Server Side Development

ITM 602

Java Server Pages

Topics

- JSP Fundamentals
- JSP Scripting Elements
- JSP Implicit Objects
- JSP Directives
- JSP Actions
- JSP Example (Loan Calculator)
- Servlets & JSPs together
- Tag Libraries
- Deploying and Running a JSP Application

Java Server Pages (JSP)

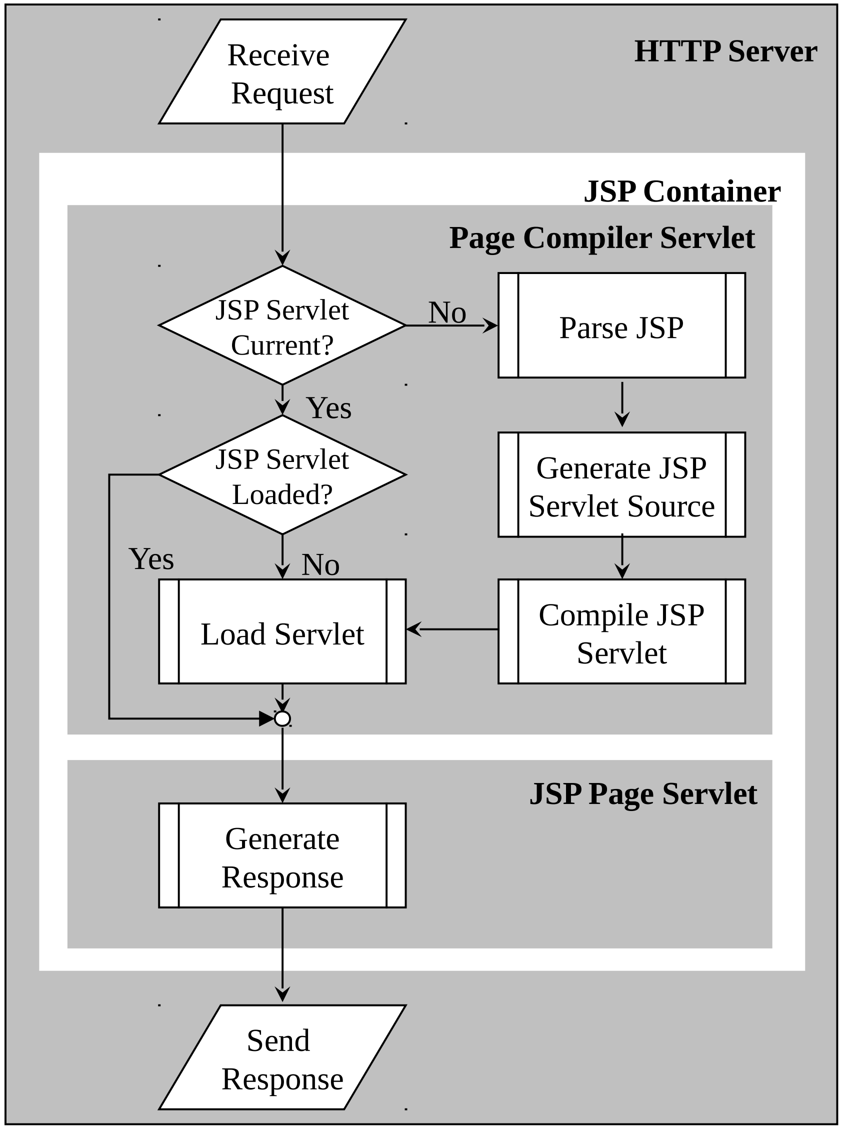
Fundamentals

- Java Server Pages are HTML pages embedded with snippets of Java code.
  - It is an inverse of a Java Servlet
- Four different elements are used in constructing JSPs
  - Scripting Elements
  - Implicit Objects
  - Directives
  - Actions

Java Server Pages (JSP)

Architecture

- JSPs run in two phases
  - Translation Phase
  - Execution Phase
- In translation phase JSP page is compiled into a servlet
  - called JSP Page Implementation class
- In execution phase the compliled JSP is processed



Scripting Elements

Types

- There are three kinds of scripting elements
  - Declarations
  - Scriptlets
  - Expressions

Declarations

Basics

- Declarations are used to define methods & instance variables
  - Do not produce any output that is sent to client
  - Embedded in <%! and %> delimiters

  Example:

```
<%!
    Public void jspDestroy() {
        System.out.println("JSP Destroyed");
    }
    Public void jspInit() {
        System.out.println("JSP Loaded");
    }
    int myVar = 123;
%>
```

