CSE Learning Hub

Advanced JAVA Programming
(3160707)

DEGREE
COMPUTER Engineering | SEM: 6
Notes By: Harsh Porwal

Chapter:1

1. What are the differences between TCP and UDP socket? How are they created in JAVA?
2. Explain IP address.
3. Explain DNS and its working.
4. Explain Java.net package.
5. Explain Socket overview.
6. Compare sockets and server socket.
7. What is a Datagram socket? Explain it with an example.
8. Explain usages of InetAdress.

Chapter:2

1. Explain JBDC architecture.
2. Explain and Difference statement , prepared statement and callable statement.
3. What is dependency injection.
4. What is result interface? Write various method for resultSet interface. Write a code to update record using this interface.
5. What is JDBC? List out all various types of JDBC driver. Write a code snippet for each type of JDBC connection.
6. Explain Use of DatabaseMetaData with example.
7. Explain JDBC transaction management in details.
8. Write a JDBC program for banking application in which consider bank table with attributes AccountNo, CustomerName, Phone and Address, and perform followings:
    a. Insert two records using prepared statement.
    b. Display all the records

Chapter:3

1. List and explain servlet life cycle. Explain role of web container.
2. Explain use of servletconfig and servletcontext object with example.
3. Write a servlet code which reads the student details from web page and stores it in database.

4. What is request dispatcher? Difference between request dispatcher's forward () and include () method?
5. What is servlet filter? List the application.
6. Explain the configuration of filter using deployment descriptor.
7. What is session? List different ways to manage session and explain its working.
8. Explain cookie and functionality of cookie
9. Difference between GET and POST method.

Chapter: 4

1. List advantage of JSP and differentiate JSP vs Servlet.
2. Explain JSP life cycle.
3. What do you mean by MVC architecture? Explain its role in modern application and list advantages. OR Explain JSP architecture.
4. Explain use of JSP page directive tag with its attributes. OR Explain Page, Include, Taglib Directive.
5. Explain use of action tag with example.
6. Explain JSP implicit (in-built) object. Explain its use.
7. What is session object in JSP? Show its use with proper example. (148)
8. List JSTL core tags and explain any four with example.
9. Explain various tags in JSTL with example.
    i. Core, XML, SQL, Function
10. Explain difference between include directive and jsp: include action tag?
11. Which action tags are used to access the JavaBeans from a JSP page?
12. What is Expression Language EL in JSP explain with suitable example program
13. Develop a web application as following to demonstrate the use of JSTL SQL tag library. • Create a web page to store the registration detail (Name, email and contact number) of user in the database. • Create a web page for user to update registration detail in the database. • Create a web page to display the list of all registered users.
14. Explain JSP Exception Handling.

Chapter: 5

1. Explain JSF and its advantage and also architecture of JSF.
2. Explain JSF life cycle with diagram. OR Explain different phases of JSF request processing life cycle.
3. Explain JSF facelet tags and their use.
4. Discuss JSF converter tag.
5. List the JSF validation tags and its example.
6. Write a code snippet to show the use of JSF action event.
7. Write a short note on JSF Libraries: Prime Faces.
8. Explain JSF expression language with the help of suitable JSF program.

Chapter: 6

1. What is Hibernate? Explain advantage of using hibernate than SQL and JDBC.
2. Draw and Explain Hibernate architecture.
3. What is ORM? Explain object relational mapping in hibernate.
4. What is OR mapping? Explain the components of hibernate.cfg.xml file
5. Difference between HQL and SQL.

Chapter: 7

1. Explain architecture of spring MVC Framework and explain its features.
2. What is Spring IOC container.
3. Briefly explain spring bean life cycle.
4. Briefly What is Spring AOP? What are join points and cut points?

Chapter: 1

(1) What are the differences between TCP and UDP sockets? How are they created in JAVA?

TCP	UDP
Stands for transmission control protocol.	Stands for User datagram protocol.
TCP is connection-oriented protocol. Connection oriented means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting data.	UDP is a Datagram oriented protocol, because there is no overhead for opening a connection, maintaining a connection and terminating a connection. UDP is efficient for the broadcast and multicast types of network transmission.
TCP is reliable as it guarantees the delivery of data to the destination router.	The delivery of data to the destination cannot be guaranteed in UDP.
It provides extensive error-checking mechanisms.	It has only basic error checking mechanisms.
It provides flow control and acknowledgement of data.	It does not provide flow control.
An acknowledgment segment is present.	An acknowledgment segment is not present.
Retransmission of lost packets is possible in TCP.	Retransmission of lost packet is not possible in UDP.
TCP is slower than UDP.	UDP is faster, simpler, and more efficient than TCP.
TCP does not support broadcasting.	UDP supports broadcasting.
TCP is used by HTTP. HTPPs, FTP, SMTP.	UDP is used by DNS, DHCP, TFTP, SNMP.

- Example of creates TCP connection in JAVA.

```
try {
    Socket socket = new Socket("localhost", 8080);
    // Now you can send and receive data through the socket
} catch (IOException e) {
    // Handle the exception
}
```

- In this example, we create a new Socket object by specifying the host and port we want to connect to. Once the connection is established, we can send and receive data through the socket.

- Example of creates UDP connection in JAVA.

```
try {
    DatagramSocket socket = new DatagramSocket();
    InetAddress address = InetAddress.getByName("localhost");
    byte[] buffer = "Hello, world!".getBytes();
    DatagramPacket packet = new DatagramPacket(buffer, buffer.length, address, 8080);
    socket.send(packet);
} catch (IOException e) {
    // Handle the exception
}
```

- In this example, we create a new Datagram Socket object and use it to send a Datagram Packet containing a byte array of data. We specify the destination address and port in the datagram Packet constructor.

(2) Explain IP address and its classes.

- An IP address is a unique numerical identifier assigned to every device connected to a computer network that uses the internet protocol for communication. It allows devices to communicate with each other across a network by specifying the sources and destination of data packets.
- There is main two IP addresses: IPv4 and IPv6.
- IPv4 consists of 32-bit address is dotted-decimal notation (e.g. 192.168.0.1) and supports up to approximately 4.3 billion unique addresses. IPv4 is becoming exhausted because of the increasing number of devices connected to it.
- IPv6 is the newer and more advanced type of IP address. It consists of a 128-bit address in hexadecimal notation (e.g..2001:0db8:85a3:0000:0000:8a2e:0370:7334) and supports up to approximately 340 undecillion addresses.

Class	Range	Purpose
Class A	1 to 126	Very few large organizations use this class.
Class B	128 to 191	Medium Size organization use this class.
Class C	192 to 223	Relatively small organization use this class.
Class D	224 to 239	This class is used for multicast groups.
Class E	240 to 254	This class is reserved for experimental purpose.

(3) Explain DNS and Its Working.

- DNS stands for Domain name system.
- DNS is a directory service that provides mapping between the name of a host on the network and its numerical address.
- It is very difficult to remember numerical information, but it is simple to remember textual information. Consider that we want to access Facebook, then accessing it using IP address is not comfortable, rather if we have the address https://www.facebook.com then accessing and remembering textual information is very simple.

com	Commercial organizations.
gov	Government organizations.
edu	Educational organizations.
net	Network group.
org	Non-profit organizations.

❖ **Working:**

- A user enters a domain name (www.example.com) into their web browser.
- The browser sends a DNS query to a local DNS resolver, which is typically provided by the user's Internet Service Provider (ISP).
- If the local DNS resolver has the IP address for the requested domain name cached in its memory, it returns the IP address to the browser, and the browser can connect to the website directly.
- If the local DNS resolver does not have the IP address for the requested domain name cached, it sends a query to one of the root DNS servers.
- The root DNS server responds with a referral to a Top-Level Domain (TLD) server. For example, if the requested domain name is www.example.com, the root DNS server would refer the query to the .com TLD server.
- The TLD server responds with a referral to a Domain Name Server (DNS) responsible for the next level of the domain hierarchy.
- The DNS server responsible for the domain name returns the IP address for the requested domain name to the local DNS resolver.
- The local DNS resolver caches the IP address and returns it to the browser, which can now connect to the website.

(4) Explain Java .net Packages.

- The java.net package in Java provides classes and interfaces that enable network programming, such as communication between clients and servers over the internet. This package contains a variety of classes that enable you to create and manage network connections, send and receive data over those connections, and work with URLs and URIs.

URL	A class that represents a Uniform Resource Locator (URL), which is used to identify resources on the internet, such as web pages, images, and other types of files.
URI	A class that represents a Uniform Resource Identifier (URI), which is a more general form of a URL that can be used to identify any type of resource.
URLConnection	An abstract class that represents a connection to a resource identified by a URL.
Socket	A class that represents a client-side socket, which can be used to establish a connection to a server over a network.
ServerSocket	A class that represents a server-side socket, which can be used to listen for incoming connections from clients.
DatagramSocket	A class that provides support for sending and receiving datagrams, which are small packets of data that can be sent over a network.
InetAddress	A class that represents an IP address, which is used to identify networked devices on the internet.
Proxy	A class that represents a proxy server, which can be used to provide an intermediary between clients and servers on a network.

(5) Explain Java Socket Overview.

- Java socket programming is used for communication between the applications running on different JRE.
- Java socket programming can be connection-oriented or connection less.
- Socket and Server Socket classes are used for connection-oriented socket programming and Datagram socket and Datagram Packet classes are used for connection-less socket programming.
- The java.net.Socket class provides methods for creating client-side sockets that connect to a server, as well as server-side sockets that listen for incoming client connections.
- Once a connection is established between a client and server, data can be sent and received using the Input Stream and Output Stream objects associated with the socket.
- Here's an overview of the basic steps involved in using sockets in Java:
  - **Create a Socket object:** You can create a client-side socket by passing the IP address and port number of the server to the Socket constructor. To create a server-side socket, you call the ServerSocket constructor with the port number you want to listen on.
  - **Establish a connection:** For client-side sockets, you can establish a connection to the server using the connect() method. For server-side sockets, you can wait for incoming client connections using the accept() method.
  - **Send and receive data:** Once a connection is established, you can use the InputStream and OutputStream objects associated with the socket to send and receive data.
  - **Close the socket:** When you're done using the socket, you should close it using the close() method.

(6) Difference between Socket and ServerSocket.

Socket	Server Socket
Socket used at Client side.	Server Socket used on the server side.
It is used for sending the request to the server.	It is used for listening to the client.
This class encapsulates the behaviour of the active side.	This class encapsulates the behaviour of the passive side.
Establish connection with connect()	Establish connection with listen()
Socket s = new Socket (“localhost”,1111);	ServerSocket ss = new ServerSocket(1111);

(7) Explain Datagram Socket with example.

