Explain JSP objects and write a JSP program to display the string "GGSIPU".

JSP objects are Java objects that are made available to JSP pages by the web container. They can be used to access information about the current request, the current response, the current JSP page, and the web application. They can be categorized into two types:

1. Implicit Objects: These are pre-defined objects automatically created and made available by the JSP container for every JSP page. We can directly access them without any explicit declaration. e.g. request, response, session, application, out, page, pageContext, config, exception.

2. Explicit Objects: These are Java objects that you explicitly declare and create within your JSP page using JavaBeans or other classes.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>GGSIPU</title>
</head>
<body>
<h1>GGSIPU</h1>
</body>
</html>
```

Write a XML Program to display the information about the Voter in the Voter Id Card. Describe your own tags and write appropriate DTD for it.

XML Program

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE voter-id-card SYSTEM "voter_id_card.dtd">

<voter-id-card id="ABC1234567">
  <voter-info>
    <name>John Doe</name>
    <father-name>Richard Doe</father-name>
    <mother-name>Jane Doe</mother-name>
    <dob>1980-01-01</dob>
    <address>
      <city>City</city>
      <state>State</state>
    </address>
  </voter-info>
  <constituency-info>
    <polling-station>XYZ Polling Station</polling-station>
    <booth-number>123</booth-number>
  </constituency-info>
```

```
</voter-id-card>
```

DTD

```
<!ELEMENT voter-id-card (voter-info, constituency-info)>

<!ELEMENT voter-info (name, father-name, mother-name, dob, address)>

<!ELEMENT name (#PCDATA)>
<!ELEMENT father-name (#PCDATA)>
<!ELEMENT mother-name (#PCDATA)>
<!ELEMENT dob (#PCDATA)>

<!ELEMENT address (city, state)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>

<!ELEMENT constituency-info (polling-station, booth-number)>
<!ELEMENT polling-station (#PCDATA)>
<!ELEMENT booth-number (#PCDATA)>

<!ATTLIST voter-id-card id ID #REQUIRED>
```

Using JDBC concepts write a program to perform the task of Insertion, Updation and Deletion of records in MS-Access database/ORACLE/MySQL database for Employee Management System. Assume appropriate table schema with suitable records inside it to perform the task.

```
CREATE TABLE employees (

    id INT PRIMARY KEY,

    name VARCHAR(255) NOT NULL

);
```

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

public class SimpleEmployeeManagement {

    private static final String DB_URL = "jdbc:mysql://localhost:3306/your_database_name";
    private static final String USER = "your_username";
    private static final String PASSWORD = "your_password";

    public static void main(String[] args) {
        try (Connection connection = DriverManager.getConnection(DB_URL, USER, PASSWORD))
{
            // Insert a new employee record
            insertEmployee(connection, 101, "John Doe");

            // Update an existing employee record
            updateEmployee(connection, 101, "John Doe Jr.");

            // Delete an employee record
            deleteEmployee(connection, 101);

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

    private static void insertEmployee(Connection connection, int id, String name) throws
SQLException {
        String insertQuery = "INSERT INTO employees (id, name) VALUES (?, ?)";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(insertQuery)) {
            preparedStatement.setInt(1, id);
            preparedStatement.setString(2, name);
            int rowsInserted = preparedStatement.executeUpdate();
            System.out.println(rowsInserted + " rows inserted.");
        }
    }

    private static void updateEmployee(Connection connection, int id, String name) throws
SQLException {
        String updateQuery = "UPDATE employees SET name=? WHERE id=?";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(updateQuery)) {
            preparedStatement.setString(1, name);
            preparedStatement.setInt(2, id);
            int rowsUpdated = preparedStatement.executeUpdate();
            System.out.println(rowsUpdated + " rows updated.");
        }
    }

    private static void deleteEmployee(Connection connection, int id) throws SQLException
{
        String deleteQuery = "DELETE FROM employees WHERE id=?";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(deleteQuery)) {
            preparedStatement.setInt(1, id);
            int rowsDeleted = preparedStatement.executeUpdate();
            System.out.println(rowsDeleted + " rows deleted.");
        }
    }
}
```

What is Servlet? Compare and Contrast between Generic and HTTP Servlet.

Servlet is a class that extends the capabilities of the servers and responds to the incoming requests.

Advantages: better performance, portable, robust, secure

Disadvantages:

1. If the number of clients increases, it takes more time for sending the response.
2. It uses platform dependent language

GenericServlet	HttpServlet
Extends object class and implements Servlet, ServletConfig and Serializable interfaces.	Extends GenericServlet and implements a Serializable interface
Defined by javax.servlet package	Defined by javax.servlet.http package
Immediate subclass of servlet interface	Immediate subclass of GenericServlet
Service method is abstract	Service method is non abstract
Protocol independent	Protocol dependent
Not commonly used	Commonly used

Write complete HTML Program to design a 'Library Management System'. Apply appropriate JavaScript Validations etc.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Library Management System</title>
</head>
<body>

    <h1>Library Management System</h1>

    <form id="bookForm" onsubmit="return validateForm()">
        <label for="bookTitle">Book Title:</label>
        <input type="text" id="bookTitle" name="bookTitle" required>

        <label for="author">Author:</label>
        <input type="text" id="author" name="author" required>

        <label for="publicationYear">Publication Year:</label>
        <input type="number" id="publicationYear" name="publicationYear"
min="1800" max="2023" required>

        <button type="submit">Add Book</button>
    </form>

    <script>
```

```
        function validateForm() {
            var bookTitle = document.getElementById('bookTitle').value;
            var author = document.getElementById('author').value;
            var publicationYear = 
document.getElementById('publicationYear').value;

            // Simple validation example (you can add more complex validations)
            if (bookTitle.trim() === '' || author.trim() === '' || 
isNaN(publicationYear)) {
                alert('Please fill in all fields correctly.');
                return false;
            }

            // If all validations pass, you can submit the form
            return true;
        }
    </script>

</body>
</html>
```

Session Tracking in Servlets/JSP

Session tracking is a mechanism used in web applications to maintain state information between multiple requests from the same client. There are four techniques for session tracking:

1. Cookies:

Description: Cookies are small pieces of data stored on the client's machine. They can be used to store session information.

