

=> Significance of Binding Activites to a Services.

1. Efficient communication :

Binding allows efficient communication between an activity and a service.

2. Resource management :

Boond services help manage resources effectively. Since a bound service only runs while it has clients bound to it, it helps in conserving system resources, unlike a started service which continues to run until explicity stopped.

3. Inter-process communication (IPC) :

Binding is essential for IPC. If the service is running in a different process, the binding mechanism ensures that the activity can still interact with the service as if it were in the same process, using Android's binder framework.

4. Tight Coupling :

For scenarios where an activity needs to tightly couple with a service, binding is the preferred approach as it allows the activity to control and interact with the service lifecycle closely.

Eg:-

1. Define the service : Implement the service class and override the onBind() method to return an IBinder instance.

```
public class MyService extends Service {
    private final IBinder binder = new LocalBinder();
    public class LocalBinder extends Binder {
        MyService getService() {
            return MyService.this;
        }
    }
    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }
}
```

2. Bind to the service : In the activity, bind to the service using bindService()

```
public class MainActivity extends AppCompatActivity {
    MyService myService;
    boolean isBound = false;
    private ServiceConnection connection = new ServiceConn
            -ection() {
        @Override
        public void onServiceConnected(ComponentName
                className, IBinder service) {
            LocalBinder binder = (LocalBinder) service;
            myService = binder.getService();
            isBound = true;
        }
        @Override
        public void onServiceDisconnected(ComponentName
                arg0) {
            isBound = false;
        }
    };
    @Override
    protected void onStart() {
        super.onStart();
        Intent intent = new Intent(this, MyService.class);
        bindService(intent, connection, Context.BIND_AUTO_
                CREATE);
    }
    @Override
    protected void onStop() {
        super.onStop();
        if (isBound) {
            unbindService(connection);
            isBound = false;
        }
    }
}
```