PART A (Marks : 02)

1 Distinguish between client and server-side scripting.

Feature	Client-Side Scripting	Server-Side Scripting
Execution Location	Executes in the user's browser.	Executes on the server before being sent to the client.
Language Examples	JavaScript, HTML, CSS	Java, Python, PHP, Node.js
Use Cases	Validations, interactivity, animations	Database interactions, data processing, session handling

2 Write a code segment to store current server time in session using Java servlet API

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.time.LocalDateTime;

public class TimeServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
      HttpSession session = request.getSession();
      session.setAttribute("currentServerTime", LocalDateTime.now());

      PrintWriter out = response.getWriter();
      out.println("Server time stored in session.");
  }
}
```

3 Enlist three methods that is central to the lifecycle of a servlet.

- **init()**: Initializes the servlet, called once when the servlet is first loaded.
- **service()**: Processes requests; called for each client request.
- **destroy()**: Cleans up resources before the servlet is destroyed.

4 Summarize the applications of servlets.

- Dynamic web content generation
- Data processing and database interaction
- Session management and user tracking
- E-commerce and transaction handling

5 Write the URL encoding of the string B&O Railroad down 3.2%.

Encoded string:

B%26O%20Railroad%20down%203.2%25

6 When are cookies created? how long does a cookie last?

- **When Cookies Are Created**: Cookies are created when the server sends the Set-Cookie header in the response to the client.
- **Duration of Cookies**: Cookies can last for a session (deleted on browser close) or have a specific expiration time.

7 How sessions are handled in Servlets?

Servlets use the HttpSession API to handle sessions. HttpSession allows you to store user-specific information on the server between HTTP requests, ensuring data consistency across interactions.

8 Summarize few techniques to implement data storage in web technology.

- Session Storage: Stores data for the user's session.
- Cookies: Stores small amounts of data on the client side.
- Database: Stores data persistently on the server.

9 Express the information in state of the thread.

1 **Write the purpose of Driver Manager.**

DriverManager manages a list of database drivers, establishes a connection to the database, and allows the retrieval of Connection objects for database interaction.

2 **Write the syntax for Connection String.**

String url = "jdbc:mysql://hostname:port/dbname";

3 **What do you mean by Class.forName?**

Class.forName("driverClassName") loads the JDBC driver class at runtime, allowing the Java application to establish a connection with a database.

4 **Write Java code for Create Connection Agent.**

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

public class DatabaseConnector {
   public Connection createConnection() {
      try {
         Class.forName("com.mysql.jdbc.Driver");
         return DriverManager.getConnection("jdbc:mysql://hostname:port/dbname", "username",
"password");
      } catch (ClassNotFoundException | SQLException e) {
         e.printStackTrace();
         return null;
      }
   }
}
```

1 **Write a xml code to describe a baseball player information.**

```
<player>
  <name>John Doe</name>
  <team>Yankees</team>
  <position>Pitcher</position>
  <average>0.289</average>
</player>
```

2 **What is use of XML Namespace and give syntax**

Purpose: XML Namespaces are used to avoid element name conflicts in XML documents by qualifying names of elements and attributes.

Syntax:

```
<root xmlns:prefix="namespaceURI">
  <prefix:element>Content</prefix:element>
</root>
```

3 **How is XML parsing done with SAX?**

- SAX (Simple API for XML) is an event-driven XML parser where each element triggers an event. The parser reads the XML document sequentially and invokes callback methods (like startElement, endElement) for handling.

4 **Express various XML schema types.**

- **Simple Types**: Define data constraints for elements and attributes (e.g., string, integer).
- **Complex Types**: Define structured data with nested elements and attributes.

5 **Indicate few examples of XML Vocabularies.**

- **MathML**: Used for describing mathematical notation.
- **SVG**: Used for scalable vector graphics.
- **XHTML**: An XML-based extension of HTML.

6 **Why is XSLT an important tool in development of web applications?**

XSLT (Extensible Stylesheet Language Transformations) is used to transform XML data into different formats like HTML, making it vital for rendering data-driven web content.

7 **How to deal with syntax errors on JSP page?**

Syntax errors in JSP can be managed by enabling the development environment's debugging tools, using try-catch blocks, and thoroughly testing the JSP page.

8 **Write a JSP code to retrieve data from database**

```
<%@ page import="java.sql.*" %>
<%
  Connection conn = null;
  Statement stmt = null;
```

```
    try {
        Class.forName("com.mysql.jdbc.Driver");
        conn = DriverManager.getConnection("jdbc:mysql://hostname:port/dbname", "username",
"password");
        stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM tableName");

        while (rs.next()) {
            out.println(rs.getString("columnName"));
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (stmt != null) stmt.close();
        if (conn != null) conn.close();
    }
%>
```

9 Compare JSP and Servlets.

Feature	JSP	Servlets
Usage	Primarily for presentation layer	Primarily for application logic and backend
Code Structure	Mixture of HTML and Java	Pure Java code
Maintenance	Easier for front-end modifications	More challenging for front-end modifications

10 How to access the Java Bean class? Give example

To access a JavaBean in JSP:

1. Define the bean:
   ```
   <jsp:useBean id="beanName" class="packageName.BeanClass" scope="session"/>
   ```
2. Set or get a property:
   ```
   <jsp:setProperty name="beanName" property="propertyName" value="value"/>
   <jsp:getProperty name="beanName" property="propertyName"/>
   ```

PART B (Marks: 16/12)

1 (i) **Write a simple servlet code to implement 'HelloWorld program'.**

(i) **Interpret additional HttpServlet methods with a java servlet program.**

i)

Simple Java Servlet Program: "Hello World"

HelloWorld.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class HelloWorld extends HttpServlet
{
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
 {
 //Set content type
response.setContentType("text/html");
 //Get the stream to write data
PrintWriter out = response.getWriter();
 //Write HTML in stream
out.println("<html><body>");
out.println("<h1>Hello World</h1>");
out.println("</body></html>");
 //Close the stream
 out.close();
 }}
```

web.xml

```
<web-app>
 <servlet>
<servlet-name>sampleprogram</servlet-name>
<servlet-class>HelloWorld</servlet-class>
 </servlet>
 <servlet-mapping>
<servlet-name>sampleprogram</servlet-name>
<url-pattern>/myprog</url-pattern>
 </servlet-mapping>
```

```
</web-app>
```

Web.xml – it is a deployment descriptor file used in java web applications.
• It follows servlet specification
• It is a configuration file- provide instruction to web container about how to deploy and manage the web application.

(ii) Additional HttpServlet Methods with Example

The HttpServlet class provides several other methods apart from doGet() and doPost(), each designed to handle specific HTTP methods.

