[Please subscribe our channel & Share with others to support us]

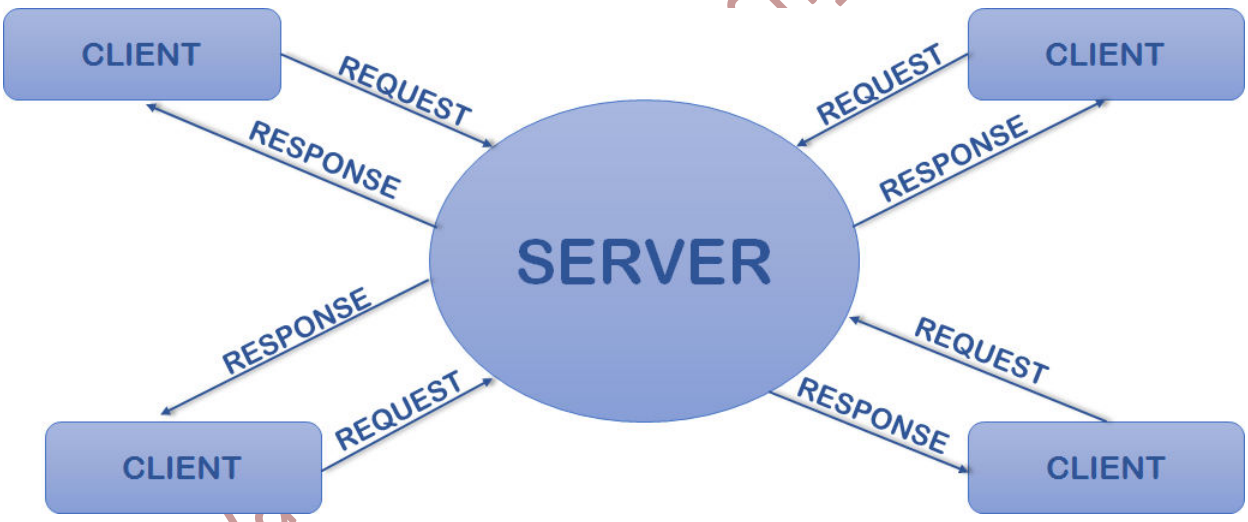
Youtube Channel Name: Techy Adesh

Adv Java Notes (B.Tech.CSE VI Sem)

Unit-1

Servlet: Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the web server, process the request & produce the response, then send response back to the web server.

To understand this better Lets have a look on client server architecture as following diagram.

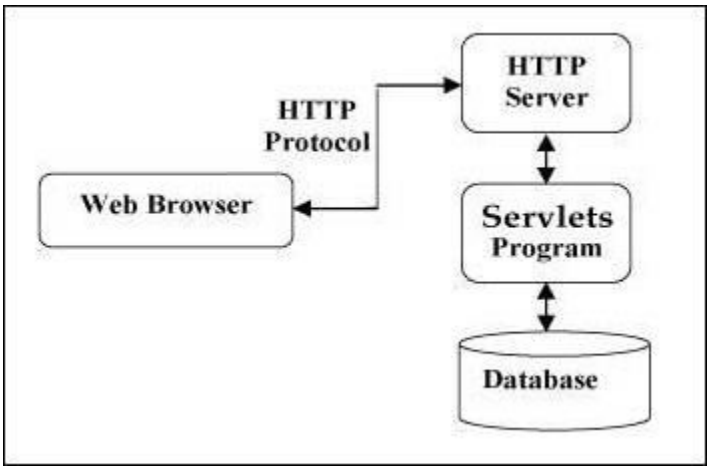


Properties of Servlet:

(a) Servlets work on the server-side.
(b) Servlets are capable of handling complex requests obtained from web server

(C) Servlets came to solve the problem came due to CGI (Common gateway interface) implementation.

