SRM Institute of Science and Technology

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Department of Computer Applications

Circular – 2023-24

B.Sc. CS 6th Sem

SERVICE ORIENTED ARCHITECTURE (UCS23D03J)

List of Programs

Lab 1: Create DTD file for student information and create a valid well-formed XML document to store student information against this DTD file

Title: Creating DTD and Valid XML Document for Student Information

Aim: To understand and implement Document Type Definitions (DTD) for defining the structure of an XML document and to create a well-formed XML document that is valid against the defined DTD.

Procedure:

1. **Define the Student DTD:**
   - Open a text editor (e.g., Notepad, VS Code).
   - Define the root element (e.g., `students`).
   - Define child elements for each student (e.g., `student`).
   - Define attributes for the `student` element (e.g., `id`).
   - Define sub-elements for `student` (e.g., `name`, `age`, `major`).
   - Save the file as `student.dtd`.
2. **Create the XML Document:**
   - Open a new text editor window.
   - Declare the XML version and encoding.
   - Link the XML document to the `student.dtd` file using a DOCTYPE declaration.
   - Create the root element and populate it with multiple `student` elements.
   - Ensure each `student` element contains the required sub-elements and attributes as defined in the DTD.
   - Save the file as `students.xml`.
3. **Validate the XML Document:**
   - Use an XML validator (e.g., online XML validator, XML editor with validation capabilities, or a simple Java/Python program using an XML parser) to check if `students.xml` is well-formed and valid against `student.dtd`.

Source Code:

`student.dtd`

```
<!ELEMENT students (student+)>
<!ELEMENT student (name, age, major)>
<!ATTLIST student id CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT major (#PCDATA)>
```

students.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE students SYSTEM "student.dtd">
<students>
    <student id="S001">
        <name>Alice Smith</name>
        <age>20</age>
        <major>Computer Science</major>
    </student>
    <student id="S002">
        <name>Bob Johnson</name>
        <age>21</age>
        <major>Electrical Engineering</major>
    </student>
    <student id="S003">
        <name>Charlie Brown</name>
        <age>19</age>
        <major>Information Technology</major>
    </student>
</students>
```

Input: No direct input is required for these files. The input is the content of the DTD and XML files themselves.

Expected Output:

- The `student.dtd` file should correctly define the structure for student information.
- The `students.xml` file should be well-formed (syntactically correct XML).
- When validated against `student.dtd`, `students.xml` should be reported as "valid," indicating it conforms to the DTD's rules. If there are errors, the validator should report them.

Lab 2: Create XMS schema for student information and create a valid well-formed XML document to store student information against this XMS schema file.

Title: Creating XML Schema (XSD) and Valid XML Document for Student Information

Aim: To understand and implement XML Schema Definition (XSD) for defining the structure and data types of an XML document, and to create a well-formed XML document that is valid against the defined XSD schema.

Procedure:

1. **Define the Student XSD Schema:**
   - Open a text editor.
   - Define the target namespace and schema location.
   - Define a complex type for `student` including elements like `name`, `age`, `major` with appropriate data types (e.g., `xs:string`, `xs:integer`).
   - Define an attribute for `student` (e.g., `id`).
   - Define a root element (e.g., `students`) that contains a sequence of `student` elements.
   - Save the file as `student.xsd`.
2. **Create the XML Document:**
   - Open a new text editor window.
   - Declare the XML version and encoding.
   - Reference the `student.xsd` schema using `xsi:schemaLocation` and the target namespace.
   - Create the root element and populate it with multiple `student` elements.
   - Ensure each `student` element contains the required sub-elements and attributes, adhering to the data types defined in the XSD.
   - Save the file as `students_xsd.xml`.
3. **Validate the XML Document:**
   - Use an XML schema validator (e.g., online XSD validator, XML editor with schema validation, or a Java/Python program using an XML parser that supports XSD validation) to check if `students_xsd.xml` is well-formed and valid against `student.xsd`.

Source Code:

`student.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://example.com/students"
           xmlns="http://example.com/students"
           elementFormDefault="qualified">

    <xs:element name="students">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="student" type="studentType"
maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="studentType">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="age" type="xs:integer"/>
            <xs:element name="major" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:string" use="required"/>
    </xs:complexType>

</xs:schema>
```

students_xsd.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<students xmlns="http://example.com/students"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://example.com/students student.xsd">
    <student id="S001">
        <name>Alice Smith</name>
        <age>20</age>
        <major>Computer Science</major>
    </student>
    <student id="S002">
        <name>Bob Johnson</name>
        <age>21</age>
        <major>Electrical Engineering</major>
    </student>
    <student id="S003">
        <name>Charlie Brown</name>
        <age>19</age>
        <major>Information Technology</major>
    </student>
</students>
```

Input: No direct input is required for these files. The input is the content of the XSD and XML files themselves.

Expected Output:

- The `student.xsd` file should correctly define the structure and data types for student information.
- The `students_xsd.xml` file should be well-formed.
- When validated against `student.xsd`, `students_xsd.xml` should be reported as "valid," indicating it conforms to the XSD's rules. Any deviation (e.g., incorrect data type, missing required element) should result in a validation error.

Lab 3: Using XSL display student information in tabular format.

Title: Displaying Student Information in Tabular Format using XSLT

Aim: To learn how to transform XML data into other formats, specifically HTML tables, using XSL Transformations (XSLT).

Procedure:

1. **Prepare the XML Data:** Use the `students_xsd.xml` file created in Lab 2, or a similar XML file containing student data.
2. **Create the XSLT Stylesheet:**
   - Open a text editor.
   - Define the XSLT version and root element (`xsl:stylesheet`).
   - Use `xsl:template` to match the root element (`/` or `students`).
   - Inside the template, create an HTML table structure (`<table>`, `<tr>`, `<th>`).
   - Use `xsl:for-each` to iterate over each `student` element.
   - Inside the `xsl:for-each` loop, create table rows (`<tr>`) and table data cells (`<td>`).
   - Use `xsl:value-of` to extract the values of `id` attribute, `name`, `age`, and `major` elements and place them into the respective table cells.
   - Save the file as `student_transform.xsl`.
3. **Perform the Transformation:**
   - Use an XSLT processor (e.g., a web browser capable of processing XSLT, a command-line XSLT tool like `xsltproc`, or a programming language like Java with JAXP) to apply `student_transform.xsl` to `students_xsd.xml`.
   - The output will be an HTML file.
4. **View the Output:** Open the generated HTML file in a web browser.

Source Code:

`students_xsd.xml` (Re-using from Lab 2 for consistency)