```
// Import necessary packages
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

// Define a servlet with the URL pattern "/httpMethods"
@WebServlet("/httpMethods")
public class HttpMethodsServlet extends HttpServlet {

  // Method to handle HTTP GET requests
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<h1>This is a GET request</h1>");
    out.close();
  }
  // Method to handle HTTP POST requests
  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<h1>This is a POST request</h1>");
    out.close();
  }

  // Method to handle HTTP PUT requests
  protected void doPut(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<h1>This is a PUT request</h1>");
    out.close();
  }

  // Method to handle HTTP DELETE requests
```

```
    protected void doDelete(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>This is a DELETE request</h1>");
        out.close();
    }
}
```

Explanation: This servlet demonstrates additional HttpServlet methods:

doPost(): Handles HTTP POST requests, often used for submitting forms or sending data to the server.

doPut(): Handles HTTP PUT requests, typically used for updating data on the server.

doDelete(): Handles HTTP DELETE requests, commonly used to delete data on the server

2 (i) With a neat sketch and java servlet code, demonstrate the life cycle of a servlet.
(ii) Examine the architecture of servlet with a diagram.

The Servlet Life Cycle consists of a sequence of stages that a servlet goes through, from initialization to destruction. Here's a breakdown:

1. **Loading the Servlet Class**
   - When the web container (e.g., Tomcat) receives a request for a servlet that hasn't been loaded, it loads the servlet class into memory.
2. **Servlet Instantiation**
   - The web container creates a single instance of the servlet. This instance is used throughout the servlet's lifecycle to handle multiple requests, ensuring efficient resource usage.
3. **Initialization (init() method)**
   - The init() method is called only once by the web container right after the servlet instance is created. This is where any necessary initialization logic for the servlet is implemented, such as establishing database connections or loading configuration settings.
   - **Code Example**:

```
public void init(ServletConfig config) throws ServletException {
    // Initialization code here
}
```

4. **Processing Requests (service() method)**
   - The service() method is called each time the servlet receives a new client request. This method determines the type of request (e.g., GET, POST) and routes it to the corresponding doGet(), doPost(), doDelete(), etc., method based on the HTTP request type.
   - A new thread is created for each request, allowing the servlet to handle concurrent requests.
   - **Code Example**:

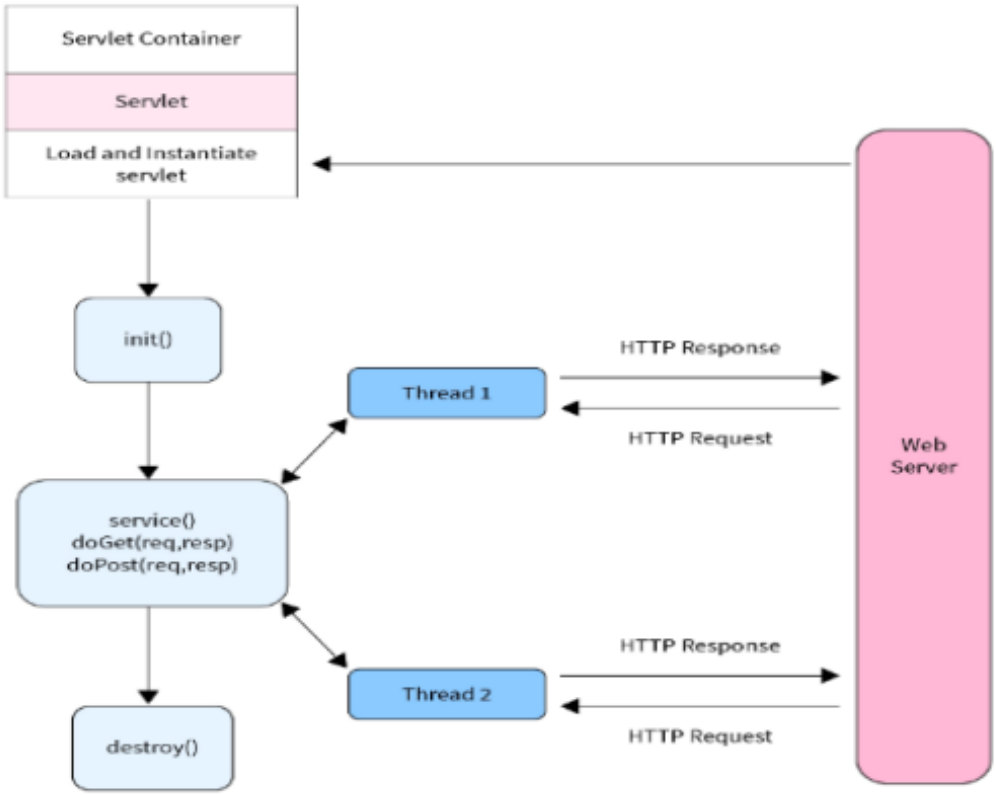
```
public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
    // Handles client requests (GET, POST, etc.)
}
```

5. **Destruction (destroy() method)**
   - The destroy() method is called when the servlet is about to be taken out of service, either due to server shutdown or re-deployment. It performs cleanup tasks, such as closing

database connections and freeing other resources.

- After destroy() is called, the servlet instance is eligible for garbage collection.
- **Code Example**:

```
public void destroy() {
   // Cleanup code here
}
```



- When a request is made for a servlet, the web container first loads and initializes the servlet if it hasn't been loaded already, calling the init() method.
- For each client request, the service() method is called. It determines the HTTP request type and calls the relevant method (doGet(), doPost(), etc.).
- Since servlets handle multiple requests concurrently, each request spawns a new thread.
- The destroy() method is called only when the servlet is about to be removed, ensuring that resources are properly released.

3 **With a java servlet example program, illustrate the how session tracking done using cookies.**

Java Servlet Session Management Example

1. **HTML Form (Index.java)**
   - The HTML form (Index.java) prompts the user to enter their name.
   - When the form is submitted, it sends a POST request to servlet1, which is mapped to FirstServlet.