Servlet Architecture



Differences Between Servlet & CGI

Servlet	CGI(Common Gateway Interface)
1)Servlets are portable and efficient.	1)CGI is not portable
In Servlets, sharing of data is possible.	In CGI, sharing of data is not possible.
Servlets can directly communicate with the web server.	CGI cannot directly communicate with the web server.
Servlets are less expensive than CGI.	CGI are more expensive than Servlets.
Servlets can handle the cookies.	CGI cannot handle the cookies.

Servlet API:

We need to install two packages for servlet implementation

- **1) javax.servlet**
- **2) javax.servlet.http**

In the following diagram we will see which component of servlet exists in which class of interface.

Component	***Type***	***Package***
Servlet	**Interface**	**javax.servlet.***
ServletRequest	**Interface**	**javax.servlet.***
ServletResponse	**Interface**	**javax.servlet.***
GenericServlet	**Class**	**javax.servlet.***
HttpServlet	**Class**	**javax.servlet.http.***
HttpServletRequest	**Interface**	**javax.servlet.http.***
HttpServletResponse	**Interface**	**javax.servlet.http.***
Filter	**Interface**	**javax.servlet.***
ServletConfig	**Interface**	**javax.servlet.***

Type of Servlet :

There are mainly two types of servlets -

1)Generic Servlet: Generic servlet is protocol independent servlet. It implements the Servlet and ServletConfig interface. It may be directly extended by the servlet. Writing a servlet in in GenericServlet is very easy. It has only init() and destroy() method of ServletConfig interface in its life cycle. It also implements the log method of ServletContext interface

.

2) HTTP Servlet: HttpServlet is HTTP (Hyper Text Transfer Protocol) specific servlet. It provides an abstract class HttpServlet for the developers for extend to create there own HTTP specific servlets. The sub class of HttpServlet must overwrite at least one method given below-

doGet()

doPost()

doPost()

doTrace()

doDelete()

init()

destroy()

getServiceInfo()

There is no need to overrride service() method. All the servlet either Generic Servlet or Http Servlet passes there config parameter to the Servlet interface

Servlet Life Cycle: The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet.

Steps in the life cycle of Servlet.

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.

Servlet with IDE: Servlet program may be run in different IDEs like Netbeans , Eclipse Etc we will see one by one

. **Servlet with Eclipse:** Eclipse IDE is open source , free and may be downloaded from this link.

http://www.eclipse.org/downloads/.

download the eclipse ide for JavaEE developers.

Steps to create the servlet File:

- Create a Dynamic web project
- create a servlet
- add servlet-api.jar file
- Run the servlet

Servlet with Netbeans: Netbeans IDE is open source , free and may be downloaded from this link.

http://www.eclipse.org/downloads/.

download the Netbeans ide for J2EE developers.

Steps to create the servlet File in Netbeans:

- Create a Web Application project
- create a servlet file
- add servlet-api.jar file
- Add Servlet configuration in web.xml (auto add is also there.)
- Run the servlet

ServletRequest Interface

An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header information , attributes etc.

ServletRequest Collabration

The exchange of information among servlets of a particular Java web application is known as **Servlet Collaboration**.

It can be done by following types.

1) RequestDispatcher in Servlet

The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration

There are two method using requestdispatcher.

1) **void forward(ServletRequest request,ServletResponse response):**

It forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server

2) **void include(ServletRequest request,ServletResponse response)**:

It also Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

2) SendRedirect:

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

```
response.sendRedirect("http://www.techyadesh.com");
```

ServletConfig:

- ServletConfig is an object containing some initial parameters or configuration information created by Servlet Container and passed to the servlet during initialization.
- ServletConfig is for a particular servlet, that means one should store servlet specific information in web.xml and retrieve them using this object.

ServletContext: ServletContext is the object created by Servlet Container to share initial parameters or configuration information to the whole application.

JSP:- **JSP** used to create web application but it is an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

Advantages of JSP over Servlet:

1)Easy to maintain

2)Less code

3) More functionality available

4) Fast Development

The Lifecycle of a JSP Page:

The JSP pages have the following phases.

