Course: Web Application Development
Lab Instructor: Assoc. Prof. Dr. Nguyen Van Sinh
Email: nvsinh@hcmiu.edu.vn

Lab 8 - Java Beans and MVC (21/4/2024)

Content:

- **Introduction Java Beans**
- **MVC (Modeling View Controller) Architecture**
- **Practices and refer:**
- *https://www.javatpoint.com/java-bean*
- *https://www.tutorialspoint.com/jsp/jsp_java_beans.htm*

Duration: 3 hours

Part 1: Introduction to Java Beans

- What are Beans?

After all, beans are merely regular Java classes that follow some simple conventions defined by the JavaBeans specification; beans extend no particular class, are in no particular package, and use no particular interface.

- Why use Bean?

Because we already understand the benefit of using separate Java classes instead of embedding large amounts of code directly in JSP pages. Separate classes are easier to write, compile, test, debug, and reuse.

- Basic Bean Use in JSP

- jsp:useBean. In the simplest case, this element builds a new bean. It is normally used as follows:

  ```
  <jsp:useBean id="beanName" class="package.Class" />
  ```

- jsp:getProperty. This element reads and outputs the value of a bean property. Reading a property is a shorthand notation for calling a method of the form getXxx. This element is used as follows:

  ```
  <jsp:getProperty name="beanName" property="propertyName" />
  ```

- jsp:setProperty. This element modifies a bean property (i.e., calls a method of the form setXxx). It is normally used as follows:

  ```
  <jsp:setProperty name="beanName"
      property="propertyName"
      value="propertyValue" />
  ```

- Example 1: create a java bean to set and get string

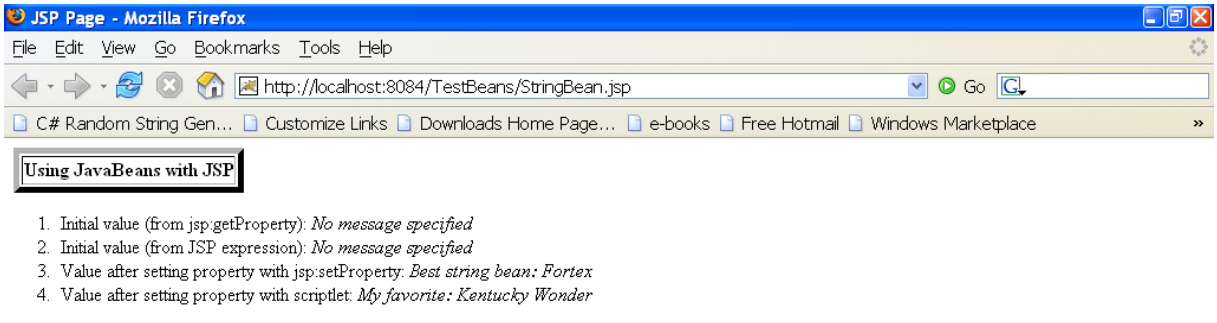
- Open NetBeans, create a new project "TestBean" base on web application.
- Create a new file "StringBean.java" with the source below:

```
/** A simple bean that has a single String property
 *  called message.
 */

public class StringBean {
  private String message = "No message specified";

  public String getMessage() {
    return(message);
  }

  public void setMessage(String message) {
    this.message = message;
  }
}
```

- Create a new file "StringBean.jsp" with source code below in the tag <body>:

```
<BODY>
  <TABLE BORDER=5 ALIGN="CENTER">
    <TR><TH CLASS="TITLE">
        Using JavaBeans with JSP</TABLE>
  <jsp:useBean id="stringBean" class="NameOfPackage.StringBean" />
  <OL>
  <LI>Initial value (from jsp:getProperty):
      <I><jsp:getProperty name="stringBean"
                          property="message" /></I>
  <LI>Initial value (from JSP expression):
      <I><%= stringBean.getMessage() %></I>
  <LI><jsp:setProperty name="stringBean"
                       property="message"
                       value="Best string bean: Fortex" />
      Value after setting property with jsp:setProperty:
      <I><jsp:getProperty name="stringBean"
                          property="message" /></I>
  <LI><% stringBean.setMessage("My favorite: Kentucky Wonder"); %>
      Value after setting property with scriptlet:
      <I><%= stringBean.getMessage() %></I>
  </OL>
</BODY>
```

