SRINIVAS UNIVERSITY
INSTITUTE OF COMPUTER SCIENCE AND INFORMATION SCIENCES
Second Semester MCA Examination
Subject Name: Advanced Java

Question Bank: Part - A (5x1=5) Part – B (5x7=35) Part – C (1x10=10) Total Marks:50

Unit-I: JDBC: Introduction to JDBC, JDBC Driver types, JDBC database connections, JDBC Statements, PreparedStatement, CallableStatement, ResultSet, JDBC data types, transactions, Batch Processing, Stored Procedure

Part - A

1. What is Core Java?
2. What is Advanced Java?
3. Define JVM.
4. What is full form of JDBC?
5. Thin JDBC driver is called as _________________
6. What is JDBC Statements?
7. Define batch processing.
8. What is Stored Procedure in SQL?

Part - B

1. Explain the difference between Core java vs Advanced java
2. What are the Advanced Java applications?
3. Define JDBC Concept with example.
4. Explain the JDBC driver types with example.
5. Explain the JDBC data types.
6. Explain the properties of transaction managements
7. Explain the Advantage of Transaction Managements
8. Describe the JDBC Statement, CallableStatement, and PreparedStatement
9. Explain the Batch Processing in JDBC with example.

Part - C

1. Explain the components of JDBC with neat diagram
2. Write a java program to access the MySql database
3. Explain the steps to create a JDBC Database Connection

Unit-2: Servlet: Servlet structure, Life Cycle of a Servlet, Using Tomcat for Servlet Development, The Servlet API, Handling Client Request: Form data, Handling client HTTP request and server HTTP Response, HTTP status codes, Handling Cookies, Session tracking, Database Access

Part - A

1. What is servlet?
2. List out the two package used for servlet creation
3. Why use the session tracking in web application?
4. What is HTTP?
5. What is RMI?
6. What is CORBA?
7. What is XML?
8. What is CGI?
9. What is Web server?
10. What is Browser?
11. What is Cookies?

Part - B

1. What are the advantages of servlets?
2. Explain the Servlets perform of major tasks
3. Explain the life cycle of a Servlet.
4. Explain the Servlet directory structure
5. Explain types of session techniques.
6. Explain the Database Access in servlet

Part - C

1. Explain the javax.servlet and javax.servlet.http packages of importance in web application
2. Write a Servlet program to link with HTML program
3. Explain benefits of Using Cookies in Servlets
4. Write a Servlet program to display the how many times visited the website counting using with Cookies

Unit-3: JSP: Overview of JSP Technology, Need of JSP, Advantages of JSP, Life Cycle of JSP Page, JSP Processing, JSP Application Design with MVC, Setting Up the JSP Environment, JSP Directives, JSP Action, JSP Implicit Objects, JSP Form Processing, JSP Session and Cookies Handling, JSP Session Tracking JSP Database Access, JSP Standard Tag Libraries, JSP Custom Tag, JSP Expression Language, JSP Exception Handling

Part - A

1. What is JSP?
2. What is MVC?
3. What is ASP?
4. What is Model Layer?
5. What is View Layer?
6. What is Controller Layer?
7. What is Implicit Objects?

Part - B

1. What are the advantages of JSP?
2. Explain the life cycle of a JSP.
3. Explain the JSP processing.
4. Explain types of JSP Implicit Objects
5. Explain the JSP form processing.
6. Explain the JSP Database access with example
7. Define JSP Standard Tag Libraries and Custom Tag
8. Define JSP expression Language
9. Define JSP Exception handling with example

Part - C

1. What are the steps to execute JSP page?
2. Explain the MVC Architecture in JSP.
3. Explain the JSP Session and Cookies Handling in JSP

Unit -4: Hibernate Introduction, Hibernate Configuration, Hibernate Concepts, Hibernate O-R Mapping, Manipulating and Querying, Hibernate Query Language, Criteria Queries, Native SQL, Transaction and Concurrency

Part - A

1. What is Hibernate?
2. What is ORM?
3. What is HQL?
4. Define advantage of HQL
5. What is Criteria queries?
6. Define Native SQL.
7. What is Java Persistence?

Part - B

1. Write short notes about the Hibernate concepts.
2. Explain the Hibernate O-R mapping process.
3. What are the Hibernate Advantages?
4. Explain the Hibernate Architecture
5. Explain the advantage of using native SQL queries in Hibernate
6. Explain the Transaction Interface in Hibernate

Part - C

1. Explain the Hibernate configuration approaches details.
2. How do create web application using hibernate?
3. Explain the Hibernate Mapping Types

UNIT- 5 Spring Framework: Spring Basics, Spring Container, Spring AOP, Spring Data Access, Spring O-R/mapping, Spring Transaction Management, Spring Remoting and Enterprise Services, Spring Web MVC Framework, Securing Spring Application.

Part - A

1. What is spring?
2. What is POJO?
3. What is IoC?
4. What is AOP?
5. What is Spring Security?
6. What is java beans?
7. What is Dependency Injection?

Part - B

1. Explain the application of Spring
2. Write short notes about the AOP Terminologies.
3. Write short notes about the Spring ORM technique.
4. Explain the Programmatic vs Declarative Transactions

5. Describe the Spring Remoting technologies
6. Explain the Spring Enterprise Services.
7. Explain the Spring Web MVC Framework design
8. What are the benefits of using Spring Security Application?

Part - C

1. Explain the Spring Framework components.
2. Explain the details of Spring Container types

Question and Answer keys:

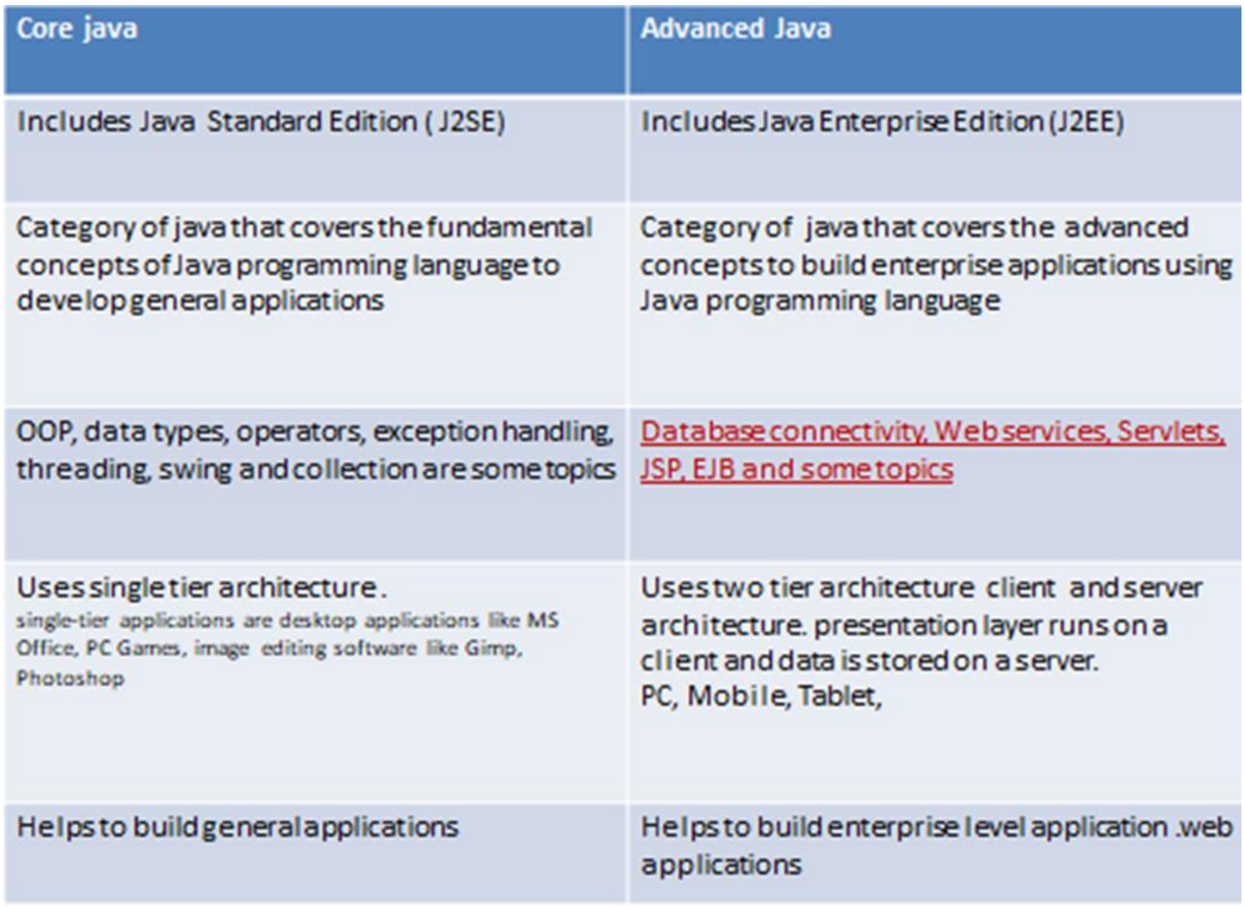
Unit-I: JDBC: Introduction to JDBC, JDBC Driver types, JDBC database connections, JDBC Statements, PreparedStatement, CallableStatement, ResultSet, JDBC data types, transactions, Batch Processing, Stored Procedure

Part - A

1. **What is Core Java?**
   Core Java covers the fundamental concepts in the Java programming language.**(J2SE)**
2. **What is Advanced Java?**
   Advanced Java is build enterprise level applications. Advance Java i.e. JEE (Java Enterprise Edition) and Client-Server architecture for Web Application Development**(J2EE)**
3. **Define JVM.**
   JVM provides a runtime environment for driving Java applications or code. JVM is an abstract machine that converts the Java bytecode into a machine language. It is also capable of running the programs written by programmers in other languages (compiled to the Java bytecode)
4. **What is full form of JDBC**?
   Java Database Connectivity (JDBC) is an application programming interface (API) for the Java programming
5. **Thin JDBC driver is called as _________________**
   Type-4 JDBC driver also known as 'thin driver' or Direct to Database Pure Java Driver. It is portable, the fastest among all JDBC drivers and database dependent. The thin driver converts JDBC calls directly into the vendor-specific database protocol.
6. **What is JDBC Statements?**
   A JDBC statement is used to execute the different queries of the database. JDBC statement is a bunch of ResultSet, and we can use it as a method to get the desired object of the ResultSet.
7. **Define batch processing.**
   A Batch Processing allows you to group related SQL statements into a batch and submit them with one call to the database. When you send several SQL statements to the database at once, you reduce the amount of communication overhead, thereby improving performance. Or **Batch processing groups multiple queries into one unit and passes it in a single network trip to a database.**
8. What is Stored Procedure in SQL?
   A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again. an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it. It can also **pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.**

Part - B

1. Explain the difference between Core java vs Advanced java



Core java	Advanced Java
Includes Java Standard Edition (J2SE)	Includes Java Enterprise Edition (J2EE)
Category of java that covers the fundamental concepts of Java programming language to develop general applications	Category of java that covers the advanced concepts to build enterprise applications using Java programming language
OOP, data types, operators, exception handling, threading, swing and collection are some topics	Database connectivity, Web services, Servlets, JSP, EJB and some topics
Uses single tier architecture . single-tier applications are desktop applications like MS Office, PC Games, image editing software like Gimp, Photoshop	Uses two tier architecture client and server architecture. presentation layer runs on a client and data is stored on a server. PC, Mobile, Tablet,
Helps to build general applications	Helps to build enterprise level application .web applications

2. What are the Advanced Java applications?

There are a wide range of applications for advanced Java. Typically, programmers use it for web and network-focused applications and databases. Some of its applications include:
Mobile: Java is popular with mobile app developers because of its compatibility range.
Graphical user interfaces (GUIs): When developing GUIs within corporate networks, programmers often use advanced Java .
Web: Advanced Java is a popular choice for web applications, as it's easy to use and has a high level of security.
Enterprise: Developers of enterprise applications, such as banking applications, often use advanced Java because of its advantageous runtime environment and compatibility with web services.
Scientific: Advanced Java is a popular choice for developers for coding mathematical and scientific calculations.
Gaming: Game developers often use advanced Java for designing 3D games.
Big data: Databases commonly use advanced Java to help organize large volumes of information.
Distributed applications: Developers frequently use Java for distributed applications because of its persistent and dynamic nature.
Cloud-based applications: Java is a popular choice for cloud-based applications, as it's compatible with software as a service (SaaS) and similar applications for platforms (PaaS) and infrastructure (IaaS).

3. Define JDBC Concept with example.

Definition of JDBC (Java Database Connectivity) JDBC is an API(Application programming interface) used in java programming to interact with databases. The classes and interfaces of JDBC allow the application to send requests made by users to the specified database.

Purpose of JDBC

Enterprise applications created using the JAVA EE technology need to interact with databases to store application-specific information. So, interacting with a database requires efficient database connectivity, which can be achieved by using the ODBC(Open database connect ivity) driver. This driver is used with JDBC to interact or communicate with various kinds of databases such as Oracle, MS Access, Mysql, and SQL server database.

Components of JDBC : There are generally four main components of JDBC through which it can interact with a database. They are as mentioned below:

1. JDBC API: It provides various methods and interfaces for easy communication with the database. It provides two packages as follows, which contain the java SE and Java EE platforms to exhibit WORA(write once run anywhere) capabilities.

```
java.sql.*;
```

It also provides a standard to connect a database to a client application.

2. JDBC Driver manager: It loads a database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.
3. JDBC Test suite: It is used to test the operation(such as insertion, deletion, updation) being performed by JDBC Drivers.
4. JDBC-ODBC Bridge Drivers: It connects database drivers to the database. This bridge translates the JDBC method call to the ODBC function call. It makes use of the sun.jdbc.odbc package which includes a native library to access ODBC characteristics.

4. Explain the JDBC driver types with example.

JDBC Driver Types with example.
JDBC Driver is a software component that enables java application to interact with the database.
There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver. In Java 8, the JDBC-ODBC Bridge has been removed.Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

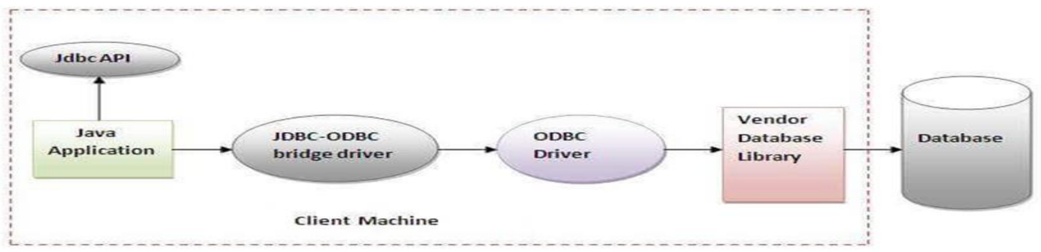


Figure- JDBC-ODBC Bridge Driver

2) Native-API driver
The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

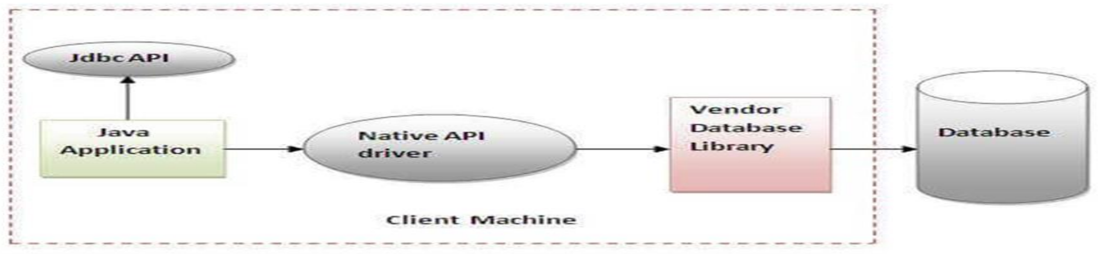


Figure- Native API Driver

3) Network Protocol driver
The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

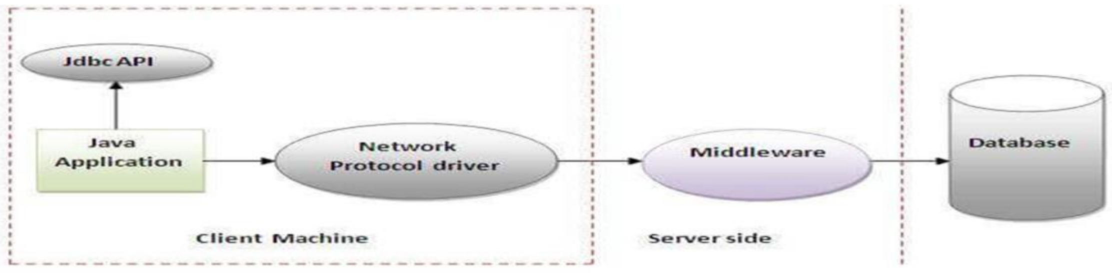


Figure- Network Protocol Driver

4) Thin driver
The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

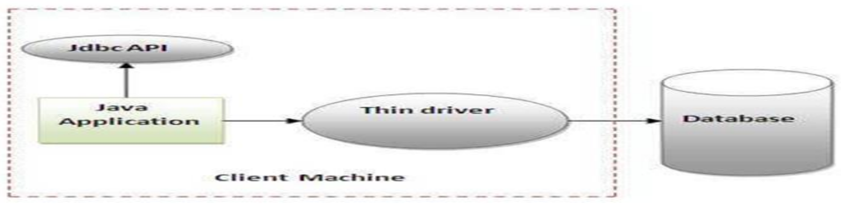


Figure- Thin Driver

5. Explain the JDBC data types.

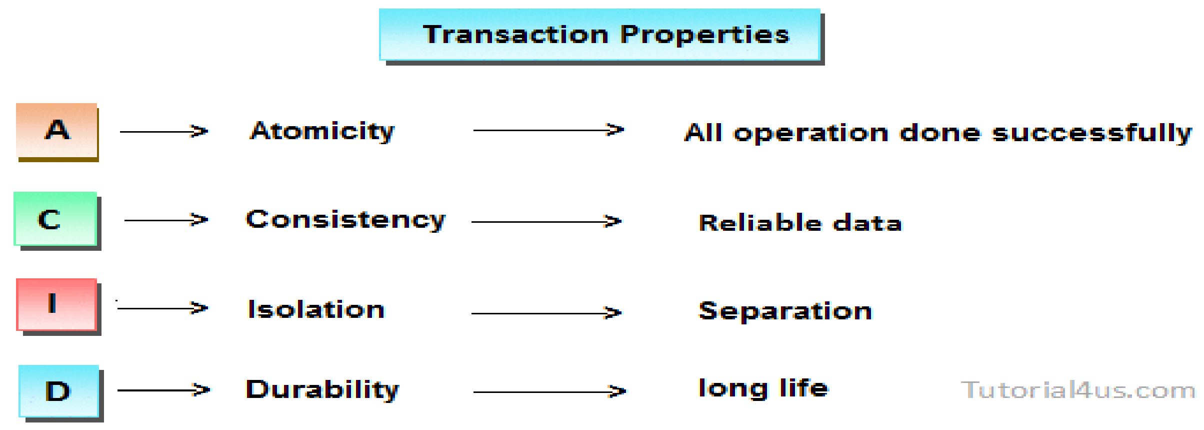
The JDBC driver converts the Java data type to the appropriate JDBC type, before sending it to the database. It uses a default mapping for most data types. For example, a Java int is converted to an SQL INTEGER. Default mappings were created to provide consistency between drivers.

The following table summarizes the default JDBC data type that the Java data type is converted to, when you call the setXXX() method of the PreparedStatement or CallableStatement object or the ResultSet.updateXXX() method. Example

S.NO	SQL	JDBC/Java	setXXX	updateXXX
1	VARCHAR	java.lang.String	setString	updateString
2	CHAR	java.lang.String	setString	updateString
3	LONGVARCHAR	java.lang.String	setString	updateString
4	BIT	boolean	setBoolean	updateBoolean
5	NUMERIC	java.math.BigDecimal	setBigDecimal	updateBigDecimal
6	TINYINT	byte	setByte	updateByte
7	SMALLINT	short	setShort	updateShort
8	INTEGER	int	setInt	updateInt
9	BIGINT	long	setLong	updateLong
10	REAL	float	setFloat	updateFloat
11	FLOAT	float	setFloat	updateFloat

6. **Explain the properties of transaction managements**

Every transaction follows some transaction properties these are called ACID properties



ACID

Atomicity: Atomicity of a transaction is nothing but in a transaction either all operations can be done or all operation can be undone, but some operations are done and some operation are undone should not occure.

Consistency: Consistency means, after a transaction completed with successful, the data in the data store should be a reliable data this reliable data is also called as consistent data.

Isolation: Isolation means, if two transaction are going on same data then one transaction will not disturb another transaction.

Durability: Durability means, after a transaction is completed the data in the data store will be permanent until another transaction is going to be performed on that data.

