Homework 07 - Can I get a Doctor?

Description

In this homework, you will equip a doctor to fend off illness. Currently, their treatment plan is missing, so you will properly code up the behavior for their treatment so that they can heal their patients! In doing so, you will learn about polymorphism, interfaces, and searching.

Purpose

Polymorphism is one of the most important topics in Object-Oriented Programming. In this homework, you will use interfaces, superclasses, and subclasses. In doing so, you will learn about polymorphism, as the superclass will take on several forms with different use cases. Furthermore, you will have a concrete use case for interfaces - you will see that we utilize the interface as a type rather than just simply a wrapper for a method. Finally, you will implement a proper `compareTo` method and use it in implementing a searching algorithm.

Instructions

Read the PDF carefully to ensure you have a thorough understanding of the assignment!

When you submit to Gradescope, know that hidden tests often depend on the visible ones. Therefore, make sure you are passing the visible tests when you submit your homework.

High Level Overview

- A `Patient` is someone who requires healing
- Anything with a `HealAbility` can heal
- A `Treatment` is something that can heal patients. There are two types of treatments.
  - A `ScheduledTreatment` is a treatment for doctors appointments. It was scheduled ahead of time. For example, an appointment with Stamps. Out of all the patients in a waiting room, the one who made the appointment will get treated.
  - A `EmergencyTreatment` is a treatment for emergencies. The most injured patient will get healed first. For example, an emergency room. Out of the patients waiting, the most injured one will get treated first.
- A `Doctor` applies treatments to patients
- The `Driver.java` is for testing purposes.

Provided Files (and code)

- `Doctor.java`
  - Has the `name` instance variable
  - Has 2 partly implemented constructors - you will complete these constructors.
  - Has a complete `toString` method
- `Patient.java`
  - Has the `name`, `patientID`, and `health` instance variables
  - Has 2 complete getters
  - Has a complete constructor
  - Has a complete `toString` method

- `Driver.java`
  - Has a basic test
  - This basic test is not comprehensive, so be sure to create your own
- `SortingMethod.txt`
  - This sorting method is implemented for you
  - You will paste this sorting method in the `HealAbility` interface you create

Directions

You will finish implementing `Doctor.java` and `Patient.java`. You will also create the interface `HealAbility.java` and the classes `Treatment.java`, `ScheduledTreatment.java`, and `EmergencyTreatment.java`.

The following classes are listed in the recommended order for implementation and modification.

`HealAbility.java`
Create a file containing the `HealAbility` interface. Remember, anything that implements the `HealAbility` interface is able to heal patients. Therefore, do the following in the `HealAbility` inteface:

- Create an abstract `performHeal` method that takes in an array of `Patient` and returns void
- Create an abstract `getHealAmount` method that has no parameters and returns an `int`
- Put the sorting method from `SortingMethod.txt` in the `HealAbility` interface.

`Treatment.java`
Create a file containing the `Treatment` class. Again, treatments are something that heal patients. `Treatment` implements the `HealAbility` interface. Because treatment is a very broad term, this class should be **abstract**. Do the following:

- Have `Treatment` implement the `HealAbility` interface.
- Create the `heal` instance variable. This is an `int` representing how much healing the `Treatment` gives.
- Create a 1-parameter constructor that takes in an `int` and assigns it to the `heal` instance variable.
- Implement the `toString` method
  - Properly overridden
  - Returns `"Treatment with {heal} heal"`, where `{heal}` is the `heal` variable in this class.

`Patient.java`
Modify the provided file. Remember, a patient is anyone who requires healing. The code contains the following already implemented for you in `Patient`: the fields `name`, `health`, `patientID`, a 3-argument constructor, an `equals` method, and a `toString` method. Note: a `Patient` object's health can go above what was initially set. Also, assume `patientID` will always be unique among instances of `Patient`. You should not modify these instance variables, constructors, or methods. However, do the following in `Patient`:

- Implement the generic version of the `Comparable` interface. Your code should only be able to compare Patient objects.
- Override the `compareTo` method
  - You should be able to put `@Override` on the line before the method header
  - Takes in a `Patient` object and returns an `int`, adhering to the API contract (Comparable Interface)
  - The method body should compare 2 `Patient` objects based on the `patientID` attribute. If the current `Patient` instance's `patientID` is less than the `Patient` passed in, return a negative number. If it is greater, return a positive number. If their `patientID` attributes are equal, return 0. These numbers tell us if one `Patient` object is "greater" or "less than" another, which is great for putting them in order! For example if we call `compareTo` on a `Patient` object with `health` 3 and `patientID` 8, and the parameter is a `Patient` object with `health` 3 and `patientID` 6, we should return a positve number. However, if it was a `Patient` object of `health` 4 and `patientID` 2 and the other one had `health` 5 and `patientID` 5, the `compareTo` method would return a negative number.

- public `increaseHealth` method
  - This method lets a `Patient` be healed
  - The method should take in a `HealAbility` object (since anything that can heal, can heal a `Patient`)
  - This method should not return anything
  - Increase the `Patient` object's `health` by the amount in the `HealAbility` object. (How can we get the health of a `HealAbility`?)
  - Should print `"Patient {name} has been healed by {heal} health points!"`, where `{name}` is the `Patient` object's `name` and `{heal}` is how much the patient's health increased.

