★ Singleton Class :-

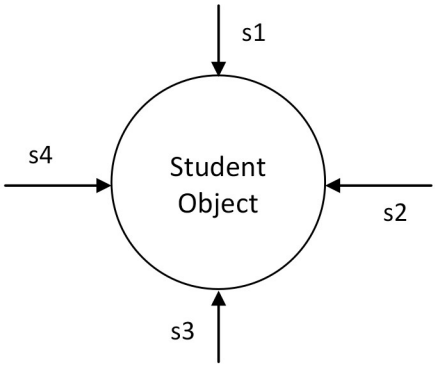
- It is one of the design patterns.
- A class which is having only one object is called as Singleton class.
- Whenever we are having a requirement where we have to avoid multiple object creation then we will go for singleton design pattern.
- Objects are costly resource it means that whenever we create an object it will consume more memory inside a RAM.
- **Program for singleton class :-**

```
package org.jsp.singleton;
public class Singleton {

 // making the Singleton class object as private
 // for avoiding access outside the class
 private static Singleton obj = null;

 // creating constructor as private
 // for avoiding object creation
 private Singleton() {
 }

 // creating getInstance() method which having Singleton return type
 // for creating instance of the class
 public static Singleton getInstance() {

       //checking if the object is already created or not
       if(obj == null) {
             // if the object is not created
             // creating object
             obj = new Singleton();
       }

       // if the object is created
       // returning same reference
       return obj;
 }

}
```

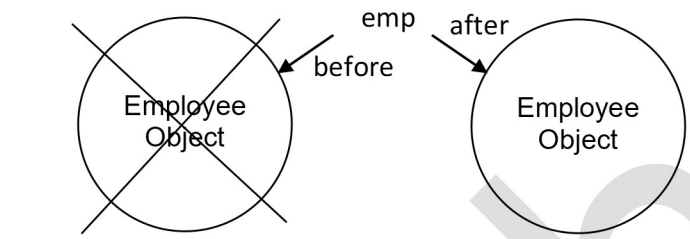
```
package org.jsp.singleton;

public class SingletonDriver {

 public static void main(String[] args) {
       Singleton obj1 = Singleton.getInstance();
       Singleton obj2 = Singleton.getInstance();
       System.out.println(obj1);
       System.out.println(obj2);   // same reference as obj1
 }

}
```

Note:-

- In java one object (class) can have more than one reference.



- One reference (instance) will always point to one object at a time.



Destroyed by Garbage Collector

- **Program :-**

```
package org.jsp.examples;

class Employee {
}

public class EmployeeDriver {

  public static void main(String[] args) {

        Employee emp = new Employee();
        System.out.println(emp);
        emp = new Employee();
        System.out.println(emp);

  }

}
```

- The object which does not have any reference is known as abandoned object (unwanted object).
- Abandoned object will be destroyed by the Garbage Collector (G.C.) implicitly.

Introduction to Advance Java

- By using core java knowledge we can develop only stand alone application.

★ Stand Alone Application :-

- The application which is running on a single machine or computer is known as standalone application.
- There are two types of standalone application,
  i. GUI Based Application
  ii. CUI Based Application

i. GUI Based Application :-

- The applications which are running on single machine provides visual interface for user interaction.
- User can interact with the help of input devices like, keyboard, mouse, etc.
- It is also known as desktop application.
- GUI stands for Graphical User Interface.
- E.g. Calculator, Word, Excel, etc.

ii. CUI Based Application :-

- The applications which are running on single machine through command prompt, console or terminal.
- It is also known as console based application.
- CUI stands for Character User Interface.
- E.g.

```
import java.util.Scanner;
class Driver{
    public static void main(String[] args){
            Scanner sc = new Scanner(System.in);
            int n = sc.nextInt();
            System.out.println(n);
        }
}
```

- The weightage of standalone application is very less in IT market.
- The weightage of web application in IT market is very high but we cannot develop web application with the help of core java knowledge.

★ Why we have to learn Advance Java?

- To develop web applications we have to learn advance java.

★ Web Application :-

- The application which is providing services over the internet is known as web application.
- E.g. swiggy, zomato, ola, uber, amazon, etc.

★ Editions of Java :-

i. Java Standard Edition (JSE / J2SE) :-

- Core Java, JDBC

ii. Java Enterprise Edition (JEE / J2EE) :-

- Hibernate, EJB (Enterprise Java Beans), Servlets & JSP, JPA, Spring

iii. Java Micro Edition (JME / J2ME) :-

- Mobile Application

JDBC

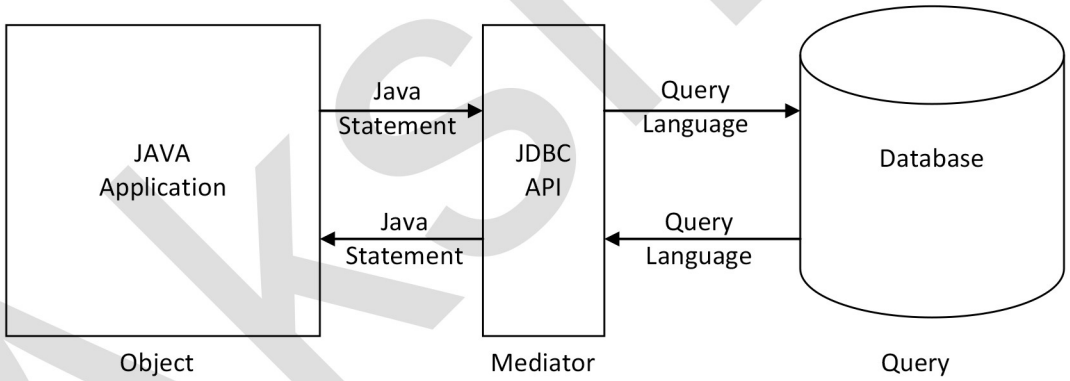
★ API :-

- API stands for Application Programming Interface.
- API will help to communicate between two or more application.
- Whenever java application needs to communicate with database then we will take the help of API i.e. JDBC API.

★ JDBC :-

- It is an API.
- It is acting as a mediator between java application and database.
- JDBC API will help to convert java statements to database understandable language (Query language) and vice versa.
- JDBC is introduced by Sun Microsystems in 1995.
- JDBC API's predefined classes and interfaces are present in java.sql package.
- E.g. Connection, ResultSet, Statement, PreparedStatement, etc.
- JDBC API will provide step by step approach in order to communicate with database.
- By using JDBC API we can perform CRUD operations on database.

★ How JDBC API will work?



- When java and database wants to communicate with each other, they cannot communicate directly because java knows only object and database knows only query.
- In order to overcome this situation we use JDBC API.
- First java statements will be given to JDBC API then JDBC API will convert java statements to query language.
- Again when database wants to send a response first it gives to JDBC API then JDBC API converts those queries to java statements.

Note:-

- JDBC API cannot perform tasks alone internally it will take help of driver software.

★ Driver Software :-

- It is vendor responsibility to provide the driver software.
- Based on the driver software JDBC API will understand on which database understandable language it has to convert java statements.
- Driver software will provide the implementation of abstract method which is present inside a JDBC interface.
- Driver software is database dependent.
- Because each and every database are having their own driver software.
- If we want to use MySQL database then compulsory we should add MySQL driver software only.
- Whenever we are changing a database we should also change the driver software respectively.

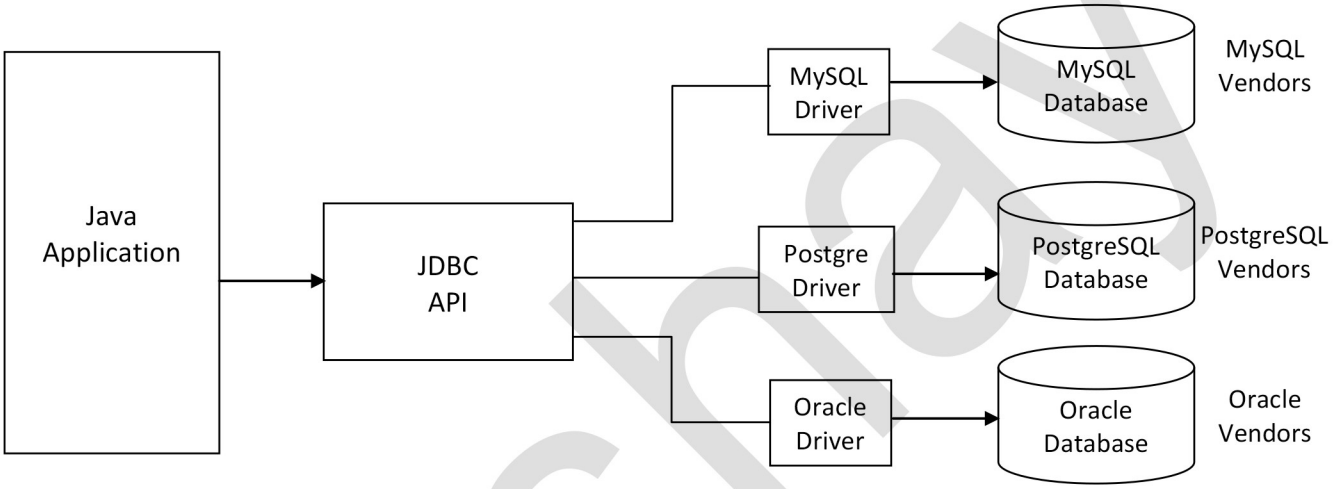


Figure: - JDBC Architecture

★ Difference between MySQL and PostgreSQL :-

MySQL	PostgreSQL
i. UI for MySQL is MySQL workbench.	i. UI for PostgreSQL is pgAdmin.
ii. It is less secured.	ii. It is more secured.
iii. When our frequent operation is read the data from database then we will go for MySQL database.	iii. When our frequent operation is read and write then we will go for PostgreSQL database.
iv. Generally we use MySQL database for ecommerce application.	iv. Generally we use PostgreSQL database for heavy application.
v. E.g. Flipkart, Amazon, etc.	v. E.g. Banking Application, Instagram, etc.

Note:-

- Compared to MySQL, PostgreSQL database is advanced version.

★ Maven Project :-

- In order to communicate with java application and database we are going to use maven project structure.

★ Why Maven?

- We can perform same task by using normal java project but programmer has to create some files, folders explicitly.
- Even though programmer is creating normal java project, if programmer will do small mistake also our project will not run as expected.
- Each and every time programmer has to build the structure so it's a time consuming process.
- Instead of doing modification in the same normal java project we will go for predefined project structure i.e. maven project.
- Maven project is open source which is provided by apache software foundation.

★ pom.xml :-

- It is an xml file.
- The full form of pom is project object model.
- It is having all the dependencies and project related information like group id, artifact id.
- group id :- org.jsp, artifact id :- project name (jdbccrud)
- Inside a pom.xml we have to write everything in the form of tag.

★ Maven Repository :-

- It is a common place where we will get all the dependencies.

★ How to update maven project?

- Select the maven project → Right click → Go to maven → Update maven → Check the force update check box → Click ok.

★ Dependencies :-

- It is a parent tag of all the dependencies.
- It is a paired tag.
- If we want to add any dependency in simple maven project then we will add in between the dependencies tag.

★ Steps to connect to database :-

i. Load / Register the Driver
ii. Create connection / Establish connection
iii. Create statement
iv. Execute query
v. Close the connection

i. Load / Register the Driver :-

- First you have to load driver class into a memory because our JDBC API will get to know that what type of database we are using.
- It also helps JDBC API to converts java statements to that respective database queries.

ii. Create connection / Establish connection :-

- If we want to perform any operation on database then we have to establish the connection between java application and database.
- While establishing the connection we have to provide username, password, url of that respective database.
- If we are providing any wrong credentials then the connection will not get established.
- If we want to perform any operation on any database compulsory we have to provide username, password, url because all databases are secured and we cannot perform task directly on database.

iii. Create statement :-

- It will act as a container where it will load data from the java application and it will unload into database.

iv. Execute query :-

- In this step we are going to execute all the queries which we have written inside java application.

v. Close the connection :-

- Once after performing an operation on database we are going to destroy / close the connection because connection is a costly resource.
- If you are not closing the connection then there is might be possibility of data leakage.

Note:-

- We are using RDBMS database i.e PostgreSQL
- We are going to store data in the form of table.

★ How to create database in PostgreSQL?

- Select PostgreSQL → Right click → Create → Database → General → Provide database name → jdbccrud (database name) → Click on save.

★ How to create table in PostgreSQL?

- Select database → Schemas → Tables → Right click → Create → Table → General → table_name (employee) → Columns → Click on (+) icon to add columns in table → Provide column name and datatypes → and lastly click on save.

★ How to check column is created or not?

- Select table (employee) → Right click → Select view / edit data → All Rows.

★ How to load the Driver class?

- To load the driver class we are going to take the help of a class named as Class itself.

★ Class:-

- It is a predefined final class.
- It is present in java.lang package.
- It is having so many predefined methods in that we are going to take the help of method i.e. forName() method.

★ forName(String driverClassName):-

- It is static method its return type is Class.
- We will give argument as fully qualified name of driver class.

Note:-

- Whenever we are using forName() method always it will throws CheckedException i.e. ClassNotFoundException.
- Fully qualified name is combination of class name and package name.

★ Program for insert data from java application to PostgreSQL database :-

```
package org.jsp.jdbccrud;
import java.sql.Statement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class EmployeeDriver {

	public static void main(String[] args) {

		try {
			//1 :- load or register the driver class
			Class.forName("org.postgresql.Driver");
			System.out.println("Driver is loaded");

			//2 :- Establish the connection
			String url = "jdbc:postgresql://localhost:5432/jdbccrud";
			String username = "postgres";
			String password = "root";
			Connection con = DriverManager.getConnection(url,username,password);
			System.out.println("Connection is created."+con);

			//3 :- create statement
			Statement st = con.createStatement();
			System.out.println("Statement:"+st);

			//4 :- execute query
			String sql = "insert into employee values (202,'Ram',24)";
			st.execute(sql);
			//5 :- close the connection
			con.close();
			System.out.println("Data inserted successfully.");

		} catch (ClassNotFoundException | SQLException e) {
			e.printStackTrace();
		}

	}

}
```

★ Connection :-

- Connection is an interface which is present in java.sql package.
- Whenever we want to create Connection object we will take help of getConnection() method because we don't know that who is child implementing class of Connection interface.
- But getConnection() method knows who is the child implement class of Connection interface.