This is the result after running:



- Example 2: Create a java bean for counter

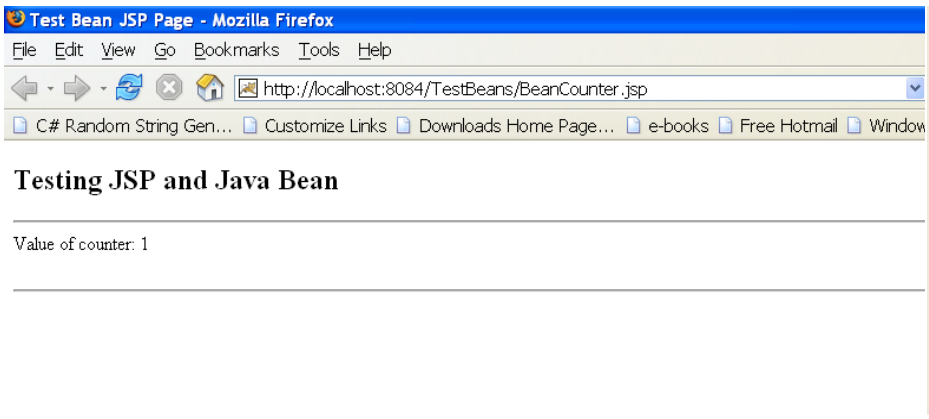
- Create a java file "Counter.java"

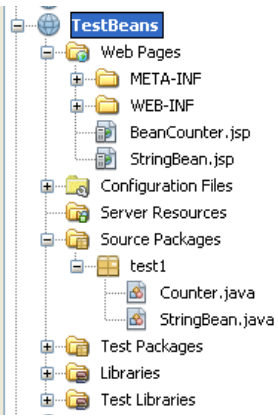
```
public class Counter {
    int count = 0;
    public Counter(){

    }
    //this method to get value
    public int getCount(){
        count++;
        return this.count;
    }
    //this method reset count value
    public void setCount(int counter){
        this.count = count;
    }
}
```

- Create a java file "BeanCounter.jsp"

```
<body>
   <h2>Testing JSP and Java Bean</h2><hr>
        <jsp:useBean id="counter"
                     scope="session"
                     class="test1.Counter"/>
        <jsp:setProperty name="counter"
                         property="count"
                         param="count"/>
        <%
            out.println("Value of counter:
                         "+counter.getCount()+"<BR>");
        %>
</body>
```

This is the result after running:





Organizational Structure of TestBeans

Part 2: MVC Architecture

- As a design pattern

MVC encompasses more of the architecture of an application than is typical for a design pattern. When considered as a design pattern, MVC is fundamentally the same as the Observer pattern.

- Model
  Is the domain-specific representation of the information on which the application operates. Domain logic adds meaning to raw data (for example, calculating whether today is the user's birthday, or the totals, taxes, and shipping charges for shopping cart items).
  Many applications use a persistent storage mechanism (such as a database) to store data. MVC does not specifically mention the data access layer because it is understood to be underneath or encapsulated by the model.
- View
  Renders the model into a form suitable for interaction, typically a user interface element. Multiple views can exist for a single model for different purposes.
- Controller
  Processes and responds to events (typically user actions) and may indirectly invoke changes on the model

