Name: _____

USC loginid (e.g., ttrojan):_____

CS 455 Midterm Exam 1
Spring 2011 [Bono]

Feb. 17, 2011

There are 4 problems on the exam, with 50 points total available. There are 7 pages to the exam, including this one; make sure you have all of them. There is also a double-sided one-page code handout that accompanies the exam. If you need additional space to write any answers, you may use the backs of exam pages (just direct us to look there). If you have scratch work in addition to your answer, circle your final answer, so we know what to grade.

Put your name and USC ID number at the top of the exam. Please read over the whole test before beginning. Good luck!

Remote DEN students only: Do not write on the backs of pages. If additional space is needed, ask proctor for additional blank page(s), put your name on them, and attach them to the exam.

	value	score
Problem 1	10 pts.	
Problem 2	10 pts.	
Problem 3	10 pts.	
Problem 4	20 pts.	
TOTAL	50 pts.	

Problem 1 [10 pts.]

Consider the following code fragment:
(Note: more about Point class on code handout.)

```
Point p1 = new Point(10, 20);
Point p2 new Point(p1);
Point p3 = p1;
p1.translate(5, 10);
p2.translate(5, 10);
p3.translate(5, 10);
System.out.println(p1 + " " + p2 + " " + p3);
```

Part A [5]. In the space below, draw a box-and-pointer diagram (a.k.a., memory diagram) showing all object variables, objects, and their state as they have changed during the code sequence.

Part B [3]. What is printed by the code? For the purpose of this problem assume a Point is printed as follows: `[x, y]`

Part C [2]. How many `Point` objects are created by the code above?

Problem 2 [10 pts.]

Change the `CashRegister` class from the textbook so a manager can track some information about the day's sales (the class itself and space for your answer is given on the next two pages). Here is a sample interaction to show the new functionality (calls to new methods shown in bold):

```
CashRegister reg = new CashRegister();


reg.recordPurchase(3.95);
reg.recordPurchase(22.99);
System.out.println(. . . reg.getTotal());
reg.enterPayment(30);
System.out.println(. . . reg.giveChange());
                                  // first customer finishes transaction
reg.recordPurchase(7.24);
reg.recordPurchase(7.24);
reg.recordPurchase(42.50);
System.out.println(. . . reg.getTotal());
reg.enterPayment(60);
System.out.println(. . . reg.giveChange());
                                 // second customer finishes transaction


. . . [many other sales take place throughout the day] . . .


reg.recordPurchase(4.99);
System.out.println(. . . reg.getTotal());
reg.enterPayment(10);
System.out.println(. . . reg.giveChange());
                           // last customer of the day finishes transaction


// call new methods for end of the day processing
System.out.println(. . . reg.getTotalSales());
System.out.println(. . . reg.avgSalesPerCustomer());
reg.reset();   // reset all totals for the next day
```

Problem 2 (cont.)

Here and continued on the next page is the complete `CashRegister` class definition *without* the new functionality. Add your new code inline with the existing code or to the right of the code with arrows clearly indicating where your new code would be inserted. For your convenience we already added the empty method bodies in bold.

```
/**
   A cash register totals up sales and computes change due.
*/
public class CashRegister {

   private double purchase;
   private double payment;



   /**
      Constructs a cash register with no money in it.
   */
   public CashRegister() {
      purchase = 0;
      payment = 0;
   }


   /**
      Records the sale of an item.
      @param amount the price of the item
   */
   public void recordPurchase(double amount) {
      double total = purchase + amount;
      purchase = total;
   }


   /**
      Gets total of all purchases made by this customer.
   */
    public double getTotal() {
       return purchase;
    };


   /**
      Enters the payment received from the customer.
      @param amount the amount of the payment
   */
   public void enterPayment(double amount) {
      payment = amount;
   }
```

[`CashRegister` class definition continued next page]

Problem 2 (cont.)

```
   /**
      Computes the change due and resets the machine for the next
          customer.
      @return the change due to the customer
   */
   public double giveChange() {
      double change = payment - purchase;
      purchase = 0;
      payment = 0;
      return change;
   }



   /**
      Returns total sales since the last reset(); if before first
         reset(), sales since it was created.
      @return total sales
   */
   public double getTotalSales() {




   }

   /**
      Computes the average sales per customer (how much each
           customer spent on average) since last reset();
           if before first reset(), avg since it was created.
       @return average sales per customer
   */
   public double avgSalesPerCustomer() {






   }

   /**
      Resets the machine to initial state.
          (Zeroes out all sales, etc.)
   */
   public void reset() {







   }
} // end of class CashRegister
```

Problem 3 [10 pts. total]

Implement the `formatSentence` method below that reads words from the given scanner until end of file printing them out formatted as a sentence. A sentence has one space between each word and a period right after the last word.

Example input:

```
The
big           dog
went
    home
```

Corresponding output:

```
The big dog went home.
```

```
// prints a version of text read from parameter "in" formatted as a
// sentence.
// if there are no words to read from "in", prints nothing.

public void formatSentence(Scanner in) {
```

Problem 4 [20 pts. total]

Implement the `hasPeak` method, which tells us whether its array of int data values, when plotted, has a single peak. This means that the sequence of values consists wholly of an increasing part followed by a decreasing part. You may assume no two consecutive values in the array are the same (i.e., no flat areas). Here are several examples:

vals:	**hasPeak(vals):**	**vals:**	**hasPeak(vals):**
3 7	false	3 5 2	true
7 3	false	3 2 1	false
3 5 9 2	true	3 5 2 3	false
3 5 9 3 2	true	2	false
<empty>	false		

```
public boolean hasPeak(int[] vals) {
```