- **Translation** – JSP pages doesn't look like normal java classes, actually JSP container parse the JSP pages and translate them to generate corresponding servlet source code. If JSP file name is home.jsp, usually its named as home_jsp.java.
- **Compilation** – If the translation is successful, then container compiles the generated servlet source file to generate class file.
- **Class Loading** – Once JSP is compiled as servlet class, its lifecycle is similar to servlet and it gets loaded into memory.
- **Instance Creation** – After JSP class is loaded into memory, its object is instantiated by the container.
- **Initialization** – The JSP class is then initialized and it transforms from a normal class to servlet. After initialization, ServletConfig and ServletContext objects become accessible to JSP class.
- **Request Processing** – For every client request, a new thread is spawned with ServletRequest and ServletResponse to process and generate the HTML response.
- **Destroy** – Last phase of JSP life cycle where it's unloaded into memory.

JSP lifecycle methods:

jspInit() declared in JspPage interface. This method is called only once in JSP lifecycle to initialize config params.

1. **_jspService(HttpServletRequest request, HttpServletResponse response)** declared in HttpJspPage interface and response for handling client requests.
2. **jspDestroy()** declared in JspPage interface to unload the JSP from memory.

JSP implicit object:

1. **out**: javax.servlet.jsp.JspWriter
2. **request**: javax.servlet.http.HttpServletRequest
3. **response**: javax.servlet.http.HttpServletResponse
4. **session**: javax.servlet.http.HttpSession
5. **application**: javax.servlet.ServletContext
6. **exception**: javax.servlet.jsp.JspException
7. **page**: java.lang.Object
8. **pageContext**: javax.servlet.jsp.PageContext
9. **config**: javax.servlet.ServletConfig

JSP Scripting Elements:

Scripting Element	Example
Comment	<%-- comment --%>
Directive	<%@ directive %>
Declaration	<%! declarations %>
Scriptlet	<% scriplets %>
Expression	<%= expression %>

JSP Expression Language:

Syntax of EL : $(expression)

Simple JSP page using EL can be written as following

```
<html>
<body>
<a>Expression is:</a>
${1+2};
</body>
</html>
```

JSTL:

(JSTL) is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications.

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating the existing custom tags with the JSTL tags.

- **Core Tags**
- **Formatting tags**
- **SQL tags**
- **XML tags**
- **JSTL Functions**
- `<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>`

JSTL:

(JSTL) is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications.

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating the existing custom tags with the JSTL tags.

JSP MVC:

MVC stands for Model View and Controller. It is a **design pattern** that separates the business logic, presentation logic and data.

Controller acts as an interface between View and Model. Controller intercepts all the incoming requests.

Model represents the state of the application i.e. data. It can also have business logic.

View represents the presentation i.e. UI(User Interface).

Unit-2

What is Android?

: Android is an open-source operating system based on Linux with a Java programming interface for mobile devices such as Smartphone (Touch Screen Devices who supports Android OS) as well for Tablets too.

Android was developed by the Open Handset Alliance (OHA), which is led by Google. The Open Handset Alliance (OHA) is a consortium of multiple companies like Samsung, Sony, Intel and many more to provide services and deploy handsets using the android platform.

Android History:

Initially, Google launched the first version of Android platform on Nov 5, 2007, from that onwards Google released a lot of android versions under a codename based on desserts, such as Apple Pie, Banana Bread, Cupcake, Donut, Éclair, Froyo, Gingerbread, Jellybeans, Kitkat, Lollipop, marshmallow, etc. and made a lot of changes and additions to the android platform.