  - The functions and variables defined are available to the JSP Page as well as to the servlet in which it is compiled

Scriptlets

Basics

- Used to embed java code in JSP pages.
  - Contents of JSP go into _JSPpageservice() method
  - Code should comply with syntactical and semantic constuct of java
  - Embedded in <% and %> delimiters

Example:

```
<%
      int x = 5;
      int y = 7;
      int z = x + y;
%>
```

Expressions

Basics

- Used to write dynamic content back to the browser.
  - If the output of expression is Java primitive the value is printed back to the browser
  - If the output is an object then the result of calling toString on the object is output to the browser
  - Embedded in <%= and %> delimiters

Example:

- <%=“Fred”+ “ “ + “Flintstone %>

  prints “Fred Flintstone” to the browser
- <%=Math.sqrt(100)%>

  prints 10 to the browser

Java Implicit Objects

Scope

- Implicit objects provide access to server side objects
  - e.g. request, response, session etc.
- There are four scopes of the objects
  - Page: Objects can only be accessed in the page where they are referenced
  - Request: Objects can be accessed within all pages that serve the current request.

    (Including the pages that are forwarded to and included in the original jsp page)
  - Session: Objects can be accessed within the JSP pages for which the objects are defined
  - Application: Objects can be accessed by all JSP pages in a given context

Java Implicit Objects
List

- request: Reference to the current request
- response: Response to the request
- session: session associated woth current request
- application: Servlet context to which a page belongs
- pageContext: Object to access request, response, session and application associated with a page
- config: Servlet configuration for the page
- out: Object that writes to the response output stream
- page: instance of the page implementation class (this)
- exception: Available with JSP pages which are error pages

Java Implicit Objects

Example

```
<html>
  <head>
    <title>Implicit Objects</title>
  </head>
  <body style="font-family:verdana;font-size:10pt">
    <p>
      Using Request parameters...<br>
      <b>Name:</b> <%= request.getParameter("name") %>
    </p>
    <p>
      <% out.println("This is printed using the out implicit
              variable"); %>
    </p>
    <p>
      Storing a string to the session...<br>
      <% session.setAttribute("name", "Meeraj"); %>
      Retrieving the string from session...<br>
      <b>Name:</b> <%= session.getAttribute("name") %>
    </p>
    <p>
      Storing a string to the application...<br>
      <% application.setAttribute("name", "Meeraj"); %>
      Retrieving the string from application...<br>
      <b>Name:</b>
      <%= application.getAttribute("name") %>
    </p>
    <p>
      Storing a string to the page context...<br>
      <% pageContext.setAttribute("name", "Meeraj"); %>
      Retrieving the string from page context...</br>
      <b>Name:</b>
      <%= pageContext.getAttribute("name") %>
    </p>
  </body>
</html>
```

Example Implicit Objects

Deploy & Run

- Save file:
  - $TOMCAT_HOME/webapps/jsp/Implicit.jsp
- Access file
  - http://localhost:8080/jsp/Implicit.jsp?name=Sanjay
- Results of the execution

```
Using Request parameters...
        Name: sanjay
This is printed using the out implicit variable
Storing a string to the session...
        Retrieving the string from session...
        Name: Meeraj
Storing a string to the application...
        Retrieving the string from application...
        Name: Meeraj
Storing a string to the page context...
        Retrieving the string from page context...
        Name: Meeraj
```

Directives

Basics & Types

- Messages sent to the JSP container
  - Aids the container in page translation
- Used for
  - Importing tag libraries
  - Import required classes
  - Set output buffering options
  - Include content from external files
- The jsp specification defines three directives
  - Page: provder information about page, such as scripting language that is used, content type, or buffer size
  - Include – used to include the content of external files
  - Taglib – used to import custom actions defined in tag libraries

Page Directives

Basics & Types

- Page directive sets page properties used during translation
  - JSP Page can have any number of directives
  - Import directive can only occur once
  - Embedded in <%@ and %> delimiters
- Different directives are
  - Language: (Default Java) Defines server side scripting language (e.g. java)
  - Extends: Declares the class which the servlet compiled from JSP needs to extend
  - Import: Declares the packages and classes that need to be imported for using in the java code (comma separated list)
  - Session: (Default true) Boolean which says if the session implicit variable is allowed or not
  - Buffer: defines buffer size of the jsp in kilobytes (if set to none no buffering is done)

Page Directives

Types con't.