- Datagrams are collection of information sent from one device to another device via the established network.
- When the datagram is sent to the targeted device, there is no assurance that it will reach to the target device safely and completely. It may get damaged or lost in between.
- Likewise, the receiving device also never know if the datagram received is damaged or not. The UDP protocol is used to implement the datagrams in Java.
- In Java, a DatagramSocket is a socket that provides a connectionless, packet-oriented communication channel. It is used for sending and receiving datagrams, which are packets of data that are sent over a network.
- A DatagramSocket works by sending and receiving datagrams through a network interface, without establishing a separate connection for each packet. This makes it more efficient for applications that need to send a large number of small packets, such as real-time video or audio streaming.

```java
import java.net.*;

public class DatagramExample {
    public static void main(String[] args) throws Exception {
        // Create a DatagramSocket to send and receive datagrams
        DatagramSocket socket = new DatagramSocket();

        // Create a DatagramPacket with the data to send
        byte[] data = "Hello, world!".getBytes();
        InetAddress address = InetAddress.getByName("localhost");
        int port = 8000;
        DatagramPacket packet = new DatagramPacket(data, data.length, address, port);

        // Send the datagram packet
        socket.send(packet);

        // Receive a datagram packet
        byte[] buffer = new byte[1024];
        DatagramPacket receivedPacket = new DatagramPacket(buffer, buffer.length);
        socket.receive(receivedPacket);

        // Display the received data
        String receivedData = new String(receivedPacket.getData(), 0, receivedPacket.getLength());
        System.out.println("Received data: " + receivedData);

        // Close the socket
        socket.close();
    }
}
```

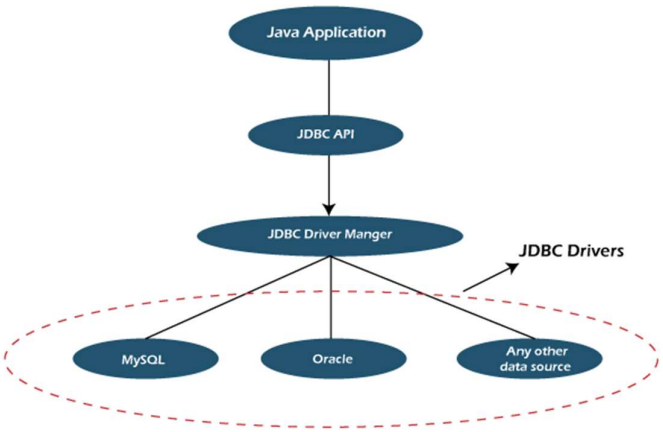
(8) Explain the usages of Inetaddress.

- Java InetAddress class represents an IP address. The java.net.IneAddress class provides methods to get the IP of any host name for example www.harshporwal.me, www.google.com, etc.
- An IP address is represented by 32-bit or 128-bit unsigned number. An instance of InetAddress represents the IP address with its corresponding host name.
- There are two types of addresses: Unicast and Multicast.
  The Unicast is an identifier for a single interface whereas Multicast is an identifier for a set of interfaces.
- It can be used for various purposes related to networking, such as:
  - **Resolving domain names:** InetAddress provides a method called *getByName()* that can be used to resolve a domain name to an IP address. For example, InetAddress.getByName("www.google.com") returns the IP address of the Google website.
  - **Retrieving local host information:** InetAddress provides a method called *getLocalHost()* that returns an InetAddress object representing the local host. This can be useful for finding out the IP address of the current machine.
  - **Checking reachability:** InetAddress provides a method called *isReachable()* that can be used to check whether a particular IP address is reachable over the network. This can be useful for testing network connectivity.
  - **Creating socket connections:** InetAddress can be used to create socket connections to other machines on the network. For example, *InetAddress.getByName("192.168.0.1")* returns the InetAddress object representing the machine with IP address 192.168.0.1, which can be used to create a socket connection to that machine.

Chapter: 2

(1) Explain JDBC Architecture.

- JDBC stands for Java Database Connectivity.
- It is a java API that allows Java applications to interact with relational database.
- JDBC API is a set of classes, interfaces and exception used for establishing connection with data source. This JDBC API is defined in the java.sql and java.sqlpackages.



- **Application:** This layer represents the application or program that needs to interact with the database. It uses the JDBC API to send SQL statements to the database and process the results.
- **JDBC API:** This is the core of the JDBC architecture. It provides a set of classes and interfaces that allow the application to interact with the database. The API is divided into two parts: the JDBC Manager and the JDBC Driver.
- **JDBC Manager:** This is the part of the API that manages the JDBC driver. It loads the driver, establishes a connection to the database, and sends SQL statements to the database.
- **JDBC Driver:** This is the component that translates the JDBC API calls into database-specific calls. There are four types of JDBC drivers: Type 1, Type 2, Type 3, and Type 4.
- **Database:** This is the actual database with which the application needs to interact. It can be any relational database that supports JDBC.

(2) Explain Prepare and Callable Statement with example.

Statement	PreparedStatment	CallableStatment
It is used to execute normal SQL queries.	It I used to execute parameterized or dynamic SQL queries.	It is used to call the stored procedure.
It is preferred when a particular SQL query is to be executed only once.	It is preferred when a particular query is to be executed multiple times.	It is preferred when the stored procedures are to be executed,
You cannot pass the parameters to SQL query using this interface.	You can pass the parameters to SQL query at run time using this interface.	You can pass 3 types of parameters using this interface. They are: IN, OUT, and IN OUT
This interface is mainly used for DDL statements like CREATE, ALTER, DROP, etc.	It is used for any kind of SQL queries which are to be executed multiple times.	It is used to execute stored procedures and functions.
Performance of this is very low.	Performance of this is better than statement.	Performance of this is high.

- **Prepared Statement:** It is used to execute SQL queries that have parameters. It is used when you need to execute same SQL statement repeatedly with different values for the parameter.

```
String sql = "INSERT INTO customers (name, email, phone) VALUES (?, ?, ?)";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, "John Doe");
pstmt.setString(2, "johndoe@example.com");
pstmt.setString(3, "123-456-7890");
int rowsInserted = pstmt.executeUpdate();
```

- **Callable Statement:** It is used to execute stored procedures in a database. It is used when you need to execute a stored procedures that returns one or more result set output parameters. Syntax for calling a stored procedure using this is different from that of a regular SQL statement.

```
String sql = "{CALL get_customer_details(?)}";
CallableStatement cstmt = conn.prepareCall(sql);
cstmt.setInt(1, 12345);
ResultSet rs = cstmt.executeQuery();
while (rs.next()) {
    // process the result set
}
```

(3) Explain Dependency injection with example.

- Dependency injection is a design pattern used in advanced Java programming to manage dependencies between objects in a more flexible and maintainable way.
- The pattern is used to remove the dependency of one class on another by passing the dependency to it, instead of having it create the dependency itself.
- Dependency injection allows you to define a set of dependencies required by a class, and then inject those dependencies into the class at runtime.
- This makes your code more modular and easier to maintain, since you can change the behaviour of your classes without having to modify their code directly.

- **Constructor Injection:** In this type of dependency injection, dependencies are provided to the class through the constructor. The class declares its dependencies as parameters of the constructor and the dependencies are injected at the time of object creation. This ensures that the class has all the necessary dependencies to function properly.

- **Setter Injection:** In this type of dependency injection, dependencies are provided to the class through setter methods. The class declares its dependencies as private fields and provides public setter methods to set those fields. The dependencies are injected using the setter methods after the object is created.

- **Field Injection:** In this type of dependency injection, dependencies are provided to the class through public fields. The dependencies are injected directly into the public fields after the object is created.

Example:

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

public class Car {
    private final Engine engine;

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

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

- In the above code, we have **Engine** class which has a single method to start the engine. The **Car** class has a constructor that requires an **Engine** object to be passed in. The **start** method of the **Car** class uses the **Engine** object to start the engine and print a message indicating that the car has started.

```
public class Main {
    public static void main(String[] args) {
        Engine engine = new Engine();
        Car car = new Car(engine);
        car.start();
    }
}
```

- In the above code, we have created an instance of the **Engine** class and an instance of the **Car** class. We have passed the **Engine** instance to the **Car** constructor using constructor injection. We then call the start method of the **Car** class, which uses the **Engine** object to start the engine and print a message indicating that the car has started.
- By using constructor injection, we have decoupled the **Car** class from its dependency on the **Engine** class. We can easily replace the **Engine** implementation without modifying the **Car** class. We can also easily test the **Car** class in isolation by providing a mock implementation of the **Engine** class.

(4) Explain ResultSet with various methods.

- This interface is an important interface which is used to access the database table with general width and unknow length.
- The table rows are retrieved in sequence using ResultSet object. Within a row its column values can be accessed in any order.
- A ResultSet maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row.
- ResultSet object can be created using **executeQuery ()** method.

- **Navigation Methods:**
  - **next ():** Moves the cursor to the next row in the result set. Returns true if the next row exists, false if it does not.
  - **previous ():** Moves the cursor to the previous row in the result set. Returns true if the previous row exists, false if it does not.
  - **first ():** Moves the cursor to the first row in the result set.
  - **last ():** Moves the cursor to the last row in the result set.
  - **absolute (int row):** Moves the cursor to the specified row in the result set.

- **Reading Methods:**
  - **getInt (String columnLabel):** Retrieves the value of the specified column as an integer.
  - **getString (String columnLabel):** Retrieves the value of the specified column as a string.
  - **getDouble (String columnLabel):** Retrieves the value of the specified column as a double.
  - **getDate (String columnLabel):** Retrieves the value of the specified column as a java.sql.Date object.

- **Update Methods:**
  - **updateInt (String columnLabel, int value):** Updates the value of the specified column to the given integer value.
  - **updateString (String columnLabel, String value):** Updates the value of the specified column to the given string value.
  - **updateDouble (String columnLabel, double value):** Updates the value of the specified column to the given double value.
  - **updateDate (String columnLabel, java.sql.Date value):** Updates the value of the specified column to the given java.sql.Date value.
- **Example:**

```
try (Connection conn = DriverManager.getConnection(url, user, password);
     Statement stmt = conn.createStatement();
     ResultSet rs = stmt.executeQuery("SELECT * FROM users")) {
    while (rs.next()) {
        int id = rs.getInt("id");
        String name = rs.getString("name");
        double salary = rs.getDouble("salary");
        java.sql.Date hireDate = rs.getDate("hire_date");
        System.out.println("ID: " + id + ", Name: " + name + ", Salary: " +
        salary + ", Hire Date: " + hireDate);

        rs.updateString("name", "John");
        rs.updateDouble("salary", salary * 1.1);
        rs.updateRow();
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

- In the above code, we have a ResultSet object obtained from executing a SELECT query against a user's table.
- We use the next() method to move the cursor to the next row in the result set, and then use the various get methods to read the values of the columns.
- We then use the update methods to update the values of the name and salary columns, and finally call the updateRow () method to save the changes to the database.

(5) Explain Various types of JDBC drivers.

- JDBC is java AP that provides a standard way to connect to and interact with databases from java applications. In order to use JDBC, you need a JDBC driver, which is a software component that allows java application to communicate with a database.
- JDBC are responsible for translating Java methods call into the appropriate database-specific protocol.
- JDBC classes are contained in the Java package **java.sql** and **javax.sql**.
- It helps you into three programming activity:
    - Connect to data database.
    - Send queries and update statements to the database.
    - Retrieve and process the results received from database in answer to your query.

- There are 4 types of JDBC drivers:
    - Type-1 driver or JDBC-ODBC bridge driver
    - Type-2 driver or Native-API driver
    - Type-3 driver or Network Protocol driver
    - Type-4 driver or thin driver

- **(Type - 1) JDBC-ODBC Bridge Driver:** This driver is built on top of the ODBC (Open Database Connectivity) API, which is a platform-independent API for accessing data stores. The JDBC-ODBC bridge driver is a type 1 driver that provides JDBC access via ODBC drivers.
    - **Advantages:** Easy to install, work with different devices.
    - **Disadvantages:** Relatively slow, not suitable for production environment.

- **(Type - 2) (Thick Driver) Native-API Driver:** This driver is a type 2 driver that uses the database-specific native API to communicate with the database.
- It is called thick driver because this driver uses native API or partly Java driver to communicate to client's database.
    - **Advantages:** Provide better performance than JDBC-ODBC.
    - **Disadvantages:** Platform specific and may not work with all databases.

- **(Type - 3) Network Protocol Driver:** This driver is a type 3 driver that uses a middle-tier server to communicate with the database.
    - **Advantages:** Platform independent, Work with many different databases, provides good performance.
    - **Disadvantages:** Require installation of middleware server

- **(Type - 4) JDBC Driver or Thin Driver:** This driver converts the JDBC call to network protocol used by the database directory so that the client application can directly communicate to the data base server.
- It Is also called thin driver because this driver is written in pure java, its portable across all platform, which mean you can use same JAR file to connect to MySQL even if your JAVA program is running on windows or Linux.
    - **Advantages:** Easy to use, Provides better performance.
    - **Disadvantages:** Specific to particular database may not work with different databases, most expensive, Require separate license for each database.

(6) Explain DatabaseMetaData with example.

- Data about data is knows as metadata.
- The DatabaseMetaData interface provides methods to get information about the database you have connected with like, database name, database driver version, maximum column length, etc.

Method	Description
getDriverName ()	Retrieves the name of the current JDBC driver
getDriverVersion ()	Retrieves the version of the current JDBC driver
getUserName ()	Retrieves the user name.
getDatabaseProductName ()	Retrieves the name of the current database.
getDatabaseProductVersion ()	Retrieves the version of the current database.
getNumericFunctions ()	Retrieves the list of the numeric functions available with this database.
getStringFunctions ()	Retrieves the list of the numeric functions available with this database.
getSystemFunctions ()	Retrieves the list of the system functions available with this database.
getURL ()	Retrieves the URL for the current database.
supportsSavepoints ()	Verifies weather the current database supports save points
supportsStoredProcedures ()	Verifies weather the current database supports stored procedures.
supportsTransactions ()	Verifies weather the current database supports transactions.

