OLD QUESTIONS PART 1

```
public class IncreaseDiscountServlet extends HttpServlet {

      protected void doPost(HttpServletRequest request,
                            HttpServletResponse response)
          throws ServletException, IOException {
A:          UserProfile user = getUserProfile(request.getParameter("user"));
B:          int discount = user.getDiscount();
C:          ++discount;
D:          user.setDiscount(discount);
            ...
      }

      ...
}
```

Figure 2. Servlet with concurrency bug. The purpose of this servlet is to increase by 1 the level of discount that a customer receives. Note the statements labeled A–D.

6. [4pts] The servlet code in Figure 2 contains a concurrency-related defect that can cause discount additions to be lost. Fill in the blanks below to describe a step-by-step scenario, involving two threads T1 and T2, that illustrates how a discount increase can be lost. (The first step has been filled in for you.)

- First, thread __T1__ executes line __A___
- Next, thread __T1__ executes line __B__
- Next, thread __T1__ executes line __C__
- Next, thread __T2__ executes line __A__
- Next, thread __T2__ executes line __B__
- Next, thread __T2__ executes line __C__
- Next, thread __T2__ executes line __D__
- Finally, thread __T1__ executes line __D__

7. [2pts] True or false? The above concurrency error could be corrected by making the UserProfile follow the monitor pattern (e.g., by making all its methods `synchronized`).

a. True
[✓ b.] False

8. [2pts] Imagine that you just joined a development team that uses svn for version control and collaboration. To start contributing to the project, what svn operation would you most likely invoke first?

   [✓ a.] checkout

   b. commit

   c. export

   d. revert

   e. update

9. [2pts] Now, imagine that you have a working copy, but other team members have pushed changes into the repository since you created the working copy. What svn operation would you use to pull those changes into your working copy?

   a. checkout

   b. commit

   c. export

   d. revert

   [✓ e.] update

10. [2pts] Imagine that you try to commit your changes into the repository, but you get an out-of-date error. What would your next svn operation most likely be?

   a. checkout

   b. commit

   c. export

   d. revert

   [✓ e.] update

Artist

ArtistID	ArtistName
21233	Pixies
55621	Cure
10311	Smiths

ArtistAlbum

ArtistID	AlbumID
21233	75531
10311	00598
21233	31300

Album

AlbumID	AlbumTitle	AlbumYear
75531	Surfer Rosa	1988
00598	Meat Is Murder	1984
31300	Come On Pilgrim	1987

Figure 3. Database tables for a music catalog. All columns in ArtistAlbum are foreign keys.

11. [6pts] Consider the database tables in Figure 3. Fill the table below to match the result returned by the following query. Cross out any cells in the table that you do not need. Don't forget to label the columns.

```
SELECT ArtistName, AlbumTitle, AlbumYear FROM
(Artist INNER JOIN ArtistAlbum
 ON Artist.ArtistID = ArtistAlbum.ArtistID)
INNER JOIN Album ON ArtistAlbum.AlbumID = Album.AlbumID
ORDER BY ArtistName, AlbumYear, AlbumTitle;
```

ArtistName	AlbumTitle	AlbumYear	
Pixies	Come On Pilgrim	1987	
Pixies	Surfer Rosa	1988	
Smiths	Meat Is Murder	1984	

13. [3pts] Fill in each blank with either *GET* or *POST*.

- GET requests must be idempotent.
- GET requests have no data payload.
- Clicking a hyperlink in a webpage typically produces a GET request.

14. [2pts] Which one of these mechanisms is most often used to handle HTTP sessions.

  a. Bookmarks
  b. Browser history
  [✓ c.] Cookies
  d. HTTP WRITE method
  e. ServletConfig object

15. [3pts] For each of the following types of artifacts, tell which role in an MVC architecture the artifact would typically play. Give the full name of the role. (Giving only the letter M, V, or C will earn you only partial credit.)

- Plain old Java class: Model
- Servlet class: Controller
- JSP: View

For questions 16 and 17, imagine you are developing a web app for an on-line DVD rental store. Each of the questions asks you to implement a feature. In your solutions, use the classes defined in the Appendix (especially the domain ones; assume they are already implemented).

16. [15pts] For this question, your task is to implement a feature that displays the contents of a user's shopping cart. Assume that as the user browses other DVD-store pages, she adds videos to her shopping cart. You must implement either one servlet or one JSP (your choice) that takes an HTTP GET request and displays the contents of the user's shopping cart as shown in Figure 4. Assume that each session object has an attribute "cart" that refers to a ShoppingCart object. There is additional space on the next page for your answer.

JSP Solution:

```
<!DOCTYPE html>
<html>
<body>
<h1> Shopping Cart</h1>
<ul>
<% ShoppingCart cart =
        (ShoppingCart)session.getAttribute("cart");
    int total = 0;
    For (int i=0; i< cart.size(); ++i) {
        Movie m = cart.get(i);
        total += m.getPrice();
%>
<li><%= m.getTitle() %>(<%= m.getYear() %>) -
    $<%= m.getPrice() %>
<% } %>
Total: $<%= total %>
```

```
</body>
</html>
```

17. [15pts] For this question, your task is to implement one servlet and one JSP that work together (as in MVC) to compute and display recommendations for movies that the user might like. Your servlet must take an HTTP POST request, which includes a "targetMovieID" parameter, and must display a list of recommendations for movies similar to the target movie (using the MovieRecommender class). The recommendations must be formatted as in Figure 5. Your code need not display the form that produced the POST request (assume some other servlet or JSP does that). There is additional space on the next page for your answer.

