Mobile Application Development

Model Questions

Short Answer Questions

1. **Fragments**

Ans: Fragments are modular sections of an activity, which have their own lifecycle and can be added or removed while the activity is running.

2. **Define Android**

Ans: Android is an open-source operating system based on Linux, designed primarily for touchscreen mobile devices such as smartphones and tablets.

3. **Describe mobile technologies**

Ans: Mobile technologies encompass a range of wireless communication systems including cellular networks (3G, 4G, 5G), Wi-Fi, Bluetooth, and near-field communication (NFC), enabling data transmission and connectivity for mobile devices.

4. **Data binding with example**

Ans: Data binding is a technique to bind UI components in layouts to data sources, allowing automatic updates of the UI when the data changes.

Example: <data>

<variable

name="user"

type="com.example.User"/>

</data>

<TextView android:text="@{user.name}"/>

5. **Different methods to store data in Android**

Ans: Shared Preferences

Internal Storage
External Storage
SQLite Databases
Content Providers
Network Connection

6. **Emulator in Android Studio**

- Linear Layout

- Relative Layout
- Constraint Layout
- Frame Layout
- Table Layout
- Grid Layout

7. Different layouts in Android

Ans: An emulator is a virtual device that mimics the hardware and software of a physical Android device, allowing developers to test and debug applications.

8. Splash screen. Why is it required in Android Studio

Ans: A splash screen is an initial screen that appears while the app is loading. It is used to improve user experience by displaying branding or a loading indicator during app startup.

9. SQLite

Ans: SQLite is a lightweight, embedded relational database management system included in Android, used for storing structured data locally on the device.

10. Content providers with respect to Android Operating System

Ans: Content providers manage access to a central repository of data, enabling data sharing between different applications.

Long Answer Questions

11. Life cycle methods of activities with examples

The lifecycle of an activity in Android consists of several methods that manage its state transitions. These methods ensure the activity behaves correctly as it moves through its lifecycle.

- **onCreate()**: Called when the activity is first created. It's where you initialize your activity. You usually set the content view and bind data to UI components here.

  Example:

  ```
  java
  Copy code
  @Override
  protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  // Initialization code here
  }
  ```

- **onStart()**: Called when the activity is becoming visible to the user. At this point, the activity is not yet interactive.

Example:

```
java
Copy code
@Override
protected void onStart() {
super.onStart();
// Code to start necessary processes
}
```

- **onResume()**: Called when the activity will start interacting with the user. The activity is now at the top of the activity stack and receiving user input.

  Example:

  ```
  java
  Copy code
  @Override
  protected void onResume() {
  super.onResume();
  // Code to resume operations paused in onPause()
  }
  ```

- **onPause()**: Called when the system is about to put the activity into the background. Use this method to pause ongoing actions that should not continue while the activity is not in the foreground.

  Example:

  ```
  java
  Copy code
  @Override
  protected void onPause() {
  super.onPause();
  // Pause ongoing actions
  }
  ```

- **onStop()**: Called when the activity is no longer visible to the user. This can happen because the activity is being destroyed, a new activity is starting, or the activity is going into the background.

  Example:

  ```
  java
  Copy code
  @Override
  protected void onStop() {
  super.onStop();
  // Release resources or save data
  }
  ```

- **onDestroy()**: Called before the activity is destroyed. This is the final call that the activity receives. Clean up any resources including threads, open connections, etc.

  Example:

```
java
Copy code
@Override
protected void onDestroy() {
super.onDestroy();
// Cleanup code
}
```

- **onRestart()**: Called after the activity has been stopped, just prior to it being started again.

  Example:

```
java
Copy code
@Override
protected void onRestart() {
super.onRestart();
// Code to restart the activity
}
```

12. **Google maps. What are the different methods of integrating a Google map with your mobile application**

Ans: Google Maps integration in an Android application can be done using the Google Maps Android API. Here are the steps to integrate Google Maps:

1. **Add the Google Maps dependency**: Add the following line to your `build.gradle` (app-level) file:

```
gradle
Copy code
implementation 'com.google.android.gms:play-services-maps:17.0.0'
```

2. **Get an API Key**: Obtain an API key from the Google Cloud Console and add it to your `AndroidManifest.xml`:

```
xml
Copy code
<meta-data
android:name="com.google.android.geo.API_KEY"
android:value="YOUR_API_KEY"/>
```