```
<?xml version="1.0" encoding="UTF-8"?>
<students xmlns="http://example.com/students"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://example.com/students student.xsd">
    <student id="S001">
        <name>Alice Smith</name>
        <age>20</age>
        <major>Computer Science</major>
    </student>
    <student id="S002">
        <name>Bob Johnson</name>
        <age>21</age>
        <major>Electrical Engineering</major>
    </student>
    <student id="S003">
        <name>Charlie Brown</name>
        <age>19</age>
        <major>Information Technology</major>
    </student>
</students>
```

`student_transform.xsl`

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

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:s="http://example.com/students">
    <xsl:template match="/">
        <html>
            <head>
                <title>Student Information</title>
                <style>
                    table {
                        width: 80%;
                        border-collapse: collapse;
                        margin: 20px auto;
                    }
                    th, td {
                        border: 1px solid #ddd;
                        padding: 8px;
                        text-align: left;
                    }
                    th {
                        background-color: #f2f2f2;
                    }
                    h1 {
                        text-align: center;
                    }
                </style>
            </head>
            <body>
                <h1>Student Details</h1>
                <table>
                    <thead>
                        <tr>
                            <th>ID</th>
                            <th>Name</th>
                            <th>Age</th>
                            <th>Major</th>
                        </tr>
                    </thead>
                    <tbody>
                        <xsl:for-each select="s:students/s:student">
                            <tr>
                                <td><xsl:value-of select="@id"/></td>
                                <td><xsl:value-of select="s:name"/></td>
                                <td><xsl:value-of select="s:age"/></td>
                                <td><xsl:value-of select="s:major"/></td>
                            </tr>
                        </xsl:for-each>
                    </tbody>
                </table>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Input: The `students_xsd.xml` file (or equivalent student XML data) and the `student_transform.xsl` stylesheet.

Expected Output: An HTML file that, when opened in a web browser, displays the student information in a well-formatted table with columns for ID, Name, Age, and Major.

Lab 4: Create web calculator service in NET Beans and consume it.

Title: Creating and Consuming a Web Calculator Service in NetBeans (Java)

Aim: To learn how to develop a simple SOAP-based web service using Java in NetBeans, deploy it, and then create a client application within NetBeans to consume this service.

Procedure:

Part A: Creating the Web Service

1. **Start NetBeans:** Open NetBeans IDE.
2. **Create New Project:**
   - Go to `File > New Project...`.
   - Select `Java Web > Web Application`. Click `Next`.
   - Give a project name (e.g., `CalculatorWebService`). Click `Next`.
   - Choose a server (e.g., GlassFish Server or Apache Tomcat). Click `Finish`.
3. **Create Web Service:**
   - In the `Projects` window, right-click on `CalculatorWebService` project.
   - Select `New > Web Service...`.
   - Give a web service name (e.g., `Calculator`).
   - Provide a package name (e.g., `com.example.calculator`). Click `Finish`.
4. **Implement Calculator Methods:**
   - NetBeans will generate a basic web service class.
   - Add methods for basic arithmetic operations (add, subtract, multiply, divide) to the `Calculator` class. Annotate them with `@WebMethod`.
   - Example:
   - `@WebMethod(operationName = "add")`
   - `public int add(@WebParam(name = "i") int i, @WebParam(name = "j") int j) {`
   - `    return i + j;`
   - `}`

5. **Deploy the Web Service:**
   - Right-click on the `CalculatorWebService` project and select `Clean and Build`.
   - Right-click again and select `Deploy`. NetBeans will deploy the service to the configured server.
6. **Verify WSDL:**
   - After deployment, NetBeans usually provides a link to the WSDL (Web Services Description Language) file in the output window or by right-clicking the web service and selecting "Test Web Service". This WSDL describes the service's operations.

Part B: Consuming the Web Service

1. **Create New Client Project:**
   - Go to `File > New Project...`.
   - Select `Java > Java Application`. Click `Next`.
   - Give a project name (e.g., `CalculatorServiceClient`). Click `Finish`.
2. **Add Web Service Client:**
   - In the `Projects` window, right-click on `CalculatorServiceClient` project.
   - Select `New > Web Service Client...`.

    - Choose `WSDL URL` and paste the WSDL URL of your deployed `CalculatorWebService`.
    - Provide a package name for the generated client artifacts (e.g., `com.example.calculator.client`). Click `Finish`.
3. **Implement Client Logic:**
    - NetBeans will generate a web service reference and client-side stubs.
    - In your `CalculatorServiceClient`'s `main` method (or another class), instantiate the web service proxy.
    - Call the web service methods (e.g., `add`, `subtract`) using the proxy object.
    - Print the results to the console.

Source Code:

`Calculator.java` (Web Service - part of CalculatorWebService project)

```
package com.example.calculator;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

@WebService(serviceName = "Calculator")
public class Calculator {

    /**
     * Web service operation for addition.
     * @param i The first integer.
     * @param j The second integer.
     * @return The sum of i and j.
     */
    @WebMethod(operationName = "add")
    public int add(@WebParam(name = "i") int i, @WebParam(name = "j") int j)
{
        return i + j;
    }

    /**
     * Web service operation for subtraction.
     * @param i The first integer.
     * @param j The second integer.
     * @return The difference of i and j.
     */
    @WebMethod(operationName = "subtract")
    public int subtract(@WebParam(name = "i") int i, @WebParam(name = "j")
int j) {
        return i - j;
    }

    /**
     * Web service operation for multiplication.
     * @param i The first integer.
     * @param j The second integer.
     * @return The product of i and j.
     */
    @WebMethod(operationName = "multiply")
    public int multiply(@WebParam(name = "i") int i, @WebParam(name = "j")
int j) {
        return i * j;
    }

    /**
     * Web service operation for division.
     * @param i The first integer (numerator).
     * @param j The second integer (denominator).
     * @return The quotient of i and j. Returns 0 if j is 0 to avoid division
by zero error.
     */
    @WebMethod(operationName = "divide")
    public double divide(@WebParam(name = "i") int i, @WebParam(name = "j")
int j) {
        if (j == 0) {
            System.out.println("Error: Division by zero.");
            return 0.0; // Or throw an exception
        }
        return (double) i / j;
    }
}
```

`Main.java` (Client - part of CalculatorServiceClient project)

```
package com.example.calculator.client;

public class Main {

    public static void main(String[] args) {
        try {
            // Create an instance of the generated web service client proxy
            com.example.calculator.Calculator_Service service = new
com.example.calculator.Calculator_Service();
            com.example.calculator.Calculator port =
service.getCalculatorPort();

            // Call the web service methods
            int sum = port.add(10, 5);
            System.out.println("10 + 5 = " + sum);

            int difference = port.subtract(10, 5);
            System.out.println("10 - 5 = " + difference);

            int product = port.multiply(10, 5);
            System.out.println("10 * 5 = " + product);

            double quotient = port.divide(10, 3);
            System.out.println("10 / 3 = " + quotient);

            double quotientByZero = port.divide(10, 0);
            System.out.println("10 / 0 = " + quotientByZero); // Will print
error message from service

        } catch (Exception ex) {
            System.out.println("An error occurred: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

Input: For the client, the input values are hardcoded in the `main` method (e.g., `10`, `5`, `3`, `0`).

Expected Output: The console output from the client application should show the results of the arithmetic operations:

```
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 3 = 3.3333333333333335
```

```
Error: Division by zero.
10 / 0 = 0.0
```

Lab 5: Create web calculator service in .NET and create client to consume this service

Title: Creating and Consuming a Web Calculator Service in .NET (C#)

Aim: To learn how to develop a simple SOAP-based web service using C# in Visual Studio (.NET Framework or .NET Core with WCF for SOAP), deploy it, and then create a client application to consume this service.

Procedure:

Part A: Creating the Web Service (ASP.NET Web Service - .NET Framework)

1. **Start Visual Studio:** Open Visual Studio.
2. **Create New Project:**
   - Go to `File > New > Project...`.
   - Select `ASP.NET Web Application (.NET Framework)`. Click `Next`.
   - Give a project name (e.g., `CalculatorWebServiceDotNet`). Click `Create`.
   - Choose `Empty` template and check `Web Forms` (or `Web API` if you want to use a more modern approach, but for SOAP, Web Forms is simpler here). Click `Create`.
3. **Add Web Service (ASMX):**
   - In `Solution Explorer`, right-click on the project.
   - Select `Add > New Item...`.
   - Search for `Web Service (ASMX)`. Give it a name (e.g., `CalculatorService.asmx`). Click `Add`.
4. **Implement Calculator Methods:**
   - Open `CalculatorService.asmx.cs`.
   - Uncomment `[System.Web.Script.Services.ScriptService]` if you plan to call it from JavaScript, otherwise leave it.
   - Add methods for basic arithmetic operations (add, subtract, multiply, divide). Annotate them with `[WebMethod]`.
   - Example:
   - `[WebMethod]`
   - `public int Add(int a, int b) {`
   - `    return a + b;`
   - `}`

5. **Build and Run:**
   - Build the solution (`Build > Build Solution`).
   - Run the project (`Debug > Start Without Debugging` or F5). This will deploy the service to IIS Express and open it in a browser.
6. **Verify WSDL:**
   - In the browser, you will see a page for `CalculatorService.asmx`. Click on the service name (e.g., `CalculatorService`) to see the list of available operations and a link to the WSDL (usually `http://localhost:port/CalculatorService.asmx?wsdl`).

Part B: Consuming the Web Service (Console Application)

1. **Create New Client Project:**
   - In the same solution, right-click on the solution in `Solution Explorer`.
   - Select `Add > New Project...`.

- Select `Console App (.NET Framework)` or `Console App (.NET Core)`. Click `Next`.
- Give a project name (e.g., `CalculatorServiceClientDotNet`). Click `Create`.

2. **Add Service Reference:**
   - In `Solution Explorer`, right-click on the `CalculatorServiceClientDotNet` project.
   - Select `Add > Service Reference...` (for .NET Framework) or `Add > Connected Service...` and then `Microsoft WCF Web Service` (for .NET Core).
   - In the "Add Service Reference" dialog:
     - Enter the WSDL URL of your `CalculatorService.asmx` (e.g., `http://localhost:port/CalculatorService.asmx?wsdl`).
     - Give a namespace (e.g., `CalculatorServiceReference`).
     - Click `Go` to discover services, then `OK`.
3. **Implement Client Logic:**
   - Open `Program.cs` in the client project.
   - Create an instance of the generated service client.
   - Call the web service methods using the client object.
   - Print the results to the console.

