Questions

1. Write down the step to change the Icon for android app?

Ans-:-To change the app icon in Android, follow these steps:

Prepare App Icons:

Create app icons in various sizes for different screen resolutions. Common sizes include 48x48, 72x72, 96x96, 144x144, and 192x192 pixels.

Place Icons in the Res Directory:

Place the generated icons in the res/mipmap directory of your Android project. Ensure icons are appropriately named (e.g., ic\_launcher.png).

Update Manifest File:

Open the AndroidManifest.xml file and locate the <application> element.

Modify the android:icon attribute to reference your new icon resource. Example: android:icon="@mipmap/ic\_launcher"

Sync and Build:

Sync your project with Gradle by clicking "Sync Now" if prompted.

Rebuild your project to apply the changes.

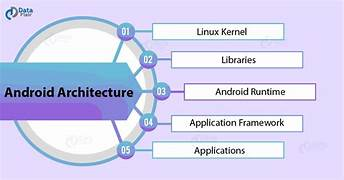
Run the App:

Run the app on an emulator or physical device to see the updated app icon.

Ensure to follow the naming conventions for your app icons and choose appropriate sizes to support different screen densities.

1. Draw and explain the architecture of android.

Ans:-



Android architecture is a layered framework comprising:

**Linux Kernel:** Provides core system functionalities like memory management and device drivers.

Android is built on top of the Linux kernel. The Linux kernel serves as the core of the Android operating system, providing essential services such as memory management, process management, security, and device drivers. The collaboration between the Android Open Source Project (AOSP) and the Linux community ensures that Android can take advantage of the latest features, improvements, and security updates in the Linux kernel.

Android versions are associated with specific Linux kernel versions, but it's important to note that different devices might run slightly different versions due to manufacturer customizations and optimizations. The Linux kernel version used in Android can be found by looking at the kernel version on a specific device or by exploring the source code of the Android Open Source Project.

As of my last knowledge update in January 2022, Android 7 (Nougat) typically used Linux kernel version 3.18 or 4.x, but this can vary depending on the device and any updates or modifications made by manufacturers.

If you're interested in the specific Linux kernel version used in a particular Android device, you can follow the steps mentioned earlier to check the device settings or use the uname -a command in a terminal or shell on the device.

**Hardware Abstraction Layer (HAL):** Bridges the gap between the kernel and device hardware, ensuring compatibility. Native Libraries: Essential libraries for functionalities like graphics rendering, database access, and web browsing.

**Android Runtime (ART/Dalvik):** Executes and manages Android applications' bytecode.

Android Runtime (ART) is the default runtime environment used by the Android operating system for executing applications written in the Java programming language. It was introduced in Android 4.4 (KitKat) as a replacement for the earlier Dalvik runtime. ART is responsible for converting Android application bytecode into native machine code that can be executed by the device's hardware.

Here are key aspects of Android Runtime (ART):

1. **AOT Compilation:**

One significant difference between ART and its predecessor, Dalvik, is the compilation approach. ART uses Ahead-of-Time (AOT) compilation, whereas Dalvik used Just-In-Time (JIT) compilation. With AOT, the app's bytecode is translated into native machine code at the time of installation, whereas JIT performed this translation at runtime.

1. **Improved Performance:**

AOT compilation can lead to improved application performance compared to Dalvik's JIT compilation. By converting bytecode to native machine code during installation, the runtime overhead during app execution is reduced.

1. **Enhanced Garbage Collection:**

ART features a more efficient garbage collection mechanism compared to Dalvik. The garbage collector in ART is designed to minimize interruptions and improve overall system responsiveness by reducing the frequency and duration of garbage collection pauses.

1. **Optimizations:**

ART includes various optimizations aimed at improving the execution speed and overall efficiency of Android applications. These optimizations include in lining, loop unrolling, and other techniques to make the compiled code more efficient.

1. **64-Bit Support:**

ART supports 64-bit architectures, allowing Android devices to take advantage of the increased memory address space and improved performance offered by 64-bit systems.

