GL BAJAJ INSTITUTE OF TECHNOLOGYAND MANAGEMENT

GREATER NOIDA

WEB -TECHNOLOGY LAB

PRACTICAL FILE

(KIT-551)

SESSION:2023-2024

SUBMITTED BY:

RAGINI AGRAWAL

IT-C

2101920130134

SUBMITTED TO:

Ms. JYOTI RAJ SINGH

(IT- DEPARTMENT)

INDEX

S. No.	Experiment Name	Date of Experiment	Date of Submission	Remark /Grade	Signature
1	Write HTML/Java scripts to display your CV in navigator, your Institute website, Department Website and Tutorial website for specific subject.				
2	Write an HTML program to design an entry form of student details and send it to store at database server like SQL, Oracle or MS Access.				
3	Write programs using Java script for Web Page to display browsers information.				
4	Write a Java applet to display the Application Program screen i.e. calculator and other.				
5	Writing program in XML for creation of DTD, which specifies set of rules. Create a style sheet in CSS/ XSL & display the document in internet				
6	Program to illustrate JDBC connectivity. Program for maintaining database by sending queries. Design and implement a simple servlet book query with the help of JDBC & SQL. Create MS Access Database, Create on				
7	Install TOMCAT web server and APACHE. Access the above developed static web pages for books web site, using these servers by putting the web				

S. No.	Experiment Name	Date of Experiment	Date of Submission	Remark /Grade	Signature
8	Assume four users user1, user2, user3 and user4 having the passwords pwd1, pwd2, pwd3 and pwd4 respectively. Write a servlet for doing the following. Create a Cookie and add these four user id's and passwords to this Cookie. 2. Read the user id and passwords entered in the Login form and				
9	Install a database (MySQL or Oracle). Create a table which should contain at least the following fields: name, password, email-id, phone number Write a java program/servlet/JSP to connect to that database and extract data from the tables and display them. Insert the details of the users who register with the web site, whenever a				
10	Write a JSP which insert the details of the 3 or 4 users who register with the web site by using registration form. Authenticate the user when he submits the login form using the user name and				

PROGRAM 1

Objective:

Write HTML/Java scripts to display your CV in navigator, your Institute website, Department Website and Tutorial website for specific subject.

Theory:

Introduction:

HTML stands for Hyper Text Markup Language. HTML is the computer language that is used to create documents for display on the Web. Many editors exist to create Web Pages – Word, Front Page, and Dream Weaver are just a few. Nevertheless, each of these software programs (editors) performs the exact same task – they all generate HTML language.

The HTML language consists of a series of HTML tags. Learning HTML involves finding out what tags are used to mark the parts of a document and how these tags are used in creating an HTML document.
Tags are instructions that tell your browser what to show on a Web page. They break up your document into basic sections. All tags start with a < (left bracket) and end with a > (right bracket).

Basic HTML Tags

<HTML> </HTML>

This tag tells your browser that the file contains HTML-coded information. All html tags must be placed between the open <HTML> tag and the closed tag </HTML> The file extension .html also indicates the document is an HTML document. All html documents MUST be saved with the .html file extension.

<HEAD> </HEAD>

The head tag identifies the first part of your HTML-coded document. The title tag (explained below) must be places between the open <HEAD> tag and the closed </HEAD> tag.

<BODY> </BODY>

The largest part of your HTML document is the body, which contains the content of your document (displayed within the text area of your browser window). All HTML tags that pertain to the body of your HTML document must be places between the open <BODY> tag and the closed </BODY> tag. The tag has attributes which you can use to set the colours of your background, text, links, and also to include your own background image.

They are as follows:

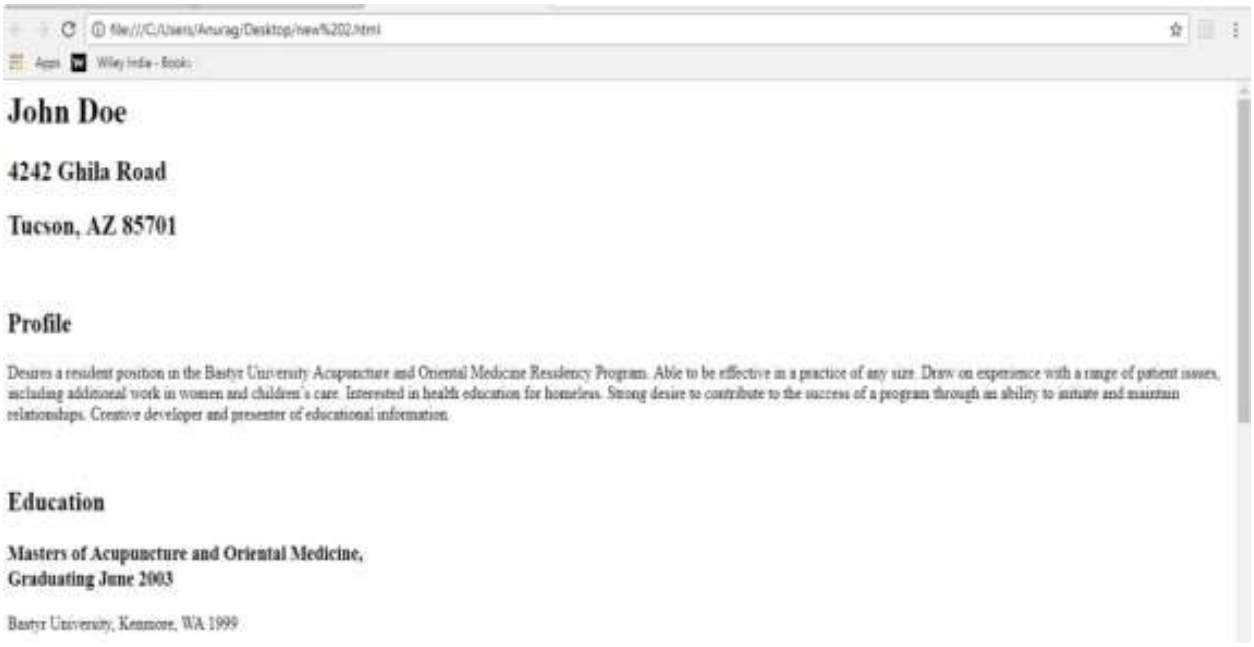
BGCOLOR="white" Sets the background color (other color names: red, black, blue etc)
TEXT="black" Sets the body text color
LINK="blue" Sets the unvisited hypertext links
VLINK ="purple" Sets the visited hypertext links