```
<html>
  <head> <title>Servlet program</title> </head>
  <body>
    <div>
      <form action="servlet1" method="POST">
        Name: <input type="text" name="userName" />
        <br />
        <input type="submit" value="go" />
      </form>
    </div>
  </body>
</html>
```

2. **FirstServlet - Handling the Initial Request**
   - When the form is submitted, FirstServlet retrieves the userName parameter from the request.
   - A new HttpSession object is created (or retrieved if it already exists), and the user name is stored in the session as an attribute with the key "uname".
   - The servlet then displays a welcome message and provides a link (visit) for the user to revisit the servlet.

```
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
```

```
public class FirstServlet extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    // Retrieve userName from form input
    String n = request.getParameter("userName");

    // Display a welcome message with the user name
    out.println("Welcome " + n);

    // Create or retrieve session and store user name
    HttpSession session = request.getSession();
    session.setAttribute("uname", n);

    // Link to SecondServlet
    out.println("<a href='servlet2'> visit</a>");
    out.close();
  }
}
```

3. **SecondServlet - Retrieving Session Data**
   - When the user clicks on the "visit" link, SecondServlet is invoked with a GET request.
   - SecondServlet retrieves the session using request.getSession(false), which prevents creating a new session if one doesn't exist.
   - The uname attribute is retrieved from the session, and the user is greeted with "Hello" and their name.

```
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
public class SecondServlet extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
```

```
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // Retrieve the existing session (if it exists)
        HttpSession session = request.getSession(false);
        // Retrieve user name from session attribute
        String n = (String) session.getAttribute("uname");
        out.print("Hello " + n);
        out.close();
    }
}
```

4. **Web Deployment Descriptor (web.xml)**
   - This file maps URLs to servlets, enabling the server to route requests to the correct servlet based on the URL pattern.
   - servlet1 is mapped to FirstServlet, and servlet2 is mapped to SecondServlet.

```
<web-app>
  <servlet>
    <servlet-name>s1</servlet-name>
    <servlet-class>FirstServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>s1</servlet-name>
    <url-pattern>/servlet1</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>s2</servlet-name>
    <servlet-class>SecondServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>s2</servlet-name>
    <url-pattern>/servlet2</url-pattern>
  </servlet-mapping>
</web-app>
```

4 **Write a program that allows the user to select a favorite programming language and post the choice to the server. The response is a web page in which the user can click a link to view a list of book recommendations. The cookie previously stored on the client are read by the servlet and form a web page contains the book recommendation. Use servlet, cookies and HTML**

1. HTML Form (index.html)

This form allows the user to choose a favorite programming language. The form sends a POST request to the SelectLanguageServlet to save the choice.

```
<!DOCTYPE html>
<html>
<head>
  <title>Select Your Favorite Programming Language</title>
</head>
<body>
  <h1>Select Your Favorite Programming Language</h1>
  <form action="selectLanguage" method="POST">
    <label for="language">Choose a language:</label>
    <select name="language" id="language">
      <option value="Java">Java</option>
      <option value="Python">Python</option>
      <option value="JavaScript">JavaScript</option>
      <option value="C++">C++</option>
    </select>
    <br><br>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

2. SelectLanguageServlet - Setting the Favorite Language Cookie

This servlet handles the user's selection, sets a cookie to store the chosen programming language, and responds with a page containing a link to view recommended books.

```
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
```

```
@WebServlet("/selectLanguage")
public class SelectLanguageServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Get the selected language from the form
        String language = request.getParameter("language");

        // Set a cookie to store the user's favorite language
        Cookie languageCookie = new Cookie("favoriteLanguage", language);
        languageCookie.setMaxAge(60 * 60 * 24); // Cookie lasts for one day
        response.addCookie(languageCookie);

        // Display a confirmation message with a link to view book recommendations
        out.println("<html><body>");
        out.println("<h2>Your favorite language is " + language + "</h2>");
        out.println("<p><a href='bookRecommendations'>View Book Recommendations</a></p>");
        out.println("</body></html>");
    }
}
```

3. BookRecommendationsServlet - Generating Book Recommendations

This servlet reads the favoriteLanguage cookie, and based on its value, it provides a list of recommended books for the chosen programming language.

```
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/bookRecommendations")
public class BookRecommendationsServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Retrieve the favorite language from cookies
        String favoriteLanguage = null;
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                if ("favoriteLanguage".equals(cookie.getName())) {
                    favoriteLanguage = cookie.getValue();
                    break;
                }
            }
        }
```

```
        // Generate the book recommendations based on the favorite language
        out.println("<html><body>");
        out.println("<h1>Book Recommendations for " + (favoriteLanguage != null ? favoriteLanguage
    : "Your Selected Language") + "</h1>");

        if (favoriteLanguage == null) {
            out.println("<p>No favorite language selected.</p>");
        } else {
            out.println("<ul>");
            switch (favoriteLanguage) {
                case "Java":
                    out.println("<li>Effective Java by Joshua Bloch</li>");
                    out.println("<li>Java: The Complete Reference by Herbert Schildt</li>");
                    break;
                case "Python":
                    out.println("<li>Python Crash Course by Eric Matthes</li>");
                    out.println("<li>Fluent Python by Luciano Ramalho</li>");
                    break;
                case "JavaScript":
                    out.println("<li>JavaScript: The Good Parts by Douglas Crockford</li>");
                    out.println("<li>You Don't Know JS by Kyle Simpson</li>");
                    break;
                case "C++":
                    out.println("<li>The C++ Programming Language by Bjarne Stroustrup</li>");
                    out.println("<li>Effective Modern C++ by Scott Meyers</li>");
                    break;
                default:
                    out.println("<li>General Programming Books</li>");
                    break;
            }
            out.println("</ul>");
        }
        out.println("</body></html>");
    }
}
```

