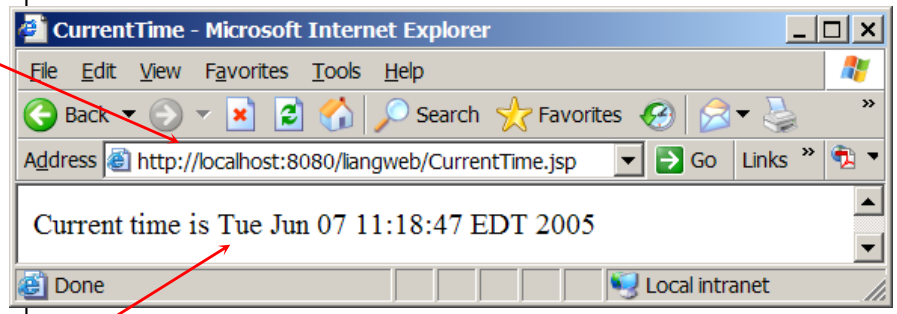
Chapter 38 JavaServer Page

Objectives

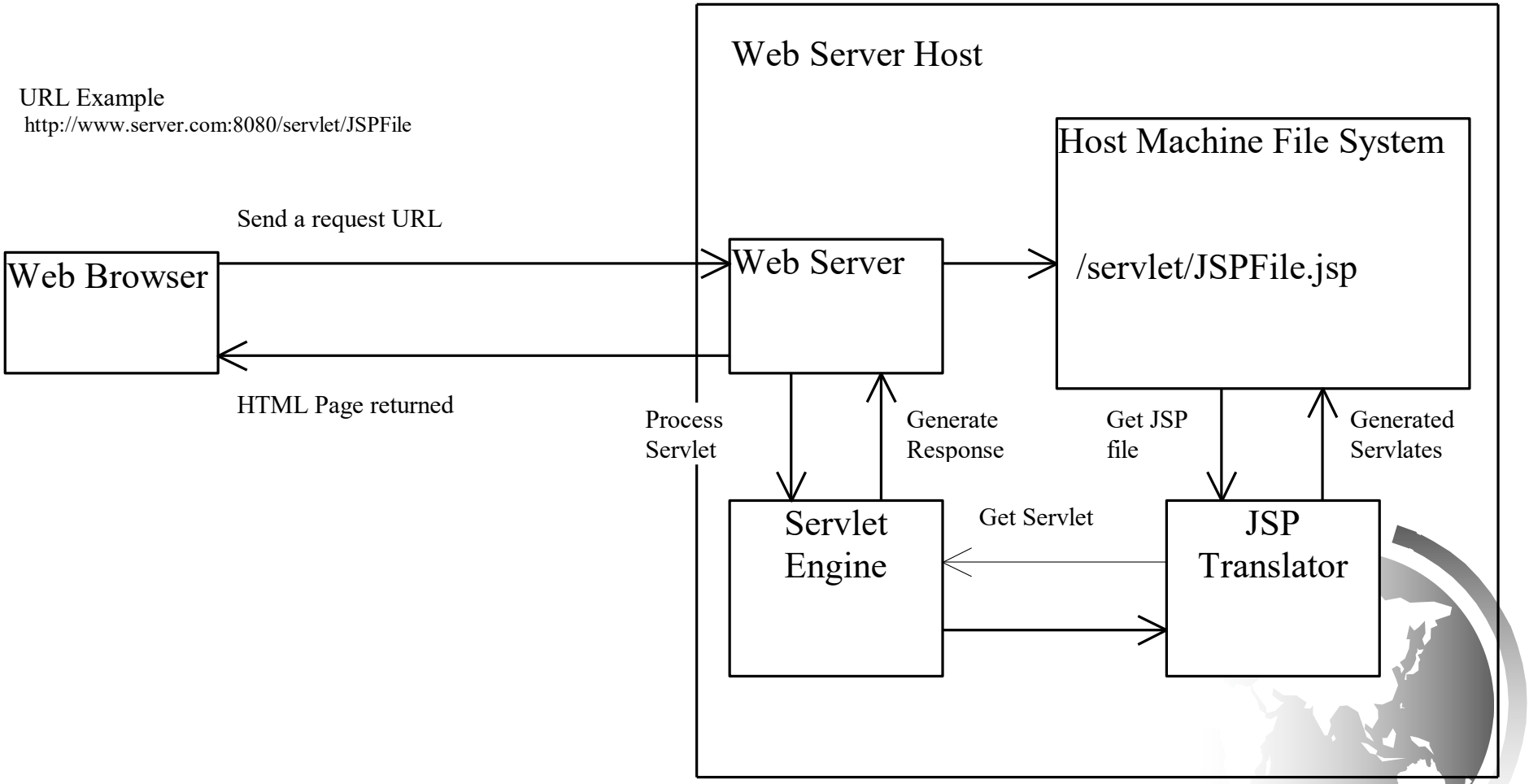
- To create a simple JSP page (§38.2).
- To explain how a JSP page is processed (§38.3).
- To use JSP constructs to code JSP script (§38.4).
- To use predefined variables and directives in JSP (§§38.5-38.6).
- To use JavaBeans components in JSP (§38.7-38.9).
- To develop database applications using JSP (§38.7-38.9).
- To forward requests from a JSP page to another (§38.10).