Android Version:

Release Date	Version	API Level	Version Name
September 23, 2008	Android 1.0	1	Apple Pie
February 9, 2009	Android 1.1	2	Banana Bread
April 30, 2009	Android 1.5	3	Cupcake

Release Date	Version	API Level	Version Name
September 15, 2009	Android 1.6	4	Donut
October 26, 2009	Android 2.0	5	Eclair
December 3, 2009	Android 2.0.1	6	
January 12, 2009	Android 2.1	7	
May 20, 2010	Android 2.2	8	Froyo
January 18, 2011	Android 2.2.1	8	
January 22, 2011	Android 2.2.2	8	
November 21, 2011	Android 2.2.3	8	
December 6, 2010	Android 2.3	9	Gingerbread
February 9, 2011	Android 2.3.1	9	
July 25, 2011	Android 2.3.3	10	
September 2, 2011	Android 2.3.4	10	
February 22, 2011	Android 3.0.x	11	Honeycomb
May 10, 2011	Android 3.1.x	12	

Release Date	Version	API Level	Version Name
July 15, 2011	Android 3.2.x	13	
October 18,2011	Android 4.0	14	Ice Cream Sandwich
October 19, 2011	Android 4.0.1	14	
November 28, 2011	Android 4.0.2	14	
December 16, 2011	Android 4.0.3	15	
February 4, 2012	Android 4.0.4	15	
July 9, 2012	Android 4.1	16	Jelly Bean
July 23, 2012	Android 4.1.1	16	
October 9, 2012	Android 4.1.2	16	
November 13, 2012	Android 4.2	17	
November 27, 2012	Android 4.2.1	17	
February 11, 2013	Android 4.2.2	17	
July 24, 2013	Android 4.3	18	
October 31, 2013	Android 4.4	19	Kitkat

Release Date	Version	API Level	Version Name
June 23, 2014	Android 4.4.1, 4.4.2, 4.4.3, 4.4.4	19	
October 17, 2014	Android 5.0	21	Lollipop
March 09, 2015	Android 5.1	22	
October 5, 2015	Android 6.0	23	Marshmallow
December 7, 2015	Android 6.0.1	23	
August 22, 2016	Android 7.0	24	Nougat
October 4, 2016	Android 7.1	25	
	Android 8.0	26	O

Android Architecture: Android architecture is a software stack of components to support mobile device needs. Android software stack contains a Linux Kernel, collection of c/c++ libraries which are exposed through an application framework services, runtime, and application.

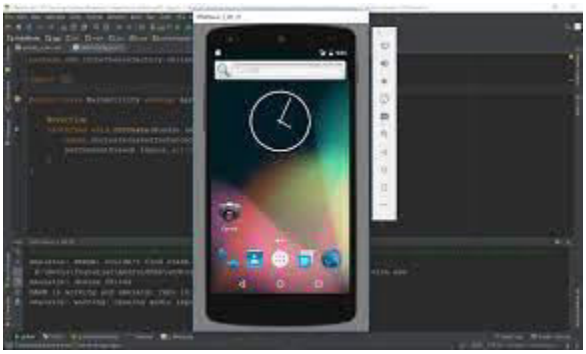
Following are main components of android architecture those are

1. Applications
2. Android Framework
3. Android Runtime
4. Platform Libraries
5. Linux Kernel

In these components, the **Linux Kernel** is the main component in android to provide its operating system functions to mobile and **Dalvik Virtual Machine** (**DVM**) which is responsible for running a mobile application.

Android Emulator :

The Android Emulator **simulates Android devices on your computer** so that you can test your application on a variety of devices and Android API levels without needing to have each physical device. The emulator provides almost all of the capabilities of a real Android device



Android Hello World Simple App :

1) **activity_main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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="com.tutlane.helloworld.MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

2) MainActivity.java

```
package com.techyadesh.helloworld;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Ouput:

Android Application Components (Activities, Intents, Views, Layouts, Services)

In android, **application components** are the basic building blocks of an application and these components will act as an entry point to allow system or user to access our app.

The following are the basic core application components that can be used in Android application.

1)Activities

2) Intents

3)Content Providers

4)Broadcast Receivers

5)Services

Android Widgets:

We have a different type of UI controls available in android to implement the user interface for our android applications.

Button, EditText, AutoCompleteTextView, ToggleButton, DatePicker, TimePicker, ProgressBar etc. are the commonly used UI or input controls in android applications.

Android Activity:

In android, **Activity** represents a single screen with a user interface (UI) of an application and it will acts an entry point for users to interact with an app.

Generally, the android apps will contain multiple screens and each screen of our application will be an extension of Activity class. By using activities, we can place all our android application UI components in a single screen.

From the multiple activities in android app, one activity can be marked as a **main activity** and that is the first screen to appear when we launch the application. In android app each activity can start another activity to perform different actions based on our requirements.

We have to enter the details about the activity in android manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest …..>
    <application …..>
        <activity android:name=".MainActivity" >
          …….
          …….
        </activity>
     …….
</application>
</manifest>
```