4. web.xml - Servlet Mappings (if not using @WebServlet annotation)

In case servlet annotations are not preferred, use web.xml to configure servlet mappings:

```
<web-app>
    <servlet>
        <servlet-name>SelectLanguageServlet</servlet-name>
        <servlet-class>SelectLanguageServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>SelectLanguageServlet</servlet-name>
        <url-pattern>/selectLanguage</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>BookRecommendationsServlet</servlet-name>
        <servlet-class>BookRecommendationsServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>BookRecommendationsServlet</servlet-name>
        <url-pattern>/bookRecommendations</url-pattern>
```

```
    </servlet-mapping>
</web-app>
```

Explanation of Flow:

1. **User Selection**: The user selects a programming language from the index.html form and submits it to SelectLanguageServlet.
2. **Setting Cookie**: SelectLanguageServlet saves the selected language as a cookie named favoriteLanguage.
3. **Viewing Recommendations**: The user clicks the link to BookRecommendationsServlet, which reads the favoriteLanguage cookie and displays book recommendations accordingly.

5 **Examine the key issues arises due to concurrency in servlets. Address the issues with solution.**

1. Race Conditions

- **Issue**: Race conditions occur when multiple threads access shared resources simultaneously, leading to unpredictable or inconsistent data.
- **Solution**: To prevent race conditions:
    - **Avoid Using Instance Variables**: Since servlets are multithreaded, each request is handled by a separate thread but shares the same instance of the servlet class. Avoid using instance variables, which could be modified by multiple threads simultaneously. Instead, store request-specific data in local variables within methods, as local variables are thread-safe.
    - **Use Synchronized Blocks Carefully**: If shared resources (e.g., counters) are necessary, use synchronized blocks to limit access to one thread at a time. However, excessive synchronization can impact performance.

```
// Example of synchronized block to handle shared resource
public void incrementCounter() {
   synchronized(this) {
      counter++; // Only one thread at a time can modify counter
   }
}
```

2. Data Inconsistency and Stale Data

- **Issue**: When multiple requests modify shared data simultaneously, data inconsistency can arise. Similarly, stale data occurs when one thread reads data that another thread has already updated.
- **Solution**:
    - **Atomic Variables**: For simple operations (e.g., counting), use atomic classes like AtomicInteger, which provide thread-safe operations.
    - **Double-checked Locking with Caching**: If you have to cache data (e.g., database results), implement double-checked locking to update only when necessary.

```
// Example using AtomicInteger for thread-safe increment
private AtomicInteger counter = new AtomicInteger(0);

public void incrementCounter() {
```

```
    counter.incrementAndGet(); // Atomic operation
  }
```

3. Deadlock and Resource Contention

- **Issue**: Deadlock occurs when two or more threads wait indefinitely for resources that are locked by each other. Resource contention happens when threads compete for limited resources, leading to slow response times.
- **Solution**:
  - **Minimize Locks and Use Reentrant Locks**: Minimize the use of locks, and where necessary, use ReentrantLock with timeout to avoid blocking indefinitely.
  - **Connection Pooling**: Use connection pooling for resources like database connections, file handles, or network sockets. This ensures efficient management and reduces contention.

```
// Example using ReentrantLock with timeout
  private final ReentrantLock lock = new ReentrantLock();

  public void accessResource() {
    try {
      if (lock.tryLock(1, TimeUnit.SECONDS)) { // Attempt to acquire lock
        // Critical section
      }
    } catch (InterruptedException e) {
      // Handle interruption
    } finally {
      lock.unlock();
    }
  }
```

Practices to Manage Concurrency in Servlets

1. **Servlet Context Attribute Synchronization**: For shared data across the application (like configuration or application-level counters), use the ServletContext object with synchronized blocks or atomic operations.
2. **Use Stateless Servlets**: Design servlets as stateless components wherever possible. Stateless servlets don't retain information between requests, minimizing concurrency issues.
3. **Thread-safe Collections**: For shared collections, use thread-safe collections like ConcurrentHashMap instead of HashMap

1 (i) Demonstrate the steps to connect servlet page to a database using JDBC.
(ii) Write a java servlet code to illustrate database connectivity using JDBC.

(i) Steps to Connect a Servlet Page to a Database Using JDBC

1. **Load the JDBC Driver**: Load the appropriate JDBC driver class to enable the servlet to communicate with the database.
    - Example: Class.forName("com.mysql.cj.jdbc.Driver"); for MySQL.
2. **Establish the Database Connection**: Create a connection to the database by specifying the URL, username, and password. Use the DriverManager class to establish this connection.
    - Example: Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/db_name", "username", "password");
3. **Create a Statement**: Prepare a Statement or PreparedStatement object to execute SQL queries.
    - Example: PreparedStatement stmt = conn.prepareStatement("SELECT * FROM users WHERE id = ?");
4. **Execute the Query**: Execute the SQL query through the statement object and retrieve results, if any.
    - Example: ResultSet rs = stmt.executeQuery();
5. **Process the Results**: Process the ResultSet to fetch data from the query execution.
6. **Close the Connection**: Close the ResultSet, Statement, and Connection objects to free up resources.

