Kandivli Education Society's

B. K. SHROFF COLLEGE OF ARTS & M. H. SHROFF COLLEGE OF COMMERCE

An Autonomous College | NAAC Re-accredited 'A' Grade

ISO 9001 : 2015 Certified • 'Best College 2017-18' award from University of Mumbai

JOURNAL

IN THE COURSE

ENTERPRISE JAVA

SUBMITTED BY

KRISH GABANI

S.Y.B.SC.IT

SDIT012A

SEMESTER IV

UNDER THE GUIDANCE OF

ASST. PROFESSOR BEENA KAPADIA

ACADEMIC YEAR

2023 - 2024

Kandivli Education Society's

B. K. SHROFF COLLEGE OF ARTS & M. H. SHROFF COLLEGE OF COMMERCE

An Autonomous College NAAC Re-accredited 'A' Grade

ISO 9001 : 2015 Certified • 'Best College 2017-18' award from University of Mumbai

CERTIFICATE

This is to certify that Mr. Krish Gabani of Second Year B.SC.IT, Div.: A, Roll No. 12 of Semester IV (2023 - 2024) has successfully completed the Journal for the course Enterprise Java as per the guidelines of KES' Shroff College of Arts and Commerce, Kandivali(W), Mumbai-400067.

Teacher In-charge **Principal**

Asst. Professor Beena Kapadia **Dr. L. Bhushan**

SDIT007B Enterprise Java

No.	Title/Aim	Date	Faculty Sign
1	Create a simple calculator application using servlet.		
2	Create a servlet application for a login page. If the username and password are correct, then it says message "Hello <username>" else a message "login failed".		
3	Create a servlet application that uses Cookies to store the number of times a user has visited servlet.		
4	Create a servlet application demonstrating the use of session creation and destruction. Also check whether the user has visited this page first time or has visited earlier also using sessions.		
5	Create a JSP application to demonstrate working of implicit object.		
6	Create a JSP application to demonstrate the use of Expression language.		
7	Create a JSP application to demonstrate the use of JSTL.		
8	Create a Currency Converter application using EJB.		
9	Develop simple EJB application to demonstrate Servlet Hit count using Singleton Session Beans.		
10	WAP in java to send the email using java mail API.		
11	Demonstrate the working of Soap base Calculator web service application.		

PRACTICAL 1:

Create a simple calculator application using servlet.

Describe the packages:

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

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

Describe the methods with its return type and the parameters, if any:

void doPost():
The doPost method is called by the server (via the service method) when the client requests a POST request. It is used to send information to the server.

void doGet(): The doGet method is called by the server (via the service method) when the client requests a GET request. It is used to retrieve information from the server.

void setContentType():

The Content-Type header is used to indicate the media type of the resource. The media type is a string sent along with the file indicating the format of the file.

PrintWriter getWriter():
It doesn't *make* a connection to the browser - the browser has already made a connection to the server. It either buffers what you write in memory, and then transmits the data at the end of the request, or it makes sure all the headers have been written to the network connection and then returns a PrintWriter which writes data directly to that network connection.

String getParameter():
The getParameter() method in getting data, especially form data, from a client HTML page to a JSP page is dealt with here. The request.getParameter() is being used here to retrieve form data from client side.

<u>CODE:</u>

<u>index.html:</u>

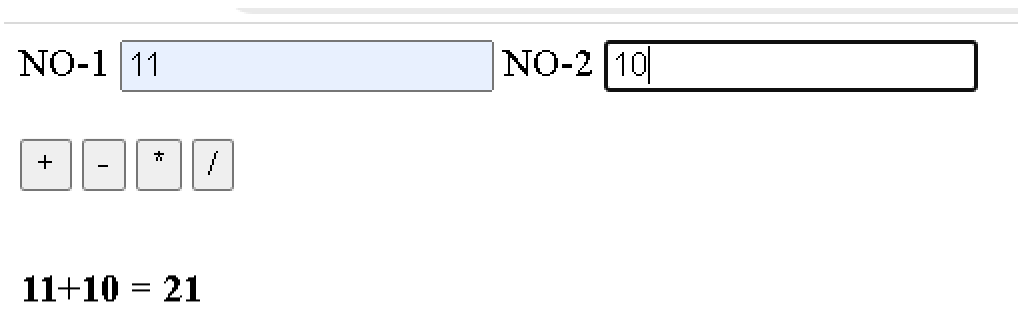
```
 <html>
<body>
<form method=post action="CalcServlet">
NUMBER-1 <input type=number  name="n1">
NUMBER-2 <input type=number  name="n2">  <br>
<input type=submit value="+" name="clk">
<input type=submit value="-" name="clk">
<input type=submit value="*" name="clk">
```

```
<input type=submit value="/" name="clk">
</form>
</body>
</html>
```

CalcServlet.java:

```
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
public class CalcServlet extends HttpServlet
{
        public void doPost(HttpServletRequest req, HttpServletResponse
res) throws  ServletException, IOException
       {
            res.setContentType("text/html");
          PrintWriter out = res.getWriter();
          int a=Integer.parseInt(req.getParameter("n1"));
          int b=Integer.parseInt(req.getParameter("n2"));
          int c=0;
          String op=req.getParameter("btn");
           if (operator.equals("+"))
          {
              c=a+b;

          }
          else if (operator.equals("-"))
          {
           c=a-b;
          }
           else if (operator.equals("*"))
          {
           c=a*b;
          }
           else if (operator.equals("/"))
          {
          c=a/b;
          }
           out.println("<b>"+a+ operator +b+" = "+c+"<b>");
        }
    }
```

OUTPUT:



```
NO-1 11 NO-2 10

11+10 = 21
```

PRACTICAL 2:

Create a servlet application for a login page. If the username and password are correct, then it says message "Hello <username>" else a message "login failed".

Describe following classes:

HttpServletRequest:
It is used to provide request information for HTTP servlets.

HttpServletResponse:
It is used to send the response information for HTTP servlets.

ServletException:
The ServletException is a generic exception which can be thrown by servlets encountering difficulties.

<u>CODE:</u>

<u>index.html:</u>

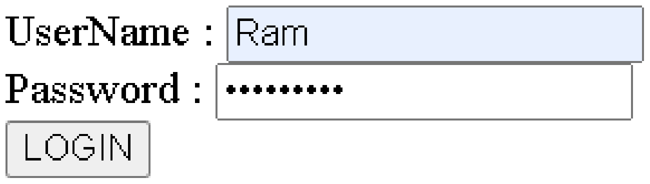
```
<html>
   <body>
```

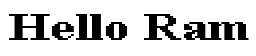
```
      <form action="LoginServlet" method="post">
         UserName : <input type="text" name="un"><br>
         Password : <input type="password" name="pass"> <br>
         <input type="submit" value="LOGIN">
       </form>
   </body>
</html>
```

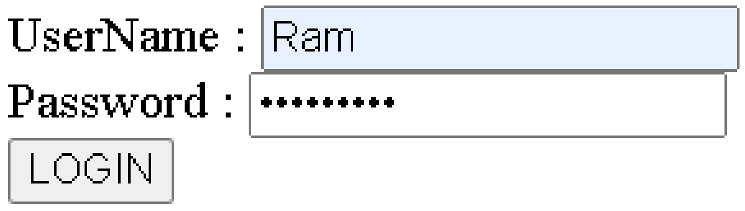
LoginServlet.java:

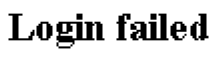
```
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
public class LoginServlet extends HttpServlet
{
   public void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
 {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
     String username=req.getParameter("un");
     String password=req.getParameter("pass");
     String msg="";
     if (username .equals("Ram") && password.equals("Robin@123"))
     {
         msg="Hello "+username;
     }
     else
     {
        RequestDispatcher rd=req.getRequestDispatcher(“index.html”);
         msg="Login failed";
         rd.include(req,res);
      }
      out.println("<b>"+msg+"<b>");
   }
}
```

OUTPUT:


UserName : Ram
Password : ••••••••••


Hello Ram


UserName : Ram
Password : ••••••••••


Login failed

PRACTICAL 3:

Create a servlet application that uses Cookies to store the number of times a user has visited servlet.

Describe the following constructor with overloaded parameters: Cookie:

A cookie is basically just a name-value attribute that is issued by the server, stored on a client machine and returned by the client whenever it is accessing a certain group of URLs on a specified server.