Android Intents:

In android, **Intent** is a messaging object which is used to request an action from another app component. Generally, in android, **Intents** will help us to maintain the communication between app components from the same application as well as with the components of other applications.

Android intents are mainly used to:

- Start the service
- Launch an activity
- Display a web page
- Display a list of contacts
- Broadcast a message
- Dial a phone call etc.

Types of Android Intents:

There are two types of intents in android: implicit and explicit.

1) Implicit Intent

Implicit Intent doesn't specifiy the component. In such case, intent provides information of available components provided by the system that is to be invoked.

For example, you may write the following code to view the webpage.

```
1.  Intent intent=new Intent(Intent.ACTION_VIEW);
2.  intent.setData(Uri.parse("http://www.techyadesh.com"));
3.  startActivity(intent);
```

2) Explicit Intent

Explicit Intent specifies the component. In such case, intent provides the external class to be invoked.

```
1.  Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
2.  startActivity(i);
```

Android Fragments

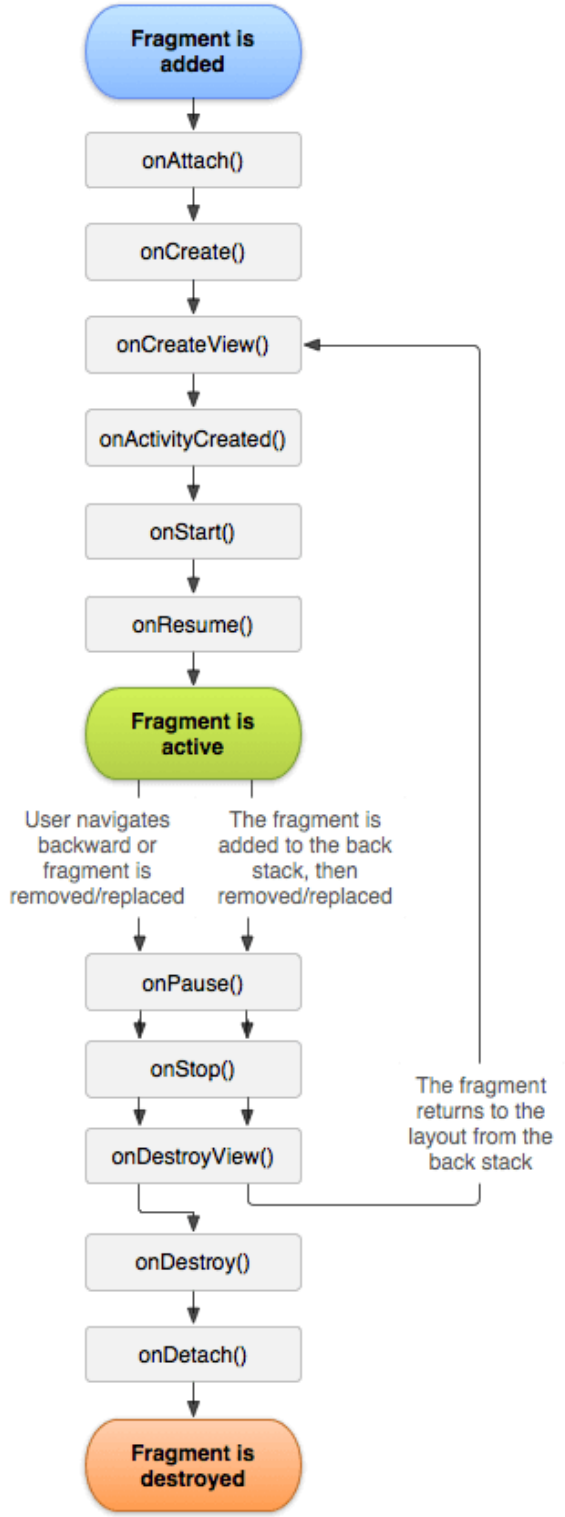
Android Fragment is the part of activity, it is also known as sub-activity. There can be more than one fragment in an activity. Fragments represent multiple screen inside one activity.