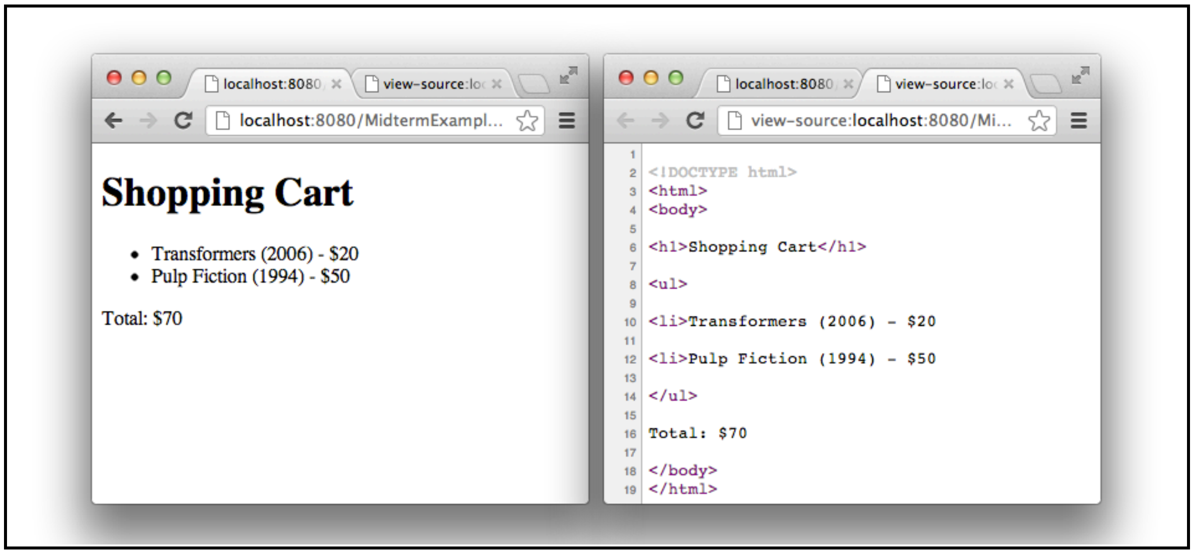
```
public class RecommendMoviesServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse resp) {
        String target = request.getParameter("targetMovieID");
        Vector<Movie> recs = MovieRecommender.recommendSimilar(target);
        request.setAttribute("recs", recs);
        RequestDispatcher view =
            request.getRequestDispatcher("recommend.jsp");
        view.forward(request, resp);
    }
}
```

recommend.jsp

```
<!DOCTYPE html>
<html>
<body>
<h1>Recommendations</h1>
```

```
<%  Vector<Movie> recs =
          (Vector<Movie>)requert.getAttribute("recs");
      For(int i=0;  i < recs.size(); ++i) {
          Movie m = recs.get(i);
%>
Movie: <%= m.getTitle() %> <br>
Director: <%= m.getDirector() %> <br>
Length: <%= m.getLength() %> minuter <br><br>
<%  }  %>
</body>
</html>
```

Appendix


localhost:8080
view-source:lo
localhost:8080/MidtermExampl...
view-source:localhost:8080/Mi...

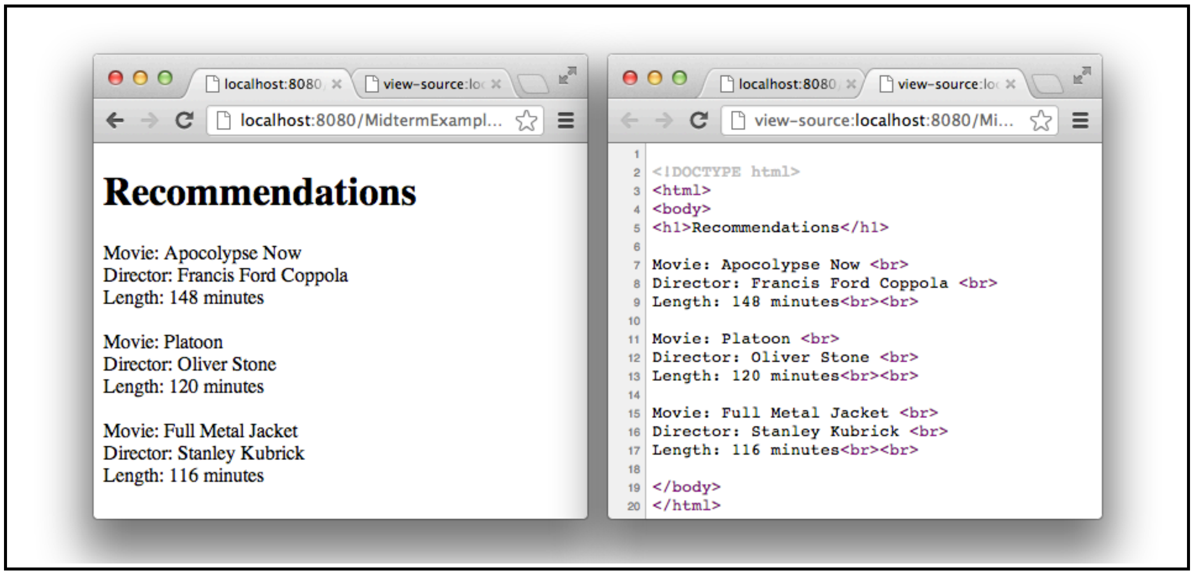
Shopping Cart

- Transformers (2006) - $20
- Pulp Fiction (1994) - $50

Total: $70

```html

<!DOCTYPE html>
<html>
<body>

<h1>Shopping Cart</h1>

<ul>

<li>Transformers (2006) - $20

<li>Pulp Fiction (1994) - $50

</ul>

Total: $70

</body>
</html>
```

Figure 4. Sample output of a display shopping cart feature. On the right is the raw HTML.


localhost:8080
view-source:lo
localhost:8080/MidtermExampl...
view-source:localhost:8080/Mi...