★ getConnection() :-

- It is a static method present in DriverManager class.
- It is an overloaded method and its return type is Connection.
- In this method we are going to pass url, username and password.
- If all credentials are valid then it will create a Connection object else it will throw SQLException.
- String url = "jdbc:postgresql://localhost:5432/jdbccrud";
- String user = "postgres";
- String pass = "root";

★ DriverManager class :-

- It is utility class.
- It is present in java.sql package.
- It is introduced in JDK version 1.1

Note:-

- A class which is having only static methods is known as Utility class.
- E.g. DriverManager, Collections, etc.

url = "jdbc:postgresql://localhost:5432:/jdbccrud";

- **jdbc :-** In the above url jdbc is a protocol (protocol is set of rules).
- **postgresql :-** It is a database name.
- **localhost :-** It is a host name (i.e. where is our database present).
- **5432 :-** It is a port number of postgreSQL database.
- **jdbccrud :-** It is a database name in which our table is created.

★ Statement :-

- Statement is an interface.
- It is present in java.sql package.
- Whenever we need to create an object of Statement interface then we will take help of createStatement() method, because we don't know the child implementing class of Statement interface but createStatement() method knows who is child implementing class of Statement interface.
- Once we got Statement object we can call execute() method, executeUpdate() method and executeQuery() method.

★ createStatement() method :-

- It is an abstract method present inside Connection interface.
- Whenever we want to call this method first we have to create object of Connection interface then using address of that object we can call createStatement() method.
- The return type of createStatement() method is Statement.
- If Statement object is not created then it will throws SQLException.

★ execute(String sql) :-

- This method is use to execute a query.
- Its return type is boolean.
- If we want to call this method then we have to create an object of Statement interface.
- With the help of Statement object reference we can call the execute() method.
- Whenever we using this method it will always throw SQLException.

★ close() method :-

- close() method will helps to close the connection.
- It is an abstract method present inside Connection interface.
- Whenever we want to call this method then we have to create object of Connection interface.
- With the help of Connection object reference we can call close() method.
- This method will throw SQLException.
- The return type of this method is void.

★ How to register a Driver class?

- Whenever we want to register driver class then we will take the help of registerDriver() method.

★ registerDriver(Driver driver):-

- It is static method present in DriverManager class.
- Its return type is void.
- This method will take argument as Driver class.
- Whenever we are calling this method compulsory we have to pass the Driver class object.
- This method will always throws SQLException and it is recommended to handle it.

Program:-

```
package org.jsp.jdbccrud;

import java.sql.DriverManager;
import java.sql.SQLException;
import org.postgresql.Driver;

public class RegisterDriverMethod {

	public static void main(String[] args) {
		try
		{
			Driver driver = new Driver();
			DriverManager.registerDriver(driver);
			System.out.println("Driver register successfull.");
		}
		catch (SQLException e)
		{
			e.printStackTrace();
		}

	}
}
```

Note:-

- Among load and register driver class, it is recommended to go for load driver class.
- Because we can achieve load driver class without creating an object.
- When we are registering driver class we have to create an object of driver class and it will consume more memory and it is costly resource.

★ **Establish connection by using getConnection(String str) method which is taking only one arguments :-**

★ **getConnection(String str) :-**

- It is a static method present in DriverManager class.
- It will take only one argument i.e. url.
- String url = “jdbc:postgresql://localhost:5432/database_name?user=postgres& password=root”;
- In the above url “ ? ” is acting as a separator.
- If url is correct it will create Connection object else it will throws SQLException.

★ ResultSet :-

- Whenever we are executing select query from Java Application to the database a response database will give the data in the form of table.
- To hold the table data in java application we will take the help of ResultSet.
- It is the interface present in java.sql package.
- ResultSet will also act as cursor and initially it will point on the top of table.
- To move the cursor from top to the table next row ResultSet will provide next() method.

★ next() method:-

- It is a non-static method present in ResultSet interface.
- Its return type is boolean.
- next() method will helps to check do we have a next row or not.
- If we are having next row then it will return true and also it will move the cursor to the next row.
- If next row does not exist then it will return false.
- To fetch a data from each and every column ResultSet will provide “getX()” method.

★ getX() method:-

- Inside a getX() method we can pass either column number or column name.
- The number of columns and the number of getX() method should be same when you are calling getX() method.
- E.g. getInt(), getString(), getDouble(), etc.

★ How to create ResultSet object?

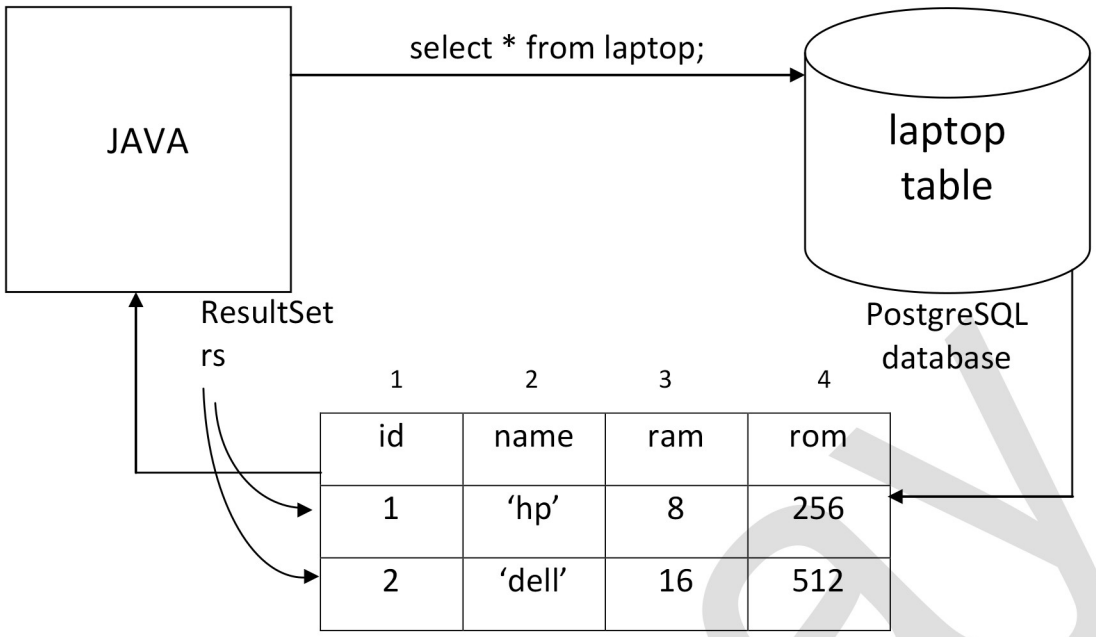
- There are two ways to create an object of ResultSet,
  i. getResultSet()
  ii. executeQuery()

i. getResultSet() method:-

- It is abstract method present in Statement interface.
- Its return type is ResultSet.
- Whenever we want to call this method first we have to create Statement interface object.

Note:-

- When we are calling getResultSet(), next() and getX() method it will throw SQLException.

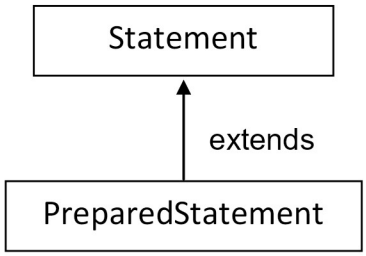



```
int id = rs.getInt(1);
String name = rs.getString(2);
int ram = rs.getInt(3);
int rom = rs.getInt(4);
```

- From JDBC 4.0 version onwards it is optional to load or register the Driver class.
- Inside a url we are mentioning the database name, based on that database name respective Driver class will be loaded into the memory.
- While establishing the Connection we used getConnection(url,user,pass) which is taking three arguments and getConnection(url) which is taking only one argument.
- Out of these two method the recommended way is getConnection(url,user,pass) which is taking three arguments because in this method we are going to provide url, username, password separately.
- Inside a url we are providing all the credentials like username, password.
- Once anyone got the url they will also get the username and password and they can access the database, they can steal the data from database [i.e. it is not secure].
- getConnection(url,user,pass) is recommended way to establish the Connection because it improves code readability and it is secured.

★ PreparedStatement :-

- It is a child interface of Statement interface.
- It is present in java.sql package.
- It will help us to take both static and dynamic data.



- If you want to create object of PreparedStatement, as a programmer we don't know the implemented child class of PreparedStatement interface but preparedStatement() method knows who is the child implementing class of PreparedStatement interface.
- preparedStatement() method will help us to create an object of PreparedStatment interface.

★ preparedStatement(String sql) :-

- It is an abstract method present in Connection interface.
- Its return type is PreparedStatement.
- We can call this method with the help of Connection object reference.
- It will throw SQLException.

★ Difference between Statement and PreparedStatement :-

Statement	PreparedStatement
i. It is a parent interface.	i. It is a child interface.
ii. It will take only static data.	ii. It will take both static and dynamic data.
iii. It will not support placeholder (i.e. delimiter).	iii. It will support placeholder (i.e. delimiter).
iv. It is less in performance.	iv. It is high in performance because of precompiled query.
v. It will not take data at runtime.	v. It will take data at runtime.
vi. createStatement() method is used to create an object of Statement() interface.	vi. preparedStatement() method is used to create an object of PreparedStatement interface.

★ execute() :-

- It is a non static method which is present in Statement interface.
- Its return type is boolean.
- We can use execute() method to both select and non select queries.
- If we are using to non select query then it will return false.
- If we are using to select query then it will return true.

★ executeUpdate() :-

- It is a non static method which is present in Statement interface.
- Its return type is int.
- We can use executeUpdate() method only for non select query if we try for select query then we will get Exception.
- It will return output based on the queries that how many rows got affected inside a database.

★ executeQuery() :-

- It is a non static method which is present inside a Statement interface.
- Its return type is ResultSet.
- We cannot use executeQuery() to non select query but we can use to select queries.
- This method will help us to create an object of ResultSet interface.

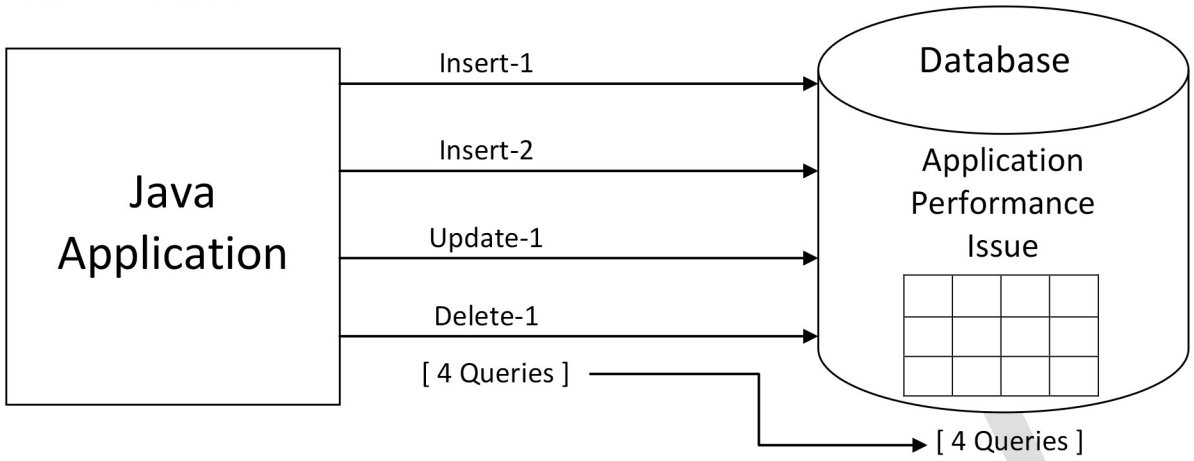
Note:-

- execute(), executeUpdate() and executeQuery() method will always throws a SQLException.

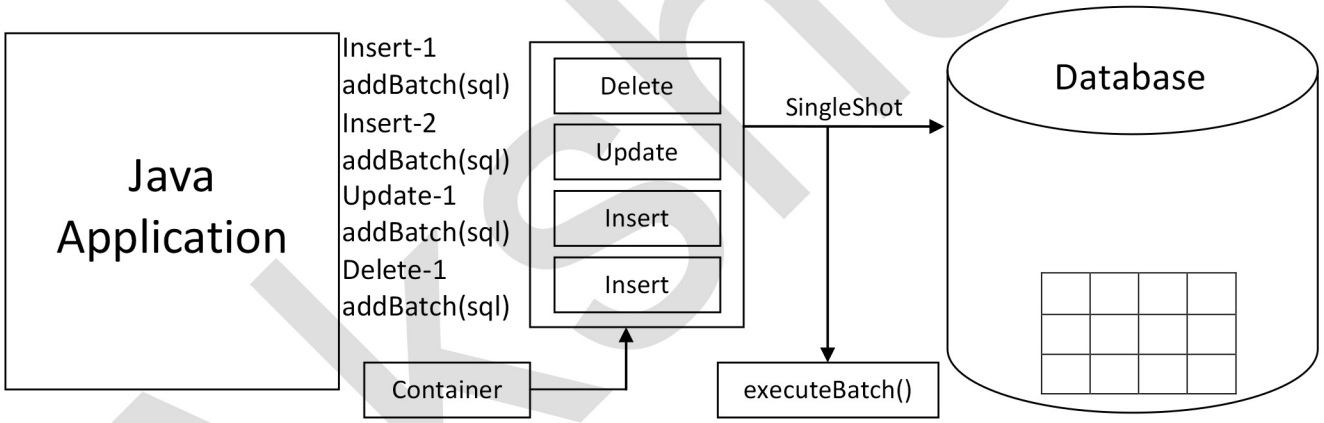
★ Difference between execute(), executeUpdate() and executeQuery() methods :-

execute()	executeUpdate()	executeQuery()
i. It is a non static method which is present inside a Statement interface.	i. It is a non static method which is present inside a Statement interface.	i. It is a non static method which is present inside a Statement interface.
ii. Its return type is boolean.	ii. Its return type is int.	ii. Its return type is ResultSet.
iii. We can execute both select and non select queries.	iii. We can only execute non select query.	iii. We can only execute select query.