7. **Explain the Advantage of Transaction Managements**

Advantages: Fast performance It makes the performance fast because database is hit at the time of commit.

Types of Transaction

Local Transaction

Distributed or global transaction

A. Local Transaction

A local transaction means, all operation in a transaction are executed against one database.

For example;

If transfer money from first account to second account belongs to same bank then transaction is local transaction.

B. Global Transaction

A global transaction means, all operations in a transaction are executed against multiple database.

For Example;

If transfer money from first account to second account belongs to different banks then the transaction is a global transaction.

- **Note:** Jdbc technology perform only local transactions. For global transaction in java we need either EJB or spring framework.

Useful Connection Methods (for Transactions

- **getAutoCommit/setAutoCommit**
  - By default, a connection is set to auto-commit
  - Retrieves or sets the auto-commit mode
- **commit**
  - Force all changes since the last call to commit to become permanent
  - Any database locks currently held by this `Connection` object are released
- **rollback**
  - Drops all changes since the previous call to commit
  - Releases any database locks held by this `Connection` object

Main Advantage of Transaction Mangaement: Fast performance It makes the performance fast because database is hit at the time of commit.

8. Describe the JDBC Statement, CallableStatement, and PreparedStatement

The JDBC Statement, CallableStatement, and PreparedStatement interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database. They also define methods that help bridge data type differences between Java and SQL data types used in a database.

The following table provides a summary of each interface's purpose to decide on the interface to use.

Interfaces	Recommended Use
Statement	Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.
PreparedStatement	Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.
CallableStatement	Use this when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

9. Explain the Batch Processing in JDBC with example.

Batch Processing allows you to group related SQL statements into a batch and submit them with one call to the database. When you send several SQL statements to the database at once, you reduce the amount of communication overhead, thereby improving performance.Batch processing in JDBC. It follows following steps:

1. Load the driver class
2. Create Connection
3. Create Statement
4. Add query in the batch
5. Execute Batch
6. Close Connection

1.Load the driver class

Class.forName()

An efficient way to load the JDBC driver is to invoke the Class. forName(). newInstance() method, specifying the name of the driver class, as in the following example: Class.

The class loading process triggers a static initialization routine that registers the driver instance with the DriverManager and associates this class with the database engine identifier, such as oracle or postgres. After the registration is complete, we can use this identifier inside the JDBC URL as jdbc:oracle.

2.Create Connection

Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

JDBC makes it possible to establish a connection with a data source, send queries and update statements, and process the results . Simply, JDBC makes it possible to do the following things within a Java application: Establish a connection with a data source. Send queries and update statements to the data source.

The JDBC Connection class, java. sql. Connection, represents a database connection to a relational database .Before you can read or write data from and to a database via JDBC, you need to open a connection to the database

3.Create a Statement

Create a Statement: From the connection interface, you can create the object for this interface . It is generally used for general-purpose access to databases and is useful while using static SQL statements at runtime. Syntax: Statement statement = connection.

The JDBC Statement, CallableStatement, and PreparedStatement interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database. They also define methods that help bridge data type differences between Java and SQL data types used in a database.

4.Add query in the batch

The addBatch() method of Statement, PreparedStatement, and CallableStatement is used to add individual statements to the batch .

The executeBatch() is used to start the execution of all the statements grouped together.

Batch is a group of SQL statements that are executed at one time by SQL Server. These statements are sent to SQL Server by a program, such as the Query Analyzer. The opposite of a batch query is a single query, containing only one SQL statement.

5 and 6 Execute Batch and Close Connection

int[] executeBatch() The executeBatch() method begins the execution of all the statements grouped together . The method returns an integer array, and each element of the array represents the updated count for the respective update statement.

Part - C

1. Explain the components of JDBC with neat diagram

Four main components of JDBC through which it can interact with a database.

1. JDBC API: It provides various methods and interfaces for easy communication with the database. It provides two packages as follows, which contain the java SE and Java EE platforms to exhibit WORA(write once run anywhere) capabilities.

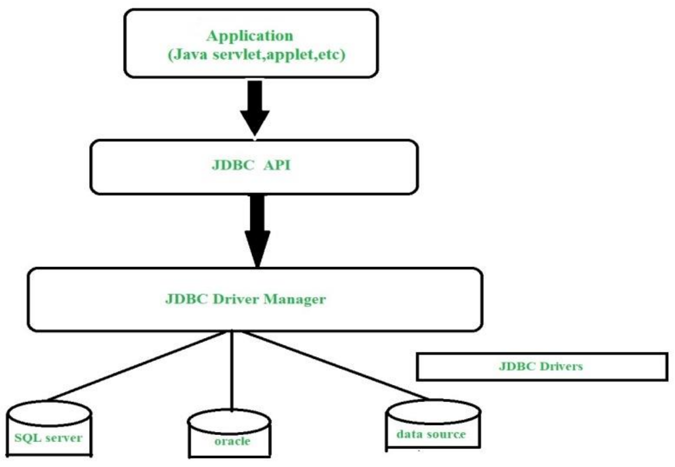
java.sql.*;

It also provides a standard to connect a database to a client application.

2. JDBC Driver manager: It loads a database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.

3. JDBC Test suite: It is used to test the operation(such as insertion, deletion, updation) being performed by JDBC Drivers.

4. JDBC-ODBC Bridge Drivers: It connects database drivers to the database. This bridge translates the JDBC method call to the ODBC function call. It makes use of the sun.jdbc.odbc package which includes a native library to access ODBC characteristics.



Description:

Application: It is a java applet or a servlet that communicates with a data source.

The JDBC API: The JDBC API allows Java programs to execute SQL statements and retrieve results. Some of the important classes and interfaces defined in JDBC API are as follows:

DriverManager: It plays an important role in the JDBC architecture. It uses some database-specific drivers to effectively connect enterprise applications to databases.

JDBC drivers: To communicate with a data source through JDBC, you need a JDBC driver that intelligently communicates with the respective data source.

2. Write a java program to access the MySql database

Create a Java JDBC program to access the Mysql database:

Step 1: Data base creation

```
Mysql  database:
mysql> use college
Database changed
mysql> desc mca;
Mysql>create table  mca(S_id  int(5)primary key,Sname varchar(20),DOB  date,Address
varchar(20),Email_id varchar(20));
Mysql>insert into mca  values(1001,"Raja",'2023-07-09',"Chennai","ss@gmail.com");
Mysql>insert into mca  values(1001,"John",'2023-07-10',"Mangalore","vv@gmail.com");

mysql> select * from mca;
+------+-------+------------+-----------+--------------+
| S_id | Sname | DOB        | Address   | Email_id     |
+------+-------+------------+-----------+--------------+
| 1001 | Raja  | 2023-07-09 | Chennai   | ss@gmail.com |
| 1002 | John  | 2023-07-10 | Mangalore | vv@yahoo.com |
+------+-------+------------+-----------+--------------+
2 rows in set (0.00 sec)
```

Step 2: Mysql database with JDBC program connection code

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class domo {

    public static void main(String args[]) {
        try {
            Connection con = (Connection)
DriverManager.getConnection("jdbc:mysql://localhost:3306/college", "root", "root");
            Statement stnt = con.createStatement();
            String query = "select*from mca";
            ResultSet rs = stnt.executeQuery(query);
            while (rs.next()) {
                for (int i=1;i<=5;i++){

                    System.out.print(rs.getString(i));
                    System.out.println("|");
                }
                System.out.println();

            }
        }
        catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

```
Output:
run:
1001|
Raja|
2023-07-09|
Chennai|
ss@gmail.com|
1002|
John|
2023-07-10|
Mangalore|
vv@yahoo.com|
BUILD SUCCESSFUL (total time: 0 seconds)
```

Result: Thus program has been successfully executed.

3. Explain the steps to create a JDBC Database Connection

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows: 5 Steps to connect to the database in java

1. Register the driver class
2. Create the connection object
3. Create the Statement object
4. Execute the query
5. Close the connection object

1) Register the driver class

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

```
public static void forName(String className)throws ClassNotFoundException
```

Note: Since JDBC 4.0, explicitly registering the driver is optional. We just need to put vender's Jar in the classpath, and then JDBC driver manager can detect and load the driver automatically.

Example to register the OracleDriver class

Here, Java program is loading oracle driver to esteblish database connection.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2) Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

1) public static Connection getConnection(String url)throws SQLException

2) public static Connection getConnection(String url,String name,String password) throws SQLException

Example to establish connection with the Oracle database

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","password");

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

public Statement createStatement()throws SQLException

Example to create the statement object

Statement stmt=con.createStatement();

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database.

This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

public ResultSet executeQuery(String sql)throws SQLException

Example to execute query

```
ResultSet rs=stmt.executeQuery("select * from emp");

 while(rs.next()){

System.out.println(rs.getInt(1)+" "+rs.getString(2));

}
```

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

public void close()throws SQLException

Example to close connection

con.close();

Note: Since Java 7, JDBC has ability to use try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement.

Unit-2: Servlet: Servlet structure, Life Cycle of a Servlet, Using Tomcat for Servlet Development, The Servlet API, Handling Client Request: Form data, Handling client HTTP request and server HTTP Response, HTTP status codes, Handling Cookies, Session tracking, Database Access

Part - A

1. **What is servlet?**

A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.

2. **List out the two package used for servlet creation**

There are two main packages which this chapter makes use of: **javax. servlet** and javax. servlet. http. The javax.servlet package provides all of the base classes and interfaces defining both what a Servlet is, and how it interacts with the Web server that is running it.The javax.servlet.http package provides classes specific to handling HTTP requests. It provides the HttpServlet class used in this chapter, which implements the appropriate interfaces from javax.servlet.

3. **Why use the session tracking in web application?**

To recognize the user It is used to recognize the particular user. Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.

4. **What is HTTP?**

Hypertext Transfer Protocol (HTTP) is a method for encoding and transporting information between a client (such as a web browser) and a web server. HTTP is the primary protocol for transmission of information across the Internet

5. **What is RMI?**

Remote Method Invocation (RMI) is a Java technology in which an object running in Java Virtual Machine (JVM) could be invoked from another object running in a different JVM. The technology provides a remote access of objects in Java programming language. The RMI technology consists of a server and a client.

6. **What is CORBA?**

CORBA stands for Common Object Request Broker Architecture. The original idea was to create a single universal standard for how objects across different platforms, programming languages, and network protocols can communicate with each other in a seamless manner.

7. **What is XML?**

Extensible Markup Language (XML) lets you define and store data in a shareable manner. XML supports information exchange between computer systems such as websites, databases, and third-party applications.

8. **What is CGI?**

The Common Gateway Interface (CGI) standard is a data-passing specification used when a Web server must send or receive data from an application such as a database. A CGI script passes the request from the Web server to a database, gets the output and returns it to the Web client.

9. **What is Web server?**

A web server is software and hardware that uses HTTP (Hypertext Transfer Protocol) and other protocols to respond to client requests made over the World Wide Web. The main job of a web server is to display website content through storing, processing and delivering webpages to users.

10. What is Browser?

A software application used to access information on the World Wide Web is called a Web Browser. When a user requests some information, the web browser fetches the data from a web server and then displays the webpage on the user's screen.

11. What is Cookies?

Cookies help that website remember information about your visit, which can both make it easier to visit the site again and make the site more useful to you. Other technologies, including unique identifiers used to identify a browser, app or device, pixels, and local storage, can also be used for these purposes.

Part - B

1. What are the advantages of servlets?

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. But Servlets offer several advantages in comparison with the CGI.

 Performance is significantly better.
 Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
 Servlets are platform-independent because they are written in Java.
 Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So Servlets are trusted.
 The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already

Example:

1. Servlets are platform-independent, as they follow the Java standard.
2. Servlets provide easy dynamic response generation, allowing them to create dynamic web pages.
3. Servlets provide an efficient way to manage session information, which can be stored on the server.
4. Servlets can handle multiple requests simultaneously, improving the scalability of the system.

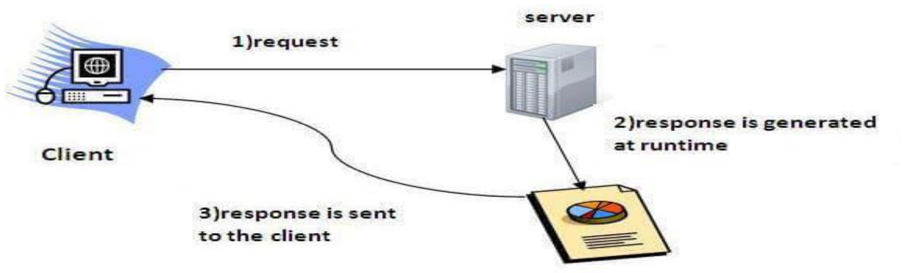
2. Explain the Servlets perform of major tasks

Servlets perform the following major tasks:

Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.

 Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
 Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
 Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
 Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks. Example figure below

show the servlet tasks


server
1)request
Client
2)response is generated at runtime
3)response is sent to the client

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.

3. Explain the life cycle of a Servlet.

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet The servlet is initialized by calling the init () method. The servlet calls service() method to process a client's request. The servlet is terminated by calling the destroy() method. Finally, servlet is garbage collected by the garbage collector of the JVM.
Now let us discuss the life cycle methods in details.

The init() method :

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets. The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started. When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.
The init method definition
looks like this:

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

The service() method:

The service() method is the main method to perform the actual task. The servlet container(i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client. Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.
Here is the signature of this method:

```
public void service(ServletRequest request,
ServletResponse response)
throws ServletException, IOException{
```

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client. The doGet() and doPost() are most frequently used methods within each service request. Here is the signature of these two methods

The doGet() Method
A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
// Servlet code
}
```

The doPost() Method
A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
// Servlet code
}
```

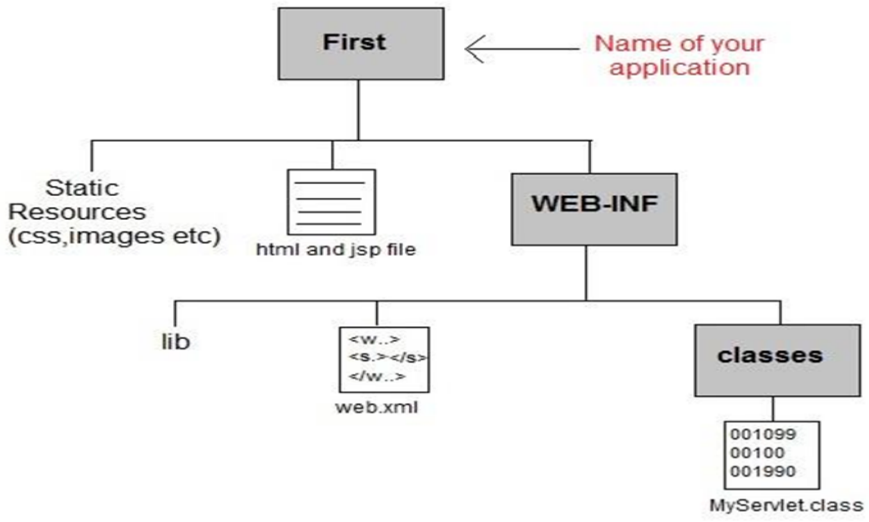
The destroy() method:
The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

```
public void destroy() {
// Finalization code...
}
```

4. Explain the Servlet directory structure

Sun Microsystem defines a unique directory structure that must be followed to create a servlet application. Create the above directory structure inside Apache-Tomcat\webapps directory. All HTML, static files(images, css etc) are kept directly under Web application folder. While all the Servlet classes are kept inside classesfolder.

The web.xml(deployement descriptor) file is kept under WEB-INFfolder.



Servlet directory structure

The Root Directory: The root directory of you web application can have any name. In the above example the root directory name is mywebapp .Under the root directory, you can put all files that should be accessible in your web application.
The WEB-INF Directory : The WEB-INF directory is located just below the web app root directory. This directory is a meta information directory. Files stored here are not supposed to be accessible from a browser (although your web app can access them internally, in your code).
web.xml : The web.xml file contains information about the web application, which is used by the Java web server / servlet container in order to properly deploy and execute the web application.
classes Directory : The classes directory contains all compiled Java classes that are part of your web application. The classes should be located in a directory structure matching their package structure.
lib folder : The lib directory contains all JAR files used by your web application. This directory most often contains any third party libraries that your application is using.

5. Explain types of session techniques.

Basically there are four techniques which can be used to identify a user session.
1. Cookies, 2. Hidden Fields,3. URL Rewriting,4. Session Tracking API
Cookies , Hidden Fields and URL rewriting involves sending a unique identifier with each request and servlets determines the user session based on the identifier. Session API uses the other three techniques internally and provides a session tracking in much convenient and stable way.
1.Cookies Cookie is a key value pair of information, sent by the server to the browser and then browser sends back this identifier to the server with every request there on.

There are two types of cookies:

Session cookies - are temporary cookies and are deleted as soon as user closes the browser. The next time user visits the same website, server will treat it as a new client as cookies are already deleted.

Persistent cookies - remains on hard drive until we delete them or they expire.

If cookie is associated with the client request, server will associate it with corresponding user session otherwise will create a new unique cookie and send back with response.

Simple code snippet to create a cookie with name sessionId with a unique value for each client:

Cookie cookie = new Cookie("sessionID", "some unique value");
response.addCookie(cookie);

User can disable cookie support in a browser and in that case server will not be able to identify the user so this is the major disadvantage of this approach.

2.Hidden Field: Hidden fields are the input fields which are not displayed on the page but its value is sent to the servlet as other input fields. For example
<input type=”hidden” name=”sessionId” value=”unique value”/>
is a hidden form field which will not displayed to the user but its value will be send to the server and can be retrieved using **request.getParameter(“sessionId”)** in servlet.

As we cannot hardcode the value of hidden field created for session tracking purpose, which means we cannot use this approach for static pages like HTML. In short with this approach, HTML pages cannot participate in session tracking with this approach.Another example will be any get requests like clicking of any link so above two are the major disadvantages of this approach.
3.URL Rewriting: URL Rewriting is the approach in which a session (unique) identifier gets appended with each request URL so server can identify the user session. For example if we apply URL **rewriting on http://localhost:8080/HelloWorld/SourceServlet , it will become something like http://localhost:8080/HelloWorld/SourceServlet?jSessionId=XYZ where jSessionId=XYZ is the attached session identifier and value XYZ will be used by server to identify the user session.**

There are several advantages of URL rewriting over above discussed approaches like it is browser independent and even if user’s browser does not support cookie or in case user has disabled cookies, this approach will work. Another advantage is , we need not to submit extra hidden parameter.As other approaches, this approach also has some disadvantages like we need to regenerate every url to append session identifier and this need to keep track of this identifier until the conversation completes.
4.Session Tracking API: Servlets provide a convenient and stable session-tracking solution using the HttpSession API. This interface is built on the top of above discussed approaches.Session tracking in servlet is very simple and it involves following steps.
Get the associated session object (HttpSession) using request.getSession().
To get the specific value out of session object, call getAttribute(String) on the HttpSession object.
To store any information in a session call setAttribute(key,object) on a session object.
To remove the session data , call removeAttribute(key) to discard a object with a given key.
To invalidate the session, call invalidate() on session object. This is used to logout the logged in user.

6. Explain the Database Access in servlet

Create the table Employee in TEST database as follows −

```
mysql> use TEST;
mysql> create table Employees (
  id int not null,
  age int not null,
  first varchar (255),
  last varchar (255)
);
Query OK, 0 rows affected (0.08 sec)
mysql>
```

Create Data Records
Finally you create few records in Employee table as follows

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
Query OK, 1 row affected (0.05 sec)
mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');
Query OK, 1 row affected (0.00 sec)
 mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
Query OK, 1 row affected (0.00 sec)
 mysql>
```

Accessing a Database :Here is an example which shows how to access TEST database using Servlet.