3. **Use a MapFragment or MapView**:
   - Using a `MapFragment` in your layout file:

```
xml
Copy code
<fragment
android:id="@+id/map"
android:name="com.google.android.gms.maps.SupportMapFragment"
android:layout_width="match_parent"
android:layout_height="match_parent"/>
```

   - In your activity:

```
java
Copy code
SupportMapFragment    mapFragment    =    (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
mapFragment.getMapAsync(new OnMapReadyCallback() {
@Override
public void onMapReady(GoogleMap googleMap) {
// Configure the map here
}
});
```

4. **Using MapView**:

```
xml
Copy code
<com.google.android.gms.maps.MapView
android:id="@+id/mapView"
android:layout_width="match_parent"
android:layout_height="match_parent"/>
```

- In your activity:

```
java
Copy code
MapView mapView = findViewById(R.id.mapView);
mapView.onCreate(savedInstanceState);
mapView.getMapAsync(new OnMapReadyCallback() {
@Override
public void onMapReady(GoogleMap googleMap) {
// Configure the map here
}
});
```

13. Toast with an example program

Ans: A Toast in Android is a small popup message that provides simple feedback about an operation.

Example program:

```
java
   Copy code
   import android.os.Bundle;
   import android.widget.Toast;
   import androidx.appcompat.app.AppCompatActivity;

   public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

   // Display a Toast message
   Toast.makeText(this, "Hello, World!", Toast.LENGTH_SHORT).show();
   }
   }
```

14. **Different components on the development environment of Android Studio**

Ans: Android Studio is the official IDE for Android development and includes several key components:

- **Code Editor**: Provides code completion, refactoring, and syntax highlighting for multiple languages.
- **Layout Editor**: A visual editor to design the UI of your app.
- **Emulator**: A virtual device to test your applications.
- **Logcat**: A tool for viewing system logs and debugging.
- **AVD Manager**: Manages Android Virtual Devices (emulators).
- **SDK Manager**: Manages different versions of the Android SDK and other libraries.
- **Build System (Gradle)**: Manages project build configurations and dependencies.
- **Profiler**: Tools for monitoring app performance, memory usage, and network activity.
- **Version Control Integration**: Supports Git, SVN, and other version control systems.

15. **Program to send email and sms messages in mobile application development**

Ans:

16. **Types of adaptors of Android Studio**

Ans: Adapters in Android are used to bind data to views like ListView, GridView, and RecyclerView. Types include:

- **ArrayAdapter**: Binds arrays to ListView.
- **BaseAdapter**: A base class for custom adapters.
- **CursorAdapter**: Binds data from a Cursor to ListView.
- **RecyclerView.Adapter**: Binds data to RecyclerView, allowing for more flexible and efficient handling of large datasets.

17. **Menu with its types supported by Android Studio?**

Ans: Menus in Android provide a way to present actions and options to users. Types include:

- **Options Menu**: Appears when the user presses the menu button or the app bar's overflow menu.
- **Context Menu**: Appears when the user performs a long-press on an item.
- **Popup Menu**: A small menu that anchors to a specific view and appears below it.

Creating an options menu:

```
java
Copy code
@Override
public boolean onCreateOptionsMenu(Menu menu) {
getMenuInflater().inflate(R.menu.menu_main, menu);
return true;
}
```

Creating a context menu:

```
java
Copy code
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo) {
super.onCreateContextMenu(menu, v, menuInfo);
getMenuInflater().inflate(R.menu.context_menu, menu);
}
```

18. Desktop publishing application (DTP)?

Ans: Desktop publishing applications are software tools used to create documents with a professional layout. These applications combine text, images, and graphics to produce items like brochures, newsletters, posters, and books. Examples include Adobe InDesign, QuarkXPress, and Microsoft Publisher.

19. List the different types of adaptors in Android Studio

Ans: Array **Adapter**: Simple adapter for single views.

Base Adapter: Provides a base class for custom adapters.
Cursor Adapter: Works with database cursors to populate ListViews.
Recycler View.Adapter: Used with RecyclerView for efficient list management.
Simple Adapter: Maps static data to views defined in an XML file.

20. How are menus created in Android Application

Ans: Menus are created by defining menu items in XML and inflating them in activities or fragments.