Describe the methods with its return type and the parameters, if any:
String valueOf():
The valueOf () method serves as a versatile utility in various programming contexts.

void addCookie():
The addCookie method is part of the HttpServletResponse class.When a servlet wants to send a cookie to the client (usually a web browser), it uses this method.The servlet sets appropriate HTTP headers in the response, and the browser stores the cookie on the user’s computer.

String getValue():
The getValue() method fetches the value of a tuple class object from the specified index.

CODE:

index.html:

```
<html>
   <body>
      <h4>
          <a href="CalcServlet"> CLICK HERE</a>
          VISIT TO SERVLET PAGE
          </h4>
   </body>
```

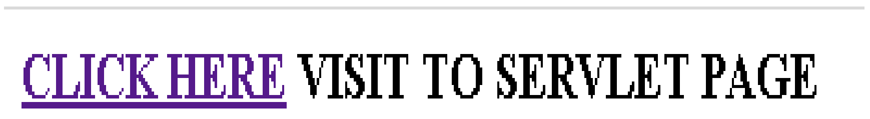
```
</html>
```

<u>CookieServlet.java:</u>

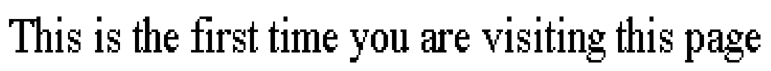
```
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
public class CookieServlet extends HttpServlet
{
   private int i=1;
   public void doGet(HttpServletRequest req,      HttpServletResponse
res)
       throws ServletException,IOException
   {
         res.setContentType("text/html");
         PrintWriter out = res.getWriter();
         String k=String.valueOf(i);
         Cookie ck= new Cookie("visit",k);
         res.addCookie(ck);
         int count=Integer.parseInt(ck.getValue());
         if(count==1)
         {
```

```
                    out.println("This is the first time you are visiting this page");
                }
              else
                {       synchronized(this)
                        {
                           out.println("You visited this page "+i+" times");
                        }
                }
             i++;
         }
}
```

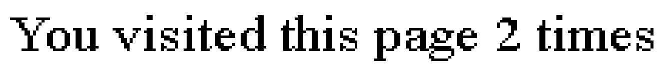
OUTPUT:



```
CLICK HERE VISIT TO SERVLET PAGE
```



```
This is the first time you are visiting this page
```



```
You visited this page 2 times
```

PRACTICAL 4:

Create a servlet application demonstrating the use of session creation and destruction. Also check whether the user has visited this page first time or has visited earlier also using sessions.

Session bean encapsulates business logic only, it can be invoked by local, remote and webservice client.

It can be used for calculations, database access etc.

The life cycle of session bean is maintained by the application server (EJB Container).

Types of Session Bean

There are 3 types of session bean.

1) Stateless Session Bean: It doesn't maintain state of a client between multiple method calls.

2) Stateful Session Bean: It maintains state of a client across multiple requests.

3) Singleton Session Bean: One instance per application, it is shared between clients and supports concurrent access.

<u>CODE:</u>

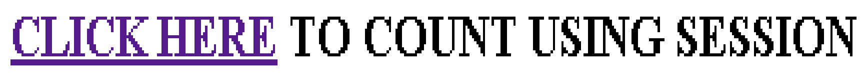
<u>index.html:</u>

```
<html>
    <body>
        <h4>
         <a href="CountVisitSrssionServlet">CLICK HERE</a>
            TO COUNT USING SESSION
        </h4>
    </body>
</html>
```

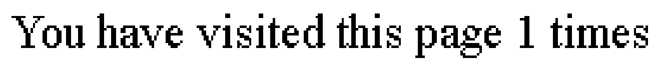
CountVisitSessionServlet.java:

```
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
public class CountVisitSessionServlet extends HttpServlet
{
    private int counter;
    public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
    {
       res.setContentType("text/html");
       PrintWriter out = res.getWriter();
       HttpSession session=req.getSession(true);
       if(session.isNew())
        {
           out.print("This is the first time you are visiting this page");
           ++counter;
        }
        else
        {
            synchronized(this)
               {
                   if(counter==10)
                         {
                             session.invalidate();
                             counter=0;
                             req.getSession(false);
                          }

                    else
                   {
                       out.print("You have visited this page "+(++counter)+ "
times");

                   }
                  }
         }
     }
}
```

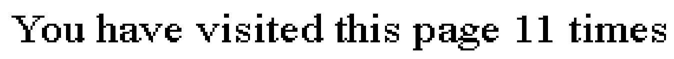
OUTPUT:



```
CLICK HERE TO COUNT USING SESSION
```



```
You have visited this page 1 times
```



```
You have visited this page 11 times
```

PRACTICAL 5:

Create a JSP application to demonstrate working of implicit object.

Describe the following methods with its return type and the parameters, if any, also write the implicit object name for those methods:

String getQueryString():
Returns the query string that is contained in the request URL after the path.

String getContextPath():
Return the portion of the request URI that indicates the context of the request.

String getRemoteHost():
Returns the fully qualified name of the client or the last proxy that sent the request.

String getCharacterEncoding():
Returns the name of the character encoding used in the body of this request.

String getContentType():
Returns the length, in bytes, of the request body and made available by the input stream , or -1 if the length is not known.

Locale getLocale():
Returns the preferred Locale that the client will accept content in, based on the Accept-Language header.

String getId():
Returns a string containing the unique identifier assigned to the session.

long getCreationTime():
Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

long getLastAccessedTime():
Returns the last time the client sent a request associated with this session.

<u>CODE:</u>

<u>index.html:</u>

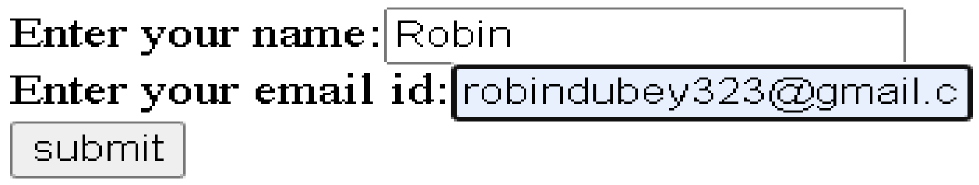
```
<html>
   <body>
   <center>
```

```
        <form action="ImplicitObjectEx.jsp">
           Enter your name    :<input type="text" name="myname"><br>
           Enter your email-id:<input type="text" name="mymailid"><br>
           <input type="submit" value="submit">
        </form>
    </center>
    </body>
</html>
```

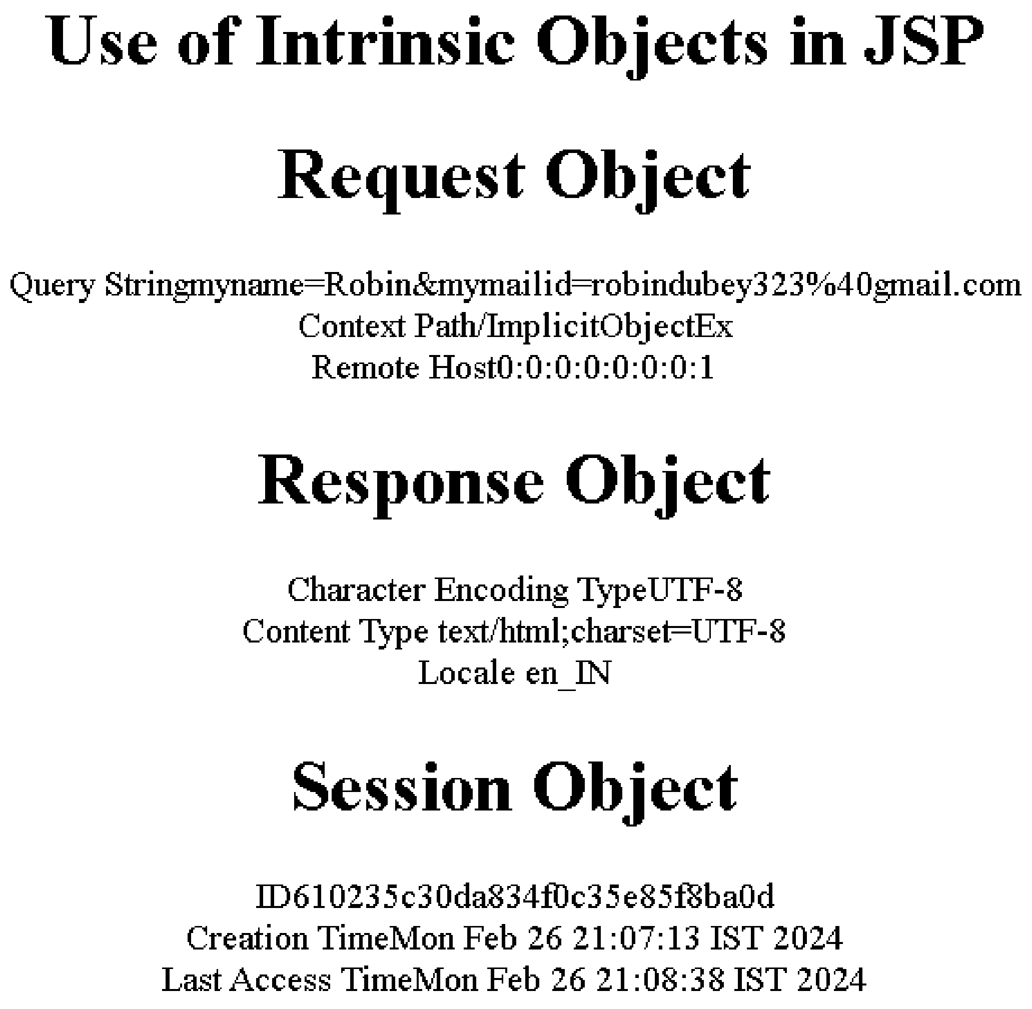
ImplicitObjectEx.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
<body>
    <center>
        <h1>Use of Intrinsic Objects in JSP</h1>
        <h1>Request Object</h1>
        Query String<%=request.getQueryString() %><br>
        Context Path<%=request.getContextPath() %><br>
        Remote Host<%=request.getRemoteHost() %><br>
        <h1>Response Object</h1>
        Character Encoding Type<%=response.getCharacterEncoding()
%><br>
        Content Type <%=response.getContentType() %><br>
        Locale <%=response.getLocale() %><br>
        <h1>Session Object</h1>
        ID<%=session.getId() %><br>
        Creation Time<%=new java.util.Date(session.getCreationTime())
%><br>
        Last Access Time<%=new
java.util.Date(session.getLastAccessedTime()) %><br>
    </center>
</body>
</html>
```

OUTPUT:



Enter your name: Robin
Enter your email id: robindubey323@gmail.c
submit