Recommendations

Movie: Apocolypse Now
Director: Francis Ford Coppola
Length: 148 minutes

Movie: Platoon
Director: Oliver Stone
Length: 120 minutes

Movie: Full Metal Jacket
Director: Stanley Kubrick
Length: 116 minutes

```html

<!DOCTYPE html>
<html>
<body>
<h1>Recommendations</h1>

Movie: Apocolypse Now <br>
Director: Francis Ford Coppola <br>
Length: 148 minutes<br><br>

Movie: Platoon <br>
Director: Oliver Stone <br>
Length: 120 minutes<br><br>

Movie: Full Metal Jacket <br>
Director: Stanley Kubrick <br>
Length: 116 minutes<br><br>

</body>
</html>
```

Figure 5. Sample output of a movie-recommendation feature. On the right is the raw HTML.

Java API Excerpts

Class HttpServlet

- protected void doGet(HttpServletRequest request, HttpServletResponse response)
- protected void doPost(HttpServletRequest request, HttpServletResponse response)

Class HttpServletRequest

- String getParameter(String name)
- RequestDispatcher getRequestDispatcher(String path)

Class RequestDispatcher

- void forward(ServletRequest request, ServletResponse response)

Interface HttpSession

- Object getAttribute(String name)
- void setAttribute(String name, Object value)
- boolean isNew()

Class Vector<E>

- public int size()
    - Returns the number of components in this vector.
- public E get(int index)
    - Returns the element at the specified position in this Vector.

Domain Classes

Class Movie

- public String getTitle()
- public int getLength()
    - Returns number of minutes.
- public String getDirector()
- public int getPrice()
    - Returns dollars (no cents).
- public int getYear()

Class MovieRecommender (Note the method is static.)

- public static Vector<Movie> recommendSimilar(String targetMovieID)
    - Given a movie ID, returns a list of recommended movies.

Class ShoppingCart

- public int size()
    - Returns the number of movies in the cart.
- public Movie get(int i)
    - Returns the i+1th movie in the cart (i.e., starts at 0 like array indices).

OLD QUESTIONS PART 2

For each of the following, circle the answer that best answers the question.

1. ~~[3pts] Which of the following is not a problem in software engineering?~~
    a. ~~Gathering complete and valid requirements~~
    b. ~~Estimating the cost of a software project~~
    c. ~~Removing defects from software~~
    d. ~~Maintaining software over time~~
    [✓ e.] ~~None of the above (i.e., all are problems)~~

2. ~~[3pts] Which of the following is not an agile value?~~
    a. ~~Individuals and interactions over processes and tools~~
    b. ~~Working software over comprehensive documentation~~
    c. ~~Customer collaboration over contract negotiation~~
    d. ~~Responding to change over following a plan~~
    [✓ e.] ~~None of the above (i.e., all are agile values)~~

3. ~~[3pts] Which of the following problems does iterative development directly address?~~
    [✓ a.] ~~Unstable requirements~~
    b. ~~Design erosion~~
    c. ~~Program comprehension~~
    d. ~~All of the above~~
    e. ~~None of the above~~

4. [3pts] Which of the following is not true about GET and POST?
    a. GET is generally idempotent, and POST is generally not
    b. POST has a data payload, and GET does not
    c. Both GET and POST requests have request headers
    [✓ d.] GET is an HTTP method, and POST is an HTTPS method
    e. None of the above

5. [3pts] Which of the following correctly defines what it means to be an idempotent operation?
    a. Has side effects
    b. Has one parameter
    c. Can only be applied once per session
    [✓ d.] Can be applied multiple times without changing the result beyond the initial application
    e. None of the above

7. [5pts] For the following URL, match the part of the URL with the name for that part.

$$\underbrace{\text{http:/}}_{\text{(i)}}\text{/}\underbrace{\text{www.wickedlysmart.com}}_{\text{(ii)}}\underbrace{\text{:80}}_{\text{(iii)}}\underbrace{\text{/beeradvice/select/beer1.html}}_{\text{(iv)}}\underbrace{\text{?color=dark\&taste=malty}}_{\text{(v)}}$$

i.	b	a.	Resource (with path)
ii.	e	b.	Protocol
iii.	c	c.	Port
iv.	a	d.	Parameters
v.	d	e.	Server

8. [3pts] Which SVN client operation creates a working copy?
   [✓ a.] checkout
   b. commit
   c. update
   d. revert
   e. None of the above

9. [3pts] Which SVN client operation may report a conflict and then do nothing?
   a. checkout
   [✓ b.] commit
   c. update
   d. revert
   e. None of the above

10. [3pts] Which SVN client operation should you run as the first step toward resolving a conflict, assuming your goal is to do a merge?
    a. checkout
    b. commit
    [✓ c.] update
    d. revert
    e. None of the above

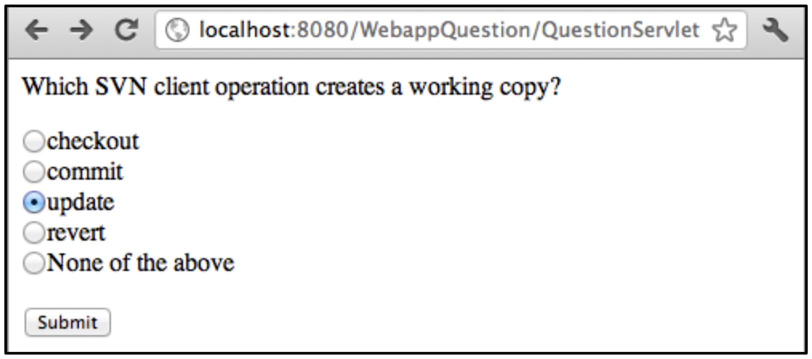
11. [6pts] For each of the following artifacts, tell which role in an MVC architecture the artifact would typically play. Give the full name of the role. (Giving only the letter M, V, or C will earn you only partial credit.)