(ii) Java Servlet Code to Illustrate Database Connectivity Using JDBC

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/UserServlet")
public class UserServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String jdbcUrl = "jdbc:mysql://localhost:3306/db_name";
      String jdbcUser = "username";
      String jdbcPassword = "password";
      Connection conn = null;
      PreparedStatement stmt = null;
      ResultSet rs = null;

      try {
         // Step 1: Load the JDBC driver
         Class.forName("com.mysql.cj.jdbc.Driver");

         // Step 2: Establish a connection
         conn = DriverManager.getConnection(jdbcUrl, jdbcUser, jdbcPassword);

         // Step 3: Create a SQL statement
         String sql = "SELECT * FROM users";
         stmt = conn.prepareStatement(sql);

         // Step 4: Execute the query
         rs = stmt.executeQuery();

         // Step 5: Process the results
         out.println("<html><body><h2>User List</h2><ul>");
         while (rs.next()) {
            String userName = rs.getString("name");
            out.println("<li>" + userName + "</li>");
         }
         out.println("</ul></body></html>");

      } catch (ClassNotFoundException e) {
         e.printStackTrace();
         out.println("<p>Error loading JDBC driver.</p>");
      } catch (SQLException e) {
         e.printStackTrace();
         out.println("<p>Error connecting to database.</p>");
      } finally {
         // Step 6: Close resources
         try {
            if (rs != null) rs.close();
```

```
            if (stmt != null) stmt.close();
            if (conn != null) conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
  }
}
```

Explanation of the Code

1. **Annotation**: @WebServlet("/UserServlet") binds the servlet to a specific URL, in this case, /UserServlet.
2. **Database Connection**: The JDBC URL, username, and password are defined to connect to the database.
3. **JDBC Steps**:
   - **Driver Loading**: The MySQL driver is loaded.
   - **Connection Establishment**: Using DriverManager, the servlet establishes a connection.
   - **SQL Statement Execution**: The servlet prepares and executes a SELECT query to fetch user data.
4. **HTML Output**: The servlet dynamically creates an HTML page with user data retrieved from the database.
5. **Resource Management**: The ResultSet, Statement, and Connection are closed in the finally block to prevent resource leaks.

2 Develop a shopping cart application using servlets and JDBC, where users can add, remove, and view items in their cart.

1. Set Up the MySQL Database

First, you'll need a database to store the product information and cart details.

Database Schema

```
-- Create database
CREATE DATABASE ShoppingCart;

-- Use the database
USE ShoppingCart;

-- Create table for products
CREATE TABLE products (
    product_id INT AUTO_INCREMENT PRIMARY KEY,
    product_name VARCHAR(100),
    product_price DECIMAL(10, 2)
);

-- Insert some sample products
INSERT INTO products (product_name, product_price) VALUES ('Laptop', 999.99);
INSERT INTO products (product_name, product_price) VALUES ('Phone', 599.99);
INSERT INTO products (product_name, product_price) VALUES ('Headphones', 199.99);

-- Create table for shopping cart (for a specific session)
CREATE TABLE cart (
    cart_id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    product_id INT,
    quantity INT,
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

2. Set Up the Servlet Application

In your Java web application, you will need a servlet to handle the cart actions (add, remove, view).

Directory Structure:

```
webapp/
  ├── WEB-INF/
  │   ├── classes/
  │   ├── lib/
  │   └── web.xml
  ├── index.jsp
  └── CartServlet.java
```

3. Create the JDBC Utility Class

This class will help with connecting to the MySQL database.

JDBCUtil.java (Database Connection Utility)

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

public class JDBCUtil {
    private static final String URL = "jdbc:mysql://localhost:3306/ShoppingCart";
    private static final String USER = "root";  // Change this to your MySQL username
    private static final String PASSWORD = "password";  // Change this to your MySQL password

    public static Connection getConnection() throws SQLException {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver"); // Load MySQL JDBC driver
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (Exception e) {
            e.printStackTrace();
            throw new SQLException("Database connection error");
        }
    }
}
```

4. Create the Shopping Cart Servlet

This servlet will handle adding, removing, and viewing items in the cart.

CartServlet.java

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

public class CartServlet extends HttpServlet {

    // Display Cart
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String action = request.getParameter("action");
        if ("view".equals(action)) {
            viewCart(request, response);
        } else {
```

```
            listProducts(request, response);
        }
    }

    // Add to Cart
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String action = request.getParameter("action");
        if ("add".equals(action)) {
            addToCart(request, response);
        } else if ("remove".equals(action)) {
            removeFromCart(request, response);
        }
    }

    // Show Products
    private void listProducts(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        try (Connection conn = JDBCUtil.getConnection()) {
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM products");

            request.setAttribute("products", rs);
            RequestDispatcher dispatcher = request.getRequestDispatcher("index.jsp");
            dispatcher.forward(request, response);
        } catch (SQLException e) {
            e.printStackTrace();
            response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        }
    }

    // Add Item to Cart
    private void addToCart(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        int productId = Integer.parseInt(request.getParameter("productId"));
        int quantity = Integer.parseInt(request.getParameter("quantity"));
        HttpSession session = request.getSession();

        try (Connection conn = JDBCUtil.getConnection()) {
            // Check if the product is already in the cart
            PreparedStatement stmt = conn.prepareStatement("SELECT * FROM cart WHERE
product_id = ?");
            stmt.setInt(1, productId);
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {
                // Update the quantity
                PreparedStatement updateStmt = conn.prepareStatement("UPDATE cart SET quantity =
quantity + ? WHERE product_id = ?");
                updateStmt.setInt(1, quantity);
                updateStmt.setInt(2, productId);
                updateStmt.executeUpdate();
            } else {
                // Add new product to cart
```

```
            PreparedStatement insertStmt = conn.prepareStatement("INSERT INTO cart (product_id,
quantity) VALUES (?, ?)");
            insertStmt.setInt(1, productId);
            insertStmt.setInt(2, quantity);
            insertStmt.executeUpdate();
        }

        response.sendRedirect("CartServlet?action=view");
      } catch (SQLException e) {
        e.printStackTrace();
        response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
      }
    }

    // Remove Item from Cart
    private void removeFromCart(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
      int cartId = Integer.parseInt(request.getParameter("cartId"));

      try (Connection conn = JDBCUtil.getConnection()) {
        PreparedStatement stmt = conn.prepareStatement("DELETE FROM cart WHERE cart_id =
?");
        stmt.setInt(1, cartId);
        stmt.executeUpdate();

        response.sendRedirect("CartServlet?action=view");
      } catch (SQLException e) {
        e.printStackTrace();
        response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
      }
    }

    // View Cart
    private void viewCart(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
      try (Connection conn = JDBCUtil.getConnection()) {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM cart JOIN products ON cart.product_id
= products.product_id");

        request.setAttribute("cartItems", rs);
        RequestDispatcher dispatcher = request.getRequestDispatcher("cart.jsp");
        dispatcher.forward(request, response);
      } catch (SQLException e) {
        e.printStackTrace();
        response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
      }
    }
}
```