- Different directives are (cont'd.)
  - autoFlush:When true the buffer is flushed when max buffer size is reached (if set to false an exception is thrown when buffer exceeds the limit)
  - isThreadSafe: (default true) If false the compiled servlet implements SingleThreadModel interface
  - Info: String returned by the getServletInfo() of the compiled servlet
  - errorPage: Defines the relative URI of web resource to which the response should be forwarded in case of an exception
  - contentType: (Default text/html) Defines MIME type for the output response
  - isErrorPage: True for JSP pages that are defined as error pages
  - pageEncoding: Defines the character encoding for the jsp page

Page Directives

Example

```
<%@
      page language="java"
      buffer="10kb"
      autoflush="true"
      errorPage="/error.jsp"
      import="java.util.*, javax.sql.RowSet"
%>
```

Include Directive

Basics

- Used to insert template text and JSP code during the translation phase.
  - The content of the included file specified by the directive is included in the including JSP page
- Example
  - <%@ include file=“included.jsp” %>

JSP Actions

Basics & Types

- Processed during the request processing phase.
  - As opposed to JSP directives which are processed during translation
- Standard actions should be supported by J2EE compliant web servers
- Custom actions can be created using tag libraries
- The different actions are
  - Include action
  - Forward action
  - Param action
  - useBean action
  - getProperty action
  - setProperty action
  - plugIn action

JSP Actions

Include

- Include action used for including resources in a JSP page
  - Include directive includes resources in a JSP page at translation time
  - Include action includes response of a resource into the response of the JSP page
  - Same as including resources using RequestDispatcher interface
  - Changes in the included resource reflected while accessing the page.
  - Normally used for including dynamic resources
- Example
  - <jsp:include page=“inlcudedPage.jsp”>
  - Includes the the output of includedPage.jsp into the page where this is included.

JSP Actions

Forward

- Forwards the response to other web specification resources
  - Same as forwarding to resources using RequestDispatcher interface
- Forwarded only when content is not committed to other web application resources
  - Otherwise an IllegalStateException is thrown
  - Can be avoided by setting a high buffer size for the forwarding jsp page
- Example
  - <jsp:forward page="Forwarded.html">
  - Forwards the request to Forwarded.html

JSP Actions

Param

- Used in conjunction with Include & Forward actions to include additional request parameters to the included or forwarded resource
- Example

```
<jsp:forward page=“Param2.jsp”>
    <jsp:param name=“FirstName” value=“Sanjay”>
</jsp:forward>
```

  - This will result in the forwarded resource having an additional parameter FirstName with a value of Sanjay

JSP Actions

useBean

- Creates or finds a Java object with the defined scope.
  - Object is also available in the current JSP as a scripting variable
- Syntax:

```
<jsp:useBean id=“name”
scope=“page | request | session | application”
class=“className” type=“typeName” |
bean=“beanName” type=“typeName” |
type=“typeName” />
```

  - At least one of the type and class attributes must be present
  - We can’t specify values for bith the class and bean name.
- Example

```
<jsp:useBean id=“myName” scope=“request”
        class=“java.lang.String”>
        <% firstName=“Sanjay”; %>
</jsp:useBean>
```

JSP Actions

get/setProperty

- getProperty is used in conjunction with useBean to get property values of the bean defined by the useBean action
- Example (getProperty)
  - <jsp:getProperty name="myBean" property="firstName" />
  - Name corresponds to the id value in the useBean
  - Property refers to the name of the bean property
- setProperty is used to set bean properties
- Example (setProperty)
  - <jsp:setProperty name="myBean" property="firstName" value="Sanjay"/>
  - Sets the name property of myBean to SanjayExample (setProperty)
  - <jsp:setProperty name="myBean" property="firstName" param="fname"/>
  - Sets the name property of myBean to the request parameter fname
  - <jsp:setProperty name="myBean" property="*">
  - Sets property to the corresponding value in request