★ Batch Execution :-



- In the above diagram, whenever we are writing any query each and every query is hitting the database.
- We are also increasing the number of queries which are hitting the database because of this we will get application performance issue.
- To overcome above disadvantage we go for " Batch Execution " concept.



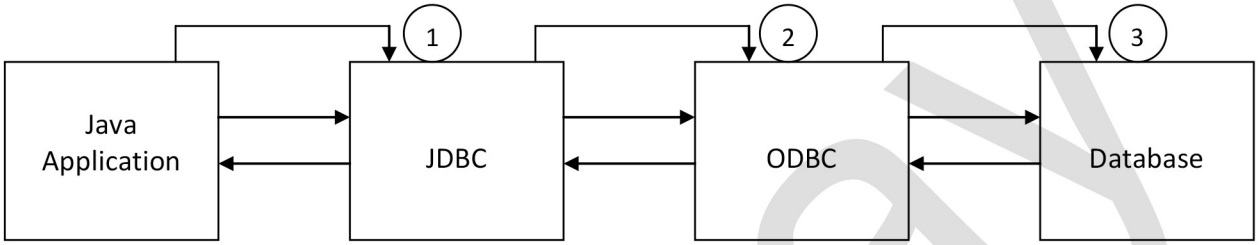
- Execution of multiple queries in a single shot is known as batch execution.
- The queries which we want to execute will be added in to the container.
- To add a query inside the container we use addBatch() method.
- To execute all the queries which are stored inside the container we use executeBatch() method.
- We are executing multiple queries in a single shot means only one query is hitting the database due to which the number of queries hitting the database reduces.
- By using batch execution concept we can improve application performance.
- **addBatch()** and **executeBatch()** methods are present in **Statement** interface.

★ Types of Driver Software's :-

i. Type-1 [OR] JDBC-ODBC Driver Software
ii. Type-2 [OR] Native-API
iii. Type-3 [OR] Network Protocol
iv. Type-4 [OR] Thin Driver

i. Type-1 :-

- It is also known as JDBC-ODBC Driver software.
- JDBC means Java Database Connectivity.
- ODBC means Open Database Connectivity.



- Its implementation is given by Microsoft using C language.
- It is database independent but platform dependent.
- It follows 3 steps conversion process.
- It is a heavy weight application.
- We can use it up to JDK 1.7 version.

ii. Type-2 :-

- It is also known as Native API.
- It is a heavy application.
- It is database dependent.
- It is a costly resource [You have to pay to use it].

iii. Type-3 :-

- It is also known as network protocol.
- It is neither database dependent nor platform dependent.
- It is used for web applications.
- It is a costly resource.

iv. Type-4 :-

- It is also known as thin driver.
- Its implementation is given by using java language.
- It is platform independent.
- It is free to use.
- It is lightweight application.
- We can develop both stand alone and web applications.

Note:-

- Among above four types we are using thin driver software.

★ Heavyweight Application or Software :-

- The application which takes more space inside system to run (consumes more memory) is known as heavyweight application.

★ Lightweight Software :-

- The application which takes less space inside system to run.

Note:-

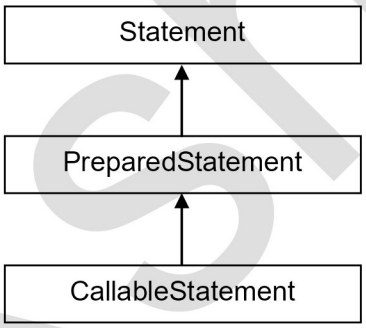
- We will always close the connection in finally block.

★ Stored Procedure :-

- It is a group of SQL statements, it is used to perform specific task.

★ Callable Statement :-

- If you want to call stored procedure inside a java application then we will go for CallableStatement.
- It is an interface present inside java.sql package.
- It is child interface of PreparedStatement interface.



- If you want to call stored procedure we will use a syntax :-
  call storedProcedureName(?,?,?,?,…);
- If you want to create an object of CallableStatement then we will take help of prepareCall(String sql) method.

★ prepareCall(String sql) :-

- It is an abstract method present inside Connection interface.
- Its return type is CallableStatement.

★ Disadvantage when you are closing Connection inside finally block :-

- In each and every program we have to close connection by using finally block.
- When we are writing same code again and again in each and every program then it will increase the number of lines of code and it will increase the duplicate code (boiler plate code).
- **To overcome this disadvantage we will go for try with resource statement.**

★ try with resource statement:-

- It is introduced in JDK 1.7 version or 7v.
- It helps to close the connection resources (object) which we created inside heap.
- Inside try with resource statement we can declare or we can create more than one resource.
- It will close the connection or resources automatically.
- It helps to remove boiler plate code.
- It makes code more readable.

★ Inside try with resource statement we can declare or create only an object which implements AutoClosable or Closable interface :-

```
try( Object obj = new Object() )
{

}
```

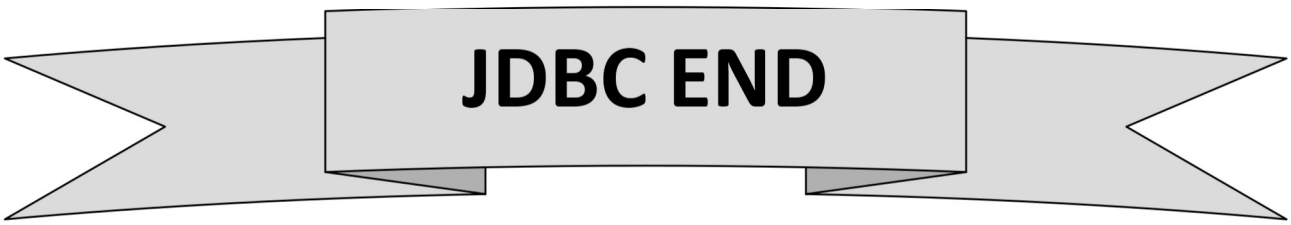
Output:-
Compile Time Error

★ AutoClosable :-

- It is a functional interface present in java.lang package.
- It is introduced in JDK 1.7 version.
- It is a predefined functional interface.
- It is having only one abstract method i.e. close().

Note:-

- Closable is a child interface of AutoClosable interface.
- The key benefit is that resources declared in the " try with resource block " are automatically closed at the end of the block regardless whether the execution is successful or an exception is thrown.
- This eliminates the need to manually close the Connection / Resources in the finally block.



Introduction to Servlets

★ Servlet :-

- It is used to develop web application.
- By using servlet we can overcome the limitations of CGI (Common Gateway Interface).

★ CGI :-

- CGI stands for Common Gateway Interface.
- It is an interface where we write script using programming languages such as C, C++, or Perl in order to communicate with server.

★ Limitations of CGI :-

i. It is having slow performance.
ii. Because each and every request will consume more memory inside server.
iii. When multiple users visit the website, there is a chance the server may crash, which means it is not scalable.
iv. It is difficult to maintain.

★ Server :-

- Server is software where it receives, processes and responds to the request coming from browser / client.

★ Why do we need Server?

i. It helps to run the application 24/7.
ii. It helps to maintain large amount of data.
iii. It allows multiple users at a time.

Note:-

- In the market there are so many servers like apache tomcat, JBoss, GlassFish, etc.
- Among all the above servers we are going to use “apache tomcat”.

★ Apache Tomcat Server :-

- It is provided by Apache Software Foundation.
- It is an open source tool.
- It works on many operating systems without any issue.

★ Client :-

- Client is software which sends a request and gets a response from server.
- Client is also known as browser.
- We have different types of clients like Chrome, Mozilla, Firefox, MS Edge.

★ Steps to download Apache Tomcat Server :-

- Click on the search button in the top right corner in the Eclipse IDE → Search for servers → Click on the option → A link will appear in the terminal → Right click on that link → New → Server → Expand apache folder → Tomcat version 9.0v → Select and click on next → Download and install → Accept terms and conditions → Finish → Select the folder where you want to download Apache tomcat → It will take while (there will appear a progress bar in bottom right corner) → Click finish.

★ To start the server follow these steps :-

- Select the server (in the console) → Right click on it → Start → go and check console → it will show server start up in milliseconds.

★ Steps to create web maven project :-

- File → New → Other → Search for maven project → Next → Don't check the "Simple Maven Project "→ Next → Search filter " org.apache.maven " → Uncheck the " Show the last version of archtype only " and select the one which has artifact id as " maven-archtype-webapp " and version should be 1.4 → Next give group id as org.jsp → artifact id as " ServletFirst " → Finish → On console " Y : " this should be present → Type Y and hit enter → Build success.

★ web.xml :-

- It is an xml file.
- It is also known as deployment descriptor file.
- In this file we are going to write configuration in the form of tags.

★ How to do servlet configuration inside "web.xml" file?

```
<servlet>
    <servlet-name>Anyname</servlet-name>
    <servlet-class>Fully qualified name of servlet class</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>Anyname</servlet-name>
    <url-pattern>/url</url-pattern>
</servlet-mapping>
```

Should be same name

Note:-

- Inside web.xml file we can do more than one servlet class configuration.
- You might be get an error as "Tomcat failed to start" means that you have done some mistake in configuration file (web.xml).

★ How to overcome package error?

- Right click on project → Click on properties → Click on java compiler option → Click on restore default → Click on apply and close → Click on yes.

★ How to create HTML file?

- Expand src folder → expand main folder → expand webapp folder → Select webapp folder → Right click on webapp folder → Click on new → Click on other → Search for "HTML" file → Select HTML file and Click on next → Provide name to the file (extension is optional) → Click on finish.

★ How to add src/main/java folder?

- Select the project → Right click on project → Click on properties → Select "java build path" → Click on "order and export" → Check "JRE system library" and "Maven dependencies" → Click on apply → Click on apply and close.

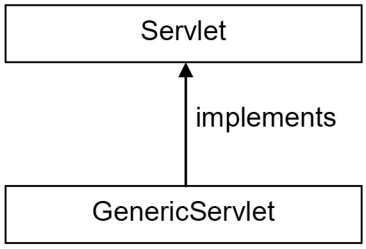
★ How to kill server which is already in use?

- Search for command prompt in windows search → Run as administrator → Type the command → netstat <space> -ano <space> |findstr <space> :8080 → Hit enter → You will get **LISTENING 5068** (you might get different PID) → Then type command → taskkill <space> /PID <space> 5068 <space> /F → hit enter → You will get the output as "The process with PID 5068 has been terminated".

★ Servlet :-

- It is server side technology it takes request from client and it processes the request it serve dynamic response back to the client.
- It is an interface which is present in javax.servlet package.
- It has total of 5 abstract methods :-
  i. public void init(ServletConfig config);
  ii. public ServletConfig getServletConfig();
  iii. public void service(ServletRequest req, ServletResponse resp);
  iv. public String getServletInfo();
  v. public void destroy();

★ GenericServlet :-



- GenericServlet is a child implementing class of Servlet interface.
- It is an abstract class, because it gives the implementation to all the abstract methods except service() method.
- It is present inside javax.servlet package.
- When we are extending GenericServlet class it is mandatory to override the service() method.

★ Disadvantages of GenericServlet :-

i. It is protocol independent. It means it will accept any type of protocol request like http request, ftp (file transfer protocol) request, smtp (simple mail transfer protocol) request, etc.
ii. It does not have built in support or built in methods like doGet(), doPost(), doDelete(), etc. To accept type of request (get and post).
iii. For each and every request we have to write logic to check what type of request it is.
iv. When we are writing logic inside each and every servlet class then we are increasing the number of lines of code and also boiler plate code (duplicate code).
v. When we are writing a boiler plate code it will effects on code readability.
vi. It is very had to maintain.
vii. To overcome all these problems we will go for HttpServlet class.

★ HttpServlet :-

- It is a child class of GenericServlet.
- It is an abstract class.
- It is present in "javax.servlet.http" package.
- HttpServlet does not have any abstract method in it.
- Inside HttpServlet class we have doGet(), doPost(), doDelete() method, etc.
- All these methods are having basic implementations and that is the reason why java made HttpServlet class as abstract.
- We can create object of HttpServlet class and based on the requirement we have to override methods and have to write business logic.
- Java doesn't want people to make any modification in "HttpServlet" class.

★ Methods of HttpServlet :-

i. protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
{

}

ii. protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
{

}

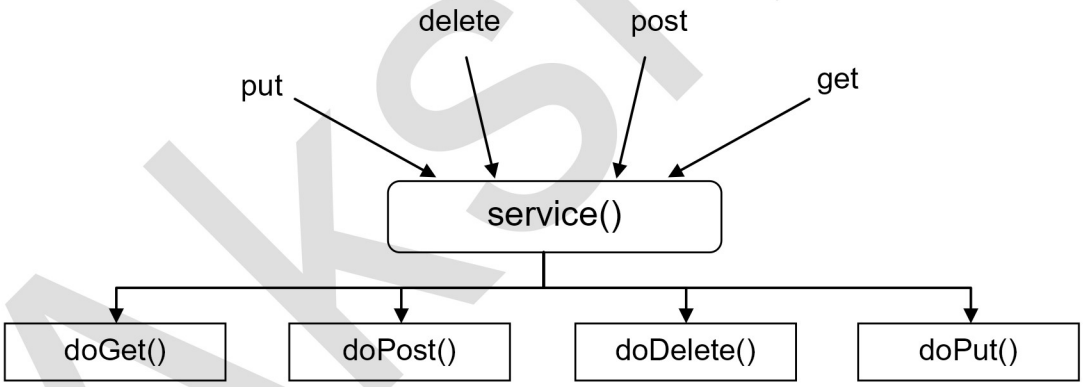
iii. protected void doHead()
iv. protected void doPut()
v. protected void doDelete()

★ Difference between get and post method :-

get	post
i. get is the default one.	i. post is not the default one.
ii. When sending get type of request, data will be sent through url.	ii. Here data would be sent through body.
iii. get method is not secured since the data is visible in url.	iii. post method is secured.
iv. Using get we can send only limited data (2048 characters).	iv. Using post we can send unlimited data.
v. When we are sending get type of request we can override either doGet() or service() method.	v. When we are sending post type of request we can override either doPost() or service() method.