```java
import java.sql.*;

public class DatabaseMetaDataExample {
   public static void main(String[] args) throws Exception {
      // Load the MySQL driver
      Class.forName("com.mysql.jdbc.Driver");

      // Connect to the database
      Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydatabase",
      "username", "password");

      // Get the metadata
      DatabaseMetaData metadata = con.getMetaData();

      // Print some information about the database
      System.out.println("Database Name: " + metadata.getDatabaseProductName());
      System.out.println("Database Version: " + metadata.getDatabaseProductVersion());
      System.out.println("Driver Name: " + metadata.getDriverName());
      System.out.println("Driver Version: " + metadata.getDriverVersion());

      // Close the connection
      con.close();
   }
}
```

- In this example, we first load the MySQL driver and connect to the database using the **DriverManager.getConnection ()** method. We then obtain the DatabaseMetaData object using the **Connection.getMetaData ()** method.
- We use the various methods of the DatabaseMetaData interface to obtain information about the database, such as its name, version, and the available schema names. Finally, we close the connection using the **Connection.close ()** method.
- Using the DatabaseMetaData interface, you can obtain a wealth of information about a database that can be useful in developing database applications.

(7) Explain Transaction Management and Object Serialization.

➢ Transaction Management:

- Transaction management is the process of ensuring data consistency and integrity in a database when multiple transaction are occurring simultaneously.
- A transaction is a sequence of operation that must be completed as a single unit of work, either all operations completed successfully or the entire transaction is rolled back.
- Transaction management is used to provide data integrity, correct application semantics and consistent view of data during concurrent access.
- Transaction management describe using the ACID properties.
  - **A**tomicity means either all operation gets successful or none.
  - **C**onsistency means the database must be consistent state.
  - **I**solation is a property which indicates isolation of one transaction from other.
  - **D**urability means after transaction been committed, it will remain as it is.

Method	Description
Void setAutoCommit ()	The true value indicates that each transaction is committed automatically. By default, it is true. It can be set false for manual transaction.
void commit ()	It is used for committing the transaction.
Void rollback ()	It is used for cancelling the transaction.

```java
import java.sql.*;

public class TransactionExample {
   public static void main(String[] args) throws Exception {
      // Load the MySQL driver
      Class.forName("com.mysql.jdbc.Driver");

      // Connect to the database
      Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydatabase",
      "username", "password");

      // Set the auto-commit to false
      con.setAutoCommit(false);

      try {
         // Perform some database operations as part of a transaction
         Statement stmt = con.createStatement();
         stmt.executeUpdate("INSERT INTO mytable (id, name) VALUES (1, 'John')");
         stmt.executeUpdate("UPDATE mytable SET name = 'Jane' WHERE id = 1");

         // Commit the transaction
         con.commit();
      } catch (SQLException e) {
         // Roll back the transaction if an error occurs
         con.rollback();
      } finally {
         // Set the auto-commit back to true and close the connection
         con.setAutoCommit(true);
         con.close();
      }
   }
}
```

- In this example, we first load the MySQL driver and connect to the database using the **DriverManager.getConnection()** method.
- We set the auto-commit to false so that the transactions are not committed automatically after each statement execution.
- Then we perform some database operations as part of a transaction and use a try-catch block to catch any SQL exceptions that may occur.
- If an exception occurs, we roll back the transaction using the **Connection.rollback ()** method. Finally, we set the auto-commit back to true and close the connection.

➢ **Object Sterilization:** It is a process of converting an object into a stream of bytes that can be saved to a file or sent over a network. The reverse process of converting stream bytes back into object is called deserialization. It is used to save the state of the object. It is useful for helping the object to travel across network.

(8) Write a JDBC program for banking application in which consider bank table with attributes Account No, Customer Name, Phone and Address and perform followings:

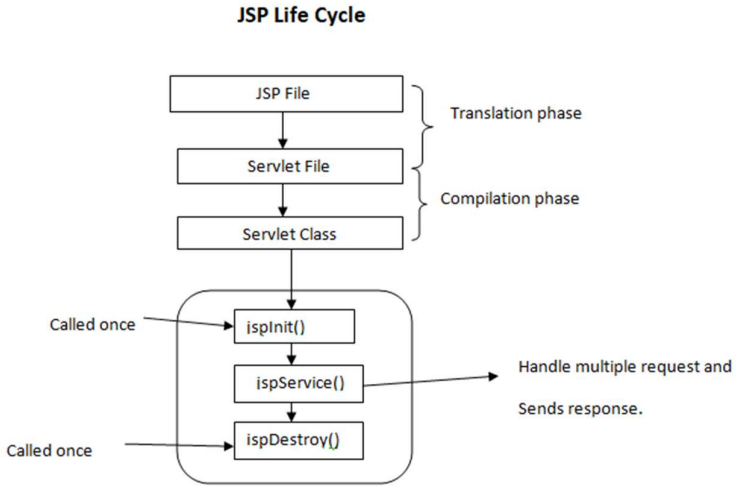
- **Insert two records using prepared statement.**
- **Display all the records.**

```java
import java.sql.*;

public class BankApplication {
    public static void main(String[] args) {
        try {
            // Load the MySQL driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Create a connection to the database
            Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/bank", "USERNAME", "PASSWORD");

            // Insert two records using prepared statement
            insertRecord(connection, 1001, "John Doe", "1234567890", "123 Main St");
            insertRecord(connection, 1002, "Jane Smith", "0987654321", "456 Oak St");
            displayRecords(connection);
            connection.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void insertRecord(Connection connection, int accountNo, String customerName, String phone, String address) throws SQLException {
        String insertQuery = "INSERT INTO bank (AccountNo, CustomerName, Phone, Address) VALUES (?, ?, ?, ?)";

        PreparedStatement statement = connection.prepareStatement(insertQuery);
        statement.setInt(1, accountNo);
        statement.setString(2, customerName);
        statement.setString(3, phone);
        statement.setString(4, address);
        statement.executeUpdate();
    }

    private static void displayRecords(Connection connection) throws SQLException {
        String selectQuery = "SELECT * FROM bank";
        Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery(selectQuery);
        while (resultSet.next()) {
            int accountNo = resultSet.getInt("AccountNo");
            String customerName = resultSet.getString("CustomerName");
            String phone = resultSet.getString("Phone");
            String address = resultSet.getString("Address");

            System.out.println("Account No: " + accountNo + ", Customer Name: " + customerName + ", Phone: " + phone + ", Address: " + address);
        }
    }
}
```

Chapter: 3

(1) List and explain servlet life cycle. Explain role of web container.

➢



- The entire life cycle of a servlet is managed by the Servlet container which uses the **javax.servlet.Servlet** interface to understand the servlet object and manage.
- The servlet life cycle consists of three main phases:

➢ **Initialization (init ()):**

- When a servlet is loaded into the web container for the first time, the container initializes it by calling the servlet's **init ()** method.
- This method is called only once during the lifetime of a servlet.
- It is used for performing any initialization tasks, such as setting up database connections, initializing resources, or configuring the servlet.
- If an error occurs during initialization, the container will not load the servlet, and it will not be available to handle requests.

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

- **Request handling (service ()):**
  - After the servlet has been initialized, the container starts calling its **service ()** method to handle incoming HTTP requests.
  - The **service ()** method reads the request and generates a response using the appropriate business logic.
  - The container passes two objects to the **service ()** method: an object representing the client's request (an instance of the **HttpServletRequest** class) and an object representing the server's response (an instance of the **HttpServletResponse** class).
  - The **service ()** method can handle different types of HTTP requests, such as GET, POST, PUT, and DELETE, by examining the HTTP method specified in the request

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

- **Destruction (destroy ()):**
  - When the container decides to shut down a servlet, it calls the servlets **destroy ()** method.
  - This method is called only once during the lifetime of a servlet, just before the servlet is removed from memory.
  - It is used for releasing any resources held by the servlet, such as database connections or file handles.

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

- Web Container play crucial role in servlet lifecycle. It is used for
  - Loading the servlet into memory when first request is received.
  - Calling the servlet init () method to perform any initialization tasks.
  - Creating new thread to handle each incoming request and passing the request and response object to the service () method.
  - Managing the lifecycle of servlet, including shutting down and feeding up memory when it is no longer needed.

(2) Explain difference of ServletConfig and ServletContext object with example.

ServletConfig	ServletContext
ServletConfig object is one per servlet class.	ServletContext object is global to the entire web application.
Object of ServletConfig will be created during the initialization process of the servlet.	Object of ServletContext will be created at the time of web application deployment
We have to give the request explicitly in order to create the ServletConfig object for the first time.	ServletContext object can be available even before giving the first request.
Scope: As long as a servlet is executing, the ServletConfig object will be available, it will be destroyed once the servlet execution is completed.	Scope: As long as a web application is executing, the ServletContext object will be available, and it will be destroyed once the application is removed from the server.
ServletConfig object is used while only one servlet requires information shared by it.	ServletContext object is used while application requires information shared by it.
getServletConfig () method is used to obtain ServletConfig object.	getServletContext () method is used to obtain ServletContext object
In web.xml — <init-param> tag will be appeared under <servlet-class> tag.	In web.xml — <context-param> tag will be appeared under <web-app> tag.
Ex. Suppose, one is building a job portal and desires to share different email ids (which may get change over time) to recruiter and job applicant. So, he decides to write two servlets one for handling recruiter's request and another one for the job applicant.	**Ex.** Suppose, the name of one's job portal is "NewWebsite.tg". Showing the website name at the top of webpages delivered by different servlets, one needs to store the website name in every servlet inviting redundancy. Since the information shared by ServletContext can be accessed by every Servlet, it is better to go with ServletContext and retrieve the website name using getServletContext.getInitParameter("Name") whenever required

(3) What is request dispatcher? Difference between request dispatcher's forward () and include () method?

- In Java Servlets, the RequestDispatcher interface provides a way to forward a request from one servlet to another or to include the response from another servlet in the response of the current servlet. The RequestDispatcher interface is implemented by the ServletRequest object, and is used to interact with the web container to forward or include a request.
- The two methods provided by the RequestDispatcher interface are:
    - **Forward (ServletRequest request, ServletResponse response):** This method forwards the request from one servlet to another servlet or JSP page. The request is sent to the new servlet or JSP page, and the response is sent back to the client from the new servlet or JSP page. The original servlet is not allowed to send any response after forwarding the request.
    - **Include (ServletRequest request, ServletResponse response):** This method includes the response from another servlet or JSP page in the response of the current servlet. The response from the included servlet or JSP page is appended to the current response, and the current servlet can continue processing the request and send its own response.

Forward ()	Include ()
This client request will get forwarded to the forwarding page.	This method another
We have to give the request explicitly in order to create this object for first time.	This object can be available even before giving the first request.
This will be created during initialization process of the servlet.	This is created at the time of web application development.
The client request will get forwarded to the forwarding page.	By this method another file will be included in the current file.
This is faster to execute.	This is slower to execute.

(4) What is servlet filter? List the application.

- In Java Servlets, a filter is a Java class that is used to intercept and process requests and responses before they are sent to the servlet or JSP page that handles them.
- Filters are used to perform various tasks, such as authentication, logging, URL rewriting, data compression, and encryption.
- Filters are defined in the web.xml deployment descriptor file, and can be associated with a specific URL pattern, servlet, or JSP page. When a request is received by the web container, it is first passed through the filter chain, which is a series of filters that are defined in the deployment descriptor file.
- **The key advantages of using servlet filters are:**
  - **Reusability:** Filters can be reused across multiple servlets and JSP pages, reducing the amount of duplicate code in the application.
  - **Modularity:** Filters can be added or removed from the filter chain without affecting the servlet or JSP page, making it easy to modify the behaviour of the application.
  - **Separation of concerns:** Filters allow different parts of the application to be responsible for different aspects of request processing, making the code easier to maintain.
- **Some common applications of servlet filters are:**
  - **Authentication:** Filters can be used to authenticate users before they are granted access to a servlet or JSP page. The filter can redirect the user to a login page if they are not authenticated.
  - **Logging:** Filters can be used to log requests and responses to a file or database, allowing administrators to monitor the application for errors and security breaches.
  - **Compression:** Filters can be used to compress the response data before it is sent to the client, reducing the amount of data that needs to be transferred over the network.
  - **Encryption:** Filters can be used to encrypt the request and response data, providing an additional layer of security for sensitive information.
  - **URL rewriting:** Filters can be used to rewrite the URLs of the request, making them more user-friendly or search engine-friendly.

(5) Explain Configuration of filter using deployment descriptor.

- In java Servlet, a filter can be configured using the **web.xml** deployment descriptor. The **web.xml** file is located to in the **WEB-INF** directory of the web application and contains configuration information for entire application.
- **To configure a filter using the web.xml file, you need to perform the following steps:**
- **Define the filter:** You can define a filter using the **<filter>** tag. The **<filter>** tag has two child elements: **<filter-name>** and **<filter-class>.** The **<filter-name>** element is used to specify a name for the filter, and the **<filter-class>** element is used to specify the fully qualified name of the Java class that implements the filter.