JSP Actions

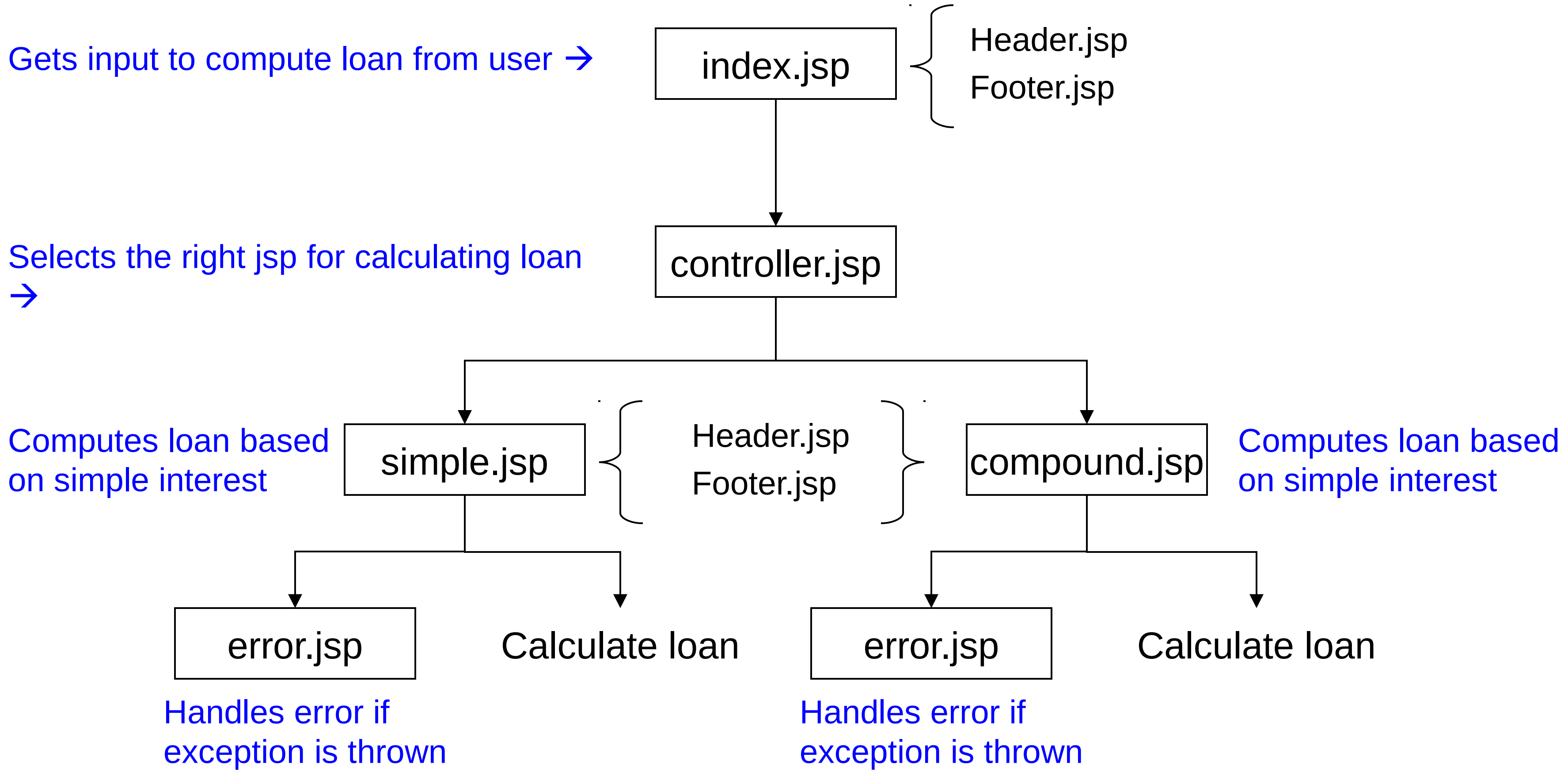
plugIn

- Enables the JSP container to render appropriate HTML (based on the browser type) to:
  - Initiate the download of the Java plugin
  - Execution of the specified applet or bean
- plugIn standard action allows the applet to be embedded in a browser neutral fashion
- Example

```
<jsp: plugin type=“applet” code=“MyApplet.class”
      codebase=“/”>
      <jsp:params>
      <jsp:param name=“myParam” value=“122”/>
      </jsp:params>
      <jsp:fallback><b>Unable to load applet</b></jsp:fallback>
</jsp:plugin>
```

Example

Loan Calculator


Gets input to compute loan from user →
index.jsp
Header.jsp
Footer.jsp
Selects the right jsp for calculating loan →
controller.jsp
Computes loan based on simple interest
simple.jsp
Header.jsp
Footer.jsp
compound.jsp
Computes loan based on simple interest
error.jsp
Calculate loan
error.jsp
Calculate loan
Handles error if exception is thrown
Handles error if exception is thrown