```
Use of Intrinsic Objects in JSP

Request Object

Query Stringmyname=Robin&mymailid=robindubey323%40gmail.com
Context Path/ImplicitObjectEx
Remote Host0:0:0:0:0:0:0:1

Response Object

Character Encoding TypeUTF-8
Content Type text/html;charset=UTF-8
Locale en_IN

Session Object

ID610235c30da834f0c35e85f8ba0d
Creation TimeMon Feb 26 21:07:13 IST 2024
Last Access TimeMon Feb 26 21:08:38 IST 2024
```

PRACTICAL 6:

Create a JSP application to demonstrate the use of Expression language.

Describe the syntax of Expression Language.
EL allows you to create expressions both (a) arithmetic and (b) logical. Within a JSP EL expression, you can use integers, floating point numbers, strings, the built-in constants true and false for boolean values, and null.

Syntax for Expression Language (EL):

```
$ {
   expression
   }
```

EL expressions can use parentheses to group sub expressions.

e.g., ${(1 + 2) * 3} equals 9, but ${1 + (2 * 3)} equals 7.

Empty operator in JSP Expression Language is used for prefix operation to check whether a value is empty or null.

```
${

  Empty “”

  }
```

<u>CODE:</u>

<u>index.html:</u>

```
<html>

   <body>

      <a href="Index.jsp">Click Here to Visit</a>Index Page<br>
```

```
        <a href="ExpressionLanguage.jsp">Click Here to
Visit</a>ExpressionLanguage<br>
        <a href="ELArithemeticOperator.jsp"> Click Here to
Visit</a>ELArithemeticOperator <br>
        <a href="ELLogicalOperator.jsp">Click Here to
Visit</a>ELLogicalOperator<br>
        <a href="ELRelationalOperator.jsp"> Click Here to
Visit</a>ELRelationalOperator<br>
        <a href="ELconditionalOperator.jsp"> Click Here to
Visit</a>ELconditionalOperator<br>
        <a href="EmptyOperator.jsp">Click Here to Visit</a>Empty
Operator<br>
    </body>
</html>
```

Index.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
    <body>
        <h3>welcome to Index Page</h3>
        <%
            session.setAttribute("user","Admin");
        %>
        <%
```

```
            Cookie ck=new Cookie("name","mycookie");
            response.addCookie(ck);
        %>
        <form action="ExpressionLanguage.jsp">
            Enter Name:<input type="text" name="name" /><br/><br/>
            <input type="submit" value="Submit"/>
        </form>
    </body>
</html>
```

ExpressionLanguage.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
    <body>
        <h1>Expression Language</h1>
        Welcome, ${ param.name }
        Session Value is ${ sessionScope.user }
        Cookie name is , ${cookie.name.value}
    </body>
</html>
```

ELArithemeticOperator.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
    <body>
        <h1>Arithmetic Operator</h1>
        5*5+4: ${5*5+4} <br>
        1.4E4+1.4: ${1.4E4+1.4}<br>
        10 mod 4: ${10 mod 4}<br>
        15 div 3: ${15 div 3}<br>
    </body>
</html>
```

ELLogicalOperator.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
    <body>
        <%-- LogicalOperator --%>
        <h2>Logical Operator</h2>
        true and true: ${true and true}<br>
        true && false: ${true && false}<br>
        true or true: ${true or true}<br>
        true || false: ${true || false}<br>
        not true: ${not true}<br>
        !false: ${!false}
    </body>
</html>
```

<u>ELRelationalOperator.jsp:</u>

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
    <body>
        <h2>Relational Operator</h2>
        10.0==10: ${10.0==10} <br>
        10.0 eq 10: ${10.0 eq 10} <br>
        ((20*10)!= 200): ${((20*10)!= 200)} <br>
        3 ne 3: ${3 ne 3} <br>
        3.2>=2: ${3.2>=2} <br>
        3.2 ge 2: ${3.2 ge 2} <br>
        2<3: ${2<3} <br>
        4 lt 6: ${4 lt 6} <br>
        2 <= 4: ${2 <= 4} <br>
        4 le 2: ${4 le 2}
    </body>
</html>
```

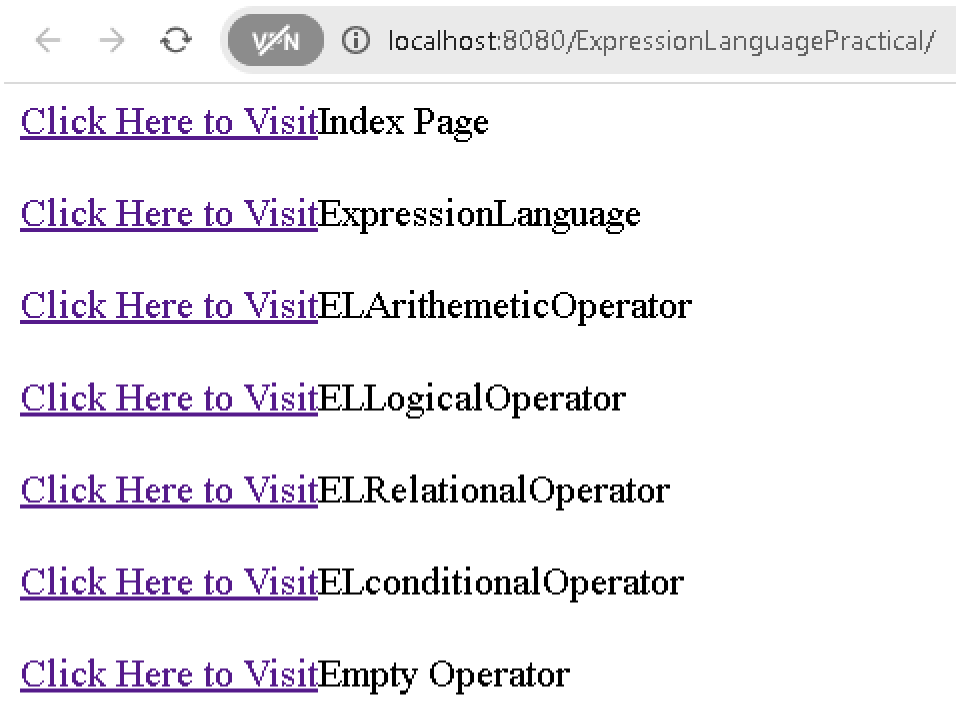
ELconditionalOperator.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
    <body>
        <h2>Conditional Operator</h2>
        The result of 10>2 is: "${(10>1)?'greater':'lesser'}"
    </body>
</html>
```

EmptyOperator.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
    <body>
        <h1>Empty Operator Example</h1>
        The Value for the Empty operator is:: ${empty "txxt"}
    </body>
</html>
```

