Name: _____

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

CS 455 Midterm Exam 1
Spring 2014 [Bono]

Monday, Feb. 24, 2014

There are 6 problems on the exam, with 58 points total available. There are 9 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 username at the top of the exam. Please read over the whole test before beginning. Good luck!

	value	score
Problem 1	11 pts.	
Problem 2	6 pts.	
Problem 3	7 pts.	
Problem 4	6 pts.	
Problem 5	8 pts.	
Problem 6	20 pts.	
TOTAL	58 pts.	

Selected methods of Java `Point` class:

`new Point(x, y)`

Constructs point object with given x and y values.

`p.translate(dx, dy)`

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

Problem 1 [11 pts.]

Consider the following program: (Note: more about `Point` class on the cover page of the exam.)

```
public class Prob1 {
  public static void foo(Point a, Point b) {
    a.translate(3, 4);
    b = new Point(21, 25);
    System.out.println(a + " " + b);
  }
  public static void main(String[] args) {
    Point x = new Point(5, 10);
    Point y = new Point(12, 18);
    x = y;
    foo(x, y);
    System.out.println(x + " " + y);
  }
}
```

Part A [7]. 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 parameters.

Part B [4]. What is printed by the code? For the purpose of this problem assume a `Point` is printed as follows: `[x, y]`

Problem 2 [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) {
   String weather = "cold";
   if (temp >= 60) {           // 60 is about 16 C
      weather = "cool";
   }
   else if (temp >= 70) {      // 70 is about 21 C
      weather = "just right";
   }
   else if (temp >= 80) {      // 80 is about 27 C
      weather = "hot";
   }
   else if (temp >= 90) {       // 90 is about 32 C
      weather = "boiling";
   }
   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 3 [7 pts]

The following method doesn't work as advertised:

```
/**
   Returns true iff all the elements in the array arr are in increasing order with
   no duplicate values.
*/
public static boolean isIncr(int[] arr) {
   if (arr.length < 2) {
      return true;
   }
   boolean incr = false;
   for (int i = 0; i < arr.length-1; i++) {
      if (arr[i] < arr[i+1]) {
         incr = true;
      }
      else {
         incr = false;
      }
   }
   return incr;
}
```

Part A [3]. Show example data for which the method gives the wrong answer and what the method returns in that case:

`arr`:

return value of `isIncr(arr)`:

Part B [4]. 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 [6 points]

Part A. Consider a class, `Time`, that represents a time of day such as 9 A.M. or 3:30 P.M. Give two choices of instance variables that can be used for implementing the class (show types and names). For each choice, briefly describe your instance variables, including any restrictions on, or some examples of, valid values.

Choice 1:

Choice 2:

Part B. Suppose the implementor of the `Time` class changes from one representation to the other (e.g., from Choice 1 to Choice 2), keeping the public interface unchanged. What do the programmers who use the `Time` class (i.e., programmers of `Time` client code) need to do and why? (2 sentences maximum)

[space left here because of how the paging turned out]

Problem 5 [8 pts]

Complete the implementation of the class `Turtle`, which simulates a turtle moving back and forth on a horizontal line (e.g., you could think of it as an integral number line).

The turtle can only move left or right by one unit. Initially it's facing right, but it can turn to change its direction. Here's an example of how we might use the Turtle.

```
public static void main(String[] args) {
   Turtle yertle = new Turtle(3); // starts at position 3, facing right
   yertle.move();
   yertle.move();
   System.out.println(yertle.getPosition());   // he's now at position 5
   yertle.turn();   // turns in the opposite direction
   yertle.move();
   yertle.move();
   yertle.move();   // after this move he's at position 2
   yertle.turn();
   yertle.turn();   // facing left again
}
```

[The class interface and space for your answer is given on the next page →]

Problem 5 (cont.)

Detailed descriptions of the methods appear with the class interface below as well as space for your answer:

```
//   Turtle that can move back and forth in a one-dimensional world.
public class Turtle{




   //   Create a turtle at position startPosition, facing right
   public Turtle(int startPosition) {




   }

   //  Move the turtle by one unit in the direction he is facing.
   public void move() {




   }

   //  Turn turtle in the opposite direction.
   public void turn() {




   }

   //   Get turtle's current position.
   public int getPosition() {




   }
}
```

Problem 6 [20 pts]

Write the *Java* function `shrink` which shrinks a non-empty partially-filled array so that every `k` adjacent elements is replaced with the sum of those elements. If the size of the partially-filled array is not divisible by `k`, you just add up what's left over for the last element of the updated array.

Here are some examples (only the filled part of the array is shown):

`arr`	**`size`**	**`k`**	**`arr`** after the call to shrink	return value of shrink
[2,3,6,10,7,4,…]	*6*	*2*	*[5,16,11,…]*	*3*
[2,3,6,5,3,7,1,9,…]	*8*	*3*	*[11,15,10,…]*	*3*
[2,3,6,10,7,…]	*5*	*2*	*[5,16,7,…]*	*3*
[2,3,6,10,7,…]	*5*	*1*	*[2,3,6,10,7,…]*	*5*
[3,…]	*1*	*2*	*[3,…]*	*1*
[2,3,…]	*2*	*3*	*[5,…]*	*1*
[2,3,…]	*2*	*2*	*[5,…]*	*1*

Note: this method does not make a new array: `arr.length` will be the same after the call as it was before. Also, all the work can be done "in place" – no temporary arrays needed.

Reminder: for a partially-filled array, we only ever look at the values in the "filled" part of the array. We do not care what is in the other part of the array (the "…" in the above examples).

Reminder: the modulus operator is `%` (computes remainder of an integer division).

```
/**
   Replaces elements of partially-filled array, whose elements are in arr[0]
through arr[size-1], with sums of every k adjacent elements.  Afterwards,
the values can be found in arr[0] through arr[size'-1], where size' is the
return value of the method (the new size of the partially-filled array).
PRE: 0 < size <= arr.length   and   k > 0
*/
public static int shrink(int[] arr, int size, int k) {
```

[the above method header/comments and space for your answer is given on the next page →]

Problem 6 (cont.)

```
/**
   Replaces elements of partially-filled array, whose elements are in arr[0]
through arr[size-1], with sums of every k adjacent elements.  Afterwards,
the values can be found in arr[0] through arr[size'-1], where size' is the
return value of the method (the new size of the partially-filled array).

PRE: 0 < size <= arr.length   and   k > 0

*/

public static int shrink(int[] arr, int size, int k) {
```