Name: ____

USC NetID (e.g., `ttrojan`):____________________________

CS 455 Midterm Exam 1
Fall 2014 [Bono]

Tuesday, Sept. 30, 2014

There are 6 problems on the exam, with 63 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	10 pts.	
Problem 2	9 pts.	
Problem 3	6 pts.	
Problem 4	6 pts.	
Problem 5	12 pts.	
Problem 6	20 pts.	
TOTAL	63 pts.	

Selected methods of Java `Rectangle` class:

`new Rectangle(x, y, width, height)`

Constructs rectangle object whose top-left corner is at (`x`, `y`) and with given `width` and `height`.

`r.translate(dx, dy)`

Translates `Rectangle r` the indicated distance, to the right along the x coordinate axis by `deltaX`, and downward along the y coordinate axis by `deltaY`. This is a mutator.

`r.getX()`   `r.getY()`

Gets the x or y coordinate, respectively of `Rectangle r`

Problem 1 [10 pts.]

Consider the following program:
(Note: reminder about the `Rectangle` methods used here appears on the cover page of the exam.)

```
public class Prob1 {
   public static void main(String[] args) {
      Rectangle r = new Rectangle(30, 40, 100, 100);    // a 100x100 square
      Rectangle s = new Rectangle(15, 10, 100, 100);
                                                   // another 100x100 square
      for (int i = 0; i < 10; i++) {
         r = s;
         r.translate(1,1);
         // *
      }
      System.out.println(r.getX() + " " + r.getY());
      System.out.println(s.getX() + " " + s.getY());
   }
}
```

Part A [6]. 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 up to the ***first*** time we get to the comment labeled `*`. For the answer to this question do not update the diagram for later iterations.

Part B [4]. What is printed by the code?

Problem 2 [9 pts.]

Consider the following static method that is supposed to return the maximum value in an array of non-negative values. So if we had the array [7, 4, 9, 2, 6] it should return 9. However, the current implementation doesn't work correctly.

```
// PRE: arr.length > 0 and arr contains only non-negative values
public static int max(int[] arr) {

   int maxSoFar = 0;

   for (int i = 0; i < arr.length; i++) {

      if (arr[i] > arr[maxSoFar]) {

         maxSoFar = i;

      }

   }

   return maxSoFar;

}
```

Part A [2]. What is the return value of this method as currently written when given the array:

[7, 4, 9, 2, 6]

Part B [3]. Write a single sentence describing what the method *currently* does in general.

Part C [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 new code should be inserted, crossing out code that you would get rid of, etc. The new version should behave as described at the top of this page.

Problem 3 [6 pts]

Complete the static method, `prob3`, below that takes three numbers and prints "`all the same`" if they are all the same, "`all different`" if they are all different, and "`neither`" otherwise.

```
public static void prob3(int a, int b, int c) {
```

Problem 4 [6 points]

Consider the following implementation of a class `Square`:

```
public class Square {

   private int length;

   private int area;

   // create a square with given length for a side
   public Square(int sideLength) {

      length = sideLength;

      area = sideLength*sideLength;

   }

   // get the area of the square
   public int getArea() { return area; }

   // double the length of each side
   public void grow() { length = 2 * length; }

}
```

Part A. The code has a bug. **Fix the bug.** Do not rewrite the whole class, but rather make your changes right into the code above, using arrows to show where new code should be inserted, crossing out code that you would get rid of, etc.

Part B. Refactor your code for class `Square` to make it less prone to such errors in the future. For example, if we add more methods later, we'd be less likely to make the same kind of mistake from the original code. Your refactored version will also not have the bug, of course. **Make your changes using the copy of the original code below so we don't lose your answer to part A. Note: it's possible you already made this improvement for part A. If so, just write "already improved" below.** Hint: it involves more than just changing one method.

```
public class Square {

   private int length;

   private int area;

   // create a square with given length for a side
   public Square(int sideLength) {

      length = sideLength;

      area = sideLength*sideLength;

   }

   // get the area of the square
   public int getArea() { return area; }

   // double the length of each side
   public void grow() { length = 2 * length; }

}
```

Problem 5 [12 pts]

Complete the implementation of the class `Turtle2D`, which simulates a turtle moving around on a two-dimensional grid (sort of like Drunkard, but we get to control where he moves).