    a. A JSP ____View____

    b. A plain old Java class ____Model____

    c. A servlet class ____Controller____

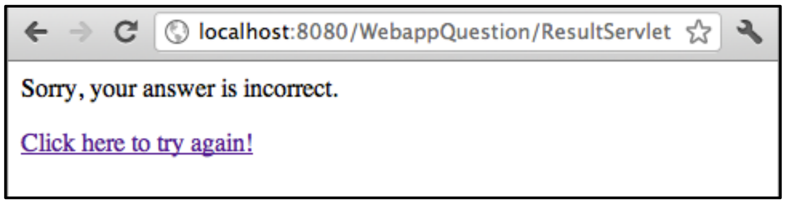
12. [15pts] For this problem, you must create a servlet for a quizzing web app. The app already has a QuestionServlet servlet that responds to HTTP GET requests with a trivia question that looks like this:


localhost:8080/WebappQuestion/QuestionServlet

Which SVN client operation creates a working copy?

- ○ checkout
- ○ commit
- ● update
- ○ revert
- ○ None of the above

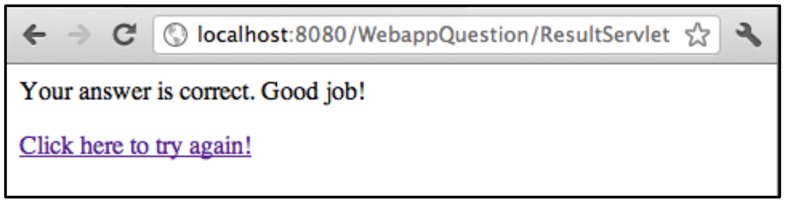
Your job is to create a ResultServlet servlet that receives the user's answer when he/she presses the submit button, and that responds with a message like this:


localhost:8080/WebappQuestion/ResultServlet

Sorry, your answer is incorrect.

Click here to try again!

or like this:


localhost:8080/WebappQuestion/ResultServlet

Your answer is correct. Good job!

Click here to try again!

Relevant excerpts from the Java API and the HTML that QuestionServlet generated for the above trivia question is given on page 11. Your ResultServlet must use the QuestionManager class (described on page 12) to retrieve the correct answers to questions.

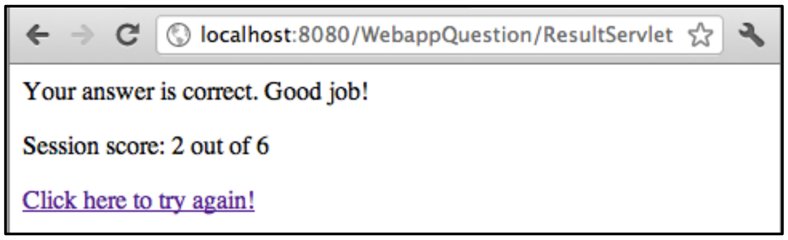
You need not include `import` statements or the web.xml file in your solution. Just write the Java code for the ResultServlet class. Also, I do <u>not</u> expect you to use JSPs in your solution.

Write your answer on the next page.

Write your answer to Question 12 on this page.

```
public class ResultServlet extends HttpServlet {

  protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {
    Enumeration<String> params = request.getParameterNames();
    String questId = params.nextElement();
    String userAns = request.getParameter(questId);
    String correctAns = QuestionManager.getInstance().getAnswerId(questId);
    PrintWriter out = response.getWriter();

    out.println("<html>");
    out.println("<body>");
    out.println("<p>");

    if (userAns.equals(correctAns)) {
      out.println("Your answer is correct. Good job!");
    } else {
      out.println("Sorry, your answer is incorrect.");
    }

    out.println("</p>");
    out.println("<p>");
    out.println("<a href=\"QuestionServlet\">Click here to try again!</a>");
    out.println("</p>");
    out.println("</body>");
    out.println("</html>");
  }
}
```

13. [15pts] For this problem, you must enhance your previous servlet by making it keep track of the user's score for the current session:



Your answer is correct. Good job!