Loan Calculator

index.jsp

```
<html>
  <head>
    <title>Include</title>
  </head>
  <body style="font-family:verdana;font-size:10pt;">
    <%@ include file="header.html" %>
    <form action="controller.jsp">
      <table border="0" style="font-family:verdana;font-size:10pt;">
        <tr>
          <td>Amount:</td>
          <td><input type="text" name="amount" />
        </tr>
        <tr>
          <td>Interest in %:</td>
          <td><input type="text" name="interest"/></td>
        </tr>
        <tr>
          <td>Compound:</td>
          <td><input type="radio" name="type" value="C"
              checked/></td>
        </tr>
<tr>
          <td>Simple:</td>
          <td><input type="radio" name="type" value="S"
              /></td>
        </tr>
        <tr>
          <td>Period:</td>
          <td><input type="text" name="period"/></td>
        </tr>
      </table>
      <input type="submit" value="Calculate"/>
    </form>
    <jsp:include page="footer.jsp"/>
  </body>
</html>
```

Loan Calculator

Miscelaneous

controller.jsp

```
<%
  String type = request.getParameter("type");
  if(type.equals("S")) {
%>
<jsp:forward page="/simple.jsp"/>
<%
  } else {
%>
 <jsp:forward page="/compound.jsp"/>
<%
  }
%>
```

error.jsp

```
<%@ page isErrorPage="true" %>
<html>
  <head>
    <title>Simple</title>
  </head>
  <body style="font-family:verdana;font-size:10pt;">
    <%@ include file="header.html" %>
    <p style="color=#FF0000"><b><%=
            exception.getMessage() %></b></p>
    <jsp:include page="footer.jsp"/>
  </body>
</html>
```

header.jsp

```
<h3>Loan Calculator</h3>
```

footer.jsp

```
<%= new java.util.Date() %>
```

Loan Calculator

simple.jsp

```
<%@ page errorPage="error.jsp" %>
<%!
  public double calculate(double amount, double
          interest, int period) {
    if(amount <= 0) {
      throw new IllegalArgumentException("Amount
          should be greater than 0: " + amount);
    }
    if(interest <= 0) {
      throw new IllegalArgumentException("Interest
          should be greater than 0: " + interest);
    }
    if(period <= 0) {
      throw new IllegalArgumentException("Period should
          be greater than 0: " + period);
    }
    return amount*(1 + period*interest/100);
  }
%>
```

```
<html>
  <head>
    <title>Simple</title>
  </head>
  <body style="font-family:verdana;font-size:10pt;">
    <%@ include file="header.html" %>
    <%
      double amount =
          Double.parseDouble(request.getParameter("amo
          unt"));
      double interest =
          Double.parseDouble(request.getParameter("inter
          est"));
      int period =
          Integer.parseInt(request.getParameter("period"));
    %>
    <b>Pincipal using simple interest:</b>
    <%= calculate(amount, interest, period) %>
    <br/><br/>
    <jsp:include page="footer.jsp"/>
  </body>
</html>
```