OUTPUT:



localhost:8080/ExpressionLanguagePractical/

Click Here to VisitIndex Page

Click Here to VisitExpressionLanguage

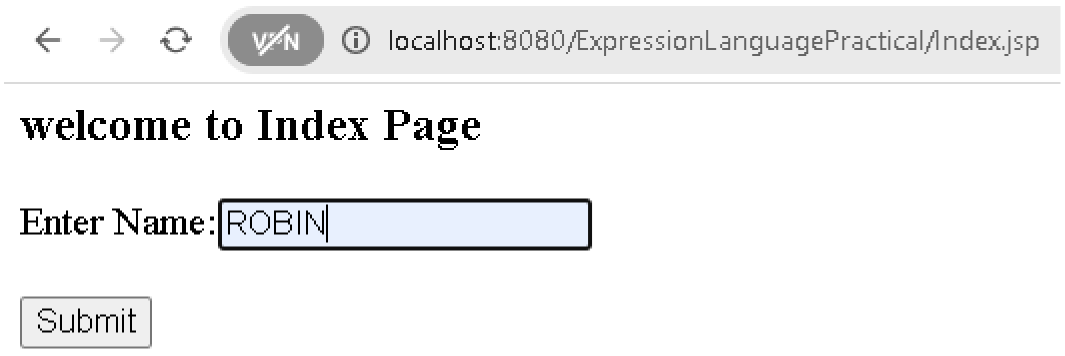
Click Here to VisitELArithemeticOperator

Click Here to VisitELLogicalOperator

Click Here to VisitELRelationalOperator

Click Here to VisitELconditionalOperator

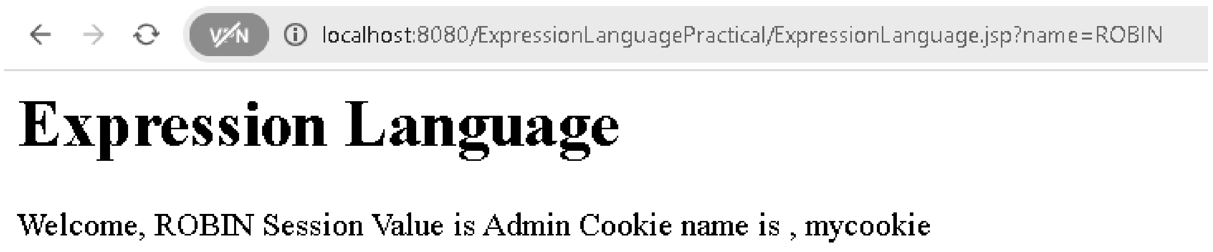
Click Here to VisitEmpty Operator



localhost:8080/ExpressionLanguagePractical/Index.jsp

welcome to Index Page

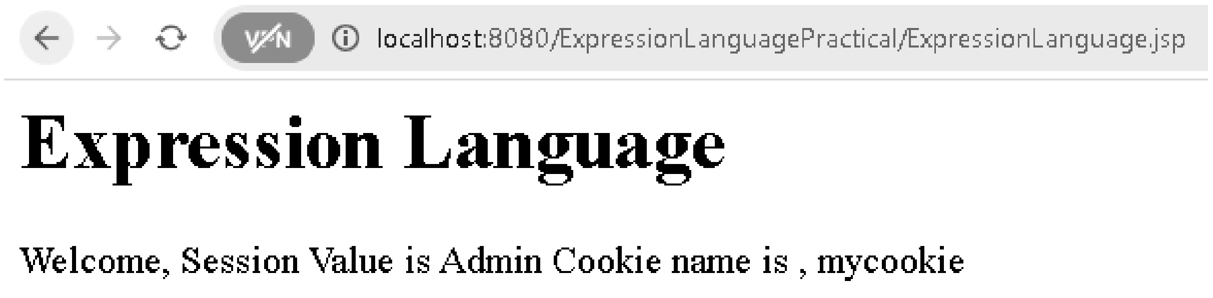
Enter Name: ROBIN



localhost:8080/ExpressionLanguagePractical/ExpressionLanguage.jsp?name=ROBIN

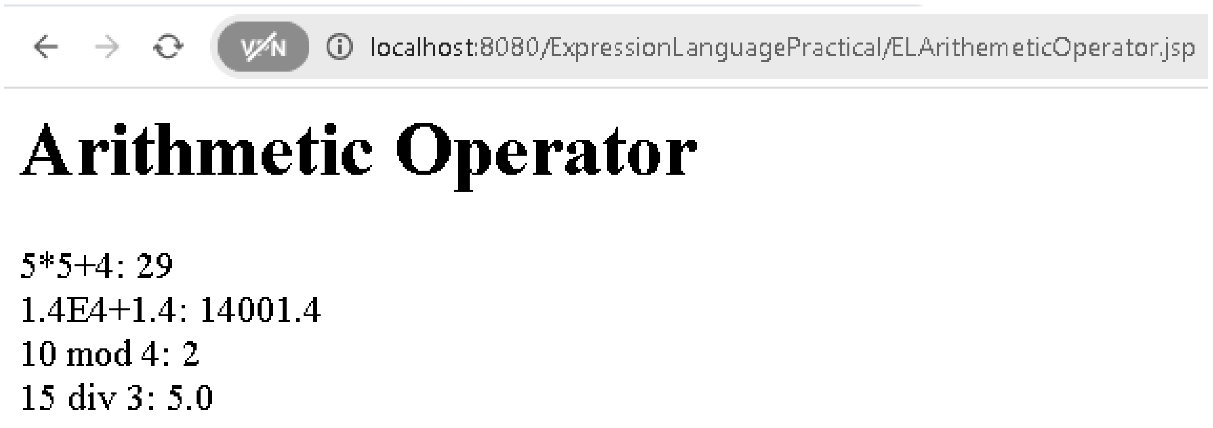
Expression Language

Welcome, ROBIN Session Value is Admin Cookie name is , mycookie



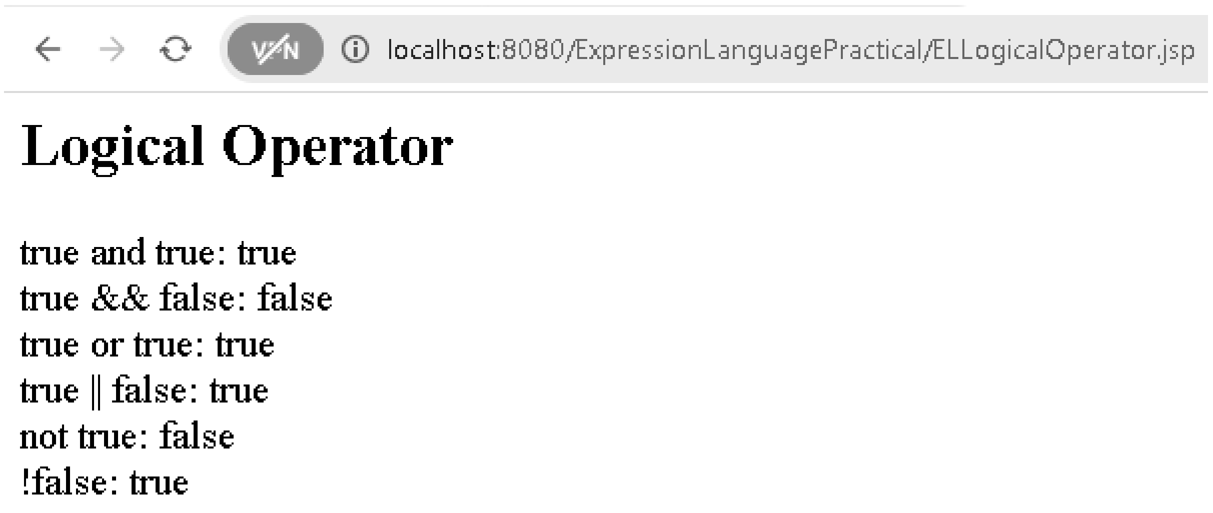
localhost:8080/ExpressionLanguagePractical/ExpressionLanguage.jsp

```
Expression Language

Welcome, Session Value is Admin Cookie name is , mycookie
```



