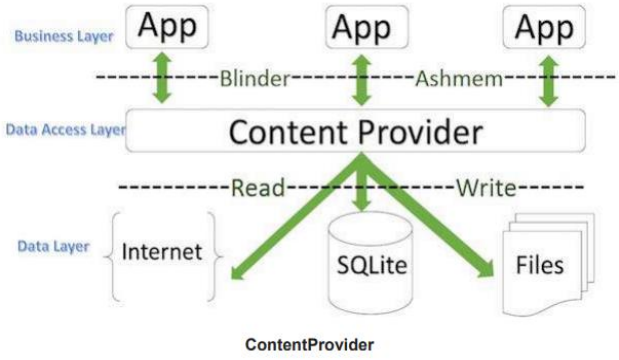
Unit -4 Important Questions

1. Explain content provider?
2. Metion the types of maps?
3. What is database Helper Class?
4. What is data binding?
5. Metion the different event handlings?
6. Mention the different ways to access the SD cards in an android?
7. Create an Application to send Email.
8. Create an Application to develop Login Window Using UI Controls.
9. Explain the steps of generating APK files?
10. What is the use of APK file?
11. Explain the Role Of GPS?
12. Explain the Steps to create, open, and close the database cursor?

1. Explain content provider?

A) A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentReslover class. It's a way to share data between different applications securely. A content provider can use different ways to store its data and data can be stored in a database, in files, or even over a network.



Here's a breakdown of the storage options:

1. **Database**: The content provider can store its data in a database, typically a SQLite database. This is a common method because databases are efficient for handling structured data and complex queries.

2. **Files**: Data can also be stored in files. This method might be used for handling unstructured data, such as media files (images, videos) or large datasets.
3. **Network**: A content provider can also store data over a network. This could mean the data is fetched from a remote server or cloud storage.

Android ships with many useful content providers, including the following:

➤ Browser—Stores data such as browser bookmarks, browser history, and so on
➤ CallLog—Stores data such as missed calls, call details, and so on
➤ Contacts—Stores contact details
➤ MediaStore—Stores media files such as audio, video, and images
➤ Settings—Stores the device's settings and preferences

Content URI is the key concept of Content Providers. To access the data from a content provider, Content URI is used as a query string.
Structure of a Content URI is **Content: //authority /optionalPath /optionalID**

1. **Content:** It represents that the given URI is content URI.
2. **Authority:** Name of the Content provider like contacts, browser, and etc.It must be unique for every content provider.
3. **OptionalPath:** Specifies the type of data provided by the Content provider. For example, if you are getting all the contacts from the **Contacts** content provider, then the data path would be people and URI would look like this **content://contacts/people**
4. **OptionalID:** It is a numeric Value it specifies the specific record requested. For example, example, if you are looking for a contact number 5 in the Contacts content provider then URI would look like this **content: //contacts/people/5.**

2. Mention the types of maps?

A) Android provides facility to integrate Google map in our application. Google map displays your current location, navigate location direction, search location etc. It allows users to interact with dynamic maps that provide detailed geographical information. Users can view their current location, search for specific places, get directions for navigation, and explore points of interest.
   - Google Maps provide various types; each of these map types provides a unique way of viewing and interacting with the Map.

1. **Normal View:** This is the Standard roadmap view, which displays streets, road names and geographical features.
   googleMap.setMapType (GoogleMap.Map_TYPE_NORMAL);

2. **Satellite View:** This view displays the satellite images of the area, showing detailed and realistic images of the terrain and buildings.
   googleMap.setMapType (GoogleMap.Map_TYPE_SATELLITE);
3. **Terrain View:** This View emphasizes terrain information, showing topography and physical landscape features such as mountains, valleys, and bodies of water.
   googleMap.setMapType (GoogleMap.Map_TYPE_TERRAIN);
4. **Hybrid View:** This View Combines the normal roadmap view with satellite images, providing street names and geographical features overlaid on real world imagery.
   googleMap.setMapType (GoogleMap.Map_TYPE_HYBRID);
5. **None:** Displays no base map tiles, allowing developers to overlay custom data on a blank canvas.
   googleMap.setMapType (GoogleMap.Map_TYPE_NONE);

3. What is database Helper Class?

A) SQLite is a C-language library that provides a relational database management system i.e. used to perform database operations on android devices such as storing, manipulating or retrieving persistent data from the database. It is self-contained, serverless, zero-configuration, and transactional.In Android, it is the default database engine used for local data storage.

