CIS 203: Midterm 1, Weeks 1-6 / Spring 2023

Directions:

- **This is a closed book, closed notes midterm. The only information you may use is the provided reference page.**
- **Please read and follow all directions carefully.**
- **On the separate paper provided, respond as indicated by each item. Please be sure to put your name on EACH page of your solutions and number the problems.**
- **Please do not write solutions on this paper. I WILL NOT GRADE ANYTHING ON THIS TEST PAPER.**
- **The point value for each question is indicated. There are 72 points total.**
- **You will hand in this paper, the reference page, AND your answer pages.**
- **You have the entire class time for this midterm (1 hour 15 minutes).**

1. **Writing a class.** (22 pts) Below is the partial implementation of a class called `Line` that represents a line segment between two `Points` (information for using `Point` is on the reference page). Write the implementations for the `Line` methods as described for each part of the problem.

```
public class Line {

   private Point point1; // one end point of the line
   private Point point2; // the other end point of the line
}
```

   a. (3 pts) Write a constructor that accepts two `Point` parameters for the values of `point1` and `point2` and sets each field value equal to the appropriate parameter value.

   b. (3 pts) Write a default constructor that sets the values of both fields of `Line` to (0, 0) without explicitly setting the field values inside the default constructor.

   c. (4 pts) Write getter/accessor methods `getPoint1` and `getPoint2.` The methods will return the value of the appropriate field to the calling code.

   d. (4 pts) Write setter/mutator methods `setPoint1` and `setPoint2` for the `Line` class. Each method accepts a `Point` parameter and changes the value of the appropriate field to the value of the incoming parameter.

   e. (4 pts) Write the `toString` method for the `Line` class, so that the method provides a meaningful format for the `Line`, in the form "`[(3, 7), (-1, 4)]`". The coordinate values shown are examples only.

   f. (4 pts) Write a method called `getSlope` that returns the slope of this line, a `double` value. The slope of a line between points `(x1, y1)` and `(x2, y2)` is equal to `(y2 - y1) / (x2 - x1)`. If the two points have the same x-coordinates, the denominator is zero and the slope is undefined, so you should throw an `IllegalStateException` in this case, handling the error with a simple `try/catch`.

2. **Interfaces:** (20 pts) Refer to the `Vehicle` interface on the reference page for this problem. You do not have to write code beyond what is needed for implementing the `Vehicle` interface (*i.e.,* no constructors, setters, getter, *etc*.).

   a. (10 pts) Write a class named `Train` that implements the `Vehicle` interface, using the following information.

   - `Train` has a class constant, `double TOP_SPEED`, that defines its maximum speed.
   - A `Train` can travel at top speed for 20 hours without refueling. A refueling stop costs 2 hours. A `Train`'s range is calculated as the product of the top speed and hours traveled, taking refueling into account.

   b. (10 pts) Write a class named `Plane` that implements the `Vehicle` interface, using the following information.

   - `Plane` has two class constants, `double TOP_SPEED`, that defines its maximum speed, and `int TIME_LIMIT`, the maximum number of hours the pilot can fly.
   - A `Plane` can travel at top speed for 14 hours without a layover for refueling. A refueling stop costs 1 hour, unless the number of flight hours exceeds the pilot's time limit. If that happens, a refueling layover takes 4 hours, for a crew change. A `Plane`'s range is calculated as the product of the top speed and hours traveled, taking refueling and crew change layovers into account.

3. **Arrays and `ArrayLists`:** (20 pts) You will implement a method named `minToFront()` first using an array and then a different implementation that uses an `ArrayList`. Method `minToFront()` takes a sequence of integers as a parameter and moves the minimum value in the sequence to the front, otherwise preserving the order of the elements. For example, if a variable called `someSeq` stores the following values: `{3, 8, 92, 4, 2, 17, 9}` and you make this call: `minToFront(someSeq);` it should store the following values after the call: `{2, 3, 8, 92, 4, 17, 9}` You may assume that the sequence is non-empty.

   a. (10 pts) Implement `minToFront()` using an array to store the sequence. The parameter for the method will be an array of integers.

   b. (10 pts) Implement `minToFront()` again using an `ArrayList` to store the sequence. The parameter for the method will be an `ArrayList` of integers.

4. **Hardware and the Machine Instruction Cycle** (10 pts).

   a. (5 pts) Refer to the following pseudocode for this problem.
   (`IDR`: instruction decode register; `PC`: program counter)

```
while (computer is on) {
    fetch instruction RAM[program counter] -> IDR
          finish by adding 1 to PC;
    decode IDR;
    execute decoded instruction, modify program counter if necessary;
```

   When might "modify program counter" actually be necessary? What *kind* of machine instruction might change the `PC`?

   b. (5 pts) There is a computer with 16 gigabytes of RAM. The whole RAM is similar to what data structure that you have used in CIS 203? What is the type of the elements of the RAM?