ALINK="red" Sets the active hypertext links (the color of the hypertext link when you have your mouse button depressed)

BACKGROUND Let you use an image as the background <background= Body attributes are used as part of the open <body> tag. For example:

<BODY BGCOLOR = "white" TEXT = "black" LINK = "blue" VLINK = "purple" ALINK = "red">

Output:



PROGRAM-2

Objective:

Write an HTML program to design an entry form of student details and send it to store at database server like SQL, Oracle or MS Access.

Theory:

HTML Forms

HTML Forms are required to collect different kinds of user inputs, such as contact details like name, email address, phone numbers, or details like credit card information, etc.

Forms contain special elements called controls like input box, checkboxes, radio-buttons, submit buttons, etc. Users generally complete a form by modifying its controls e.g. entering text, selecting items, etc. and submitting this form to a web server for processing.

The <form> tag is used to create an HTML form. Here's a simple example of a login form:

Input Element

This is the most commonly used element within HTML forms.It allows you to specify various types of user input fields, depending on the type attribute. An input element can be of type text field, checkbox, password field, radio button, submit button, reset button, etc. and several new input types introduced in HTML5.

The most used input types are described below.

Text Fields Text fields are one line areas that allow the user to input text.Single-line text input controls are created using an <input> element, whose type attribute has a value of text. Here's an example of a single-line text input used to take username:

Password Field

Password fields are similar to text fields. The only difference is; characters in a password field are masked i.e. shown as asterisks or dots. This is to prevent others from reading the password on the screen. This is also a single-line text input controls created using an <input> element whose type attribute has a value of password.
Here's an example of a single-line password input used to take user password:

Radio Buttons:

Radio buttons are used to let the user select exactly one option from a pre-defined set of options. It is created using an <input> element whose type attribute has a value of radio.

Checkboxes

Checkboxes allows the user to select one or more option from a pre-defined set of options. It is created using an <input> element whose type attribute has a value of checkbox.

File Select box

The file fields allow a user to browse for a local file and send it as an attachment to the form data. It normally rendered as a text box with a button that enables the user to browse for a file. However, the user can also type the path and name of the file in the text box.This is also created using an <input> element, whose type attribute value is set to file.

Textarea

Textarea is a multiple-line text input control that allows a user to enter more than one line of text. Multi-line text input controls are created using an <textarea> element.

Select Boxes

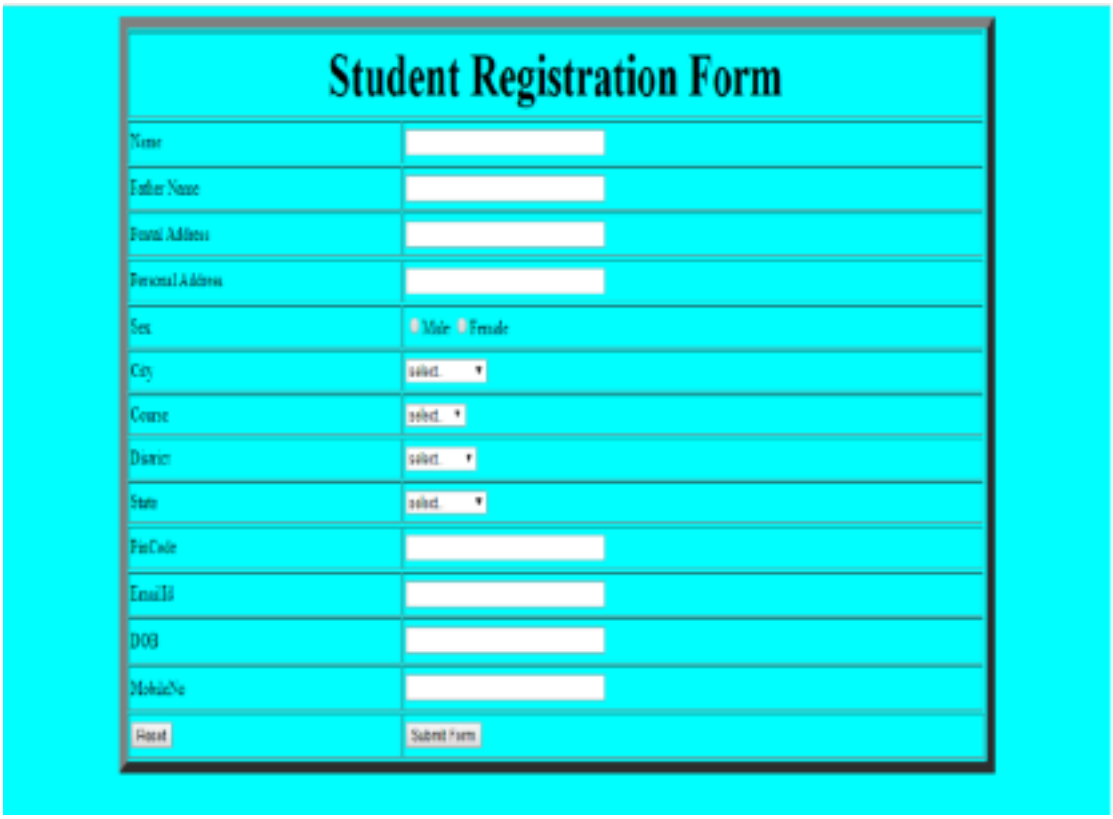
A select box is a drop down list of options that allows user to select one or more option from a pull-down list of options. Select box is created using the <select> element and <option> element. The option elements within the <select> element define each list item.
Submit and Reset Buttons

A submit button is used to send the form data to a web server. When submit button is clicked the form data is sent to the file specified in the form's action attribute to process the submitted data. A reset button resets all the forms control to default values.

