Name: ______________________________

USC Student ID Number:______________________________

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

CS 455 Midterm Exam 1
Fall 2021 [Bono]

Tuesday, Sept. 28, 2021

There are 6 problems on the exam, with 61 points total available. There are 12 pages to the exam (6 pages **double-sided**), including this one; make sure you have all of them. If you need additional space to write any answers or scratch work, pages 10, 11, and 12 are left mostly blank for that purpose. If you use those pages for answers you just need to direct us to look there. ***Do not detach any pages from this exam.***

Any remote students printing the exam should scan, and submit all pages of the exam, even if you didn't write on a particular page. If you print it single-sided, do not write on the backs of pages and do not scan blank backs of pages.

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, USC Student ID, and USC username (a.k.a., NetID) at the top of the exam. Also put your NetID at the top right of the front side of each page of the exam (including the last page). Please read over the whole test before beginning. Good luck!

Selected methods of Java `Point` class shown by example (for problems 1 and 2):

`new Point(x, y)`
    Constructs point object with given `x` and `y` values.

`new Point(p)`
    Constructs point object with the same `x` and `y` values as point object `p` has.

`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 [10 pts. total]

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

```
public class Prob1 {
   public static void main(String[] args) {
      Point p = new Point(6, 12);
      Point r = new Point(15, 10);
      for (int i = 0; i < 10; i++) {
         r = p;
         r.translate(2, 2);
         // *
      }
      System.out.println(p + " " + r);
   }
}
```

Part A [6]. In the space below, draw a single 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? For the purpose of this problem assume a `Point` is printed as follows: `[x, y]`

Problem 2 [15 pts. total]

Consider the class `Ant2D`, whose interface is shown below:

```
//   Ant that can move around in a two-dimensional world.
public class Ant2D {
   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;

   // . . .

   //   Creates an ant at location startPos, facing EAST
   public Ant2D(Point startPos) { .  . . }

   //   Gets ant's current position.
   public Point getLoc() { . . . }

   //   Gets ant's current direction
   // Returns one of NORTH, EAST, SOUTH, or WEST
   public int getDir() { . . . }

   //  Turns ant right by 90 degrees.
   public void turnRight() { . . . }

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

Part A [3]. Suppose you want to use incremental development to implement and test this class. **Indicate which methods would be in a minimal subset of methods you could implement and test together before implementing any of the other methods: [circle the method names below to indicate the subset]**

```
constructor       getLoc       getDir      turnRight        advance
```

Problem 2 (cont.)

Part B [12]. Implement the `Ant2D` class. Here's some more information about how it works:

The ant can only move forward (i.e., in the direction he is facing) 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 `Ant2D`:

```
public static void main(String[] args) {
   Ant2D adam = new Ant2D(new Point(3, 20));
   // starts at loc (3,20) facing Ant2D.EAST
   adam.advance(10);
   adam.advance(4);
   System.out.println(adam.getLoc());   // he's now at loc (17,20)
   System.out.println(adam.getDir());   // still Ant2D.EAST
   // (although the above statement will print it as a number)
   adam.turnRight();   // turns right by 90 degrees
   adam.turnRight();   // turns right by 90 degrees
   adam.turnRight();   // turns right by 90 degrees
   System.out.println(adam.getDir());   /// returns Ant2D.NORTH
   adam.advance(12);
   System.out.println(adam.getLoc());   // he's now at loc (17,8)
}
```

The class itself, and space for your answer starts here and continues onto the next page: (Note: more about `Point` class on the cover page of the exam.)

```
//   Ant that can move around in a two-dimensional world.
public class Ant2D {
   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;




   //   Creates an ant at location startPos, facing EAST
   public Ant2D(Point startPos) {



   }
```

(continued next page)

Problem 2 (cont.)