Source Code:

`CalculatorService.asmx.cs` (Web Service - part of CalculatorWebServiceDotNet project)

```
using System.Web.Services;

namespace CalculatorWebServiceDotNet
{
    /// <summary>
    /// Summary description for CalculatorService
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    // To allow this Web Service to be called from script, using ASP.NET
AJAX, uncomment the following line.
    // [System.Web.Script.Services.ScriptService]
    public class CalculatorService : System.Web.Services.WebService
    {
        [WebMethod]
        public int Add(int a, int b)
        {
            return a + b;
        }

        [WebMethod]
        public int Subtract(int a, int b)
        {
            return a - b;
        }

        [WebMethod]
        public int Multiply(int a, int b)
        {
            return a * b;
        }

        [WebMethod]
        public double Divide(int a, int b)
        {
            if (b == 0)
            {
                // You might want to throw a SOAP fault or return a specific
error value
                // For simplicity, returning 0.0 and printing to console
here.
                System.Console.WriteLine("Error: Division by zero."); // This
will appear in the server's console
                return 0.0;
            }
            return (double)a / b;
        }
    }
}
```

`Program.cs` (Client - part of CalculatorServiceClientDotNet project)

```
using System;
using CalculatorServiceClientDotNet.CalculatorServiceReference; // Adjust
namespace based on your service reference name

namespace CalculatorServiceClientDotNet
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create an instance of the generated service client
            CalculatorServiceSoapClient client = new
CalculatorServiceSoapClient();

            try
            {
                // Call the web service methods
                int sum = client.Add(20, 10);
                Console.WriteLine($"20 + 10 = {sum}");

                int difference = client.Subtract(20, 10);
                Console.WriteLine($"20 - 10 = {difference}");

                int product = client.Multiply(20, 10);
                Console.WriteLine($"20 * 10 = {product}");

                double quotient = client.Divide(20, 3);
                Console.WriteLine($"20 / 3 = {quotient}");

                double quotientByZero = client.Divide(20, 0);
                Console.WriteLine($"20 / 0 = {quotientByZero}"); // Will
print 0.0 from service

            }
            catch (Exception ex)
            {
                Console.WriteLine($"An error occurred: {ex.Message}");
            }
            finally
            {
                // Close the client to release resources
                if (client.State ==
System.ServiceModel.CommunicationState.Opened)
                {
                    client.Close();
                }
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}
```

Input: For the client, the input values are hardcoded in the `Main` method (e.g., `20`, `10`, `3`, `0`).

Expected Output: The console output from the client application should show the results of the arithmetic operations:

```
20 + 10 = 30
20 - 10 = 10
20 * 10 = 200
20 / 3 = 6.666666666666667
20 / 0 = 0
Press any key to exit...
```

(Note: The "Error: Division by zero." message from the service will appear in the server's output window, not the client's console directly, unless specifically handled with SOAP faults.)

Lab 6: Create java client to consume web service created in .NET

Title: Java Client Consuming a .NET Web Service

Aim: To demonstrate interoperability between different technology stacks by creating a Java client that consumes a SOAP-based web service developed in .NET.

Procedure:

1. **Ensure .NET Web Service is Running:** Make sure the `CalculatorWebServiceDotNet` (or similar .NET ASMX web service) created in Lab 5 is deployed and running (e.g., by running it from Visual Studio). Note down its WSDL URL (e.g., `http://localhost:port/CalculatorService.asmx?wsdl`).
2. **Create New Java Project in NetBeans/Eclipse:**
    - Open NetBeans or Eclipse.
    - Create a new `Java Application` project (e.g., `JavaDotNetClient`).
3. **Generate Java Client Stubs from WSDL:**
    - **NetBeans:** Right-click on the `JavaDotNetClient` project, select `New > Web Service Client...`. Choose `WSDL URL`, paste the .NET service's WSDL URL, and provide a package name (e.g., `com.example.dotnet.client`). Click `Finish`.
    - **Eclipse:** Right-click on the project, select `New > Other... > Web Services > Web Service Client`. Paste the WSDL URL and follow the wizard.
    - Alternatively, use `wsimport` command-line tool (part of JDK) to generate client stubs: `wsimport -keep -p com.example.dotnet.client http://localhost:port/CalculatorService.asmx?wsdl`
4. **Implement Client Logic:**
    - In the `Main` class of your Java client project, instantiate the generated web service proxy.
    - Call the web service methods (e.g., `add`, `subtract`) using the proxy object.
    - Print the results to the console.

Source Code:

(Assuming the .NET `CalculatorService.asmx` from Lab 5 is running)

`Main.java` (Java Client - part of JavaDotNetClient project)

```
package com.example.dotnet.client;

// Import the generated client classes (package name might vary based on
wsimport/IDE settings)
import com.example.calculatorwebservice.CalculatorService;
import com.example.calculatorwebservice.CalculatorServiceSoap;

public class Main {

    public static void main(String[] args) {
        try {
            // Create an instance of the generated web service client factory
            CalculatorService service = new CalculatorService();
            // Get the port/proxy to interact with the service
            CalculatorServiceSoap port = service.getCalculatorServiceSoap();

            // Call the web service methods
            int sum = port.add(30, 15);
            System.out.println("30 + 15 = " + sum);
            int difference = port.subtract(30, 15);
            System.out.println("30 - 15 = " + difference);

            int product = port.multiply(30, 5);
            System.out.println("30 * 5 = " + product);

            double quotient = port.divide(30, 4);
            System.out.println("30 / 4 = " + quotient);

            double quotientByZero = port.divide(30, 0);
            System.out.println("30 / 0 = " + quotientByZero); // Will print
0.0 from service

        } catch (Exception ex) {
            System.out.println("An error occurred while consuming .NET
service: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

Input: For the client, the input values are hardcoded in the `main` method (e.g., `30`, `15`, `5`, `4`, `0`).

Expected Output: The console output from the Java client application should show the results of the arithmetic operations:

```
30 + 15 = 45
30 - 15 = 15
30 * 5 = 150
30 / 4 = 7.5
30 / 0 = 0.0
```

Lab 7: Create .NET client to consume web service created in JAVA

Title: .NET Client Consuming a Java Web Service

Aim: To demonstrate interoperability between different technology stacks by creating a .NET client that consumes a SOAP-based web service developed in Java (e.g., using NetBeans).

Procedure:

1. **Ensure Java Web Service is Running:** Make sure the `CalculatorWebService` (or similar Java ASMX web service) created in Lab 4 is deployed and running (e.g., by running it from NetBeans/Eclipse). Note down its WSDL URL (e.g., `http://localhost:8080/CalculatorWebService/Calculator?wsdl`).
2. **Create New .NET Project in Visual Studio:**
   - Open Visual Studio.
   - Create a new `Console App (.NET Framework)` or `Console App (.NET Core)` project (e.g., `DotNetJavaClient`).
3. **Add Service Reference:**
   - In `Solution Explorer`, right-click on the `DotNetJavaClient` project.
   - Select `Add > Service Reference...` (for .NET Framework) or `Add > Connected Service...` and then `Microsoft WCF Web Service` (for .NET Core).
   - In the "Add Service Reference" dialog:
     - Enter the WSDL URL of your Java service (e.g., `http://localhost:8080/CalculatorWebService/Calculator?wsdl`).
     - Give a namespace (e.g., `JavaCalculatorServiceReference`).
     - Click `Go` to discover services, then `OK`.
4. **Implement Client Logic:**
   - Open `Program.cs` in the client project.
   - Create an instance of the generated service client.
   - Call the web service methods using the client object.
   - Print the results to the console.

Source Code:

(Assuming the Java `CalculatorWebService` from Lab 4 is running)

`Program.cs` (.NET Client - part of DotNetJavaClient project)