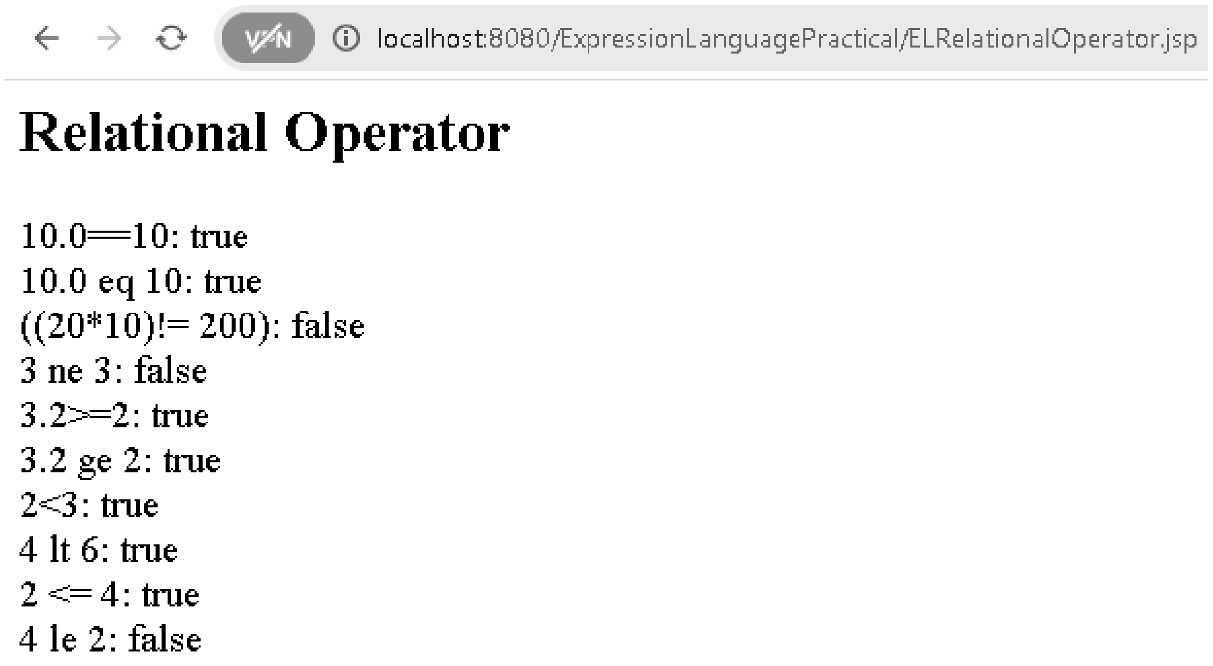
localhost:8080/ExpressionLanguagePractical/ELArithemeticOperator.jsp

```
Arithmetic Operator

5*5+4: 29
1.4E4+1.4: 14001.4
10 mod 4: 2
15 div 3: 5.0
```

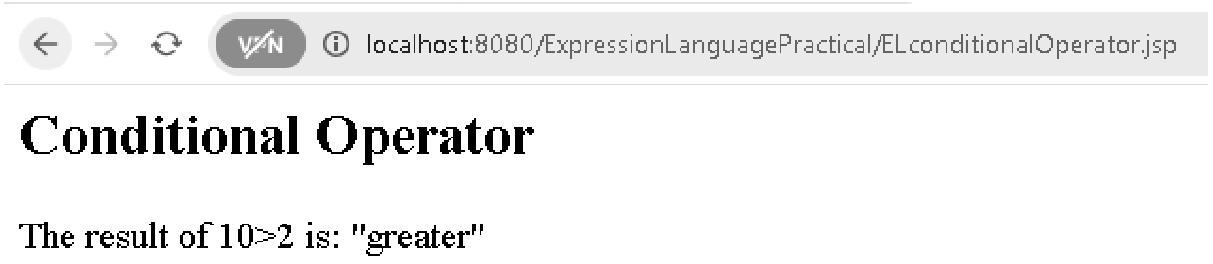


localhost:8080/ExpressionLanguagePractical/ELLogicalOperator.jsp

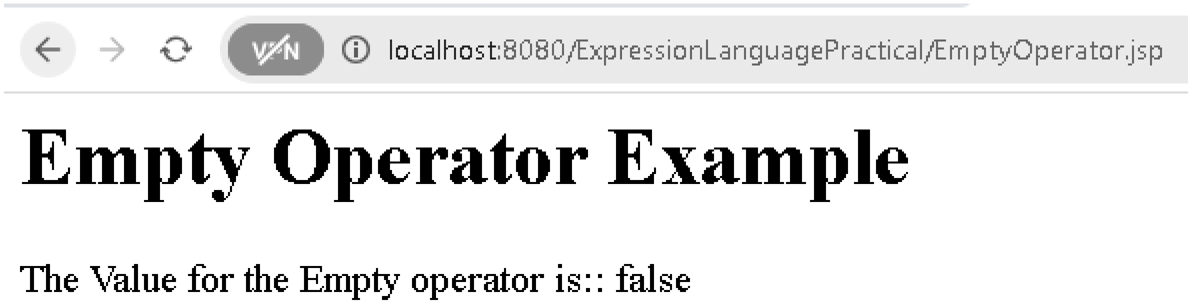
```
Logical Operator

true and true: true
true && false: false
true or true: true
true || false: true
not true: false
!false: true
```


localhost:8080/ExpressionLanguagePractical/ELRelationalOperator.jsp

```
Relational Operator

10.0==10: true
10.0 eq 10: true
((20*10)!= 200): false
3 ne 3: false
3.2>=2: true
3.2 ge 2: true
2<3: true
4 lt 6: true
2 <= 4: true
4 le 2: false
```


localhost:8080/ExpressionLanguagePractical/ELconditionalOperator.jsp

```
Conditional Operator

The result of 10>2 is: "greater"
```


localhost:8080/ExpressionLanguagePractical/EmptyOperator.jsp

```
Empty Operator Example

The Value for the Empty operator is:: false
```

PRACTICAL 7: Create a JSP application to demonstrate the use of JSTL.

Describe the syntax of if, for and choose..when…otherwise.
Syntax of if:

JSP if else loop is a conditional statement that enables a code snippet to be executed if a particular condition stands true. This conditional statement works similarly to the if else conditional statements in traditional coding languages like C/C++/Java and many more. This is a powerful impact and is commonly used in programming languages.

The syntax for JSP if-else loop is given below:

```
<%
if(condition)
{
Code or set of executable statements that should be processed if conditions stand true.
}
else if() //optional
{
Code or set of executable statements that should be processed if second condition is true while the first one stands false.
}
else //optional
{
This condition should be executed if all conditions stand false.
}
%>
```

Syntax of forEach:

JSP Foreach tag is used to iterate through the loop until a false condition is met. This is used in JSP in case we need to show a value from the array containing multiple values. For this tag to work, we should have an array of collections like List /Sets should be declared as a prerequisite as foreach requires an array as an input parameter to iterate through. It works like a "for" loop with a condition of "until we reach the end of array "in JAVA 5 or C. This is enabled by the use of tag <c:forEach> closed by closing tag </c:forEach>. This tag can be included in the JSP page with the help of the JSTL library. We need to make sure to include JSTL's latest library in the web content folder's "WEB-INF" Folder for this tag to work.

Syntax of Choose when Otherwise:
The < c:choose > tag is a conditional tag that establish a context for mutually exclusive conditional operations. It works like a Java switch statement in which we choose between a numbers of alternatives.

The <c:when > is subtag of <choose > that will include its body if the condition evaluated be 'true'.

The < c:otherwise > is also subtag of < choose > it follows &l;twhen > tags and runs only if all the prior condition evaluated is 'false'.

The c:when and c:otherwise works like if-else statement. But it must be placed inside <c:choose> tag.