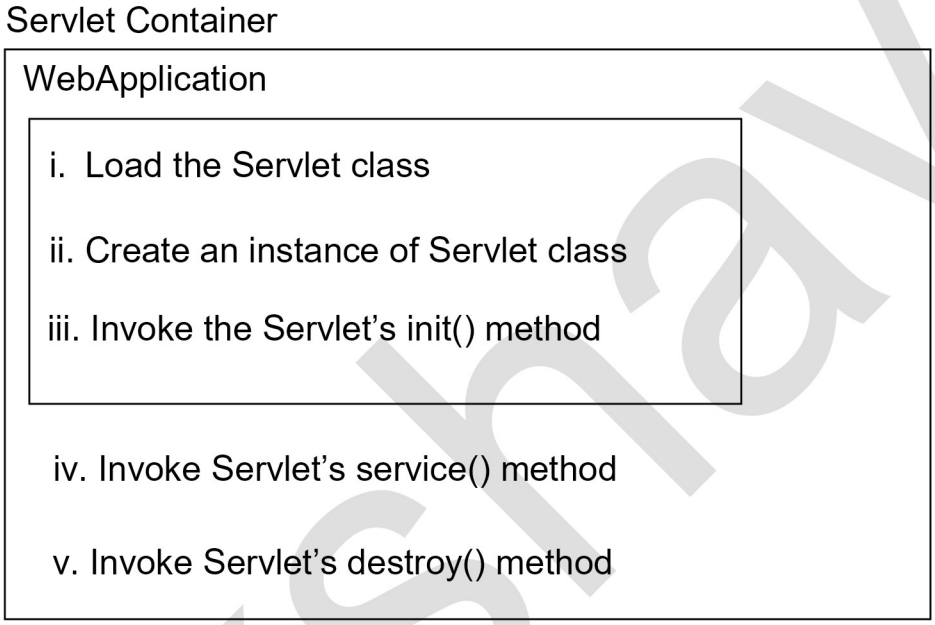
★ service() :-

- Its return type is void.
- Inside this method java programmers have written logic in such a way that it will check the request and internally it will call that respective method.
- E.g.



★ Servlet Life Cycle :-

- Whenever we run any servlet class or jsp files those classes will be loaded in a container called as "**Servlet Container**".
- It means, whenever we run any web application that web application will be loaded in a container called Servlet Container.
- The servlet container initially loads three major steps,
  i. Load the Servlet class
  ii. Create an instance of Servlet class
  iii. Invoke the Servlet's init() method



- The first three steps will be executed by server before itself i.e. during initialization of the server.
- E.g.
  The tomcat server loads all the three steps before itself so the server is "UP-RUNNING".
  Hence, the init() method is invoked only once during initialization of servlet classes in the servlet container.
- The servlet container invokes the service() method whenever the client makes any web request.
- It means, the service() method is invoked multiple times per web request calls made by the client.
- E.g. Client make any web request in the browser for 100 times the servlet container invokes service() method 100 times based on the number of HTTP calls that is made.
- Once all the requests and responses are made and finished (OR) when the whole web application process is fully completed then the servlet container invokes the destroy method.
- The destroy() method destroys the container which was created and it initially invokes GC which destroys whole memory in servlet container.

★ Important method in the servlet container :-

a. init() method :-

```
public void init() throws ServletException
{
    //servlet code
}
```

- init() method is invoked only once during the initialization of servlet container.

b. service() method :-

```
public void service(ServletRequest req, ServletResponse resp) throws ServletException, IOException
{
    //servlet code
}
```

- The service() method is invoked when web requests are made by the client.
- The service() method performs on GET, POST, PUT, DELETE i.e. doGet(), doPost(), doPut(), doDelete() methods.
- When the HTTP request is made by the client the servlet container invokes service() method.
- The service() method implicitly invokes doGet() method which handles incoming.
- The service() method later invokes doPost() method which handles the response back to the client.

c. destroy() method :-

```
public void destroy() throws ServletException
{

}
```

- The destroy() method is invoked when the whole process execution of Servlet container has finished.
- The destroy() method is invoked when the server is detached in the backend.
- The destroy() method will destroy the session which was created during initialization.
- The whole memory of the servlet container will be later destroyed by the GC which is invoked by the JVM.

★ RequestDispatcher :-

- It will help us to dispatch request from one resource to another resource (one servlet class to another servlet class OR one page to another page).
- It is an interface present in javax.servlet package.
- By using getRequestDispatcher(String str) method we can create object of RequestDispatcher.
- It is having two important methods,
  i. forward(req, resp)
  ii. include(req, resp)

i. forward(req, resp) :-

- It will ignore current page response and it will move to next page and it will display next page response.

ii. include(req, resp) :-

- It will display previous and current page response.

★ getRequestDispatcher(String str) :-

- Its return type is RequestDispatcher.
- It will take String as an argument where we can pass url of servlet class OR a page like html or jsp.
- It is an abstract method present in HttpServletRequest interface.

★ @WebServlet :-

- It is a class level annotation.
- It will help us to configure servlet class without need of xml file.
- @WebServlet("URL") or @WebServlet(urlpattern="")
- Inside @WebServlet annotation we can also do other configuration like loadOnStartup.
- It will help us to improve the code readability.

★ welcome-file-list :-

- It will help us to provide default configuration to display servlet class or html page or jsp page on the browser.
- If we are not providing explicitly welcome file list then servlet container or web container will check for index.html file.
- If index.html file is not available then servlet container or web container will check for index.jsp file.
- If both are not available then it will try to run application on the browser then we will get 404 Not Found error.
- If we want to provide welcome-file-list configuration then we will provide inside a web.xml file with the help of <welcome-file-list> tag.
- Inside a <welcome-file-list> tag we can do servlet class, html page, jsp page configuration with the help of <welcome-file> tag.
- We can do more than one welcome file configuration.
- When we are doing more than one configuration of welcome file then servlet container or web container will check for first welcome file if first welcome file is not available then it will check for next welcome file.
- If all welcome files are not available then we will get 404 Not Found error on the browser.
- <welcome-file-list>
  <welcome-file>servlet class / html file / jsp file</welcome-file>
  <welcome-file>servlet class / html file / jsp file</welcome-file>
  <welcome-file>servlet class / html file / jsp file</welcome-file>
  .
  .
  .
  </welcome-file-list>

★ How servlet will work?

- When we are sending a request from client to server we are sending request in the form of HTTP protocol.
- Server will not understand for which servlet class the request came then it will take the help of servlet container.
- Servlet container will read the url and it will take the help of web.xml file or @WebServlet annotation to understand for which servlet class request came.
- Once after reading the url, the servlet container will give the request to that respective servlet class.
- Once after giving request to servlet class servlet life cycle will start.

★ Servlet Life Cycle :-

- Servlet life cycle will be managed by servlet container.
- During the life cycle of servlet the following step are going to execute :-
  i. Loading the servlet class into memory
  ii. Instantiation of servlet class
  iii. Initialization of servlet class / invoking init() method
  iv. Request handling / invoking of service() method
  v. De-instantiation of servlet class / invoking destroy() method

i. Loading the servlet class into memory :-

- When the request is coming for the first time to the servlet class servlet container will load servlet class into memory.
- It means servlet container will take the help of Class loader to load servlet class.
- Servlet class will get stored only once inside a memory.

ii. Instantiation of servlet class :-

- Once after loading servlet class into a memory then servlet class object will be created.
- Servlet container is responsible to create an object of servlet class.
- For every servlet class only one object will be created (single pattern).

iii. Initialization of servlet class / invoking of init() method :-

- Once after creating an object of servlet class, servlet object will be initialized it means all the non static members will get initialized.
- To initialize servlet object servlet container will take the help of init() method.

iv. Request handling / invoking of service() method :-

- Once request will be received then servlet container will call service() method.
- service() method will check that what type of request and internally it will call that respective methods like doGet(), doPost(), doDelete(), etc.
- service() method will be invoked or called for each and every request.
- Whenever service() method is invoked a thread will be created to execute java program.
- From the second request onwards the servlet life cycle will start from fourth step.

v. De-instantiation of servlet class / invoking of destroy() method :-

- Once we shut down the server then servlet container will call the destroy() method.
- destroy() method is responsible to destroy the objects or threads, which are created during life cycle of server.

Note:-

- During the life cycle of servlet class init() method, service() and destroy() will play important role and all these methods declared inside servlet container.

★ HttpServletRequest :-

- It is an interface present in javax.servlet.http package.
- It is used to hold the request which is coming from client.
- Servlet container is responsible for creating object of the HttpServletRequest.

★ HttpServletResponse :-

- It is an interface present in javax.servlet.http package.
- It will help to send a response back to the client.
- Servlet container is responsible for creating object of the HttpServletResponse.

★ loadOnStartup :-

- It will help us to load all the servlet classes at the time of server start.
- It will also create the object and initialize servlet class members at the time of server start.
- We can provide load on startup configuration inside a web.xml or inside a @WebServlet() annotation.
- Inside a loadOnStartup we can provide only positive numbers like 0, 1, 2, 3 …
- When we are having more than one servlet class then loadOnStartup will load all the servlet class based on the positive number.
- The servlet class which is having lowest number will get the higher priority.
- If we are providing negative number inside a loadOnStartup then application will work normally.
- If all the servlet classes are having a same number of loadOnStartup then servlet container will decide order of servlet class.
- @WebServlet(loadOnStartup = positiveNumber)

★ sendRedirect() :-

- It is a method present inside HttpServletResponse interface.
- Its return type is void.
- It will help to send the request to internal resources like servlet class, html and jsp pages and to external resources.

★ Cookie :-

- It is used to handle session.
- It is an object where it is having some small piece of information.
- Inside the cookie we are going to store information or data in the form of key and value pair.
- Cookie is class present in javax.servlet.http package.

★ How to create cookie –

- Cookie referenceVariable = new Cookie(String key, String value)
- To add cookies on the browser or server we are going to take the help of addCookie() method.

★ How to get cookie :-

- To get the cookies we are going to take the help of getCookies() method.

```
package org.jsp.cookie;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class Demoa extends HttpServlet{
     @Override
     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

          Cookie[] cookies = req.getCookies();
          for(Cookie cookie : cookies) {
               String name = cookie.getName();
               String value = cookie.getValue();
               System.out.println(name);
               System.out.println(value);
          }

     }

}
```

★ How to check cookies are stored on the browser or not?

- Ctrl + Shift + i → Click on application → Expand cookies → Click on url i.e. http:/localhost/:8080

Note:-

- When we are sending request in the form of http protocol, every request is considered as new request, because http is a stateless protocol.
- To overcome this problem we will take the help of session management.

★ Disadvantages of Cookie :-

- Inside a Cookie we can store only String type of data, but we cannot store any object.
- To overcome this problem we go for HttpSession.

★ HttpSession :-

- It is an interface present in javax.servlet.http package.
- We will take the help of getSession() method to create an object of HttpSession.
- Inside HttpSession we can set any values and object with help of setAttribute() method.
- With the help of getAttribute() method we can get the value which is stored inside a HttpSession.

★ Disadvantages of Servlet :-

- Inside Servlet you will mix java code and html code.
- When we are writing business logic and presentation logic inside a servlet class it will affect on code readability.
- If we make any small mistake inside presentation logic in servlet class then you will not get the response as expected.
- Inside each and every servlet class we have to override either doGet(), doPost(), service() method.
- If you make any small modification inside a servlet class again we have to compile and run the application in server.
- To overcome all these problems we will go for JSP.

JSP

★ Introduction :-

- Java Server Page or Jakarta Server Page.
- It is used to develop view page inside a web application.
- Inside JSP we can write java code and html code separately.
- JSP will provide tags by using that tags we can separate java and html code.
- JSP will provide implicit object like session, request, exception, etc.

★ JSP Tags :-

- There are mainly three types of tags,
  i. Scripting Tag
  ii. Directive Tag
  iii. Action Tag

i. Scripting Tag :-

- It is used to write java code inside a JSP page.
- In Scripting tag there are also three types,
  i. Declarative / Declaration Tag
  ii. Scriptlet Tag
  iii. Expression Tag

i. Declarative / Declaration Tag :-

- Inside this tag we can declare or create variables and methods.
- The variables which are declared inside this tag, they are global variables (class level scope).
- Syntax :- <%! %>

Demo.jsp

```
<html>
<body>

<%!

int a = 10;
public int multi(){
        return a*a;
}

%>

</body>
</html>
```

Translation

Demo_jsp.jav

```
class Demo_jsp{

        int a = 10;
        public int multi(){
                return a*a;
        }

}
```

- When we are writing java code inside a jsp internally it will convert into servlet class (java code).

★ How to check jsp code converted into java file?

Go to eclipse workspace folder → Click on meta data → Click on plugins → Search for org.eclipse.wst.server.core folder (Hit Enter) → Click on tmp0 → Click on work → Click on Catalina → Click on localhost → Click on project name → click on org → click on apache → click on jsp → there you will get the java file created.

ii. Scriptlet Tag :-

- Syntax :- <% %>
- Inside this tag we can write variables, looping statements, decision statements, etc.
- The statements which we have written under this tag once after converting into java file or servlet class file.
- These statements will be written under service() method.
- E.g.