```
using System;
// Adjust namespace based on your service reference name
using DotNetJavaClient.JavaCalculatorServiceReference;

namespace DotNetJavaClient
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create an instance of the generated service client
            // The class name might vary, typically it's
ServiceNameSoapClient or ServiceNamePortClient
            CalculatorService client = new CalculatorService(); // Or
CalculatorPortClient for some Java services

            try
            {
                // Call the web service methods
                int sum = client.add(40, 20);
                Console.WriteLine($"40 + 20 = {sum}");

                int difference = client.subtract(40, 20);
                Console.WriteLine($"40 - 20 = {difference}");

                int product = client.multiply(40, 5);
                Console.WriteLine($"40 * 5 = {product}");

                double quotient = client.divide(40, 6);
                Console.WriteLine($"40 / 6 = {quotient}");

                double quotientByZero = client.divide(40, 0);
                Console.WriteLine($"40 / 0 = {quotientByZero}"); // Will
print 0.0 from service

            }
            catch (Exception ex)
            {
                Console.WriteLine($"An error occurred while consuming Java
service: {ex.Message}");
            }
            finally
            {
                // Close the client if it's a WCF client
                // if (client.State ==
System.ServiceModel.CommunicationState.Opened)
                // {
                //     client.Close();
                // }
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}
```

Input: For the client, the input values are hardcoded in the `Main` method (e.g., `40`, `20`, `5`, `6`, `0`).

Expected Output: The console output from the .NET client application should show the results of the arithmetic operations:

```
40 + 20 = 60
40 - 20 = 20
40 * 5 = 200
40 / 6 = 6.666666666666667
40 / 0 = 0
Press any key to exit...
```

Lab 8: Create java client to consume existing web service hosted in the internet

Title: Java Client Consuming an External Web Service

Aim: To learn how to create a Java client that consumes a publicly available SOAP-based web service hosted on the internet, demonstrating real-world web service consumption.

Procedure:

1. **Identify a Public Web Service:** Find a publicly available SOAP web service. A common example is a currency converter, weather service, or a simple calculator service. For this example, we'll assume a hypothetical "GlobalWeather" service.
   - *Example WSDL URL (hypothetical):* `http://www.webservicex.net/globalweather.asmx?wsdl` (Note: `webservicex.net` services might be unreliable or deprecated, use a reliable one if available).
2. **Create New Java Project in NetBeans/Eclipse:**
   - Open NetBeans or Eclipse.
   - Create a new `Java Application` project (e.g., `GlobalWeatherClient`).
3. **Generate Java Client Stubs from WSDL:**
   - **NetBeans:** Right-click on the `GlobalWeatherClient` project, select `New > Web Service Client...`. Choose `WSDL URL`, paste the external service's WSDL URL, and provide a package name (e.g., `com.example.weather.client`). Click `Finish`.
   - **Eclipse:** Right-click on the project, select `New > Other... > Web Services > Web Service Client`. Paste the WSDL URL and follow the wizard.
   - Alternatively, use `wsimport` command-line tool (part of JDK): `wsimport -keep -p com.example.weather.client http://www.webservicex.net/globalweather.asmx?wsdl`
4. **Implement Client Logic:**
   - In the `Main` class of your Java client project, instantiate the generated web service proxy.
   - Call the web service methods (e.g., `getWeather`, `getCitiesByCountry`) using the proxy object.
   - Print the results to the console.

Source Code:

(Using a hypothetical `GlobalWeather` service for demonstration)

`Main.java` (Java Client - part of GlobalWeatherClient project)

```
package com.example.weather.client;

// Imports for the generated client classes (adjust package and class names
based on actual WSDL)
// For webservicex.net/globalweather.asmx?wsdl, generated classes might be
in:
import net.webservicex.GlobalWeather;
import net.webservicex.GlobalWeatherSoap;

public class Main {

    public static void main(String[] args) {
        try {
            // Create an instance of the generated web service client factory
            GlobalWeather service = new GlobalWeather();
            // Get the port/proxy to interact with the service
            GlobalWeatherSoap port = service.getGlobalWeatherSoap();

            // Example 1: Get cities by country
            String countryName = "India";
            String citiesXml = port.getCitiesByCountry(countryName);
            System.out.println("Cities in " + countryName + ":\n" +
citiesXml);

            System.out.println("\n----------------------------------\n");

            // Example 2: Get weather for a specific city and country
            String cityName = "Mumbai";
            String weatherXml = port.getWeather(cityName, countryName);
            System.out.println("Weather in " + cityName + ", " + countryName
+ ":\n" + weatherXml);

        } catch (Exception ex) {
            System.out.println("An error occurred while consuming external
web service: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

Input: The input values are hardcoded in the `main` method (e.g., `countryName = "India"`, `cityName = "Mumbai"`).

Expected Output: The console output should display XML strings containing the requested information (cities list and weather details) from the external web service. The exact content will depend on the service's response.

Example (partial, actual output will be full XML):

```
Cities in India:
<NewDataSet>
  <Table>
    <City>Mumbai</City>
  </Table>
  <Table>
    <City>Delhi</City>
  </Table>
  ...
</NewDataSet>

----------------------------------

Weather in Mumbai, India:
<CurrentWeather>
  <Location>Mumbai, India</Location>
  <Time>...</Time>
  <Wind>...</Wind>
  <Visibility>...</Visibility>
  <SkyConditions>...</SkyConditions>
  <Temperature>...</Temperature>
  <DewPoint>...</DewPoint>
  <RelativeHumidity>...</RelativeHumidity>
  <Pressure>...</Pressure>
</CurrentWeather>
```

Lab 9: Create a RESTFUL web-services in Net beans

Title: Creating a RESTful Web Service in NetBeans (JAX-RS)

Aim: To learn how to develop a RESTful web service using Java and JAX-RS (Java API for RESTful Web Services) in NetBeans, demonstrating the principles of REST (Representational State Transfer).

Procedure:

1. **Start NetBeans:** Open NetBeans IDE.
2. **Create New Project:**
    - Go to `File > New Project...`.
    - Select `Java Web > Web Application`. Click `Next`.
    - Give a project name (e.g., `StudentRESTService`). Click `Next`.
    - Choose a server (e.g., GlassFish Server or Apache Tomcat) and Java EE version (e.g., Java EE 7 Web). Click `Next`.
    - Check `JAX-RS` under "Frameworks". Click `Finish`.
3. **Create RESTful Resource Class:**
    - In the `Projects` window, right-click on `StudentRESTService` project.
    - Select `New > RESTful Web Services from Patterns...`.
    - Choose `Simple Root Resource`. Click `Next`.
    - Give a class name (e.g., `StudentResource`).
    - Provide a package name (e.g., `com.example.student.rest`).
    - Set the `Path` (e.g., `/students`). Click `Finish`.
4. **Implement RESTful Methods:**
    - NetBeans will generate a basic resource class with `GET`, `PUT`, `POST`, `DELETE` stubs.
    - Modify the `StudentResource` class to manage a collection of student objects (e.g., using a `HashMap` for simplicity).
    - Implement methods for:
        - `GET /students`: Retrieve all students.
        - `GET /students/{id}`: Retrieve a specific student by ID.
        - `POST /students`: Add a new student.
        - `PUT /students/{id}`: Update an existing student.
        - `DELETE /students/{id}`: Delete a student.
    - Use JAX-RS annotations like `@Path`, `@GET`, `@POST`, `@PUT`, `@DELETE`, `@PathParam`, `@Produces`, `@Consumes`.
    - Define a simple `Student` POJO (Plain Old Java Object) class.
5. **Deploy and Test:**
    - Right-click on the `StudentRESTService` project and select `Clean and Build`.
    - Right-click again and select `Deploy`.
    - Test the service using a tool like Postman, Insomnia, or a web browser for `GET` requests.