```
// Loading required libraries
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
 public class DatabaseAccess extends HttpServlet{
   public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
      // JDBC driver name and database URL
      static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
      static final String DB_URL="jdbc:mysql://localhost/TEST";
      //  Database credentials
      static final String USER = "root";
      static final String PASS = "password";
      // Set response content type
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Database Result";
     String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";
            out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n" +
         "<body bgcolor = \"#f0f0f0\">\n" +
         "<h1 align = \"center\">" + title + "</h1>\n");
      try {
         // Register JDBC driver
         Class.forName("com.mysql.jdbc.Driver");
         // Open a connection
         Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
         // Execute SQL query
         Statement stmt = conn.createStatement();
         String sql;
         sql = "SELECT id, first, last, age FROM Employees";
         ResultSet rs = stmt.executeQuery(sql);
         // Extract data from result set
         while(rs.next()){
            //Retrieve by column name
            int id  = rs.getInt("id");
            int age = rs.getInt("age");
            String first = rs.getString("first");
            String last = rs.getString("last");
            //Display values
            out.println("ID: " + id + "<br>");
            out.println(", Age: " + age + "<br>");
            out.println(", First: " + first + "<br>");
            out.println(", Last: " + last + "<br>");
         }
         out.println("</body></html>");
         // Clean-up environment
         rs.close();
         stmt.close();
         conn.close();
      } catch(SQLException se) {
         //Handle errors for JDBC
         se.printStackTrace();
      } catch(Exception e) {
         //Handle errors for Class.forName
         e.printStackTrace();
      } finally {
         //finally block used to close resources
         try {
            if(stmt!=null)
               stmt.close();
         } catch(SQLException se2) {
         } // nothing we can do
         try {
            if(conn!=null)
            conn.close();
         } catch(SQLException se) {
            se.printStackTrace();
         } //end finally try
      } //end try
   }
}
```
Now let us compile above servlet and create following entries in web.xml
```
<servlet>
   <servlet-name>DatabaseAccess</servlet-name>
   <servlet-class>DatabaseAccess</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>DatabaseAccess</servlet-name>
   <url-pattern>/DatabaseAccess</url-pattern>
</servlet-mapping>
```
Now call this servlet using URL http://localhost:8080/DatabaseAccess which would display following response −
```
Database Result
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
```

Part - C

1. Explain the javax.servlet and javax.servlet.http packages of importance in web application

Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.Types of Packages.There are two types of packages in Java Servlet that are providing various functioning features to servlet Applications. The two packages are as follows:

1. javax.servlet package
2. javax.servlet.http package

Type 1: javax.servlet package: This package of Servlet contains many servlet interfaces and classes which are capacity of handling any types of protocol sAnd This javax.servlet package containing large interfaces and classes that are invoked by the servlet or web server container as they are not specified with any protocol.

Type 2: javax.servlet.http package: This package of servlet contains more interfaces and classes which are capable of handling any specified http types of protocols on the servlet. This javax.servlet.http package containing many interfaces and classes that are used for http requests only for servlet

Type 1: javax.servlet package:

Servlet: This interface describes and connects all the methods that a Servlet must implement. It includes many methods to initialize the destroy of the Servlet, and a general (service()) method which is handling all the requests are made to it. This Servlet interface is used to creating this servlet class as this class having featuring to implementing these interfaces either directly or indirectly to within it on to fetching servlets

ServletRequest: This ServletRequest interface in which examining the methods for all objects as encapsulating data information about its all requests i.e. made to the servers, this object of the ServletRequest interface is used to retrieve the information data from the user

ServletResponse: An interface examining the methods for all objects which are returning their allowed responses from the servers and object of this current interfacing objects is used to estimate the response to the end-user on the system

ServletConfig: declaring this interface ServletConfig useful to gaining accessing the configuration of its main parameters which are passing through the Servlets during the phase time of initialization and this ServletConfig object is used for providing the information data to the servlet classes external to explicitly.

ServletContext: The object of the ServletContext interface is very helpful to featuring the info. data to the web applications are explaining to it for servlets

GenericServlet: This is a generic classes examination to implement the Servlet. if you want to write the Servlet's protocols other than the HTTP, then the easy way of doing this is to extend GenericServlet rather than by directly implementing the Servlet interfaces

ServletException: it is an exception that can be thrown when the Servlet invoking a problem of some examples

ServletInputStream: This class ServletInputStream is used to reading the binary data from end user request

ServletContextEvent: in this any changes are made in the servlet context of its web application, this class notifies it to the end-user.

ServletOutputStream: This class ServletOutputStream is useful to send the transferring binary data to the end-user side of the system

Type 2: javax.servlet.http package

Classes in javax.servlet.http package

HttpServlet: in this HttpServlet purely abstracted class having features as functionality to extending and applying on the HTTP requests. They have like Service() method that is declared in the Servlet interfaces will now call its methods similar to doGet() and the doPost(), which are enabled to providing behavior to the Calling Servlet
Cookie: This Class provides the feature Servlet an interface for the storage of small portions of data information on the end-user computer or system.
HttpServletRequestWrapper and HttpServletResponseWrapper: this two wrapper classes allowing capability of the HttpServletResponse and HttpServletRequest interfaces to the servlet by its functions
HttpSessionEvent: This class HttpSessionEvent notified as any activity or changes/editing are encountered in the session of web applications in servlet
HttpSessionBindingEvent: This class notified when any attribute is bounded, unbounded or replaced in any Current session
Implementation: Example on servlet by implements

2.Write a Servlet program to link with HTML program

Servlet java program

```
package ss;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class Myservlet extends HttpServlet {
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        out.println("<h1>This Servlet program link with HTML program</h1>");
    }
  }
}
```

Client HTML program

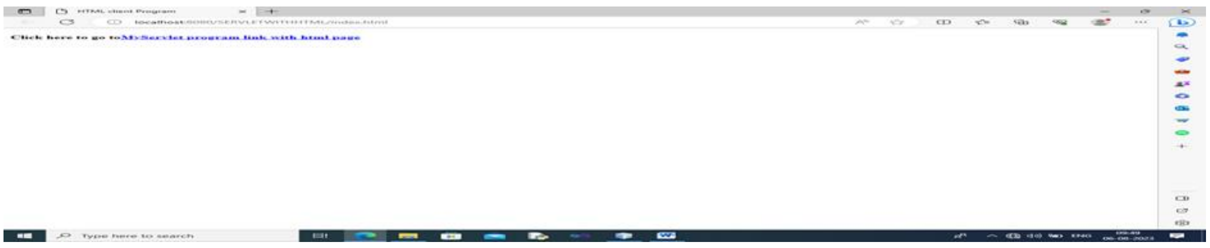
```
<html>
  <head>
    <title>HTML client Program</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <h4>Click here to go to<a href="Myservlet">MyServlet program link with html page</a></h4>
    </body>
</html>
```

Web: xml program (web.xml)

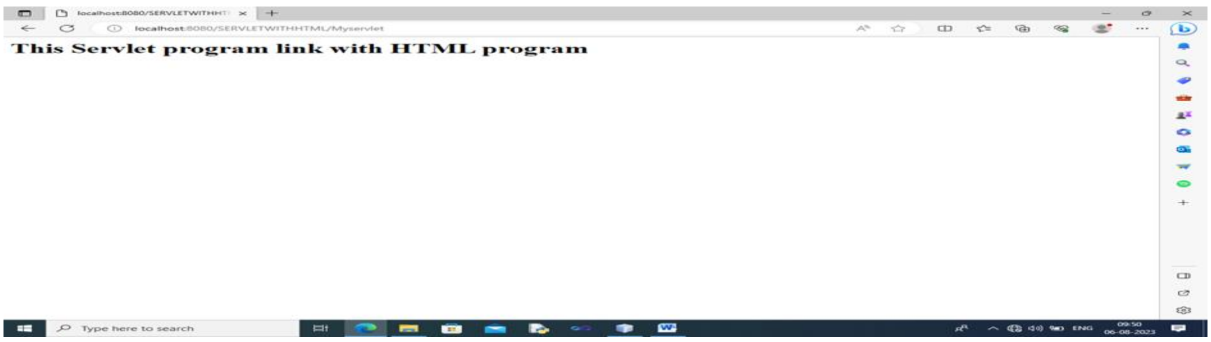
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <servlet>
        <servlet-name>Myservlet</servlet-name>
        <servlet-class>ss.Myservlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Myservlet</servlet-name>
        <url-pattern>/Myservlet</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

Output:



Click here to go toMyServlet program link with html page



This Servlet program link with HTML program

3.Explain benefits of Using Cookies in Servlets

Session tracking is an important feature in web applications that allows the server to maintain state across multiple requests from the same client. Cookies and URL rewriting are two common approaches for implementing session tracking in Servlets.

Benefits of using cookies for session tracking:

1. Cookies are a simple and widely supported mechanism for session tracking.
2. Cookies are stored on the client-side, which means that server-side memory is not used for storing session information. This can help reduce the load on the server.
3. Cookies can be configured to expire after a certain period of time or when the user closes their browser, which can help improve security and privacy.

Benefits of using URL rewriting for session tracking:

1. URL rewriting is a simple and effective mechanism for session tracking that does not require cookies.
2. URL rewriting works even if cookies are disabled or deleted, which can help ensure that session tracking remains functional.
Both cookies and URL rewriting can be effective mechanisms for session tracking in servlet

Example: benefits of Cookie class
javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.
Constructor of Cookie class
Constructor
Cookie()//constructs a cookie.
Cookie(String name, String value)//constructs a cookie with a specified name and value.

Useful Methods of Cookie class
There are given some commonly used methods of the Cookie class.
Method
public void setMaxAge(int expiry)//Sets the maximum age of the cookie in seconds.
public String getName()//Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()//Returns the value of the cookie.
public void setName(String name)//changes the name of the cookie.
public void setValue(String value)//changes the value of the cookie.

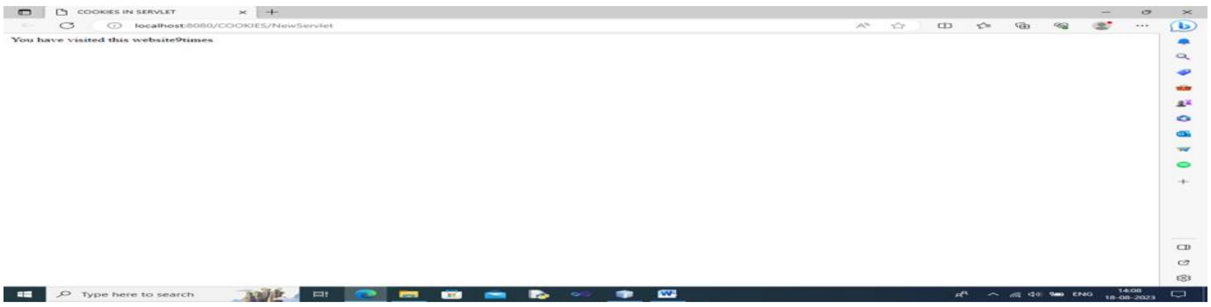
4.Write a Servlet program to display the how many times visited the website counting using with Cookies

Servlet program for Cookies:

```
package CK;
import java.io.IOException;
import java.io.PrintWriter;
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 NewServlet extends HttpServlet {
  static int i = 1;
  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
      out.println("<!DOCTYPE html>");
      out.println("<html>");
      out.println("<head>");
      out.println("<title>COOKIES IN SERVLET</title>");
      out.println("</head>");
      out.println("<body>");
      Cookie c = new Cookie("Visit", String.valueOf(i));
      response.addCookie(c);
```

```
            int j = Integer.parseInt(c.getValue());
            if (j == 1) {
               out.println("Welcome User ");
            } else {
               out.println(" You have visited  this website" + j + "times");
            }
            i++;
            out.println("</body>");
            out.println("</html>");
         }
      }
```

Output:



```
You have visited this website9times
```

Unit-3: JSP: Overview of JSP Technology, Need of JSP, Advantages of JSP, Life Cycle of JSP Page, JSP Processing, JSP Application Design with MVC, Setting Up the JSP Environment, JSP Directives, JSP Action, JSP Implicit Objects, JSP Form Processing, JSP Session and Cookies Handling, JSP Session Tracking JSP Database Access, JSP Standard Tag Libraries, JSP Custom Tag, JSP Expression Language, JSP Exception Handling

Part - A

1.What is JSP?

JSP stands for Java Server Pages, otherwise known as Jakarta Server Pages. It is a server-side technology used by web developers to create dynamic web pages that are based on HTML, XML, and other web document types.

2. What is MVC?

MVC stands for **Model View and Controller.** It is a design pattern that separates the business logic, presentation logic and data. Controller acts as an interface between View and Model. Controller intercepts all the incoming requests. Model represents the state of the application i.e. data. It can also have business logic.View represents the presentaion i.e. UI(User Interface).

3.What is ASP?

ASP stands for active server pages and it is a server-side script engine for building web pages.

4.What is Model Layer?

The Model defines the business layer of the application *This is the data layer which contains business logic of the system, and also represents the state of the application.It's independent of the presentation layer, the controller fetches the data from the Model layer and sends it to the View layer.*

5.What is View Layer?
The View defines the presentation layer of the application. This layer represents the output of the application, usually some form of UI. The presentation layer is used to display the Model data fetched by the Controller.

6.What is Controller Layer?
The Controller manages the flow of the application Controller layer acts as an interface between View and Model. It receives requests from the View layer and processes them, including the necessary validations.

7.What is Implicit Objects?

Implicit objects are a set of Java objects that the JSP Container makes available to developers on each page. These objects may be accessed as built-in variables via scripting elements and can also be accessed programmatically by JavaBeans and Servlets.JSP provide you Total 9 implicit
Part - B

1. What are the advantages of JSP?

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc. A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.Java Server Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A Java Server Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands. Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically. JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

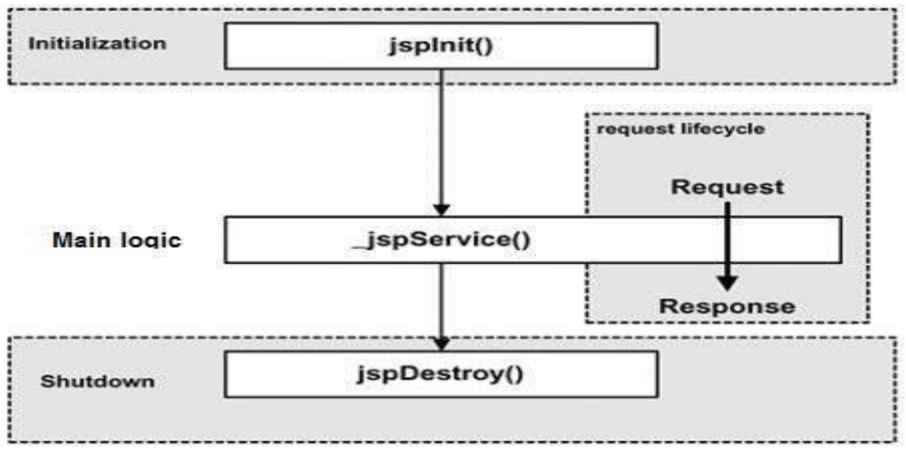
4) Less code than Servlet

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

2. Explain the life cycle of a JSP.

A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

Paths Followed By JSP
The following are the paths followed by a JSP –
1. Compilation
2. Initialization
3. Execution
4. Cleanup
The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below –



JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.
The compilation process involves three steps –
1. Parsing the JSP.
2. Turning the JSP into a servlet 3. Compiling the servlet.
JSP Initialization

When a container loads a JSP it invokes the jspInit() method before servicing any requests. If you need to perform JSP-specific initialization, override the jspInit() method −

```
public void jspInit(){
  // Initialization code...
}
```

Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP Execution

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed. Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the _jspService() method in the JSP.

The _jspService() method takes an HttpServletRequest and an HttpServletResponse as its parameters as follows

```
void _jspService(HttpServletRequest request, HttpServletResponse response) {
  // Service handling code...
}
```

The _jspService() method of a JSP is invoked on request basis. This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, GET, POST, DELETE, etc.

JSP Cleanup

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The jspDestroy() method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

The jspDestroy() method has the following form

```
public void jspDestroy() {
  // Your cleanup code goes here.
}
```

3. Explain the JSP processing.

JavaServer Pages (JSP) is a technology that helps developers create dynamic, data-driven web pages. JSP pages are compiled into Java servlets and run on the server. JSP uses a special syntax that embeds snippets of Java code within HTML, and these pages are stored as regular HTML files with a . jsp extension.

JSP program execution in web:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
 <head>
     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
       <title>JSP Page</title>
   </head>
```

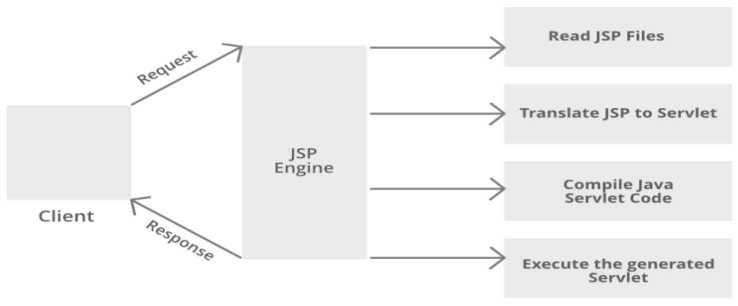
```
    <body>
      <h1>This is Srinivas Univesity Mangalore</h1>
    </body>
</html>
```

Output:

```
http://localhost:8080/HelloWorldjsp/srnivasUniversity.jsp
This is Srinivas Univesity Mangalore
```

JSP which stands for Java Server Pages. It is a server-side technology. It is used for creating web applications. It is used to create dynamic web content. In this JSP tags are used to insert JAVA code into HTML pages. It is an advanced version of Servlet Technology. It is a Web-based technology that helps us to create dynamic and platform-independent web pages. In this, Java code can be inserted in HTML/ XML pages or both. JSP is first converted into a servlet by JSP container before processing the client's request. JSP Processing is illustrated and discussed in sequential steps prior to which a pictorial media is provided as a handful pick to understand the JSP processing better which is as follows:

JSP Processing



Step 1: The client navigates to a file ending with the .jsp extension and the browser initiates an HTTP request to the webserver. For example, the user enters the login details and submits the button. The browser requests a status.jsp page from the webserver.

Step 2: If the compiled version of JSP exists in the web server, it returns the file. Otherwise, the request is forwarded to the JSP Engine. This is done by recognizing the URL ending with .jsp extension.

Step 3: The JSP Engine loads the JSP file and translates the JSP to Servlet(Java code). This is done by converting all the template text into println() statements and JSP elements to Java code. This process is called translation.

Step 4: The JSP engine compiles the Servlet to an executable .class file. It is forwarded to the Servlet engine. This process is called compilation or request processing phase.

Step 5: The .class file is executed by the Servlet engine which is a part of the Web Server. The output is an HTML file. The Servlet engine passes the output as an HTTP response to the webserver.
Step 6: The web server forwards the HTML file to the client's browser.

4. Explain types of JSP Implicit Objects

These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared. JSP Implicit Objects are also called pre-defined variables.

S.No.	Object & Description
1	request This is the HttpServletRequest object associated with the request.
2	response This is the HttpServletResponse object associated with the response to the client.
3	**out** This is the **PrintWriter** object used to send output to the client.
4	**session** This is the **HttpSession** object associated with the request.
5	**application** This is the **ServletContext** object associated with the application context.
6	**config** This is the **ServletConfig** object associated with the page.
7	**pageContext** This encapsulates use of server-specific features like higher performance **JspWriters**.
8	**page** This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class.
9	**Exception** The **Exception** object allows the exception data to be accessed by designated JSP.

The request Object:

The request object is an instance of a javax.servlet.http.HttpServletRequest object. Each time a client requests a page the JSP engine creates a new object to represent that request. The request object provides methods to get the HTTP header information including form data, cookies, HTTP methods etc.

The response Object:

The response object is an instance of a javax.servlet.http.HttpServletResponse object. Just as the server creates the request object, it also creates an object to represent the response to the client. The response object also defines the interfaces that deal with creating new HTTP headers. Through this object the JSP programmer can add new cookies or date stamps, HTTP status codes, etc.

The out Object:

The out implicit object is an instance of a javax.servlet.jsp.JspWriter object and is used to send content in a response. The initial JspWriter object is instantiated differently depending on whether the page is buffered or not. Buffering can be easily turned off by using the buffered = 'false' attribute of the page directive. The JspWriter object contains most of the same methods as the java.io.PrintWriter class. However, JspWriter has some additional methods designed to deal with buffering. Unlike the PrintWriter object, JspWriter throws IOExceptions.
Following table lists out the important methods that we will use to write boolean char, int, double, object, String, etc.

S.No.	Method & Description
1	out.print(dataType dt) Print a data type value
2	out.println(dataType dt) Print a data type value then terminate the line with new line character.
3	out.flush() Flush the stream.

The session Object:

The session object is an instance of javax.servlet.http.HttpSession and behaves exactly the same way that session objects behave under Java Servlets.The session object is used to track client session between client requests..

The application Object:

The application object is direct wrapper around the **ServletContext** object for the generated Servlet and in reality an instance of a **javax.servlet.ServletContext** object.

This object is a representation of the JSP page through its entire lifecycle. This object is created when the JSP page is initialized and will be removed when the JSP page is removed by the **jspDestroy()** method.By adding an attribute to application, you can ensure that all JSP files that make up your web application have access to it.

The config Object:

The config object is an instantiation of **javax.servlet.ServletConfig** and is a direct wrapper around the **ServletConfig** object for the generated servlet.This object allows the JSP programmer access to the Servlet or JSP engine initialization parameters such as the paths or file locations etc.The following **config** method is the only one you might ever use, and its usage is trivial −

```
config.getServletName();
```

This returns the servlet name, which is the string contained in the <**servlet-name**> element defined in the **WEB-INF\web.xml** file.

The pageContext Object:

The pageContext object is an instance of a javax.servlet.jsp.PageContext object. The pageContext object is used to represent the entire JSP page. This object is intended as a means to access information about the page while avoiding most of the implementation details. This object stores references to the request and response objects for each request. The application, config, session, and out objects are derived by accessing attributes of this object.

