Name: _____

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

CS 455 Midterm Exam 1
Fall 2011 [Bono]

Sept. 28, 2011

There are 5 problems on the exam, with 53 points total available. There are 7 pages to the exam, including this one; make sure you have all of them. There is also a 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 loginid 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	8 pts.	
Problem 5	15 pts.	
TOTAL	53 pts.	

Problem 1 [10 pts.]

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

```
public class Prob1 {
  public static Point foo(Point a) {
    a.translate(3, 4);
    return a;
  }
  public static void main(String[] args) {
    Point p1 = new Point(10, 40);
    Point p2 = new Point(foo(p1));
    Point p3 = foo(p2);
    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. This includes showing `foo`'s parameter `a`. Because `foo` is called multiple times, you can distinguish the formal parameters from the different calls with subscripts: a_1, a_2.

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.]

Let's consider a class similar to our Drunkard class from pa1, but that only can move in a one-dimensional space, that is, along the integer number line. We'll call this class `Drunkard1D`. This class will otherwise have the same functionality as our old Drunkard, except that the size of the step it takes each time is also random. It will take a step whose size will range from 1 to `Drunkard1D.MAX_STEP_SIZE`, inclusive. Implement the whole class (the class itself, interface documentation, and space for your answer is given on the next page). For this problem you will be evaluated on class design / style issues as well as code correctness.

Here's some sample code that uses our new class, and corresponding output:

```
Drunkard1D d = new Drunkard1D(3);
System.out.print(d.getCurrentLoc());  // print initial location (3)
for (int i = 0; i < numSteps; i++) {
  d.takeStep();
  System.out.print(" " + d.getCurrentLoc());
}
```

Sample output (actual output depends on what random sequence gets generated):

```
3 5 -1 -7 2 0 . . .
```

Hint: `Random` class interface given on the code handout.

[Problem continued on the next page.]

Problem 2 (cont.)

Complete the implementation of `Drunkard1D` – details of this problem are on the previous page.

```
/**
    Represents a "drunkard" doing a random walk on an integer number line.
    Drunkard chooses direction and step size randomly.  Steps range in
    distance from 1 and MAX_STEP_SIZE, inclusive.
*/
public class Drunkard1D {

   public static final int MAX_STEP_SIZE = 10;




   /**
      Creates drunkard with given starting location.
      @param startLoc starting location of drunkard
   */
   public Drunkard1D(int startLoc) {




   /**
      Takes a random-length step (see class comment above) in one of the
      two possible directions along the number line.
      Changes the current location of the drunkard.
   */
   public void takeStep() {









   /**
      gets the current location of the drunkard.
      @return an int representing drunkard's current location
   */
    public int getCurrentLoc() {
```

Problem 3 [10 pts.]

Implement the static method `readAndValidate`, which takes as its parameters a scanner, and a range of valid values. The method prompts for a value in the range given, reads the int from the scanner given, and validates that the value is in the range. By validate, we mean that it checks that the value read in was in the correct range, and if it wasn't it prints out an error message, and prompts and reads again, until the user enters a valid value. The method then returns this value. You may assume that users will always enter an int at your prompt (e.g., you do not have to handle a user typing a letter by mistake). Your output should follow the examples below (user input is in italics):

Ex1: Interaction for an example call to `readAndValidate` (this call returns `7`):

```
Enter int [1..7]: -3
ERROR: -3 not in range
Enter int [1..7]: 0
ERROR: 0 not in range
Enter int [1..7]: 7
```

Ex2: Interaction for a second example call to `readAndValidate` (this call returns `3`):

```
Enter int [1..7]: 3
```

```
/**
   Prompts for, reads and validates integer data from user.
   See above for detailed description and examples of behavior.
   @param in the Scanner to read from
   @param low the lower bound of valid range (inclusive)
   @param high the upper bound of valid range (inclusive)
   @return the validated integer read from user
*/
public static int readAndValidate(Scanner in, int low, int high)
```

Problem 4 [8 pts. total]

Suppose the following is a program a student wrote to solve the problem of figuring out how many students from a class got each score on a lab worth 4 points, when given a list of lab scores. The program is not required to do any error checking (i.e., assume valid input).

The student solution doesn't do what it is supposed to do. Java hint: the "new int[…]" expression also initializes the array (see code handout).

```
public static void main(String[] args) {
  Scanner in = new Scanner(System.in);

  System.out.print("Please enter number of scores: ");
  int numScores = in.nextInt();

  int[] counts = new int[numScores];

  System.out.println("Please enter the scores: ");

  for (int i = 0; i < numScores; i++) {

    int score = in.nextInt();
    counts[i] = counts[i] + score;

  }
                                          // reports the results in a table
  System.out.println("Score  How many got that score");

  for (int i = 0; i < counts.length; i++) {
    System.out.println(i + "      " + counts[i]);
                                       // spaces are to make columns line up
  }
}
```

Part A. Show the output that would be produced by this program for the input given below.
(I.e., output of running program presented here, not a "fixed" version of the program)

```
7
4 3 4 4 0 4 3
```

Part B. Fix the program so the program does what it was supposed to do (see problem description at the top of the page). Show your changes directly in the code above. (I.e., use arrows if necessary going from new code pointing to the exact location where it would be inserted, cross out anything that would get removed or changed.) Your new version must work with the same input format as the old one.

Problem 5 [15 pts. total]

Implement the `isLine` method, which tells us whether its array of int data values, when plotted, would be in a straight line. Here are several examples:

vals:	**isLine(vals):**	**vals:**	**isLine(vals):**
3 7	true	1 2 4 6 8	false
7 3	true	1 2 3	true
3 5 9 2	false	5 5 5 5	true
7 5 2 0 -2	false	2	false
7 5 3 1 -1	true	<empty>	false

```
public static boolean isLine(int[] vals) {
```