```
<filter>
    <filter-name>myFilter</filter-name>
    <filter-class>com.example.MyFilter</filter-class>
</filter>
```

- **Map the filter:** You can map the filter to a servlet or a URL pattern using the **<filter-mapping>** tag. The **<filter-mapping>** tag has two child elements: **<filter-name>** and **<url-pattern>.** The **<filter-name>** element is used to specify the name of the filter, and the **<url-pattern>** element is used to specify the URL pattern or servlet name that the filter should be applied to.

```
<filter-mapping>
    <filter-name>myFilter</filter-name>
    <url-pattern>/myServlet/*</url-pattern>
</filter-mapping>
```

- **Set initialization parameters (optional):** You can set initialization parameters for the filter using the **<init-param>** tag. The **<init-param>** tag has two child elements: **<param-name>** and **<param-value>.** The **<param-name>** element is used to specify the name of the parameter, and the **<param-value>** element is used to specify the value.

```
<filter>
    <filter-name>myFilter</filter-name>
    <filter-class>com.example.MyFilter</filter-class>
    <init-param>
        <param-name>myParam</param-name>
        <param-value>myValue</param-value>
    </init-param>
</filter>
```

(6) What is session? Explain different ways to manage session. And explain its working.

- In web development, a session is a way to associate a set of data with a particular user across multiple HTTP requests. When a user interacts with a web application, a session can be used to store information about that user and maintain state across requests.
- **There are several ways to manage sessions in Java web applications:**

- **Cookies:** Cookies are a common way to manage sessions in web applications. A cookie is a small text file that is stored on the client-side and sent with each subsequent request to the server. Cookies can be used to store a unique identifier that is associated with a user's session, allowing the server to identify the user across multiple requests.
- **URL Rewriting:** URL Rewriting is another way to manage sessions. In this approach, a unique session identifier is added to the URL of each page in the application. The server can then use this identifier to retrieve session
- **Hidden Form Fields:** Hidden Form Fields are used to store session data in an HTML form. When the user submits the form, the session data is sent to the server along with the other form data.
- **HttpSession Object:** The HttpSession object is a built-in mechanism for managing sessions in Java web applications. The HttpSession object is stored on the server-side and can be used to store any type of data associated with a particular session.

- **The working of sessions involves the following steps:**

- When a user first accesses a web application, a unique session ID is created by the server.
- The session ID is then stored on the client-side (e.g., in a cookie or in the URL).
- Whenever the user interacts with the web application, the session ID is sent to the server along with the request.
- The server uses the session ID to retrieve the corresponding HttpSession object (or other session data) and can then retrieve or update session data as needed.
- When the user logs out or the session times out, the session data is destroyed.

(7) Explain Cookie and its functionality.

➢ **Cookies:** -

- Cookies is a small piece of information which is stored in a client browser. It is used to identify the user.
- Cookie is created at server side and saved to client browser. Each time when client send request to the server, cookie is embedded with request.
- A cookie store name and textual value. A cookie is created by some software system on the server.
- In every HTTP communication between browser and server a header is included. The header store the information about the message and also contains the cookies.
- Cookie in a website can be used to Remember the visitor so that the server can customize the site for the user.
- It is also used in shopping cart implementation so that items added to the cart will remain there even if the user leaves the site and then come back later.
- It is also frequently used to keep track user log-in activity.

➢ **Working: -**

- **Step 1:** User makes 1st request on website.
- **Step 2:** Page set cookies value as part of response.
- **Step 3:** HTTP response contains cookies in header.
- **Step 4:** Browser save the cookies in a text file and associate this file with website.
- **Step 5:** User makes another request to the website.
- **Step 6:** Browser reads cookie value from file for each subsequent request for website.
- **Step 7:** Cookie values travel in every subsequent HTTP request for that domain.
- **Step 8:** Server for website retrieves these cookie values from request header and uses them to customize the response.

➢ There are two types of cookies: **session cookies and persistent cookies.**

- A session cookie has no expiry date.
- Persistent cookies have an expiry date.

(8) Write a servlet code which reads the student details from web page and stores it in database.

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class StudentServlet extends HttpServlet {

   public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

      // Set up the database connection
      String dbURL = "jdbc:mysql://localhost:3306/mydatabase";
      String dbUser = "myuser";
      String dbPass = "mypassword";
      Connection conn = null;
      PreparedStatement stmt = null;
      ResultSet rs = null;

      try {
         // Get the form data from the request object
         String name = request.getParameter("name");
         String address = request.getParameter("address");
         String phone = request.getParameter("phone");

         // Insert the student record into the database
         Class.forName("com.mysql.jdbc.Driver");
         conn = DriverManager.getConnection(dbURL, dbUser, dbPass);
         stmt = conn.prepareStatement("INSERT INTO students (name, address, phone) VALUES (?, ?, ?)");
         stmt.setString(1, name);
         stmt.setString(2, address);
         stmt.setString(3, phone);
         int rows = stmt.executeUpdate();

         // Display a success message
         response.setContentType("text/html");
         PrintWriter out = response.getWriter();
         out.println("<html>");
         out.println("<head>");
         out.println("<title>Student Added</title>");
         out.println("</head>");
         out.println("<body>");
         out.println("<h1>Student Added Successfully</h1>");
         out.println("<p>Name: " + name + "</p>");
         out.println("<p>Address: " + address + "</p>");
         out.println("<p>Phone: " + phone + "</p>");
         out.println("</body>");
         out.println("</html>");

      } catch (Exception e) {
         // Handle any errors that occur
         e.printStackTrace();
      } finally {
         // Close the database connection
         try {
            if (rs != null) rs.close();
            if (stmt != null) stmt.close();
            if (conn != null) conn.close();
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
   }
}
```

(9) Difference between GET and POST.

GET	POST
In GET method we cannot send large amount of data.	In POST method we can send large amount of data.
GET request is less secure.	POST method is more secure.
GET method are stored in browser history.	POST method is not stored in browser history.
GET request can be bookmarked.	POST request cannot be bookmarked.
Not preferred to send sensitive information.	Used to send sensitive information.
It has length restrictions.	It does not have any length restriction.

Chapter: 4

(1) List advantages of JSP and Differentiate JSP and Servlet.

- Simplicity: JSP pages are easy to understand and use, especially for developers familiar with HTML, JavaScript, and Java.
- Reusability: JSP pages can be reused across multiple web pages, improving code maintainability and reducing the need for redundant code.
- Custom Tag Libraries: JSP allows developers to create custom tag libraries, which can encapsulate complex business logic and make it easier to develop and manage large-scale web applications.
- Separation of Concerns: JSP allows for a clear separation of presentation logic from business logic, making it easier to maintain and update web pages without affecting the underlying code.
- Performance: JSP can be precompiled, reducing the time required for page rendering and improving overall performance.
- Integration with Java Technologies: JSP can easily integrate with other Java technologies, such as JavaBeans and Servlets, allowing for easy data exchange between different components of a web application.
- Dynamic Web Content: JSP allows for the creation of dynamic web pages that can display real-time information, improving the user experience and providing greater interactivity.

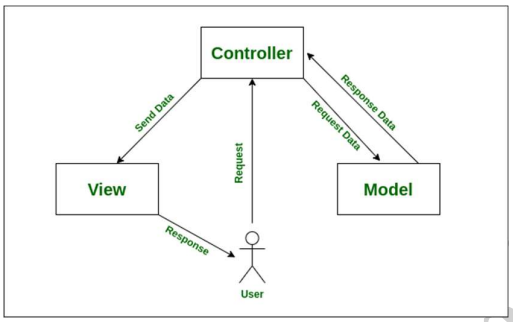
JSP	Servlet
It is a scripting language that generates dynamic content.	These are the java programs that can be compiled to generate dynamic content.
Runs slower than Servlet.	Run faster than JSP.
In model view controller JSP act as view	In model view controller servlet acts as controller.
JSP can build custom tags.	There is no facility of creating custom tags.
JSP can directly call JAVA beans.	Servlet have no facility of calling JAVA beans
It is easier to code in JSP.	The servlet are basically complex JAVA programs.

(2) Explain Life Cycle of JSP Page.

- JSP (JavaServer Pages) is a technology used to create dynamic web pages. The life cycle of a JSP page involves several stages, which are as follows:
    - **Translation:** When a JSP page is accessed for the first time, the JSP container translates the JSP page into a servlet class. This translation process includes parsing the JSP page, creating a corresponding servlet, and compiling the servlet class.
    - **Compilation:** The servlet class generated from the translation process is then compiled into a bytecode format that can be executed by the JVM.
    - **Initialization:** Once the JSP page is compiled, the JSP container initializes the servlet instance and invokes the jspInit () method. This method is called only once during the lifetime of the JSP page.
    - **Request Processing:** After initialization, the JSP container invokes the _jspService () method for each incoming request to the JSP page. This method handles the request, generates the HTML response and sends it back to the client.
    - **Destruction:** Finally, when the JSP page is no longer required or the web application is stopped, the JSP container destroys the servlet instance and invokes the jspDestroy () method.
- During the life cycle, the JSP developer can include Java code snippets, JSP tags, and custom tag libraries to generate the desired HTML response. The JSP container also provides implicit objects, such as request, response, session, and application, which can be used to access information about the client request and web application context.
- It is important to note that the jspInit () and jspDestroy() methods are optional and are only used if they are defined in the JSP page.
- Overall, the JSP life cycle is managed by the JSP container and is transparent to the JSP developer. The developer only needs to focus on writing the JSP code to generate the desired HTML response.