Loan Calculator

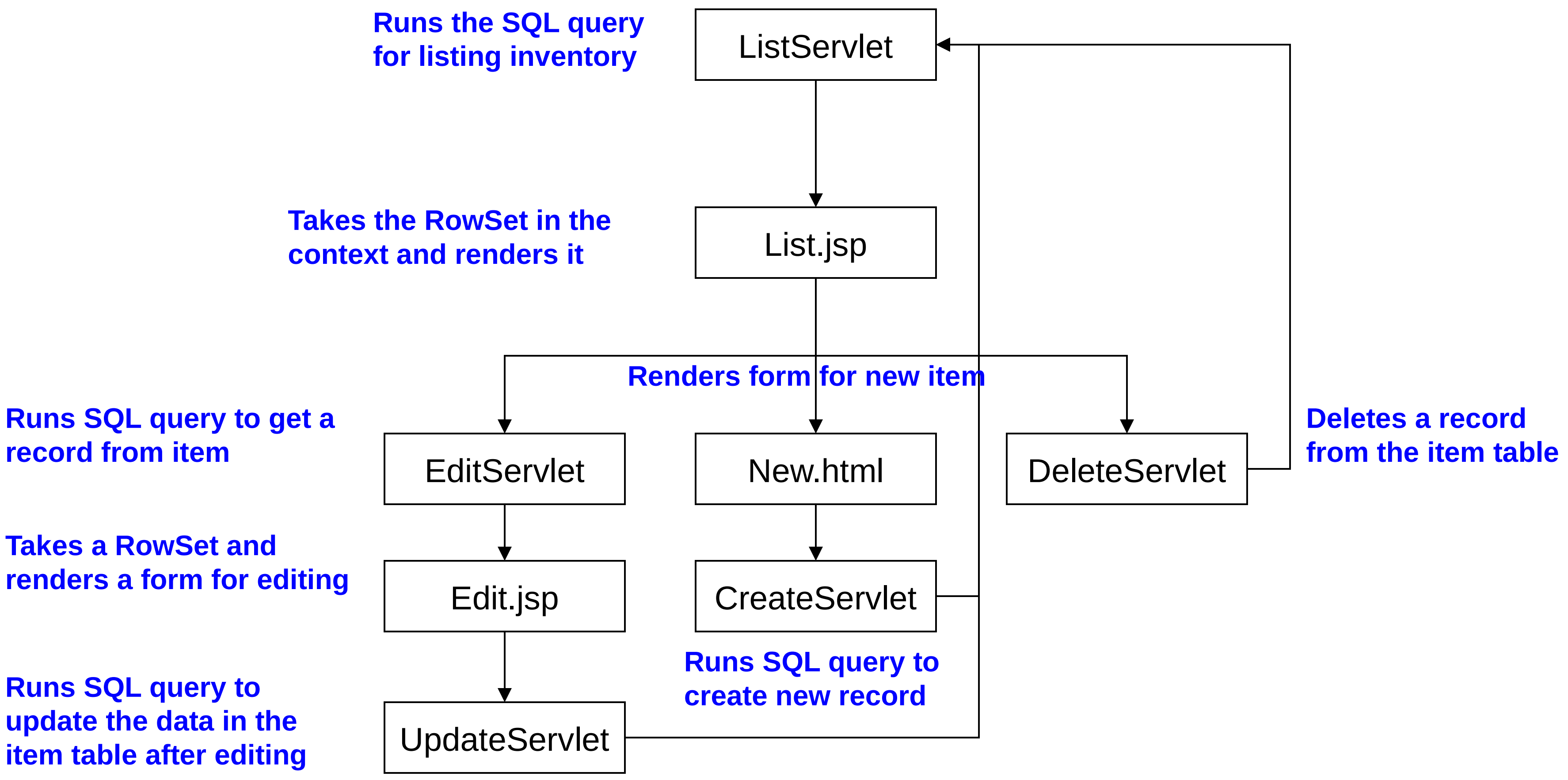
compound.jsp

```
<%@ page errorPage="error.jsp" %>
<%!
  public double calculate(double amount, double
           interest, int period) {
    if(amount <= 0) {
      throw new IllegalArgumentException("Amount
           should be greater than 0: " + amount);
    }
    if(interest <= 0) {
      throw new IllegalArgumentException("Interest
           should be greater than 0: " + interest);
    }
    if(period <= 0) {
      throw new IllegalArgumentException("Period should
           be greater than 0: " + period);
    }
    return amount*Math.pow(1 + interest/100, period);
  }
%>
```

```
<html>
  <head>
    <title>Compound</title>
  </head>
  <body style="font-family:verdana;font-size:10pt;">
    <%@ include file="header.html" %>
    <%
      double amount =
           Double.parseDouble(request.getParameter("amo
           unt"));
      double interest =
           Double.parseDouble(request.getParameter("inte
           rest"));
      int period =
           Integer.parseInt(request.getParameter("period"))
           ;
    %>
   <b>Pincipal using compound interest:</b>
    <%= calculate(amount, interest, period) %>
    <br/><br/>
    <jsp:include page="footer.jsp"/>
  </body>
</html>
```

Example

Inventory


Runs the SQL query for listing inventory
ListServlet
Takes the RowSet in the context and renders it
List.jsp
Renders form for new item
Runs SQL query to get a record from item
EditServlet
New.html
DeleteServlet
Deletes a record from the item table
Takes a RowSet and renders a form for editing
Edit.jsp
CreateServlet
Runs SQL query to create new record
Runs SQL query to update the data in the item table after editing
UpdateServlet

Inventory

ListServlet

```
package edu.albany.mis.goel.servlets;


import javax.servlet.ServletException;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;
import javax.sql.RowSet;
import sun.jdbc.rowset.CachedRowSet;
public class ListServlet extends HttpServlet {
  public void init(ServletConfig config) throws
            ServletException {
    super.init(config);
  }
  public void doPost(HttpServletRequest req,
            HttpServletResponse res)
      throws ServletException {
    doGet(req, res);
  }
```

```
public void doGet(HttpServletRequest req,
            HttpServletResponse res)
      throws ServletException {
    try {
      // Load the driver class
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      // Define the data source for the driver
      String sourceURL = "jdbc:odbc:inventoryDB";
      RowSet rs = new CachedRowSet();
      rs.setUrl(sourceURL);
      rs.setCommand("select * from item");
      rs.execute();
      req.setAttribute("rs", rs);
      getServletContext().getRequestDispatcher("/List.jsp").
        forward(req, res);
    } catch(Exception ex) {
      throw new ServletException(ex);
    }
  }
}
```