The turtle can only move forward by some number of units. Initially he's facing east, but he can turn right to change his direction. The coordinate system is as in the Java GUI system: the origin is at the upper left of the grid; `x` grows towards the right; `y` grows downward. Here's an example of how we might use the `Turtle2D`:

```
public static void main(String[] args) {
   Turtle2D yertle = new Turtle2D(new ImPoint(3, 20));
                                 // starts at loc (3,20) facing Turtle2D.EAST
   yertle.advance(10);
   yertle.advance(4);
   System.out.println(yertle.getLoc());   // he's now at loc (17,20)
   System.out.println(yertle.getDir());   // still Turtle2D.EAST
                // (although the above statement will print it as a number)
   yertle.turnRight();   // turns right by 90 degrees
   yertle.turnRight();   // turns right by 90 degrees
   yertle.turnRight();   // turns right by 90 degrees
   System.out.println(yertle.getDir());   /// returns Turtle2D.NORTH
   yertle.advance(12);
   System.out.println(yertle.getLoc());   // he's now at loc (17,8)
}
```

The class interface uses the `ImPoint` class (the class for immutable points from pa1):

Reminder of selected `ImPoint` methods by example:

`ImPoint point = new ImPoint(x,y);`	Creates a point with coordinates `(x,y)`
`int x = point.getX();`	Returns the x coordinate of the point.
`int y = point.getY();`	Returns the y coordinate of the point.
`ImPoint anotherPoint = point.translate(deltaX, deltaY);`	Returns an `ImPoint` with coordinates translated by `deltaX` and `deltaY` from this point.

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

Problem 5 (cont.)

Detailed descriptions of the methods appear with the class interface below (and on the next page) as well as space for your answer:

```
//   Turtle that can move around in a two-dimensional world.
public class Turtle2D {
   public static final int NORTH = 0;
   public static final int EAST = 1;
   public static final int SOUTH = 2;
   public static final int WEST = 3;
   public static final int NUM_DIRECTIONS = 4;







   //   Create a turtle at location startPos, facing Turtle2D.EAST
   public Turtle2D (ImPoint startPos) {








   }

   //   Get turtle's current position.
   public ImPoint getLoc() {







   }

   //   Get turtle's current direction
   // Returns one of Turtle2D.EAST, Turtle2D.SOUTH, Turtle2D.NORTH, or
   //       Turtle2D.WEST,
   public int getDir() {








   }
```

[interface and space for your answer continued on the next page . . .]

Problem 5 (cont.)

(Space for method implementations, continued.)

```
   //  Turn turtle right by 90 degrees.
   public void turnRight() {









   }


   //  Advance the turtle forward by distance units in the direction
   //  he is facing.
   //  PRE: distance >= 0   [ he can't go backwards ]
   public void advance(int distance) {










   }

}
```

Problem 6 [20 pts]

Implement the boolean function `isFibSeq` which returns true iff its argument is an array of the first *k* numbers in the Fibonacci sequence, for some *k* > 0.

The Fibonacci sequence starts as follows: 1, 1, 2, 3, 5, 8, 13, . . .
The rules for generating the sequence are, where f_n, below, means the *n*'th Fibonacci number:

$f_1 = 1$

$f_2 = 1$

for any $n > 2$, $f_n = f_{n-1} + f_{n-2}$

Thus, the nth Fibonacci number is the sum of the previous two numbers in the sequence. So, in the sequence shown above, the third value is 1 + 1 = 2; the fourth value is 1 + 2 = 3, and the fifth value is 2 + 3 = 5, etc.

Examples below (array of ints shown as a comma separated sequence of ints surrounded by square brackets):

arr	**isFibSeq(*arr*)**	***arr***	**isFibSeq(*arr*)**
[1,1,2,3,5,8,13,21]	*true*	*[1,1,2,3,5,8,14,21]*	*false*
[1,1]	*true*	*[1,4]*	*false*
[1]	*true*	*[2]*	*false*
[1,1,2]	*true*	*[2,3,5]*	*false*

```
// PRE: arr.length > 0
public static boolean isFibSeq(int[] arr) {
```