CODE:

index.html:

```
<html>
   <body>
      <center>
      <a href="setDemo.jsp"> SetDemo</a><br>
      <a href="Maxif.html"> MaxIF</a><br>
      <a href="forEachDemo.jsp"> ForEachDemo</a><br>
      <a href="outDemo.jsp"> OutDemo</a><br>
      <a href="URLDemo.jsp"> URLDemo</a><br>
```

```
        <a href="choose_when_otherwise.jsp">
choose_when_otherwise</a><br>
        </center>
    </body>
</html>
```

setDemo.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<html>
    <body>
        <c:set var="pageTitle" scope="application" value="INDIAN
INSTITUTE OF TECHNOLAGY (IIT) BOMBAY:REGISTRATION"
/>
        ${
        pageTitle
        }
    </body>
</html>
```

Maxif.html:

```
<html>
    <body>
      <form action ="IFDemo.jsp">
            Number1:-<input type="number" name="x"  /><br>
            Number2:-<input type="number" name="y"  /><br>
            <input type="submit" value="Check Max" />
        </form>
    </body>
</html>
```

<u>IFDemo.jsp:</u>

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
    <body>
        <c:set var="x" value="${param.x}"/>
        <c:set var="y" value="${param.y}"/>
        <c:if test="${x>y}">
            <font color="blue">
            <h2>The Ans is:</h2>
            </font>
            <c:out value="${x} is greater than ${y}"/>
        </c:if>
    </body>
</html>
```

forEachDemo.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
    <body>
        <c:forEach begin="1" end="10" var="i">
            The Square of <c:out value=" ${i}=${i*i}"/><br>
        </c:forEach>
    </body>
</html>
```

outDemo.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
```

```
    <body>
        <c:set var="name" value="John"/>
        My name is: <c:out value= "${name}" />
    </body>
</html>
```

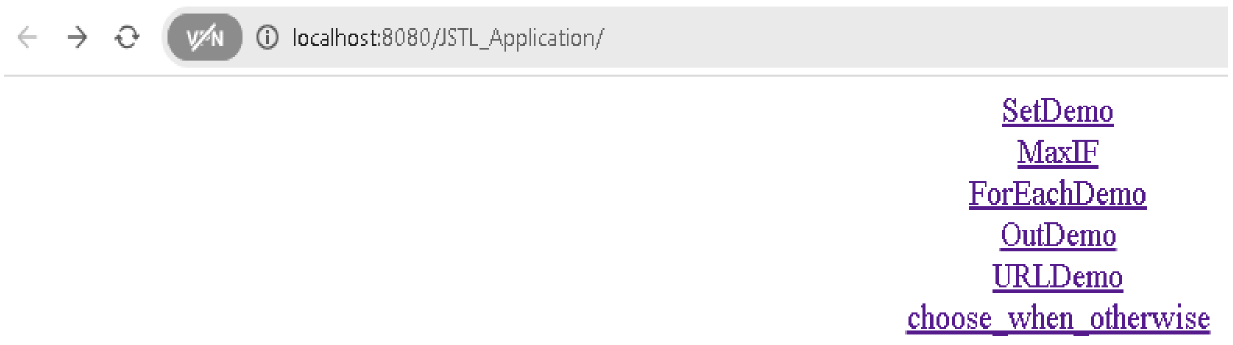
URLDemo.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
    <body>
        <c:url value="/index.html"/>
    </body>
</html>
```

choose_when_otherwise.jsp:

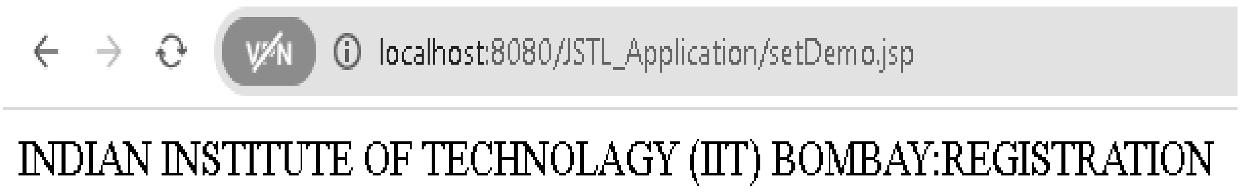
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
    <body>
        <c:set var="income" value="${4000*4}"/>
        Your Income is: <c:out value="${income}"/>
        <c:choose>
            <c:when test="${income <=1000}">
                Income is not good
            </c:when>
            <c:when test="${income > 10000}">
                Income is Very Good
            </c:when>
            <c:otherwise>
                Income is undetermined
            </c:otherwise>
        </c:choose>
    </body>
</html>
```

OUTPUT:

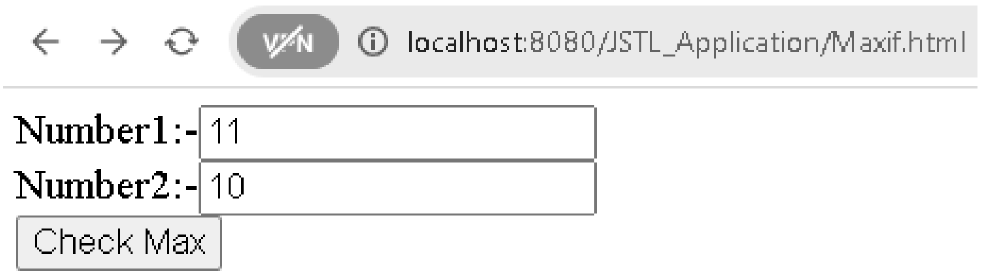


localhost:8080/JSTL_Application/

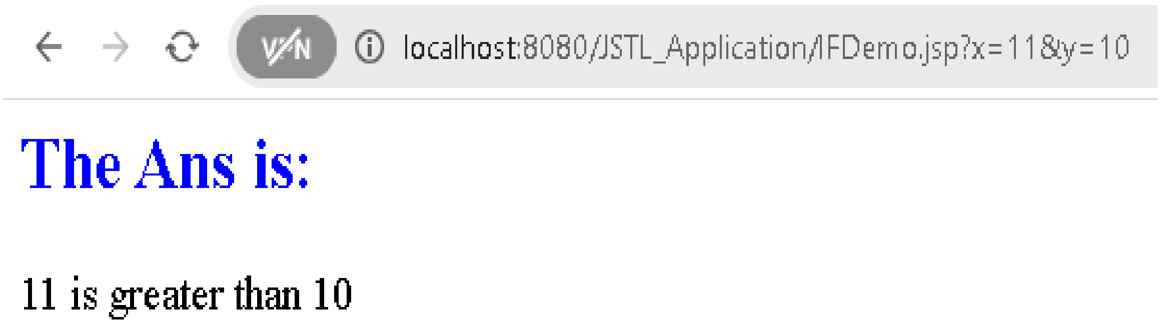
SetDemo
MaxIF
ForEachDemo
OutDemo
URLDemo
choose_when_otherwise


localhost:8080/JSTL_Application/setDemo.jsp

INDIAN INSTITUTE OF TECHNOLAGY (IIT) BOMBAY:REGISTRATION

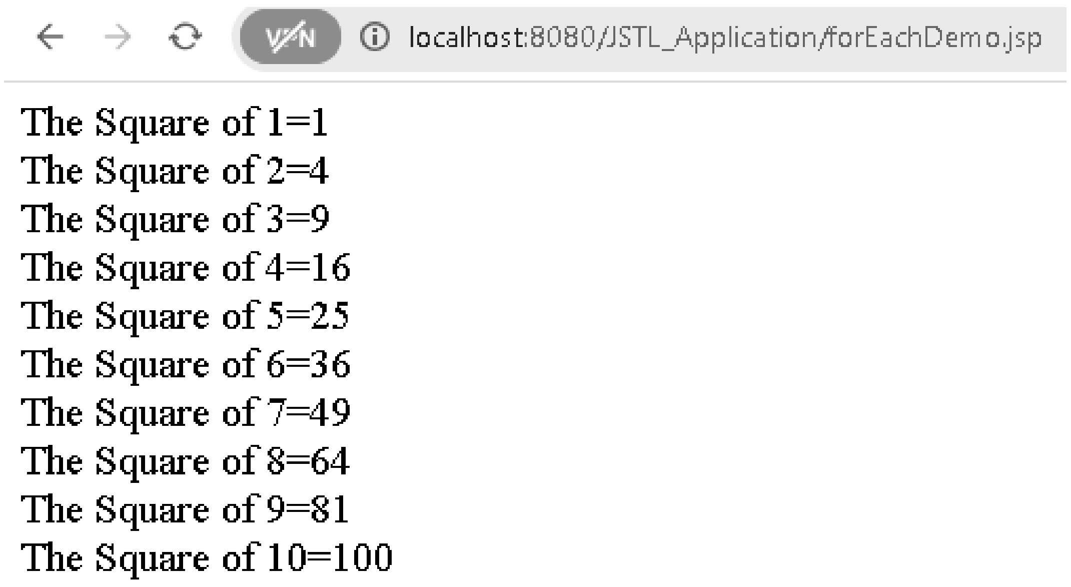

localhost:8080/JSTL_Application/Maxif.html

Number1:- 11
Number2:- 10

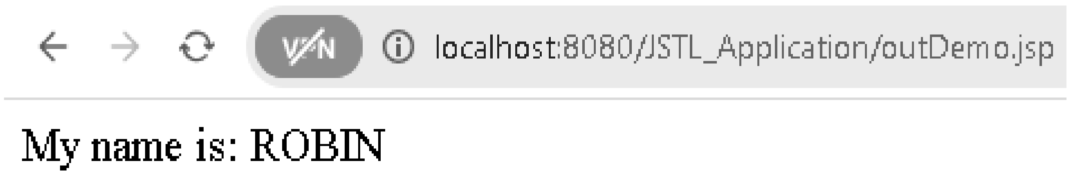

localhost:8080/JSTL_Application/IFDemo.jsp?x=11&y=10

The Ans is:

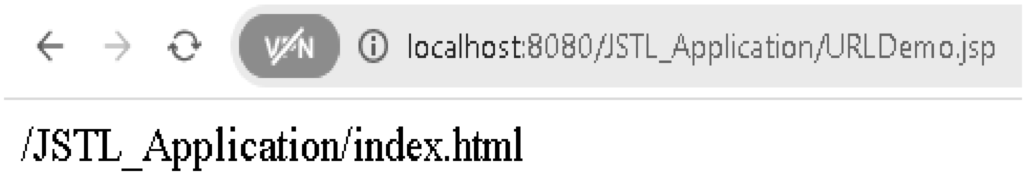
11 is greater than 10



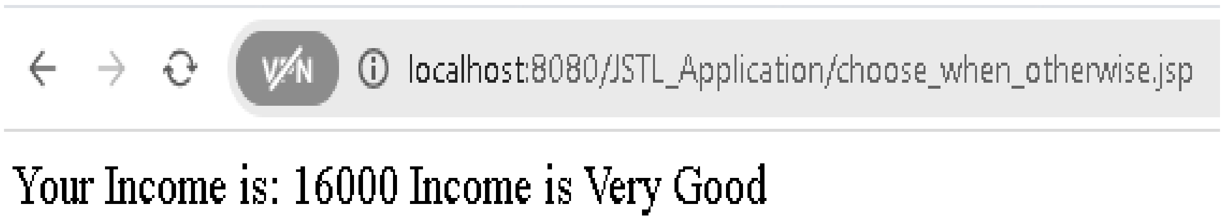
```
The Square of 1=1
The Square of 2=4
The Square of 3=9
The Square of 4=16
The Square of 5=25
The Square of 6=36
The Square of 7=49
The Square of 8=64
The Square of 9=81
The Square of 10=100
```



```
My name is: ROBIN
```



```
/JSTL_Application/index.html
```



```
Your Income is: 16000 Income is Very Good
```

PRACTICAL 8:

Create a Currency Converter application using EJB.

Describe the process of creating the object of session bean from the Servlet class.

The process of creating a session bean from a Servlet class. In a Java EE (Enterprise Edition) web application, session beans play a crucial role in managing stateful information across multiple requests.

Here are the steps involved:

Create a Java EE Web Application:

Begin by setting up a Java EE web application project. You can use tools like Eclipse, NetBeans, or IntelliJ IDEA to create the project structure.

Create a Session Bean:

1). A session bean represents a stateful component that maintains conversational state for a specific client. There are two types of session beans: Stateless and Stateful.

2). For this example, let's focus on creating a Stateful Session Bean.

3). Write a Java class that implements the session bean interface (usually javax.ejb.SessionBean or its derivatives).

4). Annotate the class with @Stateful to indicate that it's a stateful session bean.

4). Define business methods within the session bean. These methods will be accessible to clients.

Create a Servlet:

Servlets are Java classes that handle HTTP requests and responses.

Create a new Servlet class (e.g., MyServlet) within your web application project.

Implement the doGet or doPost method (depending on your use case).

Inside the servlet, you can obtain a reference to the session bean using dependency injection or by looking it up in the JNDI (Java Naming and Directory Interface) context.

Access the Session Bean from the Servlet:

In your servlet, inject the session bean using annotations (e.g., @EJB).

Alternatively, you can look up the session bean using JNDI.

Once you have a reference to the session bean, you can invoke its business methods.

CODE:

index.html:

```
<html>
    <body>
        <form action="CCServlet">
            Enter Amount <input type="number" name="amt"><br>
            Select Conversion Type
            <input type="radio" name="type" value="r2d" checked>Rupees
to Dollar
            <input type="radio" name="type" value="d2r" >Dollar to
Rupees<br>
```

```
            <input type="reset" ><input type="submit" value="CONVERT"
>
        </form>
    </body>
</html>
```

CCBean.java:

```
package pack;
import jakarta.ejb.Stateless;
@Stateless
public class CCBean implements CCBeanLocal
{
public double r2Dollar(double r)
{
    return r/75.65;
}
public double d2Rupees(double d)
{
    return d*75.65;
}
}
```

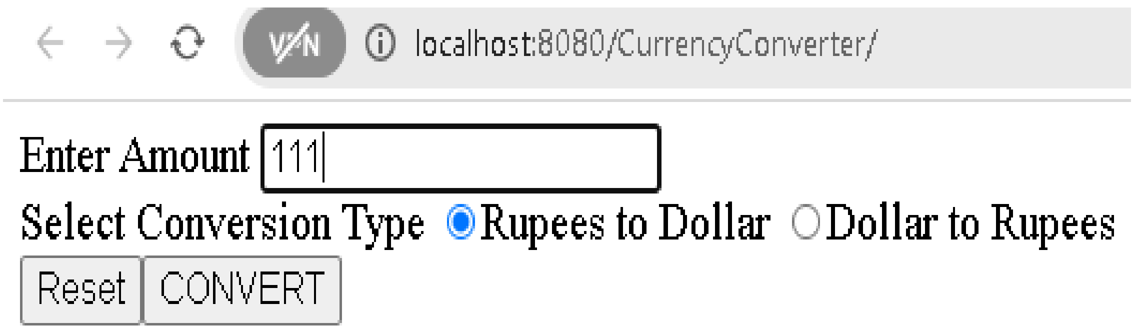
CCBeanLocal.java:

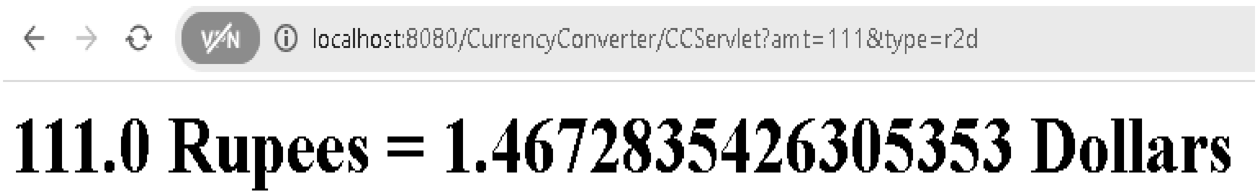
```
package pack;
import jakarta.ejb.Local;
@Local
public interface CCBeanLocal
{
   public double r2Dollar(double r);
   public double d2Rupees(double d);
}
```

CCServlet.java:

```
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.ejb.EJB;
import pack.CCBeanLocal;
public class CCServlet extends HttpServlet
{
   @EJB CCBeanLocal obj;
public void doGet(HttpServletRequest request,  HttpServletResponse
response)throws ServletException, IOException

{

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

double amt = Double.parseDouble(request.getParameter("amt"));


if(request.getParameter("type").equals("r2d"))

{

out.println("<h1>"+amt+ " Rupees = "+obj.r2Dollar(amt)+"
Dollars</h1>");

}


if(request.getParameter("type").equals("d2r"))


{

out.println("<h1>"+amt+ " Dollars = "+obj.d2Rupees(amt)+"
Rupees</h1>");


}

}

}
```

OUTPUT:


localhost:8080/CurrencyConverter/
Enter Amount 111
Select Conversion Type ● Rupees to Dollar ○ Dollar to Rupees
Reset
CONVERT



```
111.0 Rupees = 1.4672835426305353 Dollars
```

PRACTICAL 9:

Develop simple EJB application to demonstrate Servlet Hit count using Singleton Session Beans.

A Singleton Session Bean maintains the state of the bean for the complete lifecycle of the application.

Singleton Session Beans are similar to Stateless Session Beans but only one instance of the Singleton Session Bean is created in the whole application and does not terminates until the application is shut down.The single instance of the bean is shared between multiple clients and can be concurrently accessed.