Inventory

EditServlet

```
package edu.albany.mis.goel.servlets;

import javax.servlet.ServletException;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.sql.DriverManager;
import javax.sql.DataSource;
import javax.sql.RowSet;
import sun.jdbc.rowset.CachedRowSet;

public class EditServlet extends HttpServlet {
  public void init(ServletConfig config) throws ServletException {
    super.init(config);
  }
  public void doPost(HttpServletRequest req,
            HttpServletResponse res)
      throws ServletException {
    doGet(req, res);
  }
public void doGet(HttpServletRequest req,
            HttpServletResponse res)
      throws ServletException {
    try {
      // Load the driver class
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      // Define the data source for the driver
      String sourceURL = "jdbc:odbc:inventoryDB";
      RowSet rs = new CachedRowSet();
      rs.setUrl(sourceURL);
      rs.setCommand("select * from item where id = ?");
      rs.setInt(1, Integer.parseInt(req.getParameter("id")));
      rs.execute();
      req.setAttribute("rs", rs);

            getServletContext().getRequestDispatcher("/Edit.jsp
            ").forward(req, res);
    } catch(Exception ex) {
        throw new ServletException(ex);
    }
  }
}
```

Inventory
UpdateServlet

```
package edu.albany.mis.goel.servlets;

import javax.servlet.ServletException;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;
import javax.naming.InitialContext;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
public class UpdateServlet extends HttpServlet {
  public void init(ServletConfig config) throws ServletException {
      super.init(config);
  }
  public void doPost(HttpServletRequest req, HttpServletResponse res)
      throws ServletException {
    doGet(req, res);
  }
public void doGet(HttpServletRequest req, HttpServletResponse res)
     throws ServletException {
    Connection con = null;
 try {
      // Load the driver class
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
     // Define the data source for the driver
      String sourceURL = "jdbc:odbc:inventoryDB";
      // Create a connection through the DriverManager class
      con = DriverManager.getConnection(sourceURL);
      System.out.println("Connected Connection");
      PreparedStatement stmt= con.prepareStatement
        ("update item " +  "set name = ?, " + "description = ?, " + "price = ?, "
         +  "stock = ? " + "where id = ?");
      stmt.setString(1, req.getParameter("name"));
      stmt.setString(2, req.getParameter("description"));
      stmt.setDouble(3, Double.parseDouble(req.getParameter("price")));
      stmt.setInt(4, Integer.parseInt(req.getParameter("stock")));
      stmt.setInt(5, Integer.parseInt(req.getParameter("id")));
      stmt.executeUpdate();
      stmt.close();
      getServletContext().getRequestDispatcher("/List").
      forward(req, res);
    } catch(Exception ex) {
      throw new ServletException(ex);
    } finally {
      try {
        if(con != null) {
          con.close();
        }
      } catch(Exception ex) {
        throw new ServletException(ex);
      }
    }
  }
}
```

Inventory
DeleteServlet

```
package edu.albany.mis.goel.servlets;

import javax.servlet.ServletException;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;
import javax.naming.InitialContext;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
public class DeleteServlet extends HttpServlet {
  public void init(ServletConfig config) throws ServletException {
    super.init(config);
  }
  public void doPost(HttpServletRequest req, HttpServletResponse res)
      throws ServletException {
    doGet(req, res);
  }
public void doGet(HttpServletRequest req, HttpServletResponse res)
      throws ServletException {
    Connection con = null;
```

```
try {
      // Load the driver class
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      // Define the data source for the driver
      String sourceURL = "jdbc:odbc:inventoryDB";
      // Create a connection through the DriverManager class
      con = DriverManager.getConnection(sourceURL);
      System.out.println("Connected Connection");
      // Create Statement
      PreparedStatement stmt =
            con.prepareStatement("delete from item where id = ?");
      stmt.setInt(1, Integer.parseInt(req.getParameter("id")));
      stmt.executeUpdate();
      stmt.close();
      getServletContext().getRequestDispatcher("/List").
      forward(req, res);
    } catch(Exception ex) {
      throw new ServletException(ex);
    } finally {
      try {
        if(con != null) con.close();
      } catch(Exception ex) {
        throw new ServletException(ex);
      }
    }
  }
```

