Question 1. (6 points) Complete the definition of method `printTriangle(n)` below so that when it is executed it will print on `System.out` an upside down triangle of `n` rows, with `n "+"` characters in the first row, `n`-1 in the second, and so forth, finishing with one `"+"` character in the last row.

For example, execution of `printTriangle(4)` should print the following pattern

```
++++
+++
++
+
```

```
/**
 * Print an upside down triangle made up of + characters
 * @param n  the number of rows to print and the number of +
 *           characters to print in the first row
 */
public void printTriangle(int n) {

   for (int r = 1; r <= n; r++) {

      for (int c = 1; c <= n-r+1; c++) {

         System.out.print("+");

      }

      System.out.println();

   }

}
```

Another, actually simpler, solution is to count down towards 0.

```
   for (int r = n; r > 0; r--) {

      for (int c = r; c > 0; c--) {

         System.out.print("+");

      }

      System.out.println();

   }
```

Other correct solutions, including those that use while loops, also received full credit.

Question 2. (2 points) In Java we can define *interfaces* as well as classes. Here is an example of an interface definition.

```
/** Simple interface for test question */
public interface I {
   public void one(int n, double x);
   public String two();
}
```

What is required when a simple class `C` *implements* interface `I`? Circle the number of the correct answer.

(i) `C` must implement all of the methods declared in `I` and no others.

(ii) `C` must implement some, but not necessarily all, of the methods declared in `I` and no others.

[✓ (iii)] `C` must implement all of the methods declared in `I` and may also contain additional methods not declared in `I`

(iv) `C` must implement some, but not necessarily all, of the methods declared in `I` and may also contain additional methods not declared in `I`

(v) None of the above are correct

Question 3. (3 points) Consider the following short class definition.

```
public class Planet {
  private int orbit;
  public Planet(int index) {
    orbit = index;
  }
  public static void main(String[] arg) {
    Planet rock = new Planet(3);
    System.out.println(rock.getOrbit());
  }
  public int getOrbit() {
    return orbit;
  }
}
```

(a) How many instance variables are defined in class Planet? **1**

(b) How many methods (not constructors) are defined in class Planet? **2**

(c) If you type "java Planet" in the DrJava interactions pane after compiling this class definition, which one of the following choices is printed out? Circle the correct number.

(i) Error: No 'main' method in 'Planet'
[✓ (ii)] 3
(iii) Planet@15718f2
(iv) java.lang.NullPointerException:
        at Planet.main(Planet.java:8)

Question 4. (5 points) Consider the following code, stored appropriately in the files Lens.java, Camera.java, and Builder.java and compiled into the associated class files.

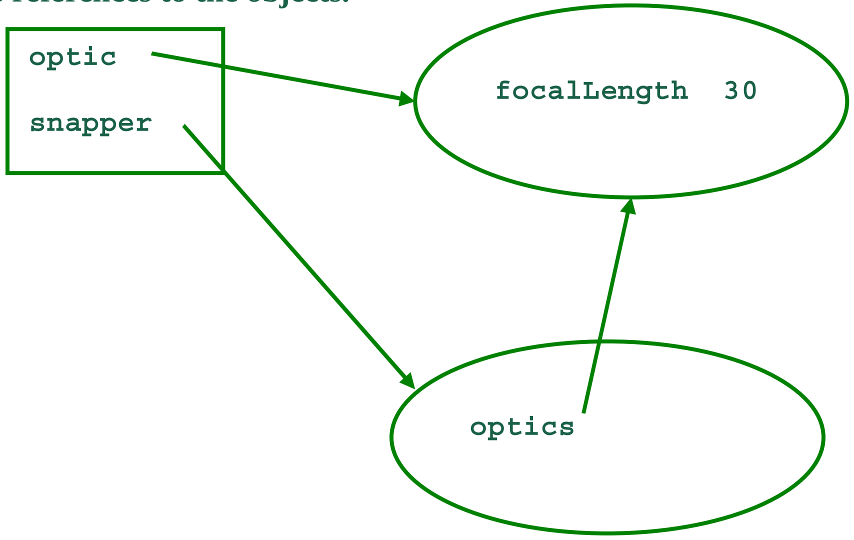
```
public class Lens {
  private int focalLength;
  public Lens(int length) {
    focalLength = length;
  }
}

public class Camera {
  private Lens optics;
  public Camera(Lens opt) {
    optics = opt;
  }
}

public class Builder {
  public static void main(String[] arg) {
    Lens optic = new Lens(30);
    Camera snapper = new Camera(optic);
    // draw the objects that exist at this point
  }
}
```

When the `main()` method of class `Builder` runs, it creates a `Lens` object and a `Camera` object. Draw a simple schematic picture (object diagram) as we have done in lecture that correctly shows the relationship between the local variables in method `main`, the `Camera` object, and the `Lens` object when the program is just about to exit the `main()` method. In the diagram of each of the objects, you should reserve and label a space for each instance variable. Fill in the correct value for a primitive type. Draw a line from a variable to an object to indicate a reference.

Note: names like `optic`, `snapper`, and `optics` are not part of the objects themselves; they are references to the objects.