1. **Dex2Oat Tool:**

The Dex2Oat tool is used in the ART compilation process. It takes the Dalvik Executable (DEX) files, which contain the compiled bytecode of Android applications, and converts them into an optimized form known as the OAT (Optimized Android Application Package) file.

1. **Compatibility:**

While ART brought significant improvements, it maintained compatibility with existing Android applications developed for Dalvik. Applications developed for older versions of Android can still run on devices using ART without modification.

Overall, Android Runtime (ART) plays a crucial role in the execution of Android applications, providing performance enhancements and optimizations to deliver a smoother user experience on Android devices.

**Application Framework:** Offers high-level services such as activity management, content providers, and the user interface. System Apps: Pre-installed applications providing core functionalities like phone, contacts, and messaging.

**Application Layer:** User-installed apps that leverage the Android framework for diverse functionalities. This architecture supports an open ecosystem and diverse device implementations.

1. **Discuss the history of Software Development.**

Ans:- Early Programming Languages (1950s-1960s): The history of software development begins with machine and assembly languages.

Machine language is the lowest-level programming language that directly corresponds to the hardware architecture of a computer. It consists of binary code, represented by sequences of 0s and 1s, which the computer's central processing unit (CPU) can directly execute. Each instruction in machine language is a specific command that the CPU understands and processes.

Key characteristics of machine language:

Binary Representation: Machine language instructions are written in binary, which is the most fundamental level of representation in a computer system.

Hardware Specific: Machine language is specific to the computer architecture and is not portable between different types of CPUs.

Direct Interaction with Hardware: Machine language instructions directly control the operations of the computer's hardware components, including the CPU, memory, and input/output devices.

Lack of Abstraction: There is minimal abstraction in machine language. Instructions correspond closely to the actual operations performed by the hardware.

**Assembly Language:**

Assembly language is a low-level programming language that serves as a human-readable representation of machine language. It uses mnemonics and symbols to represent the machine language instructions, making it easier for programmers to write and understand code. Each mnemonic corresponds to a specific machine language instruction.

Key characteristics of assembly language:

**Symbolic Representation**: Assembly language provides a symbolic representation of machine language instructions, making it more readable and easier to understand than raw binary code.

**Mnemonics**: Mnemonics are short codes or symbols that represent machine instructions. For example, "MOV" might represent a move instruction, and "ADD" might represent an addition instruction.

**One-to-One Correspondence**: Each assembly language instruction typically corresponds directly to one machine language instruction. As a result, assembly language is still closely tied to the underlying hardware architecture.

**Low-Level Abstraction**: While assembly language is more readable than machine language, it is still considered a low-level programming language. Programmers need to have a good understanding of the hardware architecture.

**Assembler:** To convert assembly language code into machine code, an assembler is used. The assembler translates the symbolic instructions into the binary code that the computer's CPU can execute.

Both machine language and assembly language are considered low-level languages, providing a level of abstraction close to the hardware. While they are not as user-friendly as high-level languages, they offer more direct control over the hardware and are essential for tasks that require precise control over system resources. High-level languages are often used for application development due to their readability and portability, with compilers or interpreters translating the code into machine or assembly language for execution on the target hardware.

**High-Level Languages and Compilers (1960s-1970s):** The development of high-level programming languages like FORTRAN, COBOL, and C allowed for more abstraction and ease of coding. Compilers translated code written in these languages into machine code, making programming more accessible.

**Graphical User Interfaces (1980s-1990s):** The advent of graphical user interfaces (GUIs) in the 1980s, popularized by operating systems like Windows and Macintosh, transformed software development.

**Internet and Web Development (1990s-Present):** The rise of the internet in the 1990s brought about a significant shift. Web development became a focal point with the creation of HTML, HTTP, and scripting languages like JavaScript.

**Mobile and Cloud Computing (2000s-Present):** The 2000s marked the rise of mobile computing, leading to the development of mobile applications. Simultaneously, cloud computing gained prominence, enabling scalable and distributed software solutions.

1. **What is Intent? Explain Types of Intent with suitable Example.**

Ans:- In Android, an Intent is a messaging object that is used to request an action from another component of the application. It can be used to start an activity, broadcast a message, or deliver a command to a service. Intents facilitate communication between different components, such as activities, services, and broadcast receivers, within an Android application or between different applications.