A Simple JSP

```
<!-- CurrentTime.jsp -->
<HTML>
<HEAD>
<TITLE>
CurrentTime
</TITLE>
</HEAD>
<BODY>
Current time is <%= new java.util.Date() %>
</BODY>
</HTML>
```



How Is a JSP Processed?



JSP Constructs

There are three types of scripting constructs you can use to insert Java code into the resultant servlet. They are *expressions*, *scriptlets*, and *declarations*.

expression

scriptlet

declaration

A JSP expression is used to insert a Java expression directly into the output. It has the following form:

```
<%= Java-expression %>
```

The expression is evaluated, converted into a string, and sent to the output stream of the servlet.

JSP Constructs

There are three types of scripting constructs you can use to insert Java code into the resultant servlet. They are *expressions*, *scriptlets*, and *declarations*.

expression

scriptlet

declaration

A JSP scriptlet enables you to insert a Java statement into the servlet's jspService method, which is invoked by the service method. A JSP scriptlet has the following form:

```
<% Java statement %>
```

JSP Constructs

There are three types of scripting constructs you can use to insert Java code into the resultant servlet. They are *expressions*, *scriptlets*, and *declarations*.

expression

scriptlet

declaration

A JSP declaration is for declaring methods or fields into the servlet. It has the following form:

<%! Java method or field declaration %>

JSP Comment

HTML comments have the following form:

<!-- HTML Comment -->

If you don't want the comment appear in the resultant HTML file, use the following comment in JSP:

<%-- JSP Comment --%>

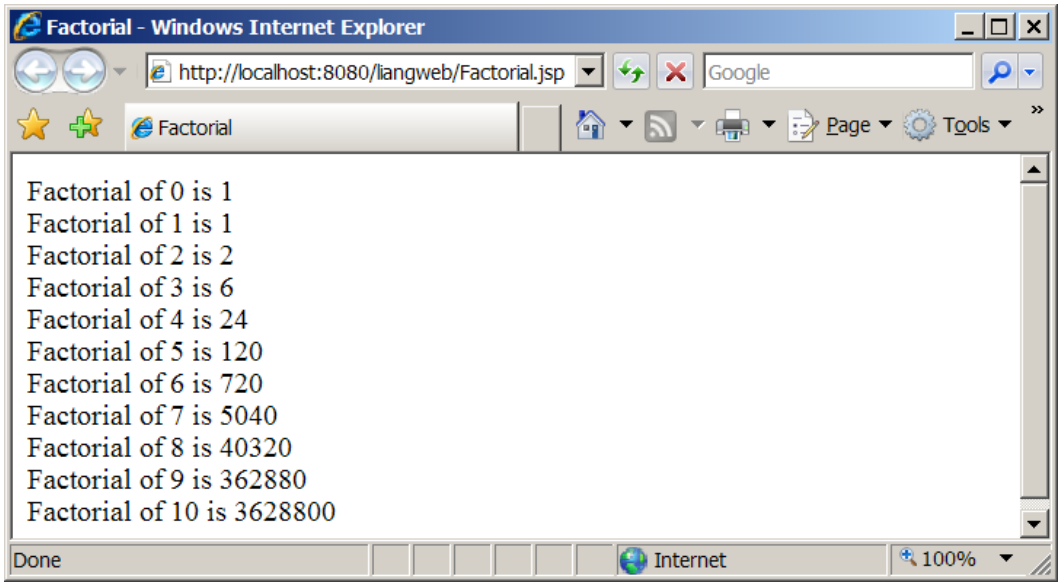
Listing 38.1 Computing Factorials

```
<HTML>
<HEAD>
<TITLE>
Factorial
</TITLE>
</HEAD>
<BODY>

<%  for (int i = 0; i <= 10; i++) { %>
Factorial of <%= i %> is
<%= computeFactorial(i) %> <br />
<%  } %>

<%! private long computeFactorial(int n) {
      if (n == 0)
        return 1;
      else
        return n * computeFactorial(n - 1);
    }
%>

</BODY>
</HTML>
```

JSP scriptlet

JSP expression

JSP declaration



JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Represents the client's request, which is an instance of HttpServletRequest. You can use it to access request parameters, HTTP headers such as cookies, hostname, etc.

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request

response

out

session

application

config

pagecontext

page

Represents the servlet's response, which is an instance of HttpServletResponse. You can use it to set response type and send output to the client.

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Represents the character output stream, which is an instance of PrintWriter obtained from response.getWriter(). You can use it to send character content to the client.

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Represents the HttpSession object associated with the request, obtained from request.getSession().

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Represents the ServletContext object for storing persistent data for all clients. The difference between session and application is that session is tied to one client, but application is for all clients to share persistent data.

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Represents the ServletConfig object for the page.

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Represents the PageContext object. PageContext is a new class introduced in JSP to give a central point of access to many page attributes.