Usage: Servlets and JSP can set and read cookies to track sessions. The `HttpServletResponse` object is used to set cookies, and the `HttpServletRequest` object is used to retrieve cookies.

2. URL Rewriting:

Description: Session information can be encoded in the URL itself. This is achieved by appending the session ID to each URL.

Usage: Servlets and JSP can use the `response.encodeURL()` method to include the session ID in URLs. The `HttpServletRequest` object automatically decodes the URL to retrieve the session ID.

3. Hidden Form Fields:

Description: Session information can be included in HTML forms as hidden input fields.

Usage: Servlets and JSP can include hidden form fields with session data, and the data is sent back to the server when the form is submitted.

4. HttpSession Object:

Description: Servlets and JSP use the `HttpSession` object to store and retrieve session-specific information. The container manages the creation and maintenance of the session.

Usage: Servlets can obtain the `HttpSession` object from the `HttpServletRequest`. JSP has implicit objects, and the `session` object represents the `HttpSession`.

Explain HTTP Request and Response Mechanism with example.

HTTP Request:

1. Request Method: The client initiates communication by sending an HTTP request to the server. The request begins with a request method, which indicates the action to be performed. Common methods include `GET`, `POST`, etc.

Example: GET /index.html HTTP/1.1

2. Headers: Following the request method, there are headers that provide additional information about the request. Headers include information such as the type of browser making the request, the type of data accepted by the client, etc.

Example: Host: www.example.com

3. Body (optional): For some request methods (like `POST`), a body can be included to send additional data to the server. This is common for submitting forms or uploading files.

Example:

POST /submit-form HTTP/1.1

username=johndoe&password=secret

HTTP Response:

1. Status Line: The server responds with a status line indicating the outcome of the request. It includes the HTTP version, a status code, and a status message.

Example: HTTP/1.1 200 OK

2. Headers: Similar to the request, the response includes headers providing additional information about the server's response, such as the type of content being sent, the date of the response, etc.

Example:

Content-Type: text/html

Content-Length: 349

3. Body: The body of the response contains the requested information or the result of the server's action.

Example:

```
 <!DOCTYPE html>

<html>
    <head>
         <title>Example Page</title>
     </head>
    <body>
         <h1>Hello, World!</h1>
    </body>
</html>
```

What are the implicit JSP objects? Enlist and explain.

Implicit JSP objects are pre-defined objects automatically created and made available by the JSP container for every JSP page. We can directly access them without any explicit declaration. e.g.

1. request: It represents the client's request and provides access to parameters sent in the request, such as form data or query parameters. It is an instance of the HttpServletRequest class. Example:

```
<% String username = request.getParameter("username"); %>
```

2. response: It allows manipulation of the HTTP response sent back to the client. It is an instance of the HttpServletResponse class. Example:

```
<% response.setContentType("text/html"); %>
```

3. out: It is an instance of the JspWriter class and provides a way to send content to the client's browser. It is used to write HTML or other content directly to the response. Example:

```
<% out.println("<h1>Hello, World!</h1>"); %>
```

4. session: It represents the user's session and provides a way to store and retrieve session-specific information. It is an instance of the HttpSession class. Example:

```
<% session.setAttribute("user", "JohnDoe"); %>
```

5. application: It represents the servlet context and provides a way to store and retrieve application-wide information. It is an instance of the ServletContext class. Example:

```
<% application.setAttribute("counter", 0); %>
```

6. pageContext: It provides access to various objects, including request, response, session, and application. It is an instance of the PageContext class. Example:

```
<% pageContext.setAttribute("message", "Welcome to JSP!"); %>
```

7. config: It provides access to the configuration information of the current JSP page. It is an instance of the ServletConfig class. Example:

```
<% String servletName = config.getServletName(); %>
```

8. page: It refers to the servlet class generated from the JSP page. It is similar to `this` in a servlet and is useful for calling methods defined in the servlet. Example:

```
<% pageContext.forward("newPage.jsp"); %>
```

How Java beans are deployed in a JSP page? Explain.

1. Create a JavaBean:

- Define a Java class with private instance variables.

- Provide public getter and setter methods to access and modify the values of these variables.

- The class should follow the JavaBeans naming conventions, with a default no-argument constructor.

```
public class MyBean {
    private String data;
    public MyBean() {
        // Default constructor
    }
    public String getData() {
        return data;
    }
    public void setData(String data) {
        this.data = data;
    }
}
```

2. Instantiate the JavaBean in the JSP page: Use the `<jsp:useBean>` tag to instantiate the JavaBean in the JSP page. This tag includes the bean in the page and makes it available for use.

```
<%@ page import="your.package.name.MyBean" %>
<jsp:useBean id="myBean" class="your.package.name.MyBean" scope="session" />
```

3. Access and use the JavaBean in the JSP page: Use the `<jsp:setProperty>` and `<jsp:getProperty>` tags to set and get the values of the JavaBean properties.

```
<jsp:setProperty name="myBean" property="data" value="Hello, JavaBean!" />
<p>Data from JavaBean: <jsp:getProperty name="myBean" property="data" /></p>
```

4. Display the JavaBean data in the JSP page: Use the JavaBean's getter methods to retrieve data and display it within HTML or other JSP tags.

```
<p>Data from JavaBean: <%= myBean.getData() %></p>
```