There are two main types of Intents in Android:

**Explicit Intent:**

Definition: An explicit intent specifies the target component by its class name. It is used when you have a specific target component in mind and want to directly invoke it.

Example:

Intent explicitIntent = new Intent(CurrentActivity.this, TargetActivity.class);

startActivity(explicitIntent);

In this example, TargetActivity is explicitly specified as the target component for the intent.

**Implicit Intent:**

**Definition:** An implicit intent does not specify the target component by its class name but instead describes the type of action to be performed. The system then determines the appropriate component to handle the intent based on the available components that have declared their ability to handle that type of action.

Example:

Intent implicitIntent = new Intent(Intent.ACTION\_VIEW, Uri.parse("https://www.example.com"));

startActivity(implicitIntent);

In this example, the implicit intent is requesting an action to view a web page. The system will launch the appropriate activity (e.g., a web browser) based on the available components that have declared support for the ACTION\_VIEW action.

Intents can also be categorized based on their purpose:

Activity Intents: Used to start a new activity within the application.

Service Intents: Used to start a service in the background.

Broadcast Intents: Used to send system-wide announcements or events.

Explicit Intents: Targeting a specific component within the application.

Implicit Intents: Allowing the system to determine the appropriate component based on the action and data provided.

**Example**

MainActivity.java

package com.mastercoding.layoutapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;

import android.net.Uri;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.Toast;

//you can use this program for the LAB no 14 also

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// Intents: facilitates communication bet. different components of an application.

// types of intents:

// 1- Explicit Intents

Button btn = findViewById(R.id.btn);

btn.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

goToSecondActivity();

}

});

/\* 2- Implicit Intents

Button btn2 = findViewById(R.id.openBrowser);

btn2.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

openWebPage();

}

});

\*/

}

}

public void goToSecondActivity(){

Intent intent = new Intent(this, SecondActivity.class );

startActivity(intent);

}

public void openWebPage(){

Uri webpage = Uri.parse("https://www.google.com");

Intent intent = new Intent(Intent.ACTION\_VIEW, webpage);

startActivity(intent);

}

}

// Enable JavaScript (optional, depending on your use case)

//WebSettings webSettings = webView.getSettings();

//webSettings.setJavaScriptEnabled(true);

// Load a URL in the WebView

//webView.loadUrl("https://www.example.com");

public void goToSecondActivity(){

Intent intent = new Intent(this, SecondActivity.class );

startActivity(intent);

}

public void openWebPage(){

Uri webpage = Uri.parse("https://www.google.com");

Intent intent = new Intent(Intent.ACTION\_VIEW, webpage);

startActivity(intent);

}

}

// Enable JavaScript (optional, depending on your use case)

//WebSettings webSettings = webView.getSettings();

//webSettings.setJavaScriptEnabled(true);

// Load a URL in the WebView

//webView.loadUrl("https://www.example.com");

SecondActivity.java

package com.mastercoding.layoutapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class SecondActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_second2);

}

}

**Main\_activity.xml**

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:app="http://schemas.android.com/apk/res-auto" xmlns:tools="http://schemas.android.com/tools" android:layout\_width="match\_parent" android:layout\_height="match\_parent" tools:context=".MainActivity">

<Button android:id="@+id/btn"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Go to Second Activity"

app:layout\_constraintBottom\_toBottomOf="parent" app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent" app:layout\_constraintTop\_toTopOf="parent"/>

<Button android:id="@+id/openBrowser"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginStart="26dp"

android:layout\_marginBottom="91dp"

android:text="Go to Google"

app:layout\_constraintBottom\_toTopOf="@+id/btn" app:layout\_constraintStart\_toStartOf="@+id/btn"/>

</androidx.constraintlayout.widget.ConstraintLayout>

**Activity\_second2.xml**

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:app="http://schemas.android.com/apk/res-auto" xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".SecondActivity">

<TextView android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginTop="292dp"

android:layout\_marginEnd="4dp"

android:text="Welcome to Second Activity"

android:textSize="32sp"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintTop\_toTopOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>

1. **Explain the services in android with suitable example.**

**Ans-** There are two types of services bound and unbound services. Let's explore the lifecycle of both types of services. Let's start with the unbound service.

