MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 7

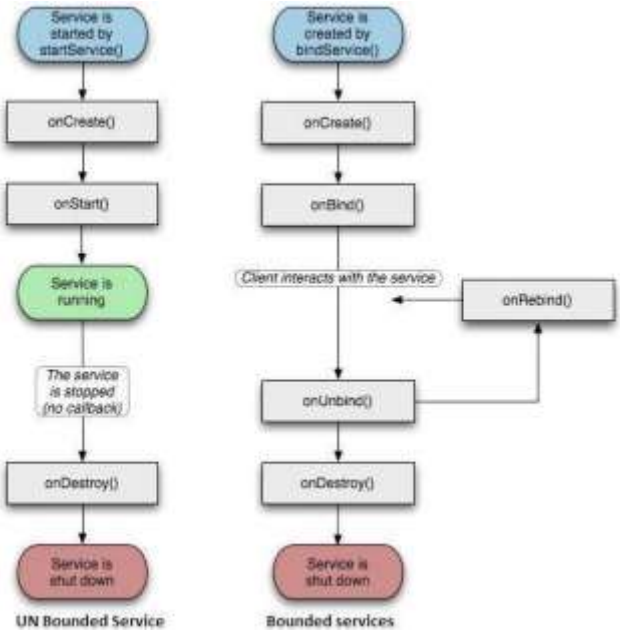
Objectives:

- To understand Services
- To understand the SQLite DB
- Practice Activities

OBJECTIVE 1: Understanding Services and its types

- A **service** is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states
- Services are used for repetitive and potentially long running operations, i.e., Internet downloads, checking for new data, data processing, updating content providers and the like.

Sr.No.	State & Description
1	**Started** A service is **started** when an application component, such as an activity, starts it by calling *startService()*. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
2	**Bound** A service is **bound** when an application component binds to it by calling *bindService()*. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).



Sr.No.	Callback & Description
1	**onStartCommand()** The system calls this method when another component, such as an activity, requests that the service be started, by calling *startService()*. If you implement this method, it is your responsibility to stop the service when its work is done, by calling *stopSelf()* or *stopService()* methods.
2	**onBind()** The system calls this method when another component wants to bind with the service by calling *bindService()*. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an *IBinder* object. You must always implement this method, but if you don't want to allow binding, then you should return *null*.
3	**onUnbind()** The system calls this method when all clients have disconnected from a particular interface published by the service.
4	**onRebind()** The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its *onUnbind(Intent)*.
5	**onCreate()** The system calls this method when the service is first created using *onStartCommand()* or *onBind()*. This call is required to perform one-time set-up.
6	**onDestroy()** The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

Service and Background Processes

Note: By default, a service runs in the same process as the main thread of the application.

Therefore, you need to use asynchronous processing in the service to perform resource intensive tasks in the background. A commonly used pattern for a service implementation is to create and run a new Thread in the service to perform the processing in the background and then to terminate the service once it has finished the processing.

MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 7

Intent Services

- You can also extend the IntentService class for your service implementation.
- The IntentService is used to perform a certain task in the background. Once done, the instance of IntentService terminates itself automatically. An example for its usage would be downloading certain resources from the internet.
- The IntentService class offers the onHandleIntent() method which will be asynchronously called by the Android system.

Objective 2: Understanding Database.

SQLite is a opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.

SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

- **Methods to get the instance of DB**

Sr.No	Method & Description
1	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)** This method only opens the existing database with the appropriate flag mode. The common flags mode could be OPEN_READWRITE OPEN_READONLY
2	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)** It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases
3	**openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)** It not only opens but create the database if it not exists. This method is equivalent to openDatabase method.
4	**openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)** This method is similar to above method but it takes the File object as a path rather then a string. It is equivalent to file.getPath()

- **Database – Insertion**

```
mydatabase.execSQL("CREATE TABLE IF NOT EXISTS TutorialsPoint(Username VARCHAR,Password
VARCHAR);");
mydatabase.execSQL("INSERT INTO TutorialsPoint VALUES('admin','admin');");
```

- **Database – Fetching**

```
Cursor resultSet = mydatbase.rawQuery("Select * from TutorialsPoint",null);
resultSet.moveToFirst();
String username = resultSet.getString(0);
String password = resultSet.getString(1);
```

- **Methods to call with Cursor Object.**

Sr.No	Method & Description
1	**getColumnCount()** This method return the total number of columns of the table.
2	**getColumnIndex(String columnName)** This method returns the index number of a column by specifying the name of the column
3	**getColumnName(int columnIndex)** This method returns the name of the column by specifying the index of the column
4	**getColumnNames()** This method returns the array of all the column names of the table.
5	**getCount()** This method returns the total number of rows in the cursor
6	**getPosition()** This method returns the current position of the cursor in the table
7	**isClosed()** This method returns true if the cursor is closed and return false otherwise

- **Database - Helper class**

For managing all the operations related to the database , an helper class has been given and is called SQLiteOpenHelper. It automatically manages the creation and update of the database. Its syntax is given below

```
public class DBHelper extends SQLiteOpenHelper {
   public DBHelper(){
      super(context,DATABASE_NAME,null,1);
   }
   public void onCreate(SQLiteDatabase db) {}
   public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}
}
```

MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 7

OBJECTIVE 3: ACTIVITIES.

Activity 1: Modify Activity 1 in Lab 6.

- Modify and store the user created story in the DB
- Add one more screen to show the user to select any user created story to view.
- On click on those stories, show user one of the old stories.

MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 7

Activity 2: Modify Activity 2 of Lab 6:

- Add the contacts into Database.

Steps	Description
1	You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to get references of all the XML components and populate the contacts on listView.
3	Create new src/DBHelper.java that will manage the database work
4	Create a new Activity as DisplayContact.java that will display the contact on the screen
5	Modify the res/layout/activity_main to add respective XML components
6	Modify the res/layout/activity_display_contact.xml to add respective XML components
7	Modify the res/values/string.xml to add necessary string components
8	Modify the res/menu/display_contact.xml to add necessary menu components
9	Create a new menu as res/menu/mainmenu.xml to add the insert contact option
10	Run the application and choose a running android device and install the application on it and verify the results.

-

MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 7

Activity 3: Create a song player.

- Create a song player which plays the song in the background (use Service)
- There should be two buttons, one to play the song and other to stop the song.