Name: _____

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

CS 455 Midterm Exam 1
Fall 2013 [Bono]

Wednesday, Oct. 2, 2013

There are 5 problems on the exam, with 53 points total available. There are 8 pages to the exam, including this one; make sure you have all of them. If you need additional space to write any answers, you may use the backs of exam pages (just direct us to look there).

Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and circling your final answers and crossing out any other answers you changed your mind about (though it's fine if you show your work).

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

	value	score
Problem 1	6 pts.	
Problem 2	10 pts.	
Problem 3	8 pts.	
Problem 4	14 pts.	
Problem 5	15 pts.	
TOTAL	53 pts.	

Selected methods of Java `Point` class:

`new Point(x, y)`

Constructs point object with given x and y values.

`new Point(p2)`

Constructs point object that has the same value as point `p2`.

`p.translate(dx, dy)`

Changes `x` and `y` values of `p` by `dx` and `dy`, respectively. I.e., if `p` had coordinates `(x, y)`, it's new value is a point with coordinates `(x+dx, y+dy)`

Problem 1 [6 pts.]

Consider the following static method that is supposed to return a String describing the weather, when given an outside temperature in Fahrenheit. (Approximately equivalent temperatures are also shown in Celsius for those of you who aren't used to Fahrenheit.) It doesn't always do the right thing.

```
public static String getWeather(int temp) {
  if (temp >= 90) { weather = "boiling"; }          // 90 is about 32 C
  if (temp < 90 && temp >= 80) { weather = "hot"; } // 80 is about 27 C
  if (temp < 80 && temp >= 70) { weather = "just right"; }// 70 is about 21 C
  if (temp < 70 && temp >= 60) {                    // 60 is about 16 C
     weather = "cool";
  }
  else {
     weather = "cold";
  }
  return weather;
}
```

Do not modify the code. **Show two example data values and the result of calling the method on each of them: the first one should be one where the existing method returns an incorrect weather description, and a second one such that the method returns an accurate weather description:**

temp	return value of **getWeather(temp)**

1. (wrong)

2. (right)

Problem 2 [10 pts.]

Consider the following code fragment:
(Note: more about `Point` class on the front page of the exam.)

```
Point p1 = new Point(5, 10);
Point p2 = new Point(p1);
Point p3 = p2;
p1.translate(3, 7);
p2.translate(3, 7);
p3.translate(3, 7);
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 3 [8 pts.]

Consider the following new version of the remove method we discussed for the names class. As before, you may assume that the helper method `lookupLoc` returns the location of the value to remove, or `-1` if it wasn't found:

```
public class Names {
    private String[] namesArr;     // the capacity is namesArr.length
    private int numNames;
    private final static int NOT_FOUND = -1;
    // . . . [not all of the Names class is shown here]

    /**
       Removes target from names, and returns true.
       If target wasn't present in names, returns false and no change
       made to names.
    */
    public boolean remove(String target) {
        int loc = lookupLoc(target);

        if (loc == NOT_FOUND) {
            return false;
        }

        for (int i = numNames - 1; i > loc; i--) {
            namesArr[i-1] = namesArr[i];
        }
        numNames--;

        return true;

    }
}
```

Part A [3]. Show an example `target` for which the method gives the wrong result and what that result is: (Hint: it may be helpful to hand trace the code to find the bug.)

Contents of `namesArr` and `numNames` before call to `remove`:

namesArr:	0	1	2	3	4	. . .
	Ann	Bob	Carol	Don	Ed	

numNames: 5

a. value of `target`: _____

b. contents of `namesArr` and `numNames` after call to `remove`:

Part B [5]. Fix the code above. Do not rewrite the whole method, but rather make your changes right into the code above, using arrows to show where your code should be inserted, crossing out code that you would get rid of, etc.

Problem 4 [14 pts]

Complete the implementation of the class `WordLengthFreq`, which computes the frequency of word lengths in a collection of Strings.

Here's an example of how we might use this class to compute the frequency of word lengths for all values from a file (using input redirection to get the data from a file):

```
public static void main(String[] args) {
   Scanner in = new Scanner(System.in);
   WordLengthFreq counter = new WordLengthFreq();
   while (in.hasNext()) {
      String word = in.next();
      counter.countWordLength(word);
   }
   counter.printFrequencies();
}
```

Sample input and output for the program above:

Contents of the file:

the farmer in the dell the farmer in the dell heigh ho the merrio the farmer in the dell

What gets printed by the program above:

```
Word Length   Frequency (count)
2             21% (4)
3             37% (7)
4             16% (3)
5             5% (1)
6             21% (4)
```

Hints and simplifying assumptions you may make:

- You may assume that no word will be greater than 80 characters long.
- Use `Math.round(double d)` to get the nearest `int` for percentage values.
- You do not have to worry about the column formatting above – you may print one space between a word length and its frequency.

Detailed descriptions of the methods appear with the class interface.

Turn the page for space for your answer (includes the class interface for you) →

Problem 4 (cont.)

```
/**
   Computes the frequency of word lengths for a collection of strings
*/
public class WordLengthFreq{









   /**
      Create an empty WordLengthFreq object.
      (i.e., 100% of word-lengths are 0, because there are no words)
   */
   public WordLengthFreq() {









   }


   /**
     Add this word to the collection for which we are computing the
     frequencies.
     PRE: 0 < word.length() <= 80
   */
   public void countWordLength(String word) {









   }
```

(class definition continued next page)

Problem 4 (cont.)

```
   /**
     Print out a table of the various word lengths and their frequencies as
     percentages, listed in increasing order by by word length.  Also shows
     number of occurrences for each of the lengths.  Lengths occurring with
zero
     frequency are not shown.
     If no words have been counted, print ”No words given”

   */
   public void printFrequencies() {
                    // we wrote the code for the header line for you
      System.out.println(“Word Length   Frequency (count)”);









   }

}
```

Problem 5 [15 pts]

Implement the static `boolean` method `hasAlt`, which returns true iff the given array consists solely of alternately increasing and decreasing pairs of adjacent values. For an array of zero or one elements hasAlt is true. The examples with the annotations to the right should make it clear what we mean by alternating pairs:

	`arr:`	return value of **`hasAlt(arr):`**	
Ex1:	`[1 2 5 8]`	`false`	*(values are all increasing)*
Ex2:	`[3 2 3]`	`true`	*(go down, go up)*
Ex3:	`[2 3 1 5 -2 3]`	`true`	*(increasing, decreasing, etc.)*
Ex4:	`[2 3 1 5 -2 -7]`	`false`	*(two in a row decreasing at end of array)*
Ex5:	`[1]`	`true`	
Ex6:	`[]`	`true`	
Ex7:	`[1 1 1]`	`false`	*(values don't go up or down)*
Ex8:	`[1 1 2]`	`false`	*(has a flat part)*

```
public static boolean hasAlt(int[] arr) {
```