Question 1. (5 points) One of your colleagues is having a terrible time with the following code, which doesn't work properly.

```
/** A simple class */
public class Thing {
   // instance variables
   private String name;     // name of this Thing
   private double amount;   // current value of this Thing

   /** construct new thing with given name and amount */
   public Thing(String name, double amount) {
      name = name;
      amount = value;
   }

   /** double the value of this Thing */
   public void double () {
      double amount = 2 * amount;
   }
}
```

(a) (2 points) Describe where you could add `System.out.println(...)` statements to the code to figure out what's wrong.

At the end of the double method (and maybe at the beginning also) and at the end of the constructor.

(b) (3 points) Exactly what is wrong? (Be sure to describe all of the problems, if there are more than one.)

Constructor

name = name; assigns the value of the parameter 'name' to itself

amount = value; value doesn't exist, if it were changed to amount (the name of the parameter), it would have the same problem as the first assignment

double

amount is redeclared in the method's scope, so the instance variable is not updated

double is a reserved word (keyword) and can't be used as the name of a method

Question 2. (4 points) Fill in the blanks below with the most appropriate word or phrase from the following list.

> **assignment, argument, boolean, class, class definition, client, conditional, constructor, declaration, double, expression, integer, instance variable, instance, invariant, local variable, loop, message, method, name thingy, parameter, precedence, precondition, postcondition, return statement, return value, scope, scratch space, state, String, this, type, void**

The **scope** of a declaration is the region of the program where that declaration is in effect. A class **invariant** describes properties of instances of that class that are always true, except, perhaps, momentarily when the state of the instance has been partially, but not completely updated. The **postcondition** of a method is something that is guaranteed to be true after the method is executed, provided that all necessary **precondition**s are true when the method is called.

Question 3. (6 points) A *perfect number* is a positive integer such that the sum of its divisors (including the number) is equal to two times the original number.

Examples of perfect numbers:
6 is perfect since 1 + 2 + 3 + 6 = 12, which is 2 * 6.
28 is perfect since 1 + 2 + 4 + 7 + 14 + 28 = 56, which is 2 * 28

Examples of numbers that are not perfect
12 is not perfect since 1 + 2 + 3 + 4 + 6 + 12 = 28, which is not equal to 2 * 12.
7 is not perfect since 1 + 7 = 8, which is not 2 * 7.

Write a Java specification and implementation (you may combine these in your solution) of a method named `isPerfect`. This method should have one parameter, a positive integer. It should return the boolean value `true` if the parameter is a perfect number, and return `false` if it is not. Include preconditions and postconditions in your code where appropriate.

```
/** Return true if n is a perfect number, otherwise false
 *  precondition: n > 0
 */
public Boolean isPerfect(int n) {
   int sumOfDivisors = 1;
   for (int k = 2; k <= n; k++) {
      if (n % k == 0) {
         sumOfDivisors = sumOfDivisors + k;
      }
   }
   return (sumOfDivisors == 2*n);
}
```

Question 4. (5 points) [No Java programming necessary.] One of your colleagues has been designing a set of classes for an online retail store application. Here are the classes and properties in the current design.

class Shirt
- color
- manufacturer
- price
- inventory ID number

class Customer
- name
- customer ID number
- items currently in shopping cart (a collection)
- total price of items currently in shopping cart

class StorePatron
- name
- patron ID number
- address

class ShoppingCart
- list of items in shopping cart (a collection)
- total price of items in shopping cart

How would you modify this design to increase cohesion and reduce coupling? Feel free to create new properties and classes, or modify the existing ones. However, you don't need to include anything that is not included somewhere in the existing set of classes and properties.

- **Move address property from StorePatron to Customer, then get rid of class StorePatron (both of these classes represent store customers)**
- **Change Customer's "items currently in shopping cart" to a reference to a ShoppingCart object, and delete Customer's "total price" property**

Question 5. (5 points) Suppose we are working on an online retail store application. Here is a class that could be used to represent a shirt.