5. Create JSP Pages
index.jsp (Display Products)

```
<%@ page import="java.sql.ResultSet" %>
```

```
<html>
<head>
  <title>Shopping Cart</title>
</head>
<body>
  <h2>Products</h2>
  <table>
    <tr>
      <th>Product Name</th>
      <th>Price</th>
      <th>Action</th>
    </tr>
    <%
      ResultSet products = (ResultSet) request.getAttribute("products");
      while (products.next()) {
    %>
    <tr>
      <td><%= products.getString("product_name") %></td>
      <td><%= products.getDouble("product_price") %></td>
      <td>
        <form action="CartServlet" method="post">
          <input type="hidden" name="action" value="add">
          <input type="hidden" name="productId" value="<%= products.getInt("product_id") %>">
          Quantity: <input type="number" name="quantity" value="1" min="1">
          <button type="submit">Add to Cart</button>
        </form>
      </td>
    </tr>
    <%
      }
    %>
  </table>
</body>
</html>
```

cart.jsp (View Cart)

jsp

Copy code

```
<%@ page import="java.sql.ResultSet" %>
<html>
<head>
  <title>Your Cart</title>
</head>
<body>
  <h2>Your Cart</h2>
  <table>
    <tr>
      <th>Product Name</th>
      <th>Quantity</th>
      <th>Price</th>
      <th>Action</th>
    </tr>
    <%
      ResultSet cartItems = (ResultSet) request.getAttribute("cartItems");
      while (cartItems.next()) {
```

```
        %>
        <tr>
          <td><%= cartItems.getString("product_name") %></td>
          <td><%= cartItems.getInt("quantity") %></td>
          <td><%= cartItems.getDouble("product_price") * cartItems.getInt("quantity") %></td>
          <td>
            <form action="CartServlet" method="post">
              <input type="hidden" name="action" value="remove">
              <input type="hidden" name="cartId" value="<%= cartItems.getInt("cart_id") %>">
              <button type="submit">Remove</button>
            </form>
          </td>
        </tr>
        <%
          }
        %>
      </table>
    </body>
    </html>
```

6. Configure the web.xml File

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
       version="3.0">
  <servlet>
    <servlet-name>CartServlet</servlet-name>
    <servlet-class>CartServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>CartServlet</servlet-name>
    <url-pattern>/CartServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

7. Run the Application

1. Compile the servlet classes and deploy the web application on your Tomcat server.
2. Access the application in your browser (http://localhost:8080/yourapp/).
3. You can now add items to your cart, view your cart, and remove items.

This is a simple shopping cart application using servlets and JDBC, which allows users to interact with a database and manage their cart. You can further enhance this application by adding user authentication, updating cart quantities, and implementing payment features.

1 Examine an XML document that marks up a business letter with DTD information.

XML Document with DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE letter SYSTEM "letter.dtd">
<letter>
  <sender>
    <name>John Doe</name>
    <address>123 Main St, City, Country</address>
    <email>johndoe@example.com</email>
  </sender>
  <recipient>
    <name>Jane Smith</name>
    <address>456 Elm St, City, Country</address>
```

```
    <email>janesmith@example.com</email>
  </recipient>
  <subject>Meeting Request</subject>
  <body>
    <p>Dear Jane,</p>
    <p>I hope this email finds you well. I would like to schedule a meeting to discuss our upcoming project.</p>
    <p>Best regards,</p>
    <p>John</p>
  </body>
</letter>
```

DTD for Business Letter (letter.dtd)

```
<!ELEMENT letter (sender, recipient, subject, body)>
<!ELEMENT sender (name, address, email)>
<!ELEMENT recipient (name, address, email)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (p+)>
<!ELEMENT p (#PCDATA)>
```

2 Construct schemas for specifying XML document structure and validating XML documents with an example.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="letter">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="sender">
          <xs:complexType>
            <xs:sequence>
```

```
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="email" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="recipient">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="email" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="subject" type="xs:string"/>
      <xs:element name="body">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="p" maxOccurs="unbounded" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

3 (i) Collect information about Extensible style sheet language.

(ii) Create an XML document that marks up various sports and their descriptions. Use XSLT to tabulate neatly the elements and attributes of the document.

i)

XSL (Extensible Stylesheet Language) is a language for expressing stylesheets that transform XML

documents into other forms like HTML, text, or other XML formats. XSL consists of three parts:
XSLT (XSL Transformations): Used to transform XML documents into different formats.
XPath: A language for navigating XML documents and extracting values.
XSL-FO (XSL Formatting Objects): Used for formatting XML data into a printable format.

ii)

XML Document with Sports Data

```
<?xml version="1.0" encoding="UTF-8"?>
<sports>
  <sport>
    <name>Soccer</name>
    <description>A team sport played with a round ball.</description>
  </sport>
  <sport>
    <name>Basketball</name>
    <description>A team sport where two teams compete to score points by shooting a ball
through a hoop.</description>
  </sport>
  <sport>
    <name>Tennis</name>
    <description>A racket sport where players hit a ball over a net.</description>
  </sport>
</sports>
```

XSLT to Tabulate the Sports Data

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <style>
          table { border-collapse: collapse; width: 100%; }
          th, td { padding: 8px; border: 1px solid #ddd; text-align: left; }
        </style>
      </head>
      <body>
        <h2>Sports List</h2>
        <table>
          <tr>
            <th>Sport</th>
            <th>Description</th>
          </tr>
          <xsl:for-each select="sports/sport">
            <tr>
              <td><xsl:value-of select="name"/></td>
              <td><xsl:value-of select="description"/></td>
            </tr>
          </xsl:for-each>
        </table>
```

```
        </body>
      </html>
    </xsl:template>
  </xsl:stylesheet>
```