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

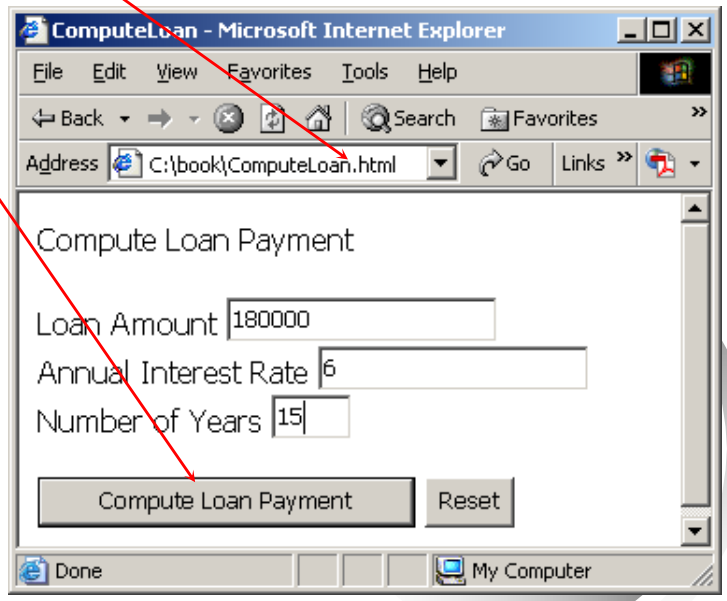
Page is an alternative to this.

Example 38.2 Computing Loan

Write an HTML page that prompts the user to enter loan amount, annual interest rate, and number of years. Clicking the Compute Loan Payment button invokes a JSP to compute and display the monthly and total loan payment.

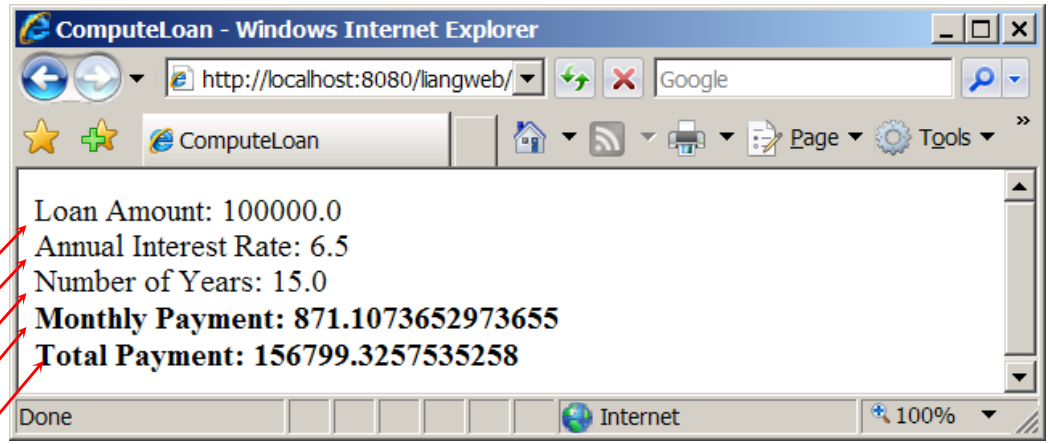
```
<!-- ComputeLoan.html -->
<html>
<head>
<title>ComputeLoan</title>
</head>
<body>
Compute Loan Payment

<form method="get" action="ComputeLoan.jsp">
<p>Loan Amount
    <input type="text" name="loanAmount"><br>
Annual Interest Rate
    <input type="text" name="annualInterestRate"><br>
Number of Years <input type="text" name="numberOfYears"
size="3"></p>
<p><input type="submit" name="Submit" value="Compute Loan
Payment">
   <input type="reset" value="Reset"></p>
</form>
</body>
</html>
```

Run



```
<!-- ComputeLoan.jsp -->
<html>
<head>
<title>ComputeLoan</title>
</head>
<body>
<% double loanAmount = Double.parseDouble(
     request.getParameter("loanAmount"));
   double annualInterestRate = Double.parseDouble(
     request.getParameter("annualInterestRate"));
   double numberOfYears = Integer.parseInt(
     request.getParameter("numberOfYears"));
   double monthlyInterestRate = annualInterestRate / 1200;
   double monthlyPayment = loanAmount * monthlyInterestRate /
     (1 - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
   double totalPayment = monthlyPayment * numberOfYears * 12; %>
Loan Amount: <%= loanAmount %><br>
Annual Interest Rate: <%= annualInterestRate %><br>
Number of Years: <%= numberOfYears %><br>
<b>Monthly Payment: <%= monthlyPayment %><br>
Total Payment: <%= totalPayment %><br></b>
</body>
</html>
```

Predefined variable



JSP Directives

A JSP directive is a statement that gives the JSP engine information about the JSP page. For example, if your JSP page uses a Java class from a package other than the java.lang package, you have to use a directive to import this package. The general syntax for a JSP directive is as follows:

```
<%@ directive attribute="value" %>, or

<%@ directive attribute1="value1"
              attribute2="value2"
              ...
              attributen="vlauen" %>
```

Three JSP Directives

Three possible directives are the following: page, include, and tablib.

page

include

tablib

page lets you provide information for the page, such as importing classes and setting up content type. The page directive can appear anywhere in the JSP file.

Three JSP Directives

Three possible directives are the following: page, include, and tablib.

page

include

tablib

include lets you insert a file to the servlet when the page is translated to a servlet. The include directive must be placed where you want the file to be inserted.

Three JSP Directives

Three possible directives are the following: page, include, and tablib.

page

include

tablib

tablib lets you define custom tags.