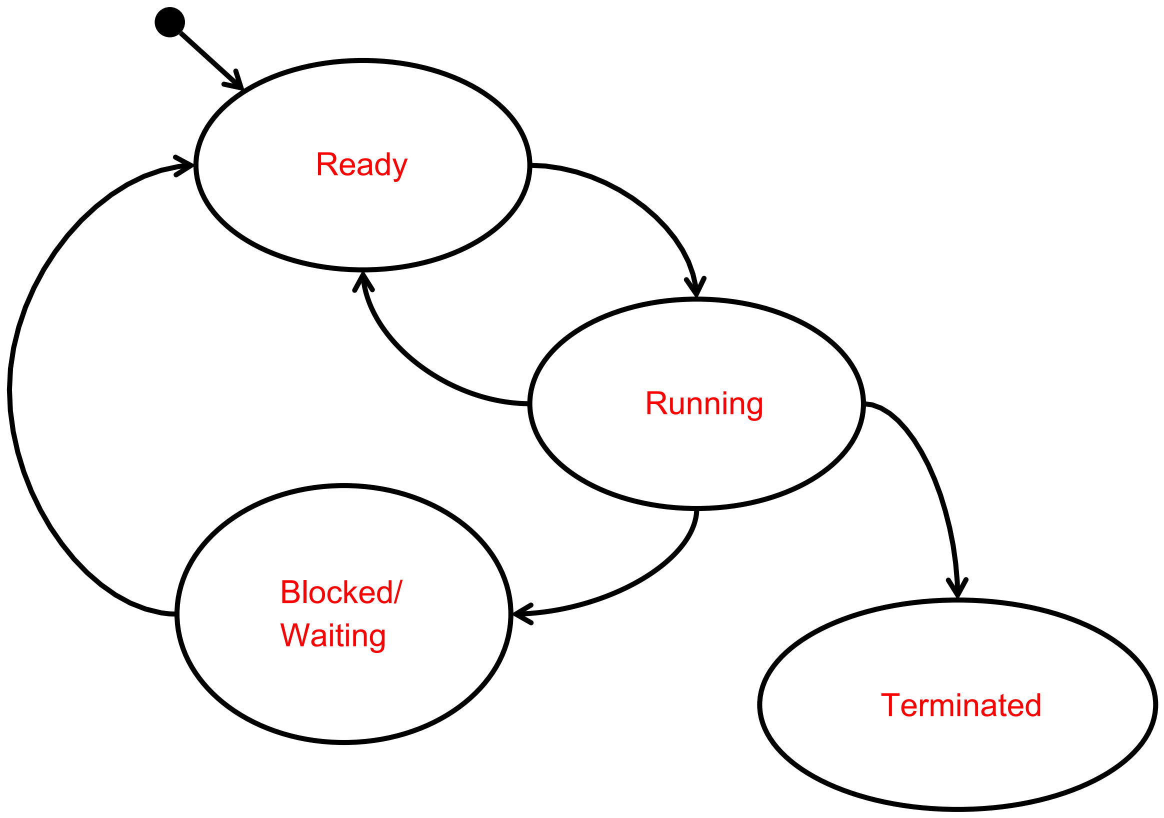
Session score: 2 out of 6

Click here to try again!

You need not rewrite all the code from your previous solution. Represent code that has not changed with ellipses (…). As before, I do not expect you to use JSPs.

```
public class ResultServlet extends HttpServlet {

  protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {
    ...
    HttpSession session = request.getSession();

    if (session.isNew()) {
      session.setAttribute("score", new Integer(0));
      session.setAttribute("attempts", new Integer(1));
    } else {
      session.setAttribute(
        "attempts",
        new Integer(((Integer)session.getAttribute("attempts")).intValue()+1)
      );
    }
    ...
    if (userAns.equals(correctAns)) {
      ...
      session.setAttribute(
        "score",
        new Integer(((Integer)session.getAttribute("score")).intValue()+1)
      );
    } else {
      ...
    }
    ...
    out.println("<p>");
    out.println("Session score: " +
                ((Integer)session.getAttribute("score")).toString() +
                " out of " +
                ((Integer)session.getAttribute("attempts")).toString());
    out.println("</p>");
    ...
  }
}
```

14. [8pts] Complete this diagram of thread states in Java by labeling the states.



15. [15pts] List all the different output values that this program might produce.

```
public class MyThread implements Runnable {

    public void run() {
        MyMainThread.x += MyMainThread.y;   // A
        MyMainThread.y = 5;                 // B
    }

}
```

```
public class MyMainThread {
    static public int x = 1;
    static public int y = 1;

    public static void main(String[] args)
        throws InterruptedException
    {
        Thread myT = new Thread(new MyThread());
        myT.start();
        x = 9;     // C
        y += x;    // D
        myT.join();
        System.out.println("x = " + x);
        System.out.println("y = " + y);
    }

}
```

ABCD: x = 9, y = 14
ACBD: x = 9, y = 14
ACDB: x = 9, y = 5
CABD: x = 10, y = 15
CADB: x = 10, y = 5
CDAB: x = 19, y = 5

Relevant Excerpts from the Java API

- Class HttpServlet
  - protected void service(HttpServletRequest req, HttpServletResponse resp)
  - protected void doGet(HttpServletRequest req, HttpServletResponse resp)
  - protected void doPost(HttpServletRequest req, HttpServletResponse resp)
- Interface HttpServletRequest
  - HttpSession getSession()
  - public Enumeration<String> getParameterNames()
  - public String getParameter(String name)
- Interface HttpServletResponse
  - public PrintWriter getWriter()
- Interface HttpSession
  - public boolean isNew()
  - public Object getAttribute(String name)
  - public void setAttribute(String name, Object value)
- Interface Enumeration<E>
  - boolean hasMoreElements()
  - E nextElement()

Example of HTML Generated by QuestionServlet

```
<html>
<body>
<p>
Which SVN client operation creates a working copy?
</p>
<form method="POST" action="ResultServlet">
<input type="radio" name="q1748" value="a1">checkout<br>
<input type="radio" name="q1748" value="a4">commit<br>
<input type="radio" name="q1748" value="a2">update<br>
<input type="radio" name="q1748" value="a3">revert<br>
<input type="radio" name="q1748" value="a5">None of the above
<p>
<input type="submit" value="Submit">
</p>
</form>
</body>
</html>
```

q1748 is the ID used by QuestionManager for the question. a1–a5 are IDs that QuestionMaker uses for each answer.

Class QuestionManager

```
public class QuestionManager {
    private static QuestionManager instance = null;

    public static QuestionManager getInstance() {
        if (instance == null) {
            instance = new QuestionManager();
        }
        return instance;
    }

    private QuestionManager() { ... }

    /**
     * @return The ID of the correct answer to questionId.
     */
    public String getAnswerId(String questionId) { ... }

    ...

}
```

QuestionManager follows the singleton pattern, which ensures that there is only one instance of the class in the system. To get the instance of QuestionManager, you must call its static getInstance() method (as opposed to using `new`).