(3) What do you mean by MVC architecture? Explain its role in modern application and list advantages.

- MVC (Model-View-Controller) is an architectural pattern commonly used in software development, particularly in web application development.
- The purpose of MVC is to separate the application's data, user interface, and control logic into distinct components, making it easier to develop and maintain complex applications.
- In an MVC architecture, the Model represents the application's data and logic, the View represents the user interface, and the Controller handles user input and updates the Model and View as necessary.



- **Model –**
  This level is considered the lowest level when compared with the View and Controller. It primarily represents the data to the user and defines the storage of all the application's data objects.
- **Views –**
  This level is majorly associated with the User Interface (UI) and it is used to provide the visual representation of the MVC model. In simpler terms, this level deals with displaying the actual output to the user. It also handles the communication between the user (inputs, requests, etc.) and the controller.
- **Controller –**
  This level takes care of the request handler. It is often considered as the brain of the MVC system- a link, so to speak, between the user and the system. The controller completes the cycle of taking the user output, converting it into desired messages, and passing them on to the views (UI).

➢ **Advantage of MVC architecture:**

- **Modularity:** By separating the application's components, MVC makes it easier to maintain and update each component individually, without affecting the others.
- **Reusability:** Each component can be reused across multiple applications, reducing development time and improving code quality.
- **Testability:** With well-defined components, it is easier to test the application's functionality, making it more reliable and reducing the risk of errors.
- **Scalability:** With a clear separation of concerns, MVC makes it easier to add new functionality to the application as needed, without affecting existing code.
- **Flexibility:** The MVC architecture can be adapted to a wide range of applications, from simple web pages to complex enterprise-level systems.

(4) Explain use of JSP page directive tag with its attribute. ***OR*** Explain Page, Include, Taglib Directive.

- JSP (JavaServer Pages) provides a page directive to define attributes of a JSP page. This directive is used to provide instructions or metadata to the JSP engine. The syntax for using the page directive is as follows:

```
<%@ page attribute="value" %>
```

- Here, attribute represents a specific attribute that can be set using the page directive, and value is the value that is assigned to that attribute. The page directive must be placed at the top of a JSP page, before any other JSP tags.
- Using the page directive, you can customize the behavior of a JSP page and make it work more effectively with the server and the user's browser.
- **Here are some common attributes of the page directive and their meanings:**

  - **language -** Specifies the scripting language used in the JSP page. The default is Java, but you can also use Groovy or other languages if they are supported by the JSP container.
    - <%@ page attribute="value" %

  - **contentType -** Sets the MIME type of the response that will be sent to the client. For example, text/html indicates an HTML response, while application/json indicates a JSON response.
    - <%@ page contentType="application/msword" %>

  - **pageEncoding -** Sets the character encoding used by the JSP page. Used to ensure that special characters are displayed correctly.
    - <%@ page attribute="value" %>

  - **buffer -** Sets the size of the buffer used to store the output of the JSP page before it is sent to the client.
    - <%@ page buffer="16KB" %>

- **isThreadSafe -** Determines whether the JSP page can be used by multiple threads at the same time. This is important in high-traffic web applications.
  - <%@ page isThreadSafe="false" %>

- **session -** Determines whether the JSP page can access the user's session data.
  - <%@ page attribute="value" %>

- **errorPage -** Specifies the URL of the JSP page that will be displayed if an error occurs during the processing of the current JSP page.
  - <%@ page errorPage="myerror.jsp" %>

- **isErrorPage -** Determines whether the current JSP page is an error page or not.
  - <%@ page isErrorPage="true" %>

- **import -** Imports one or more Java classes or packages for use in the JSP page.
  - <%@ page import="java.util.Date" %>

➢ **Include Directive:**

- It is used to include the contents of another file into the current JSP. This is useful for reusing common page element including boilerplate code.

```
<%@ include file="filename" %>
```

➢ **Taglib Directive:**

- It is used to store custom tag libraries that can be used in JSP. A custom tag is a user-defined JSP tag that can be reused across multiple JSP page.

```
<%@ taglib prefix="prefix" uri="uri" %>
```

(5) Explain use of JSP action tag.

- In JSP, an action tag is used to perform specific tasks like including other resources, passing parameters values, setting attributes, etc.
- There are serval types of action tags, each with its own purpose.
- **Include Action Tag:**
    - The include action tag is used to include other resources like JSP page, HTML page, or other resources in the current JSP page.

```
<jsp:include page="pageName.jsp" />
```

- **Forward Action Tag:**
    - It is used to forward the current request to other resources like a JSP page, HTML page, or a servlet.

```
<jsp:forward page="pageName.jsp" />
```

- **Param Action Tag:**
    - It is used to pass parameter values to other resources like a JSP page, HTML page or a servlet.

```
<jsp:param name="paramName" value="paramValue" />
```

- **UseBean Action Tag:**
    - It is used to create a JavaBean instance and set its properties.

```
<jsp:useBean id = "name" class = "package.class" />
```

- **Property Action Tag:**
    - It is used to set the properties of JavaBean instance.

```
<jsp:setProperty name="beanName" property="propertyName" value="propertyValue" />
```

(6) Explain JSP implicit (in-built) object with its use.

- JSP implicit objects are pre-defined objects in JSP that are available use in JSP pages without being explicitly declared.
- These objects can be used to access various types of information related to JSP request, such as request parameters, cookies and server information.
- **request:** It represent the HTTP request made by the client to the server. It can be used to access parameters passed through the request such as from data or query parameter.

```
<%= request.getParameter("parameterName") %>
```

- **response:** It represent the HTTP response that the server sends back to the client. It can be used to set headers, cookies and response status code.

```
response.setContentType("text/html");
```

- **out:** It represent the output stream used to send the response back to the client. It is commonly used to print HTML or other content to the web page.

```
<%
   out.println("Hello, world!");
%>
```

- **session:** It represent the user session on the server. It can be used to store and retrieve session-specific data for a user.

```
session.setAttribute("attributeName", attributeValue);
```

- **application:** It represent the JSP application context. It can be used to store and retrieve data that is accessible across multiple JSP pages within an application.

```
application.setAttribute("attributeName", attributeValue);
```

- **config:** It represent JSP configuration object. It can be used to retrieve initialization parameter specified in the deployment descriptor.

```
String paramValue = config.getInitParameter("parameterName");
```

(7) Explain Session Object with Example.

- A session can be defined as a specific period of a time the user spends browsing the site.
- In JSP session object is used to maintain the user's session on the server-side. It allows the developer to store and retrieve user-specified data across multiple requests, making it very useful for developing interactive web applications.
- The session object is very useful for storing user-specific data such as shopping cart contents, user preference, and login information.
- It can be accessed from any JSP page within the same web application, making it a powerful tool for building dynamic and interactive web application.

```
<%@ page language="java" %>
<html>
<head>
    <title>Session Example</title>
</head>
<body>
    <%
        // Get the session object
        HttpSession session = request.getSession();

        // Set a session attribute
        session.setAttribute("username", "JohnDoe");

        // Get a session attribute
        String username = (String) session.getAttribute("username");

        // Print the username
        out.println("Welcome, " + username);
    %>
</body>
</html>
```

- In above code, we first obtain the **HttPSession** object using the **request.getSession ()** method.
- We then use this object to set a session attribute **username** with the value **JohnDoe**.
- We then retrieve this attribute using the **getAttribute ()** method and assign it to the variable name **username**.
- Finally, we print the value of **username** to the web page using the **out.println ()** method.

(8) Explain and List JSTL core tags.

- JSTL stands for Java Server Pages Standard Tag Library.
- It is a collection of custom tags that can be used to perform common tasks in JSP pages.
- The JSTL core tags provide functionality for performing basic operation such as iteration, conditionals, and formatting.
- **<c:out>:** It is used to output the value of an expression. It is equivalent to **<%= expression %>**

```
<c:out value="${user.name}" />
```

- **<c: set>:** It is used to set the value of a variable.

```
<c:set var="username" value="${user.name}" />
```

- **<c: if>:** It is conditional tag used for testing condition and display the body if the expression evaluates true.

```
<c:if test="${user.role == 'admin'}">
  Welcome, administrator.
</c:if>
```

- **<c: choose> and <c: when>:** These tags are used to provide multiple conditional branches.

```
<c:choose>
  <c:when test="${user.role == 'admin'}">
    Welcome, administrator.
  </c:when>
  <c:when test="${user.role == 'editor'}">
    Welcome, editor.
  </c:when>
  <c:otherwise>
    Welcome, guest.
  </c:otherwise>
</c:choose>
```

- **<c: forEach>:** It is basically integration tag. It repeats the nested body content for fixed number of times over collections.

```
<c:forEach var="book" items="${bookList}">
  ${book.title} by ${book.author}
</c:forEach>
```

(9) Explain Various tags in JSTL.

- JSTL (JavaServer Pages Standard Tag Library) is a collection of custom tags that encapsulate core functionality common to many JSP applications.
- JSTL can be used to handle control flow, loops, conditional statements, and database operations in JSPs.
- **Some of the most commonly used JSTL tags include:**

- **Core Tags:** The Core tags are the most commonly used JSTL tags, and they are included in the JSTL core library. These tags provide the basic functionality needed in most web applications. Some of the most commonly used Core tags are:
  - **<c: if>:** Evaluates a condition and includes its body if the condition is true.
  - **<c: choose>:** Tests a set of mutually exclusive conditions, and executes the body of the first condition that is true.
  - **<c: set>:** Sets a value for a scoped variable, which can be used throughout the page.
  - **<c: forEach>:** Iterates over a collection of items and executes the body of the tag for each item.

- **Formatting Tags**: The Formatting tags provide a way to format and parse values in JSP pages. These tags are included in the JSTL fmt library. Some commonly used Formatting tags are:
  - **<fmt: formatDate>:** Formats a date object into a string using the specified pattern.
  - **<fmt: formatNumber>:** Formats a number object into a string using the specified pattern.
  - **<fmt: parseDate>:** Parses a string into a date object using the specified pattern.
  - **<fmt: parseNumber>:** Parses a string into a number object using the specified pattern.

- **SQL Tags:** The SQL tags are used to access databases in JSP pages. These tags are included in the JSTL SQL library. Some commonly used SQL tags are:
  - **<sql: setDataSource>:** Sets the data source for the database connection.
  - **<sql: query>:** Executes a SQL query and retrieves the result set.
  - **<sql: update>:** Executes a SQL update statement and returns the number of affected rows.

- **XML Tags:** The XML tags are used to manipulate XML data in JSP pages. These tags are included in the JSTL XML library. Some commonly used XML tags are:
  - **<x: parse>:** Parses an XML document and creates a DOM tree.
  - **<x: forEach>:** Iterates over a collection of XML nodes and executes the body of the tag for each node.
  - **<x: out>:** Writes the value of an XML node to the output.

(10) Difference between include Directive and jsp: include action tag.

- They include directive and jsp: include action tag are two different ways to include content from one JSP page in another.
- The main difference between the two is the time at which the included content is processed.
- They include directive includes the content of the specified JSP page during the translation phase, before the JSP page is compiled. The content of the included page becomes part of the main JSP page, and any changes made to the included page will not be reflected until the main JSP page is recompiled.

```
<%@ include file="header.jsp" %>
<body>
  <h1>Welcome to my website</h1>
  <p>This is the homepage of my website.</p>
</body>
```

- The jsp: include action tag, on the other hand, includes the content of the specified JSP page during the execution phase, at the point where the tag is encountered. This means that any changes made to the included page will be reflected immediately when the main JSP page is requested.

```
<jsp:include page="header.jsp" />
<body>
  <h1>Welcome to my website</h1>
  <p>This is the homepage of my website.</p>
</body>
```

Include Directive	Include action
Includes resources at translation time.	Includes resources at request time.
Better for static pages.	Better for dynamic pages.
Include the original content in the generated servlet.	Calls the include method.