Attributes for *page* Directives

import
contentType
session
buffer
autoFlush
isThreadSafe
errorPage
isErrorPage

Specifies one or more packages to be imported for this page. For example, the directive <%@ page import="java.util.*, java.text.*" %> imports java.util.* and java.text.*.

Attributes for page Directives

import
contentType
session
buffer
autoFlush
isThreadSafe
errorPage
isErrorPage

Specifies the MIME type for the resultant JSP page. By default, the content type is text/html for JSP. The default content type for servlets is text/plain.

Attributes for page Directives

import
contentType
session
buffer
autoFlush
isThreadSafe
errorPage
isErrorPage

Specifies a boolean value to indicate whether the page is part of the session. By default, session is true.

Attributes for page Directives

import
contentType
session
buffer
autoFlush
isThreadSafe
errorPage
isErrorPage

Specifies the output stream buffer size. By default, it is 8KB. For example, the directive <%@ page buffer="10KB" %> specifies that the output buffer size is 10KB. The directive <%@ page buffer="none" %> specifies that a buffer is not used.

Attributes for page Directives

import
contentType
session
buffer
autoFlush
isThreadSafe
errorPage
isErrorPage

Specifies a boolean value to indicate whether the output buffer should be automatically flushed when it is full or whether an exception should be raised when the buffer overflows. By default, this attribute is true. In this case, the buffer attribute cannot be none.

Attributes for page Directives

import
contentType
session
buffer
autoFlush
isThreadSafe
errorPage
isErrorPage

Specifies a boolean value to indicate whether the page can be accessed simultaneously without data corruption. By default, it is true. If it is set to false, the JSP page will be translated to a servlet that implements the SingleThreadModel interface.

Attributes for page Directives

import
contentType
session
buffer
autoFlush
isThreadSafe
errorPage
isErrorPage

errorPage specifies a JSP page that is processed when an exception occurs in the current page. For example, the directive <%@ page errorPage="HandleError.jsp" %> specifies that HandleError.jsp is processed when an exception occurs.

- isErrorPage specifies a boolean value to indicate whether the page can be used as an error page. By default, this attribute is false.

```
<!-- ComputeLoan.jsp -->
<html>
<head>
<title>ComputeLoan Using the Loan Class</title>
</head>
<body>
<%@ page import = "chapter38.Loan" %>
<% double loanAmount = Double.parseDouble(
     request.getParameter("loanAmount"));
   double annualInterestRate = Double.parseDouble(
     request.getParameter("annualInterestRate"));
   int numberOfYears = Integer.parseInt(
     request.getParameter("numberOfYears"));
   Loan loan = new Loan(annualInterestRate, numberOfYears,
loanAmount);
%>
Loan Amount: <%= loanAmount %><br>
Annual Interest Rate: <%= annualInterestRate %><br>
Number of Years: <%= numberOfYears %><br>
<b>Monthly Payment: <%= loan.monthlyPayment() %><br>
Total Payment: <%= loan.totalPayment() %><br></b>
</body>
</html>
```

Example: Computing Loan Using the Loan Class

Use the Loan class to simplify Example 38.2. You can create an object of Loan class and use its monthlyPayment() and totalPayment() methods to compute the monthly payment and total payment.

Import a class. The class must be placed in a package (e.g. package chapter38.

JavaBeans Component in JSP

Recall that a class is a JavaBeans component if it has the following three features:

The class is public.

The class has a public constructor with no arguments.

The class is serializable. (This requirement is not necessary in JSP.)

Using JavaBeans in JSP

To create an instance for a JavaBeans component, use the following syntax:

```
<jsp:useBean id="objectName"
scope="scopeAttribute“ class="ClassName" />
```

This syntax is equivalent to

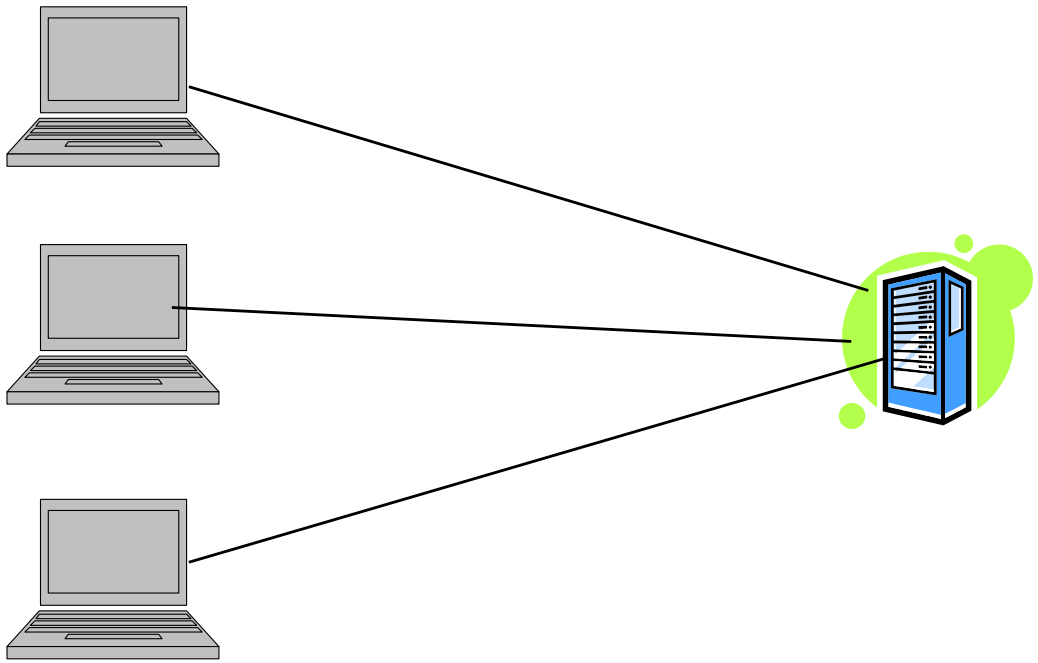
```
<% ClassName objectName = new ClassName() %>
```

except that the scope attribute specifies the scope of the object.