The pageContext object also contains information about the directives issued to the JSP page, including the buffering information, the errorPageURL, and page scope.The PageContext class defines several fields, including **PAGE_SCOPE, REQUEST_SCOPE, SESSION_SCOPE,** and **APPLICATION_SCOPE**, which identify the four scopes. It also supports more than 40 methods, about half of which are inherited from the **javax.servlet.jsp.JspContext class**. One of the important methods is **removeAttribute**. This method accepts either one or two arguments. For example, **pageContext.removeAttribute ("attrName")** removes the attribute from all scopes, while the following code only removes it from the page scope −

pageContext.removeAttribute("attrName", PAGE_SCOPE);

The page Object:

This object is an actual reference to the instance of the page. It can be thought of as an object that represents the entire JSP page. The page object is really a direct synonym for the this object.

The exception Object:

The exception object is a wrapper containing the exception thrown from the previous page. It is typically used to generate an appropriate response to the error condition.

5. Explain the JSP form processing.

JSP Form Processing

Forms are the common method in web processing. We need to send information to the web server and that information. There are two commonly used methods to send and get back information to the web server.

GET Method:

This is the default method to pass information from browser to web server.
It sends the encoded information separated by ?character appended to URL page.
It also has a size limitation, and we can only send 1024 characters in the request.
We should avoid sending password and sensitive information through GET method.

POST Method:

Post method is a most reliable method of sending information to the server.
It sends information as separate message.
It sends as text string after ?in the URL.
It is commonly used to send information which are sensitive.

JSP handles form data processing by using following methods:

getParameter():It is used to get the value of the form parameter.

getParameterValues():It is used to return the multiple values of the parameters.

getParameterNames()It is used to get the names of parameters.

getInputStream()It is used to read the binary data sent by the client.

Example:

In this example, we have taken a form with two fields.”username” and “password” with a submit button

Action_form.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-
1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Guru Form</title>
</head>
<body>
<form action="action_form_process.jsp" method="GET">
UserName: <input type="text" name="username">
<br />
Password: <input type="text" name="password" />
<input type="submit" value="Submit" />
</form>
```

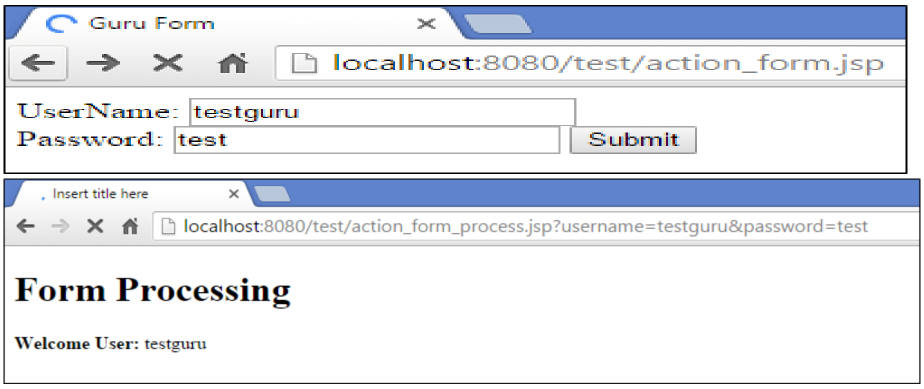
</body>
</html>

Action_form_process.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Form Processing</h1>
<p><b>Welcome User:</b>
   <%= request.getParameter("username")%>
</p>
</body>
</html>
```

Here we have defined a form and through which we have process the action to some other JSP. In action parameter, we add that JSP to which it has to be processed through GET method.
Here we are using GET method to pass the information i.e username and password.Here we are taking fields like username and password which are text fields, and we are getting the input from the user. This input can be fetched using getParameter method. Also, we have submit button with type submit type which helps us to pass the field values into action_form_process.jsp

Action_form_process.jsp , Here we get the values of the input fields from action_form.jsp using request object's getParameter method. **Output:**


Guru Form
localhost:8080/test/action_form.jsp
Submit
Insert title here
localhost:8080/test/action_form_process.jsp?username=testguru&password=test

UserName: testguru
Password: test

Form Processing

Welcome User: testguru

6. Explain the JSP Database access with example

Create Table

To create the Employees table in the EMP database, use the following steps –
Create the Employee table in the TEST database as follows – –

```
mysql> use TEST;
mysql> create table Employees
  (
    id int not null,
    age int not null,
    first varchar (255),
    last varchar (255)
  );
Query OK, 0 rows affected (0.08 sec)
mysql>
```
Create Data Records

create a few records in the Employee table as follows – –

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
Query OK, 1 row affected (0.00 sec)
```

Example: SELECT Operation
Following example shows how we can execute the SQL SELECT statement using JTSL in JSP programming

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>
<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>
 <html>
  <head>
    <title>SELECT Operation</title>
  </head>
  <body>
    <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
      url = "jdbc:mysql://localhost/TEST"
      user = "root"  password = "pass123"/>
     <sql:query dataSource = "${snapshot}" var = "result">
      SELECT * from Employees;
    </sql:query>
     <table border = "1" width = "100%">
      <tr>
        <th>Emp ID</th>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Age</th>
      </tr>
        <c:forEach var = "row" items = "${result.rows}">
        <tr>
          <td><c:out value = "${row.id}"/></td>
          <td><c:out value = "${row.first}"/></td>
          <td><c:out value = "${row.last}"/></td>
          <td><c:out value = "${row.age}"/></td>
        </tr>
      </c:forEach>
    </table>
  </body>
</html>
```

Output:

```
Emp ID  First Name       Last Name       Age
100     Zara              Ali            18
101     Mahnaz           Fatma           25
102     Zaid              Khan           30
103     Sumit            Mittal          28
```

7.Define JSP Standard Tag Libraries and Custom Tag

The Java Server Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications. JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating the existing custom tags with the JSTL tags.

JSP Standard Tag Libraries:

Standard Tag: It provides a rich layer of the portable functionality of JSP pages. It's easy for a developer to understand the code.

Code Neat and Clean: As scriplets confuse developer, the usage of JSTL makes the code neat and clean.

Automatic Javabeans Interospection Support: It has an advantage of JSTL over JSP scriptlets. JSTL Expression language handles JavaBean code very easily. We don't need to downcast the objects, which has been retrieved as scoped attributes. Using JSP scriptlets code will be complicated, and JSTL has simplified that purpose.

Easier for humans to read: JSTL is based on XML, which is very similar to HTML. Hence, it is easy for the developers to understand.

Easier for computers to understand: Tools such as Dreamweaver and front page are generating more and more HTML code. HTML tools do a great job of formatting HTML code. The HTML code is mixed with the scriplet code. As JSTL is expressed as XML compliant tags, it is easy for HTML generation to parse the JSTL code within the document.

The JSTL tags can be classified, according to their functions, into the following JSTL tag library groups that can be used when creating a JSP page −

Core Tags , Formatting tags, SQL tags, XML tags, JSTL Functions

Core Tags:

```
<%@ taglib prefix='c' uri = 'http://java.sun.com/jsp/jstl/core' %>
```

Formatting Tags:

```
<%@ taglib prefix = 'fmt' uri = 'http://java.sun.com/jsp/jstl/fmt' %>
```

SQL Tags:

```
<%@ taglib prefix = 'sql' uri = 'http://java.sun.com/jsp/jstl/sql' %>
```

XML tags:

```
<%@ taglib prefix = 'x' uri = 'http://java.sun.com/jsp/jstl/xml' %>
```

JSTL Functions:

```
<%@ taglib prefix = 'fn' uri = 'http://java.sun.com/jsp/jstl/functions' %>
```

Example: The core tags are most frequently used tags in JSP. They provide support for Iteration Conditional logic , Catch exception , url forward Redirect, etc.

To use core tags we need to define tag library first and below is the syntax to include a tag library.

Syntax :

```
<%@ taglib prefix="c" uri=http://java.sun.com/jsp/jstl/core%>
```

Here, prefix can be used to define all the core tags and uri is the library of taglib from which it is imported

JSTL Custom Tags

1. It is a user-defined JSP language element.
2. When JSP is translated into a servlet, custom tag is converted into a class which takes action on an object and is called as a tag handler.
3. Those actions when the servlet is executed are invoked by the web container.
4. To create the user-defined custom tag, we need to create the tag handler which will be extending the SimpleTagSupport and have to override doTag() method.
5. We need to create TLD where we need to map the class file in TLD.

Advantages of custom tags in JSP

Portable – An action described in a tag library must be usable in any JSP container.

Simple – Unsophisticated users must be able to understand and use this mechanism. Vendors of JSP functionality must find it easy to make it available to users as actions.

Expressive – The mechanism must support a wide range of actions, including nested actions, scripting elements inside action bodies, creation, use and updating of scripting variables.

Usable from different scripting languages – Although the JSP specification currently only defines the semantics for scripts in the Java programming language, we want to leave open the possibility of other scripting languages.

Built upon existing concepts and machinery– We do not want to reinvent what exists elsewhere. Also, we want to avoid future conflicts whenever we can predict them

Example: Syntax:

Consider we are creating testGuru tag and we can usetaghandlertestTag class, which will override doTag() method.

```
<ex:testGuru/>
Class testTag extends SimpleTagSupport{ public void doTag()}
```

To map this testTag class in TLD (Tag Library Descriptor) as JSP container will automatically create a mapping between the class file and uri which has been mentioned in the TLD file.

8. Define JSP expression Language

It has introduced in JSP 2.0 version. Expression Language is mainly used to eliminate java code from the JSP. In general, to eliminate java code and scripting elements from JSP pages, we have to use custom action which requires a lot of java code internally in order to implement simple programming like if, switch, for, and so on. To overcome the above problem, we have to use Expression Language syntax along with standard actions, custom actions, and JSTL tag library.

Syntax: ${expression}, expression is the value present which is evaluated at runtime and being sent to the output stream.

Example

To print the value of request parameter uname: We have to use java code before expression language to get a particular request parameter from the request object.

<% uname=request.getParameter(“uname”);%>

In this example, we are able to retrieve a particular request parameter without using java code by using expression language syntax. **${param.uname}**

Advantages in JSP:

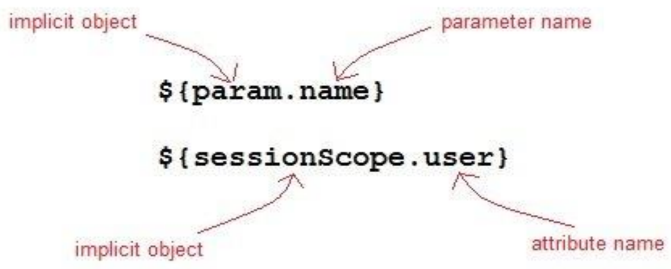
1. We can get request parameters
2. We can get any scope attributes
3. We can get values of bean objects stored in a scope
4. We can get request header values.
5. We can get context parameter values
6. We can get cookie values
7. We can get JSP implicit objects
8. We can do arithmetic, relational, logical, and conditional operations
9. We can call static methods of a class by using EL functions

The syntax of EL in a JSP is as follows:

${expr}

Here expr is a valid EL expression. An expression can be mixed with static text/values and can also be combined with other expressions to form larger expression.

Example:



EL expression can be used in two ways in a JSP page

1. **<u>As attribute values in standard and custom tags.</u>**

Example

<jsp:include page="${location}">

Where location variable is separately defines in the jsp page.

Expressions can also be used in jsp:setProperty to set a properties value, using other bean properties like : If we have a bean named Square with properties length, breadth and area.

<jsp:setProperty name="square" property="area" value="${square.length*square.breadth}" />

2. **<u>To output in HTML tag.</u>**

<h1>Welcome ${name}</h1>

To deactivate the evaluation of EL expressions, we specify the isELIgnored attribute of the page directive as below:

```
<%@ page isELIgnored ="true|false" %>
```

Example: JSP EL Implicit Objects

Implicit Object	**Description**
pageContext	It represents the PageContext object.
pageScope	It is used to access the value of any variable which is set in the Page scope
requestScope	It is used to access the value of any variable which is set in the Request scope.
sessionScope	It is used to access the value of any variable which is set in the Session scope
applicationScope	It is used to access the value of any variable which is set in the Application scope
param	Map a request parameter name to a single value
paramValues	Map a request parameter name to corresponding array of string values.
header	Map containing header names and single string values.
headerValues	Map containing header names to corresponding array of string values.
cookie	Map containing cookie names and single string values.

9. Define JSP Exception handling with example

Write code in JSP, can make some errors while writing lines of code. These errors can be classified into different types. But some errors occur when the code is not logically accurate or an internal error from the system. The concept of handling exceptions to JSP is similar to Java, where exceptions are managed using try-catch blocks.

Exception in JSP

Exceptions can be defined as an object that is thrown during a program run. Exception handling is the practice of handling errors at runtime. Exceptions can occur at any time in a web application. Therefore, the web developer must handle exceptions to be on the safe side and make the user work flawlessly on the web.

There are three types of exceptions in JSP; these are:

Checked Exception

Runtime Exception

Errors Exception

Checked Exception

"Checked exceptions" are a type of exception that is usually a user fault or a problem that cannot be foreseen by the programmer. This type of exception is checked at compile-time. Here is a list of some common "Checked exceptions" that occur in the programming domain:

IO Exception: This is an example of a checked exception where some exceptions may occur while reading and writing a file. If the file format is not supported, then the IO exception will occur.
FileNotFoundException: This is an example of a checked exception where a file in which data needs to be written is not found on that particular path on the disk.
SQLException: This is also an example of a checked exception where the file is associated with a SQL database. The exception will be if there is a problem or concern with the connectivity of the SQL database.

Runtime Exception

"Runtime exceptions" can be defined as exceptions evaded by the programmer. They are left unnoticed at compilation time. Here is the list of some of the common examples that occurred in the programming domain:

ArrayIndexOutOfBoundsException: This is a runtime exception; This occurs when the array size goes beyond the elements or index value set.

NullPointer Exception: This is also an example of a runtime exception raised when a variable or object is empty or null, and users or programmers try to access it.

ArithmeticException: This is an example of a runtime exception when a mathematical operation is not allowed under normal circumstances. A common example of such a scenario is to divide the number by 0.

Errors Exception

"Error exception" arises solely because of the user or programmer. Here you might encounter error due to stack overflows. Here is the list of some of the common examples that occurred in the programming domain:

Error: This is a subclass of throwable that indicates serious problems the applications cannot catch.
Internal Error: This error occurs when a fault occurs in the Java Virtual Machine (JVM).
Instantiation error: This error occurs when you try to instantiate an object, ultimately failing to do that.

input.html

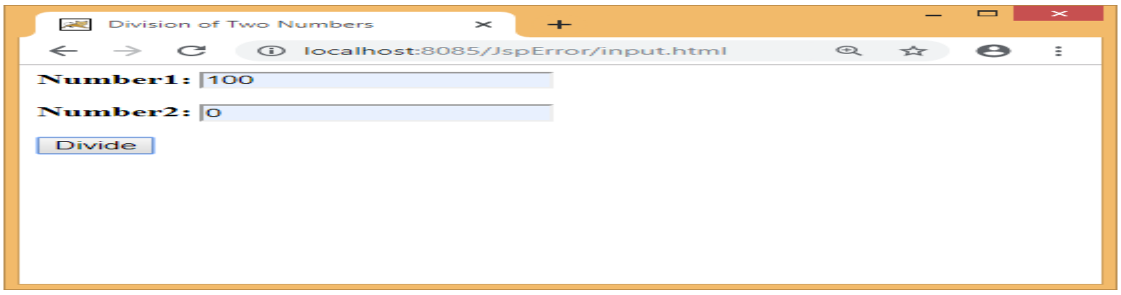
```
<html>
<head>
  <title>Division of Two Numbers</title>
</head>
<body>
  <form action="divide.jsp" method="post">
    <b>Number1: </b><input type="text" name="first" ><br><br>
    <b>Number2: </b><input type="text" name="second" ><br><br>
    <input type="submit" value="Divide">
  </form>
</body>
</html>
```

divide.jsp

```
<%@page errorPage = "error.jsp" %>
<%!
   String num1, num2;
   int a, b, c;
%>
<%
   String num1 = request.getParameter("first");
   String num2 = request.getParameter("second");
   a = Integer.parseInt(num1);
   b = Integer.parseInt(num2);
   c = a / b;
   out.print("Result is: " + c);
%>
```

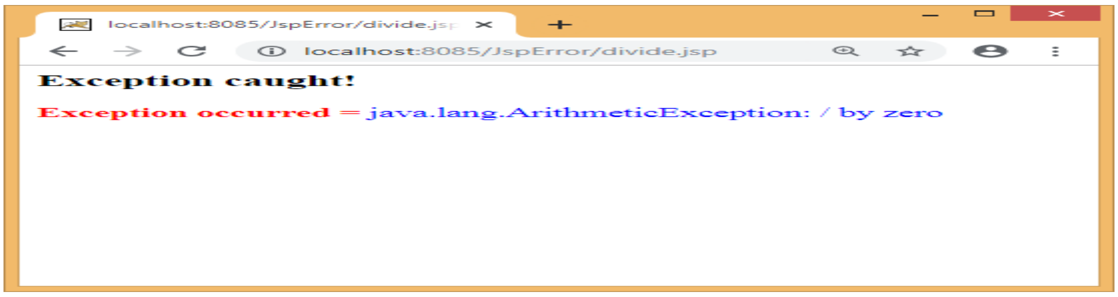
error.jsp

```
<%@ page isErrorPage = "true" %>
<h3> Exception caught!</h3>
<font color="red"><b>Exception occurred = </b></font>
<font color="blue"><%= exception %></font>
```



Number1: 100

Number2: 0



Exception caught!

Exception occurred = java.lang.ArithmeticException: / by zero

Part - C

1. What are the steps to execute JSP page?

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

Creating a simple JSP Page

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

index.jsp

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

```
<html>
<body>
<% out.print(2*5); %>
</body>
</html>
```

Output: It will print 10 on the browser.

Steps to execute the JSP Page.

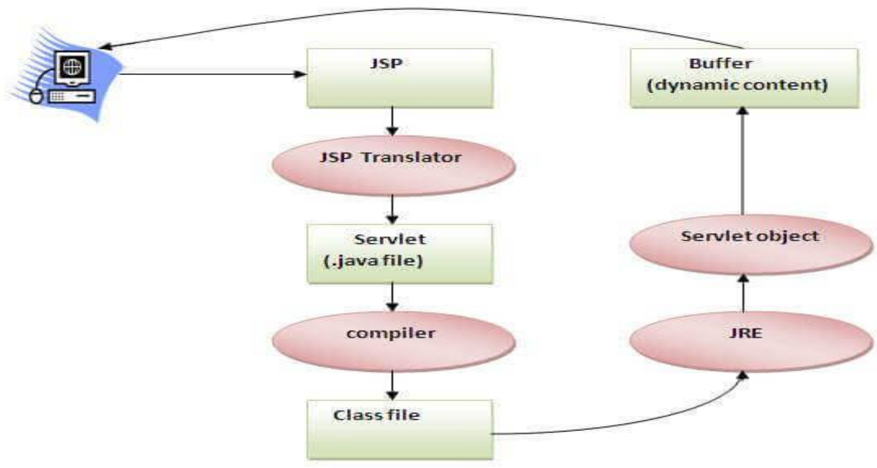
Start the server

Put the JSP file in a folder and deploy on the server

Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

1. Translation of JSP Page
2. Compilation of JSP Page
3. Classloading (the classloader loads class file)

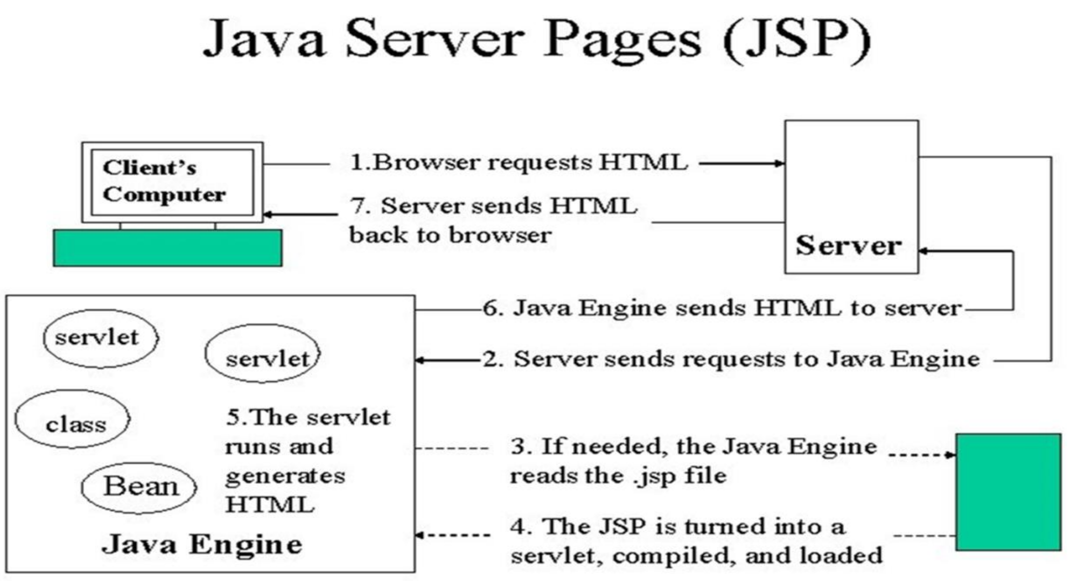
4. Instantiation (Object of the Generated Servlet is created).
5. Initialization (the container invokes jspInit() method).
6. Request processing (the container invokes _jspService() method).
7. Destroy (the container invokes jspDestroy() method).