(11) Which action tags are used to access the JavaBeans from JSP page?

- **jsp: useBean:** This action tag is used to create an instance of a JavaBean or to get a reference to an existing bean. It has attributes like id, scope, class, and type.
- **jsp: setProperty:** This action tag is used to set the properties of a bean. It has attributes like name, property, value, and param.
- **jsp: get Property:** This action tag is used to get the properties of a bean. It has attributes like name and property.
- **Jsp: forward:** It is used to forwards the request and response to another resources.
- **Jsp: include:** It is used to include another resource.
- **Jsp: plugin:** It is used to embeds another component such as applet.
- **Jsp: param:** It is used to set the parameter value. It is used in forward and include mostly.
- **jsp: fallback:** It can be used to print the message if plugin is working. It is used in jsp: plugin.

(12) What is Expression Language EL in JSP? Explain with example.

- EL is a feature in JSP that allows dynamic content to be placed within JSP page. It simplifies the retrieval and manipulation of data from Java Objects, arrays and collection in JSP. EL can be used to replace java code and script let code.
- EL expression is enclosed within **${}, # {}. ${}** is used to access the value of a variable. While **# {}** is used for more complex expression such as invoking method or performing arithmetic operations.

```
<html>
   <body>
      <h1>Welcome ${user.name}!</h1>
      <p>Your email is ${user.email}.</p>
      <p>Your account balance is ${user.balance}.</p>
   </body>
</html>
```

(13) Develop a web application as following to demonstrate the use of JSTL SQL tag library.

- **Create a web page to store the registration detail (Name, email and contact number) of user in the database.**
- **Create a web page for user to update registration detail in the database.**
- **Create a web page to display the list of all registered users.**

(14) Explain JSP Exception Handling.

- JSP (JavaServer Pages) exception handling is the process of catching and handling errors that occur during the execution of a JSP page.
- It is important to handle exceptions in JSP pages to ensure that the web application is robust and can handle errors gracefully.
- In JSP, exception handling can be done using the try-catch block, where the code that might throw an exception is placed inside the try block, and the catch block is used to handle the exception.
- The catch block can catch a specific type of exception or a general exception, depending on the requirements of the application.

```jsp
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ page import="java.sql.*" %>
<html>
<head>
    <title>Exception Handling in JSP</title>
</head>
<body>
<%
    Connection conn = null;
    try {
        // Load the driver
        Class.forName("com.mysql.jdbc.Driver");
        // Get a connection to the database
        conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydatabase", "username", "password");
        // Execute the SQL statement
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM users");
        // Display the results
        while (rs.next()) {
            out.println("ID: " + rs.getInt("id") + "<br>");
            out.println("Name: " + rs.getString("name") + "<br>");
        }
        rs.close();
        stmt.close();
        conn.close();
    } catch (SQLException e) {
        // Handle SQL exception
        out.println("SQLException: " + e.getMessage());
    } catch (Exception e) {
        // Handle all other exceptions
        out.println("Exception: " + e.getMessage());
    } finally {
        try {
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            // Handle SQL exception
            out.println("SQLException: " + e.getMessage());
        } }
%>
</body>
</html>
```

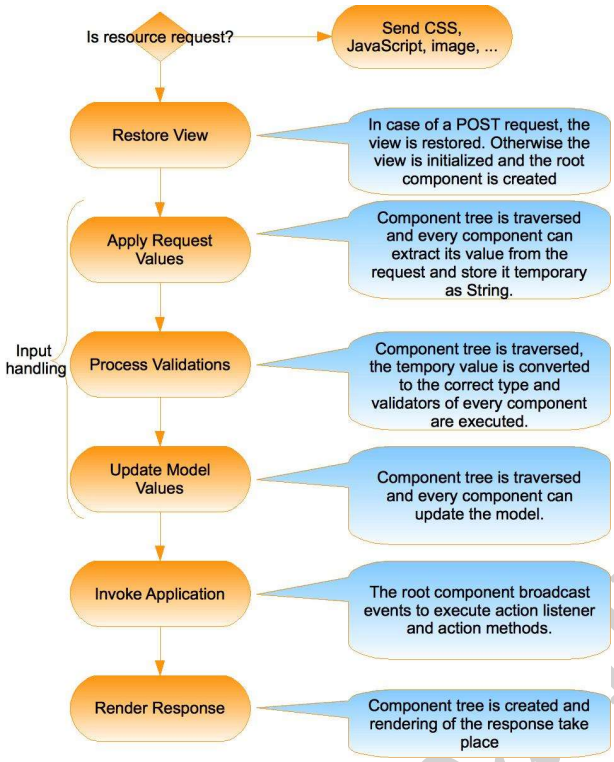
- In the above example, the code that might throw an exception is placed inside the try block.
- If an SQL exception occurs, it is caught in the first catch block, and the message is displayed to the user.
- If any other exception occurs, it is caught in the second catch block, and the message is displayed to the user.
- The finally block is used to ensure that the database connection is closed, regardless of whether an exception occurs or not.
- The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer.
- Overall, JSP exception handling is a crucial aspect of developing robust and reliable web applications. By properly handling exceptions, you can ensure that your application is able to handle errors gracefully and provide a better user experience.

Chapter: 5

(1) What is JSF? Explain its advantage and disadvantage.

- JSF stands for Java Server Faces.
- JSF is a web application framework used to develop the web applications with the rich user interface.
- This framework is based on MVC architecture. There are other technologies that make use of MVC pattern are JSP, Spring, ASP.NET and so on.
- It simplifies construction of User Interface component. These UI components can establish communication with database or event handler.
- **Advantage:**
  - **Component Based framework:** Which make it easy to create and manage reusable UI components that can be used across multiple pages of an application.
  - **Platform independent:** JSF is built on top of it, making it easy to develop application that can run on any operating system.
  - **Rich set of UI:** JSF provides rich set of built-in UI components that can be used to create complex user interfaces without writing much code.
  - **Easy to integrate with other JAVA technologies:** JSF can be easily integrate with other java technologies, such as JSP, Java servlet and Enterprise JavaBeans.
  - **Built-in Validation Support:** JSF provides built-in support for data validation, making it easy to ensure that user input is valid and meets the requirement of application.
- **Disadvantage:**
  - **Difficult to learn:** JSF can be difficult to learn for beginners.
  - **Performance issue:** Sometimes it suffers from performance issues, especially when used with large, complex application that requires a lot of processing power.
  - **Complexity:** The framework is sometimes perceived as overly complex, and it can be difficult to troubleshoot issue when they arise.

(2) Explain JSF Life Cycle. OR Explain different phases of JSF request processing life cycle.



- **Restore View:**
  - This is the first phase in JSF request processing life cycle. This phase is used for constructing view to display in the front end.
  - This view is stored in FaceContext instance and using the information in this instance single request is processed.
- **Apply Request View:**
  - After the component tree is restored, each component in the tree extracts its new value from the request parameters by using its decode method.
  - The value is then stored locally on the component.
  - If the conversion of the value fails, an error message associated with the component is generated and queued on FaceContext.
  - This message will be displayed during the render response phase, along with ant validation errors resulting from the process validation phase.

- **Process Validation:**
  - During this phase, the java server faces implementation processes all validators registered on the component in the tree.
  - It examines the component attribute that specify the rules for the validation and compares these rules to the local value stored for the component.
  - If the local value is invalid, the JavaServer Faces implementation adds an error message to the FaceContext instance, and the life cycle advances directly to the render response phase so that the page is rendered again with the error message displayed.
- **Update Model Values:**
  - If the Java Server faces implementation find that the data is valid then, it moves over the component tree and set the server-side object properties to component's local value.
  - If the local data cannot be converted to the types specified by the bean properties, the life cycle advances directly to the render response phase so that the page is rendered with error display.
- **Invoke Application:**
  - In this phase JSF handles the application-level events.
  - If the application needs to redirect to a different web application resource or generate a response that does not contain any Java Server Face components, it can call FaceContext.reponseComplete.
- **Render Response:**
  - During this phase, the Java Server Faces implementation directs for rendering the page to JSP container if the application is using JSP pages.
  - For initial request, the component represented on the page will be added to the component tree as JSP container executes the page.
  - If this is not an initial request, the component tree is already built so component need not be added again.
  - In either case, the components will render themselves as the JSP container server traverses the tag in the page.

(3) Explain JSF Facelet and their use.

- JSF facelet is a templating language that provides an alternative to the JSP technology used in JSF.
- It allows developer to create reusable templates for building their user interface of a web application.
- Facelet is a powerful but lightweight page declaration language that is used to build JavaServer Faces view using HTML style templates and to build components tree.
- **Tags:**
  - **ui: insert:** Used in template file, it defines content that is going to replace by the file that load the template.
  - **ui: define:** Defines content that is inserted into template with a matching ui: insert tag.
  - **ui: include:** Similar to "jsp: include", includes content from another XHTML page.
  - **ui: composition:** If used with "template" attribute, the specified template is loaded, and the children of this tag defines the template layout; Otherwise, it's a group of elements, that can be inserted somewhere. In addition, JSF removes all tags "outside" of "ui: composition" tag.
- **Advantages:**
  - **Improved performance:** Facelets is designed to be more efficient and faster than JSP.
  - **Easy to learn:** Facelets has a simpler syntax than JSP, which makes it easier for developers to learn and use.
  - **More powerful:** Facelets provides additional features, such as template composition and support for AJAX, which are not available in JSP.
  - **Better separation of concerns:** Facelets makes it easy to separate the presentation logic from the business logic, which improves maintainability and scalability of the application.

(4) Discuss JSF converter tag.

- During the web application, the data entered by the user need to be converted into their appropriate types. There are various inbuilt converter tags available.
- JSF converter tag are used to convert the input data entered by the user in a format that can be stored or manipulated by the application.
- **Number Converter Tag:**
  - The JSF number converter tag is used to convert a number entered by the user into a format that can be used by the application.

minIntegerDigits	For defining minimum number of digits in integer part
maxIntegerDigits	For defining maximum number of digits in integer part
minFractionDigits	For defining minimum number of digits in fraction part
maxFractionDigits	For defining maximum number of digits in fraction part
Type	For representing percent, currency or number.
CurrencySymbol	For converting the currency values to currency symbol.
groupingUsed	If grouping separators need to be used then it is used.

```
<h:outputText value = "100.12345" >
   <f:convertNumber minFractionDigits = "2" />
</h:outputText>
```

- **Converter Date Time Tag:**
  - This tag is used to convert the String to a Date or time object and o display it in different formats.
  - This tag has a dedicated set of attribute values for formatting both Date and Time values.

Pattern	Formatting pattern of date or time
Type	It can be date, time or both.
dateStyle	Default, short, medium, long or full.
timeStyle	Default, short, medium, long or full.
timeZone	It is used for specific formatting or parsing.

```
<h:form>
   <h:inputText id = "dateInput" value = "#{userData.date}"
      label = "Date" >
      <f:convertDateTime pattern = "dd-mm-yyyy" />
   </h:inputText>
   <h:commandButton value = "submit" action = "result"/>
</h:form>
```

(5) List the validation tags with their example.