Demo.jsp

```
<html>
<body>

<%

for(int i=0;i<4;i++){
        out.print(i);
}

%>

</body>
</html>
```

Translation

Demo_jsp.jav

```
class Demo_jsp{

    void _jspService(){
        for(int i=0;i<4;i++){
            out.print(i);
        }
    }

}
```

iii. Expression Tag :-

- Syntax :- <%= %>
- Expression tag will help us to print data on browser.
- Expression tag will help to make code more concise and readable.
- E.g.
  <%= “Hello” %>
  <%= i %>

★ Examples on Scripting Tags :-

i. first.jsp :-

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>First Example</title>
</head>
<body>

<%="Welcome to JSP"%>

</body>
</html>
```

ii. second.jsp :-

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Second Example</title>
</head>
<body>

<%
     int a = 10;
     int b = 20;
%>

<%="Sum is "+(a+b)+"<br>" %>
<%="Product is "+(a*b) %>

</body>
</html>
```

iii. form.jsp and third.jsp :-

form.jsp :-

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Form Example</title>
</head>
<body>

<form action="third.jsp">
     Name : <input type="text" name="name">
     <input type="submit">
</form>

</body>
</html>
```

third.jsp :-

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Third Example</title>
</head>
<body>
     <%
               String name = request.getParameter("name");
     %>
     <%="Hello "+name %>
</body>
</html>
```

iv. fourth.jsp :-

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Fourth Example</title>
</head>
<body>
<%!
     public String msg(){
         return "Good evening";
     }
%>
<% String greeting = msg();%>
<%=greeting %>
</body>
</html>
```

v. five.jsp :-

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Fifth Example</title>
</head>
<body>
<%
     int a = 10;
     int b = 20;
     String res = (a>b)?a+" is greater":b+" is greater";
%>
<%=res %>
</body>
</html>
```

vi. six.jsp :-

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Sixth Example</title>
</head>
<body>

<%
     session.setAttribute("user", "postgres");
%>
<%=session.getAttribute("user")%>

</body>
</html>
```

vii. seven.jsp :-

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Seventh Example</title>
</head>
<body>
<%
     int random = (int)(Math.random()*10);
%>

<h1>Random number is</h1>

<%=random %>
</body>
</html>
```

viii. eight.jsp :-

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Eight Example</title>
</head>
<body>
<%!
     String[] s = {"red","green","yellow"};
%>
<%=s[1] %>
</body>
</html>
```

ix. nine.jsp :-

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Nine Example</title>
</head>
<body>
<%!
     int[] i = {1,2,3,4};
%>
<%
     for(int ele:i){
%>
          <%=ele %>
<%
     }
%>
</body>
</html>
```

x. ten.jsp :-

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.util.Date" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Tenth Example</title>
</head>
<body>

<%="Current Date is : "+new Date() %>

</body>
</html>
```

ii. Directive Tag :-

- JSP Directive.
- It will help JSP container how to process JSP page.
- There are three types of Directive tags,
  i. page Directive
  ii. include Directive
  iii. taglib Directive

i. page Directive :-

- Syntax :- <%@page attribute=”value” %>
- Attributes :- language, contentType, pageEncoding, import, session, isErrorPage, errorPage.
- It will help us to defines page setting like language, contentType, pageEncoding, import, session, isErrorPage, errorPage, isELIgnored=”false”, extends.

ii. include Directive :-

- Syntax :- <%@ include file = “value” %>
- It will help us to include content of one JSP file into another file at compile time.

★ Expression Language:-

- It is a feature of JSP where it will simplify to access the data which is stored inside a request and session attribute without writing java code.

iii. taglib Directive :-

- It will help us to import JSTL tag into JSP page.
- Syntax :- <% taglib url="" prefix="value" %>

★ JSP Life Cycle :-

- JSP life cycle will be managed by JSP container.
- Following steps will be followed during the life cycle of JSP
  i. Translation Phase
  ii. Compilation Phase
  iii. Loading servlet class into memory
  iv. Instantiation of servlet class
  v. Invoking of _jspInit() method
  vi. Request handling / invoking of _jspService() method
  vii. Invoking of _jspDestroy() method

i. Translation Phase :-

- In this phase JSP file will be converted into java file.

ii. Compilation Phase :-

- Once after converting JSP file into java file, java file will get compiled.
- Once after successful compilation .class file will get generated.

iii. Loading servlet class into memory :-

- After the compilation of java file the JSP container will load the java file i.e. servlet class into memory with the help of Classloader.

iv. Instantiation of servlet class :-

- After loading the servlet class JSP container will create the object of servlet class.
- JSP container is responsible for creating an object of servlet class.

v. Invoking of _jspInit() method / Initialization of servlet class :-

- After the instantiation of servlet class the JSP container will invoke the _jspInit() method to initialize the servlet class object.

vi. Request Handling / Invoking of _jspService() method :-

- After the initialization of servlet class JSP container will invoke the _jspService() method for request handling of client.
- Invoking of _jspService() method depends on the request coming from client i.e. if client made 100 requests then _jspService() method will get invoked 100 times.

vii. Invoking of _jspDestroy() method :-

- When server is stopped / shutdown then JSP container will invoke the _jspDestroy() method.

★ JSTL (JSP Standard Tag Library) :-

- It is collection of predefined tag.
- It will help to simplified JSP develop.
- By using JSTL tag we can avoid completely scriptlet tag.
- JSTL will provide predefined tags like core, function
- By using core tag we can write decision statement, printing statement.
- JSTL tag will convert internally java code when JSP file compiled.
- Function tag – it used to perform **common string operations** inside JSP pages without writing Java code.
- To write JSTL tag inside a JSP page we will take help of taglib directive.

iii. Action Tag :-

- It will help to perform specific operation or tasks inside JSP file like forward, include, etc.

★ JSP Implicit Object :-

- In JSP file to write java code we use frequently some objects like HttpServletRequest, HttpServletResponse, PrintWriter, etc.
- Inside JSP frequent used object is created and stored inside a reference variable by default these object is known as JSP implicit object.
  i. request
  ii. response
  iii. out
  iv. session
  v. exception

Hibernate

★ Disadvantages of JDBC :-

i. In JDBC, programmer is responsible to handle exception (ClassNotFoundException, SQLException).
ii. When programmer is handling exception then length of the code will get increase and it will affect on code readability.
iii. In JDBC programmer should create table.
iv. In JDBC we are performing operation in the form of query but not in object.
v. In JDBC we are using SQL query in order to perform operation but Hibernate will provide its own query language that is HQL.
vi. Each and every time you have to perform some 5 steps operation (Boiler plate code).
vii. Performance wise compared to JDBC, Hibernate is best choice.

★ ORM (Object Relational Mapping) :-

- It will help to convert java object into database rows / columns.
- In the market we are having different ORM tools like TopLink, iBATIS, and Hibernate, etc.
- So among these we are going to use Hibernate.

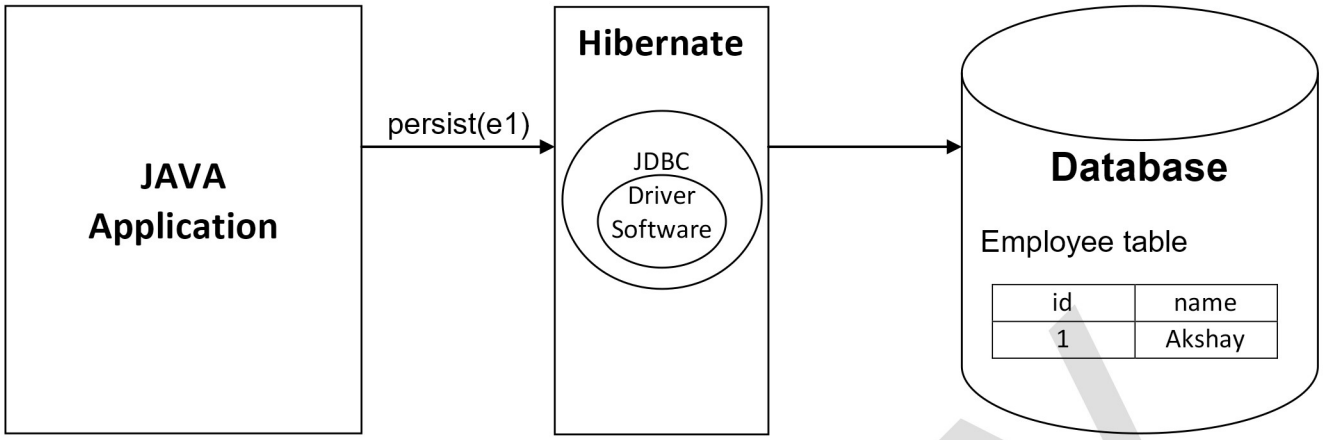
★ Hibernate :-

- It is ORM tool.
- It is introduced by Gavin King in 2001.
- Hibernate will helps to perform CRUD operation in a form of object.
- Hibernate will acts as an intermediate between java application and database.

★ Advantages of Hibernate :-

i. Hibernate is open source.
ii. Hibernate will create table automatically.
iii. Hibernate will handle the exception internally.
iv. Hibernate will use internally cache concept to improve the application performance.
v. Hibernate is having its own query language i.e. HQL.
vi. Hibernate will provide methods like persist(), save(), remove(), …

★ How Hibernate will work?



- When we are performing operation on database by using hibernate, hibernate will not perform task alone.
- Internally hibernate will use JDBC and JDBC will use Driver Software.
- When we are using hibernate then we have to add 2 dependencies inside pom.xml.

★ How to provide hibernate configuration inside maven project?

- src/main/resource → META-INF → persistence.xml
- In order to provide configuration of hibernate we will create folder that is META-INF under src/main/resource.
- We will create xml file under META-INF folder the name of the xml file should be persistence.xml

★ Entity Class :-

- A class which is annotated with **@Entity** annotation is known as Entity class.

Note:-

- Hibernate will create table only for Entity class.

★ How to make a class as Entity class :-

i. Class should be annotated with @Entity annotation.
ii. We have to use @Id annotation on top of variable which you want to make primary key.
iii. We can have n number of parameterized constructor but compulsory we have to define no argument or default constructor.
iv. Generate getters and setters methods.
v. If you want to implement Serializable interface but it is optional.

★ Persistence :-

- It is a class which is present in javax.persistence package.
- It is having so many static methods in that we are going to take the help of **createEntityManagerFactory(String persistenceUnitName)** method.

★ createEntityManagerFactory(String persistenceUnitName) :-

- It is a static method which is present in Persistence class.
- Its return type is EntityManagerFactory.
- It will take String as an argument where we need to pass persistence unit name.
- It will read all the properties from persistence.xml file then if all credentials are valid like url, username, password then it will establish the connection between java application and database.

★ EntityManagerFactory :-

- It is an interface which is present in javax.persistence package.
- createEntityManagerFactory() method will help us to create an object of EntityManagerFactory.

★ createEntityManager() :-

- It is abstract method which is present inside EntityManagerFactory interface.
- Its return type is EntityManager.
- It will help us to create an object of EntityManager interface.

★ EntityManager :-

- It is an interface which is present in javax.persistence package.
- Hibernate will provide predefined methods to perform CRUD operations, all those methods are present in EntityManager interface.

★ getTransaction() :-

- It is an abstract method present inside EntityManager interface.
- It will help us to create object of EntityTransaction interface.

★ EntityTransaction :-

- It is an interface which is present in javax.persistence package.
- In hibernate if you want to do any transaction then hibernate will provide predefined methods.
- All those predefined methods are present inside EntityTransaction interface.

i. begin() :-

- It will help us to start or begin the transaction.

ii. commit() :-

- It will help us to make permanent change inside database.

Note:-

- When we are performing fetch or read operation then creating object EntityTransaction is optional.

★ Predefined methods provided by hibernate to perform CRUD operations :-

i. persist(Object entity) :-

- This method will help us to store the object inside the database as one row.
- Inside persist() method we have to pass Entity class object.

ii. merge(T entity) :-

- It will helps to perform update operation.

iii. remove(Object entity) :-

- It will help us to remove the object from the database.

iv. find(Class<T> entityClass, Object primaryKey) :-

- It will help us to fetch or read the data from the database.

★ Assignment (02-04-2025) :-

- Perform CRUD operations on Student class which is having fields like id, age, name, subject, marks using hibernate.

studentcrud project pom.xml file :-

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>org.jsp</groupId>
	<artifactId>studentcrud</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<dependencies>
		<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>5.6.15.Final</version>
		</dependency>
		<!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
		<dependency>
			<groupId>org.postgresql</groupId>
			<artifactId>postgresql</artifactId>
			<version>42.7.3</version>
		</dependency>
	</dependencies>
</project>
```

studentcrud project persistence.xml file :-

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
             http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
             version="2.1">

    <persistence-unit name="akshay" transaction-type="RESOURCE_LOCAL">

        <properties>
            <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" />
           <!-- DB Driver -->
            <property name="javax.persistence.jdbc.url"
value="jdbc:postgresql://localhost:5432/hibernatecrud" />
            <!-- DB Name -->
            <property name="javax.persistence.jdbc.user" value="postgres" />
            <!-- DB User -->
            <property name="javax.persistence.jdbc.password" value="root" />
            <!-- DB Password -->

          <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
          <!-- DB Dialect -->
            <property name="http://hibernate.hbm2ddl.auto" value="update" />
            <!-- create / create-drop / update -->
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.show_sql" value="true" />
            <!-- Show SQL in console -->
            <property name="hibernate.format_sql" value="true" />
            <!-- Show SQL formatted -->
        </properties>

    </persistence-unit>

</persistence>
```

studentcrud project Student.java class :-

```
package org.jsp.entity;

import javax.persistence.Entity;
import javax.persistence.Id;
@Entity
public class Student {

     @Id
     int id;
     int age;
     String name;
     String subject;
     double marks;

     Student(){
     }

     public Student(int id, int age, String name, String subject, double
marks) {
           super();
           this.id = id;
```

```
            this.age = age;
            this.name = name;
            this.subject = subject;
            this.marks = marks;
      }

      @Override
      public String toString() {
            return "Student [id=" + id + ", age=" + age + ", name=" + name + ",
subject=" + subject + ", marks=" + marks
                        + "]";
      }
}
```

studentcrud project StudentDriver.java class :-

```
package org.jsp.driver;

import java.util.Iterator;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import org.jsp.entity.Student;
public class StudentDriver {

	public static void main(String[] args) {

		EntityManagerFactory emf =
				Persistence.createEntityManagerFactory("pavan");