Source Code:

`Student.java` (POJO - part of StudentRESTService project)

```
package com.example.student.model;

public class Student {
    private String id;
    private String name;
    private int age;
    private String major;

    public Student() {
    }

    public Student(String id, String name, int age, String major) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.major = major;
    }

    // Getters and Setters
    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

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

    public String getMajor() {
        return major;
    }

    public void setMajor(String major) {
        this.major = major;
    }

    @Override
    public String toString() {
        return "Student{" + "id=" + id + ", name=" + name + ", age=" + age +
", major=" + major + '}';
    }
}
```

`StudentResource.java` (RESTful Resource - part of StudentRESTService project)

```
package com.example.student.rest;

import com.example.student.model.Student;
import java.util.HashMap;
import java.util.Map;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.UriInfo;
import javax.ws.rs.Produces;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PUT;
import javax.ws.rs.PathParam;
import javax.ws.rs.POST;
import javax.ws.rs.DELETE;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;

/**
 * REST Web Service for managing student information.
 */
@Path("students") // Base path for this resource
public class StudentResource {

    @Context
    private UriInfo context;

    // In-memory storage for simplicity. In a real app, this would be a
database.
    private static Map<String, Student> students = new HashMap<>();

    static {
        students.put("S001", new Student("S001", "Alice Smith", 20, "Computer
Science"));
        students.put("S002", new Student("S002", "Bob Johnson", 21,
"Electrical Engineering"));
        students.put("S003", new Student("S003", "Charlie Brown", 19,
"Information Technology"));
    }

    /**
     * Creates a new instance of StudentResource
     */
    public StudentResource() {
    }

    /**
     * Retrieves representation of an instance of
com.example.student.rest.StudentResource
     * GET method to retrieve all students.
     * @return an instance of java.lang.String
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON) // Specifies that this method
produces JSON
    public Response getAllStudents() {
        return Response.ok(students.values()).build();
    }

    /**
     * GET method to retrieve a single student by ID.
     * @param id The ID of the student to retrieve.
     * @return a Response object containing the student or a 404 Not Found.
     */
    @GET
    @Path("{id}") // Path parameter for student ID
    @Produces(MediaType.APPLICATION_JSON)
    public Response getStudentById(@PathParam("id") String id) {
        Student student = students.get(id);
        if (student == null) {
            return Response.status(Status.NOT_FOUND).entity("Student with ID
" + id + " not found.").build();
        }
        return Response.ok(student).build();
    }
    /**
     * POST method to create a new student.
     * @param student The Student object to be created (sent in JSON format).
     * @return a Response object indicating success or failure.
     */
    @POST
    @Consumes(MediaType.APPLICATION_JSON) // Specifies that this method
consumes JSON
    @Produces(MediaType.APPLICATION_JSON)
    public Response createStudent(Student student) {
        if (students.containsKey(student.getId())) {
            return Response.status(Status.CONFLICT).entity("Student with ID "
+ student.getId() + " already exists.").build();
        }
        students.put(student.getId(), student);
        // Return 201 Created with the location of the new resource
        return Response.status(Status.CREATED)
                       .entity(student)

.location(context.getAbsolutePathBuilder().path(student.getId()).build())
                       .build();
    }

    /**
     * PUT method to update an existing student.
     * @param id The ID of the student to update.
     * @param updatedStudent The updated Student object (sent in JSON
format).
     * @return a Response object indicating success or failure.
     */
    @PUT
    @Path("{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response updateStudent(@PathParam("id") String id, Student
updatedStudent) {
        if (!students.containsKey(id)) {
            return Response.status(Status.NOT_FOUND).entity("Student with ID
" + id + " not found for update.").build();
        }
        if (!id.equals(updatedStudent.getId())) {
            return Response.status(Status.BAD_REQUEST).entity("ID in path
does not match ID in body.").build();
        }
        students.put(id, updatedStudent); // Overwrite existing student
        return Response.ok(updatedStudent).build();
    }

    /**
     * DELETE method to delete a student.
     * @param id The ID of the student to delete.
     * @return a Response object indicating success or failure.
     */
    @DELETE
    @Path("{id}")
    @Produces(MediaType.TEXT_PLAIN)
    public Response deleteStudent(@PathParam("id") String id) {
        Student removedStudent = students.remove(id);
        if (removedStudent == null) {
            return Response.status(Status.NOT_FOUND).entity("Student with ID
" + id + " not found for deletion.").build();
        }
        return Response.ok("Student with ID " + id + " deleted
successfully.").build();
    }
}
```

Input:

- **GET /students:** No input.
- **GET /students/{id}:** Path parameter `id` (e.g., `/students/S001`).
- **POST /students:** JSON body representing a new student (e.g., `{"id":"S004", "name":"Diana Prince", "age":22, "major":"Physics"}`).
- **PUT /students/{id}:** Path parameter `id` and JSON body representing the updated student (e.g., `/students/S001` with `{"id":"S001", "name":"Alice Wonderland", "age":21, "major":"Computer Science"}`).
- **DELETE /students/{id}:** Path parameter `id` (e.g., `/students/S002`).

Expected Output:

- **GET /students:** A JSON array of all student objects.
- `[`
- `    {"id":"S001","name":"Alice Smith","age":20,"major":"Computer Science"},`
- `    {"id":"S002","name":"Bob Johnson","age":21,"major":"Electrical Engineering"},`
- `    {"id":"S003","name":"Charlie Brown","age":19,"major":"Information Technology"}`
- `]`

- **GET /students/S001:** A JSON object for student S001.
- `{"id":"S001","name":"Alice Smith","age":20,"major":"Computer Science"}`

- **POST /students (with S004):** HTTP Status `201 Created` and the JSON of the newly created student.
- `{"id":"S004","name":"Diana Prince","age":22,"major":"Physics"}`

- **PUT /students/S001 (with updated name/age):** HTTP Status `200 OK` and the JSON of the updated student.
- `{"id":"S001","name":"Alice Wonderland","age":21,"major":"Computer Science"}`

- **DELETE /students/S002:** HTTP Status `200 OK` and a success message.
- `Student with ID S002 deleted successfully.`

- For invalid IDs or conflicting operations, appropriate HTTP error codes (e.g., `404 Not Found`, `409 Conflict`, `400 Bad Request`) and error messages.

Lab 10: Using JAXP SAX echo given xml file on console.

Title: Echoing XML File to Console using JAXP SAX Parser

Aim: To understand and implement the SAX (Simple API for XML) parser using JAXP (Java API for XML Processing) to parse an XML document and echo its content to the console. SAX is an event-driven parser, suitable for large XML files.

Procedure:

1. **Prepare an XML File:** Use a simple XML file (e.g., `students_xsd.xml` from Lab 2, or a smaller custom XML).
2. **Create a Java Project:** Create a new Java application project in your IDE (e.g., `SAXParserDemo`).
3. **Implement a Custom SAX Handler:**
   - Create a Java class that extends `org.xml.sax.helpers.DefaultHandler`.
   - Override methods like `startDocument()`, `endDocument()`, `startElement()`, `endElement()`, and `characters()`.
   - Inside these methods, print information about the XML events to the console (e.g., element names, attributes, character data).
4. **Implement the SAX Parser Logic:**
   - In your `main` method:
     - Create a `SAXParserFactory` instance.
     - Create a `SAXParser` from the factory.
     - Create an instance of your custom SAX handler.
     - Call the `parse()` method of the `SAXParser`, passing the XML file and your handler.
5. **Run the Application:** Execute the Java application.

Source Code:

`sample.xml` (You can use `students_xsd.xml` or this simple one)

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book category="cooking">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="children">
        <title lang="en">Harry Potter</title>
        <author>J.K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
</bookstore>
```

`MySAXHandler.java` (Custom SAX Handler)

```
package com.example.sax;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
```

```
public class MySAXHandler extends DefaultHandler {

    private StringBuilder currentValue = new StringBuilder();

    @Override
    public void startDocument() throws SAXException {
        System.out.println("--- SAX Parsing Started ---");
    }

    @Override
    public void endDocument() throws SAXException {
        System.out.println("--- SAX Parsing Finished ---");
    }

    @Override
    public void startElement(String uri, String localName, String qName,
Attributes attributes) throws SAXException {
        // Reset the current value for new element
        currentValue.setLength(0);
        System.out.print("START_ELEMENT: " + qName);
        if (attributes.getLength() > 0) {
            System.out.print(" (Attributes: ");
            for (int i = 0; i < attributes.getLength(); i++) {
                System.out.print(attributes.getQName(i) + "=" +
attributes.getValue(i) + (i < attributes.getLength() - 1 ? ", " : ""));
            }
            System.out.print(")");
        }
        System.out.println();
    }

    @Override
    public void endElement(String uri, String localName, String qName) throws
SAXException {
        System.out.println("END_ELEMENT: " + qName + " (Value: " +
currentValue.toString().trim() + ")");
    }

    @Override
    public void characters(char[] ch, int start, int length) throws
SAXException {
        // Append characters to the current value buffer
        currentValue.append(new String(ch, start, length));
    }

    @Override
    public void ignorableWhitespace(char[] ch, int start, int length) throws
SAXException {
        // Optionally handle ignorable whitespace
    }