Scope Attributes

application
session
page
request

Specifies that the object is bound to the application. The object can be shared by all sessions of the application.



Scope Attributes

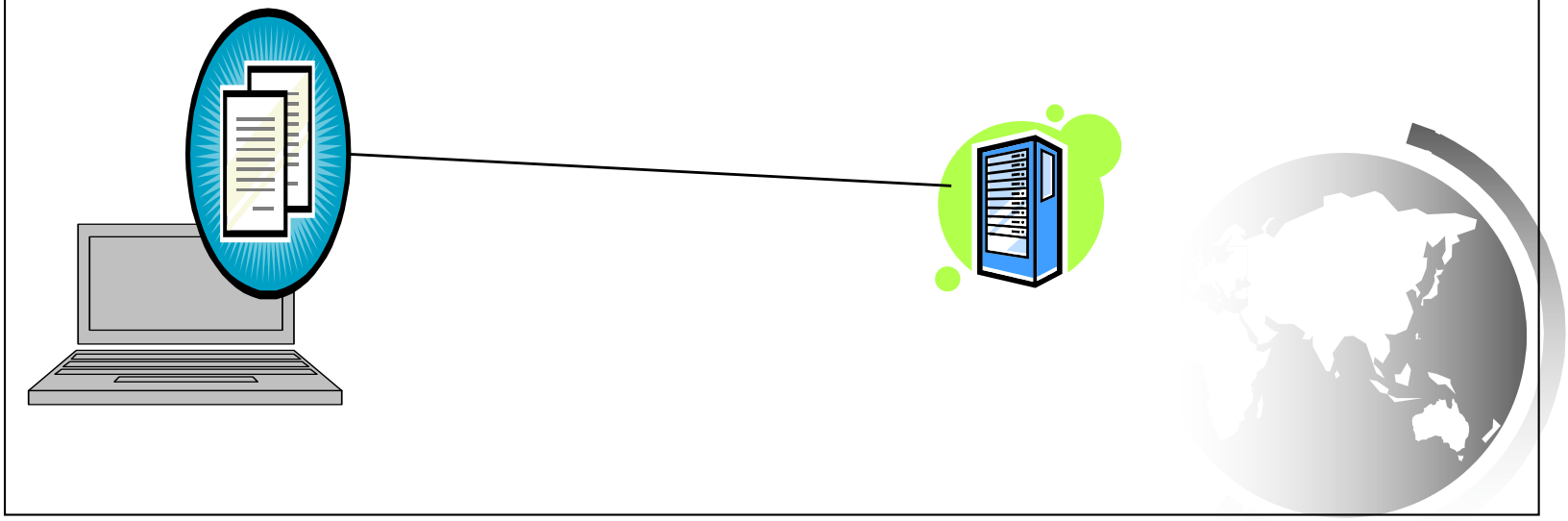
application

session

page

request

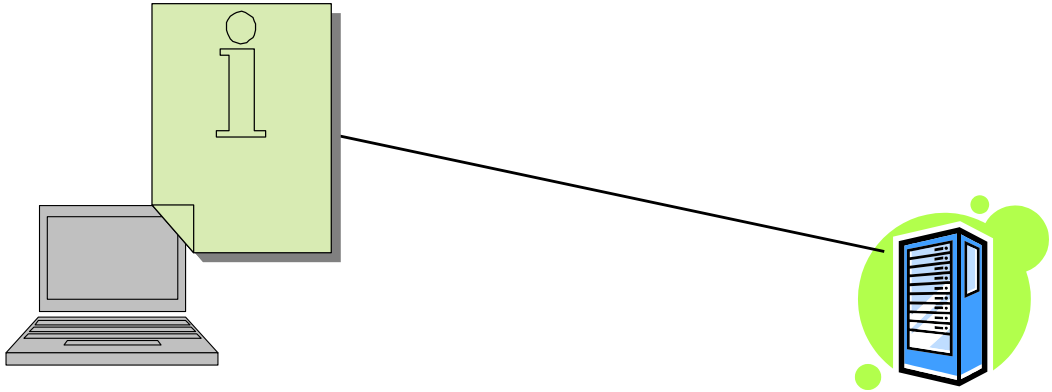
Specifies that the object is bound to the client's session. Recall that a client's session is automatically created between a Web browser and Web server. When a client from the same browser accesses two servlets or two JSP pages on the same server, the session is the same.