Database Helper Class: Define the helper class for database creation and management.

```
public class MyDatabaseHelper extends SQLiteOpenHelper {
   private static final String DATABASE_NAME = "mydatabase.db";
   private static final int DATABASE_VERSION = 1;

   public MyDatabaseHelper(Context context) {
      super(context, DATABASE_NAME, null, DATABASE_VERSION);
   }

   @Override
   public void onCreate(SQLiteDatabase db) {
      String CREATE_TABLE_USERS = "CREATE TABLE users (id INTEGER
PRIMARY KEY, name TEXT, age INTEGER)";
      db.execSQL(CREATE_TABLE_USERS);
   }

   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
```

```
        db.execSQL("DROP TABLE IF EXISTS users");
        onCreate(db);
    }
}
```

Inserting Data

To insert data into the database:

```
public void insertUser(String name, int age) {

    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();

    values.put("name", name);

    values.put("age", age);

    db.insert("users", null, values);

    db.close();

}
```

4. What is data binding?

A) Data Binding is a feature in Android that allows you to bind UI components in your layouts to data sources in your app using a declarative format rather than programmatically.

Enable Data Binding: Modify the build.gradle file to enable data binding.
 Create a Layout with Data Binding: Wrap the root layout element in a <layout> tag and define data variables.
 Create a Data Model: Create a class that represents the data to be bound to the layout.
 Bind Data in Activity: Use DataBindingUtil to set up data binding in the activity and bind the data model to the layout.

1. *Enable Data Binding*

In your project's build.gradle file, enable data binding:

```
android {
    ...
```

```
    buildFeatures {
        dataBinding true
    }
}
```

2. Create a Layout with Data Binding

Modify your layout XML file to use data binding by wrapping the root layout element in a <layout> tag. Define a data variable that will be bound to the layout.

activity_main.xml:

```
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable
            name="user"
            type="com.example.myapp.User" />
    </data>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:padding="16dp">

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

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{String.valueOf(user.age}" />

    </LinearLayout>
</layout>
```

3. Create a Data Model

Create a data model class that will be used in the data binding.

User.java:

```
package com.example.myapp;

public class User {
    private String name;
    private int age;

    public User(String name, int age) {
        this.name = name;
```

```
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

4. Bind Data in Activity

In your activity, set up data binding and bind the data model to the layout.

MainActivity.java:

```
        // Set up data binding
    ActivityMainBinding binding=DataBindingUtil.setContentView (this, R.layout.activity_main);

        // Create a User object
        User user = new User("John Doe", 30);

        // Bind the User object to the layout
        binding.setUser(user);
    }
}
```

5. Mention the different event handlings?

A) In Android development, event handling refers to the mechanism of capturing and responding to user interactions or system events within an application. Here are some common types of event handling techniques used in Android:

Button Click Event Handling:

- Handles user clicks on buttons or other clickable views.
- Defined in XML using the android:onClick attribute or programmatically using setOnClickListener.

1. Button Click Event Handling

This is the most common type of event handling where you handle user clicks on buttons or other clickable views.

XML Layout (activity_main.xml):

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click Me"
    android:onClick="onButtonClick" />
```

- **Activity Code (MainActivity.java)**:

```
public void onButtonClick(View view) {
    // Handle button click here
// Example: Toast.makeText(this, "Button Clicked", Toast.LENGTH_SHORT).show();
    }
```

Menu Item Click Event Handling:

- Handles clicks on options menu items or action bar items.
- Implemented in the activity using onOptionsItemSelected method to detect menu item selection.

Touch Event Handling:

- Captures touch gestures like tap, swipe, and pinch.
- Implemented by overriding onTouchEvent method in custom views or gesture detectors (GestureDetector).

List Item Click Event Handling:

- Handles clicks on items in a ListView, RecyclerView, or similar list-based views.
- Implemented using OnItemClickListener for ListView or RecyclerView click listeners for RecyclerView.

Text Change Listener:

- Monitors changes to text input fields like EditText.
- Implemented using TextWatcher interface methods (beforeTextChanged, onTextChanged, afterTextChanged) to track text changes.