- It is a common practice to validate the data entered by the user before being processed by the server web application.

f: validateLongRange	Validates the input value is within specified range of long value.
f: validateDoubleRange	Validates the input value is within specified range of double value.
f: validateRegex	Validates the input value matches the specified regular expression
f: validateLength	Validates the length of input value is within the specified range.
f: validateRequired	Validates the input value is not null and not an empty string.

```
<h:form>
   <h:panelGrid columns="2">
      <h:outputLabel for="name" value="Name: " />
      <h:inputText id="name" value="#{user.name}" required="true">
         <f:validateLength minimum="3" maximum="20" />
      </h:inputText>

      <h:outputLabel for="age" value="Age: " />
      <h:inputText id="age" value="#{user.age}" required="true">
         <f:validateLongRange minimum="18" maximum="60" />
      </h:inputText>

      <h:outputLabel for="email" value="Email: " />
      <h:inputText id="email" value="#{user.email}" required="true">
         <f:validateRegex pattern="^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$" />
      </h:inputText>

      <h:outputLabel for="phone" value="Phone: " />
      <h:inputText id="phone" value="#{user.phone}" required="true">
         <f:validateRegex pattern="^\d{3}-\d{3}-\d{4}$" />
      </h:inputText>

      <h:commandButton value="Submit" action="#{user.save}" />
   </h:panelGrid>
</h:form>
```

(6) Write a code snippet to show the use of JSF action event.

➢ Suppose you have a web page with a form that allows users to input two numbers and click a button to calculate their sum. When the button is clicked, you want the sum to be displayed on the page.

```
@ManagedBean
@ViewScoped
public class CalculatorBean {
    private int num1;
    private int num2;
    private int sum;

    public int getNum1() {
        return num1;
    }

    public void setNum1(int num1) {
        this.num1 = num1;
    }

    public int getNum2() {
        return num2;
    }

    public void setNum2(int num2) {
        this.num2 = num2;
    }

    public int getSum() {
        return sum;
    }

    public void calculateSum(ActionEvent event) {
        sum = num1 + num2;
    }
}
```

- The **getNum1(), setNum1(), getNum2(),** and **setNum2()** methods are getter and setter methods for the num1 and num2 fields.
- The **getSum ()** method is a getter method for the sum field.
- The **calculateSum ()** method is an action method that is invoked when the "Calculate Sum" button is clicked. The method receives an ActionEvent object as a parameter, which is used to indicate that the method is an action method. The method simply adds the two input numbers and sets the result to the sum field.

- To use the CalculatorBean class in a JSF page, you can define a form with two input fields for the numbers, a button to calculate the sum, and an output label to display the result. Here is an example code for the JSF page:

```
<h:form>
    <h:outputLabel for="num1" value="Number 1:"/>
    <h:inputText id="num1" value="#{calculatorBean.num1}" required="true"
        validatorMessage="Please enter a valid number."/>
    <h:message for="num1" style="color:red;"/>

    <br/>

    <h:outputLabel for="num2" value="Number 2:"/>
    <h:inputText id="num2" value="#{calculatorBean.num2}" required="true"
        validatorMessage="Please enter a valid number."/>
    <h:message for="num2" style="color:red;"/>

    <br/>

    <h:commandButton value="Calculate Sum" actionListener="#{calculatorBean.calculateSum}"/>

    <br/>

    <h:outputLabel for="sum" value="Sum:"/>
    <h:outputText id="sum" value="#{calculatorBean.sum}"/>
</h:form>
```

- The value attributes of **<h: inputText>** tags are set to **#{calculatorBean.num1}** and #{calculatorBean.num2} which bind these input fields to the **num1** and **num2** properties of calculatorBean, respectively.
- The **<h: message>** tags display the error messages for invalid input fields, and the **<h: commandButton>** tag defines a button that triggers the calculate () method when clicked.
- The **<h: outputLabel>** tag displays the result of the calculation in the result property of calculatorBean.
- The **calculate ()** method retrieves the values of **num1** and **num2** properties of calculatorBean, performs the addition, and sets the result to the result property of **calculatorBean.**
- The action attribute of the **<h: commandButton>** tag is set to **#{calculatorBean.calculate}** which calls the calculate() method when the button is clicked

(7) Write a short note on JSF Libraries: Prime Faces.

- PrimeFaces is a popular open-source user interface (UI) component library for JavaServer Faces (JSF) web applications.
- It provides a collection of over 100 ready-to-use UI components, such as charts, calendars, menus, and dialog boxes, that are customizable and easy to use.
- PrimeFaces includes a variety of responsive and mobile-friendly UI components with support for popular web technologies such as HTML5, CSS3, and JavaScript.
- It also supports popular component frameworks, such as Twitter Bootstrap and jQuery UI.
- PrimeFaces can be easily integrated with any JSF implementation, such as Mojarra and MyFaces.
- It supports both server-side and client-side validation, and can be used to develop both desktop and mobile web applications.

- **Some of the key features of PrimeFaces include:**

  - **Extensive set of UI components:** PrimeFaces provides over 100 UI components, including charts, tables, forms, and dialogs, that can be easily customized and used in your application.
  - **Responsive design:** PrimeFaces includes responsive design that works well on different screen sizes, making it easy to create mobile-friendly web applications.
  - **AJAX support:** PrimeFaces provides AJAX support, which means that components can be updated without requiring a full page reload, leading to a better user experience.
  - **Theme support:** PrimeFaces includes a theming mechanism that allows you to easily change the look and feel of your application.
  - **Client-side validation:** PrimeFaces supports client-side validation, which improves the user experience by catching errors before the form is submitted to the server.

(8) Write JSF expression language with suitable program.

- JSF expression language allows the user to access the data dynamically from the Java Bean component.
- JSF Expression Language (EL) is a powerful feature that simplifies the process of writing dynamic content in a JSF web application.
- It allows developers to bind component properties to backing bean properties and execute method expressions.
- **Property Expression:** It is used to access the value of a property in a managed bean.
  Syntax: `#{beanName.propertyName}`
  Access this property: `#{myBean.message}`

- **Method Expression:** It is used to invoke method in a managed bean.
  Syntax: `#{beanName.methodName(param1, param2, ...)}`
  Access this method: `#{myBean.calculateSum(10, 20)}`

- **Example:**

```
<h:outputLabel value="Enter your name:" />
<h:inputText value="#{userBean.name}" />
<h:commandButton value="Submit" action="#{userBean.greetUser}" />

<h:outputText value="Welcome #{userBean.name}!" />
```

- The second line defines an input text field, and its value attribute is bound to the property "name" of a backing bean named "userBean" using EL. This means that when the user enters a name in the input field, the value is automatically stored in the "name" property of the "userBean" object.
- The third line defines a command button with a static label "Submit". Its action attribute is set to a method expression "#{userBean.greetUser}". This means that when the user clicks the button, the "greetUser" method of the "userBean" object is invoked.
- The last line defines an output text component that displays a personalized message using the "name" property of the "userBean" object. Message is generated using EL, which retrieves the value of the "name" property and concatenates it with a static string "Welcome "

Chapter: 6

(1) What is Hibernate? Explain advantage of using Hibernate over SQL and JDBC.

- Hibernate is a popular open-source Object-Relational Mapping (ORM) tool for Java Platform.
- Hibernate allow Java developer to create persistent class or persistent data without bothering about how to handle data.
- Hibernate query language is an extension to SQL.
- The main task of hibernate is mapping of java classes to database tables. That means Java Data types are mapped to the SQL data types.
- Hibernate provide data query and retrieval facilities. This feature avoids wastage of development time in manual data handling in SQL and JDBC.
- The goal of hibernate is to relief the developer from 95% of common data persistence related programming tasks.
- The high performances, robust database application with java can be easily built using hibernate.
- There are two alternatives for using hibernate: Use of SQL for creating and retrieving the data in database but this alternative is not as much attractive as hibernate. The second alternative is use of Enterprise Java Beans EJB. The EJB is a really good alternative for hibernate but it required EJB container.

- **Advantage of Hibernate Over JDBC:**

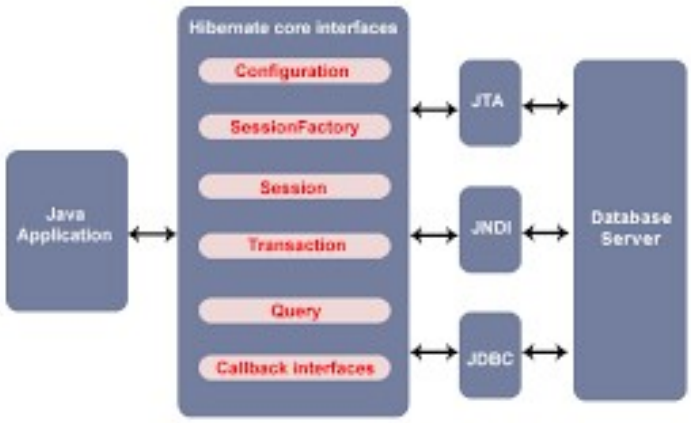
- Hibernate code works for almost all the database such as MySQL, Oracle and so on.
- While using hibernate, there is no need to have knowledge of SQL because in hibernate the table is treated as object. But while working with JDBC we need to know SQL.
- Hibernate has a support of cache, hence performance is increased by placing data in cache. In JDBC java cache has to be implemented.
- Hibernate is a database independent because you need not change the HQL queries when you change the database like MySQL, oracle , because of this its easy to migrate to the new database.

➢ **Advantage of Hibernate over SQL:66666**

- The main advantage of hibernate is that hibernate avoids writing huge queries. Because most of the work is taken care by mapping and also criteria class is very useful for complex joins. We can also avoid JDBC API by using hibernate.
- In case you using hibernate the you need not required the SQL because the HQL us much like SQL.
- Hibernate query language uses classes and property instead of tables and column In SQL we need join statement to merge the table where as HQL need not require because in hibernate we utilizing hbm.xml file.

(2) Draw and Explain Hibernate architecture.

- Hibernate is widely used object relational mapping (ORM) framework that enables developers to work with databases using object-oriented concepts.
- Hibernate has a layered architecture which helps the user to operate without having to know the underlying APIs.
- Hibernate makes use of database and configuration data to provide persistence service to the application.



Hibernate Architecture

- **Configuration Object:** It is the first Hibernate object you create in any Hibernate application.
  - It is usually created only once during application initialization.
  - It represents a configuration or properties file required by the hibernate.
  - Database connection is handled through one or more configuration file supported by hibernate.
  - Class mapping setup creates the connection between the Java classes and database table.
  - It checks whether the config file is syntactically correct or not.
  - If the config file is not valid then it will throw an exception. If it is valid then it creates a meta-data in memory and returns the meta-data to object to represent the config file.

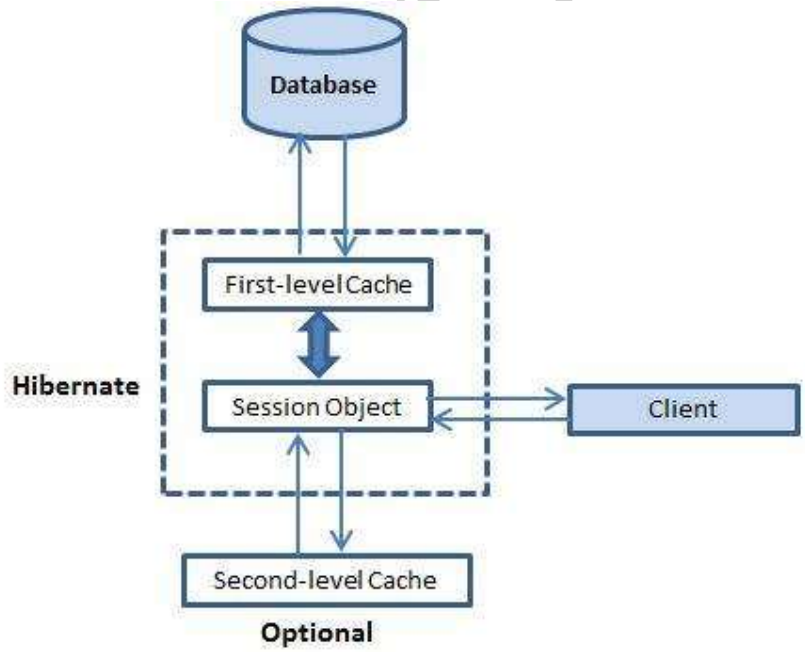
- **Session Factory Object:** Session factory is an interface which is present in org.hibernatet package and it is used to create Session Object.
  - It is immutable thread-safe in nature.
  - These components create and manage hibernate session object, which represent a single database transaction.
  - The session factory is also responsible for creating and managing a pool of database connections.