Unbound services are started using the Start service method and operate independently of components that started them. They can continue running even if the component that started them. The Oncreate method is called when the service is initially created.

It's used for one time initialization tasks.

The Onstart Command Method is called when the service is starting.

Using the Start service method, it provides the intent path to the start service method and other parameters.You perform background tasks. In this method, you can return values like start sticky or start not sticky to indicate how the system should handle the service.

If it is killed due to the resource constraints, then the service is stopped by itself or a client

and the service is stopped. So This onDestroy() is called when the service is about to be destroyed.You should release any resources you've acquired and perform clean-up tasks.

The second type of services is the bound services.

The bound services are usually used to establish a connection between components, activities and the service. They are started using the Bind service method and are closely tied to the components that bound to them.

When all components unbind from the service, the service is automatically destroyed.

The Oncreate method as in unbound services.

This method is called when the service is initially created.

The unbind method and bind method is called when a component binds to the service.

The service can perform, set up and provide the client with this binder.

This allows the client to interact with the service and all clients and bind by calling unbind.

Service method on unbind method is called when all components have unbound from the service.You might return it through here if you want the service to continue running even no more components are bound, but typically returning false leads to the service being destroyed when unbound on rebind method.

This method is called when a component that was already bound to the service rebinds using the bind.service method and the Ondestroy method similar to unbound services.This method is called when the service is about to be destroyed.Perform cleanup tasks here.

To summarize, the unbound services can run independently of components and continue running even if the starting component is destroyed.

Bound services are more tightly connected to the components that bind to them.

They are automatically destroyed when all bound components unbind from them.

Be cautious about managing the service lifecycle properly.

If you don't manage it correctly, it can lead to resource leaks or unintended behaviour in your application.

Bound services and unbound (or started) services in Android serve different purposes and are suitable for different use cases. Here are some common use cases for each type of service:

Unbound (Started) Services:

Background Operations without Direct Interaction:

Use Case: If you have a task that needs to run in the background, such as downloading a file, performing periodic updates, or syncing data, you can use an unbound service.

Explanation: The service can be started by an application component (e.g., an activity) using startService(). It will continue running until the task is completed or until explicitly stopped.

Foreground Services:

Use Case: For long-running operations that need to continue even if the app is in the background, you can use an unbound service with foreground capabilities.

Explanation: Foreground services show a persistent notification to the user, making them aware that the service is running. This is often used for tasks like music playback or GPS tracking.

Task Execution in the Background:

Use Case: If you have a task that needs to be executed in the background, such as processing data, you can use an unbound service.

Explanation: The service can run independently, allowing the main application components to continue executing without being blocked by the background task.

Bound Services:

Inter-Component Communication:

Use Case: When different components of an application need to communicate with a common service and share data or functionalities.

Explanation: Activities, fragments, or other components can bind to the service using bindService(). This allows them to interact with the service, call its methods, and exchange data.

Remote Service Access:

Use Case: When an application wants to interact with a service in a different process or even on a different device.

Explanation: Bound services can be implemented with the Android Interface Definition Language (AIDL) to provide a remote interface, allowing communication between components in different processes.

Resource Sharing:

Use Case: When multiple components need access to a common resource managed by the service, such as a database connection or a sensor.

Explanation: Bound services can maintain stateful connections, and multiple components can bind to the service to share the resource.

Dynamic Functionality:

Use Case: When you want to provide dynamic functionality that can be accessed by different parts of your application.

Explanation: Components can dynamically bind and unbind to the service as needed, allowing for a flexible and modular architecture.

In summary, unbound services are suitable for background tasks that don't require direct interaction with components, while bound services are useful for scenarios where multiple components need to communicate with a common service, share resources, or access dynamic functionality. The choice between them depends on the specific requirements of your application.