Most frequently used form attributes are:

Attribute	Description
Name	The name of the form.
Action	URL of the program that processes the information submitted via form.
Method	The HTTP method that the browser uses to submit the form. Possible values are get and post.
Target	A name or keyword indicating the target page where the result of the script will be displayed.

Output:


Student Registration Form
Name
Father Name
Postal Address
Personal Address
Sex
Male
Female
City
select
Course
select
District
select
State
select
PinCode
EmailId
DOB
MobileNo
Reset
Submit Form

PROGRAM 3

Objective:

Write programs using Java script for Web Page to display browsers information.

Theory:

The navigator object contains information about the browser

Navigator Object Properties

Property	Description
appCodeName	Returns the code name of the browser.
appName	Returns the name of the browser.
appVersion	Returns the version information of the browser
cookieEnabled	Determines whether cookies are enabled in the browser.
geolocation	Returns a Geolocation object that can be used to locate the user's position.
language	Returns the language of the browser.
onLine	Determines whether the browser is online .
platform	Returns for which platform the browser is compiled.
product	Returns the engine name of the browser.
userAgent	Returns the user-agent header sent by the browser to the server.

OUTPUT :

for Mozilla browser:

Name Netscape
Version 5.0 (X11)
Codename Mozilla
Cookie enabletrue
Java Enablefunction javaEnabled() { [native code] }
Mime type[object MimeTypeArray] PlatformLinux i686
Plug ins[object PluginArray]
System Languageundefined
User languageMozilla/5.0 (X11; Ubuntu; Linux i686; rv:59.0) Gecko/20100101 Firefox/59.0
User AgentMozilla/5.0 (X11; Ubuntu; Linux i686; rv:59.0) Gecko/20100101 Firefox/59.0

OUTPUT :

For Chromium browser:

Name Netscape
Version 5.0 (X11; Linux i686) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/64.0.3282.167 Chrome/64.0.3282.167 Safari/537.36
Codename Mozilla
Cookie enabletrue
Java Enablefunction javaEnabled() { [native code] }
Mime type[object MimeTypeArray]
PlatformLinux i686
Plug ins[object PluginArray]
System Languageundefined
User languageMozilla/5.0 (X11; Linux i686) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/64.0.3282.167 Chrome/64.0.3282.167 Safari/537.36
User AgentMozilla/5.0 (X11; Linux i686) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/64.0.3282.167 Chrome/64.0.3282.167 Safari/537.36

PROGRAM 4

Objective:

Write a Java applet to display the Application Program screen i.e. calculator and other.

Theory

Applet is a special type of program that is embedded in the web page to generate the dynamic content. It runs inside the browser and works at client side.

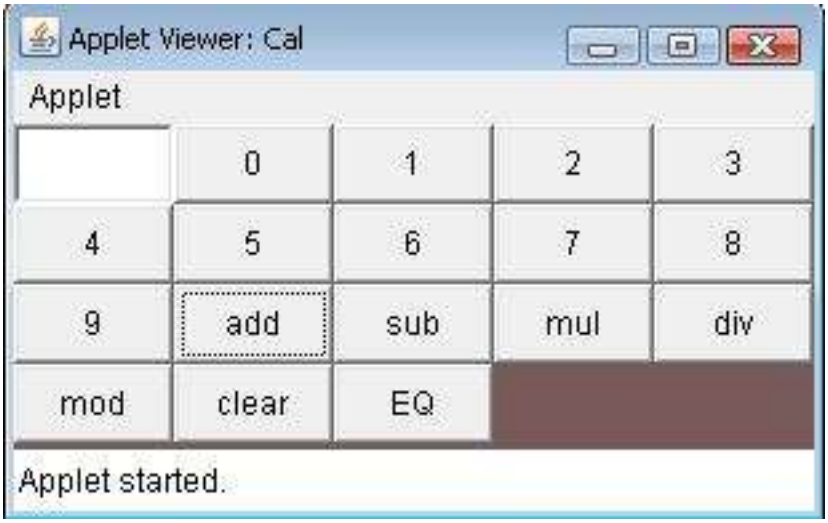
There are many advantages of applet. They are as follows

a)It works at client side so less response time

b)Secured

c)It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

OUTPUT:



PROGRAM 5

Objective:

Writing program in XML for creation of DTD, which specifies set of rules.

Create a style sheet in CSS/ XSL & display the document in internet explorer.

Theory:

XML Document Type Declaration, commonly known as DTD, is a way to describe precisely the XML language. DTDs check the validity of structure and vocabulary of an XML document against the grammatical rules of the appropriate XML language.

An XML document can be defined as:

- **Well-formed:** If the XML document adheres to all the general XML rules such as tags must be properly nested, opening and closing tags must be balanced, and empty tags must end with '/>', then it is called as well-formed.

OR

- **Valid:** An XML document said to be valid when it is not only well-formed, but it also conforms to available DTD that specifies which tags it uses, what attributes those tags can contain, and which tags can occur inside other tags, among other properties.

Types:

DTD can be classified on its declaration basis in the XML document, such as:

- Internal DTD
- External DTD

When a DTD is declared within the file it is called Internal DTD and if it is declared in a separate file it is called External DTD.
We will learn more about these in the chapter DTD Syntax.