Note: jspInit(), _jspService() and jspDestroy() are the life cycle methods of JSP.

As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

Example:

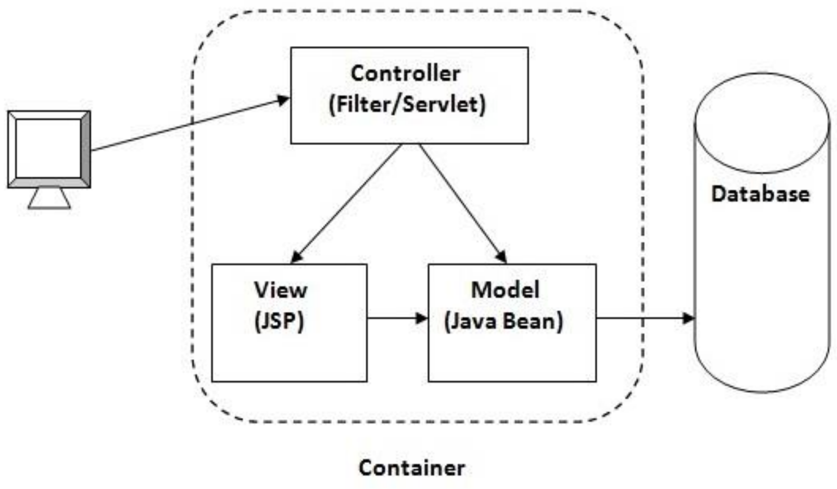

Java Server Pages (JSP)
Client's Computer
1.Browser requests HTML
7. Server sends HTML back to browser
Server
6. Java Engine sends HTML to server
2. Server sends requests to Java Engine
servlet
servlet
class
Bean
5.The servlet runs and generates HTML
Java Engine
3. If needed, the Java Engine reads the .jsp file
4. The JSP is turned into a servlet, compiled, and loaded

2. Explain the MVC Architecture in JSP.

A MVC stands for Model View and Controller. It is a design pattern that separates the business logic, presentation logic and data. Controller acts as an interface between View and Model. Controller intercepts all the incoming requests. Model represents the state of the application i.e. data. It can also have business logic. View represents the presentaion i.e. UI(User Interface).

Architecture

1. Navigation Control is centralized
2. Easy to maintain the large application



MVC is a systematic way to use the application where the flow starts from the view layer, where the request is raised and processed in controller layer and sent to model layer to insert data and get back the success or failure message.

Model Layer:

This is the data layer which consists of the business logic of the system. It consists of all the data of the application

It also represents the state of the application.

It consists of classes which have the connection to the database.

The controller connects with model and fetches the data and sends to the view layer.

The model connects with the database as well and stores the data into a database which is connected to it.

View Layer:

This is a presentation layer.

It consists of HTML, JSP, etc. into it.

It normally presents the UI of the application.

It is used to display the data which is fetched from the controller which in turn fetching data from model layer classes.

This view layer shows the data on UI of the application.

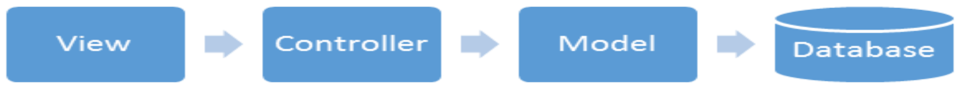
Controller Layer:

It acts as an interface between View and Model.

It intercepts all the requests which are coming from the view layer.

It receives the requests from the view layer and processes the requests and does the necessary validation for the request.

MVC is a systematic way to use the application where the flow starts from the view layer, where the request is raised and processed in controller layer and sent to model layer to insert data and get back the success or failure message. M stands for Model,V stands for View and C stands for controller.



This request is further sent to model layer for data processing, and once the request is processed, it sends back to the controller with required information and displayed accordingly by the view

The advantages of MVC are:
1. Easy to maintain
2. Easy to extend
3. Easy to test
4. Navigation control is centralized

3. Explain the JSP Session and Cookies Handling in JSP
Session tracking in JSP. HTTP is a "stateless" protocol which means each time a client retrieves a Webpage, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.
Maintaining Session Between Web Client And Server
Cookies
A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie. This may not be an effective way as the browser at times does not support a cookie. It is not recommended to use this procedure to maintain the sessions.
Hidden Form Fields
A web server can send a hidden HTML form field along with a unique session ID as follows –
<input type = "hidden" name = "sessionid" value = "12345">
This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or the POST data. Each time the web browser sends the request back, the session_id value can be used to keep the track of different web browsers.
This can be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.
URL Rewriting
append some extra data at the end of each URL. This data identifies the session; the server can associate that session identifier with the data it has stored about that session.
For example, with http://tutorialspoint.com/file.htm;sessionid=12345, the session identifier is attached as sessionid = 12345 which can be accessed at the web server to identify the client.
URL rewriting is a better way to maintain sessions and works for the browsers when they don't support cookies. The drawback here is that you will have to generate every URL dynamically to assign a session ID though page is a simple static HTML page.
The session Object: Apart from the above mentioned options, JSP makes use of the servlet provided HttpSession Interface. This interface provides a way to identify a user across.
1. a one page request or
2. visit to a website or
3. store information about that user

By default, JSPs have session tracking enabled and a new HttpSession object is instantiated for each new client automatically. Disabling session tracking requires explicitly turning it off by setting the page directive session attribute to false as follows −

```
<%@ page session = "false" %>
```

The JSP engine exposes the HttpSession object to the JSP author through the implicit **session** object. Since **session** object is already provided to the JSP programmer, the programmer can immediately begin storing and retrieving data from the object without any initialization or **getSession()**.

Example: This example describes how to use the HttpSession object to find out the creation time and the last-accessed time for a session. We would associate a new session with the request if one does not already exist.

```
<%@ page import = "java.io.*,java.util.*" %>
<%
  // Get session creation time.
  Date createTime = new Date(session.getCreationTime());
    // Get last access time of this Webpage.
  Date lastAccessTime = new Date(session.getLastAccessedTime());
  String title = "Welcome Back to my website";
  Integer visitCount = new Integer(0);
  String visitCountKey = new String("visitCount");
  String userIDKey = new String("userID");
  String userID = new String("ABCD");
  // Check if this is new comer on your Webpage.
  if (session.isNew() ){
    title = "Welcome to my website";
    session.setAttribute(userIDKey, userID);
    session.setAttribute(visitCountKey,  visitCount);
  }
  visitCount = (Integer)session.getAttribute(visitCountKey);
  visitCount = visitCount + 1;
  userID = (String)session.getAttribute(userIDKey);
  session.setAttribute(visitCountKey,  visitCount);
%>
<html>
  <head>
    <title>Session Tracking</title>
  </head>
    <body>
    <center>
      <h1>Session Tracking</h1>
    </center>
        <table border = "1" align = "center">
      <tr bgcolor = "#949494">
        <th>Session info</th>
        <th>Value</th>
      </tr>
      <tr>
        <td>id</td>
        <td><% out.print( session.getId()); %></td>
      </tr>
      <tr>
        <td>Creation Time</td>
        <td><% out.print(createTime); %></td>
      </tr>
      <tr>
        <td>Time of Last Access</td>
        <td><% out.print(lastAccessTime); %></td>
      </tr>
      <tr>
        <td>User ID</td>
        <td><% out.print(userID); %></td>
      </tr>
      <tr>
        <td>Number of visits</td>
        <td><% out.print(visitCount); %></td>
      </tr>
    </table>
    </body>
</html>
```

Now put the above code in main.jsp and try to access http://localhost:8080/main.jsp. Once you run the URL, you will receive the following result –

Welcome to my website

Session Information

Session info	value
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	0

Unit -4: Hibernate Introduction, Hibernate Configuration, Hibernate Concepts, Hibernate O-R Mapping, Manipulating and Querying, Hibernate Query Language, Criteria Queries, Native SQL, Transaction and Concurrency

Part - A

1. What is Hibernate?

Hibernate is java based ORM tool that provides framework for mapping application domain objects to the relational database tables

2. What is ORM?

Object Relational Mapping (ORM) is a technique used in creating a "bridge" between object-oriented programs and, in most cases, relational databases.

3. What is HQL?

Hibernate Query Language is the object-oriented query language of Hibernate Framework. HQL is very similar to SQL except that we use Objects instead of table names, that makes it more close to object oriented programming.

4. Define advantage of HQL

A **database independent**, **polymorphic queries supported**, and **easy to learn for Java programmers**. The Query interface provides object-oriented methods and capabilities for representing and manipulating HQL queries.

5. What is Criteria queries?

The Criteria API is one of the most common ways of constructing queries for entities and their persistent state. It is just an alternative method for defining JPA queries. Criteria API defines a platform-independent criteria queries.

6. Define Native SQL.

Native SQL queries are useful when you need to perform complex queries that cannot be expressed using Hibernate's Query Language (HQL) or Criteria API. **<u>Native SQL queries can be used to perform complex joins, aggregate functions, and subqueries</u>**

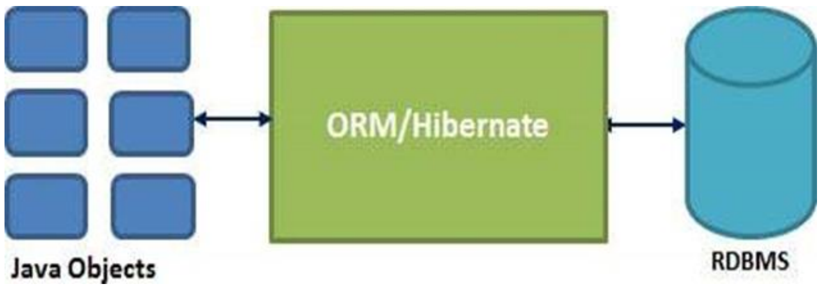
7. What is Java Persistence?

Data Persistence is a means for an application to persist and retrieve information from a non-volatile storage system. The Java™ Persistence API (JPA) provides a mechanism for managing persistence and **object-relational mapping and functions since the EJB 3.0 specifications.**

Part - B

1. **Write short notes about the Hibernate concepts.**

Hibernate is the Object-Relational Mapping (ORM) framework in Java created by Gavin King in 2001. It simplifies the interaction of a database and the Java application being developed. It is an ORM tool that is powerful and lightweight. Another important thing is that this is a high-performance open-source tool. Hibernate implements Java Persistence API specifications and is a powerful object-relational persistence and query service for applications developed in Java. Hibernate is an ORM tool that maps database structures with Java objects dynamically at runtime. Using Hibernate, a persistent framework, allows the developers to focus on just business logic code writing despite writing accurately, as well as a good persistence layer that consists of writing the SQL Queries, connection management, and JDBC Code. Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieves the developer from 95% of common data persistence related programming tasks. Hibernate sits between traditional Java objects and database server to handle all the works in persisting those objects based on the appropriate O/R mechanisms and patterns



Hibernate Advantages

1. Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
2. Provides simple APIs for storing and retrieving Java objects directly to and from the database.
3. If there is change in the database or in any table, then you need to change the XML file properties only.
4. Abstracts away the unfamiliar SQL types and provides a way to work around familiar Java Objects.
5. Hibernate does not require an application server to operate.
6. Manipulates Complex associations of objects of your database.
7. Minimizes database access with smart fetching strategies.
8. Provides simple querying of data.

Supported Databases

Hibernate supports almost all the major RDBMS. Following is a list of few of the database engines supported by Hibernate −

• HSQL Database Engine
• DB2/NT
• MySQL
• PostgreSQL
• FrontBase
• Oracle
• Microsoft SQL Server Database
• Sybase SQL Server
• Informix Dynamic Server

2. Explain the Hibernate O-R mapping process.

Hibernate is a free and open-source object-relational mapper for Java. This simple approach gets rid of all the problems with JDBC. Hibernate develops persistence logic, which arranges and prepares the data for subsequent use. Its advantages over other frameworks include being open-source, portable, and an ORM tool. Hibernate mappings are one of the key features of hibernate and they establish the relationship between two database tables as attributes in your model that allows you to easily navigate the associations in your model and criteria queries.

O/RM itself can be defined as a software or product that represents and/or convert the data between the application (written in Object-Oriented Language) and the database. In other words we can say O/RM maps an object to the relational database table. Hibernate is also an O/RM and is available as open source project. To use Hibernate with Java you will be required to create a file with .hbm.xml extension into which the mapping information will be provided. This file contains the mapping of Object with the relational table that provides the information to the Hibernate at the time of persisting the data. ORM abstracts the application from the process related to database such as saving, updating, deleting of objects from the relational database table.

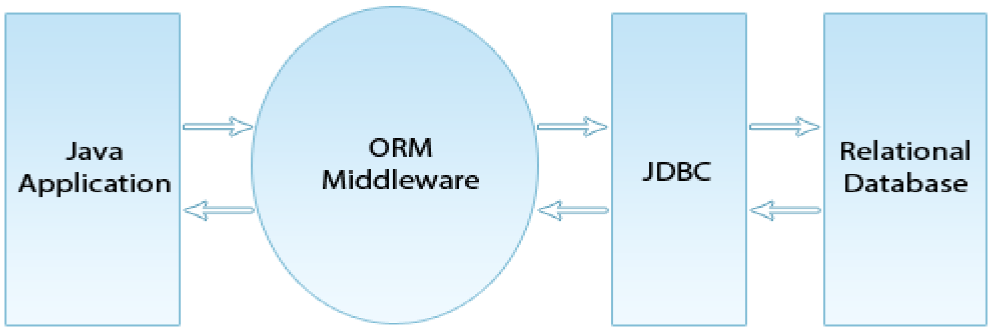
Example :

```
Employee.java
package srinivas;
public class Employee
       {
       private long empId;
       private String empName;
       public Employee() {
       }
       public long getEmpId() {
     return this.empId;
       }
       public void setEmpId(long empId) {
         this.empId = empId;
       }
       public String getEmpName() {
         return this.empName;
       }
       public void setEmpName(String empName) {
         this.empName = empName;
       }
}
```