**Example:-**

This project(Music Player app) will demonstrate the use of services in android. It has two button. Start and stop button. When you click on start button, automatically the default ringtone for mobile will play and if you click on button stop, it will stop the default playing ringtone.

You can see the UI for Apps

1. Create a java class name as MyCustomService in package(create this class where your activity file is available)

package com.mastercoding.musicplayerapp;

import android.app.Service;

import android.content.Intent;

import android.media.MediaPlayer;

import android.os.IBinder;

import android.provider.Settings;

import androidx.annotation.Nullable;

//this is lab no 21

public class MyCustomService extends Service {

// To Play music, we need a media player object

private MediaPlayer player;

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

// this will play the audio of default ringtone in the device

player = MediaPlayer.create(

this,

Settings.System.DEFAULT\_RINGTONE\_URI

);

// play the ringtone audio on loop (continuously)

player.setLooping(true);

player.start();

return START\_STICKY;

}

@Override

public void onDestroy() {

super.onDestroy();

player.stop();

@Nullable

@Override

public IBinder onBind(Intent intent) {

return null;

}}

}

1. Write bellow code in MainActivity.java

package com.mastercoding.musicplayerapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

TextView txt;

Button start\_btn, stop\_btn;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

start\_btn = findViewById(R.id.start\_btn);

stop\_btn = findViewById(R.id.stop\_btn);

txt = findViewById(R.id.textView);

start\_btn.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent serviceIntent = new Intent(getApplicationContext(),

MyCustomService.class);

startService(serviceIntent);

}

});

stop\_btn.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent serviceIntent = new Intent(getApplicationContext(),

MyCustomService.class);

stopService(serviceIntent); }

}); }

}

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

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:background="@drawable/back"

tools:context=".MainActivity">

<TextView

android:id="@+id/textView"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_marginTop="28dp"

android:gravity="center"

android:text="The Services App"

android:textColor="@color/white"

android:textSize="32sp"

android:textStyle="bold"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintHorizontal\_bias="0.0"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

<Button

android:id="@+id/start\_btn"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginBottom="114dp"

android:text="Start Service"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent" />

<Button

android:id="@+id/stop\_btn"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginBottom="36dp"

android:text="Stop Service"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintEnd\_toEndOf="@+id/start\_btn" />

</androidx.constraintlayout.widget.ConstraintLayout>

OutPut Screen



1. **List any 5 terminology in android and explain it.**

**Ans:-**

* **APK**
* **Layout XML**
* **Intent**
* **Manifest file**
* **fragment**

APK (Android Package Kit):

Explanation: An APK is the package file format used by the Android operating system to distribute and install mobile applications. It contains the compiled code, resources, and other necessary files for an Android app. When users download and install apps from the Google Play Store or other sources, they are essentially installing the APK file.

Layout XML:

Explanation: In Android development, user interfaces are often defined using XML files known as layout files. These files specify the arrangement and appearance of UI elements such as buttons, text fields, and images. The XML layout files are then inflated at runtime to create the visual structure of the app.

Intent:

Explanation: An Intent is a messaging object used for communication between different components of an Android app or between different apps. It represents an intention to perform an action, such as launching an activity, starting a service, or broadcasting a message. Intents are a key mechanism for enabling collaboration between different parts of an Android application.

Manifest File:

Explanation: The AndroidManifest.xml file is a crucial configuration file in every Android app. It provides essential information about the app to the Android system, such as the app's components (activities, services, broadcast receivers), permissions it requires, hardware and software features it supports, and more. The manifest file is a central part of the app's structure and is used by the system to understand and manage the app.

Fragment:

Explanation: A Fragment is a modular and reusable component in Android development that represents a portion of a user interface or behavior within an activity. Fragments are commonly used to create responsive and flexible UIs, especially for larger screen devices like tablets. They can be combined within activities to create a multi-pane UI or reused across multiple activities.

These terms are fundamental to Android development, and understanding them is crucial for anyone involved in creating Android applications.

1. **How we can managed the Application resources in android. Give suitable example.**

**Ans:-**

Drawable and shape

The res folder short for resources is a crucial directory within your Android project structure. It contains various types of resources that your app uses to provide different aspects of its user interface, such as layouts, strings, images, colours, styles and more. These resources are organized into subdirectories within the res folder, each serving a specific purpose.