Syntax

Basic syntax of a DTD is as follows:

```
<!DOCTYPE element DTD identifier

[

        declaration1

        declaration2

        ……..

]>
```

In the above syntax

- **DTD** starts with <!DOCTYPE> delimiter.
- An **element** tells the parser to parse the document from the specified root element.
- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **external subset**.
- **The square brackets [ ]** enclose an optional list of entity declarations called **internal subset**.

Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To reference it as internal DTD, standalone attribute in XML declaration must be set to yes. This means the declaration works independent of external source.

Syntax

The syntax of internal DTD is as shown:

<!DOCTYPE root-element [element-declarations]>

where *root-element* is the name of root element and element-declarations is where you declare the elements.

Example

Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
        <!ELEMENT address (name,company,phone)>
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT company (#PCDATA)>
        <!ELEMENT phone (#PCDATA)>
]>
<address>
        <name>Tanmay Patil</name>
        <company>TutorialsPoint</company>
        <phone>(011) 123-4567</phone>
</address>
```

External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL. To reference it as external DTD, standalone attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

Syntax

Following is the syntax for external DTD:

```
<!DOCTYPE root-element SYSTEM "file-name">
```

Where *file-name* is the file with *.dtd* extension.

Example

The following example shows external DTD usage.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
                <name>Tanmay Patil</name>
                <company>TutorialsPoint</company>
                 <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** are as shown:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

1. XSLT

EXtensible Stylesheet Language Transformation commonly known as XSLT is a way to transform the XML document into other formats such as XHTML.

<xsl:template> defines a way to reuse templates in order to generate the desired output for nodes of a particular type/context.various template are used with the template like name, match ,mode, priority.

<xsl:value-of> tag puts the value of the selected node as per XPath expression, as text.

Declaration:
Following is the syntax declaration of **<xsl:value-of>** element.

```
<xsl:value-of
  select = Expression
  disable-output-escaping = "yes" | "no" >
</xsl:value-of>
```

<xsl:for-each> tag applies a template repeatedly for each node.

Declaration
Following is the syntax declaration of <xsl:for-each> element

```
<xsl:for-each
  select = Expression >
</xsl:for-each>
```

<xsl:sort> tag specifies a sort criteria on the nodes.

Declaration
Following is the syntax declaration of <xsl:sort> element.

```
<xsl:sort
  select = string-expression
  lang = { nmtoken }
```

```
  data-type = { "text" | "number" | QName }
  order = { "ascending" | "descending" }
  case-order = { "upper-first" | "lower-first" } >
</xsl:sort>
```

<xsl:if> tag specifies a conditional test against the content of nodes.

Declaration
Following is the syntax declaration of <xsl:if> element.

```
<xsl:if
  test = boolean-expression >
</xsl:if>
```

<xsl:choose> tag specifies a multiple conditional tests against the content of nodes in conjunction with the <xsl:otherwise> and <xsl:when> elements.

Declaration
Following is the syntax declaration of <xsl:choose> element.

```
<xsl:choose >
</xsl:choose>
```

Output:

In Internet Explorer

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler

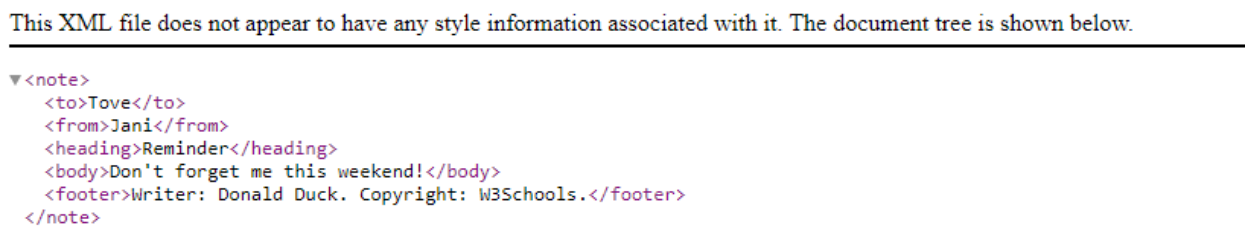
2. XML

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM “Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>

DTD:

<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)><!ELEMENT heading (#PCDATA)> <!ELEMENT body (#PCDATA)>
]>

OUTPUT:



PROGRAM 6

Objective:

Program to illustrate JDBC connectivity. Program for maintaining database by sending queries. Design and implement a simple servlet book query with the help of JDBC & SQL. Create MS Access Database, create on ODBC link, Compile & execute JAVA JDVC Socket.

Theory:

Creating JDBC Application

There are following six steps involved in building a JDBC application –

•Import the packages: Requires that you include the packages containing the JDBC classes needed for database programming. Most often, using import java.sql.* will suffice.

•Register the JDBC driver: Requires that you initialize a driver so you can open a communication channel with the database.

•Open a connection: Requires using the DriverManager.getConnection() method to create a Connection object, which represents a physical connection with the database.

•Execute a query: Requires using an object of type Statement for building and submitting an SQL statement to the database.

•Extract data from result set: Requires that you use the appropriate ResultSet.getXXX() method to retrieve the data from the result set.

•Clean up the environment: Requires explicitly closing all database resources versus relying on the JVM's garbage collection.

STEPS:

```
//STEP 1. Import required packages
import java.sql.*;
public class FirstExample {
        // JDBC driver name and database URL
        static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
        static final String DB_URL = "jdbc:mysql://localhost/EMP";
        // Database credentials
        static final String USER = "username";
        static final String PASS = "password";
        public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        try{
        //STEP 2: Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");
```