Inventory

CreateServlet

```
package edu.albany.mis.goel.servlets;

import javax.servlet.ServletException;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;
import javax.naming.InitialContext;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
public class CreateServlet extends HttpServlet {
  public void init(ServletConfig config) throws ServletException {
    super.init(config);
  }
  public void doPost(HttpServletRequest req, HttpServletResponse res)
      throws ServletException {
    doGet(req, res);
  }
  public void doGet(HttpServletRequest req, HttpServletResponse res)
      throws ServletException {
   Connection con = null;
    try { // Load the driver class
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      // Define the data source for the driver
      String sourceURL = "jdbc:odbc:inventoryDB";
      // Create a connection through the DriverManager class
      con = DriverManager.getConnection(sourceURL);
      System.out.println("Connected Connection");
       PreparedStatement stmt = con.prepareStatement
          ("insert into item " + "(name,description,price,stock) " +
           "values (?, ?, ?, ?)");
      stmt.setString(1, req.getParameter("name"));
      stmt.setString(2, req.getParameter("description"));
      stmt.setDouble(3, Double.parseDouble(req.getParameter("price")));
      stmt.setInt(4, Integer.parseInt(req.getParameter("stock")));
      stmt.executeUpdate();
      stmt.close();
      getServletContext().getRequestDispatcher("/List").forward(req, res);

    } catch(Exception ex) {
      throw new ServletException(ex);
    } finally {
      try {
        if(con != null) con.close();
      } catch(Exception ex) {
        throw new ServletException(ex);
      }
    }
  }
}
```

Inventory

Edit.jsp

```
<%@page contentType="text/html"%>
<jsp:useBean id="rs" scope="request" type="javax.sql.RowSet" />
<html>
  <head>
    <title>Inventory - Edit</title>
  </head>
  <body style="font-family:verdana;font-size:10pt;">
    <%
      if(rs.next()) {
    %>
    <form action="Update">
      <input name="id" type="hidden" value="<%= rs.getString(1) %>"/>
      <table cellpadding="5" style="font-family:verdana;font-size:10pt;">
        <tr>
          <td><b>Name:</b></td>
          <td>
            <input name="name" type="text" value="<%= rs.getString(2) %>"/>
          </td>
        </tr>
        <tr>
          <td><b>Description:</b></td>
          <td>
            <input name="description" type="text" value="<%= rs.getString(3) %>"/>
          </td>
        </tr>
        <tr>
          <td><b>Price:</b></td>
          <td>
            <input name="price" type="text" value="<%= rs.getString(4) %>"/>
          </td>
        </tr>
        <tr>
          <td><b>Stock:</b></td>
          <td>
            <input name="stock" type="text" value="<%= rs.getString(5) %>"/>
          </td>
        </tr>
        <tr>
          <td></td>
          <td>
            <input type="submit" value="Update"/>
          </td>
        </tr>
      </table>
    <%
      }
    %>
  </body>
</html>
```

Inventory

Edit.jsp

```
<%@page contentType="text/html"%>
<jsp:useBean id="rs" scope="request" type="javax.sql.RowSet" />

<html>
  <head>
    <title>Inventory - List</title>
  </head>
  <body style="font-family:verdana;font-size:10pt;">
    <table cellpadding="5" style="font-family:verdana;font-size:10pt;">
      <tr>
        <th>Name</th>
        <th>Description</th>
        <th>Price</th>
        <th>Stock</th>
        <th></th>
        <th></th>
      </tr>
      <%
        while(rs.next()) {
      %>
      <tr>
        <td><%= rs.getString(2) %></td>
        <td><%= rs.getString(3) %></td>
        <td><%= rs.getString(4) %></td>
        <td><%= rs.getString(5) %></td>
        <td>
          <a href="Delete?id=<%= rs.getString(1) %>">
            Delete
          </a>
        </td>
        <td>
          <a href="Edit?id=<%= rs.getString(1) %>">
            Edit
          </a>
        </td>
      </tr>
      <%
        }
      %>
    </table>
    <a href="New.html">New Item</a>
  </body>
</html>
```

Inventory

New.html

```
<html>
  <head>
    <title>Inventory - Add New Item</title>
  </head>
  <body style="font-family:verdana;font-size:10pt;">

    <form action="Create">
      <table cellpadding="5" style="font-family:verdana;font-size:10pt;">
        <tr>
          <td><b>Name:</b></td>
          <td><input name="name" type="text"/></td>
        </tr>
        <tr>
          <td><b>Description:</b></td>
          <td><input name="description" type="text"/></td>
        </tr>
        <tr>
          <td><b>Price:</b></td>
          <td><input name="price" type="text"/></td>
        </tr>
        <tr>
          <td><b>Stock:</b></td>
          <td><input name="stock" type="text"/></td>
        </tr>
        <tr>
          <td></td>
          <td><input type="submit" value="Create"/></td>
        </tr>

      </table>


  </body>
</html>
```