employee.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="srinivas.Employee" table="employee">
<id name="empId" type="long" column="Id" >
<generator class="assigned"/>
</id>
<property name="empName">
<column name="Name" />
</property>
</class>
</hibernate-mapping>
```

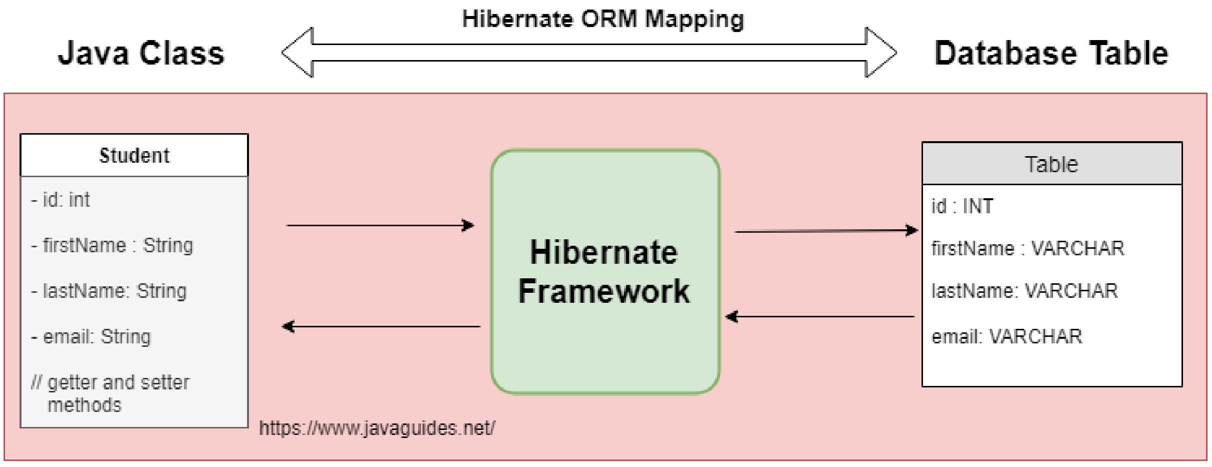
The figure given below demonstrate you how ORM abstracts the application to the database related process and vice-versa :



O – R Mapping process

The above figure demonstrates that an ORM abstracts the both in respect to database how an application tries to persist the data and in respect to an application that how objects are stored in database.

For example, the below diagram shows a Hibernate Object Relational Mapping between Student Java class and student table in the database.



Hibernate provides a reference implementation of Java Persistence API, which makes it a great choice as an ORM tool with the benefits of loose coupling.

3. What are the Hibernate Advantages?

Java Persistence API (JPA) is a Java specification that provides certain functionality and standard to ORM tools. The javax.persistence package contains the JPA classes and interfaces.

Advantages of Hibernate Framework

1) Open Source and Lightweight

Hibernate framework is open source under the LGPL license and lightweight.

2) Fast Performance

The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.

3) Database Independent Query

HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

4) Automatic Table Creation

Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

5) Simplifies Complex Join

Fetching data from multiple tables is easy in hibernate framework.

6) Provides Query Statistics and Database Status

Hibernate supports Query cache and provide statistics about query and database status.

Advantages of Hibernate over JDBC

- Hibernate removes a lot of boiler-plate code that comes with **JDBC** API, the code looks cleaner and readable.
- Hibernate supports inheritance, associations, and collections. These features are not present with **JDBC** API.
- Hibernate implicitly provides transaction management, in fact, most of the queries can't be executed outside a transaction. In JDBC API, we need to write code for transaction management using commit and rollback.
- **JDBC** API throws `SQLException` which is a checked exception, so we need to write a lot of try-catch block code. Most of the time it's redundant in every JDBC call and used for transaction management. Hibernate wraps JDBC exceptions and throws JDBCException or HibernateException un-checked exception, so we don't need to write code to handle it. Hibernate built-in transaction management removes the usage of try-catch blocks.
- **Hibernate Query Language (HQL)** is more object-oriented and close to a **Java programming language**. For **JDBC**, we need to write native SQL queries.
- Hibernate supports caching that is better for performance, **JDBC** queries are not cached hence performance is low.

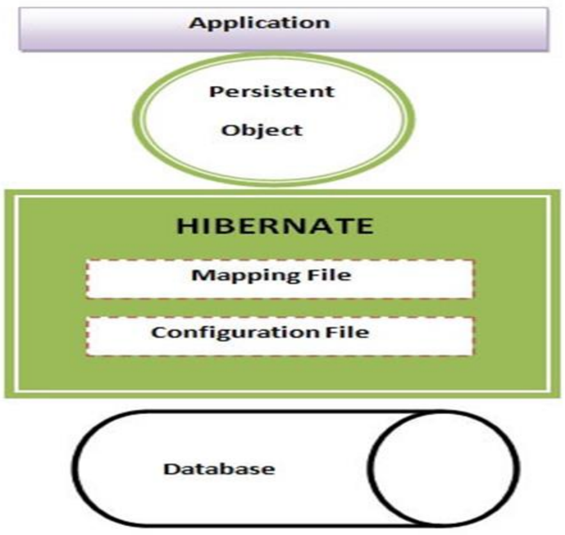
- Hibernate provides an option through which we can create database tables too, for JDBC tables must exist in the database.
- Hibernate configuration helps us in using JDBC-like connection as well as JNDI `DataSource` for a connection pool. This is a very important feature in enterprise applications and completely missing in **JDBC** API.
- Hibernate supports JPA annotations, so the code is independent of the implementation and easily replaceable with other ORM tools. **JDBC** code is very tightly coupled with the application.

4. Explain the Hibernate Architecture

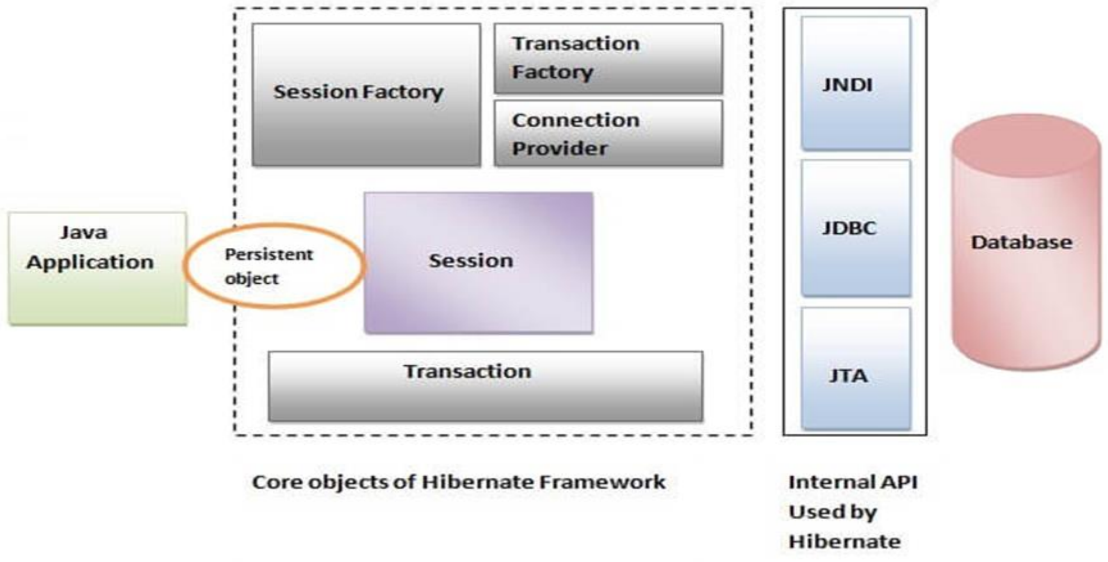
The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.

The Hibernate architecture is categorized in four layers.
Java application layer
Hibernate framework layer
Backhand api layer
Database layer

The diagram of hibernate architecture:



This is the high level architecture of Hibernate with mapping file and configuration file.



Hibernate framework uses many objects such as session factory, session, transaction etc. along with existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

Elements of Hibernate Architecture

For creating the first hibernate application, we must know the elements of Hibernate architecture. They are as follows:

SessionFactory

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

Session

The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

Transaction

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

ConnectionProvider

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

TransactionFactory

It is a factory of Transaction. It is optional.

5. Explain the advantage of using native SQL queries in Hibernate

Hibernate is a popular object-relational mapping (ORM) tool used in Java applications. It allows developers to map Java objects to database tables and perform CRUD (create, read, update, delete) operations on the database without writing SQL queries manually. Native SQL queries are useful when you need to perform complex queries that cannot be expressed using Hibernate's Query Language (HQL) or Criteria API. Native SQL queries can be used to perform complex joins, aggregate functions, and subqueries. To execute a native SQL query in Hibernate, create an SQLQuery object and set the SQL statement to be executed.

Advantages of using native SQL queries in Hibernate

While Hibernate provides a powerful and easy-to-use ORM framework for accessing relational databases, there are some scenarios where native SQL queries can offer advantages over using the Hibernate Query Language (HQL) or Criteria API. Here are some advantages of using native SQL queries in Hibernate:

Performance: Native SQL queries can be optimized for performance and may outperform HQL queries or Criteria queries in some cases. This is because native SQL queries are executed directly by the database, bypassing the Hibernate framework and any overhead that it may introduce.

Complex queries: Native SQL queries can handle complex queries that are difficult or impossible to express using HQL or the Criteria API. For example, queries involving complex joins or subqueries can be expressed more easily and efficiently using native SQL.

Integration with legacy systems: In some cases, legacy systems may require the use of native SQL queries to access the database. By providing support for native SQL queries, Hibernate can integrate more easily with legacy systems and allow them to be modernized more gradually.

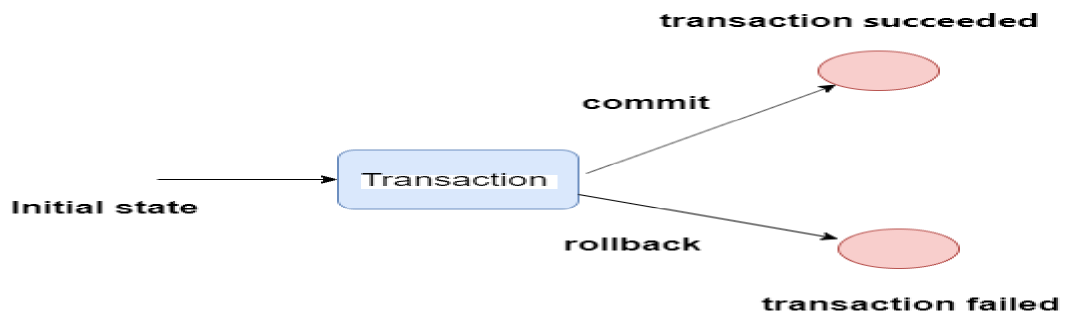
Access to database-specific features: Native SQL queries allow access to database-specific features that may not be available through Hibernate or JPA. This can include advanced features such as stored procedures, triggers, or custom data types.

Familiarity with SQL: Many developers are familiar with SQL and may prefer to use native SQL queries to express their queries in a more familiar syntax. This can make the code easier to read and maintain, especially for developers who are new to Hibernate or JPA.

1. Hibernate provides its own implementation of native SQL queries, which includes additional features not found in the JPA specification.
2. Hibernate Native Query is created using the createSQLQuery() method of the Session interface, which returns an instance of the SQLQuery interface.
3. Hibernate Native Query provides support for additional mapping options, including the ability to map the result set of a query to a non-entity class using the addScalar() method.
4. Hibernate Native Query also provides additional methods for controlling the caching behavior of the query, including setCacheable(), setCacheRegion(), and setCacheMode().

6. Explain the Transaction Interface in Hibernate

Transaction:
A transaction simply represents a unit of work. Generally speaking, a transaction is a set of SQL operations that need to be either executed successfully or not at all.



A transaction can be described by ACID properties (Atomicity, Consistency, Isolation, and Durability).
A database must satisfy the ACID properties (Atomicity, Consistency, Isolation, and Durability) to guarantee the success of a database transaction.

1. **Atomicity:** Each transaction should be carried out in its entirety; if one part of the transaction fails, then the whole transaction fails.

2. **Consistency:** The database should be in a valid state before and after the performed transaction.

3. **Isolation:** Each transaction should execute in complete isolation without knowing the existence of other transactions.

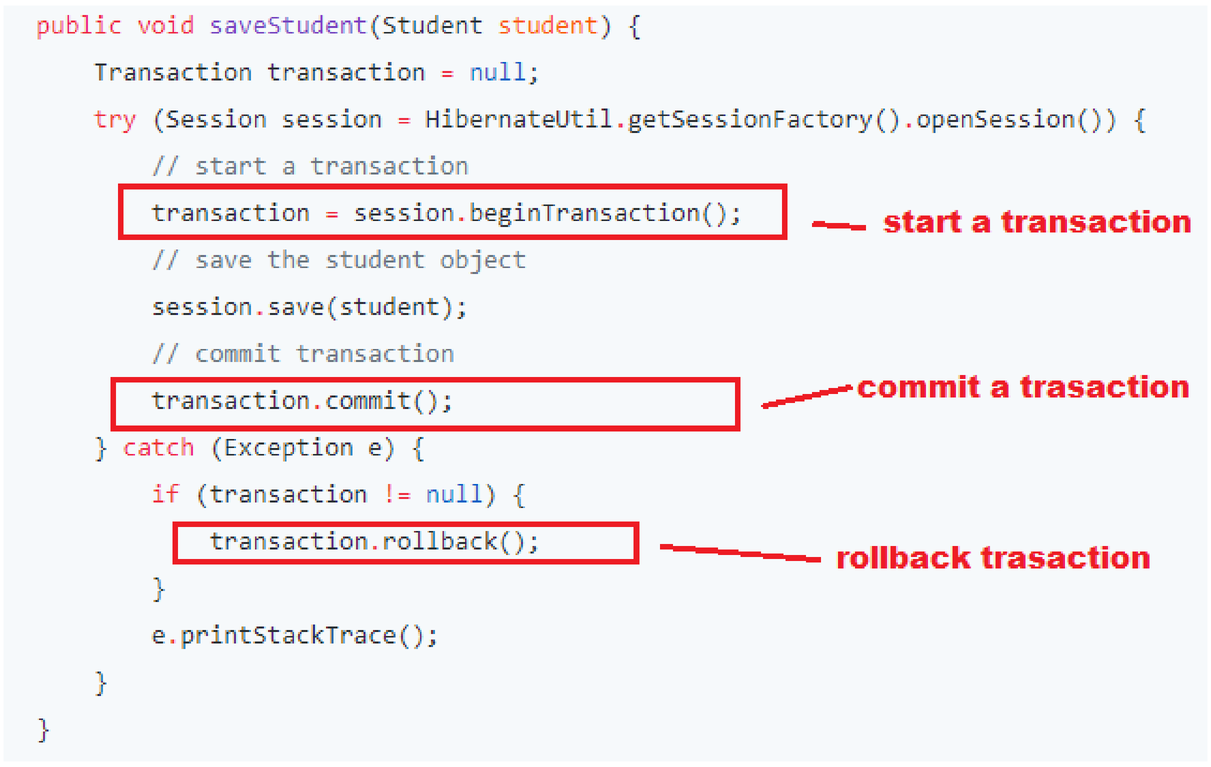
4. **Durability:** Once the transaction is complete, the changes made by the transaction are permanent (even in the occurrence of unusual events such as power loss).

Transaction Interface in Hibernate
In hibernate framework, we have Transaction interface that defines the unit of work. It maintains abstraction from the transaction implementation (JTA,JDBC).A transaction is associated with Session and instantiated by calling session.beginTransaction().
The methods of Transaction interface are as follows:

1. **void begin()** starts a new transaction.
2. **void commit()** ends the unit of work unless we are in FlushMode.NEVER.
3. **void rollback()** forces this transaction to rollback.
4. **void setTimeout(int seconds)** it sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
5. **boolean isAlive()** checks if the transaction is still alive.

6. **void registerSynchronization(Synchronization s)** registers a user synchronization callback for this transaction.
7. **boolean wasCommited()** checks if the transaction is commited successfully.
8. **boolean wasRolledBack()** checks if the transaction is rolledback successfully.

Example:



```
public void saveStudent(Student student) {
    Transaction transaction = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        // start a transaction
        transaction = session.beginTransaction();
        // save the student object
        session.save(student);
        // commit transaction
        transaction.commit();
    } catch (Exception e) {
        if (transaction != null) {
            transaction.rollback();
        }
        e.printStackTrace();
    }
}
```

Part - C

1. Explain the Hibernate configuration approaches details.

Hibernate needs to know about the database configurations to connect to the database. There are three approaches with which we can do the configurations. These approaches are-

Programmatic Configurations – Hibernate does provides a way to load and configure database and connection details programmatically.

XML configurations – We can provide the database details in an XML file. By default hibernate loads the file with name hibernate.cfg.xml but we can load the file with custom name as well.

Properties configurations- This approach uses a properties file for the configuration. By default hibernate loads the file with name hibernate.properties but we can load the file with custom name as well.

Programmatic Configurations

Hibernate does provide a Configuration class which provides certain methods to load the mapping files and database configurations programmatically .

a.Loading Mapping Files

To load the mapping files (also known as hbm files) there are three ways–

1. **addResource()** – We can call addResource() method and pass the path of mapping file available in a classpath. To load multiple mapping files, simply call addResources() method multiple times.

   Below code snippet uses addResource() method to load user.hbm.xml and account.hbm.xml file available in com.hibernate.tutorial package.

   ```
   Configuration configuration = new Configuration();
   configuration.addResource("com/hibernate/tutorial/user.hbm.xml");
   configuration.addResource("com/hibernate/tutorial/account.hbm.xml ");
   ```

2. **addClass()-** Alternatively we can call addClass() method and pass in the class name which needs to persist. To add multiple classes , we can call addClass() multiple times.

   Below code snippet uses addClass() method to load user and account classes available in com.hibernate.tutorial package.

   ```
   Configuration configuration = new Configuration();
   configuration.addClass("com.hibernate.tutorial.user.class");
   configuration.addClass("com.hibernate.tutorial.user.account.class ");
   ```

3. **addJar() –** We can call addJar() to specify the path of jar file containing all the mapping files. This approach provides a generic way and need not to add mapping every time we add a new mapping.

   ```
   Configuration configuration = new Configuration();
   configuration.addJar(new File("mapping.jar"))
   ```

b. Loading Database Configurations

In earlier section we saw how to load the mapping files programmatically, but along with mapping files, Hibernate requires database configurations as well.

We can use the Configuration object to load the database configurations from an instance of properties files, System properties or even set the database properties directly. Lets discuss all approaches in details.

setProperty()- we can call setProperty() method on configuration object and pass the individual property as a key value pair. We can call setProperty() multiple times.

Below code specifies hibernate dialects , driver class , database connection url, database credentials using setProperty() method.

```
<strong><strong><strong><strong>Configuration configuration = new  Configuration();
configuration.setProperty("hibernate.dialect", " org.hibernate.dialect.MySQLDialect");
configuration.setProperty("hibernate.connection.driver_class", "com.mysql.jdbc.Driver");
configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/tutorial");
configuration.setProperty("hibernate.connection.username", "root");
configuration.setProperty("hibernate.connection.password", "password");
</strong></strong></strong></strong>
```

setProperties()- we can call setProperties() method on the configuration object to load the properties from system properties or can pass the property file instance explicitly.

Below code specifies load the database configurations from System properties. All the database configurations will be configured using “ Java -Dproperty=value “

```
Configuration configuration = new Configuration();
configuration.setProperties(System.getProperties());
```

Alternatively, we can create a properties file having database configurations and pass its instance in setProperties()

So we saw how we can load both mapping files and database configuration files programmatically so together the code snippet looks like below

```
<strong><strong><strong><strong>Configuration configuration = new
Configuration();
configuration.addResource("com/hibernate/tutorial/user.hbm.xml");
configuration.addResource("com/hibernate/tutorial/account.hbm.xml ");
configuration.setProperty("hibernate.dialect", "
org.hibernate.dialect.MySQLDialect");
configuration.setProperty("hibernate.connection.driver_class",
"com.mysql.jdbc.Driver");
configuration.setProperty("hibernate.connection.url", "
jdbc:mysql://localhost:3306/tutorial");
configuration.setProperty("hibernate.connection.username", "root");
configuration.setProperty("hibernate.connection.password", "password");
</strong></strong></strong></strong>
```

XML Configurations

The XML configuration approach is widely used and Hibernate loads the configurations from a file with name hibernate.cfg.xml from a class path. Alternatively, we can create an XML file with another name and pass the name of the file.

The sample XML file looks like below-

```
<strong><strong><strong><strong><?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
        <session-factory>
```

```
            <property name="hibernate.connection.url">
             jdbc:mysql://localhost:3306/tutorial
         </property>
              <property name="hibernate.connection.username">
               root
              </property>
              <property name="hibernate.connection.password">
                 password
             </property>
             <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
             <property name="show_sql">false</property>
             <property name="hibernate.connection.driver_class">
                  com.mysql.jdbc.Driver
             </property>

            <mapping resource="user.hbm.xml"/>
        </session-factory>
</hibernate-configuration>
</strong></strong></strong></strong>
```

To add mapping resources we can use <mapping resource> tag in hibernate.cfg.xml file .This is similar to addResource() method. Similarly, we can use <mapping jar> and <mapping class> tags for addJar() and addClass()

Just need to instantiate Configuration object like below

```
Configuration configuration = new Configuration().configure();
```

new Configuration() call will load the hibernate.properties file and calling configure() method on configuration object loads hibernate.cfg.xml. In case any property is defined in both hibernate.properties and hibernate.cfg.xml file then xml file will get precedence.

To load the custom files, we can call

```
Configuration configuration = new
Configuration().configure("/configurations/myConfiguration.cfg.xml")
```

The Above code snippet will load myConfiguration.cfg.xml from the configuration subdirectory of class path.

Note: We can skip prefix "hibernate" from the hibernate properties like hibernate.connection.password is equivalent to connection.password

Properties file Configurations

Hibernate looks for a file named hibernate.properties file in the class path. Properties file provides the similar functionality as XML file provides with the difference of " we cannot add a mapping resource in properties file)

Below is the sample content of hibernate.properties file

```
<strong><strong><strong><strong>hibernate.connection.url =
jdbc:mysql://localhost:3306/tutrial hibernate.connection.username = root
hibernate.connection.password = password
hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.show_sql = true</strong></strong></strong></strong>
```

2. How do create web application using hibernate?

Hibernate is an Object-Relational Mapping (ORM) framework for Java. It simplifies database interaction by allowing developers to use regular Java objects to represent database tables. To create a web application with hibernate. For creating the web application, using JSP for presentation logic, Bean class for representing data and DAO class for database codes.

index.jsp

This page gets input from the user and sends it to the register.jsp file using post method.

```
<form action="register.jsp" method="post">
Name:<input type="text" name="name"/><br><br/>
Password:<input type="password" name="password"/><br><br/>
Email ID:<input type="text" name="email"/><br><br/>
<input type="submit" value="register"/>"
 </form>
```

register.jsp

This file gets all request parameters and stores this information into an object of User class. Further, it calls the register method of UserDao class passing the User class object.

```
<%@page import="com.javatpoint.mypack.UserDao"%>
<jsp:useBean id="obj" class="com.javatpoint.mypack.User">
</jsp:useBean>
<jsp:setProperty property="*" name="obj"/>
  <%
int i=UserDao.register(obj);
if(i>0)
out.print("You are successfully registered");
%>
```

User.java

It is the simple bean class representing the Persistent class in hibernate.

```
package com.javatpoint.mypack;
 public class User {
private int id;
private String name,password,email;

//getters and setters


}
```

user.hbm.xml

It maps the User class with the table of the database.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
 "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">
   <hibernate-mapping>
 <class name="com.javatpoint.mypack.User" table="u400">
 <id name="id">
 <generator class="increment"></generator>
 </id>
 <property name="name"></property>
 <property name="password"></property>
 <property name="email"></property>
 </class>
    </hibernate-mapping>
```

UserDao.java

A Dao class, containing method to store the instance of User class.

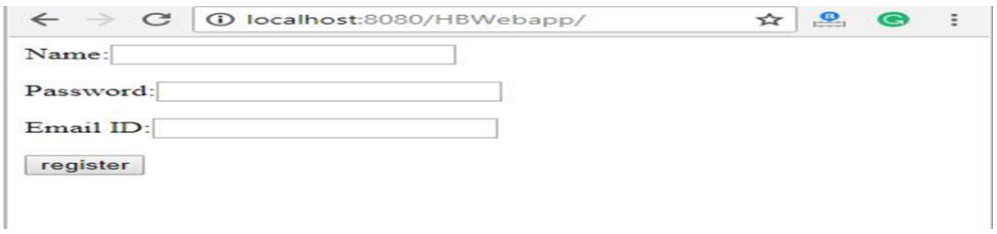
```
package com.javatpoint.mypack;
  import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
   public class UserDao {
  public static int register(User u){
 int i=0;
   StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure("hib
ernate.cfg.xml").build();
 Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();
SessionFactory factory = meta.getSessionFactoryBuilder().build();
Session session = factory.openSession();
Transaction t = session.beginTransaction();
  i=(Integer)session.save(u);
  t.commit();
session.close();
  return i;
    }
}
```