```
   //   Gets ant's current position
   public Point getLoc() {




   }

   //   Gets ant's current direction
   // Returns one of NORTH, EAST, SOUTH, or WEST
   public int getDir() {




   }

   //  Turns ant right by 90 degrees
   public void turnRight() {




   }

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




   }
}
```

Reminder of some Linux shell commands and other shell syntax (for problem 2 below):

```
ls   cd   cp   mv   rm   mkdir   rmdir   more   diff   ~   *   ..   <   >   .
```

Problem 3 [4 pts]

The directory in Vocareum that has all the lecture code examples is `$ASNLIB/public` in the Vocareum *Lecture code* area. Write a sequence of one or more Linux commands to copy *all* the files from the `09-16` subdirectory of the lecture code examples directory to a new directory named `09-16` in your own Lecture code workspace home directory.

Relevant information:

- You may assume there are no subdirectories in the existing `09-16` directory, just regular files.
- `$ASNLIB` is a shell variable defined on Vocareum that expands to the correct *absolute* path to get to that specific directory.
- A reminder of the names of some common Linux commands and related syntax appears above this problem.
- The commands below ($ is the Linux prompt) show the starting conditions in your Lecture code workspace, including the current contents of that workspace.

  ```
  $ cd
  $ ls
  arrayTests   08-31   09-09
  ```

Problem 4 [9 pts.]

Below is the specification for the Bar class you used in PA1:

`Bar` class
A labeled bar that can serve as a single bar in a bar graph. The text for the label is centered under the bar.

```
public Bar(int bottom, int left, int width, int applicationHeight,
              double scale, Color color, String label)
```

Creates a labeled bar. You give the height of the bar in application units (e.g., population of a particular state), and then a scale for how tall to display it on the screen (parameter scale).

Parameters:

`bottom`	location of the bottom of the bar
`left`	location of the left side of the bar
`width`	width of the bar (in pixels)
`applicationHeight`	height of the bar in application units
`scale`	how many pixels per application unit
`color`	the color of the bar
`label`	the label under the bar

```
public void draw(Graphics2D g2)
```

Draws the labeled bar

Parameters:

`g2`	the graphics context

Suppose you are creating the Bar objects to draw a bar graph like we did for our program (recall this happens inside `paintComponent`). Consider the arguments you pass into each call to the constructor.

Part A. Which of the following parameters (described above) will get passed the *same* argument value for each bar in the graph? [circle all that apply]

```
bottom     left      width     applicationHeight     scale      color       label
```

Part B. Which of the following parameters (described above) will get passed an argument value that *depends on the current dimensions of the window*? [circle all that apply]

```
bottom     left      width     applicationHeight     scale      color       label
```

Part C. Which of the following parameters (described above) will get passed an argument value that *depends on the number of trials to display with this bar*? [circle all that apply]

```
bottom     left      width     applicationHeight     scale      color       label
```

Problem 5 [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 6 [17 pts total]

Part A [15]. Implement the static `neighborAverage` method, which takes an array of ints called `nums`, and returns an array containing the averages of sequences of three adjacent neighbors from the `nums` array. I.e., the first element (index `0`) of the resulting array will be the average of the first three values in `nums` (at indices `0, 1,` and `2`), the second element will be the average of the fourth through sixth values, etc. The resulting array should be the exact size to store these results (so any 0s in the resulting array would come from an average value that is 0).

Please note the precondition below to make this problem easier.

Here are some examples (answers shown below are not exact with respect to roundoff errors in Java):

```
nums                              neighborAverage(nums)
[-1, -1, -4]                      [-2.0]
[5, 5, 5, 1, 3, 2, 3, 0, 0]       [5.0, 2.0, 1.0]
[2, 4, 4, 5, 3, 2]                [3.33334, 3.33334]
[1, 2, 3, 4]    does not satisfy the precondition - you do not have to
                           handle this case.
```

```
// PRE: the length of nums is > 0 and is a multiple of three
public static double[] neighborAverage(int[] nums) {
```

Part B [2]. Complete the following assert statement so that it checks the method precondition stated above. Complete the statement below assuming it would appear in the scope of `neighborAverage`.

```
assert ____________________________________________________________ ;
```

Extra space for answers or scratch work.

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.

Extra space for answers or scratch work (cont.)

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.

Extra space for answers or scratch work (cont.)

If you put any of your answers here, please write a note on the question page directing us to look here. Also label any such answers here with the question number and part, and circle the answer.