		//create Entity Manager
		EntityManager em = emf.createEntityManager();

		//create transaction
		EntityTransaction et = em.getTransaction();

		//insert operation
		et.begin();
		Student s = new Student(2,26, "Aniket","Java",90);
		em.persist(s);
		et.commit();
		System.out.println("Record Inserted");

		//read operation
//		Student s = em.find(Student.class, 1);
//		System.out.println(s.toString());
//		System.out.println("Record Fetched");

		//update operation
//		et.begin();
//		Student s = em.find(Student.class, 1);
//		s.setAge(28);
//		s.setName("Akshay");
//		s.setSubject("SQL");
//		s.setMarks(50);
//		em.merge(s);
//		et.commit();
//		System.out.println("Record Updated");

		//delete operation
//		et.begin();
//		Student s = em.find(Student.class, 1);
//		em.remove(s);
//		et.commit();
//		System.out.println("Record Deleted");
	}
}
```

★ Mapping :-

- It will help us to establish connection between the tables.
- Mapping is one of the key features of Hibernate.
- We can establish connection in two ways,
  i. Unidirectional Mapping
  ii. Bidirectional Mapping
- Hibernate will provide annotations to achieve mapping, those are,
  i. One to one
  ii. One to many
  iii. Many to one
  iv. Many to many

i. Unidirectional Mapping :-

- In unidirectional mapping one table is having another table details.
- E.g.

Customer Table

ID	Name	Payment	OrderID
101	Ram	Success	201
102	Shyam	Success	202
103	Akash	Success	203

Order Table

OrderID	Name	Price
201	Pulav	500
202	Biryani	300
203	Pav Bhaji	200

- In the above table Customer table is having Order Table information but Order table does not have the Customer table information.
- When we are fetching customer data then we can also fetch order data.
- When we are fetching order information we cannot fetch customer information because it is unidirectional mapping.

ii. Bidirectional Mapping :-

- In Bidirectional mapping both tables having each other details.
- E.g.

Customer Table

Cust_ID	Name	Payment	OrderID
101	Ram	Success	201
102	Shyam	Success	202
103	Akash	Success	203

Order Table

OrderID	Name	Price	Cust_ID
201	Pulav	500	101
202	Biryani	300	102
203	Pav Bhaji	200	103

- In the above table Customer table is having Order table information and Order table having customer table information.
- So when we are fetching customer data then we can also fetch order data.
- Also when we are fetching order information we can also fetch customer information because of bidirectional mapping.

★ Persistence.xml :-

- It is an xml file.
- Inside xml file we are going to write hibernate configuration like url, username and password in the form of tag.
- Inside Persistence.xml we can have more than one database configuration.
- We will create Persistence.xml file under META-INF folder.
- Syntax :-

```
<persistence-unit name="anyName" >
    <properties>
        <property>  </property>
    </properties>
</persistence>
```

★ JPA :-

- JPA stands for **Java Persistence API**.
- It will provide certain functionality and standard to ORM tool.
- JPA classes and interfaces are present in javax.persistence package.
- Recently they change package name to jakarta.persistence package.
- But all the functionality is same just package name is changed.

Hibernate	JPA
i. Hibernate will use Hibernate.cfg.xml file for configuration.	i. JPA will use Persistence.xml file for configuration.
ii. In Hibernate classes and interfaces are present inside org.hibernate.annotations package.	ii. In JPA all the JPA classes and interfaces are present inside javax.persistence package.
iii. In Enterprise based application the Hibernate is not taken defaultly.	iii. In spring based application or enterprise application JPA is taken defaultly.
iv. In Hibernate we use SessionFactory, Session to perform CRUD operations on database.	iv. In JPA we will use EntityManagerFactory, EntityManager and EntityTransaction in order to perform CRUD operations on database.
v. Example :- @Fetch, @Generic, @Generator, etc.	v. Example :- @Entity, @Id, @Table

★ POJO Class :-

- POJO stands for Plain Old Java Object.
- It is an entity class whose object will be stored inside database.

★ Hibernate Life Cycle :-

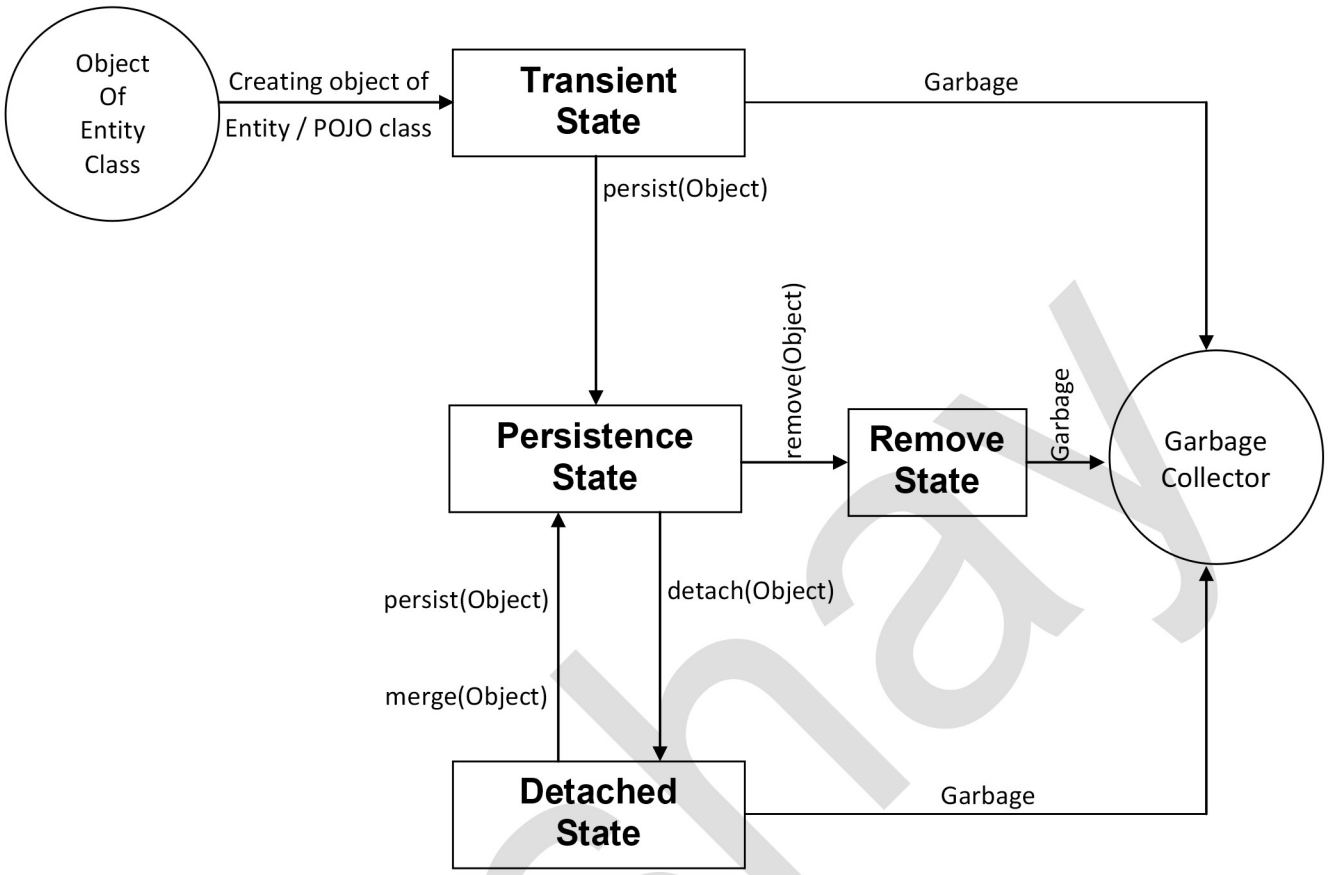


Figure: - Life Cycle Of Hibernate

- Following steps will be followed during the life cycle of hibernate
  i. Transient State
  ii. Persistence State
  iii. Detached State
  iv. Remove State

i. Transient State :-

- When we are creating object of POJO class then we are in transient state.
- In transient state if we do any modification then that modification will not be affected to database.
- If we are not moving transient state to other state then POJO class object or entity class object will get destroyed.

ii. Persistence State :-

- To move from transient state to persistence state we will take the help of persist() method.
- If we do any modification then it will affect on database.

iii. Detached State :-

- To move from persistence state to detached state then we will take the help of detach() method.
- If we are doing any modification then it will not affect on database.
- If we are not moving any other state then POJO class object or entity class object will get destroyed.
- If we want to move detached state to persistence state then we will take the help of persist() method or merge() method.

iv. Remove State :-

- To move from persistence state to remove state then we will take the help of remove() method.
- Modification which we are doing it will affect on database.
- The object which we are removing from database will get destroyed by garbage collector.

★ Composite Key :-

- If we want to make more than one column as a primary key then we will go for composite key concept.

★ How to achieve Composite Key using hibernate?

- First you have to create one separate class and inside that class we will write the variables or we declare the variables which we want to make as primary key.
- For that specific class we will annotate with **@Embeddable** annotation.
- That specific class should implements **Serializable** interface (It is marker interface which is present inside java.io package).
- We should have either no argument or default constructor.
- Generate public getters and setters method.
- It is recommended to override equals() and hashcode() methods.
- Entity class has a specific class as a variable.
- On top of that variable we will write **@EmbeddedId** inside an entity class.

★ Cache :-

- It is a key feature of hibernate.
- It will help us to reduce the number of queries which are hitting to the database.
- Cache will help us to improve the application performance.
- There are two types of Cache,
  i. First-level cache
  ii. Second-level cache

i. First-level cache :-

- Hibernate will use first level cache as default.
- We cannot disable first level cache.
- First level cache managed by Entity Manager it means for every Entity Manager separate first-level cache is used.

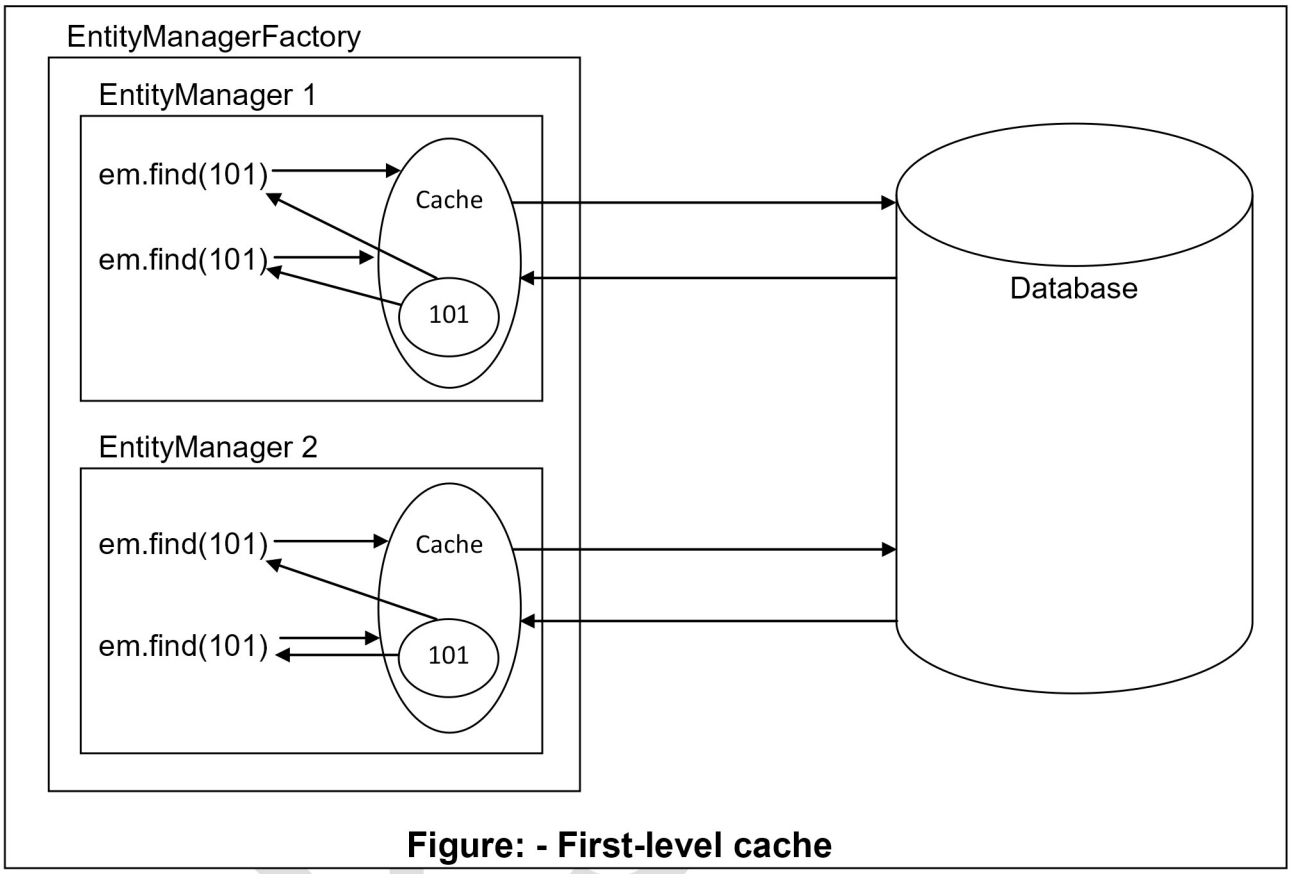


Figure: - First-level cache

- In Hibernate, the first-level cache is associated with the EntityManager.
- When we run a query for the first time using an EntityManager, Hibernate fetches the data from the database and stores it in the **Cache** (memory).
- If we run the same query again in the same EntityManager, Hibernate will not go to the database again. Instead, it will return the data from the **Cache**.
- This helps in reducing the number of queries which are hitting database and improves performance.
- The first-level cache is specific to a single EntityManager. If we use another EntityManager, it will not have the data Cached by the first one.

ii. Second-level cache :-

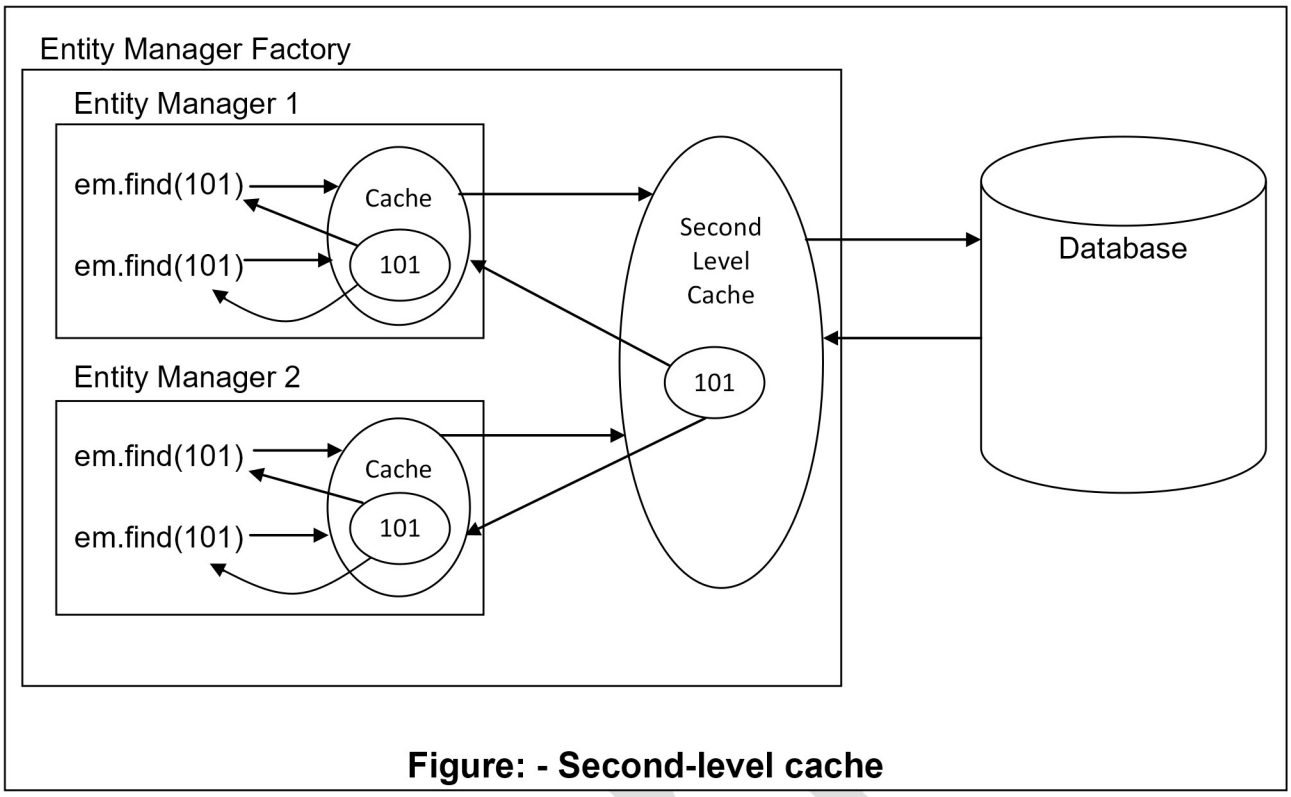


Figure: - Second-level cache

- If you want to use second level cache then explicitly we need to enable.
- To enable second level cache we will take the help of **@Cacheable** annotation.
- For every entity manager factory second level cache will be created.
- We are having different second level cache in that we are using **EH-Cache.**

★ How to use second level EH-Cache?

- Go to maven repository → search for Hibernate-Ehcache → Click on search → Select Hibernate Ehcache Relocation → Select a version which is same as hibernate core version (it is recommended to choose the same version) → copy the dependency and paste in pom.xml file.
- https://codeshare.io/aloyB5 search for this site and take the persistence file.

★ How the second-level cache works?

- The second-level cache is used to store data across multiple EntityManagers.
- Unlike the first-level cache, which is limited to one EntityManager, second-level cache allows sharing of cached data between different EntityManagers.
- Data fetched by any EntityManager can be stored in the second-level cache. If another EntityManager requests the same data, it will be served from the second-level cache instead of hitting the database.
- This helps in improving performance and reducing the number of queries which are hitting to the database across the whole application.
- The second-level cache is specific to a single EntityManagerFactory, and it allows sharing cached data across multiple Entity Managers.

★ HQL :-

- HQL stands for Hibernate Query Language.
- It is an object oriented, database independent query language where it will help us to communicate with the database.
- Inside HQL we will use entity class name and variable.
- HQL will support named parameter and positional parameter.
- It is always recommended to use named parameter because compared to positional parameter named parameter is more readable.
- HQL will take the help of **Query** interface in order to communicate with the database.

★ Query :-

- Query is an interface which is present in javax.persistence package.
- We will take the help of **createQuery()** method to create an object of Query interface.
- Query interface having so many methods in that we are going to use below methods,
  i. setParameter(int position, Object value);
  ii. setParamater(String name, Object value);
  iii. getResultList();
  iv. getSingleResult();
  v. executeUpdate();

i. setParameter(int position, Object value) :-

- Its return type is Query.
- It will help us to set the value for position parameters which is used in HQL.

ii. setParameter(String name, Object value) :-

- Its return type is Query.
- It will help us to set the value for named parameters which is used in HQL.

iii. getResultList() :-

- Its return type is List.
- It will help us to get all the objects which are stored inside the database.

iv. getSingleResult() :-

- Its return type is Object.
- When we know that based on the HQL query we will get only one object then we will use the getSingleResult() method.

v. executeUpdate() :-

- Its return type is int.
- It is used to execute HQL query.

★ Cascade :-

- If we want to tell Hibernate to perform same operation on entity dependent object that are performed on entity object then we will go for **Cascade** concept.
- We can use **Cascade** concept if there is a relation between entities / table.
- We can use **Cascade** as mentioned below,