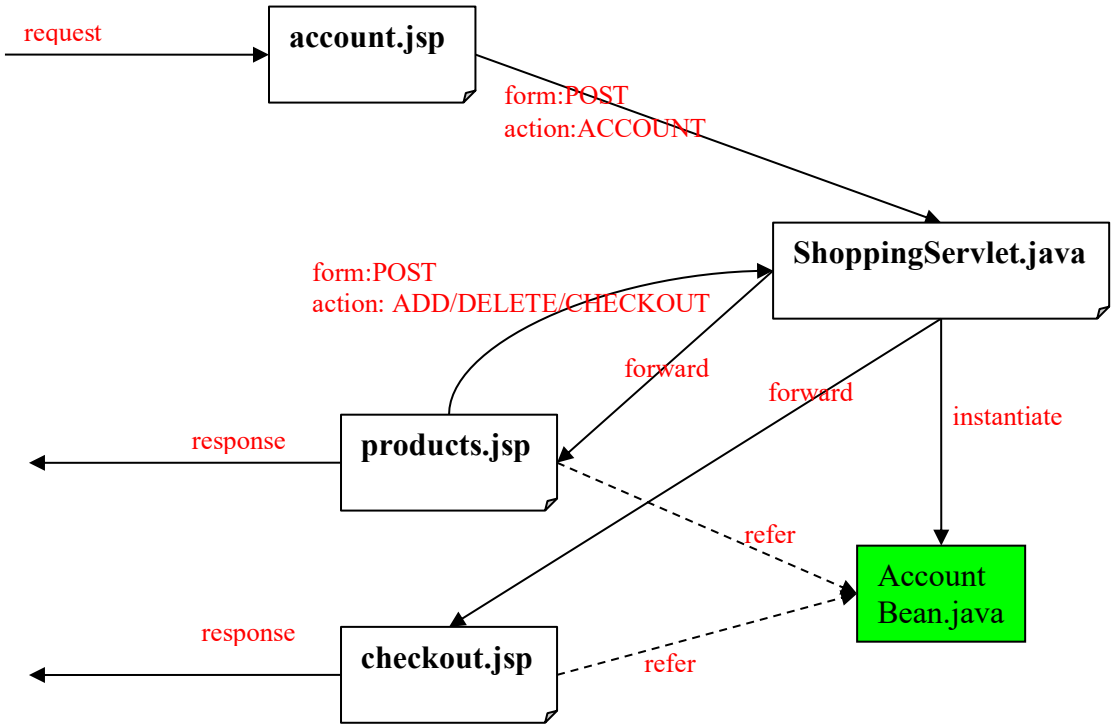
- Implementing MVC with RequestDispatcher (refer chapter 15.2)

The most important point about MVC is the idea of separating the business logic and data access layers from the presentation layer. The syntax is quite simple, and in fact you should be familiar with much of it already. Here is a quick summary of the required steps; the following subsections supply details.

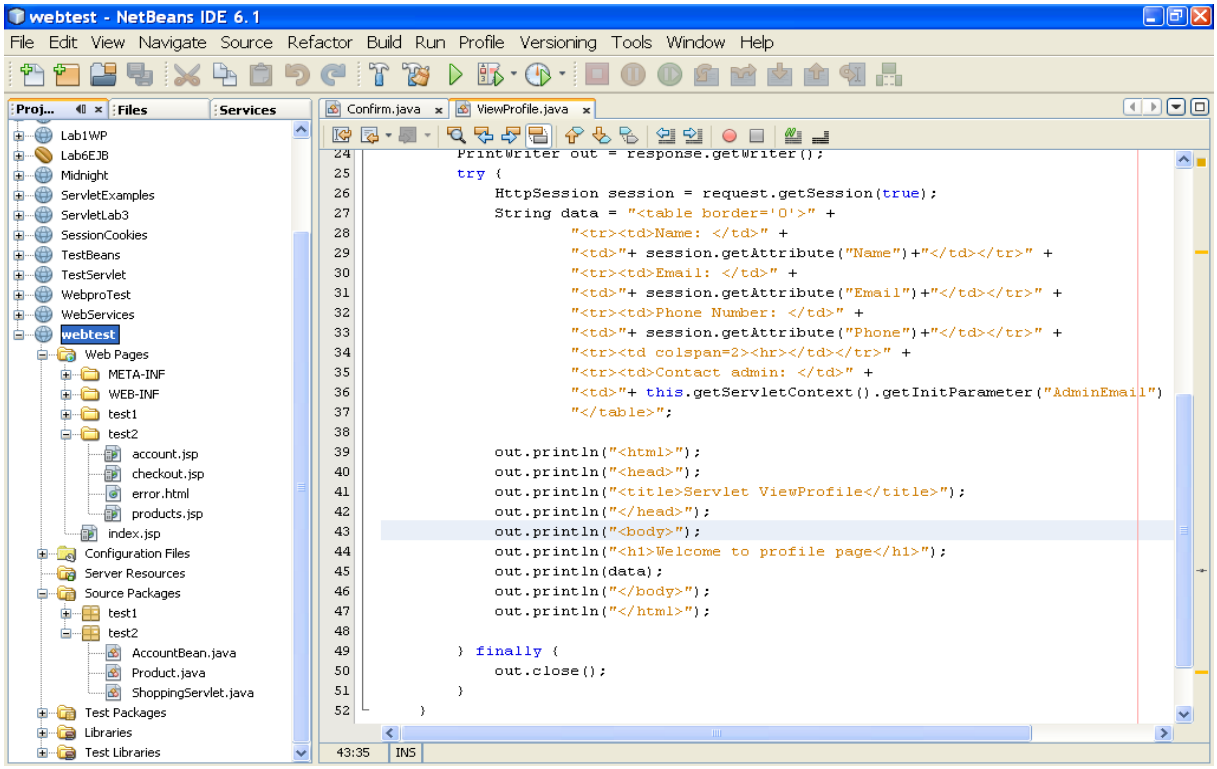
- Define beans to represent the data. Beans are just Java objects that follow a few simple conventions. Your first step is define beans to represent the results that will be presented to the user.
- Use a servlet to handle requests.
- Populate the beans. The servlet invokes business logic (application-specific code) or data-access code to obtain the results. The results are placed in the beans that were defined in step 1.
- Store the bean in the request, session, or servlet context. The servlet calls `setAttribute` on the request, session, or servlet context objects to store a reference to the beans that represent the results of the request.
- Forward the request to a JSP page. The servlet determines which JSP page is appropriate to the situation and uses the `forward` method of `RequestDispatcher` to transfer control to that page.
- Extract the data from the beans. The JSP page accesses beans with `jsp:useBean` and a `scope` matching the location of step 4. The page then uses `jsp:getProperty` to output the bean properties. The JSP page does not create or modify the bean; it merely extracts and displays data that the servlet created.