Broadcast Receiver:

- Listens for system-wide broadcast messages sent by the system or other apps.
- Registered in the manifest file or dynamically in the activity to handle specific broadcast intents (IntentFilter).

6. Mention the different ways to access the SD cards in an android?

A) Simple Ways to Access the SD Card (External Storage) in Android

1. **Using Environment.getExternalStorageDirectory()**:

   This method provides access to the primary external storage directory. It's straightforward but note its behavior changes on Android 10 (API level 29) and above.

   File externalStorageDir = Environment.getExternalStorageDirectory();

   Note: On newer Android versions (API level 29+), this method returns a directory specific to your app, not shared storage. For shared storage, consider using Storage Access Framework (SAF).

2. **Using Storage Access Framework (SAF)**:

   SAF allows users to access documents and other files across different storage providers through a system UI. This is essential for Android 10 and above due to scoped storage restrictions.

   - **Opening a Directory Picker**:

     Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT_TREE);
     startActivityForResult(intent, REQUEST_CODE_OPEN_DIRECTORY);

   - **Accessing Documents**:

     Once a directory is chosen, you can access files using the content URI returned by SAF.

3. **Permissions**:

Ensure your app has the necessary permissions declared in the manifest:

<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

Starting from Android 10, you also need to handle permissions for scoped storage and use SAF for accessing files outside your app's sandbox.

7. Create an Application to send Email?

Step By Step sending an implementation of sending a email:

Step: 1 Project Setup

Create an android project by selecting "Empty Views activity" and naming the project "email-example" and Choosing java as a language.

Step: 2 Create the UI

Design the UI to input the recipients email address, subject, and message body. This UI will allow users to enter the necessary details to send an email.

activity main.xml code:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android:padding="16dp">

<EditText
android:id="@+id/email_address"
android:layout width="match_parent"
android:layout_height="wrap_content"
android:hint="Email Address"
android: inputType="textEmailAddress" />

<EditText
android:id="@+id/email_subject"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:hint="Subject"
android: inputType="text" />

<EditText
```

```
android:id="@+id/email_message"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:hint="Message"
android: inputType="textMultiline" />

<Button
android:id="@+id/send_email"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Send Email" />

</LinearLayout>
```

Explanation:
* The LinearLayout' provides a vertical layout for the Ul components.
* The 'EditText' fields allow the user to input the recipient's email address, subject, and message content.
* The "Button' triggers the email-sending functionality when clicked.

Step3. Implement Main Activity

Implement the activity to handle sending emails using intents. This activity manages the user Interactions and handles the email-sending process.

```
public class MainActivity extends AppCompatActivity (
private EditText emailAddress;
private EditText emailsubject;
private EditText emailMessage;
private Button sendEmail;
@Override
protected void onCreate(Bundle savedInstanceState)
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 emailAddress = findViewById(R.id.email_address);
 emailSubject = findViewById(R.id.email_subject);
 emailMessage = findViewById(R.id.email_message);
 sendEmail = findViewById(R.id.send_email);
 sendEmailButton.setOnClickListener(new View.OnClickListener() {
 @Override
  public void onClick(View v) {
    sendEmail();

            }
```

```
    });
}

private void sendEmail() {

String emailAddress = emailAddress.getText().toString();
String subject = emailSubject.getText().toString();
String message = emailMessageEditText.getText().toString();

Intent emailIntent = new Intent(Intent.ACTION_SEND);
emailIntent.setType("message/rfc822");
emailIntent.putExtra (Intent.EXTRA_EMAIL, new String[] {emailAddress});
emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
emailIntent.putExtra (Intent.EXTRA_TEXT, message);
if (emailIntent.resolveActivity(getPackageManager()) != null) {
startActivity(emailIntent);
 }
else {
Toast.makeText (this, "Email Client Not Installed",
Toast.LENGTH_SHORT).show ();
    }
}
```

8. Create an Application to develop Login Window Using UI Controls.

A) For Answer refer Model paper2 15th Question.

9. Explain the steps of generating APK files?

A) For Answer refer Model paper1 14th Question.

10. What is the use of APK file?

A) For Answer refer Model paper2 17th Question.

11. Explain the Role Of GPS?

A) For Answer refer Model paper1 15th Question.

12. Explain the Steps to create, open, and close the database cursor?

A) For Answer refer Model paper1 17th Question.