```
cascade = CascadeType.ALL
cascade = CascadeType.REMOVE
cascade = CascadeType.PERSIST
cascade = CascadeType.MERGE
cascade = CascadeType.DETACH
```

★ Fetch :-

- Fetch type will help to define when to load entity dependent object from database.
- If we want to use fetch type concept then there should be relation between entities / tables.
- Fetch type LAZY, it will load entity dependent object from database, when it is accessed explicitly.
- Fetch type EAGER, it will load entity dependent object from database, when main entity is loaded.
- There are two types of fetch
  i. LAZY
  ii. EAGER
- Default fetch types for mapping
  i. OneToOne → EAGER
  ii. OneToMany → LAZY
  iii. ManyToOne → EAGER
  iv. ManyToMany → LAZY
- If we want to change the fetch type explicitly we have to change,

```
fetch = FetchType.EAGER
fetch = FetchType.LAZY
```

★ Annotations used in Hibernate:-

Annotation	Description
i. @Entity	• Used to create an Entity class and map it to a table in the database.
ii. @Id	• Used to mark a field as the primary key of an entity.
iii. @Table	• Used to specify the name of the database table that the entity will be mapped to.
iv. @Column	• Used to define the mapping between the entity class attributes and a column in the corresponding database table.
v. @GeneratedValue	• Used to specify how the primary key value should be generated.
vi. @OneToOne	• One to one relationship is a relationship where each entity in one table corresponds to one entity in another table.
vii. @OneToMany	• One to many relationship is a type of association where one entity can be associated with multiple instances of another entity.
viii. @ManyToOne	• In many to one relationship, many instances of an entity are associated with one instance of another entity.
ix. @ManyToMany	• Many to many relationship is a type of association where multiple instances of one entity can be associated with multiple instances of another entity.
x. @Embeddable	• It is used to mark a class whose instances can be stored as a part another entity.
xi. @EmbeddedId	• It is used to define composite key for an entity.
xii. @Cacheable	• Used to create a second level cache. • It is second level cache which is shared across multiple entity managers.

HIBERNATE END

SPRING

★ Framework :-

- It is a collection of predefined classes, interfaces and other libraries (or) components where it will helps to simplify development.
- E.g. Collection, Hibernate, Spring, etc…

★ Advantages of Framework :-

- Reduce time because already logic is written based on requirement the classes and interfaces are used.
- Reusability (we can use multiple times classes, interfaces, etc…).
- Programmer has to focus on business logic only.

★ EJB :-

- EJB stands for Enterprise Java Beans.
- It is an API.
- The applications which are developed by using EJB are heavyweight application.
- EJB does not have its own servers.
- The applications which are developed by using EJB those applications are tightly coupled applications.

★ Tightly Coupled :-

- One class is highly dependent on another class.
- It means if we do any modification in one class it will affect to other dependent class.

★ Spring :-

- Spring is a framework.
- It is introduced by Rod Johnson in 2003.
- Previous name of spring is interface-21.
- It is an alternative of EJB.
- By using spring we can develop loosely coupled applications.
- By using spring we can develop enterprise application as well as web application.
- Spring is a framework of frameworks because it is having other modules like springcore, spring mvc, spring boot, spring JPA, spring JDBC, spring security, etc...
- By using spring we can develop light weight application.

★ Loosely Coupled :-

- One class is not highly dependent on another class.
- It means if we do modification in one class it will not affect to other class.

★ Enterprise Application :-

- It is software where it will help to build business applications and we will handle large amount of data.
- E.g. CRM, HRM, etc…

★ Difference between EJB and Spring :-

EJB	Spring
i. EJB is an API.	i. Spring is a framework.
ii. By using EJB we can develop heavy weight application.	ii. By using spring we can develop light weight application.
iii. The applications which are developed by using EJB are tightly coupled applications.	iii. The applications which are developed by using Spring are loosely coupled applications.
iv. EJB does not have its own servers.	iv. Spring has its own servers.

★ Spring Container :-

- Spring container is responsible to manage objects of bean class.
- A class whose object is created by spring container is known as bean class.
- If require spring container will inject one object into another object.
- From object creation to object destruction spring container will manage object life cycle.
- If we want spring container to create object then we will provide configuration file.
- Spring container provides different-different containers those are,
  i. Core-Container
  ii. J2EE-Container
  iii. Web-Container

★ IOC Container :-

- IOC means Inversion Of Control.
- It is a key feature of spring framework.
- It will make application as loosely coupled.
- It is having Core container and J2EE container.

i. Core Container :-

- If you want to use core container we will take the help of BeanFactory interface.

★ BeanFactory :-

- It is an interface.
- It is present in org.springframework.beans.factory package.
- If you want to create an object of BeanFactory then we will take the help of BeanFactory child implementing class i.e. **XmlBeanFactory.**

★ getBean(String name) :-

- Its return type is Object.
- It is present inside BeanFactory interface.
- It will help to return the object which is created inside spring container.

★ XmlBeanFactory :-

- It is child implementing class of BeanFactory interface.
- It is deprecated in JDK 1.5 version.
- It will take Resource as an argument.

★ Resource :-

- It is an interface which is present in org.springframework.core.io package.
- If we want to create object of Resource interface then we will go for ClassPathResource class which is the child implementing class of Resource interface.

★ ClassPathResource :-

- It is child implementing class of Resource interface.
- It is having so many constructors in that we are using constructor which is taking String as an argument.
- Inside that constructor we are going to provide configuration file name (xml file).

★ Disadvantages of BeanFactory :-

- When you are using a BeanFactory we are writing deprecated code.
- It is a lazy instantiation, it means that core container will not create an object until and unless we call getBean() method.
- We have to do configurations using xml file, but xml file configuration is outdated.
- BeanFactory will not support for annotations by default.

★ J2EE – Container :-

★ ApplicationContext :-

- It is an interface which is present in org.springframework.context package.
- It is an early instantiation, it means that the object will be created during the class loading process.
- If we want to a create object of ApplicationContext then we will go for child implementing class i.e. ClassPathXmlApplicationContext.

★ ClassPathXmlApplicationContext :-

- It is child implementing class of ApplicationContext interface.
- It is having so many constructors in that we are going to take the help of constructor which is taking String as an argument.

★ .xml file (Spring configuration) :-

- For spring configuration file we can give name based on our requirement.
- Spring container will take the help of configuration file to create an object of bean class.
- Inside this xml file we are going to provide bean id, class, init-method, ref, etc…

★ Example of BeanFactory :-

Bean Class:-

```
package org.jsp.springcore;

public class Student {
	int id;
	String name;
	public Student() {
		System.out.println("Hi I am student class");
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
}
```

Driver Class:-

```
package org.jsp.springcore;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class StudentDriver {

	public static void main(String[] args) {
		Resource resource = new ClassPathResource("my_cnf.xml");
		BeanFactory bf = new XmlBeanFactory(resource);
		Student bean = (Student) bf.getBean("stu");
		System.out.println("id : "+bean.id);
		System.out.println("name : "+bean.name);

	}
}
```

.xml file:-

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-
3.0.xsd">
      <bean id="stu" class="org.jsp.springcore.Student">

            <property name="id" value="101"></property>
            <property name="name" value="ram"></property>

      </bean>
</beans>
```

★ **Example of ApplicationContext :-**

Bean Class:-

```
package org.jsp.j2eecontainer;

public class Employee {

      int id;
      String name;
      Employee(){
            System.out.println("I am employee class constructor.");
      }
      public Employee(int id, String name) {
            super();
            this.id = id;
            this.name = name;
      }
      public int getId() {
            return id;
      }
      public void setId(int id) {
            this.id = id;
      }
      public String getName() {
            return name;
      }
      public void setName(String name) {
            this.name = name;
      }
      public void m1() {
            System.out.println("I am m1()");
      }

}
```

Driver Class:-

```
package org.jsp.j2eecontainer;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class EmpDriver {
	public static void main(String[] args) {
		ApplicationContext ap = new
					ClassPathXmlApplicationContext("my_cnf.xml");
		Employee bean = (Employee) ap.getBean("emp");
		System.out.println(bean);
		bean.m1();
	}
}
```

.xml file:-

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
		http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/context
		http://www.springframework.org/schema/context/spring-context-
3.0.xsd">

	<bean id="emp" class="org.jsp.j2eecontainer.Employee">
		<!-- used for setters method -->
		<!-- <property name="id" value="201"></property>
		<property name="name" value="Ram"></property> -->
		<!-- used for constructor argument -->
		<constructor-arg value="201"></constructor-arg>
		<constructor-arg value="Shyam"></constructor-arg>
	</bean>
</beans>
```

★ @Component :-

- It is class level annotation.
- It will help us to make a class as bean class.
- Spring container will create object of only those classes which are annotated with **@Component** annotation.
- E.g.