Part 3: Practices

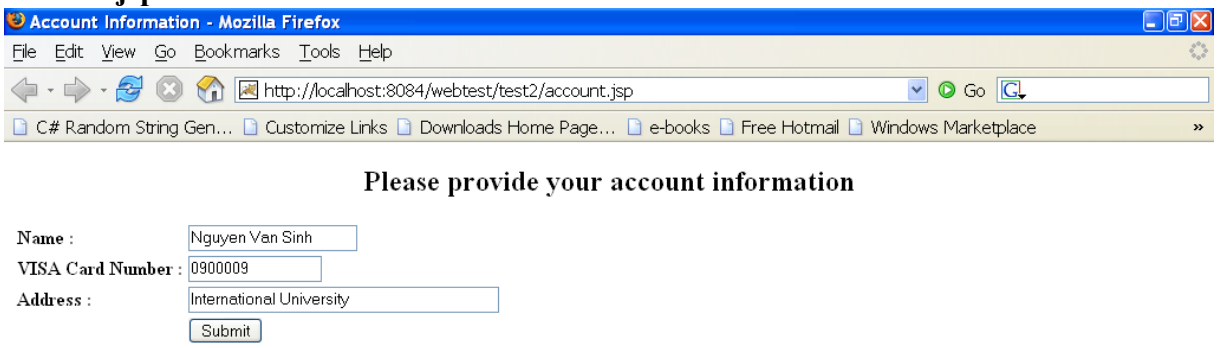
In this part, you will make a simple web application using MVC architecture which applies Servlet, JSP and JavaBeans technology. This web application demonstrates a typical scenario for buying products online which contains the following steps: **provide VISA account information, select products**, and **summary transaction** information.
The big picture of that web application is depicted as follows:



This is the structure of web application.