hibernate.cfg.xml

It is a configuration file, containing informations about the database and mapping file.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">
  <hibernate-configuration>
  <session-factory>
 <property name="hbm2ddl.auto">create</property>
 <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
 <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
 <property name="connection.username">system</property>
 <property name="connection.password">jtp</property>
 <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
      <mapping resource="user.hbm.xml"/>
</session-factory>
  </hibernate-configuration>
```

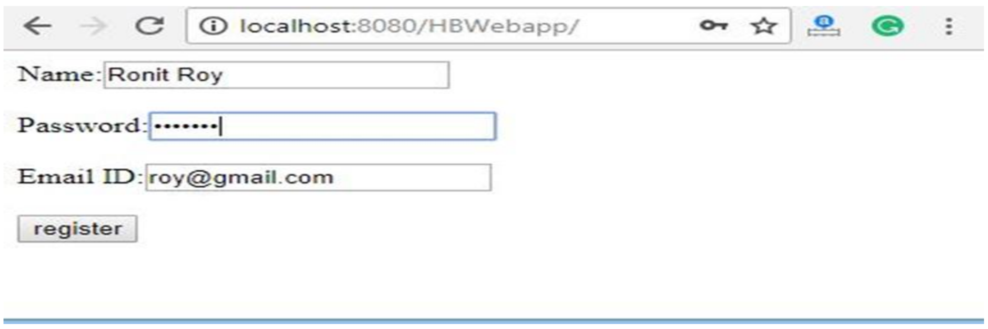
Output:



Name:

Password:

Email ID:



localhost:8080/HBWebapp/

Name: Ronit Roy

Password: ••••••••

Email ID: roy@gmail.com

register

Output: You are successfully registered

3. Explain the Hibernate Mapping Types

The Java data types are mapped into RDBMS data types when you create a Hibernate mapping document. Java or SQL databases do not use the types stated and used in the mapping files. They are known as Hibernate mapping types and can convert Java data types to SQL data types and vice versa.

Primary Types of Hibernate Mapping

There are mainly three different kinds of mapping. As follows:

1. One to one: One property is mapped to another attribute in this form of relationship in a way that only one-to-one mapping is maintained. An illustration will help you comprehend this better. Suppose, for instance, that only one employee works for one department.

When a worker cannot work for another department at the same time, the mapping is known as one-to-one.

2. One to many: One property is mapped to many other characteristics in this form of relationship by being mapped to another attribute in a specific way. An illustration will help you comprehend this better. as in the case of a student who belongs to multiple groups, like a simultaneous cultural organization, sports team, and robotics team.

The student and group relationship in this scenario is referred to be a many-to-one relationship.

3. Many to many: One attribute is mapped to another attribute in this type of connection so that any number of attributes can be linked to other attributes without a limit on the quantity. An example will help you better comprehend this. For instance, in the library, one individual may check out a number of books, and one book may be issued to a number of other books.

Many to many partnerships are what this form of relationship is known as. Before implementation, there must be a thorough knowledge of the business use case for this complex relationship.

Types of Mapping

The primary fundamental forms of Hibernate mapping types are:

1. Primitive Types: Data types such as "integer," "character," "float," "string," "double," "Boolean," "short," "long," etc., are defined for this type of mapping. These can be found in the hibernate framework to map java data types to RDBMS data types.

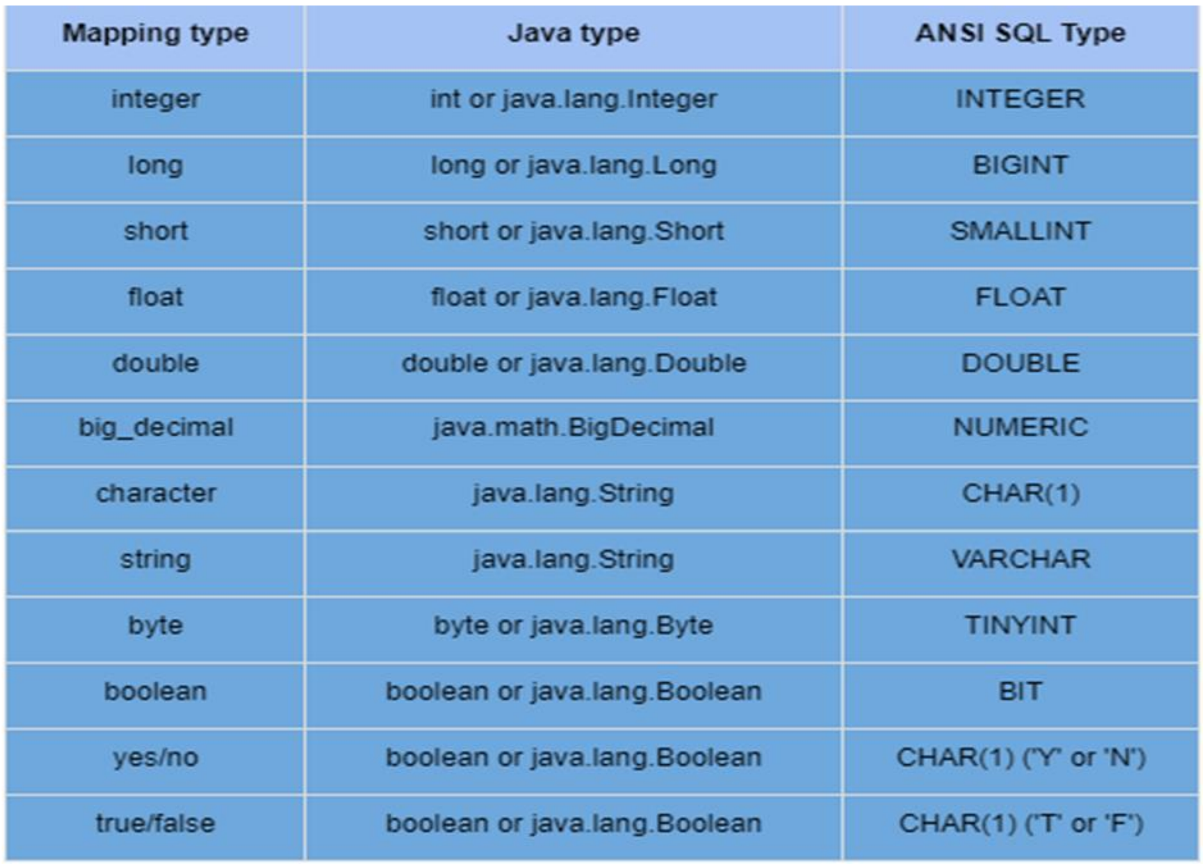
2. Binary and Large Object Types: They include "clob," "blob," "binary," "text," etc. To maintain the data type mapping of big things like images and videos, blob and clob data types are present.

3. Date and Time Types: "Date," "time," "calendar," "timestamp," etc. are some examples. We have data type mappings for dates and times that are similar to primitive.

4. JDK-related Types: This category includes some mappings for things outside the scope of the previous kind of mappings. "Class," "locale," "currency," and "timezone" are among them.

Example 1: Primitive Types

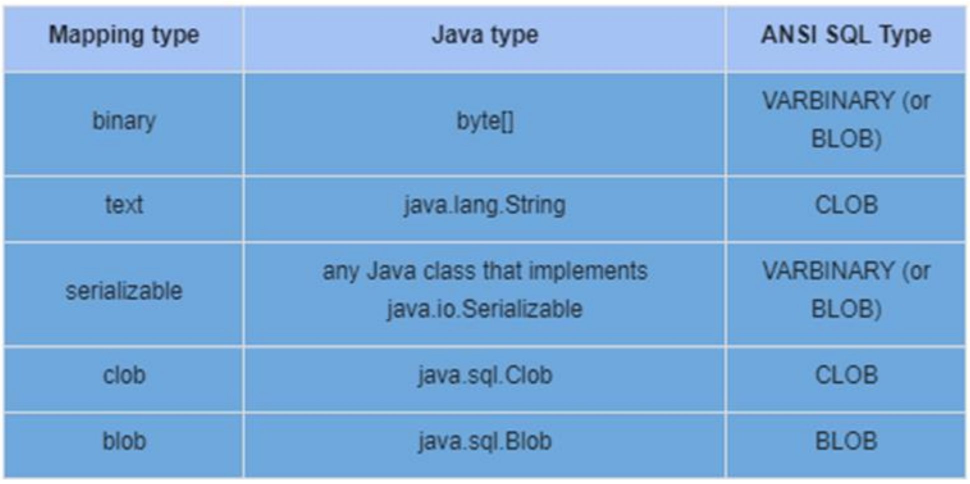
Some of the primitive forms of Hibernate mapping types:



Mapping type	Java type	ANSI SQL Type
integer	int or java.lang.Integer	INTEGER
long	long or java.lang.Long	BIGINT
short	short or java.lang.Short	SMALLINT
float	float or java.lang.Float	FLOAT
double	double or java.lang.Double	DOUBLE
big_decimal	java.math.BigDecimal	NUMERIC
character	java.lang.String	CHAR(1)
string	java.lang.String	VARCHAR
byte	byte or java.lang.Byte	TINYINT
boolean	boolean or java.lang.Boolean	BIT
yes/no	boolean or java.lang.Boolean	CHAR(1) ('Y' or 'N')
true/false	boolean or java.lang.Boolean	CHAR(1) ('T' or 'F')

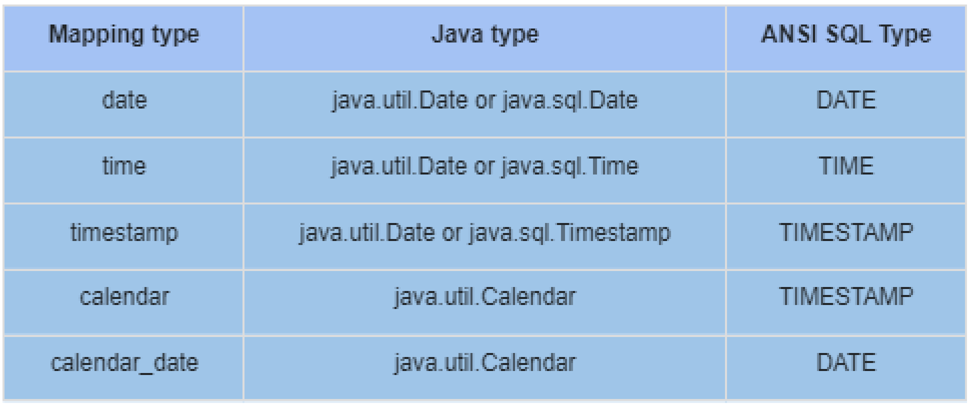
Example:2 Binary and Large Object Types

Binary and Large object types are one of the hibernate mapping types.



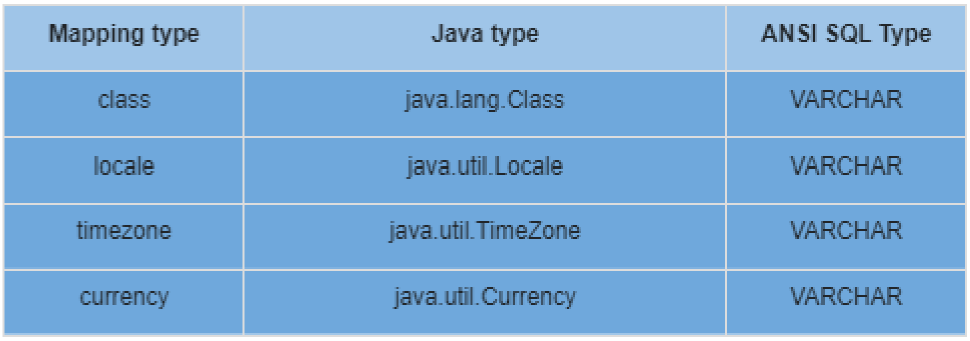
Mapping type	Java type	ANSI SQL Type
binary	byte[]	VARBINARY (or BLOB)
text	java.lang.String	CLOB
serializable	any Java class that implements java.io.Serializable	VARBINARY (or BLOB)
clob	java.sql.Clob	CLOB
blob	java.sql.Blob	BLOB

Example -3:Date and Time Types: Hibernate mapping types also contain the Date and Time types.



Mapping type	Java type	ANSI SQL Type
date	java.util.Date or java.sql.Date	DATE
time	java.util.Date or java.sql.Time	TIME
timestamp	java.util.Date or java.sql.Timestamp	TIMESTAMP
calendar	java.util.Calendar	TIMESTAMP
calendar_date	java.util.Calendar	DATE

Example – 4 JDK-related Types : The fourth category of Hibernate mapping types is JDK-related types



Mapping type	Java type	ANSI SQL Type
class	java.lang.Class	VARCHAR
locale	java.util.Locale	VARCHAR
timezone	java.util.TimeZone	VARCHAR
currency	java.util.Currency	VARCHAR

UNIT- 5 Spring Framework: Spring Basics, Spring Container, Spring AOP, Spring Data Access, Spring O-R/mapping, Spring Transaction Management, Spring Remoting and Enterprise Services, Spring Web MVC Framework, Securing Spring Application.

Part - A

1. What is spring?

Spring framework makes the easy development of JavaEE application. It is a lightweight, loosely coupled and integrated framework for developing enterprise applications in java.

2. What is POJO?

POJO stands for Plain Old Java Object. It is an ordinary Java object, not bound by any special restriction other than those forced by the Java Language Specification and not requiring any classpath. **POJOs are used for increasing the readability and re-usability of a program**

3. What is IoC?

Spring Framework. Inversion of Control (IOC) is a design principle employed by the Spring framework to invert the control of object instantiation and dependency management from the programmer to the framework itself.

4. What is AOP?

Aspect-Oriented Programming (AOP) is a programming paradigm that allows developers to modularize crosscutting concerns, such as logging and security, in their applications. Spring AOP is a powerful framework for implementing AOP in Java applications, and it is integrated with the popular Spring Framework.

5. What is Spring Security?

Spring Security is a framework that provides authentication, authorization, and protection against common attacks. With first class support for securing both imperative and reactive applications, it is the de-facto standard for securing Spring-based applications.

6. What is java beans?

A bean is just a Java object that is made when the application starts by the Spring framework. The object could be a configuration, a service, a database connection factory. JavaBeans are classes which encapsulate several objects into a single object. It helps in accessing these object from multiple places.

7. What is Dependency Injection?

Dependency Injection is a fundamental aspect of the Spring framework, through which the Spring container “injects” objects into other objects or “dependencies”. Simply put, this allows for loose coupling of components and moves the responsibility of managing components onto the container.

Part - B

1**. Explain the application of Spring**

Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable, and reusable code.

Spring framework is an open source Java platform. It was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003.

Spring is lightweight when it comes to size and transparency. The basic version of Spring framework is around 2MB. The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make J2EE development easier to use and promotes good programming practices by enabling a POJO-based programming model.

Applications of Spring

Following is the list of few of the great benefits of using Spring Framework −

- **POJO Based** - Spring enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product such as an application server but you have the option of using only a robust servlet container such as Tomcat or some commercial product.
- **Modular** - Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about the ones you need and ignore the rest.
- **Integration with existing frameworks** - Spring does not reinvent the wheel, instead it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, and other view technologies.
- **Testablity** - Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBeanstyle POJOs, it becomes easier to use dependency injection for injecting test data.
- **Web MVC** - Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over-engineered or less popular web frameworks.
- **Central Exception Handling** - Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.
- **Lightweight** - Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
- **Transaction management** - Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

2. Write short notes about the AOP Terminologies.

Aspect-oriented Programming (AOP) complements Object-oriented Programming (OOP) by providing another way of thinking about program structure. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Aspects enable the modularization of concerns (such as transaction management) that cut across multiple types and objects. (Such concerns are often termed "crosscutting" concerns in AOP literature.)

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Dependency Injection helps you decouple your application objects from each other and AOP helps you decouple cross-cutting concerns from the objects that they affect. AOP is like triggers in programming languages such as Perl, .NET, Java, and others.Spring AOP module provides interceptors to intercept an application. For example, when a method is executed, you can add extra functionality before or after the method execution.

AOP Terminologies

Before we start working with AOP, let us become familiar with the AOP concepts and terminology. These terms are not specific to Spring, rather they are related to AOP

Terms & Description

Aspect
This is a module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging. An application can have any number of aspects depending on the requirement.

Join point
This represents a point in your application where you can plug-in the AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring AOP framework.

Advice
This is the actual action to be taken either before or after the method execution. This is an actual piece of code that is invoked during the program execution by Spring AOP framework.

Pointcut
This is a set of one or more join points where an advice should be executed. You can specify pointcuts using expressions or patterns as we will see in our AOP examples.

Introduction

An introduction allows you to add new methods or attributes to the existing classes.

Target object
The object being advised by one or more aspects. This object will always be a proxied object, also referred to as the advised object.

Weaving
Weaving is the process of linking aspects with other application types or objects to create an advised object. This can be done at compile time, load time, or at runtime.

Types of Advice:

Spring aspects can work with five kinds of advice mentioned as follows

Sr.No	Advice & Description
1	**before** Run advice before the a method execution.
2	**after** Run advice after the method execution, regardless of its outcome.
3	**after-returning** Run advice after the a method execution only if method completes successfully.
4	**after-throwing** Run advice after the a method execution only if method exits by throwing an exception.
5	**around** Run advice before and after the advised method is invoked.

3. Write short notes about the Spring ORM technique.

Object–relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between a relational database and the heap of an object-oriented programming language.

Spring-ORM is a technique or a Design Pattern used to access a relational database from an object-oriented language. ORM (Object Relation Mapping) covers many persistence technologies. They are as follows:

JPA(Java Persistence API): It is mainly used to persist data between Java objects and relational databases. It acts as a bridge between object-oriented domain models and relational database systems.
JDO(Java Data Objects): It is one of the standard ways to access persistent data in databases, by using plain old Java objects (POJO) to represent persistent data.
Hibernate – It is a Java framework that simplifies the development of Java applications to interact with the database.
Oracle Toplink, and iBATIS: Oracle TopLink is a mapping and persistence framework for Java development.

For the above technologies, Spring provides integration classes so that each of these techniques can be used following Spring principles of configuration, and easily integrates with Spring transaction management.For each of the above technologies, the configuration consists of injecting the DataSource bean into some kind of SessionFactory or EntityManagerFactory, etc. For pure JDBC(Java Database Connectivity), integration classes are not required apart from JdbcTemplate because JDBC only depends on a DataSource. If someone wants to use an ORM like JPA(Java Persistence API) or Hibernate then you do not need spring-JDBC, but only this module.

Note: The Spring Framework is an application framework and also an inversion of the control container for the Java platform. The framework's core features can be used by any of the Java applications, but there are some extensions for building web applications on top of the Java EE (Enterprise Edition) platform.

Advantages of the Spring Framework About ORM Frameworks

1. Due to the Spring framework, you do not need to write extra codes before and after the actual database logic such as getting the connection, starting the transaction, committing the transaction, closing the connection, etc.
2. Spring has an IoC(Inversion of control) approach which makes it easy to test the application.
3. Spring framework provides its API for exception handling along with the ORM framework.
4. By using the Spring framework, we can wrap our mapping code with an explicit template wrapper class or AOP(Aspect-oriented programming) style method interceptor.

4. Explain the Programmatic vs Declarative Transactions

Spring framework provides an abstract layer on top of different underlying transaction management APIs. Spring's transaction support aims to provide an alternative to EJB transactions by adding transaction capabilities to POJOs. Spring supports both programmatic and declarative transaction management. EJBs require an application server, but Spring transaction management can be implemented without the need of an application server.

Local vs. Global Transactions: Local transactions are specific to a single transactional resource like a JDBC connection, whereas global transactions can span multiple transactional resources like transaction in a distributed system. Local transaction management can be useful in a centralized computing environment where application components and resources are located at a single site, and transaction management only involves a local data manager running on a single machine. Local transactions are easier to be implemented. Global transaction management is required in a distributed computing environment where all the resources are distributed across multiple systems. In such a case, transaction management needs to be done both at local and global levels. A distributed or a global transaction is executed across multiple systems, and its execution requires coordination between the global transaction management system and all the local data managers of all the involved systems.

Programmatic vs. Declarative

Spring supports two types of transaction management −

- Programmatic transaction management − This means that you have to manage the transaction with the help of programming. That gives you extreme flexibility, but it is difficult to maintain.
- Declarative transaction management − This means you separate transaction management from the business code. You only use annotations or XML-based configuration to manage the transactions.

Declarative transaction management is preferable over programmatic transaction management though it is less flexible than programmatic transaction management, which allows you to control transactions through your code. But as a kind of crosscutting concern, declarative transaction management can be modularized with the AOP approach. Spring supports declarative transaction management through the Spring AOP framework.

Programmatic transaction management is usually a good idea only if you have a small number of transactional operations. For example, if you have a web application that requires transactions only for certain update operations, you may not want to set up transactional proxies by using Spring or any other technology. In this case, using the TransactionTemplate may be a good approach. Being able to set the transaction name explicitly is also something that can be done only by using the programmatic approach to transaction management.On the other hand, if your application has numerous transactional operations, declarative transaction management is usually worthwhile. It keeps transaction management out of business logic and is not difficult to configure. When using the Spring Framework, rather than EJB CMT, the configuration cost of declarative transaction management is greatly reduced.

Programmatic Transaction Management

1. Allows us to manage transactions through programming in our source code.
2. This means hardcoding transaction logic between our business logic.
3. We use programming to manage transactions
4. Flexible, but difficult to maintain with large amount of business logic. Introduces boilerplate between business logic.
5. Preferred when relative less transaction logic is to be introduced.

Declarative Transaction Management

1. Allows us to manage transactions through configuration.
2. This means separating transaction logic with business logic.
3. We use annotations (Or XML files) to manage transactions.
4. Easy to maintain. Boilerplate is kept away from business logic.
5. Preferred when working with large amount of Transaction logic.

Spring offers both programmatic and declarative transactions.

Programmatic means you have transaction management code surrounding your business code. This gives extreme flexibility, but is difficult to maintain and, well, boilerplate.Declarative means you separate transaction management from the business code. You can use annotations or XML based configuration.

Declarative Transaction Management allows to eliminate any dependencies on the transaction framework from the Java code. The four participants to provide the transaction support are transaction manager, proxy factory, transaction interceptor, and a set of transaction attributes.

Suggest to use Declarative Transaction Management, Alternative for HibernateTemplates either NamedJDBCTemplate or simpleJDBCTemplate

5. Describe the Spring Remoting technologies