Scope Attributes

application
session
page
request

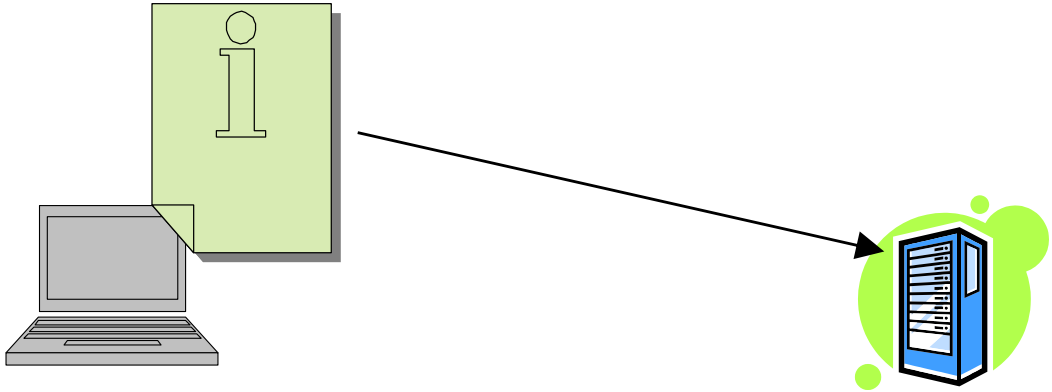
The default scope, which specifies that the object is bound to the page.



Scope Attributes

application
session
page
request

Specifies that the object is bound to the client's request.



How Does JSP Find an Object

When <jsp:useBean id="objectName" scope="scopeAttribute" class="ClassName" /> is processed, the JSP engine first searches for the object of the class with the same id and scope. If found, the preexisting bean is used; otherwise, a new bean is created.

Another Syntax for Creating a Bean

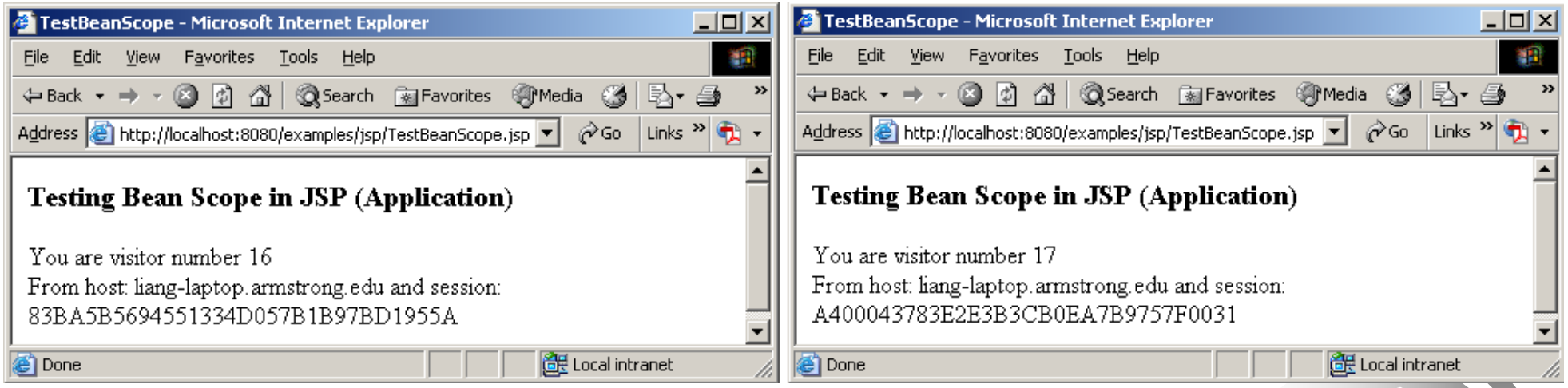
Here is another syntax for creating a bean using the following statement:

```
<jsp:useBean id="objectName"  scope="scopeAttribute“
    class="ClassName" >

 some statements

</jsp:useBean>
```

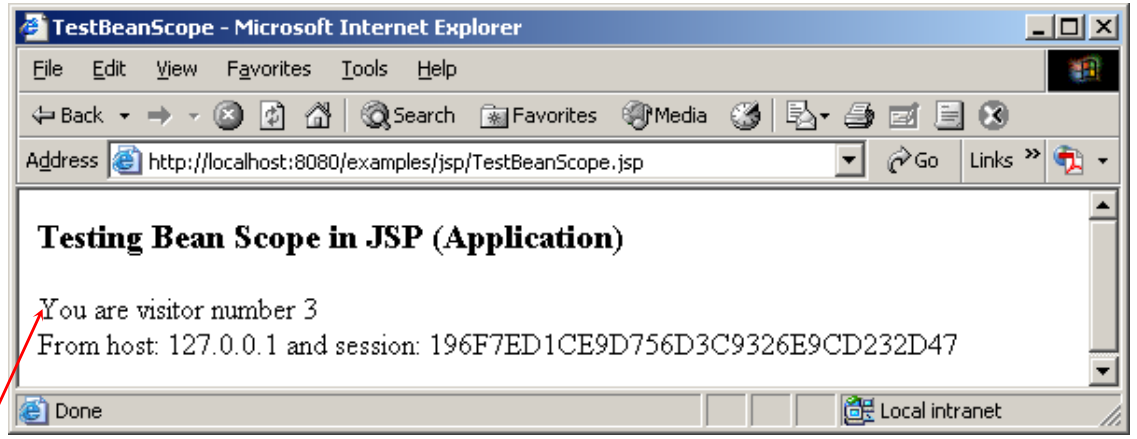
The statements are executed when the bean is created. If the bean with the same id and className already exists, the statements are not executed.