OLD QUESTIONS PART 3

1. [10pts] Write HTML would create web page depicted in Figure 1. Your solution must include the following types of HTML elements (and no other types): **!DOCTYPE**, **a** (with **href** attribute), **body**, **h1**, **head**, **html**, **img** (with **src** attribute), **li**, **p**, **title**, **ul**. The link should go to http://acm.org/. Assume the image is in **WebContent/images/kitty.jpg**.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>My Web Page</title>
</head>
<body>

<h1>My Heading (Level 1)</h1>

<p>My paragraph.</p>

<p>Paragraph with <a href="http://acm.org">link to ACM</a>.</p>

<img src="resources/kitty.jpg">

<ul>
<li>Item 1
<li>Item 2
</ul>

</body>
</html>
```

2. [10pts] Consider the following scenario involving Subversion (SVN). Alice and Bob are both working on a shared project **MyProj** that is stored in an SVN repository. Bob does a **checkout** on the project. What does SVN do when Bob issues the **checkout** command?

When Bob issues the **checkout** command, svn creates a working copy of the **MyProj** files on Bob's machine.

Next, Bob edits the **MyProj** file **Foo.java**. Then, he does a **commit**. What does SVN do when Bob issues the **commit** command?

When Bob issues the **commit** command, svn creates a new revision of the **MyProj** files with Bob's edits in the repository.

Next, Alice does a **checkout** on **MyProj**. Then, Alice and Bob both edit **Foo.java** in parallel. Foo.java has over 100 lines of code. Alice edits a couple lines at the top of the file, and Bob edits a couple lines at the bottom of the file. Then, Bob does a **commit**. Finally, Alice does a **commit**. What does SVN do when Alice issues the **commit** command?

When Alice issues the **commit** command, svn responds with an out-of-date error and does nothing further.

What SVN command should Alice issue next and what would the result of the command be?

Alice should issue the **update** command. The result of the command will be that svn inserts Bob's edits into her working copy of **Foo.java**.

After the previous command, what command should Alice issue?

Alice should issue the **commit** command.

Imagine that the University of Memphis provides students with snacks (chips and a drink) during class and has a web app that students can use to order their snacks. Students go to a web page that presents them with the form depicted in Figure 2. By filling out and submitting the form, students can choose what flavor of drink and type of chips they will receive. After a student submits the form, he/she is presented with a page like that depicted in Figure 5, or in the event of an error, the error page in Figure 6.

3. [5pts] Below is a list of the components that make up this web app. For each one, tell what part of an MVC architecture it belongs to. Put the full name of the part (i.e., do not just put "V").

_____View_______ An HTML page with the form in Figure 2

_____View_______ A JSP for the order-summary page (see Figure 5)

_____View_______ A JSP for the order-error page (see Figure 6)

_____Model______ A plain old Java class that represents a snack order (see Figure 3)

_____Model______ A plain old Java class for storing/retrieving snack order records (see Figure 4)

__Controller___ A servlet class that receives requests from the Figure 2 form, uses the plain old Java classes (Figure 3 and Figure 4) to service the request, and responds using the JSPs (Figure 5 and Figure 6).

4. [15pts] On the next page, reverse engineer the servlet class for ordering a snack. That is, write Java code that implements the servlet.

   - The order-summary JSP assumes that the request contains an attribute with the name "order" that maps to a **SnackOrder** object that contains the order info.
   - Note the JSP paths in Figure 5 and Figure 6.
   - Note the Java API excerpts in Figure 7.

Write your answer to question 4 here.

```
@WebServlet("/orderSnack.do")
public class OrderSnackServlet extends HttpServlet {
     private static final long serialVersionUID = 1L;

     protected  void  doPost(HttpServletRequest  request,  HttpS-
ervletResponse response) throws ServletException, IOException {
          String drink = "";
          String chips = "";
          drink = request.getParameter("drink");
          chips = request.getParameter("chips");
          SnackOrder order = new SnackOrder();
          order.setDrink(drink);
          order.setChips(chips);
          request.setAttribute("order", order);
          SnackOrderDao dao = new SnackOrderDao();
          if (dao.insertOrder(order) == -1) {
               request.getRequestDispatcher("WEB-
INF/orderError.jsp").forward(request, response);
          } else {
               request.getRequestDispatcher("WEB-
INF/orderSummary.jsp").forward(request, response);
          }
     }

}
```

5. [8pts] Reverse engineer order-summary JSP (Figure 5). As in the previous question, assume that the request contains an attribute with the name "order" that maps to a **SnackOrder** object that contains the order info.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" import="com.snack.*" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Order Summary</title>
</head>
<body>

<h1>Your Order</h1>

<ul>
<li>Drink: <%= ((SnackOrder) request.getAttribute("order")).getDrink() %>
<li>Chips: <%= ((SnackOrder) request.getAttribute("order")).getChips() %>
</ul>

</body>
</html>
```

E-commerce sites, like Amazon.com, commonly allow users to add items to a "shopping cart" while they browse. Then, when they've selected all the items they want, they can "check out" and purchase the items in their cart.

Consider the **Cart** class excerpt in Figure 9 that implements shopping-cart functionality and the servlet **doGet** method in Figure 10 that removes the most expensive item from the cart.

6. [6pts] Based on the above code, describe a scenario in which a call to **doGet** removes an item from the cart that is not the most expensive item. (Hint: concurrency.) Make sure that your answer is thorough and concise.