Eg. Insert the image and add to the background of activity.

Step 1 -copy any small image.jpg, give name in small letter.

Step 2- add this image into drwable folder

Step 3- write bellow attribute inside the layout tag of xml layout file.

android:background=”@drwable/backgroud”

Note - background is image name.

Custom draw able shape

Step 1-right click on drawable folder-new-select draw able resource file-

Step 2-write file name-”rounded\_button”

Step 3- automatically rounded\_button.xml file will be created

Step 4-write a bellow code in rounded\_button.xml file

Step 5- go to MainActivity.xml file and add button widgit. And add bellow code for getting the shape

android:background=”@drawable/rounded\_button”

app:backgroundTint=”@null” //if you get an error then add this line.

Step 6-run your application

**MipMAp folder**

To change the icon of your app- right click on mipmap-new-image asset-

You can also change the logo image of your icon. Download the logo- go to path- give the image path, it will change the logo of your application.

Mipmap is used to defined the icon for your application.

Value folder

values folder in Android Studio Values folder contains XML files that define various types of resources that your app uses to provide different aspects of its user interface and behavior. These resources are organized into the values folder to centralize configuration settings, localization and styling information.

Open color.xml file and write bellow code inside the <Resources>tag

<Resources>

<color name=”blue\_sky>#06BFCF</color>

</Resources>

Now go tho layout.xml file(MainActivity.xml)

Create a new TextVeiw and add bellow code

android:textColor=”@color/sky\_blue”

**String folder**

1)Go to String.xml folder- and write a bellow code inside <resources> tag

<string name=”text1”>UserName</string>

2) go to mainActivity.xml file and bellow code

<TextView

android:text=”@string/text1”

/>

1. **Write about SQLite API with suitable example.**

**Ans:- SQLite** is an **open-source relational database** i.e. used to perform database operations on android devices such as storing, manipulating or retrieving persistent data from the database.

It is embedded in android by default. So, there is no need to perform any database setup or administration task.

Here, we are going to see the example of SQLite to store and fetch the data. Data is displayed in the logcat.

**SQLiteOpenHelper** class provides the functionality to use the SQLite database.

Android has features available to handle changing database schemas, which mostly depend on using the SQLiteOpenHelper class. SQLiteOpenHelper is designed to get rid of two very common problems.

1. When the application runs the first time - At this point, we do not yet have a database. So we will have to create the tables, indexes, starter data, and so on.
2. When the application is upgraded to a newer schema - Our database will still be on the old schema from the older edition of the app. We will have option to alter the database schema to match the needs of the rest of the app.

The android.database.sqlite.SQLiteOpenHelper class is used for database creation and version management. For performing any database operation, you have to provide the implementation of **onCreate()** and **onUpgrade()** methods of SQLiteOpenHelper class.

Constructors of SQLiteOpenHelper class

There are two constructors of SQLiteOpenHelper class.

1) SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)

-creates an object for creating, opening and managing the database.

2) SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version, DatabaseErrorHandler errorHandler)

-creates an object for creating, opening and managing the database. It specifies the error handler.

#### Methods of SQLiteOpenHelper class

There are many methods in SQLiteOpenHelper class. Some of them are as follows:

1. public abstract void onCreate(SQLiteDatabase db)-called only once when database is created for the first time. It’s called when there is no database and the app needs one. It passes us a SQLiteDatabase object, pointing to a newly-created database, that we can populate with tables and initial data.
2. public abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)- called when database needs to be upgraded. It’s called when the schema version we need does not match the schema version of the database, It passes us a SQLiteDatabase object and the old and new version numbers. Hence we can figure out the best way to convert the database from the old schema to the new one.
3. public synchronized void close () - closes the database object.
4. public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) called when database needs to be downgraded.