    @Override
    public void warning(org.xml.sax.SAXParseException e) throws SAXException
{
        System.err.println("WARNING: " + e.getMessage());
    }

    @Override
    public void error(org.xml.sax.SAXParseException e) throws SAXException {
        System.err.println("ERROR: " + e.getMessage());
        throw e; // Re-throw to stop parsing on error
    }

    @Override
    public void fatalError(org.xml.sax.SAXParseException e) throws
SAXException {
        System.err.println("FATAL ERROR: " + e.getMessage());
        throw e; // Re-throw to stop parsing on fatal error
    }
}
```

`SAXEcho.java` (Main Class)

```
package com.example.sax;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import java.io.File;

public class SAXEcho {

    public static void main(String[] args) {
        String xmlFilePath = "sample.xml"; // Make sure this file is in the
project root or provide full path

        try {
            // 1. Create a SAXParserFactory
            SAXParserFactory factory = SAXParserFactory.newInstance();
            // Optional: Set namespace awareness
            factory.setNamespaceAware(true);
            // Optional: Set validation (if DTD/Schema is referenced)
            // factory.setValidating(true);

            // 2. Create a SAXParser
            SAXParser saxParser = factory.newSAXParser();

            // 3. Create an instance of your custom handler
            MySAXHandler handler = new MySAXHandler();

            // 4. Parse the XML file
            System.out.println("Parsing file: " + xmlFilePath);
            saxParser.parse(new File(xmlFilePath), handler);

        } catch (Exception e) {
            System.err.println("An error occurred during SAX parsing: " +
e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Input: The `sample.xml` file (or any valid XML file you specify in `xmlFilePath`).

Expected Output: The console should display a detailed log of SAX events as the parser processes the `sample.xml` file, showing the start and end of the document, elements, attributes, and character data.

Example (partial):

```
--- SAX Parsing Started ---
Parsing file: sample.xml
START_ELEMENT: bookstore
START_ELEMENT: book (Attributes: category=cooking)
START_ELEMENT: title (Attributes: lang=en)
END_ELEMENT: title (Value: Everyday Italian)
```

```
START_ELEMENT: author
END_ELEMENT: author (Value: Giada De Laurentiis)
START_ELEMENT: year
END_ELEMENT: year (Value: 2005)
START_ELEMENT: price
END_ELEMENT: price (Value: 30.00)
END_ELEMENT: book (Value: )
START_ELEMENT: book (Attributes: category=children)
START_ELEMENT: title (Attributes: lang=en)
END_ELEMENT: title (Value: Harry Potter)
START_ELEMENT: author
END_ELEMENT: author (Value: J.K. Rowling)
START_ELEMENT: year
END_ELEMENT: year (Value: 2005)
START_ELEMENT: price
END_ELEMENT: price (Value: 29.99)
END_ELEMENT: book (Value: )
END_ELEMENT: bookstore (Value: )
--- SAX Parsing Finished ---
```

Lab 11: Using JAXP DOM echo given xml file on console

Title: Echoing XML File to Console using JAXP DOM Parser

Aim: To understand and implement the DOM (Document Object Model) parser using JAXP (Java API for XML Processing) to parse an XML document and echo its content to the console. DOM creates an in-memory tree representation of the XML document, suitable for navigation and manipulation.

Procedure:

1. **Prepare an XML File:** Use the same `sample.xml` (or `students_xsd.xml`) file as in Lab 10.
2. **Create a Java Project:** Create a new Java application project in your IDE (e.g., `DOMParserDemo`).
3. **Implement the DOM Parser Logic:**
    - In your `main` method:
        - Create a `DocumentBuilderFactory` instance.
        - Create a `DocumentBuilder` from the factory.
        - Parse the XML file using `documentBuilder.parse()` to get a `Document` object (the DOM tree).
        - Implement a recursive helper method to traverse the DOM tree.
        - Inside the traversal method, print information about each `Node` (element name, attribute name/value, text content) to the console.
4. **Run the Application:** Execute the Java application.

Source Code:

`sample.xml` (Same as Lab 10)

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book category="cooking">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="children">
        <title lang="en">Harry Potter</title>
        <author>J.K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
</bookstore>
```

`DOMEcho.java` (Main Class)

```
package com.example.dom;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.io.File;

public class DOMEcho {

    public static void main(String[] args) {
        String xmlFilePath = "sample.xml"; // Make sure this file is in the
project root or provide full path

        try {
            // 1. Create a DocumentBuilderFactory
            DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
            // Optional: Set namespace awareness
            factory.setNamespaceAware(true);
            // Optional: Set validation (if DTD/Schema is referenced)
            // factory.setValidating(true);

            // 2. Create a DocumentBuilder
            DocumentBuilder builder = factory.newDocumentBuilder();

            // 3. Parse the XML file to get a Document object (DOM tree)
            System.out.println("Parsing file: " + xmlFilePath);
            Document document = builder.parse(new File(xmlFilePath));

            // Normalize the document (optional, but good practice)
            document.getDocumentElement().normalize();

            System.out.println("--- DOM Tree Content ---");
            // Start traversing the DOM tree from the root element
            printNode(document.getDocumentElement(), 0);

        } catch (Exception e) {
            System.err.println("An error occurred during DOM parsing: " +
e.getMessage());
            e.printStackTrace();
        }
    }

    /**
     * Recursively prints information about a DOM node and its children.
     * @param node The current node to print.
     * @param indentLevel The current indentation level for pretty printing.
     */
    private static void printNode(Node node, int indentLevel) {
        String indent = "  ".repeat(indentLevel); // For pretty printing

        switch (node.getNodeType()) {
            case Node.ELEMENT_NODE:
                Element element = (Element) node;
                System.out.println(indent + "Element: <" +
element.getTagName() + ">");

                // Print attributes
                NamedNodeMap attributes = element.getAttributes();
                if (attributes.getLength() > 0) {
                    System.out.println(indent + "  Attributes:");
                    for (int i = 0; i < attributes.getLength(); i++) {
                        Node attribute = attributes.item(i);
                        System.out.println(indent + "    " +
attribute.getNodeName() + " = \"" + attribute.getNodeValue() + "\"");
                    }
                }
                break;
            case Node.TEXT_NODE:
                String text = node.getNodeValue().trim();
                if (!text.isEmpty()) {
                    System.out.println(indent + "Text: \"" + text + "\"");
                }
                break;
            case Node.COMMENT_NODE:
                System.out.println(indent + "Comment: ");
                break;
            case Node.PROCESSING_INSTRUCTION_NODE:
                System.out.println(indent + "Processing Instruction: <?" + 
node.getNodeName() + " " + node.getNodeValue() + "?>");
                break;
            case Node.CDATA_SECTION_NODE:
                System.out.println(indent + "CDATA: <![CDATA[" + 
node.getNodeValue().trim() + "]]>");
                break;
            // Add other node types if needed (e.g., DOCUMENT_NODE, 
DOCUMENT_TYPE_NODE)
            default:
                // System.out.println(indent + "Node Type: " + 
node.getNodeType() + " Name: " + node.getNodeName());
                break;
        }

        // Recursively call for child nodes
        NodeList children = node.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
            printNode(children.item(i), indentLevel + 1);
        }
    }
}
```

Input: The `sample.xml` file (or any valid XML file you specify in `xmlFilePath`).

Expected Output: The console should display a hierarchical representation of the XML document's content, showing elements, their attributes, and text nodes, reflecting the in-memory DOM tree structure.

Example (partial):

```
Parsing file: sample.xml
--- DOM Tree Content ---
Element: <bookstore>
  Element: <book>
    Attributes:
      category = "cooking"
    Element: <title>
      Attributes:
        lang = "en"
      Text: "Everyday Italian"
    Element: <author>
      Text: "Giada De Laurentiis"
    Element: <year>
      Text: "2005"
    Element: <price>
      Text: "30.00"
  Element: <book>
    Attributes:
      category = "children"
    Element: <title>
      Attributes:
        lang = "en"
      Text: "Harry Potter"
    Element: <author>
      Text: "J.K. Rowling"
```