Spring has integration classes for remoting support that use a variety of technologies. The Spring framework simplifies the development of remote-enabled services. It saves a significant amount of code by having its own API. The remote support simplifies the building of remote-enabled services, which are implemented by your standard (Spring) POJOs. Spring now supports the following remoting technologies:

1. Remote Method Invocation (RMI)
2. Hessian
3. Burlap
4. Spring's HTTP invoker
5. JAX-RPC
6. JAX-WS
7. JMS

1. Exposing services using RMI

Transparently expose your services across the RMI infrastructure by using Spring's RMI support. After you've done this, you'll have a setup that's similar to remote EJBs, except there's no standard support for security context propagation or remote transaction propagation. When utilizing the RMI invoker, Spring provides hooks for such extra invocation context, so you may, for example, plug-in security frameworks or custom security credentials.

Using the RmiServiceExporter to export the service.
Client-side service integration.

2. Using Hessian to call services remotely through HTTP

Spring has its own remoting service that supports HTTP serialization. HTTP Invoker makes use of the classes HttpInvokerServiceExporter and HttpInvokerProxyFactoryBean.

• Configuring the DispatcherServlet for Hessian and company.
• Using the HessianServiceExporter to expose your beans
• Connecting the client to the service

3. Using Burlap

It is the same as Hessian but XML-based implementation provided by Coucho. The classes used in Burlap are BurlapServiceExporter and BurlapProxyFactoryBean.

4. Using HTTP invokers to expose services

Spring HTTP invokers employ the regular Java serialization technique to expose services through HTTP, as opposed to Burlap and Hessian, which are both lightweight protocols with their own compact serialization mechanisms. This is especially useful if your arguments and return types are complicated and cannot be serialized using the serialization procedures used by Hessian and Burlap (refer to the next section for more considerations when choosing a remoting technology).

• Displaying the service object
• Client-side service integration

5. Exposing servlet-based web services using JAX-RPC

Spring provides a convenience base class for JAX-RPC servlet endpoint implementations – ServletEndpointSupport. To expose our AccountService we extend Spring's ServletEndpointSupport class and implement our business logic here, usually delegating the call to the business layer.

Using JAX-RPC to gain access to online services
JAX-RPC Bean Mappings Registration
Adding your own JAX-RPC Handler

6. Using JAX-WS to provide servlet-based web services

SpringBeanAutowiringSupport is a useful basic class for JAX-WS servlet endpoint implementations. We modify Spring's SpringBeanAutowiringSupport class to expose our AccountService and execute our business logic here, generally outsourcing the request to the business layer.

7. JMS

Using JMS (Java Message Service) as the underlying communication protocol, it is also feasible to provide services transparently. The Spring Framework's JMS remoting functionality is fairly minimal – it transmits and receives on the same thread and in the same non-transactional Session, thus performance will be very implementation dependant. It's worth noting that these single-threaded and non-transactional restrictions only apply to Spring's JMS remoting functionality.

6. Explain the Spring Enterprise Services.

Spring framework helps develop various types of applications using the Java platforms. It provides an extensive level of infrastructure support. Spring also provides the "Plain Old Java Objects" (POJOs) mechanisms using which developers can easily create the Java SE programming model with the full and partial JAVA EE(Enterprise Edition). Spring strives to facilitate the complex and unmanageable enterprise Java application development revolution by offering a framework that incorporates technologies, such as:

Aspect-oriented Programming (AOP),
Dependency Injection (DI),
Plain Old Java Object (POJO)

Aspect-oriented Programming (AOP)
AOP provides more modularity to the cross-cutting challenges in applications.
Here are the functions we can use in our applications as per certain real-time challenges:
Logging
Caching
Transaction management
Authentication
AOP has the in-built object-oriented programming capabilities to define the structure of the program, where OOP modularity is established in classes.

In AOP, the primary unit of modularity is a factor (cross-cutting concern). This allows users to use AOP to build custom aspects and declarative enterprise services. The IoC container does not depend on AOP; it provides the custom enabled based capabilities which allow writing logic as per the programming method.

Dependency Injection (DI)

Dependency Injection is the core of Spring Framework. We can define the concept of Spring with the Inversion of Control (IoC). DI allows the creation of dependent objects outside of a class and provides those objects to a class in different ways. Dependency Injection can be utilized while defining the parameters to the Constructor or by post-construction using Setter methods.

The dependency feature can be summarized into an association between two classes. For example, suppose class X is dependent on class Y. Now, it can create many problems in the real world, including system failure. Hence such dependencies need to be avoided. Spring IOC resolves such dependencies with Dependency Injection. Here, it indicates that class Y will get injected into class X by the IoC. DI thus makes the code easier to test and reuse.

While creating a complex Java application, application classes should be independent of other Java classes to improve the possibility of reusing these classes and to test them independently of other classes during unit testing. Dependency Injection enables these classes to be together, and at the same time, keeps them independent.

Plain Old Java Object (POJO)

POJO stands for Plain Old Java Object. It is an ordinary Java object, not bound by any special restriction other than those forced by the Java Language Specification and not requiring any classpath. POJOs are used for increasing the readability and re-usability of a program. POJOs have gained the most acceptance because they are easy to write and understand. They were introduced in EJB 3.0 by Sun Microsystems.

Spring Enterprise Services

Easy start.

Spring has a number of out-of-the-box functions and modules that your team can easily add and customize. Thus, Spring increases developer productivity by focusing on business logic rather than writing code.

Open-source community.

In addition to a number of out-of-the-box features, Spring has a large open-source community that makes it much easier to find a solution to almost any problem. With a lot of people using Spring, a tech team can find explanations for a specific issue or even a ready-to-use solution quickly and effortlessly. Another way to solve the problem is to view an open-source code and even debug it.

Extensibility.

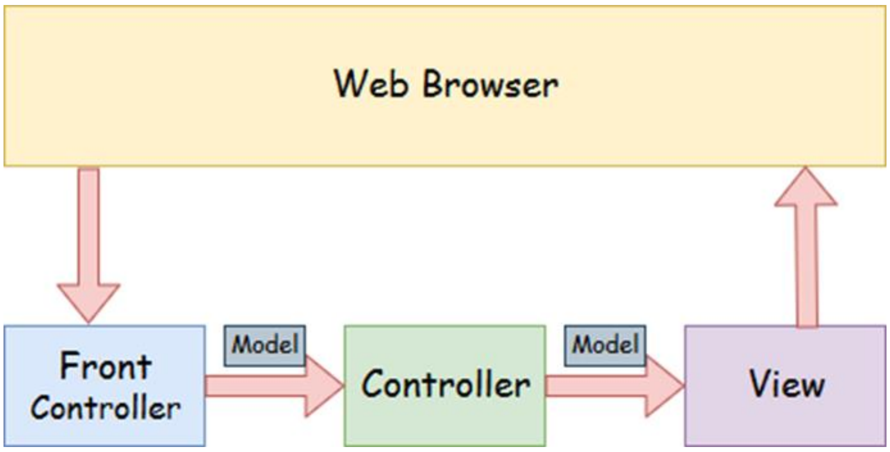
One of the core concepts of Spring is to provide solutions to the most common problems and at the same time leave a convenient place to create/extend custom solutions for a specific problem.

Therefore, if there is no suitable module for solving a specific problem, or the existing one does not fit exactly as needed, then a tech team can expand the existing module or write its own.
Speed.
Rapid start, fast shutdown, and optimized execution are the standard that Spring provides by default. In addition, the Spring kit contains tools for speeding up iterations (LiveReload in Spring DevTools) and getting a quick project start (Spring Initializr at start.spring.io).This way, Spring makes Java faster, and therefore more productive.
Reliability.
Spring is a well-known DI container that retains most modern coding practices, which in turn leads to testable and therefore more reliable code. Read more on https://tech-stack.com/blog/spring-framework-for-enterprise-applications-main-benefits-to-get-and-common-mistakes-to-avoid/

7. Explain the Spring Web MVC Framework design

A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

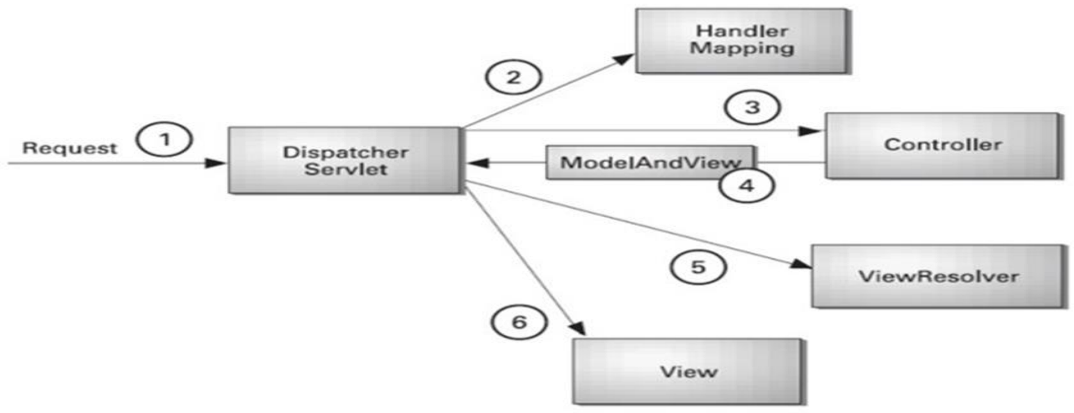
A Spring MVC provides an elegant solution to use MVC in spring framework by the help of **DispatcherServlet**. Here, **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.



- **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
- **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
- **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.

- **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.

Understanding the flow of Spring Web MVC



- As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller.
- The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
- The controller returns an object of ModelAndView.
- The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.

Advantages

Separate roles - The Spring MVC separates each role, where the model object, controller, command object, view resolver, DispatcherServlet, validator, etc. can be fulfilled by a specialized object.
Light-weight - It uses light-weight servlet container to develop and deploy your application.
Powerful Configuration - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
Rapid development - The Spring MVC facilitates fast and parallel development.
Reusable business code - Instead of creating new objects, it allows us to use the existing business objects.
Easy to test - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.
Flexible Mapping - It provides the specific annotations that easily redirect the page.

8. What are the benefits of using Spring Security Application?

Spring Security provides ways to perform authentication and authorization in a web application. We can use spring security in any servlet based web application.

Some of the benefits of using Spring Security are:

1. Proven technology, it's better to use this than reinvent the wheel. Security is something where we need to take extra care, otherwise our application will be vulnerable for attackers.
2. Prevents some of the common attacks such as CSRF, session fixation attacks.
3. Easy to integrate in any web application. We don't need to modify web application configurations, spring automatically injects security filters to the web application.
4. Provides support for authentication by different ways - in-memory, DAO, JDBC, LDAP and many more.
5. Provides option to ignore specific URL patterns, good for serving static HTML, image files.
6. Support for groups and roles.

Authentication and authorization are two critical security aspects provided by Spring Security.

Authentication

- Authentication is the process of verifying the Identity of a user or system. Spring Security provides a wide range of authentication options, including in-memory authentication, JDBC authentication, LDAP authentication, and OAuth 2.0 authentication.

The authentication process typically involves the following steps

- The user provides their credentials (e.g. username and password).
- The credentials are validated against a configured authentication provider (e.g. a user database).
- If the credentials are valid, a security context is established for the user.

- Once authentication is complete, Spring Security provides a range of options for authorization.

Authorization

Authorization refers to the process of determining whether a user is allowed to perform a particular action or access a particular resource.

Spring Security's authorization options include:

Role-based access control: This is a common authorization strategy where access to resources is granted based on the user's role or group membership.

Spring Security provides support for role-based access control through the use of annotations, configuration files, or expressions.

Permission-based access control: This strategy grants access to specific resources based on the user’s permissions or privileges. Spring Security supports permission-based access control through the use of annotations or expressions.

Web access control: This is a specialized form of authorization that controls access to web resources such as URLs or HTTP methods. Spring Security provides support for web access control through the use of request-matching rules and access control lists.

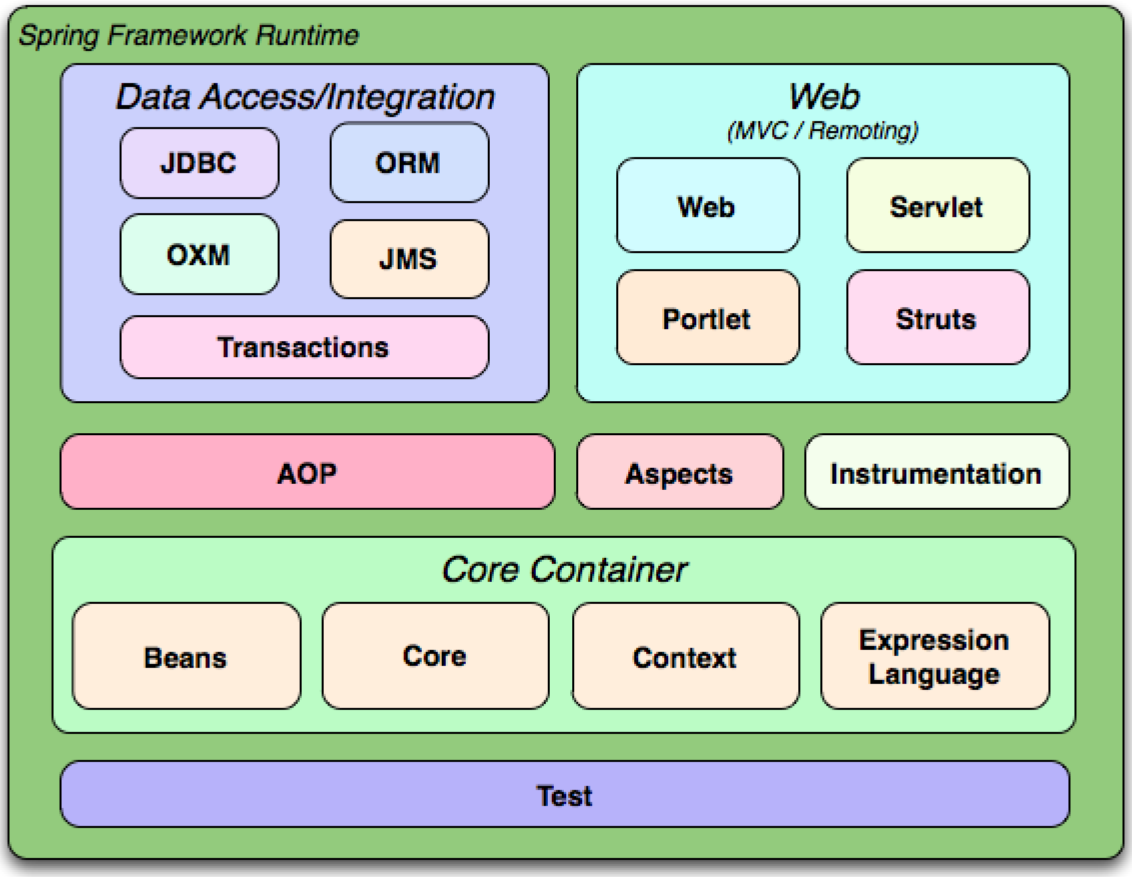
Part - C

1. Explain the Spring Framework components.

The Spring Framework:

The Spring Framework Inversion of Control (IoC) component addresses this concern by providing a formalized means of composing disparate components into a fully working application ready for use. The Spring Framework codifies formalized design patterns as first-class objects that you can integrate into your own application(s). Numerous organizations and institutions use the Spring Framework in this manner to engineer robust, maintainable applications.

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test, as shown in the following diagram.



Components of Spring

Core Container:

The *Core Container* consists of the Core, Beans, Context, and Expression Language modules.

The *Core and Beans* modules provide the fundamental parts of the framework, including the IoC and Dependency Injection features. The `BeanFactory` is a sophisticated implementation of the factory pattern. It removes the need for programmatic singletons and allows you to decouple the configuration and specification of dependencies from your actual program logic.

The *Context* module builds on the solid base provided by the *Core and Beans* modules: it is a means to access objects in a framework-style manner that is similar to a JNDI registry. The Context module inherits its features from the Beans module and adds support for internationalization (using, for example, resource bundles), event-propagation, resource-loading, and the transparent creation of contexts by, for example, a servlet container. The Context module also supports Java EE features such as EJB, JMX ,and basic remoting. The `ApplicationContext` interface is the focal point of the Context module.

The *Expression Language* module provides a powerful expression language for querying and manipulating an object graph at runtime. It is an extension of the unified expression language (unified EL) as specified in the JSP 2.1 specification. The language supports setting and getting property values, property assignment, method invocation, accessing the context of arrays, collections and indexers, logical and arithmetic operators, named variables, and retrieval of objects by name from Spring's IoC container. It also supports list projection and selection as well as common list aggregations.

Data Access/Integration

The *Data Access/Integration* layer consists of the JDBC, ORM, OXM, JMS and Transaction modules.

The JDBC module provides a JDBC-abstraction layer that removes the need to do tedious JDBC coding and parsing of database-vendor specific error codes.

The *ORM* module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis. Using the ORM package you can use all of these O/R-mapping frameworks in combination with all of the other features Spring offers, such as the simple declarative transaction management feature mentioned previously.

The OXM module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.

The Java Messaging Service (JMS) module contains features for producing and consuming messages.

The Transaction module supports programmatic and declarative transaction management for classes that implement special interfaces and for *all your POJOs (plain old Java objects)*.

Web

The *Web* layer consists of the Web, Web-Servlet, Web-Struts, and Web-Portlet modules.

Spring's *Web* module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context. It also contains the web-related parts of Spring's remoting support.

The *Web-Servlet* module contains Spring's model-view-controller (*MVC*) implementation for web applications. Spring's MVC framework provides a clean separation between domain model code and web forms, and integrates with all the other features of the Spring Framework.

The *Web-Struts* module contains the support classes for integrating a classic Struts web tier within a Spring application. Note that this support is now deprecated as of Spring 3.0. Consider migrating your application to Struts 2.0 and its Spring integration or to a Spring MVC solution.

The *Web-Portlet* module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

AOP and Instrumentation

Spring's *AOP* module provides an *AOP Alliance*-compliant aspect-oriented programming implementation allowing you to define, for example, method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated. Using source-level metadata functionality, you can also incorporate behavioral information into your code, in a manner similar to that of .NET attributes.

The separate *Aspects* module provides integration with AspectJ.
The *Instrumentation* module provides class instrumentation support and classloader implementations to be used in certain application servers.

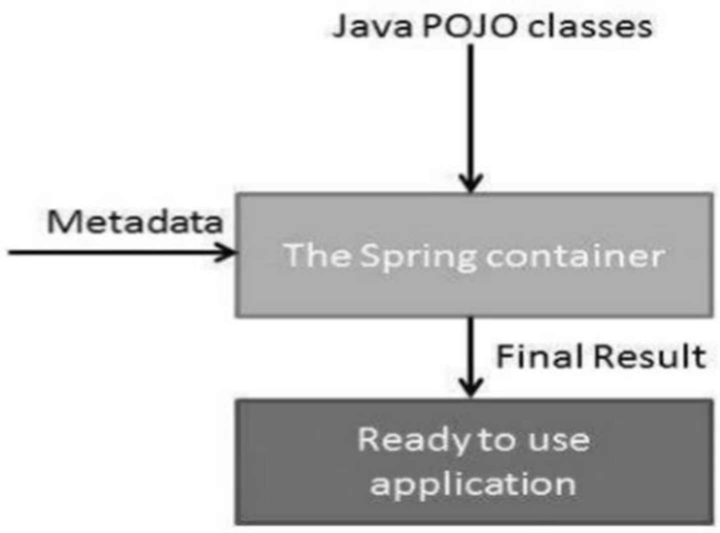
Test

The Test module supports the testing of Spring components with JUnit or TestNG. It provides consistent loading of Spring ApplicationContexts and caching of those contexts. It also provides mock objects that you can use to test your code in isolation.

2. Explain the details of Spring Container types

The Spring container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The Spring container uses DI to manage the components that make up an application. These objects are called Spring Beans.

The container gets its instructions on what objects to instantiate, configure, and assemble by reading the configuration metadata provided. The configuration metadata can be represented either by XML, Java annotations, or Java code. The following diagram represents a high-level view of how Spring works. The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.



Spring provides the following two distinct types of containers.

Container & Description

Spring BeanFactory Container

This is the simplest container providing the basic support for DI and is defined by the *org.springframework.beans.factory.BeanFactory* interface. The BeanFactory and related interfaces, such as BeanFactoryAware, InitializingBean, DisposableBean, are still present in Spring for the purpose of backward compatibility with a large number of third-party frameworks that integrate with Spring.

Spring ApplicationContext Container

This container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners. This container is defined by the *org.springframework.context.ApplicationContext* interface.

The ApplicationContext container includes all functionality of the BeanFactorycontainer, so it is generally recommended over BeanFactory. BeanFactory can still be used for lightweight applications like mobile devices or applet-based applications where data volume and speed is significant.

The differences between BeanFactory vs ApplicationContext in order to get a clear cutaway understanding of the spring IoC container which is as shown below in a tabular format below as follows:

The differences between BeanFactory vs ApplicationContext container

Feature	BeanFactory	ApplicationContext
Annotation Support	No	Yes
Bean Instantiation/Wiring	Yes	Yes
Internationalization	No	Yes
Enterprise Services	No	Yes
ApplicationEvent publication	No	Yes
Automatic BeanPostProcessor registration	No	Yes
Loading Mechanism	Lazy loading	Aggressive loading
Automatic BeanFactoryPostProcessor registration	No	Yes