#### Methods of SQLiteDatabase class

There are many methods in SQLiteDatabase class. Some of them are as follows:

1. void execSQL(String sql)-> executes the sql query not select query.
2. long insert(String table, String nullColumnHack, ContentValues values)->inserts a record on the database. The table specifies the table name, nullColumnHack doesn't allow completely null values. If second argument is null, android will store null values if values are empty. The third argument specifies the values to be stored.
3. int update(String table, ContentValues values, String whereClause, String[] whereArgs)-> it is used to updates a row.
4. Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)

It will returns a cursor over the resultset.

DatabaseHelper.java

package com.example.welcome;

import android.content.ContentValues;

import android.content.Context;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

import androidx.annotation.Nullable;

public class DatabaseHelper extends SQLiteOpenHelper {

String Database\_Name="paruldatabase";

int Database\_version=1;

//Table and column name

String Table\_name="student";

String id="\_id";

String column\_name="name";

String column\_age ="age";

public DatabaseHelper(@Nullable Context context, @Nullable String name, @Nullable SQLiteDatabase.CursorFactory factory, int version) {

super(context, name, factory, version);

}

@Override

public void onCreate(SQLiteDatabase db) {

// Create your table

String createTableQuery = "CREATE TABLE " + Table\_name + "(" +

id + " INTEGER PRIMARY KEY AUTOINCREMENT, " + column\_name + " TEXT, " + column\_age + " INTEGER)";

db.execSQL(createTableQuery);

}

@Override

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

//we will write when we require some logic

}}

MainActivity.java

package com.example.welcome;

import androidx.appcompat.app.AppCompatActivity;

import android.database.Cursor;

import android.os.Bundle;

import android.util.Log;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// Create an instance of your DBHelper

DatabaseHelper dbhelper = new DatabaseHelper(this, "paruldatabase", null, 1);

}

}

1. **How we do Share the data between the applications with content provider.**

**Ans-** Content providers are one of the primary building blocks of Android applications, providing content to applications. They encapsulate data and provide it to applications through the single ContentResolver interface. A content provider is only required if you need to share data between multiple applications. For example, the contacts data is used by multiple applications and must be stored in a content provider. If you don't need to share data amongst multiple applications you can use a database directly via android.database.sqlite.SQLiteDatabase.

When a request is made via a ContentResolver the system inspects the authority of the given URI and passes the request to the content provider registered with the authority. The content provider can interpret the rest of the URI however it wants.

The UriMatcher class is helpful for parsing URIs.

The primary methods that need to be implemented are:

1. onCreate()- which is called to initialize the provider query which returns data to the caller
2. insert()- which inserts new data into the content provider
3. update() -which updates existing data in the content provider
4. delete() -which deletes data from the content provider
5. getType()- which returns the MIME type of data in the content provider

Keep in mind that using a Content Provider might be more complex than other data-sharing mechanisms like Intent or Broadcasts. Consider using it when you need a structured and secure way to share data across applications.

In Android, a Content Provider is a component that manages access to a central repository of data. It allows different applications to share and access data in a secure and controlled manner. If you want to share data between applications using a Content Provider, you need to follow these steps:

Step 1: Define the Content Provider

Create a user defined Content Provider class that extends ContentProvider. You need to implement various methods to handle data access and manipulation. Below is a basic example:

import android.content.ContentProvider;

import android.content.ContentValues;

import android.database.Cursor;

import android.net.Uri;

public class MyContentProvider extends ContentProvider {

// Define your database or data storage mechanism here

@Override

public boolean onCreate() {

// Initialize your data storage here

return true;

}

@Override

public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {

// Implement query functionality

return null;

}

@Override

public Uri insert(Uri uri, ContentValues values) {

// Implement insert functionality

return null;

}

@Override

public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {

// Implement update functionality

return 0;

}

@Override

public int delete(Uri uri, String selection, String[] selectionArgs) {

// Implement delete functionality

return 0;

}

@Override

public String getType(Uri uri) {

// Return the MIME type for the content URI

return null;

}

}

Step 2: Declare the Content Provider in the Manifest

Declare your Content Provider in the AndroidManifest.xml file:

<provider

android:name=".MyContentProvider"

android:authorities="com.example.mycontentprovider"

android:exported="true"

android:grantUriPermissions="true" />

The android:authorities attribute is a unique identifier for your Content Provider. Make sure it's unique across all applications.