Assume there are two threads T1 and T2, each of which is about to execute the **doGet** method. Also assume that there are three items in the cart. The item at index 0 is most expensive; the item at index 1 is least expensive; and the item at index 2 has a price that falls in between the other two items.

T1 executes first. It completes the call to **findMostExpensive**, which returns the index 0. But then a context switch occurs.

T2 executes next. It completes the entire call to **doGet**. Since the item at index 0 is still the most expensive, T2 removes that item from the cart. As a result, the other items slide forward, so now the item at index 0 is the least expensive and the item at index 1 is most expensive.

Another context switch occurs, and T1 starts executing again. Since it previously got the index 0 from the **findMostExpensive** call, it proceeds with removing the item at index 0. Of course, due to T2's actions, the item at index 0 is no longer the most expensive.

7. [6pts] Rewrite the **doGet** method from Figure 10 to correct the error. I've started the method for you:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ... {
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ... {
       HttpSession session = request.getSession();
       synchronized (session) {
              ...
              Cart c = (Cart) session.getAttribute("cart");
              int x = c.findMostExpensive();
              if (x != -1) { c.removeItem(x); }
       }
       ...
}
```

8. [10pts] Given the TVGuide database in Figure 8, what would the result of the following query be?

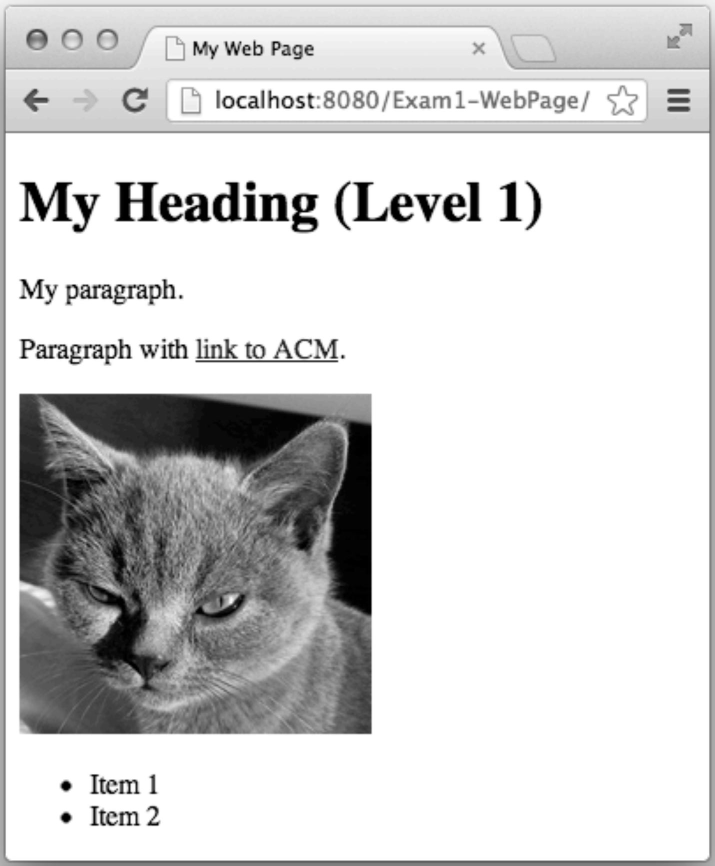
```
SELECT
    `LastName`, `NetworkName`
FROM
    (`TVNetwork` INNER JOIN `TVShowNetwork`
     ON `TVNetwork`.`NetworkID` = `TVShowNetwork`.`NetworkID`)
INNER JOIN
    (`Talent` INNER JOIN `TVShowCreator`
     ON `Talent`.`TalentID` = `TVShowCreator`.`TalentID`)
ON `TVShowNetwork`.`ShowID` = `TVShowCreator`.`ShowID`
ORDER BY `NetworkName`, `LastName`;
```

Fill the table below with your answer. Cross out any cells in the table that you do not need. Don't forget to label the columns.

LastName	NetworkName		
Groening	Comedy Central		
Groening	Fox		
Whedon	Fox		

Name: ______________________________________

Figures



My Heading (Level 1)

My paragraph.

Paragraph with link to ACM.

- Item 1
- Item 2

Figure 1. Example web page.

```
<form method="post" action="orderSnack.do">
ORDER SNACK HERE
<br>
Drink flavor: <input type="text" name="drink">
<br>
Type of chips: <input type="text" name="chips">
<br>
<input type="submit" value="Submit">
</form>
```

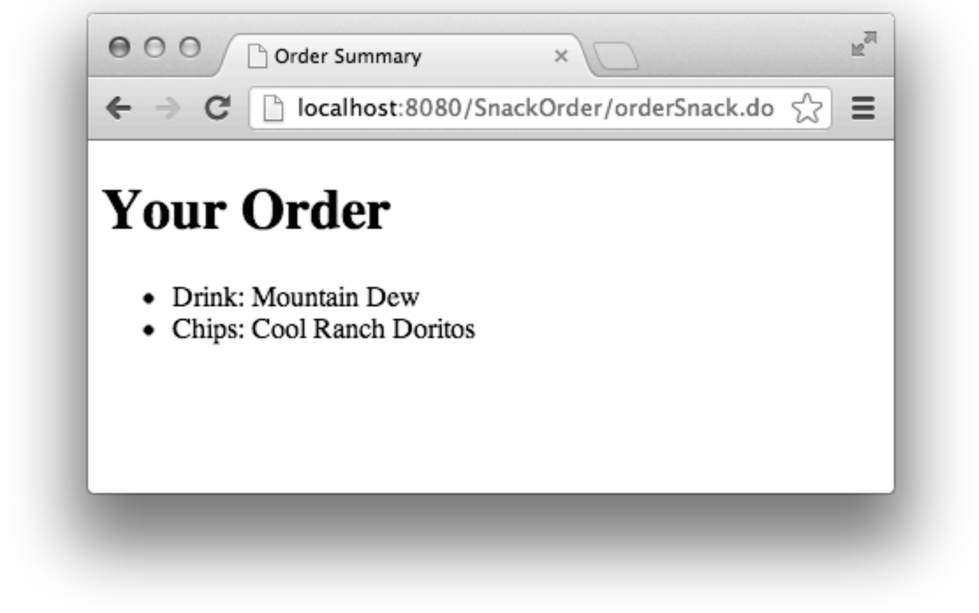
Figure 2. Snack-order form.