- **Session:** Session is an interface which is present in org.hibernate.package. Session object is created based upon session factory object.
  - It opens the connection/session with Database software through Hibernate Framework.
  - It is a light-weight object and it is not thread-safe.
  - Session object is used to perform CURD operation.

- **Transaction:** Transaction object is used whenever we perform any operation and based upon that operation there is some change in database.
  - Transaction object is used to give the instruction to the database to make the changes that happen because of operation as a permanent by using commit () method.

- **Query:** Query is an interface that present inside org.hibernate package.
  - A query instance is obtained by calling Session.creatQuery ().
  - This component is used to create and execute database queries.

- **Hibernate Caching Architecture:**
  - The performance of Hibernate web application is improved using caching.
  - The cache is actually store the data already loaded from database, so that the traffic between the application program and database will be reduced.
  - The architecture is divided into two part:
  - First Level Cache: This is a session cache and is a mandatory cache. Through this cache all request must pass.
  - Second Level Cache: This is an optional cache and first level cache will always be searched before locating an object into second level cache.



(3) What is ORM? Explain object relational mapping in hibernate.

- ORM stands for Object-Relational Mapping, Which is a programming technique used to map object to database table and vice versa.
- In other words, it is a process of converting data between incompatible type systems, such as a relational database and an object oriented programming language.
- Hibernate is an ORM framework that simplifies the process of mapping objects to relational database tables.
- ORM are usually defines in an XML document. The mapping language is Java-centric, meaning that mappings are constructed around persistent class declaration and not table declaration.
- **Explanation of Mapping Document:**
- **<hibernate-mapping>:** It is root element of hibernate mapping document. Between this tag elements are present.
- **<class>:** This element maps the class object with corresponding entity in the database. It also tells what table in the database has to access and what column in that table it should use.
- **<id>:** This element in unique identifier to identify an object. In fact this element map with the primary key of the table.
  Name: Property name of the persistence model class.
  Type: The java data type is used for that property.
  Column: Name of the column for that property of the persistence object.
- **<generator>:** Used to create primary key for new record, there are some commonly used generators:
  Increment: used to generate primary key of type long, short and int that are unique only.
  Assigned: is used when application code generates the primary key.
  Identify: supports identity columns in DB2, MySQL, MS SQL server.
  Uuid: Unique use ID of 128 bits generated from using algorithm and return type is string.
- **<property>:** Defines standard java attributes and their mapping into database schema using name attribute.

(4) Explain ORM? Explain the components of hibernate.cfg.xml file.

- ORM stands for Object-Relational Mapping, which is a programming technique used to map object to database table and vice versa.
- In other words, it is a process of converting data between incompatible type systems, such as a relational database and an object-oriented programming language.
- Hibernate is an ORM framework that simplifies the process of mapping objects to relational database tables.
- ORM are usually defines in an XML document.
- Hibernate is a popular ORM tool for Java developers.
- One of the key components of Hibernate is the hibernate.cfg.xml file, which contains the configuration settings for the ORM tool.
- The hibernate.cfg.xml file is typically located in the src/main/resources directory of a Hibernate project.
- **The components of the hibernate.cfg.xml file are:**
    - **Database connection properties:** This section of the file specifies the properties needed to connect to the database, including the database URL, username, and password.
    - **Dialect:** This section of the file specifies the SQL dialect to be used by Hibernate. The dialect is used to translate Hibernates object-oriented queries to SQL.
    - **Mapping files:** This section of the file specifies the mapping files that Hibernate should use to map Java classes to database tables.
    - **Session factory:** This section of the file specifies the session factory, which is responsible for creating sessions and managing their configuration settings.
    - **Properties:** This section of the file contains additional configuration properties for Hibernate, such as connection pool settings, caching settings, and transaction settings.

(5) What is HQL? Difference between HQL and SQL.

- HQL stands for Hibernate Query Language.
- It is an object-oriented query language used to perform database operation in Hibernate.
- It is very similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties.

SQL	HQL
Stands for Structured Query Language.	Stands for Hibernate Query Language.
It is used to perform database operation in traditional database.	It is used to perform database operation in Hibernate ORM.
It uses tables, columns and SQL function to operate on data.	It uses persistent objects and their properties to operate on the data.
These are executed directly on the database.	HQL queries are executed on the Hibernate session and then translate to SQL for execution.
SQL is a standard query language used by many relational databases.	HQL is a proprietary query language used only in Hibernate.
It does not provide features like object-oriented querying, polymorphic querying.	It provides features like object-oriented querying, polymorphic querying.
It is database-oriented table query.	It is a object-oriented query.
Native SQL is usually faster.	Non-native HQL is usually slower.
Offers complex interface to new user.	Provides user-friendly interface.
SQL code is longer than HQL code.	HQL code is smaller than SQL code.

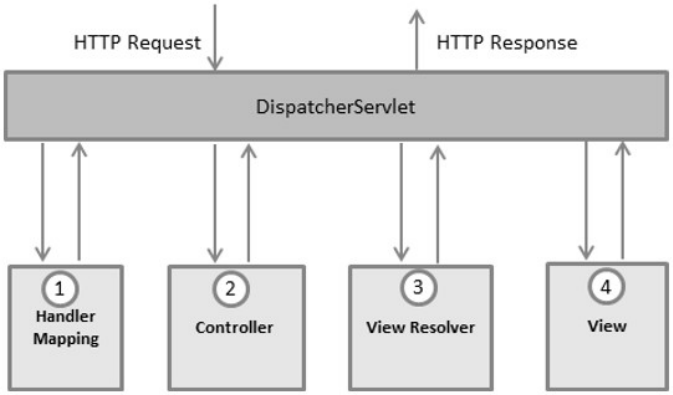
```
SQL:
select * from <table-name>;
select * from emp where empid = 1;
```

```
HQL:
from <pojo-class>;
from employee where empId = 1;
```

Chapter: 7

(1) Explain Spring MVC and Spring Framework architecture.

- The Spring web model-view-controller (MVC) framework is designed around a Dispatcher Servlet that handles HTTP requests and responses, with configurable handler mapping, view resolution, locale, time zone and theme resolution as well as support for uploading files.
- The default handler is based on the Controller and Request Mapping annotations, offering a wide range of flexible handling methods.
- **Model:** It is normally an application data containing POJO classes, i.e. Plain old Java Object Classes.
- **View:** It is responsible for showing the result. This element generates the HTML output which can be displayed on client's machine.
- **Controller:** This element is responsible for processing user request.



- The Dispatcher Servlet receives the HTTP request.
- The Dispatcher Servlet consults Handler Mapping to call appropriate controller.
- The controller takes the request and finds the appropriate service, sends GET/POST method for calling that service. This service on execution defines the business logic and returns the view name to Dispatcher Servlet.

- Using View Resolver, the Dispatcher Servlet will choose the appropriate view.
- The view will then be returned to Dispatch Servlet to be displayed on the client's browser window.

- **Benefits of MVC Framework:**
  - It provides clear separation of roles such as- controller, validator, model object, handler mapping, view resolver and so on.
  - It provides powerful and straightforward configuration of both framework and application classes as Java Beans.
  - It allows reusability of business code.
  - It also supports for customization and application-level validation.
  - It supports for simple yet powerful tag library.
  - It supports easy integration with any type of view.

- **Spring:** The spring framework is an application framework for the java platform.
  - This is an open-source framework which was initially designed by Rod Johnson. It was first released under Apache 2.0 in 2023.
  - The web application is designed using the java program and are run on the top of Java EE platform.

- **Benefits of Spring Framework:**
  - It is lightweight container framework. This framework enhances modularity and provides more readable code.
  - It provides loose coupling between different modules.
  - It supports declarative transaction, caching, validation, and formatting logging service.
  - It can be easily configured by XML schema or annotators-based style.
  - With dependency injection approach, dependencies are explicit, and evident in constructor or JavaBean properties.
  - It does not require special deployment steps.

(2) What is Spring IoC Container.

- In the Spring Framework, the Inversion of Control (IoC) container is responsible for managing the application's object lifecycle and dependency management.
- The container creates the objects, wires them together, configures them, and manages their complete lifecycle from creation to destruction.
- In simple terms, the IoC container is a framework that helps manage the dependencies between the different components of an application by creating, configuring, and managing the instances of the classes.
- The Spring IoC container provides a few different implementation options, including the classic XML-based configuration and the newer Java-based configuration.
- Both approaches allow developers to declare the beans (the application's components) and their dependencies, leaving the framework responsible for managing and injecting the dependencies into the components as needed.
- The IoC container's primary responsibility is to facilitate loose coupling between the different application components, which makes the application more modular, easier to maintain and test, and less dependent on specific implementation details.
- 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

(3) Explain Spring bean life cycle.

- The life cycle of a Spring bean is easy to understand. When a bean is instantiated, it may be required to perform some initialization to get it into a usable state.
- Similarly, when the bean is no longer required and is removed from the container, some clean-up may be required.
- Though, there are lists of the activities that take place behind the scene between the time of bean Instantiation and its destruction, this chapter will discuss only two important bean life cycle call-back methods, which are required at the time of bean initialization and its destruction.
- To define setup and teardown for a bean, we simply declare the <bean> with initmethod and/or destroy-method parameters.
- The init-method attribute specifies a method that is to be called on the bean immediately upon instantiation.
- Similarly, destroy method specifies a method that is called just before a bean is removed from the container.
- In Spring, a bean is an object that is instantiated, assembled, and otherwise managed by the Spring IoC container. The life cycle of a Spring bean can be summarized as follows:
  - **Instantiation:** During this phase, the Spring container creates a new instance of the bean using the no-arg constructor or a factory method defined in the configuration metadata.
  - **Populating Properties:** During this phase, the Spring container sets the values of the bean's properties and dependencies using setters, constructors, or fields. This is known as Dependency Injection.
  - **Initialization:** During this phase, any custom initialization code for the bean is executed. This is usually done using the init-method attribute on the bean definition.
  - **Using:** Once the initialization is complete, the bean is ready to be used.
  - **Destruction:** During this phase, any clean-up code for the bean is executed. This is usually done using the destroy-method attribute on the bean definition

(4) What is Spring AOP? What are join points and cut points?

- Spring AOP (Aspect-Oriented Programming) is a programming paradigm that allows modularization of cross-cutting concerns such as logging, security, and transaction management in a software application.
- A cross-cutting concern is a feature that affects the entire application and can be difficult to modify without affecting the entire system.
- AOP enables separation of such concerns and allows developers to define reusable modules called aspects, which encapsulate the behaviour of the cross-cutting concern.
- In AOP, the core business logic of an application is divided into modules called objects, and the cross-cutting concerns are implemented as aspects.
- The AOP framework weaves the aspects into the objects at specific points in the object's execution. These points are called join points.
- Join points represent points in the application where an aspect can be applied. Examples of join points include method executions, field access, exception handling, and object instantiation.
- A cut point is a specific join point where an aspect is actually applied. Cut points are defined by pointcut expressions, which are used to match join points in the application where the aspect will be applied.
- AOP allows the separation of cross-cutting concerns from the core business logic, and join points and cut points are used to apply the cross-cutting concerns at specific points in the object's execution.