CODE:

index.html:

```
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <meta http-equiv="Refresh" content="0; URL=ServletClient">
    </head>
    <body>
        <div>TODO write content</div>
    </body>
</html>
```

CountServletHitsBean.java:

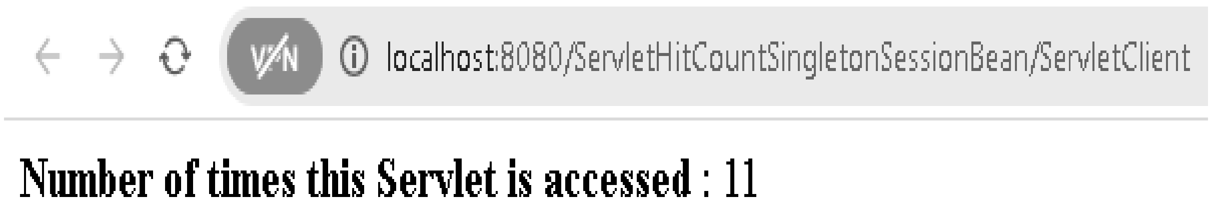
```
package pkg;
import jakarta.ejb.Singleton;
import jakarta.ejb.LocalBean;
```

```
@Singleton
@LocalBean
public class CountServletHitsBean
{
        private int hitCount;
        public synchronized int getCount()
        {
           return hitCount++;
        }
}
```

ServletClient.java:

```
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.ejb.EJB;
import jakarta.servlet.annotation.WebServlet;
import pkg.CountServletHitsBean;
@WebServlet(name = "ServletClient", urlPatterns = {"/ServletClient"})
public class ServletClient extends HttpServlet
{
    @EJB CountServletHitsBean obj;
    @Override
    public void service (HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        out.print("<b>Number of times this Servlet is accessed </b>:
"+obj.getCount());
}
}
```

OUTPUT:


localhost:8080/ServletHitCountSingletonSessionBean/ServletClient

```
Number of times this Servlet is accessed : 11
```

PRACTICAL 10:

WAP in java to send the email using java mail API.

The JavaMail is an API that is used to compose, write and read electronic messages (emails).The JavaMail API provides protocol-independent and plateform-independent framework for sending and receiving mails.The javax.mail and javax.mail.activation packages contains the core classes of JavaMail API.The JavaMail facility can be applied to many events. It can be used at the time of registering the user (sending notification such as thanks for your interest to my site), forgot password (sending password to the users email id), sending notifications for important updates etc. So there can be various usage of java mail api.

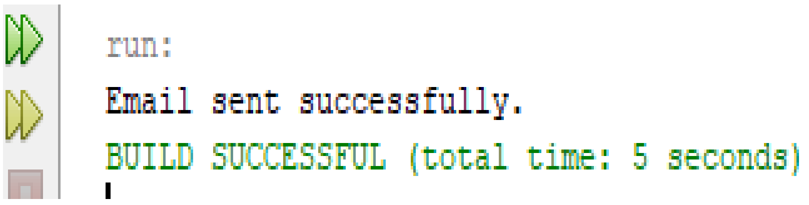
<u>CODE:</u>

<u>SendMail:</u>

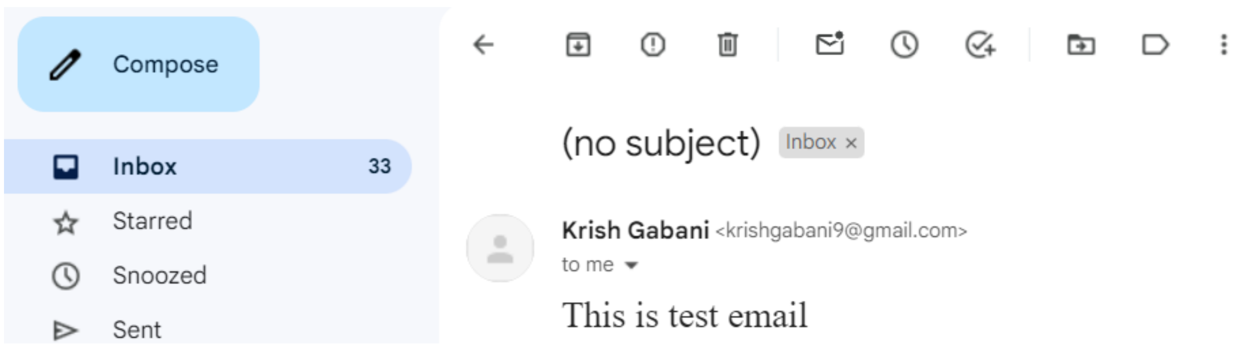
```
package sendmail;
import javax.mail.*;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import java.util.Properties;
public class SendMail
{
    public static void main(String[] args)
    {
      final String user="krishgabani9@gmail.com";
      final String password="agor rthm bhyt iycz";

      Properties props=new Properties();
      props.put("mail.smtp.auth","true");
      props.put("mail.smtp.starttls.enable","true");
      props.put("mail.smtp.host","smtp.gmail.com");
      props.put("mail.smtp.port","587");

      Session session=Session.getInstance(props,new Authenticator(){
         protected PasswordAuthentication getPasswordAuthentication(){
            return new PasswordAuthentication(user,password);
         }
      });
      try
     {
      Message message=new MimeMessage(session);
      //set the sender address
      message.setFrom(new
InternetAddress("krishgabani9@gmail.com"));
      //set the recipient address

message.setRecipients(Message.RecipientType.TO,InternetAddress.pars
e("krishgabani2@gmail.com"));
```

```
   message.setSubject("TEST EMAIL");
   message.setText("This is test email");
   Transport.send(message);

   System.out.println("Email Sent Successfully");
   }
   catch(MessagingException e)
   {
     e.printStackTrace();
   }
  }
}
```

OUTPUT:



```
run:
Email sent successfully.
BUILD SUCCESSFUL (total time: 5 seconds)
```



(no subject)

Krish Gabani <krishgabani9@gmail.com>

This is test email

PRACTICAL 11:

Demonstrate the working of Soap base Calculator web service application.

Steps to create web services:

New project -> java with Ant -> java web -> web application -> get New web Application Window -> provide the information in that window as: project name -> CalulatorApp, browse the path to save the project. Click on next -> finish.

Select and right click the application: CalculatorApp -> new -> webservice.

You will get new web service window. Provide information to that: web service name: CalcApp, package -> server -> finish.

Now right click CalcApp under web services -> Add operation -> you will get Add operation window -> in that provide the information as:

Name: add

Return type: int

Click on ADD

Name: a, type: int

Again Click on ADD

Name: b, type: int … click on OK

Now check that under the head CalcApp, you are getting operation as: add: int.

Now check that in CalcApp.java program, code will be there for add operation. The return statement in the program needs to be changed for add operation.

Replace return 0; with return a+b. Then right click CalculatorApp application and select deploy.

Then right click on CalcApp and click on test web service to get the following output.

Output: calculator web services: server side deployment

add Method invocation

____

Method parameter(s)

Type	Value
Int	43
Int	23

Method returned

int : "66"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:add xmlns:ns2="http://server/">
            <i>43</i>
            <j>23</j>
        </ns2:add>
    </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:addResponse xmlns:ns2="http://server/">
            <return>66</return>
        </ns2:addResponse>
    </S:Body>
</S:Envelope>
```

Code: calculator web services: client side

Now, create the client application as new project -> java With Ant -> Java Application -> next -> you will get new window as: new java application : project name -> CalcAppClient -> browse the folder to save -> next -> right click on CalcAppClient -> new -> web service client -> get new window as: new web service client : in that click on wsdl url and type **"http://localhost:8080/CalculatorApp/CalcApp?WSDL**" against that radio button (you can copy paste the tester path and instead of tester, write WSDL over there). Then, for package -> server -> finish.

From web service reference tree, drag and drop the method add in your class. The code will be copied in the class. Edit the code with a value and run the project.

```
package calcappclient;
public class Main {
    public static void main(String[] args)
```

```
 {
        // TODO code application logic here
        int sum=add(45,65);
        System.out.println("Addtion result is "+sum);
    }

    private static int add(int i, int j)
{
        server.CalcAppService service = new server.CalcAppService();
        server.CalcApp port = service.getCalcAppPort();
        return port.add(i, j);
    }

    }
```

Output: calculator web services: client side

```
wsimport-client-generate:
compile:
run:
Addtion result is 110
BUILD SUCCESSFUL (total time: 8 seconds)
```