Android fragment lifecycle is affected by activity lifecycle because fragments are included in activity.

Each fragment has its own life cycle methods that is affected by activity life cycle because fragments are embedded in activity.

The **FragmentManager** class is responsible to make interaction between fragment objects.

Android Fragment Life Cycle:



Android Menus:

In android, **Menu** is a part of the user interface (UI) component which is used to handle some common functionality around the application. By using Menus in our applications, we can provide better and consistent user experience throughout the application.

We can use Menu APIs to represent user actions and other options in our android application activities.

There are following type of android menus

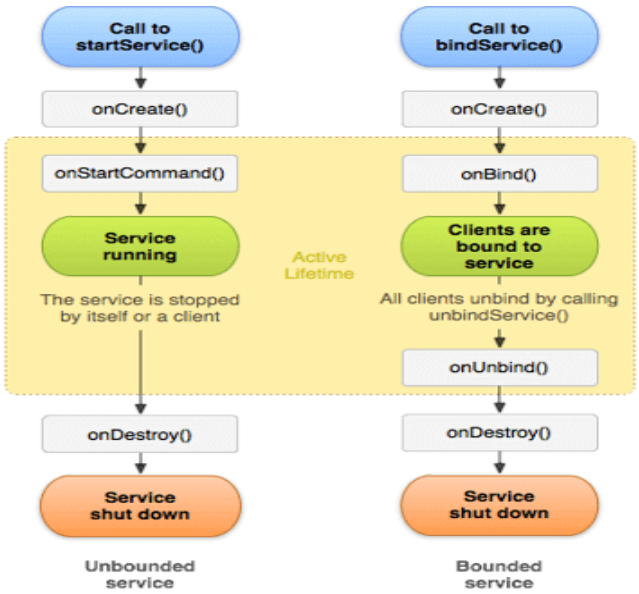
1)Option Menu

2) Context Menu

3) Popup Menu

Android Service:

In android, **Service** is a component which keep an app running in the background to perform long-running operations based on our requirements. For **Service**, we don't have any user interface and it will run the apps in the background like playing the music in the background or handle network operations when the user in a different app.



What is SQLite:

SQLite is a software library that provides a relational database management system. The lite in SQLite means lightweight in terms of setup, database administration, and required resources.

SQLite has the following noticeable features: self-contained, serverless, zero-configuration, transactional.

SQLite does NOT require a server to run.

Create Database and Tables using SQLite Helper

In android, by using **SQLiteOpenHelper** class we can easily create the required database and tables for our application. To use **SQLiteOpenHelper**, we need to create a subclass that overrides the **onCreate()** and **onUpgrade()** call-back methods.

Following is the code snippet of creating the database and tables using the **SQLiteOpenHelper** class in our android application.

```
public class DbHandler extends SQLiteOpenHelper {
    private static final int DB_VERSION = 1;
    private static final String DB_NAME = "usersdb";
    private static final String TABLE_Users = "userdetails";
    private static final String KEY_ID = "id";
    private static final String KEY_NAME = "name";
    private static final String KEY_LOC = "location";
    private static final String KEY_DESG = "designation";
    public DbHandler(Context context){
        super(context,DB_NAME, null, DB_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db){
        String CREATE_TABLE = "CREATE TABLE " + TABLE_Users + "("
                + KEY_ID + " INTEGER PRIMARY KEY AUTOINCREMENT," + KEY_NAM
E + " TEXT,"
                + KEY_LOC + " TEXT,"
                + KEY_DESG + " TEXT"+ ")";
        db.execSQL(CREATE_TABLE);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersio
n){
        // Drop older table if exist
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_Users);
```

```
        // Create tables again
        onCreate(db);
    }
}
```

If you observe above code snippet, we are creating database "**usersdb**" and table "**userdetails**" using **SQLiteOpenHelper** class by overriding **onCreate** and **onUpgrade** methods.