```
package org.jsp.j2eecontainer;

import org.springframework.stereotype.Component;

@Component
public class Laptop {

 public Laptop() {
      System.out.println("Hi I am laptop");
 }

}
```

★ @Configuration :-

- It is a class level annotation.
- It will help us to make a class as configuration class.
- E.g.

```
package org.jsp.j2eecontainer;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "org.jsp.j2eecontainer")
public class LaptopConfig {

}
```

★ @ComponentScan :-

- It is class level annotation.
- It will help us to specify package name.
- Spring container will only scan those packages which are mentioned inside a **@ComponentScan** annotation.
- We can provide multiple packages inside @ComponentScan annotation which are separated by comma (**,**).

★ **How to specify multiple packages name inside @ComponentScan annotation?**

```
package org.jsp.j2eecontainer;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "org.jsp.j2eecontainer,org.jsp.j2eecontainer1")
public class LaptopConfig {


}
```

★ **@Value :-**

- It will help us to set a value to variable, to setter() method and constructors.
- It is a variable level, method level and constructor level annotation.
- It will help us to inject a value to the bean class by spring container dynamically.

★ **How to give value to the variable by using @Value annotation on top of the variable?**

```
package org.jsp.j2eecontainer;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component
public class Laptop {

   @Value(value = "101")
   int id;

   @Value(value = "Lenovo")
   String name;

   @Override
   public String toString() {
        return "Laptop [id=" + id + ", name=" + name + "]";
   }


}
```

★ How to give value to the variable by using @Value annotation in to the constructor?

```
package org.jsp.j2eecontainer;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component
public class Laptop {

   int id;
   String name;
   public Laptop(@Value(value = "101") int id,
                 @Value(value = "Dell")String name) {
         this.id = id;
         this.name = name;
   }

}
```

★ How to give value to the variable by using @Value annotation using setter() method?

```
package org.jsp.j2eecontainer;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component
public class Laptop {

   private int id;
   private String name;

   public int getId() {
         return id;
   }

   @Value(value = "101")
   public void setId(int id) {
         this.id = id;
   }

   public String getName() {
         return name;
   }

   @Value(value = "Lenovo")
   public void setName(String name) {
         this.name = name;
   }
}
```

★ How to pass bean.class as a argument inside a getBean() method ?

```
package org.jsp.j2eecontainer;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class LaptopDriver {

	public static void main(String[] args) {

		ApplicationContext ac = new
				AnnotationConfigApplicationContext(LaptopConfig.class);
		Laptop lap = ac.getBean(Laptop.class);
		System.out.println(lap);

	}

}
```

★ Dependency Injection :-

- It is a process of injecting value or object to bean class by spring container is known as dependency injection.
- We can achieve dependency injection with help of @Value and @Autowired annotation.

★ @Autowired :-

- @Autowired annotation will help us to inject object to bean class by spring container.
- Has-a-relationship is mandatory.
- @Autowired is a constructor level, variable level, method level annotation.
- There are three types of dependency injection,
  i. Variable dependency injection
  ii. Constructor dependency injection
  iii. Setter dependency injection

i. Variable dependency injection :-

- E.g.

```
package org.jsp.injection;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Student {

 @Autowired
 Mobile mobile;

 public void use() {
       System.out.println("Student started using mobile....");
       mobile.chat();
 }

}
```

```
package org.jsp.injection;

import org.springframework.stereotype.Component;

@Component
public class Mobile {

 public void chat() {
       System.out.println("Chatting is started.....");
 }

}
```

- We can use variable injection inside a small application like payment service.
- Payment service application is communicating with payment gateway, e.g. RazerPay.
- In order to communicate with payment gateway we need to inject objects, API key and URL's dynamically.
- For this type of application we use variable injection.

ii. Constructor Dependency Injection :-

- When one object is depending on another object strongly then we will go for constructor dependency injection.
- When bean classes are having only one constructor then it is optional to use **@Autowired** annotation.
- When bean classes are having a multiple constructor then it is mandatory to use **@Autowired** annotation.
- E.g.

```java
package org.jsp.beanClasses;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Car {

	Engine engine;

	public Car() {

	}

	@Autowired
	public Car(Engine engine) {
		super();
		this.engine = engine;
	}

	public void move() {
		engine.start();
		System.out.println("Car is started moving....");
	}

}
```

```java
package org.jsp.beanClasses;

import org.springframework.stereotype.Component;

@Component
public class Engine {

	public void start() {
		System.out.println("Engine is started....");
	}

}
```

iii. Setter dependency injection :-

- When dependency injection is optional for the objects then we will go for setter dependency injection.
- When we are having only one setter method then it is optional to use **@Autowired** annotation.
- When bean classes are having multiple setters method the it is mandatory to use **@Autowired** annotation.
- E.g.

```
package org.jsp.beanClasses;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Car {

 GpsDevice gps;

 @Autowired
 public void setGps(GpsDevice gps) {
      this.gps = gps;
 }


 public void showCarLocation() {
      if(gps==null) {
           System.out.println("GPS is not installed.");
      }
      else {
           gps.getLocation();
      }
 }


}
```

```
package org.jsp.beanClasses;

import org.springframework.stereotype.Component;

@Component
public class GpsDevice {

 public void getLocation() {
      System.out.println("Deccan");
 }

}
```

★ @Bean :-

- It will help us to create an object of predefined class.
- It is a method level annotation.
- E.g.

```java
package org.jsp.predefinedClassInjection;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class DoOperation {

 @Autowired
 String s;

 public void operation() {
      System.out.println("length of string is : "+s.length());
 }

}
```

```java
package org.jsp.predefinedClassInjection;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "org.jsp.predefinedClassInjection")
public class MyConfig {

 @Bean
 public String createObject() {
      return new String("hi");
 }

}
```

★ Spring Life Cycle :-

- Spring life cycle will be managed / followed by spring container.
- The following steps will be followed during the spring life cycle,
  i. Bean instantiation
  ii. Dependency injection
  iii. Initialization
  iv. Bean is ready to use
  v. Destruction

i. Bean Instantiation :-

- Spring container will create an object of bean class.
- Spring container will search for configuration file.
- From configuration file spring container will get to know that which package it has to scan.
- Spring container will go to particular package and it will search for bean class.
- Once after bean class found spring container will create an object to the bean class.

ii. Dependency injection :-

- Spring container will inject necessary object to the bean class.

iii. Initialization :-

- Once all the properties have been set spring container will call method which is annotated with **@PostConstruct** to perform start up action.
- Inside this step spring container will open database connection, it will open cache and it will also check whether all the properties have been injected properly or not.

iv. Bean is ready to use :-

- After configuring bean class now bean class is ready to perform a task inside an application.

v. Destruction :-

- Once application will get shutdown spring container will call a method which is annotated with **@PreDestroy** annotation to perform cleanup activities.
- It means it will close database connection, it will clear the cache and bean object will be destroyed.

★ Example on spring life cycle :-

Bean Class:-

```
package org.jsp.springcore;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import org.springframework.stereotype.Component;
@Component
public class Employee {
	public Employee() {
		System.out.println("I am employee class constructor.");
	}
	@PostConstruct
	public void init() {
		System.out.println("open database connection");
		System.out.println("open cache");
		System.out.println("check properties properly injected or not");
	}
	@PreDestroy
	public void destroy() {
		System.out.println("close database connection");
		System.out.println("clear cache");
		System.out.println("bean class object destruction");
	}
}
```

Configuration Class:-

```
package org.jsp.springcore;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
@Configuration
@ComponentScan(basePackages = "org.jsp.springcore")
public class EmployeeConfig {

}
```

Driver Class:-

```
package org.jsp.springcore;
import org.springframework.context.ConfigurableApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
public class EmployeeDriver {
	public static void main(String[] args) {
		ConfigurableApplicationContext cac = new
				AnnotationConfigApplicationContext(EmployeeConfig.class);
		Employee b = cac.getBean(Employee.class);
		cac.registerShutdownHook();
	}
}
```

★ ConfigurableApplicationContext :-

- It is a child interface of ApplicationContext.
- It having ApplicationContext properties and some extra methods like **registerShutdownHook()** method, **refresh()** method.

★ @Lazy :-

- It is a class level annotation.
- It will help us to control object creation of Bean class.
- E.g.

```
package org.jsp.springcore;

import org.springframework.context.annotation.Lazy;
import org.springframework.stereotype.Component;

@Component
@Lazy
public class Employee {

 public Employee() {
      System.out.println("I am employee class constructor.");
 }

}
```

★ @Scope :-

- Default scope of spring container is singleton.
- If you want to change default scope of spring container then we will go for **@Scope** annotation.
- It is a class level annotation.
- Spring container will provide different different scope those are,
  i. Singleton,
  ii. Prototype
- E.g.
  For singleton (if we are not giving singleton by default it will take singleton)

```
package org.jsp.springcore;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
@Scope(value="singleton")
public class Employee {

 public Employee() {
      System.out.println("I am employee class constructor.");
 }

}
```

```
package org.jsp.springcore;

import org.springframework.context.ConfigurableApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class EmployeeDriver {

 public static void main(String[] args) {

       ConfigurableApplicationContext cac = new
                  AnnotationConfigApplicationContext(EmployeeConfig.class);

       Employee emp1 = cac.getBean(Employee.class);
       System.out.println(emp1);

       Employee emp2 = cac.getBean(Employee.class);
       System.out.println(emp2);

 }

}
```

- For prototype example :-

```
package org.jsp.springcore;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
@Scope(value="prototype")
public class Employee {
 public Employee() {
       System.out.println("I am employee class constructor.");
 }
}
```

```
package org.jsp.springcore;

import org.springframework.context.ConfigurableApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class EmployeeDriver {
 public static void main(String[] args) {
       ConfigurableApplicationContext cac = new
                   AnnotationConfigApplicationContext(EmployeeConfig.class);
       Employee emp1 = cac.getBean(Employee.class);
       System.out.println(emp1);
       Employee emp2 = cac.getBean(Employee.class);
       System.out.println(emp2);
 }
}
```

★ When we will get NoUniqueBeanDefinitionException?

- Consider there is a bean class i.e. Car.
- Now Car is depending on Engine.
- We need to inform spring container to inject Engine object inside a Car.
- Engine is an interface, but it is having more than one child implementing classes.
- In this situation Engine child implementing classes are eligible to create an object but spring container is expecting unique bean but it found more than one.
- In this scenario spring container is getting confusion or ambiguity.
- Then it will throw an exception i.e. NoUniqueBeanDefinitionException.

★ How to overcome NoUniqueBeanDefinitionException?

- To overcome this exception spring will provide **@Primary** and **@Qualifier** annotations.
- @Primary annotation will help us to mark bean as default choice when there is no @Qualifier candidates.
- @Qualifier annotation will help us to spring container exactly or specifically which bean class object it has to inject.
- When we are using @Primary and @Qualifier then spring container will always give priority to @Qualifier because @Qualifier is having higher priority than @Primary.

Program for @Primary & @Qualifier:-

Car Class (Bean class):-

```
package org.jsp.springcore1;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;
@Component
public class Car {
	@Autowired
	@Qualifier(value = "PE")
	Engine engine;
	public void start() {
		System.out.println("Car is started.");
		engine.hp();
	}
}
```

Engine Interface:-

```
package org.jsp.springcore1;

public interface Engine {

	void hp();

}
```

DieselEngine Class (Bean class & also child implementing class of Engine):-

```
package org.jsp.springcore1;

import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Component;
@Component
@Primary
public class DieselEngine implements Engine{

	@Override
	public void hp() {
		System.out.println("Diesel Engine hp is 2000");
	}

}
```

PetrolEngine Class (Bean class & also child implementing class of Engine):-

```
package org.jsp.springcore1;

import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Component;
@Component(value = "PE")
@Primary
public class PetrolEngine implements Engine{

	@Override
	public void hp() {
		System.out.println("Petrol engine hp is 1500");
	}

}
```

Driver Class:-

```
package org.jsp.driver;

import org.jsp.configurationClasses.Myconfig;
import org.jsp.springcore1.Car;
import org.springframework.context.ConfigurableApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Driver {
	public static void main(String[] args) {
		ConfigurableApplicationContext cac = new
				AnnotationConfigApplicationContext(Myconfig.class);

		Car c = cac.getBean(Car.class);
		c.start();
	}
}
```

★ Annotations which are used in spring core :-

Annotations		Description
1.	@Component	• Marks a Java class as a Spring component (bean). It tells Spring to manage the lifecycle of this class and register it in the application context.
2.	@Configuration	• Indicates that the class contains Spring bean definitions. Used in Java-based configuration classes.
3.	@ComponentScan	• Tells Spring where to look for components (@Component, @Service, @Repository, etc.). Commonly used with @Configuration.
4.	@Value	• Injects values into fields from properties files or environment variables.
5.	@Autowired	• Automatically injects dependencies (beans) by type into fields, constructors, or methods.
6.	@Bean	• Declares a bean to be managed by Spring, used inside @Configuration classes to define beans manually.
7.	@Lazy	• Delays the bean initialization until it's actually needed (lazy loading).
8.	@Scope	• Defines the scope of a bean (singleton, prototype, request, session, etc.), e.g., @Scope("prototype").
9.	@PostConstruct	• Marks a method to be run once after dependency injection is done, typically used for initialization.
10.	@PreDestroy	• Marks a method to be called just before the bean is destroyed, typically used for cleanup.
11.	@Primary	• Specifies a bean as the primary candidate when multiple beans of the same type are available.
12.	@Qualifier	• Used with @Autowired to resolve ambiguity when multiple beans of the same type exist, e.g., @Qualifier("myBean").