Example: Testing Bean Scope

This example creates a JavaBeans component named Count and uses it to count the number of visits to a page.



Run




```
<!-- TestBeanScope.jsp -->
<%@ page import = "chapter38.Count" %>
<jsp:useBean id="count" scope="application" class="chapter38.Count">
</jsp:useBean>
<HTML>
<HEAD>
<TITLE>TestBeanScope</TITLE>
</HEAD>
<BODY>
<H3>
Testing Bean Scope in JSP (Application)
</H3>
<% count.increaseCount(); %>
You are visitor number <%= count.getCount() %><br>
From host: <%= request.getRemoteHost() %>
and session: <%= session.getId() %>
</BODY>
</HTML>
```

```
package chapter40;

public class Count {
  private int count = 0;

  /** Return count property */
  public int getCount() {
    return count;
  }

  /** Increase count */
  public void increaseCount() {
    count++;
  }
}
```

Getting and Setting Properties

By convention, a JavaBeans component provides the getter and setter methods for reading and modifying its private properties. You can get the property in JSP using the following syntax:

```
<jsp:getProperty name="beanId“
property=“age" />
```

This is equivalent to

```
<%= beanId.getAge() %>
```

Getting and Setting Properties, cont.

You can set the property in JSP using the following syntax:

```
<jsp:setProperty name="beanId“ property=“age“
value=“30" />
```

This is equivalent to

```
<% beanId.setAge(30); %>
```

Associating Properties with Input Parameters

Often properties are associated with input parameters. Suppose you want to get the value of the input parameter named score and set it to the JavaBeans property named score. You may write the following code:

```
<% double score = Double.parseDouble(
   request.getParameter("score")); %>
<jsp:setProperty name="beanId" property="score"
  value="<%= score %>" />
```

Associating Properties with Input Parameters, cont.

This is cumbersome. JSP provides a convenient syntax that can be used to simplify it as follows:

```
<jsp:setProperty name="beanId" property="score"
  param="score" />
```

Instead of using the value attribute, you use the param attribute to name an input parameter. The value of this parameter is set to the property.

Associating All Properties

Often the bean property and the parameter have the same name. You can use the following convenient statement to associate all the bean properties in beanId with the parameters that match the property names.

```
<jsp:setProperty name="beanId" property="*" />
```

Example: Computing Loan Using JavaBeans

Use JavaBeans to simplify Example 38.3 by associating the bean properties with the input parameters.

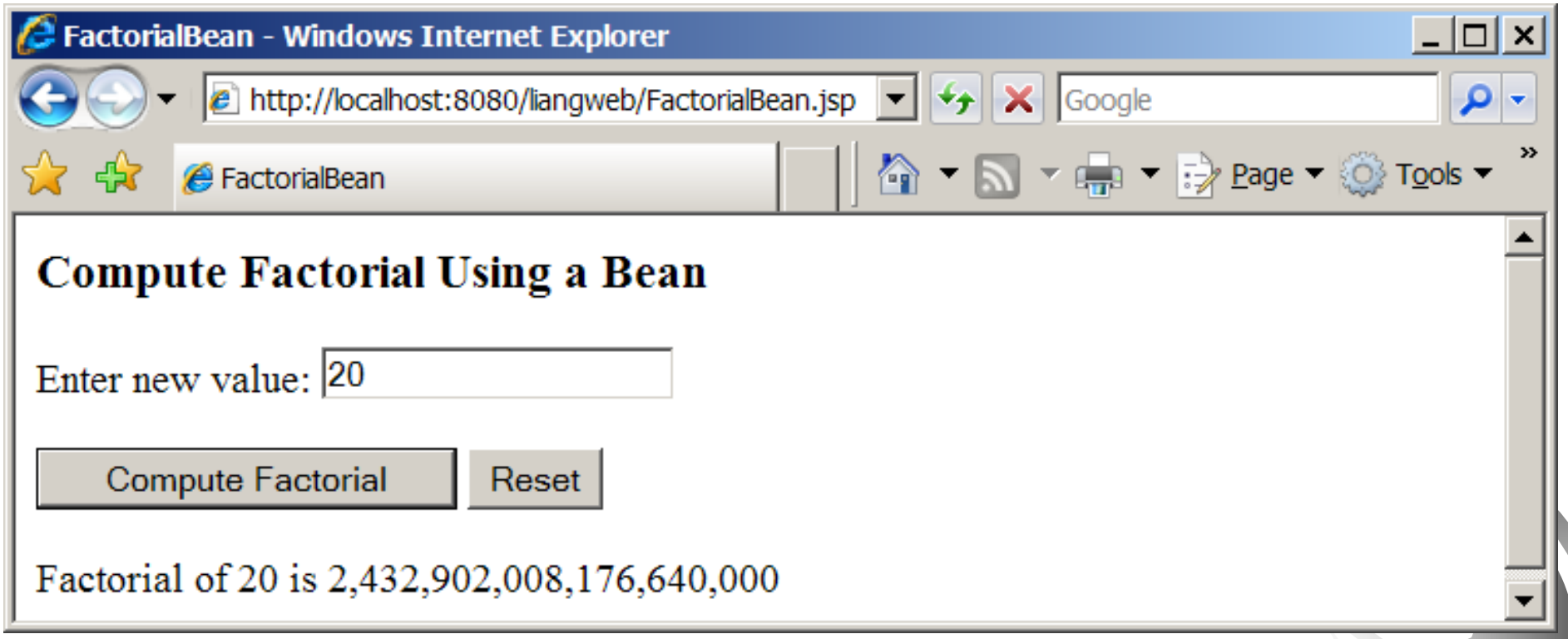
```
<!-- ComputeLoan.jsp -->
<html>
<head>
<title>ComputeLoan Using the Loan Class</title>
</head>
<body>
<%@ page import = "chapter38.Loan" %>
<jsp:useBean id="loan" class="chapter38.Loan"></jsp:useBean>
<jsp:setProperty name="loan" property="*" />
Loan Amount: <%= loan.getLoanAmount() %><br>
Annual Interest Rate: <%= loan.getAnnualInterestRate() %><br>
Number of Years: <%= loan.getNumOfYears() %><br>
<b>Monthly Payment: <%= loan.monthlyPayment() %><br>
Total Payment: <%= loan.totalPayment() %><br></b>
</body>
</html>
```