```
Element: <year>
  Text: "2005"
Element: <price>
  Text: "29.99"
```

Lab 12: Using AXIS 2 framework and TOMCAT create a simple calculator web service

Title: Creating a Calculator Web Service with Apache Axis2 and Tomcat

Aim: To learn how to develop and deploy a SOAP-based web service using the Apache Axis2 framework and Apache Tomcat server.

Procedure:

Prerequisites:

- Apache Tomcat installed and configured.
- Apache Axis2 WAR file (e.g., `axis2.war`) deployed to Tomcat's `webapps` directory.
- Apache Axis2 distribution downloaded (for client libraries and WSDL2Java tool).

Part A: Creating the Web Service (Service Side)

1. **Create a Java Project:** In your IDE (Eclipse/NetBeans), create a new Java project (e.g., `Axis2CalculatorService`).
2. **Add Axis2 Libraries:** Add the necessary Axis2 JARs to your project's build path. These are usually found in the `lib` directory of the Axis2 distribution.
3. **Develop the Service Class:**
   - Create a simple Java class (e.g., `CalculatorService`) with public methods for arithmetic operations.
   - No special annotations are required for basic POJO services in Axis2.
4. **Create services.xml:**
   - In your project, create a directory structure like `src/main/resources/META-INF`.
   - Inside `META-INF`, create a file named `services.xml`. This file describes your service to Axis2.
   - Define the service name, class name, and exposed operations.
5. **Build the AAR (Axis Archive) File:**
   - Package your service class and `services.xml` into an `.aar` file. This can be done manually by zipping the compiled class files and `META-INF` directory, or using Ant/Maven build scripts.
   - Example AAR structure:
   - `CalculatorService.aar`
   - `├── com/example/axis2/CalculatorService.class`
   - `└── META-INF/services.xml`

6. **Deploy the AAR to Tomcat/Axis2:**
   - Copy the generated `CalculatorService.aar` file into the `AXIS2_HOME/webapps/axis2/WEB-INF/services` directory (where `AXIS2_HOME` is your Tomcat directory).
   - Restart Tomcat.
7. **Verify Deployment:**
   - Open your browser and go to `http://localhost:8080/axis2/services/listServices`. You should see `CalculatorService` listed.
   - Click on `CalculatorService` to view its WSDL (e.g., `http://localhost:8080/axis2/services/CalculatorService?wsdl`).

Source Code:

CalculatorService.java

```java
package com.example.axis2;

public class CalculatorService {

    public int add(int i, int j) {
        System.out.println("CalculatorService: add(" + i + ", " + j + ")");
        return i + j;
    }

    public int subtract(int i, int j) {
        System.out.println("CalculatorService: subtract(" + i + ", " + j +
")");
        return i - j;
    }

    public int multiply(int i, int j) {
        System.out.println("CalculatorService: multiply(" + i + ", " + j +
")");
        return i * j;
    }

    public double divide(int i, int j) {
        System.out.println("CalculatorService: divide(" + i + ", " + j +
")");
        if (j == 0) {
            // In a real service, you'd throw a SOAP Fault
            System.err.println("Error: Division by zero.");
            return 0.0;
        }
        return (double) i / j;
    }
}
```

`services.xml` (located in `META-INF` directory within the AAR)

```xml
<service name="CalculatorService" scope="application">
    <description>
        A simple calculator web service using Axis2.
    </description>
    <messageReceivers>
        <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"

class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
        <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"

class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
    </messageReceivers>
    <parameter name="ServiceClass"
locked="false">com.example.axis2.CalculatorService</parameter>
    <operations>
        <operation name="add">
            <action>urn:add</action>
            <messageReceiver
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
        </operation>
        <operation name="subtract">
            <action>urn:subtract</action>
            <messageReceiver
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
        </operation>
        <operation name="multiply">
            <action>urn:multiply</action>
            <messageReceiver
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
        </operation>
        <operation name="divide">
            <action>urn:divide</action>
            <messageReceiver
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
        </operation>
    </operations>
</service>
```

Input: No direct input for service creation. The input is the Java code and `services.xml` file.

Expected Output:

- The `CalculatorService` should be successfully deployed on Tomcat under the Axis2 context.
- You should be able to access its WSDL by navigating to `http://localhost:8080/axis2/services/CalculatorService?wsdl` in your browser.
- The Tomcat console might show logs indicating successful deployment of the AAR file.

Lab 13: Using AXIS 2 framework and TOMCAT create a java client to consume calculator web service.

Title: Java Client Consuming Calculator Web Service with Apache Axis2

Aim: To learn how to create a Java client using the Apache Axis2 framework to consume the SOAP-based calculator web service deployed in Lab 12.

Procedure:

Prerequisites:

- The `CalculatorService` from Lab 12 is deployed and running on Tomcat/Axis2.
- Apache Axis2 distribution downloaded (for client libraries and WSDL2Java tool).

1. **Create a New Java Project:** In your IDE (Eclipse/NetBeans), create a new Java project (e.g., `Axis2CalculatorClient`).
2. **Add Axis2 Client Libraries:** Add the necessary Axis2 JARs (from the `lib` directory of your Axis2 distribution) to your project's build path.
3. **Generate Client Stubs using WSDL2Java:**
   - Open your command prompt/terminal.
   - Navigate to the `AXIS2_HOME/bin` directory.
   - Run the `wsdl2java` tool to generate client-side stubs from the service's WSDL.

   -
     ```
     wsdl2java -uri
     http://localhost:8080/axis2/services/CalculatorService?wsdl -o
     <output_directory> -p com.example.axis2.client
     ```

     (Replace `<output_directory>` with a path where you want the generated source files, and `com.example.axis2.client` with your desired package name).
   - Import the generated source files into your `Axis2CalculatorClient` project.
4. **Implement Client Logic:**
   - In your `main` method (or another class), instantiate the generated service stub/proxy.
   - Call the web service methods (e.g., `add`, `subtract`) using the stub object.
   - Print the results to the console.

Source Code:

(Assuming the `CalculatorService` from Lab 12 is running at `http://localhost:8080/axis2/services/CalculatorService?wsdl`)

`CalculatorClient.java` (Main Class - part of Axis2CalculatorClient project)

```
package com.example.axis2.client;

// Import the generated stub class (name might vary based on WSDL2Java
output)
import com.example.axis2.CalculatorServiceStub;

public class CalculatorClient {

    public static void main(String[] args) {
        try {
            // Create an instance of the generated stub
```

```
            // The constructor takes the service endpoint URL
            CalculatorServiceStub stub = new
CalculatorServiceStub("http://localhost:8080/axis2/services/CalculatorService
");

            // Create objects for the input parameters as defined by the
generated stub
            // For 'add' operation, wsdl2java typically creates a class like
'Add'
            CalculatorServiceStub.Add addRequest = new
CalculatorServiceStub.Add();
            addRequest.setI(50);
            addRequest.setJ(25);

            // Call the 'add' operation
            CalculatorServiceStub.AddResponse addResponse =
stub.add(addRequest);
            System.out.println("50 + 25 = " + addResponse.get_return());

            // Similarly for subtract
            CalculatorServiceStub.Subtract subtractRequest = new
CalculatorServiceStub.Subtract();
            subtractRequest.setI(50);
            subtractRequest.setJ(25);
            CalculatorServiceStub.SubtractResponse subtractResponse =
stub.subtract(subtractRequest);
            System.out.println("50 - 25 = " + subtractResponse.get_return());

            // For multiply
            CalculatorServiceStub.Multiply multiplyRequest = new
CalculatorServiceStub.Multiply();
            multiplyRequest.setI(10);
            multiplyRequest.setJ(5);
            CalculatorServiceStub.MultiplyResponse multiplyResponse =
stub.multiply(multiplyRequest);
            System.out.println("10 * 5 = " + multiplyResponse.get_return());

            // For divide
            CalculatorServiceStub.Divide divideRequest = new
CalculatorServiceStub.Divide();
            divideRequest.setI(100);
            divideRequest.setJ(3);
            CalculatorServiceStub.DivideResponse divideResponse =
stub.divide(divideRequest);
            System.out.println("100 / 3 = " + divideResponse.get_return());

            // Test division by zero
            CalculatorServiceStub.Divide divideByZeroRequest = new
CalculatorServiceStub.Divide();
            divideByZeroRequest.setI(100);
            divideByZeroRequest.setJ(0);
            CalculatorServiceStub.DivideResponse divideByZeroResponse =
stub.divide(divideByZeroRequest);
            System.out.println("100 / 0 = " +
divideByZeroResponse.get_return()); // Will be 0.0 as per service logic

        } catch (Exception ex) {
            System.out.println("An error occurred while consuming Axis2
service: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}
```

Input: For the client, the input values are hardcoded in the `main` method (e.g., `50`, `25`, `10`, `5`, `100`, `3`, `0`).