4 Create a JSP application to insert and then retrieve the following employee details of a company from the database and display
   1) Employee ID
   2) First Name
   3) Last Name

3) Age

JSP Application for Employee Details

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
  <title>Employee Details</title>
</head>
<body>
  <h2>Enter Employee Details</h2>
  <form action="InsertEmployee.jsp" method="POST">
    Employee ID: <input type="text" name="emp_id"><br><br>
    First Name: <input type="text" name="first_name"><br><br>
    Last Name: <input type="text" name="last_name"><br><br>
    Age: <input type="text" name="age"><br><br>
    <input type="submit" value="Insert">
  </form>

  <h2>Employee Details</h2>
  <table border="1">
    <tr>
      <th>Employee ID</th>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Age</th>
    </tr>
    <c:forEach var="emp" items="${employees}">
      <tr>
        <td>${emp.emp_id}</td>
        <td>${emp.first_name}</td>
        <td>${emp.last_name}</td>
        <td>${emp.age}</td>
      </tr>
    </c:forEach>
  </table>
</body>
</html>
```

InsertEmployee.jsp

```
<%@ page import="java.sql.*" %>
<%
  String emp_id = request.getParameter("emp_id");
  String first_name = request.getParameter("first_name");
  String last_name = request.getParameter("last_name");
  String age = request.getParameter("age");

  try {
```

```
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/your_database", "username", "password");
        String query = "INSERT INTO employees (emp_id, first_name, last_name, age) VALUES (?, ?, ?, ?)";
        PreparedStatement ps = con.prepareStatement(query);
        ps.setString(1, emp_id);
        ps.setString(2, first_name);
        ps.setString(3, last_name);
        ps.setInt(4, Integer.parseInt(age));
        ps.executeUpdate();
    } catch (Exception e) {
        out.println("Error: " + e.getMessage());
    }
%>
```

5 Demonstrate a method for executing JSP documents via a web browser.

Step 1: Install Apache Tomcat on Windows

1. **Download Tomcat**:
   - Go to the Apache Tomcat website and download the latest stable version of Tomcat (e.g., Tomcat 9).
2. **Install Tomcat**:
   - Extract the downloaded .zip file to a directory, for example: C:\Apache Tomcat.
3. **Set Up Environment Variables** (Optional but recommended):
   - **Set CATALINA_HOME**: This will help you refer to Tomcat easily from the command prompt.
     - Right-click on **This PC** or **My Computer** > **Properties** > **Advanced System Settings** > **Environment Variables**.
     - Under **System Variables**, click **New**, and set:
       - **Variable Name**: CATALINA_HOME
       - **Variable Value**: C:\Apache Tomcat (or wherever you installed Tomcat).
   - This step is optional, but it can make navigating and executing Tomcat commands easier.

Step 2: Start Tomcat

1. **Navigate to Tomcat's bin directory**:
   - Open **File Explorer** and go to the directory where you installed Tomcat (e.g., C:\Apache Tomcat\bin).
2. **Start Tomcat**:
   - Double-click the startup.bat file to start Tomcat.
   - A command prompt will open and Tomcat will begin running. You will see logs indicating that Tomcat has started.
3. **Verify Tomcat is Running**:
   - Open your web browser and go to http://localhost:8080. If Tomcat is running, you should see the Tomcat welcome page.

Step 3: Create a Web Application for JSP

1. **Create a New Web Application Folder**:
   - Go to the webapps directory inside your Tomcat installation folder (e.g., C:\Apache Tomcat\webapps).
   - Create a new folder for your web application (e.g., mywebapp).
2. **Create the Web Content Folder**:
   - Inside the mywebapp folder, create a folder called WEB-INF. This is where Tomcat expects configuration files to reside.

Structure Example:

```
mywebapp/
  ├── WEB-INF/
  |       └── web.xml  (configuration file)
  └── index.jsp  (your JSP file)
```

3. **Create Your JSP File**:
   - Inside the mywebapp folder (directly under it, not inside WEB-INF), create a index.jsp file. Add the following basic JSP content:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<html>
<head>
  <title>My First JSP</title>
</head>
<body>
  <h2>Hello, this is a JSP page!</h2>
```

```
</body>
</html>
```

Step 4: Access the JSP Page from Your Browser

1. **Start Tomcat (if it's not already running)**:
   - Ensure that Tomcat is running by checking the command prompt window or by confirming it's visible in your Task Manager.
2. **Access Your JSP Page**:
   - In the browser, go to:
     http://localhost:8080/mywebapp/index.jsp
   - You should see the message "Hello, this is a JSP page!" rendered on the page.

Step 5: Optional: Set Up a Database Connection (e.g., MySQL)

If you want your JSP page to connect to a database (e.g., MySQL), follow these steps:

1. **Install MySQL (if not installed)**:
   - Download and install MySQL from MySQL's website.
2. **Set Up a JDBC Data Source**:
   - Go to the conf folder inside your Tomcat installation directory (e.g., C:\Apache Tomcat\conf).
   - Open the context.xml file and add the following inside the <Context> tag to set up the connection:

```
<Resource name="jdbc/myDB" auth="Container"
    type="javax.sql.DataSource"
    username="root" password="password"
    driverClassName="com.mysql.cj.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/mydatabase"
    maxTotal="20" maxIdle="10" maxWaitMillis="-1"/>
```