`EmergencyTreatment.java`
Create a file containing the `EmergencyTreatment` class, a subclass of `Treatment`. `EmergencyTreatment` represents the treatment you get when you go to the ER and they handle the most severe cases first. `EmergencyTreatment` should do the following:

- `EmergencyTreatment` should *extend* the `Treatment` class
- Have a 1-parameter Constructor that takes in an `int` and assigns it to the `heal` instance variable. Since you don't have access to the class variables (they are in the superclass), pass these to the superclass's constructor using a special keyword.
- Have a method overriding `performHeal`
  - Takes in an array of `Patient` objects, returns nothing
  - Perform the **linear search** algorithm on the `Patient` array to find the `Patient` object with the least health
  - When you find the patient with least health, heal them for how much the treatment gives. (Recall that the `increaseHealth` method in Patient takes in a `HealAbility`: how can we pass in a `HealAbility` instance that corresponds to the current `EmergencyTreatment` instance?)
  - This method should account for empty arrays, but you can assume the array is **not** `null`

`ScheduledTreatment.java`
Create a file containing the `ScheduledTreatment` class, a subclass of `Treatment`. `ScheduledTreatment` represents the treatment you get when you make an appointment with the doctor and they handle the scheduled patient first. A `ScheduledTreatment` should do the following.

- `ScheduledTreatment` should *extend* the `Treatment` class.
- Have a `treatPatientID` instance variable. This should be an `int`. A `ScheduledTreatment` object will only treat its `treatPatientID`.
- 2-parameter Constructor. Takes in 2 `int`s and assigns to the `heal` instance variable and the `treatPatientID` instance variable respectively. Since you won't have access to a certain variable (it is in the superclass), pass it to the superclass's constructor using a special keyword.
- Method overriding `performHeal`
  - Takes in an array of `Patient` objects, returns nothing
  - Sort the array of `Patient` objects. Use the method provided in `HealAbility`. This will use the `compareTo` method in `Patient` in order to compare `Patient` objects. It will sort from "least" to "greatest" according to the `compareTo` method; once the sort is finished, if you call `patient1.compareTo(patient2)`, where `patient1` occurs in the array to the left of `patient2`, the result should be a negative number. Remember that binary search can only work if the array you're searching is already sorted!
  - Perform the **binary search** algorithm on the now-sorted `Patient` array to find the `Patient` with the same `patientID` as the `treatPatientID` instance variable in this `ScheduledTreatment` class.
  - If you find the patient, heal them for how much the treatment gives. (Recall that the `increaseHealth` method in Patient takes in a `HealAbility`: how can we pass in a `HealAbility` instance that corresponds to the current `ScheduledTreatment` instance?)
  - If you don't find the patient, don't heal anyone
  - This method should account for empty arrays, but you can assume the array is **not** `null`

`Doctor.java`
This file contains the `Doctor` class, which has attribues `name` and `treatment`, and `toString` method already implemented for you. There are also two partially completed constructors. You should not modify the provided instance variables or methods, but you need to do the following:

- Complete the two constructors
- Create an instance variable of type `Treatment`
- Create the following methods
  - A correctly named setter for the `Treatment` instance variable
  - A public `performTreatment` method
    - Takes in an array of `Patient` objects, and returns nothing
    - Should always print out `"{name} goes to heal their patients!"` (but without the quotes and `{name}` replaced with the `name` variable of this `Doctor` object)
    - Should call `performHeal` on the `Treatment` object in the class; this method should take in the array of `Patient` objects that was passed to the `performTreatment` method.

Feel free to add any additional tests to the `Driver.java` file - you will not be graded on how you test, just on your individual class and method functionality.

When the following code is run in `Driver.java`

```
Doctor doctorWho = new Doctor("Mildred", 10, 10);
Patient[] patientList = {
    new Patient("Rachna", 6, 8),
    new Patient("Tejas", 500, 100),
    new Patient("Will", 10, 10),
    new Patient("Aanya", 17, 11),
    new Patient("Julia", 10, 7)
};
System.out.println(doctorWho);
printArray(patientList);
doctorWho.performTreatment(patientList);
printArray(patientList);
```

The following should be the output to the console:

```
Doctor Mildred with treatment Treatment with 10 heal
[Patient 'Rachna' with 6 health and ID 8, Patient 'Tejas' with 500 health and ID 100, Patient
'Will' with 10 health and ID 10, Patient 'Aanya' with 17 health and ID 11, Patient 'Julia'
with 10 health and ID 7]
Mildred goes to heal their patients!
Patient Will has been healed by 10 health points!
[Patient 'Julia' with 10 health and ID 7, Patient 'Rachna' with 6 health and ID 8, Patient
'Will' with 20 health and ID 10, Patient 'Aanya' with 17 health and ID 11, Patient 'Tejas'
with 500 health and ID 100]
```

Note that this Driver class is *definitely* not comprehensive; you should be doing much more testing to ensure all your methods properly work!

Allowed Imports

To prevent trivialization of the assignment, no imports are allowed.

Checkstyle and Javadocs

You must run checkstyle on your submission. The checkstyle cap for this assignment is **20** points. If you don't have checkstyle yet, download it from Canvas -> Files/Resources. Place it in the same folder as the files you want checkstyled. Run checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the checkstyle cap mentioned above). The Java source files we provide contain no Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must javadoc your code.

Run the following to only check your javadocs.

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both javadocs and checkstyle.

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or breaks our auto grader. For that reason, do not use any of the following in your final submission: * var (the reserved keyword) * System.exit * Runtime.getRuntime.halt * Runtime.getRuntime.exit

Collaboration

Collaboration Statement

To ensure that you acknowledge collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Recall that comments are special lines in Java that begin with `//`.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `Doctor.java`

- `Patient.java`
- `Treatment.java`
- `ScheduledTreatment.java`
- `EmergencyTreatment.java`
- `HealAbility.java`

Make sure you see the message stating "HW07 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

1. Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2. Provide usage validation (e.g. forbidden imports, reserved keywords, etc)

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for all official clarifications