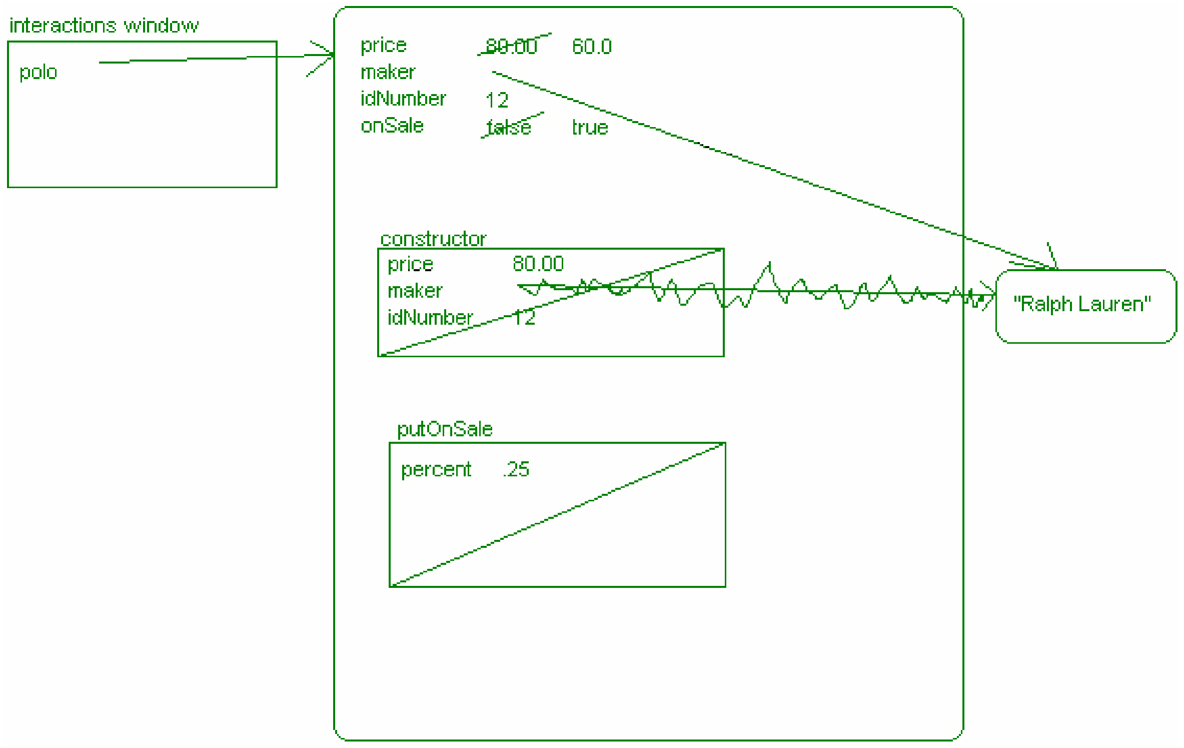
```
/** A simple Shirt class */
public class Shirt {
   // instance variables
   private double price;      // price for a single shirt
   private String maker;      // manufacturer of shirt
   private int idNumber;      // inventory id number for shirt
   private boolean onSale;    // true if shirt is currently on sale

   /** create a shirt given the price, maker, and idNumber.
    *  A newly created shirt is not on sale. */
   public Shirt (double price, String maker, int idNumber) {
      this.price = price;
      this.maker = maker;
      this.idNumber = idNumber;
      this.onSale = false;
   }

   /** If this shirt is not currently on sale, put it on sale by
    *  reducing its price by the percent given.  Return true if
    *  it is successfully put on sale, otherwise return false.
    *  precondition: percent must be between 0.0 and 1.0 */
   public boolean putOnSale (double percent) {
      if (onSale) { // already on sale – do nothing
         return false;
      } else {
         price = price * (1.0 -  percent);
         onSale = true;
         return true;
      }
   }
}
```

Draw a scope diagram, including objects and methods, that shows what happens when the following code is executed in DrJava's interactions window. Be sure to lightly cross out boxes using a single line to indicate scopes that are no longer in use.

```
Shirt polo = new Shirt(80.00, "Ralph Lauren", 12);
polo.putOnSale(.25);
```


interactions window
polo
price ~~80.00~~ 60.0
maker
idNumber 12
onSale ~~false~~ true
constructor
price 80.00
maker
idNumber 12
putOnSale
percent .25
"Ralph Lauren"

Question 6. (5 points) You have just taken over responsibility for a simple address book application. Complete the definition of method `numberOfFriendsNamed` in class `AddressBook` so it correctly returns the number of entries in the address book whose first names match the name given as a parameter. **For full credit**, your answers must use iterators correctly to process the list of friends in the `AddressBook`.

```
/** Description of one entry in the address book */
public class Friend {
   // instance variables
   private String firstName;      // first and last names
   private String lastName;
   private int phoneNumber;       // telephone number
   // methods to retrieve properties
   public String getFirstName()     { return firstName; }
   public String getLastName()      { return lastName;  }
   public int    getPhoneNumber() { return phoneNumber; }
}

/** A simple address book */
public class AddressBook {
   // instance variable
   private ArrayList friends;     // list of Friend objects
   ...
   (constructors and other methods omitted to save space)
   ...
   /** Return the number of Friend objects in this
    *  AddressBook whose first name equals parameter name */
   public int numberOfFriendsNamed(String name) {

       int nFriends = 0;
       Iterator itr = friends.iterator();
       while (itr.hasNext()) {
          Friend friend = (Friend)itr.next();
          if (name.equals(friend.getFirstName())) {
             nFriends++;
          }
          return nFriends;

   }
}
```