3. **Add MySQL JDBC Driver**:
   - Download the MySQL JDBC driver (e.g., mysql-connector-java-x.x.x.jar) from MySQL Connector/J download page.
   - Place the .jar file in the lib directory of Tomcat (e.g., C:\Apache Tomcat\lib).
4. **Access Database in Your JSP Page**:
   - You can now use JDBC to query your database in the JSP file. For example:

```
<%@ page import="java.sql.*, javax.sql.*" %>
<%
   Connection con = null;
   Statement stmt = null;
   ResultSet rs = null;
   try {
      // Get the DataSource
      Context initCtx = new InitialContext();
      DataSource ds = (DataSource) initCtx.lookup("java:comp/env/jdbc/myDB");
      con = ds.getConnection();

      // Query the database
      stmt = con.createStatement();
      rs = stmt.executeQuery("SELECT * FROM employees");
      while (rs.next()) {
         out.println("Employee ID: " + rs.getInt("emp_id") + "<br>");
         out.println("First Name: " + rs.getString("first_name") + "<br>");
         out.println("Last Name: " + rs.getString("last_name") + "<br>");
         out.println("Age: " + rs.getInt("age") + "<br><br>");
      }
```

```
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try { if (rs != null) rs.close(); } catch (Exception e) {}
            try { if (stmt != null) stmt.close(); } catch (Exception e) {}
            try { if (con != null) con.close(); } catch (Exception e) {}
        }
    %>
```

Step 6: Stop Tomcat

1. **Stop Tomcat**:
    - To stop Tomcat, go to the bin directory (C:\Apache Tomcat\bin) and double-click shutdown.bat.
    - This will stop the Tomcat server.

6 With an example, illustrate how to write a class so that it can be recognized as a JavaBeans class by JSP

In Java, a JavaBeans class is a reusable software component that follows a specific set of conventions. When you use JavaBeans in JSP (JavaServer Pages), it allows you to easily access and manipulate the properties of the JavaBeans class in your JSP files.

JavaBeans Class Conventions:

1. **Public Constructor**: A public no-argument constructor.
2. **Private Properties**: The properties (fields) should be private.
3. **Getter and Setter Methods**: For each property, there should be a corresponding public getter and setter method.
4. **Serializable**: The JavaBeans class must implement the Serializable interface, which allows the object to be easily passed between servers.

Example: Creating a JavaBeans Class

Let's create a simple JavaBeans class to represent an Employee with the following attributes:

- employeeId (ID of the employee)
- firstName (first name of the employee)
- lastName (last name of the employee)
- age (age of the employee)

Step 1: Create the JavaBeans Class (Employee.java)

```
import java.io.Serializable;

public class Employee implements Serializable {
  private int employeeId;
  private String firstName;
  private String lastName;
  private int age;

  // Public no-argument constructor
  public Employee() {
  }

  // Getter and setter methods for employeeId
  public int getEmployeeId() {
    return employeeId;
  }

  public void setEmployeeId(int employeeId) {
    this.employeeId = employeeId;
  }

  // Getter and setter methods for firstName
  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
```

```
        this.firstName = firstName;
    }

    // Getter and setter methods for lastName
    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    // Getter and setter methods for age
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

Step 2: Use the JavaBeans Class in a JSP Page

Now, let's write a JSP page to use this Employee JavaBean. In the JSP page, you can instantiate the JavaBean and set/get its properties using EL (Expression Language) or scriptlets.

Example JSP Page (employee.jsp)

```
<%@ page import="java.util.*" %>
<%@ page import="yourpackage.Employee" %>

<html>
<head>
    <title>Employee Information</title>
</head>
<body>
    <h2>Employee Details</h2>

    <!-- Create and set the properties of the Employee JavaBean -->
    <jsp:useBean id="emp" class="yourpackage.Employee" scope="session" />

    <%
        // Setting values for the Employee JavaBean using scriptlets
        Employee employee = (Employee) pageContext.getAttribute("emp");

        employee.setEmployeeId(101);
        employee.setFirstName("John");
        employee.setLastName("Doe");
        employee.setAge(30);
    %>

    <!-- Display Employee Information using Expression Language (EL) -->
    <p><strong>Employee ID:</strong> ${emp.employeeId}</p>
```

```
<p><strong>First Name:</strong> ${emp.firstName}</p>
<p><strong>Last Name:</strong> ${emp.lastName}</p>
<p><strong>Age:</strong> ${emp.age}</p>

</body>
</html>
```

Explanation of the Code:

1. **<jsp:useBean>**:
   - This tag is used to declare and instantiate the JavaBean (Employee class) in the JSP page.
   - **id="emp"**: Specifies the variable name that will be used to access the JavaBean instance.
   - **class="yourpackage.Employee"**: Specifies the full class name of the JavaBean.
   - **scope="session"**: Indicates that the JavaBean instance will be stored in the session scope (can be page, request, session, or application).
2. **Setting Values (Using Scriptlets)**:
   - Scriptlets (<% %>) are used to set the properties of the JavaBean. In the example, the employee's properties are set through the set methods of the Employee class.
3. **Accessing Values (Using EL)**:
   - Expression Language (EL) is used to retrieve the values of the JavaBean properties. EL automatically calls the corresponding get methods of the JavaBean (e.g., ${emp.employeeId}).
   - **EL** is more readable and eliminates the need for scriptlets, which is why it's recommended over traditional scriptlets for JSP development.

Step 3: Compile and Deploy the Application

1. **Compile the JavaBeans Class**:
   - Save the Employee.java class in the src folder of your web application (or use the appropriate package).
   - Compile the class to generate the .class file.
2. **Deploy the Web Application**:
   - Place your compiled Java classes in the WEB-INF/classes directory of your web application.
   - Place the JSP page in the root or appropriate subdirectory of the web application.
3. **Start Tomcat and Access the JSP Page**:
   - Deploy your application to the Tomcat webapps folder.
   - Start Tomcat (or ensure it's running).
   - Open your browser and go to http://localhost:8080/yourapp/employee.jsp.

Output:

The JSP page will display the following information:

```
Employee Details
Employee ID: 101
First Name: John
Last Name: Doe
Age: 30
```