The next two questions refer to the following class, which represents a single item in an online store’s shopping cart. You can remove this page from the test to save page-flipping while answering the following questions, however **you must hand in this page with your test to receive credit for the following questions.**

```
/** Representation of one item in the store inventory */
public class Item {
   private String name;          // item name
   private int catalogNumber;    // number of part in catalog
   private int quantity;         // how many of this item we have
                                 //   (must always be >= 0)
   private double weight;        // weight in ounces (must be > 0)
   private int price;            // price of one item in US pennies

   /** Construct a new Item object ... */
   public Item(String partName, int idNumber, double partWeight,
               int howMany, int partPrice) {
      name = partName;
      catalogNumber = idNumber;
      quantity= howMany;
      weight = partWeight;
      price = partPrice;
   }

   // methods to get Item properties (comments omitted to save space
   public String getName()    { return name; }
   public int getNumber()     { return catalogNumber; }
   public int getQuantity()   { return quantity; }
   public double getWeight()  { return weight; }
   public int getPrice()      { return price; }

   /** Add more of this item to the inventory
    * @param howMany number of items to add, must be > 0 */
   public void addItems(int howMany) {
      quantity= quantity + howMany;
   }

   /** Remove copies of this item from the inventory
    * @param howMany number of items to remove.  Must be > 0
    *                and less than or equal to numberInStock */
   public void removeItems(int howMany) {
      quantity = quantity - howMany;
   }
}
```

Question 5. (4 points) Choose the words or phrases from the following list that *best* complete the sentences below. Some of the sentences refer to the `Item` class defined on the previous page, others refer to general properties of software.

> **assignment, boolean, class, class definition, class invariant, client, cohesion, constructor, coupling, declaration, expression, instance variable, instance, local variable, loop, message, method, name thingy, parameter, precondition, postcondition, return statement, scratch space, state, this, type, void**

In the definition of class `Item`, the comment next to the declaration of instance variable `quantity` that says "must always be >= 0" is an example of a (an) _____**class invariant**_____ .

The statement "must be > 0" in the comment above method `addItems` is an example of a method ________**precondition**________ . In general, when we design software, we want to ensure that each class does only one thing well; to do this we maximize the ______**cohesion**______ of the class. To reduce the possibility that changes in one class will introduce errors in other classes, we try to reduce the ________**coupling**___________ between classes.

Question 6. (8 points) As part of our online store software, we want to have a class `ShoppingCart` to represent the items that the customer has currently selected to purchase. That class contains an instance variable `items` that is a list of items the customer has selected. Complete the implementation of method `getTotalPrice`, below, so it returns the total price of all the items currently in the shopping cart. **For full credit, you must use an iterator to access the `Item` objects in the `items` list.**

```
/** representation of a shopping cart */
public class ShoppingCart {

   // instance variables
   private ArrayList items;         // items in the shopping cart

   /** Construct a new ShoppingCart */
   public ShoppingCart() {
      items = new ArrayList();
   }

   // methods to add and remove Items from the ShoppingCart
   // omitted to save space

   /** Compute the current cost of the ShoppingCart
    * @return total price of all items in the shopping cart */
   public int getTotalPrice() {

      int total = 0;

      Iterator it = items.iterator();

      while (it.hasNext()) {

         Item thing = (Item)it.next();

         total = total + thing.getPrice() * thing.getQuantity();

      }

      return total;

   }

}
```

Note: the original intent of the question was to have each `Item` object include not only its price, but also the number of the items that were included in the shopping cart. The comments on the `quantity` instance variable were, however, a bit misleading, so no points were deducted if the answer just summed up the prices of the objects and didn't multiply each price by the corresponding quantity.

Question 7. (12 points) The UW Athletic Department needs to organize a database of the different sports offered during the year, but they are a little overworked right now, so they have asked you for help. Each individual sport is represented by a `Sport` object, which contains two strings holding the name of the sport, like “football”, and the season in which it is offered, either “fall”, “winter”, “spring”, or “summer”.

(a) Complete the definition of the `Sport` constructor so it properly initializes new `Sport` objects. **You may not change any of the variable names or other code below.**

```
/** Representation of one sport */
public class Sport {
   // instance variables
   private String sport;      // name of the sport
   private String season;     // season offered: “fall”, “winter”, …

   /** Initialize a new Sport object
    * @param sport name of the sport
    * @param season  season in which it is offered */
   public Sport(String sport, String season) {

      this.sport = sport;

      this.season = season;

   }

   /** Return the name of this sport */
   public String getSport() {
      return sport;
   }

   /** Return the season this sport is offered */
   public string getSeason() {
      return season;
   }

}
```

(b) Class `SportList` is a collection describing all of the sports offered during the year. Complete the definition of method `getSportsOfferedDuring` below so it returns a new `SportList` containing all of the `Sport` objects in the original list describing sports offered during the requested season. **For full credit you must use an iterator to process the original list.**

```
   /** A collection of sports */
   public class SportList {
      // instance variable
      private ArrayList sports;   // list of Sport objects in
                                  //   this SportList collection
      /** Construct a new, initially empty, SportList */
      public SportList() {
         sports = new ArrayList();
      }
      /** Add a new Sport to the SportList
       * @param sport  the new Sport object to add to the list */
      public void add(Sport sport) {
         sports.add(sport);
      }

      /** Return a list of all the sports offered in a season
       * @param season  the season to be selected
       * @return a list of all sports in this SportList offered
       *         during the requested season
       */
      public SportList getSportsOfferedDuring(String season) {

         SportList result = new SportList();
         Iterator it = sports.iterator();
         while (it.hasNext()) {
            Sport s = (Sport)it.next();
            if (s.getSeason().equals(season)) {
               result.add(s);
            }
         }
         return result;

      }
}
```