Step 3: Share Data

To share data, you need to define a content URI for your data and use it in both the provider and the consumer applications. For example:

// Content URI for your data

Uri contentUri = Uri.parse("content://com.example.mycontentprovider/data");

// Insert data using ContentResolver in one application

ContentValues values = new ContentValues();

values.put("column\_name", "some\_data");

getContentResolver().insert(contentUri, values);

// Query data using ContentResolver in another application

Cursor cursor = getContentResolver().query(contentUri, null, null, null, null);

Note- Remember to replace "com.example.mycontentprovider" with the authority you defined in your Content Provider.

Step 4: Permissions

To ensure that only authorized applications can access your Content Provider, you can use permissions. Define a custom permission in your Content Provider:

<provider

android:name=".MyContentProvider"

android:authorities="com.example.mycontentprovider"

android:exported="true"

android:grantUriPermissions="true"

android:permission="com.example.mycontentprovider.PERMISSION" />

Then, applications that want to access the provider should request this permission in their manifest:

<uses-permission android:name="com.example.mycontentprovider.PERMISSION" />

This ensures that only applications with the correct permission can access your Content Provider.

1. **What is the use of Telephony API? List any three classes and its methods and explain it.**

**Ans-** With Telephony API in Android, you can access information about the device's telephony status, such as network and SIM details.

Classes in Telephony API

**TelephonyManager**

Provides access to information about the telephony services on the device. Applications can use the methods in this class to determine telephony services and states, as well as to access some types of subscriber information. Applications can also register a listener to receive notification of telephony state changes.

The returned TelephonyManager will use the default subscription for all calls. To call an API for a specific subscription, use createForSubscriptionId(int) . e.g. telephonyManager = defaultSubTelephonyManager.createForSubscriptionId(subId);

Note that access to some telephony information is permission-protected. Your application cannot access the protected information unless it has the appropriate permissions declared in its manifest file. Where permissions apply, they are noted in the the methods through which you access the protected information.

TelephonyManager is intended for use on devices that implement FEATURE\_TELEPHONY . On devices that do not implement this feature, the behavior is not reliable. Requires the PackageManager#FEATURE\_TELEPHONY feature which can be detected using PackageManager.hasSystemFeature(String)

Here's a simple example of an Android program that uses the Telephony API to retrieve information about the device's telephony status:

import android.content.Context;

import android.os.Build;

import android.os.Bundle;

import android.telephony.TelephonyManager;

import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

public class TelephonyInfoActivity extends AppCompatActivity {

private TextView telephonyInfoTextView;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_telephony\_info);

telephonyInfoTextView = findViewById(R.id.telephonyInfoTextView);

// Check if the device has telephony support

if (isTelephonySupported()) {

// Get TelephonyManager instance

TelephonyManager telephonyManager = (TelephonyManager) getSystemService(Context.TELEPHONY\_SERVICE);

// Get device information using TelephonyManager

String networkOperator = telephonyManager.getNetworkOperator();

String networkOperatorName = telephonyManager.getNetworkOperatorName();

String simOperator = telephonyManager.getSimOperator();

String simOperatorName = telephonyManager.getSimOperatorName();

String phoneNumber = telephonyManager.getLine1Number();

// Display information in TextView

String telephonyInfo = "Network Operator: " + networkOperator + "\n" +

"Network Operator Name: " + networkOperatorName + "\n" +

"SIM Operator: " + simOperator + "\n" +

"SIM Operator Name: " + simOperatorName + "\n" +

"Phone Number: " + phoneNumber;

telephonyInfoTextView.setText(telephonyInfo);

} else {

telephonyInfoTextView.setText("Telephony not supported on this device.");}}

// Check if the device has telephony support

private boolean isTelephonySupported() {

TelephonyManager telephonyManager = (TelephonyManager) getSystemService(Context.TELEPHONY\_SERVICE);

return telephonyManager != null && telephonyManager.getPhoneType() != TelephonyManager.PHONE\_TYPE\_NONE;

}

}