CS 235 - Homework 3

Deadline:

11:59 pm on **Friday, September 24, 2021**.

Purpose

To get some practice writing another class, some subclasses, and some `try-catch` blocks.

How to submit:

Submit your `.java` files for this homework to the course Canvas site. (You'll be creating `.class` files, also, but you do not submit those.)

Important notes:

- Follow the initial class Java coding standards mentioned in class -- some of these include:
  - Follow the Java naming standards that have been discussed in class.
  - Attempt "javadoc-style" comments for **each** Java class and method, in the same style as you see in posted in-class examples.
  - Everything inside a set of { } must be indented by AT LEAST 3 SPACES -- and the beginnings of statements that are sequential should be indented the SAME NUMBER of spaces. (That is, sequential statements should line up.)
  - { and } should each go on their OWN line, with { lined up evenly with the preceding line, with the { }'s contents indented by at least 3 spaces, and with } lined up with the opening {. That is, handle the curly braces as you see in all posted class examples!
- ASK ME if any of these are unclear to you!

Problem 1

(This is a slightly-modified version of Week 4 Lab Exercise - Problem 3 that a few finished, but many did not. If you did finish it, you should submit it for this homework problem, also.)

Copy the following from the Week 4 Lab Exercise to your folder for Homework 3:

- `Plottable.java` (provided along with the Week 4 Lab Exercise)
- `Point.java` (that you created for the Week 4 Lab Exercise - Problem 2)

Write a small class `ColorPoint` that is a subclass of your class `Point` that also meets the following requirements. It should include:

- a private data field for a point's color

- a no-argument constructor that, when called, calls the superclass' constructor appropriately to set the (inherited) x-coordinate to 0, the (inherited) y-coordinate to 0, the (inherited) name to "", and set the color to "black".
- a 4-argument constructor that, when called with an initial x-coordinate, initial y-coordinate, an initial name, and an initial color, calls the superclass' constructor appropriately to set the (inherited) x-coordinate, the (inherited) y-coordinate, and the (inherited) name to the specified values, and then sets the `ColorPoint`'s color to the specified color
- an accessor `getColor` that returns the calling `ColorPoint`'s color
- a mutator `setColor` that changes the calling `ColorPoint`'s color
- a redefined version of the inherited method `toString`, that returns a `String` containing all of the calling `ColorPoint`'s data field values that matches this output format:

```
ColorPoint[x: 0.0, y: 13.0, name: finish, color: red]
```

  - (Note: whatever format the `double` x- and y-coordinates happen to appear by default is fine here!)
- Now, to demonstrate this, write a small Java application class `DemoPoints.java` whose `main` method contains at least the following (and you may have more than this if you'd like!):
  - create at least two `Point` objects of your choice, calling each of its constructors at least once
  - create at least two `ColorPoint` objects of your choice, calling each of its constructors at least once
  - call `System.out.println` at least four times, each time with one of your `Point` or `ColorPoint` objects as its ONLY argument, for each of your `Point` and `ColorPoint` objects.
  - call `setX`, `setY`, `setName`, and `setColor` for at least once of your `ColorPoint` objects, actually changing the state of the calling `ColorPoint`.
    - (this should show that `ColorPoint` did inherit the first 3 mutators from `Point`)
  - again call `System.out.println`, once for each `ColorPoint` object for which you called a mutator method, each time with that `ColorPoint` object as its ONLY argument
  - call method `distFrom` at least three times, printing its returned result to the screen in a descriptive message, for at least:
    - a `ColorPoint` calling object with a `ColorPoint` as its argument
    - a `Point` calling object with a `ColorPoint` as its argument
    - a `ColorPoint` calling object with a `Point` as its argument
    - ...all of these SHOULD be possible;

Submit `Plottable.java` as well as your files `Point.java`, `ColorPoint.java`, and `DemoPoints.java`.

Problem 2

(This is a slightly-modified version of Week 4 Lab Exercise - Problem 4 that a few finished, but many did not. If you did finish it, you should submit it for this homework problem, also.)

Write a Java application `PickyDemo` that does at least ONE of the following OPTIONs listed (although you may choose to implement MORE than one of these, or ALL of these, or something else *in addition*, IF you would like:

- decide which option(s) below you are going to try -- and if your application does not get at least that many command-line arguments, it should complain in a message to the screen and exit if it does not get at least that many
  - what if it gets MORE than the number of your chosen option(s)? You get to decide if your application will simply ignore them, or if it will complain and exit, or if it will do something of your choice with them.
- OPTION ONE: IF it gets *one* command line argument:
  - (If you choose this option, make sure that a copy of your Week 4 Lab Exercise - Problem 1 version of `GameDie.java` is also in your Homework 3 folder.)
  - it should try to create a `GameDie` with that number of sides.
  - BUT: if that command-line argument cannot be treated as an `int`, it should use a `try-catch` block to call `GameDie`'s no-argument constructor instead of halting with an error
  - And, it should call `System.out.println` with the resulting `GameDie` object (and you can add an additional message before or after that if you would like)
- OPTION TWO: IF it gets *two* command line arguments:
  - it should try to create a `Point` with that x-coordinate and y-coordinate and a name of whatever constant name you'd like
  - BUT: if either of those command-line arguments cannot be treated as a `double`, it should `use try-catch` block(s) to call a `Point` constructor alternatively to give it x- and y-coordinates of 0 and 0 (and whatever constant name you'd like) instead of halting with an error.
  - And, it should call `System.out.println` with the resulting `Point` object (and you can add an additional message before or after that if you would like)
- OPTION THREE: IF it gets *three* command line arguments:
  - it should try to create a `Point` with that x-coordinate and y-coordinate and that name
  - BUT: if either of the first two command-line arguments cannot be treated as a `double`, it should use `try-catch` block(s) to call a `Point` constructor alternatively to give it x- and y-coordinates of 0 and 0 along with their 3rd command-line argument instead of halting with an error.
  - And, it should call `System.out.println` with the resulting `Point` object (and you can add an additional message before or after that if you would like)
- OPTION FOUR: IF it gets *four* command line arguments:

  - it should try to create a `ColorPoint` with that x-coordinate and y-coordinate and name and color
  - BUT: if either of the first two command-line arguments cannot be treated as a `double`, it should use `try-catch` block(s) to call a `ColorPoint` constructor alternatively to give it x- and y-coordinates of 0 and 0 along with their 3rd and 4th command-line arguments instead of halting with an error.
  - And, it should call `System.out.println` with the resulting `ColorPoint` object (and you can add an additional message before or after that if you would like)

Submit your `PickyDemo.java`, and if it is used in your option(s) selected, your `GameDie.java`. (You will already have submitted `Plottable.java` and your `Point.java` and `ColorPoint.java` as part of Problem 1.

Problem 3

Consider a cooking school, named Cookery. It has a mailing list. Any person can be on the mailing list. Design and implement a public class `CookeryFan` that meets the following criteria:

- it has private data fields for the person's first name, last name, preferred e-mail address, a four-digit year they were added to the mailing list, and whether they are willing to receive fund-raising e-mails in addition to information e-mails about school events
- it has at least one constructor, a 5-parameter constructor that allows initial values of all of these data fields to be initialized
- it has an appropriate accessor/getter method for all five data fields
- it has an appropriate modifier/setter method for all of the data fields except the year one was added to the mailing list -- that data field cannot be changed by users of this class
- it overrides the `toString` method to give an appropriate `String` depiction of a `CookeryFan` instance
- add at least one additional method of your choice to this class as well
  - (For example, perhaps a method that produces a `String` summarizing certain of a fan's information,
  - or a method that returns approximately how many years a fan has been on the mailing list)

Submit your resulting `CookeryFan.java`.

Problem 4

Write a Java application in `CookeryFanPlay.java` that exercises your `CookeryFan` class from Problem 3. It should meet at least the following criteria:

- It should create at least two `CookeryFan` instances of your choice.
- It should call `System.out.println` for each of these `CookeryFan` objects (and you can add an additional message before or after those if you would like).

- It should print to standard output a blank line and then a message saying that it is going to call all five of `CookeryFan`'s accessor/getter methods for one of the `CookeryFan` instances, and then print to standard output the results of doing so.
- It should print to standard output a blank line and then a message saying that it is going to call all four of `CookeryFan`'s modifier/setter methods for one of the `CookeryFan` instances, and then it should do so,

  and then call `System.out.println` with that `CookeryFan` object (to show how its state has changed as a result).
- It should print to standard output a blank line and then a message saying that it is going to call your additional method(s) for at least one of your `CookeryFan` instances, and then print to the screen something indicating the result of that call/those calls to your additional method(s).

Submit your resulting `CookeryFanPlay.java`.

Problem 5

It turns out that some of the people on the Cookery cooking school mailing list are also students at the school. Design and implement a class `CookeryStudent` that is a subclass of `CookeryFan`, with the following additional characteristics:

- a `CookeryStudent` instance should also have private data fields for a GPA (based on the usual 4-point scale), and a culinary specialty. Include an appropriate accessor/getter method and modifier/setter method for each of these two additional data fields.
- include a 7-parameter constructor expecting the appropriate initial values for the desired new `CookeryStudent` object's first name, last name, preferred e-mail address, 4-digit year added to the mailing list, whether they are willing to receive fund-raising e-mails, 4-point-scale-based GPA, and culinary specialty.
- include a method `onHonorRoll`, which determines whether the student's GPA is greater than or equal to 3.5. (Use a named constant appropriately for this "honor roll boundary".)
- override `toString` appropriately for this class, including GPA and culinary specialty in the resulting `String` (along with first name, last name, preferred e-mail address, 4-digit year added to the mailing list, and whether they are willing to receive fund-raising e-mails).

Submit your resulting `CookeryStudent.java`.

Problem 6

Write a Java application in `CookeryStudentPlay.java` that exercises your `CookeryStudent` class from Problem 5. It should meet at least the following criteria:

- It should create at least three `CookeryStudent` instances of your choice, except one should have a GPA less than 3.5, one a GPA equal to 3.5, and the other a GPA greater than 3.5.
- It should call `System.out.println` for each of these `CookeryStudent` objects (and you can add an additional message before or after those if you would like).

- It should print to standard output a blank line and then a message saying that it is going to call all of the accessor/getter methods for one of the `CookeryStudent` instances, and then print to standard output the result of calling all **seven** of them (the five inherited from `CookeryFan` and the two new ones added to `CookeryStudent`).
- It should print to standard output a blank line and then a message saying that it is going to call `onHonorRoll` for each of the three `CookeryStudent` instances, and then decide on an appropriate way to output to the screen messages clearly giving the results of those calls.
- It should print to standard output a blank line and then a message saying that it is going to call all six of the modifier/setter methods for one of the `CookeryStudent` instances, and then it should do so (calling the four inherited from `CookeryFan` and the two new ones added to `CookeryStudent`), and then call `System.out.println` with that `CookeryStudent` object (to show how its state has changed as a result).

Submit your resulting `CookeryStudentPlay.java`.