Getting

Associating the bean properties with the input parameters.

Example: Computing Factorials Using JavaBeans

Create a JavaBeans component named FactorialBean and use it to compute the factorial of an input number in a JSP page named FactorialBean.jsp.



Run

```
<!-- FactorialBean.jsp -->
<%@ page import = "chapter38.FactorialBean" %>
<jsp:useBean id="factorialBeanId" class="chapter38.FactorialBean" >
</jsp:useBean>
<jsp:setProperty name="factorialBeanId" property="*" />
<HTML>
<HEAD>
<TITLE>
FactorialBean
</TITLE>
</HEAD>
<BODY>
<H3>
Compute Factorial Using a Bean
</H3>
<FORM method="post">
Enter new value: <INPUT NAME="number"><BR><BR>
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Compute Factorial">
<INPUT TYPE="RESET" VALUE="Reset">
<P>Factorial of
<jsp:getProperty name="factorialBeanId" property="number" /> is
<%@ page import="java.text.*" %>
<% NumberFormat format = NumberFormat.getNumberInstance(); %>
<%= format.format(factorialBeanId.getFactorial()) %>
</FORM>
</BODY>
</HTML>
```

Associating the bean properties with the input parameters.

Getting

Getting number

```
package chatper40;

public class FactorialBean {
  private int number;

  /** Return number property */
  public int getNumber() {
    return number;
  }

  /** Set number property */
  public void setNumber(int newValue) {
    number = newValue;
  }

  /** Obtain factorial */
  public long getFactorial() {
    long factorial = 1;
    for (int i = 1; i <= number; i++)
      factorial *= i;
    return factorial;
  }
}
```

DESIGN GUIDE

Mixing a lot of Java code with HTML in a JSP page makes the code difficult to read and to maintain. You should move the Java code to a .java file as much as you can.

NewFactorialBean

```
<!-- NewFactorialBean.jsp -->
<%@ page import = "chapter38.NewFactorialBean" %>
<jsp:useBean id = "factorialBeanId"
  class = "chapter38.NewFactorialBean" scope = "page" >
</jsp:useBean>
<jsp:setProperty name = "factorialBeanId" property = "*" />
<html>
  <head>
    <title>
      FactorialBean
    </title>
  </head>
  <body>
  <h3>Compute Factorial Using a Bean</h3>
  <form method = "post">
    Enter new value: <input name = "number" /><br /><br />
    <input type = "submit" name = "Submit"
      value = "Compute Factorial" />
    <input type = "reset" value = "Reset" /><br /><br />
    Factorial of
      <jsp:getProperty name = "factorialBeanId"
        property = "number" /> is
      <%= NewFactorialBean.format(factorialBeanId.getFactorial()) %>
    </form>
  </body>
</html>
```

Getting

```
<!-- DisplayTime.jsp -->
<%@page pageEncoding = "GB18030"%>
<%@ page import = "chapter38.TimeBean" %>
<jsp:useBean id = "timeBeanId"
  class = "chapter38.TimeBean" scope = "application" >
</jsp:useBean>
<jsp:setProperty name = "timeBeanId" property = "*" />
<html>
  <head>
    <title>
      Display Time
    </title>
  </head>
  <body>
  <h3>Choose locale and time zone</h3>
    Current time is
      <%=
timeBeanId.currentTimeString(timeBeanId.getLocaleIndex(),
       timeBeanId.getTimeZoneIndex()) %>
  </body>
<html>
```

Getting

TimeBean

Forwarding Requests from JavaServer Pages

Web applications developed using JSP generally consist of many pages linked together. JSP provides a forwarding tag in the following syntax that can be used to forward a page to another page.

```
<jsp:forward page="destination" />
```

Example: Browsing Database Tables

This example creates a JSP database application that browses tables. When you start the application, the first page prompts the user to enter the JDBC driver, URL, username, and password for a database. After you login to the database, you can select a table to browse. Upon clicking the Browse Table Content button, the table content is displayed.

Run