```
        //STEP 3: Open a connection
        System.out.println("Connecting to database…");
        conn = DriverManager.getConnection(DB_URL,USER,PASS);
        //STEP 4: Execute a query
        System.out.println("Creating statement…");
        stmt = conn.createStatement();
        String sql;
        sql = "SELECT id, first, last, age FROM Employees";
        ResultSet rs = stmt.executeQuery(sql);
        //STEP 5: Extract data from result set
        while(rs.next()){
                //Retrieve by column name
                int id = rs.getInt(“id");
                int age = rs.getInt(“age");
                String first = rs.getString("first");
                String last = rs.getString("last");
                //Display values
                System.out.print("ID: " + id);
                System.out.print(", Age: " + age);
                System.out.print(", First: " + first);
                System.out.println(", Last: " + last);
        }
        //STEP 6: Clean-up environment
        rs.close();
        stmt.close();
        conn.close();
        }catch(SQLException se){
         //Handle errors for JDBC se.printStackTrace();
        }catch(Exception e){
        //Handle errors for Class.forName e.printStackTrace();
        }finally{
        //finally block used to close resources
        try{
                if(stmt!=null)
                stmt.close();
        }catch(SQLException se2){
        }// nothing we can do
        try{
        if(conn!=null)
        conn.close();
        }catch(SQLException se){
        se.printStackTrace(); }
        //end finally try
        }//end try
        System.out.println("Goodbye!"); }//end main
}//end FirstExample
```

PROGRAM 7

Objective:

Install TOMCAT web server and APACHE. Access the above developed static web pages for books web site, using these servers by putting the web pages developed .

Theory:

Set the JAVA_HOME Variable

You must set the JAVA_HOME environment variable to tell Tomcat where to find Java. Failing to properly set this variable prevents Tomcat from handling JSP pages. This variable should list the base JDK installation directory, not the bin subdirectory.

On Windows XP, you could also go to the Start menu, select Control Panel, chooseSystem, click on the Advanced tab, press the Environment Variables button at the bottom, and enter the JAVA_HOME variable and value directly as:

Name: JAVA_HOME Value: C:\jdk

Set the CLASSPATH

Since servlets and JSP are not part of the Java 2 platform, standard edition, you have to identify the servlet classes to the compiler. The server already knows about the servlet classes, but the compiler (i.e., javac) you use for development probably doesn't.So, if you don't set your CLASSPATH, attempts to compile servlets, tag libraries, or other classes that use the servlet and JSP APIs will fail with error messages about unknown classes.

Name: JAVA_HOME
Value: install_dir/common/lib/servlet-api.jar

Turn on Servlet Reloading

The next step is to tell Tomcat to check the modification dates of the class files of requested servlets and reload ones that have changed since they were loaded into the server's memory. This slightly degrades performance in deployment situations, so is turned off by default. However, if you fail to turn it on for your development server, you'll have to restart the server every time you recompile a servlet that has already been loaded into the server's memory.

To turn on servlet reloading, edit install_dir/conf/server.xml and add a DefaultContext subelement to the main Host element and supply true for the reloadable attribute. For example, in Tomcat 5.0.27, search for this entry:

<Host name="localhost" debug="0" appBase="webapps" ...> and then insert the following immediately below it:

<DefaultContext reloadable="true"/>

Be sure to make a backup copy of server.xm before making the above change.

Enable the Invoker Servlet

The invoker servlet lets you run servlets without first making changes to your Web application's deployment descriptor. Instead, you just drop your servlet into WEB-INF/classes and use the URL http://host/servlet/ServletName . The invoker servlet is extremely convenient when you are learning and even when you are doing your initial development.

To enable the invoker servlet, uncomment the following servlet and servlet-mapping elements in install_dir/conf/web.xml. Finally, remember to make a backup copy of the original version of this file before you make the changes.

