Name: ______________________________________

USC Student ID Number:________________________________

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

CS 455 Midterm Exam 1
Spring 2020 [Bono]

Thursday, Feb. 20, 2020

There are 7 problems on the exam, with 65 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 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.***

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 (for problem 1):

`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)`

`String` class `compareTo()` method (for problem 5):

`s1.compareTo(s2)`

For `Strings` `s1` and `s2` in the call above, `compareTo` returns an `int` value either greater than zero, less than zero, or equal to zero, depending on which one has the greater lexicographic value (effectively, alphabetical comparison for words). If the return value is:

`< 0`	`s1` comes before `s2` in an alphabetical ordering
`> 0`	`s1` comes after `s2` in an alphabetical ordering
`== 0`	`s1` has the same value as `s2` (i.e., `s1.equals(s2) == true`)

Reminder of some Linux shell commands and other shell syntax (for problem 3):

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

Problem 1 [12 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]`

Part C [1]. How many point objects are created by the program above? ______________

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 [3 pts]

Write a sequence of at most 3 Linux commands to copy all the files from the `baz` directory into the the `blob` directory, assuming the starting conditions below: (`$` is the Linux prompt here). Note: see cover page for reminders of Linux syntax.

```
$ cd
$ ls
bar baz foo
$ ls baz
a b c
$ cd foo/blob
$ ls
d e f
```

Problem 4 [10 points]

A drunkard's walk (or random walk) simulates a drunkard walking randomly around a city (a 2D grid). Consider a class `Drunkard1D`, which models a drunkard's walk, but such that the drunkard can only move in a one-dimensional space, that is, either left or right along the integer number line. A `Drunkard1D` will be able to take a step in a random direction and of a random step size. The step size will range from 1 to `Drunkard1D.MAX_STEP_SIZE`, inclusive. **Complete the implementation of this class.** Read on for more information. Space for your answer is on the next page.

Here's some sample code that uses the Drunkard1D class, and corresponding output:

```
Drunkard1D d = new Drunkard1D(3);
System.out.print(d.getCurrentLoc());  // prints 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: Selected methods of Java `Random` class:

`new Random()`

Creates a random number generator.

`int nextInt(int max)`

Generates and returns a random integer uniformly distributed in the range `[0, max)` (i.e., 0 <= num < max)

`boolean nextBoolean()`

Generates and returns a random boolean value uniformly distributed (i.e., always returns one of `true` or `false`)

The class itself, documentation, and space for your answer is on the next page:

[Problem 4 -- continued from previous page]

```
//  Represents a "drunkard" doing a random walk along an integer number line.
//  Drunkard chooses direction and step size randomly.  Steps range in
//  distance from 1 to 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 of size in the range [1, MAX_STEP_SIZE] in
   // a random direction on the number line.
   public void takeStep() {









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

Problem 5 [13 pts. total]

Part A [5]. Consider the following working implementation of a `Names` class helper method, `lookupLoc`, that uses linear search. Unlike the Names class we did in lecture, this one uses an `ArrayList` to store the names. The current version of the method would work even if the `ArrayList` were unordered: it always looks at all `size()` elements in the `ArrayList` for *un*successful searches. **Refactor the code below so that the new version takes advantage of the fact that the names in `namesArr` are in increasing order: so that it, in general, will perform better on unsuccessful searches than the old version. (Do not implement binary search.)**

Example:

```
target:    Li
namesArr:  [Avinash, Carly, John, Mana, Peter, Sa, Yiqi]
```

we don't need to look past `Mana` at `Peter`, `Sa`, or `Yiqi` to figure out that `Li` is not there.

Note: Do not rewrite the whole method, but rather make your changes right into the code below, using arrows to show where any new code should be inserted, if necessary, crossing out code that you would get rid of, etc. Also, see front of exam for `compareTo` method for doing inequality comparisons between `String`s.

```
public class Names {

   //  representation invariant:
   //  values in namesArr are unique and in increasing alphabetical order
   private ArrayList<String> namesArr;

   . . . [rest of Names class not shown]

   //  returns location of target in namesArr or -1 if not found
   private int lookupLoc(String target) {    [do not change the method header]



     for (int i = 0; i < namesArr.size(); i++) {



        if (target.equals(namesArr.get(i))) { return i; }



     }



     return -1;

   }
}
```

Problem 5 (cont.)

Part B [8]. Come up with a good set of test cases to thoroughly test your new version of `lookupLoc` (i.e., this should include cases we would have used on the old version as well as ones designed to exercise all parts of the new code). You should provide at least 8 test cases.

You can use the following `namesArr` contents in some of your tests (Test data ***A***):

```
namesArr:  [Avinash, Carly, John, Mana, Peter, Sa, Yiqi]
```

For each test case,

a. give the exact input that would be used (for `target`; and for `namesArr`, if different from the `namesArr` given above)
b. any `namesArr` you give must satisfy the representation invariant given in class definition on previous page
c. show the expected return value of the method for that case

target	**namesArr** (you can identify the one given above as *A*)	**expected return value** (an int)

Problem 6 [6 points]

Consider the CashRegister class from the textbook and a recent lab:

```
public class CashRegister   // totals up sales and computes change due
{  . . .
   private double purchase;
   private double payment;

   //  Constructs a cash register with no money in it.
   public CashRegister()  { . . . }

   //  Records the purchase price of an item.
   //  @param amount the price of the purchased item
   public void recordPurchase(double amount) { . . . }

   //  Gets total of all purchases made.
    public double getTotal() { . . . }

   //  Processes the payment received from the customer.
   public void receivePayment(int dollars, int quarters,
                              int dimes, int nickels, int pennies)  { . . . }

   //  Computes the change due and resets the machine for the next customer.
   //  @return the change due to the customer
   public double giveChange()  { . . . }

}
```

For each of the following proposed changes to the above `CashRegister` class, indicate which *other* code would have to change as well (using one or more of the letters below to indicate answers):

(M) method bodies in CashRegister

(C) client code of CashRegister (i.e., classes and programs that use CashRegister)

(N) neither method bodies nor client code has to change.

(For each change described below, assume you are starting from the original code above, i.e., they are not done in sequence.)

Part A. You change the name of the instance variable `payment` to `totalPayment`

Part B. You change the name of the method `recordPurchase` to `addPurchase`

Part C. Currently, the `purchase` and `payment` amount instance variables are in dollars (`double`). You change them so that they are now represented as values in cents (`int`).

Part D. Instead of accepting money for the cash register as some number of dollars, quarters, dimes, etc. you accept it as a `Change` object (that contains some number of dollars, quarters, dimes, etc.), and also give the change back as a `Change` object (assume the `Change` class is already written).

Problem 7 [15 pts]

Implement the static `createSequence` method, which takes a positive `int`, *n*, and returns an array with the values [1, 1,2, 1,2,3, . . . 1,2,3,..,*n*]. Hint: the length of this sequence is the sum of the numbers from 1 to *n*, which can be described by the expression: *n**(*n*+1)/2.

Here are some examples:

n	**createSequence(n)**
1	[1]
2	[1, 1, 2]
3	[1, 1, 2, 1, 2, 3]
4	[1, 1, 2, 1, 2, 3, 1, 2, 3, 4]

```
// PRE: n > 0
public static int[] createSequence(int n) {
```

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.