```
public class SnackOrder {
      private String drink;
      private String chips;
      public void setDrink(String s) { drink = s; }
      public void setChips(String s) { chips = s; }
      public String getDrink() { return drink; }
      public String getChips() { return chips; }
}
```

Figure 3. Java class that represents a snack order.

```
public class SnackOrderDao {
      /**
       * Returns newly created order number or -1 on error.
       */
      public int insertOrder(SnackOrder order) {
             ...
      }

      ...
}
```

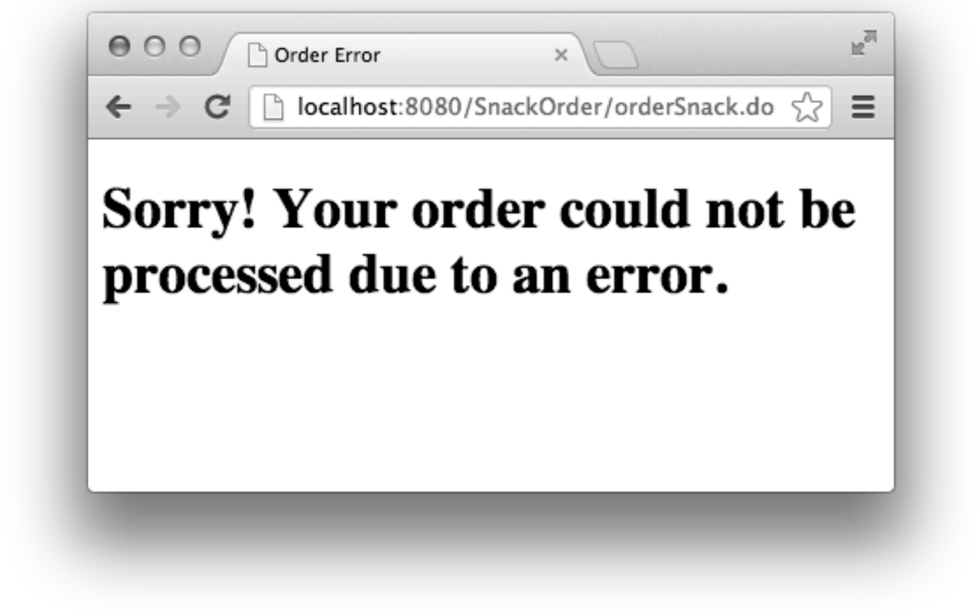
Figure 4. Java class for storing/retrieving snack-order records.



Your Order

- Drink: Mountain Dew
- Chips: Cool Ranch Doritos

Figure 5. WEB-INF/orderSummary.jsp



Sorry! Your order could not be processed due to an error.

Figure 6. WEB-INF/orderError.jsp

- Class **HttpServlet**
  - Annotation to declare URL pattern: **@WebServlet**
  - protected void **doGet**(HttpServletRequest req, HttpServletResponse resp)
  - protected void **doPost**(HttpServletRequest req, HttpServletResponse resp)
  - protected void **service**(HttpServletRequest req, HttpServletResponse resp)
- Interface **HttpServletRequest**
  - Object **getAttribute**(String name)
  - String **getParameter**(String name)
  - RequestDispatcher **getRequestDispatcher**(String path)
  - HttpSession **getSession**()
  - void **setAttribute**(String name, Object o)
- Interface **HttpServletResponse**
  - PrintWriter **getWriter**()
- Interface **RequestDispatcher**
  - void **forward**(ServletRequest request, ServletResponse response)
- Class **PrintWriter**
  - void **print**(String s)
  - void **println**(String x)
- Interface **HttpSession**
  - Object **getAttribute**(String name)
  - boolean **isNew**()
  - void **setAttribute**(String name, Object value)

Figure 7. Java API excerpts.

TVNetwork

NetworkID	**NetworkName**
111	CBS
666	Comedy Central
999	Fox

TVShow

ShowID	**ShowName**	**Seasons**
2222	The Simpsons	24
5555	Futurama	7
8888	Firefly	1

TVShowNetwork

ShowID	**NetworkID**
2222	999
5555	666
8888	999

Talent

TalentID	**LastName**	**FirstName**
333333	Groening	Matt
777777	Whedon	Joss

TVShowCreator

ShowID	**TalentID**
2222	333333
5555	333333
8888	777777

Figure 8. TVGuide database.

Class **Cart**

- private Item[] **items**
    - Array of items in the cart. Not sorted in any particular way.
- public int **findMostExpensive**()
    - Returns the index of the most expensive item in the cart, or -1 if the cart is empty.
- public void **removeItem**(int i)
    - Removes the item at index i from the cart.

Figure 9. Cart class excerpt.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ... {
       HttpSession session = request.getSession();
       ...
       Cart c = (Cart) session.getAttribute("cart");
       int x = c.findMostExpensive();
       if (x != -1) { c.removeItem(x); }
       ...
}
```

Figure 10. Servlet doGet method that removes the most expensive item from the cart.