```
<servlet>
        <servlet-name>invoker</servlet-name>
        <servlet-class>
        org.apache.catalina.servlets.InvokerServlet
        </servlet-class>
        ...
</servlet>
...
<servlet-mapping>
        <servlet-name>invoker</servlet-name>
        <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>
```

PROGRAM 8

Objective:

Assume four users user1, user2, user3 and user4 having the passwords pwd1, pwd2, pwd3 and pwd4 respectively. Write a servlet for doing the following.
1. Create a Cookie and add these four user id's and passwords to this Cookie.
2. Read the user id and passwords entered in the Login form (week1) and authenticate with the values (user id and passwords) available in the cookies.

Theory:

Servlet Life cycle:

1. Servlet class loading
2. Servlet Instantiation
3. call the init method
4. call the service method
5. call destroy method

- **Class loading and instantiation**
  If you consider a servlet to be just like any other Java program, except that it runs within a servlet container, there has to be a process of loading the class and making it ready for requests. Servlets do not have the exact equivalent of a main method that causes them to start execution. When a web container starts it searches for the deployment descriptor (WEB.XML) for each of its web applications. When it finds a servlet in the descriptor it will create an instance of the servlet class. At this point the class is considered to be loaded (but not initialized).

- **The init method**
  The HttpServlet class inherits the init method from GenericServlet. The init method performs a role slightly similar to a constructor in an "ordinary" Java program in that it allows initialization of an instance at start up. It is called automatically by the servlet container and as it causes the application context (WEB.XML) to be parsed and any initialization will be performed. It comes in two versions, one with a zero parameter constructor and one that takes a ServletConfig parameter.
  The servlet engine creates a request object and a response object. The servlet engine invokes the servlet service() method, passing the request and response objects. Once the init method returns the servlet is said to be placed into service. The process of using init to initialize servlets means that it is possible to change configuration details by modifying the deployment descriptor without having them hard coded in with your Java source and needing a re-

compilation.
void init(ServletConfig sc)

Calling the service method

The service() method gets information about the request from the request object, processes the request, and uses methods of the response object to create the client response. The service method can invoke other methods to process the request, such as doGet(), doPost(), or methods you write. The service method is called for each request processed and is not normally overridden by the programmer.

The code that makes a servlet “go” is the. servlet void service(ServletRequest req,ServletResponse res)

- **The destroy Method**
  Two typical reasons for the destroy method being called are if the container is shutting down or if the container is low on resources. This can happen when the container keeps a pool of instances of servlets to ensure adequate performance. If no requests have come in for a particular servlet for a while it may destroy it to ensure resources are available for the servlets that are being requested. The destroy method is called only once, before a servlet is unloaded and thus you cannot be certain when and if it is called.
  void destroy()

ServletConfig Class

ServletConfig object is used by the Servlet Container to pass information to the Servlet during it's initialization. Servlet can obtain information regarding initialization parameters and their values using different methods of ServletConfig class initialization parameters are name/value pairs used to provide basic information to the Servlet during it's initialization like JDBC driver name, path to database, username, password etc.

Methods of ServletConfig class

Following are the four methods of this class :

1. getInitParameter(String paramName) Returns value of the given parameter. If value of parameter could not be found in web.xml file then a null value is returned.

2. GetInitParameterNames() Returns an Enumeration object containing all the names of initialization parameters provided for this Servlet.

3. GetServletContext() Returns reference to the ServletContext object for this Servlet. It is similar to getServletContext() method provided by HttpServlet class.

4. GetServletName() Returns name of the Servlet as provided in the web.xml file or if none is provided then returns complete class path to the Servlet.

Code

A servlet that demonstrates creating a cookie with user IDs and passwords and authenticating the login credentials against the values stored in the cookie.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class LoginServlet extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      // Create a new cookie
      Cookie cookie = new Cookie("userCredentials",
"user1:pwd1,user2:pwd2,user3:pwd3,user4:pwd4");
      cookie.setMaxAge(3600); // Set cookie's max age (1 hour)
      response.addCookie(cookie);
      // Read user input from login form
      String userId = request.getParameter("userId");
      String password = request.getParameter("password");
      // Retrieve the cookie
      Cookie[] cookies = request.getCookies();
      if (cookies != null) {
         for (Cookie storedCookie : cookies) {
            if (storedCookie.getName().equals("userCredentials")) {
               String[] credentials = storedCookie.getValue().split(",");
               boolean found = false;
               for (String credential : credentials) {
                  String[] parts = credential.split(":");
                  if (parts[0].equals(userId) && parts[1].equals(password)) {
                     found = true;
                     break;
                  }
               }
               if (found) {
                  out.println("<h2>Login Successful!</h2>");
                  return;
               }
            }
         }
      }
      out.println("<h2>Login Failed! Invalid credentials.</h2>");
   }
}
```

PROGRAM 9

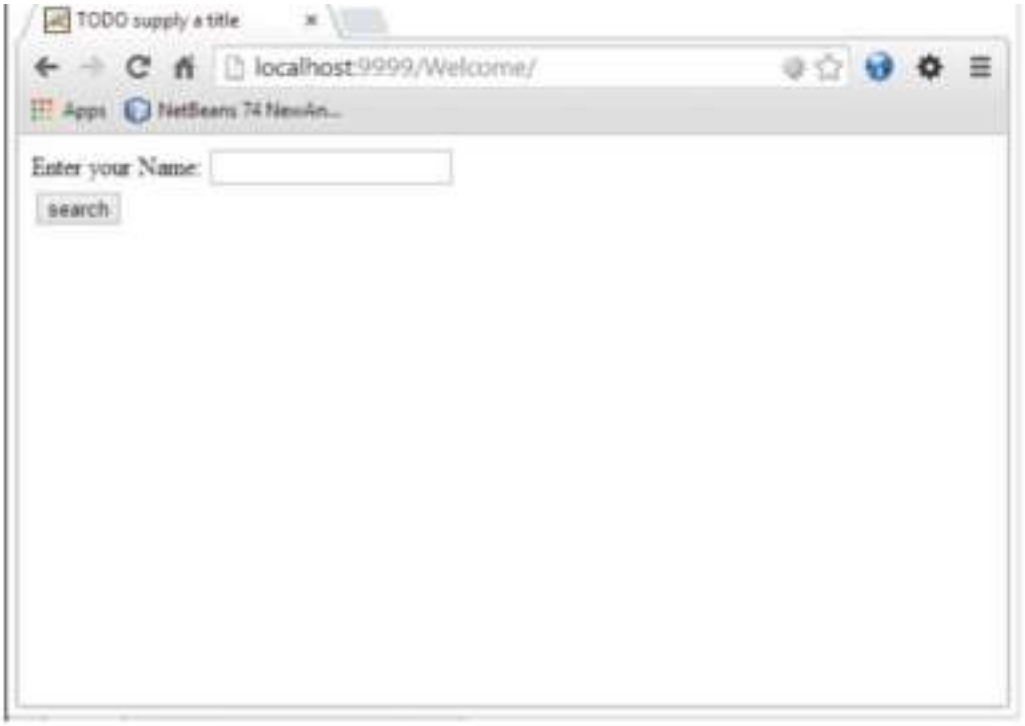
Objective:

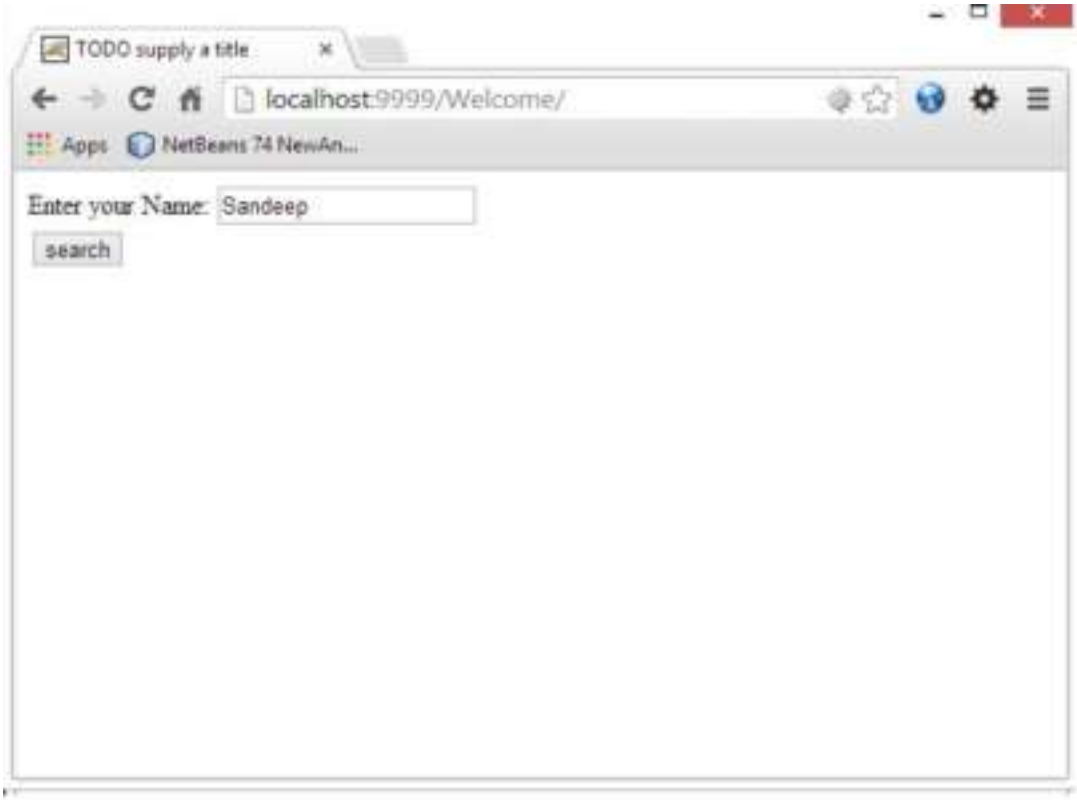
Install a database (Mysql or Oracle). Create a table which should contain at least the following fields: name, password, email-id, phone number Write a java program/servlet/JSP to connect to that database and extract data from the tables and display them. Insert the details of the users who register with the web site, whenever a new user clicks the submit button in the registration page.

Code:

```
CREATE TABLE `users` (
	`user_name` VARCHAR(100) NOT NULL,
	`password` VARCHAR(100) NOT NULL,
	`phone_number` NUMBER(100) NOT NULL,
	`email_address` VARCHAR(50) NOT NULL,
	PRIMARY KEY (`user_name`)
)
COLLATE='latin1_swedish_ci'
ENGINE=InnoDB
```

OUTPUT:


TODO supply a title
localhost:9999/Welcome/
Apps
NetBeans 74 NewAn...
Enter your Name:
search


TODO supply a title
localhost:9999/Welcome/
Apps
NetBeans 74 NewAn...
Enter your Name:
Sandeep
search


localhost:9999/Welcome/
localhost:9999/Welcome/Search?uname=Sand
Apps
NetBeans 7.4 New An...
Result:
NAME
Sandeep
PASSWORD
sandeep
EMAILID
sandy05.1991@gmail.com
COUNTRY
India

PROGRAM 10

Objective:

Write a JSP which insert the details of the 3 or 4 users who register with the the web site by using registration form. Authenticate the user when he submits the login form using the username and password.

Theory:

For creating registration form, you must have a table in the database. You can write the database logic in JSP file, but separating it from the JSP page is better approach. Here, we are going to use DAO, Factory Method, DTO and Singletion design patterns. There are many files:

o index.jsp for getting the values from the user

o User.java, a bean class that have properties and setter and getter methods.

o process.jsp, a jsp file that processes the request and calls the methods

o Provider.java, an interface that contains many constants like DRIVER_CLASS, CONNECTION_URL, USERNAME and PASSWORD

o ConnectionProvider.java, a class that returns an object of Connection. It uses the Singleton and factory method design pattern.

o RegisterDao.java, a DAO class that is responsible to get access to the database

Output:

Authenticating the user when he submits the login form using the username and password.

VALUE ADDITION

Objective:

Design and implement a simple shopping cart example with session tracking API.

Theory:

Based on the above analysis, let us summarize the components we had identified:

1. ModelList.jsp

2. ShoppingCart.jsp

3. CartBean.java

4. CartItemBean.java

5. CartController.java CartBean:

- ArrayList of CartItemBean
- totalCost of items in cart

CartItemBean:

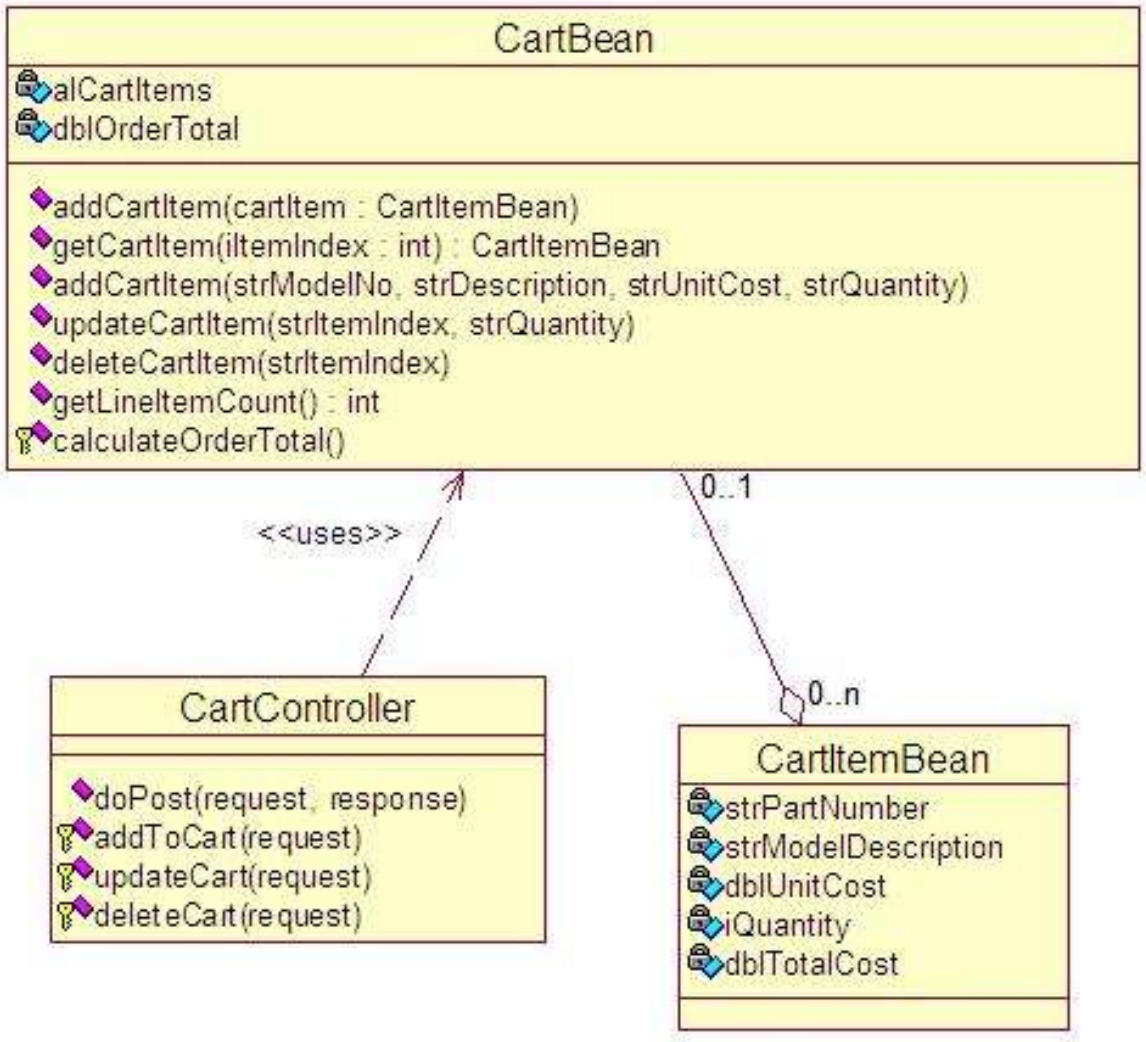
- Part number for the model
- Model description
- Unit cost
- Quantity
- Total item cost

Identifying methods:

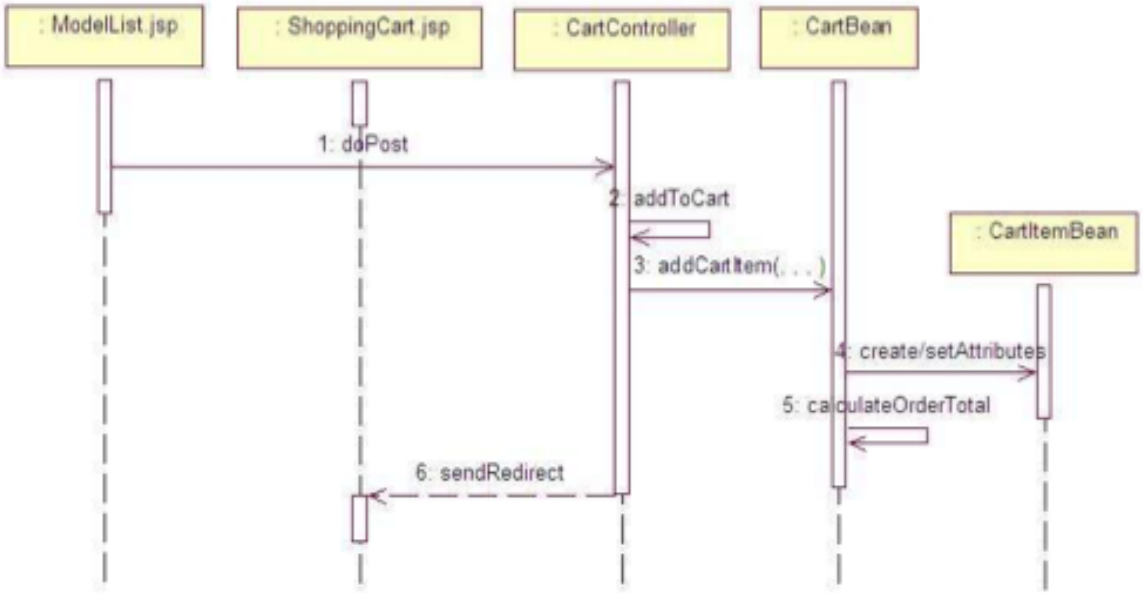
CartController:

This servlet controller needs to implement the doGet/doPost methods. It needs to handle the events for addToCart, updateCart and deleteCart. It is a better design to handle each of these events separately. So, based on this discussion, the CartController will need to have the following methods:

- doGet/doPost – Delegates request to different event methods like
- addToCart etc.addToCart – Handle the add to cart functionality
- updateCart – Handle the update cart item functionality
- deleteCart – Handle delete cart item from cart functionality



Sequence diagram for Add To Cart:



Setting up the environment for development:

The following will be approximate directory structure of the ShoppingCart project.

ShoppingCart
- ModelList.jsp
- ShoppingCart.jsp

WEB-INF

src
- in.techfreaks.shoppingcart.beans.CartBean.java
- in.techfreaks.shoppingcart.beans.CartItemBean.java
- in.techfreaks.shoppingcart.servlet.CartController.java

classes

lib
- jstl.jar
- standard.jar