This is an open books, open notes exam. But no portable computers may be used! NO TALKING!

You have three hours to complete the exam. Please be precise and concise in your answers. We advise that you work on the problems in the order in which they appear, since they are in increasing order of difficulty and an increasing order of time requirements.

Score __________________________
Questions

Basics:
1) How long does the REQUEST object live?
   - A. All the time
   - B. Only on page loads
   - C. For a specified timeout period of inactivity
   - D. Until the user closes the browser window

2) How long does the SESSION object live (on the server)?
   - A. All the time
   - B. Only on page loads
   - C. For a specified timeout period of inactivity
   - D. Until the user closes the browser window

3) How do the POST and GET methods differ?
   - A. They are identical
   - B. POST parses data into the URL, GET parses data into the HTTP header
   - C. GET parses data into the URL, POST parses data into the HTTP header
   - D. GET can be used for file uploads, while POST cannot

4) How does the server handle JSP files?
   - A. They are compiled into java source files and then compiled into class files
   - B. They are processed by a scripting engine
   - C. They are compiled into static html files whenever a client requests the page
   - D. None of the above

5) Which is not true about Beans?
   - A. Beans can simplify JSP code
   - B. Beans allow code to be more modular
   - C. Beans make presentation code more readable
   - D. Beans are central in MVC.
Assume that you need to write a JSP page that adds numbers from one to ten, and then print the output.

```jsp
<% int sum = 0;
    for(j = 0; j < 10; j++) { %>
        // XXX --- Add j to sum
        <% } %>
    // YYY --- Display the sum
```

10) Which statement when placed at the location XXX can be used to compute the sum. Select the one syntactically correct statement

- A. `<% sum = sum + j %>`
- B. `<% sum = sum + j; %>`
- C. `<%= sum = sum + j %>`
- D. `<%= sum = sum + j; %>`

11) What must be added at the location YYY to print the sum of ten numbers. Select the one syntactically correct statement

- A. `<% sum %>`
- B. `<% sum; %>`
- C. `<%= sum %>`
- D. `<%= sum; %>`

12) What does MVC stand for?

```
_________  ___________  ___________
```

13) Turning off auto-commit and having transactions that are explicitly committed is useful when working with a database. Why?

- A. It simplifies insertion statements
- B. It allows multiple operations to behave as an atomic operation
- C. It makes individual insertions significantly more efficient
- D. Transactions that invoke the commit operation always succeed

14) Something interesting you’ve learned in this class:

```
________________________________________
```

```
________________________________________
```
Consider a web application www.testmult.com/index.jsp where elementary school students test their knowledge of multiplication.

There are two counters, one counting the number of correct and one counting the false answers that a student has given in the current session. In particular, the first time a student visits the web application page he/she will see the first page (below), which prompts him/her to submit the result of the multiplication. The two numbers that are multiplied are randomly generated from the set of numbers from 10 to 99. If the student submits a correct answer then the next page will be shown (where the counter of correct answers has been incremented) and the session will continue in the same fashion. If the student submits a false answer then the “false” counter will be increased.

![First page of the multiplication test](image1.png)

![Second page of the multiplication test](image2.png)
Write the JSP “index.jsp” that provides the above functionality. Note that the two counters are session level ones. You do not need to do any form validation.

------------- index.jsp --------------
**Struts**

Now consider the following modification to the multiplication web site. There are two database tables:

- `easy_multiplications(easyID, num1, num2)` contains pairs of numbers `num1, num2` that are easy to multiply (e.g. `2 * 2`).

- `hard_multiplications(hardID, num1, num2)` contains pairs of numbers `num1, num2` that are hard to multiply (e.g. `327 * 936`)

When the student gives the URL `http://www.testmult.com/` he/she sees the following page, where the first pair of numbers “`num1, num2`” from `easy_multiplications` are displayed and the student is asked to submit the result of multiplying `num1, num2`. There are four counters, which count the successes and failures of the student: counter of correct hard, counter of correct easy, counter of wrong hard and counter of wrong easy, as shown in the following screenshot.

![Image of a multiplication test example](image)

If the student succeeds in answering a question (easy or difficult) he/she is directed to a page, such as the following, where hard questions are asked. The questions are drawn from the database, in increasing number of `easyID` or `hardID`. 
Build a struts-config.xml that provides the MVC model for the test multiplication web site and provide
- the `struts-config.xml`
- the JSP code of the “hard” page,
- the controller and form beans for the “hard” page.

Below we provide the properties and functions of the model bean. You do not need to do any form validation.

```java
package edu.ucsd.cse.exam.Beans;

public class Exams {
    Result tupleWithNum1Num2;
    /*
     * the tuple has two attributes - num1 and num2
     */

    int currentEasyID, currentHardID;
    int easyCorrect, easyWrong, hardCorrect, hardWrong;

    public Result getTupleWithNum1Num2 (int easyOrHard) {
        /*
        * provides next pair of easy or hard num1 and num2.
        * if easy then returns the Result tuple with the currentEasyID
        * and increments the currentEasyID by 1.
        * if hard then returns the Result tuple with the currentHardID
        * and increments the currentHardID by 1.
        */
    }

    public void submitEasyAnswer (int num1,
                                    int num2, int submittedResult) {
        if (num1 * num2 == submittedResult) { easyCorrect++ ; }
        else { easyWrong++ ; }
    }

    public void submitHardAnswer (int num1,
                                    int num2, int submittedResult) {
        if (num1 * num2 == submittedResult) { hardCorrect++ ; }
        else { hardWrong++ ; }
    }

    public int getEasyCorrect() { return easyCorrect ; }
    public int getHardCorrect() { return hardCorrect ; }
    public int getEasyWrong() { return easyWrong ; }
    public int getHardWrong() { return hardWrong ; }

    public Exams() {
        easyCorrect = 0 ;
        easyWrong = 0 ;
        hardCorrect = 0 ;
        hardWrong = 0 ;
    }
}
```