Define a menu in XML (res/menu/menu_main.xml):

xml

Copy code

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

<item android:id="@+id/action_settings"

android:title="Settings"

android:orderInCategory="100"

android:showAsAction="never"/>

</menu>

Inflate the menu in an activity:

java

Copy code

@Override

```
public boolean onCreateOptionsMenu(Menu menu) {

getMenuInflater().inflate(R.menu.menu_main, menu);

return true;

}
```

21. Structure of manifest.xml file

Ans: The `AndroidManifest.xml` file is the central configuration file for an Android app. It includes:

- **`<manifest>`**: Root element, includes the package name and permissions.
- **`<application>`**: Defines the application components and properties.
- **`<activity>`**: Declares an activity.
- **`<service>`**: Declares a service.
- **`<receiver>`**: Declares a broadcast receiver.
- **`<provider>`**: Declares a content provider.
- **Permissions**: Defines permissions needed by the app.

Example:

```
xml
Copy code
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.app">

<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>

<uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

22. Different user interface components used in Android

Ans: **TextView**: Displays text to the user.

EditText: An editable text field.
Button: A push button for user interaction.
ImageView: Displays images.
ListView: Displays a scrollable list of items.

RecyclerView: An advanced version of ListView for large datasets.
CheckBox: A two-state button that can be checked or unchecked.
RadioButton: Allows the user to select one option from a set.
Spinner: A drop-down list for selecting an item.
ProgressBar: Displays a progress indicator.

23. Different versions of Android operating system with API level

Ans: **Android 1.0 (API Level 1)**

Android 1.1 (API Level 2)
Android 1.5 Cupcake (API Level 3)
Android 1.6 Donut (API Level 4)
Android 2.0/2.1 Eclair (API Levels 5-7)
Android 2.2 Froyo (API Level 8)
Android 2.3 Gingerbread (API Levels 9-10)
Android 3.0-3.2 Honeycomb (API Levels 11-13)
Android 4.0 Ice Cream Sandwich (API Levels 14-15)
Android 4.1-4.3 Jelly Bean (API Levels 16-18)
Android 4.4 KitKat (API Level 19)
Android 5.0-5.1 Lollipop (API Levels 21-22)
Android 6.0 Marshmallow (API Level 23)
Android 7.0-7.1 Nougat (API Levels 24-25)
Android 8.0-8.1 Oreo (API Levels 26-27)
Android 9.0 Pie (API Level 28)
Android 10 (API Level 29)
Android 11 (API Level 30)
Android 12 (API Level 31)
Android 13 (API Level 33)

24. Action bar. How can it be manipulated in Android Studio

Ans: **Action Bar. How Can It Be Manipulated in Android Studio**

The action bar is a crucial UI element that provides navigation and action options to the user. It can be manipulated to fit the needs of the application.

- **Customizing the Action Bar**:

```
java
Copy code
ActionBar actionBar = getSupportActionBar();
if (actionBar != null) {
actionBar.setTitle("My Title");
actionBar.setDisplayHomeAsUpEnabled(true);
actionBar.setHomeButtonEnabled(true);
}
```

- **Adding Action Items**: Define items in the menu resource file (`res/menu/menu_main.xml`):

```
xml
Copy code
<menu xmlns:android="http://schemas.android.com/apk/res/android">
<item android:id="@+id/action_settings"
android:title="Settings"
android:orderInCategory="100"
android:showAsAction="never"/>
</menu>
```

Inflate the menu in the activity:

```
java
Copy code
@Override
public boolean onCreateOptionsMenu(Menu menu) {
getMenuInflater().inflate(R.menu.menu_main, menu);
return true;
}
```

- **Handling Action Item Clicks**:

```
java
Copy code
@Override
public boolean onOptionsItemSelected(MenuItem item) {
int id = item.getItemId();
if (id == R.id.action_settings) {
// Handle the settings action
return true;
}
return super.onOptionsItemSelected(item);
}
```

- **Using Toolbar as Action Bar**: Define the Toolbar in the layout:

```
xml
Copy code
<androidx.appcompat.widget.Toolbar
android:id="@+id/toolbar"
android:layout_width="match_parent"
android:layout_height="?attr/actionBarSize"
android:background="?attr/colorPrimary"/>
```

Set it as the action bar in the activity:

```
java
Copy code
Toolbar toolbar = findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
```