Android JSON Parsing:

Generally, **JSON** (JavaScript Object Notation) is one of the simplest data exchange formats to interchange the data from the server and it's the best alternative for XML. In simple words, we can say that JSON is a lightweight and structured language.

Android provides support for all JSON classes such as **JSONObject**, **JSONArray**, **JSONStringer**, etc. to parse the JSON data to get the required information in android applications.

The main advantage of JSON is, it's a **language independent** and the JSON object will contain data like key/value pair.

Following is the sample structure of JSON data in android applications.

```
{
"users": [
{
"name": "Suresh Dasari",
"designation": "Team Leader",
"location": "Hyderabad"
},
{
"name": "Rohini Alavala",
"designation": "Agricultural Officer",
"location": "Guntur"
}
]
}
```

JSON Parsing in Android

To parse the JSON data in android, we need to create an instance of **JSONObject** and **JSONArray** objects with a string that contains JSON data in it.

Following is the code snippet of parsing the JSON data in android using JSONObject and JSONArray objects to get the required information from JSON objects.

```
JSONObject jObj = new JSONObject(jsonStr);
JSONArray jsonArry = jObj.getJSONArray("users");
for(int i=0;i<jsonArry.length();i++){
    HashMap<String,String> user = new HashMap<>();
    JSONObject obj = jsonArry.getJSONObject(i);
    user.put("name",obj.getString("name"));
    user.put("designation",obj.getString("designation"));
    user.put("location",obj.getString("location"));
    userList.add(user);
}
```

Android TextToSpeech:

n android, by using **TextToSpeech** class we can easily convert our text into voice and it supports different types of speaking languages. We can choose the speaking language based on our requirements in the android application.

Generally, the android **TextToSpeech** instance can only be used to synthesize text once it has completed its initialization so implement **TextToSpeech.OnInitListener** to notify the completion of initialization.

During the initialization, we can set the audio pitch rate, audio speed, type of language to speak, etc. based on our requirements.

Following is the code snippet of converting text to voice using TextToSpeech class in android applications.

```
public class MainActivity extends AppCompatActivity implements TextToSpeec
h.OnInitListener {
    ....
   TextToSpeech textToSpeech;
    @Override
    public void onInit(int status) {
        if (status == TextToSpeech.SUCCESS) {
            int result = textToSpeech.setLanguage(Locale.US);
String text = speakText.getText().toString();
            textToSpeech.speak(text, TextToSpeech.QUEUE_FLUSH, null, null)
;
        }
```

```
    }
    ....
}
```

If you observe above code, we used **TextToSpeech.OnInitListener** to notify the completion of initialization and used **TextToSpeech** class to convert entered text to voice.

Android Multimedia:

If Using Android we can create the multimedia devices such as media player(audio) , Recording player etc.

MediaPlayer class

The **android.media.MediaPlayer** class is used to control the audio or video files