Expected Output: The console output from the Java client application should show the results of the arithmetic operations:

```
50 + 25 = 75
50 - 25 = 25
10 * 5 = 50
100 / 3 = 33.333333333333336
100 / 0 = 0.0
```

(The "Error: Division by zero." message will appear in the Tomcat server's console where the service is running, not the client's console.)

Lab 14: To create a web services in .NET

Title: Creating a Web Service in .NET (Repeat/Refinement of Lab 5 Part A)

Aim: To reinforce the process of developing a SOAP-based web service using C# in Visual Studio for the .NET platform. This lab serves as a focused exercise on service creation.

Procedure:

1. **Start Visual Studio:** Open Visual Studio.
2. **Create New Project:**
   - Go to `File > New > Project...`.
   - Select `ASP.NET Web Application (.NET Framework)`. Click `Next`.
   - Give a project name (e.g., `MyGenericWebService`). Click `Create`.
   - Choose `Empty` template and check `Web Forms`. Click `Create`.
3. **Add Web Service (ASMX):**
   - In `Solution Explorer`, right-click on the project.
   - Select `Add > New Item...`.
   - Search for `Web Service (ASMX)`. Give it a name (e.g., `HelloWorldService.asmx`). Click `Add`.
4. **Implement Service Methods:**
   - Open `HelloWorldService.asmx.cs`.
   - Add a simple method, for example, a `HelloWorld` method that returns a greeting.
   - Annotate the method with `[WebMethod]`.
   - Example:
   - `[WebMethod]`
   - `public string HelloWorld(string name)`
   - `{`
   - `    return "Hello, " + name + " from .NET Web Service!";`
   - `}`

5. **Build and Run:**
   - Build the solution (`Build > Build Solution`).
   - Run the project (`Debug > Start Without Debugging` or F5). This will deploy the service to IIS Express and open it in a browser.
6. **Verify WSDL:**
   - In the browser, you will see a page for `HelloWorldService.asmx`. Click on the service name to see the list of available operations and a link to the WSDL.

Source Code:

`HelloWorldService.asmx.cs`

```
using System.Web.Services;

namespace MyGenericWebService
{
    /// <summary>
    /// Summary description for HelloWorldService
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    // To allow this Web Service to be called from script, using ASP.NET
AJAX, uncomment the following line.
    // [System.Web.Script.Services.ScriptService]
    public class HelloWorldService : System.Web.Services.WebService
    {
        [WebMethod]
        public string HelloWorld(string name)
        {
            return "Hello, " + name + " from .NET Web Service!";
        }

        [WebMethod]
        public string GetServerTime()
        {
            return "Server time is: " + System.DateTime.Now.ToString();
        }
    }
}
```

Input: No direct input for service creation. The input is the C# code.

Expected Output:

- The `HelloWorldService` should be successfully deployed on IIS Express.
- You should be able to access its WSDL by navigating to `http://localhost:port/HelloWorldService.asmx?wsdl` in your browser.
- When you test the `HelloWorld` operation via the browser interface (by entering a name and clicking "Invoke"), you should see the XML response containing the greeting.

Example XML response for `HelloWorld("World")`:

```
<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://tempuri.org/">Hello, World from .NET Web
Service!</string>
```

Lab 15: INVOKING EJB COMPONENTS AS WEB SERVICES

Title: Invoking EJB Components as Web Services (JAX-WS over EJB)

Aim: To understand how Enterprise JavaBeans (EJBs) can be exposed as SOAP-based web services using JAX-WS (Java API for XML Web Services), leveraging the EJB container for deployment and management.

Procedure:

Prerequisites:

- Application Server supporting EJB and JAX-WS (e.g., GlassFish Server, WildFly, WebSphere, WebLogic).
- NetBeans or Eclipse IDE with server integration.

1. **Create an EJB Module Project:**
    - In NetBeans/Eclipse, create a new `Enterprise Application` project (e.g., `EJBWebServiceApp`).
    - Add an `EJB Module` to this enterprise application (e.g., `CalculatorEJBModule`).
2. **Develop the EJB Interface and Bean:**
    - **Remote Interface:** Create a remote interface for your EJB (e.g., `CalculatorRemote`) with methods for arithmetic operations. Annotate it with `@Remote`.
    - **Session Bean:** Create a stateless session bean (e.g., `CalculatorBean`) that implements the `CalculatorRemote` interface. Annotate it with `@Stateless`. Implement the arithmetic methods.
3. **Expose EJB as Web Service (JAX-WS):**
    - Annotate the EJB implementation class (`CalculatorBean`) with `@WebService`.
    - Optionally, use `@WebMethod` on specific methods if you want to control which methods are exposed. By default, all public methods are exposed.
    - You can also use `@SOAPBinding` to specify binding style (RPC/Document) and use/parameter style.
4. **Build and Deploy the Enterprise Application:**
    - Build the `EJBWebServiceApp` project.
    - Deploy the `EJBWebServiceApp` to your application server. The server will automatically generate the WSDL for the EJB-based web service.
5. **Verify WSDL:**
    - After deployment, the application server will provide a WSDL URL for your EJB web service. This URL typically follows a pattern like `http://localhost:8080/EJBWebServiceApp/CalculatorBean?wsdl` (the exact path depends on your server and deployment configuration).
    - Access this URL in a browser to confirm the WSDL is available.
6. **Create a Client (Optional, but good for testing):**
    - Create a separate Java application project (e.g., `EJBWebServiceClient`).
    - Add a web service client reference to this project using the WSDL URL of your EJB web service (similar to Lab 4 or Lab 6).
    - Implement client logic to invoke the EJB methods via the web service.

Source Code:

`CalculatorRemote.java` (Remote Interface - part of CalculatorEJBModule)

```
package com.example.ejb.calculator;
```

```
import javax.ejb.Remote;

@Remote
public interface CalculatorRemote {
    int add(int i, int j);
    int subtract(int i, int j);
    int multiply(int i, int j);
    double divide(int i, int j);
}
```

`CalculatorBean.java` (Stateless Session Bean - part of CalculatorEJBModule)

```
package com.example.ejb.calculator;

import javax.ejb.Stateless;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;
import javax.jws.soap.SOAPBinding.Use;

/**
 * Stateless Session Bean that also acts as a JAX-WS Web Service.
 */
@Stateless // Marks this as a Stateless Session EJB
@WebService(
    serviceName = "CalculatorService", // Name of the generated WSDL service
    portName = "CalculatorPort",       // Name of the generated WSDL port
    targetNamespace = "http://calculator.ejb.example.com/", // Target
namespace for WSDL
    endpointInterface = "com.example.ejb.calculator.CalculatorRemote" // Link
to the remote interface
)
@SOAPBinding(style = Style.RPC, use = Use.LITERAL) // Example: RPC style,
literal use
public class CalculatorBean implements CalculatorRemote {

    @Override
    @WebMethod // Expose this method as a web service operation
    public int add(int i, int j) {
        System.out.println("EJB Calculator: add(" + i + ", " + j + ")");
        return i + j;
    }

    @Override
    @WebMethod
    public int subtract(int i, int j) {
        System.out.println("EJB Calculator: subtract(" + i + ", " + j + ")");
        return i - j;
    }

    @Override
    @WebMethod
    public int multiply(int i, int j) {
        System.out.println("EJB Calculator: multiply(" + i + ", " + j + ")");
        return i * j;
    }

    @Override
    @WebMethod
    public double divide(int i, int j) {
        System.out.println("EJB Calculator: divide(" + i + ", " + j + ")");
        if (j == 0) {
            // In a real EJB, you might throw a custom EJB exception or a
SOAPFaultException
            System.err.println("Error: Division by zero in EJB.");
            return 0.0;
        }
        return (double) i / j;
    }
}
```

Input: No direct input for service creation. The input is the Java EJB code.

Expected Output:

- The `EJBWebServiceApp` should be successfully deployed on your application server.
- The `CalculatorBean` EJB should be available for remote invocation and also exposed as a web service.
- You should be able to access the WSDL for the EJB-based web service (e.g., `http://localhost:8080/EJBWebServiceApp/CalculatorBean?wsdl`).
- The application server's console will show deployment logs. If you create and run a client, the EJB's `System.out.println` messages will appear in the server logs.