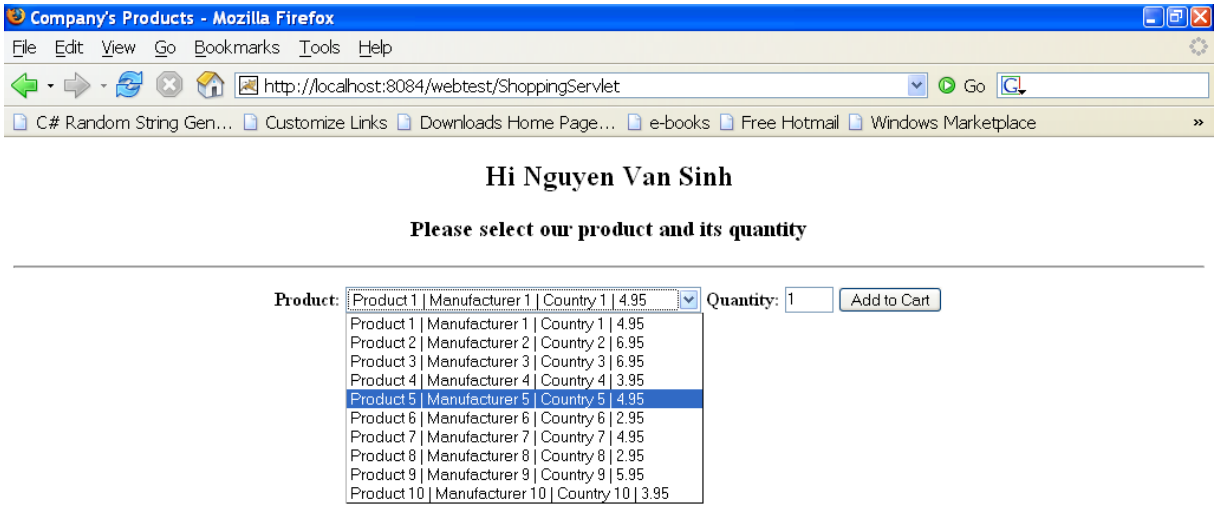
account.jsp



After click Submit, the result like picture below:

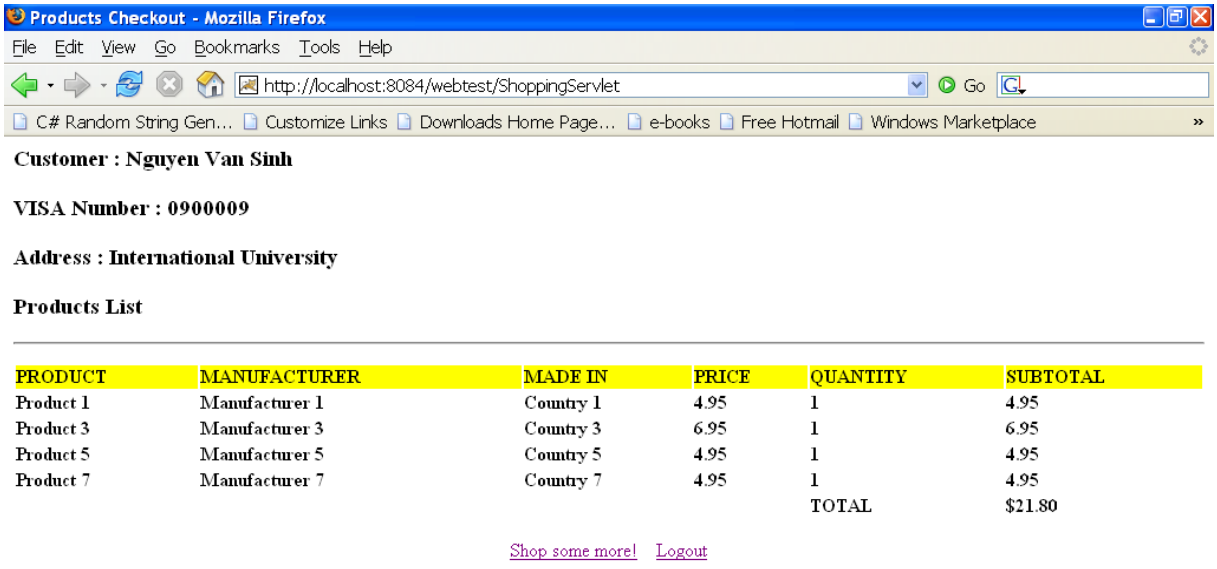
products.jsp

From this page, we can choose product then click "Add to Cart"



This is the result: In this page, you can click Delete to delete Product

When click Checkout the result like this:



If you click "Shop some more" to comeback product.jsp; click to "Logout" comeback to account.jsp