Program to create a Audio Player

```
1.  package com.example.audiomediaplayer1;
2.  import android.media.MediaPlayer;
3.  import android.net.Uri;
4.  import android.os.Bundle;
5.  import android.app.Activity;
6.  import android.view.Menu;
7.  import android.widget.MediaController;
8.  import android.widget.VideoView;
9.
10. public class MainActivity extends Activity {
11.
12.     @Override
13.     protected void onCreate(Bundle savedInstanceState) {
14.         super.onCreate(savedInstanceState);
15.         setContentView(R.layout.activity_main);
16.
17.         MediaPlayer mp=new MediaPlayer();
18.         try{
```

```
19.                 mp.setDataSource("/sdcard/Music/maine.mp3");//Write your location here

20.                 mp.prepare();
21.                 mp.start();
22.
23.             }catch(Exception e){e.printStackTrace();}
24.
25.         }
26.
27.         @Override
28.         public boolean onCreateOptionsMenu(Menu menu) {
29.             // Inflate the menu; this adds items to the action bar if it is present.
30.             getMenuInflater().inflate(R.menu.activity_main, menu);
31.             return true;
32.         }
33.
34.     }
```


Android Telephony:

The **android.telephony.TelephonyManager** class provides information about the telephony services such as subscriber id, sim serial number, phone network type etc. Moreover, you can determine the phone state etc.

We are able to make a phone call in android via intent. You need to write only three lines of code to make a phone call.


```
1.  Intent callIntent = new Intent(Intent.ACTION_CALL);
2.  callIntent.setData(Uri.parse("tel:"+8802177690));//change the number
3.  startActivity(callIntent);
```

Java Design Pattern:

A design patterns are **well-proved solution** for solving the specific problem/task.

Now, a question will be arising in your mind what kind of specific problem? Let me explain by taking an example.

Problem Given:
Suppose you want to create a class for which only a single instance (or object) should be created and that single object can be used by all other classes.

Solution:
Singleton design pattern is the best solution of above specific problem. So, every design pattern has **some specification or set of rules** for solving the problems. What are those specifications, you will see

Advantage of design pattern:

1. They are reusable in multiple projects.
2. They provide the solutions that help to define the system architecture.
3. They capture the software engineering experiences.
4. They provide transparency to the design of an application.
5. They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.
6. Design patterns don?t guarantee an absolute solution to a problem. They provide clarity to the system architecture and the possibility of building a better system.

Categorization of design patterns:

Basically, design patterns are categorized into two parts:

1. Core Java (or JSE) Design Patterns.
2. JEE Design Patterns.

Core Java Design Patterns

In core java, there are mainly three types of design patterns, which are further divided into their sub-parts:

1.Creational Design Pattern

1. Factory Pattern
2. Abstract Factory Pattern
3. Singleton Pattern
4. Prototype Pattern
5. Builder Pattern.

2. Structural Design Pattern

1. Adapter Pattern
2. Bridge Pattern
3. Composite Pattern
4. Decorator Pattern
5. Facade Pattern
6. Flyweight Pattern
7. Proxy Pattern

3. Behavioral Design Pattern

1. Chain Of Responsibility Pattern
2. Command Pattern
3. Interpreter Pattern
4. Iterator Pattern
5. Mediator Pattern
6. Memento Pattern
7. Observer Pattern
8. State Pattern
9. Strategy Pattern

10. Template Pattern
11. Visitor Pattern

JEE or J2EE Design Patterns

J2EE design patterns are built for the developing the Enterprise Web-based Applications.

In J2EE , there are mainly three types of design patterns, which are further divided into their sub-parts:

1. Presentation Layer Design Pattern

1. Intercepting Filter Pattern
2. Front Controller Pattern
3. View Helper Pattern
4. Composite View Pattern

2. Business Layer Design Pattern

1. Business Delegate Pattern
2. Service Locator Pattern
3. Session Facade Pattern
4. Transfer Object Pattern

3. Integration Layer Design Pattern

1. Data Access Object Pattern
2. Web Service Broker Pattern

Presentation Layer:

Intercepting Filter Pattern

An Intercepting Filter Pattern says that "if you want to intercept and manipulate a request and response before and after the request is processed".

Usage:

- When you want centralization, common processing across requests, such as logging information about each request, compressing an outgoing response or checking the data encoding scheme of each request.
- When you want pre and post processing the components which are loosely coupled with core request-handling services to facilitate which are not suitable for addition and removal.

Benefits:

- It provides central control with loosely coupled handlers.
- It improves reusability.