Homework 02 - Superhero Database

Problem Description

You're a programmer at Marvel Studios, and there are way too many superheroes to keep track of. You're tasked with creating a superhero database representing all the many superheroes in an organized way that utilizes inheritance and hierarchies!

Solution Description

For this assignment, create files with the following names:

- `Superhero.java`
- `ChargedSuperhero.java`
- `FlyingSuperhero.java`

Superhero.java

Instance Variables:

- All superheroes have a code name! This variable should be called `name` and it should be a `String`.
- They should have a set amount of health points. Their health can only be represented through whole numbers. This variable should be called `health`**.**
- They do a set amount of attack damage in each attack! (More on this later) This attack can only be represented through whole numbers. This variable should be called `damage`**.**
- All instance variables should be **private.**

Constructors:

- Include a constructor that takes in `name, health, and damage` values in this order.
- Make sure the constructor parameters have the same type as their corresponding fields.

Methods:

- `toString` method
    - Must properly override `Object's toString`() method
    - Should return:

      "SUPERHERO
      Name: {name}
      Health: {health}
      Strength: {damage}"

    - Hint: Use new line characters and don't include the brackets!
- `equals` method
    - Should override `Object's equals()` method
    - Two superheroes are equal if they have the same `name, health`, and `damage`.
- `attack(Superhero other)` method:
    - Deals damage to the superhero that is taken in as a parameter.
    - Only deals damage if the attacking superhero has more than 0 health.

- The amount of damage dealt to the other superhero should be equal to `damage`
- Other's `health` should go down by the value of `damage` (this Superhero's `damage`, not the other Superhero's).
- Their `health` should not go below 0 because of the damage, if it does, set it to 0.
- This method should return nothing

ChargedSuperhero.java

Must inherit from the Superhero class

Instance Variables:

- `ChargedSuperheroes` have a unique suit that allows them to charge energy and unleash a burst of energy. Create a **private** variable that represents whether their suit is charged up or not named `charged`.

Constructors:

- Include a constructor that takes in `name, health, damage,` and `charged` in this order
  - Make sure to make the correct `super` call.
- Include a default no-arguments constructor that creates a `ChargedSuperhero` with the following default values:
  - `Name` is "Black Panther", `Health` is 30, `damage` is 10, Not charged
  - Use constructor chaining to call the other constructor in this class with default values.

Methods:

- `toString` method
  - Must properly override `Object's toString()` method
  - Should return:

    "SUPERHERO
    Name: {name}
    Health: {health}
    Strength: {damage}
    Suit Charged: {charged}"

  - Must utilize `Superhero's toString()` method to optimize code

- `equals method`
  - Should override `Object's equals()` method
  - Two `ChargedSuperhero` are equal if they have the same `name, health, damage,` and `charged` values.
  - Must reuse code using `Superhero's equals()` method
- `attack(Superhero other)` method:
  - Deals damage to the superhero that is taken in as a parameter.
  - Only deals damage if the attacking superhero has more than 0 health.
  - If the `ChargedSuperhero` called the method's suit is charged, deal damage to the superhero equal to twice the amount of the value of `damage` (this Superhero's `damage`, not the other Superhero's).
  - If the suit is not charged, deal damage equal to the value of `damage` (this Superhero's `damage`, not the other Superhero's).
  - Defending superhero's `health` should not go below 0. If it does, set it back to 0.

FlyingSuperhero.java

Must inherit from the Superhero class

Instance Variables:

- `FlyingSuperheroes` have a unique suit that allows them to fly. Create a **private** variable that represents whether they are in flight or not called `flying`

Constructors:

- Include a constructor that takes in `name, health, damage, flying` in this order
  - Make sure to use the correct `super` call
- Include a default no-arguments constructor that creates a `FlyingSuperhero` with the following default values:
  - `Name` is "Captain Marvel", `Health` is 40, `Damage` is 7, Not flying
  - Use constructor chaining to call the other constructor in this class with default values given above.

Methods:

- `toString` method
  - Must properly override `Object's toString()` method
  - Should return:

    "SUPERHERO
    Name: {name}
    Health: {health}
    Strength: {damage}
    Flying: {flying}"

  - Must utilize `Superhero's toString()` method to optimize code

- `equals` method
  - Should override `Object's equals()` method
  - Two `FlyingSuperhero` are equal if they have the same `name, health, damage`, and `flying` values.
  - Must reuse code using `Superhero's equals()` method
- `attack(Superhero other)` method:
  - Deals damage to the superhero that is taken in as a parameter.
  - Only deals damage if the attacking superhero has more than 0 health.
  - If `FlyingSuperhero` is flying, there is a 50% chance that `FlyingSuperhero` will deal damage to the superhero equal to 3x the amount of the value of damage. If chance fails, deal `damage` equal to the value of `damage` (in both cases, use this Superhero's `damage`, not the other Superhero's).
    - Use `Math.random()` to calculate this chance
  - If `FlyingSuperhero` is not flying, deal damage equal to the value of damage
  - Defending superhero's health should not go below 0. If it does, set it back to 0.

Tips and Tricks

- Test your code yourself! Create a driver class, then instances of your objects, then call your methods. Use print statements to see your output!
- When overriding a method, if you use the @Override tag, you don't need to javadoc that method!
- Unless specified, you are free to choose access modifiers for your variables. Just ensure they follow the rules of encapsulation!

Example Outputs

- superheroOne is a FlyingSuperhero with name: Captain Marvel, health: 40, damage: 7, flying: no
- superheroTwo is a ChargedSuperhero with name: Black Panther, health: 30, damage: 10, charged: no

```
System.out.println(superheroTwo);
  /* SUPERHERO
  Name: Black Panther
  Health: 30
  Strength: 10
  Suit Charged: false */
superheroOne.attack(superheroTwo);
System.out.println(superheroTwo);
  /* SUPERHERO
  Name: Black Panther
  Health: 23
  Strength: 10
  Suit Charged: false */
```

Allowed Imports

To prevent trivialization of the assignment, you may not import anything.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`

Checkstyle and Javadocs

We will not take off points for the hashCode checkstyle error for this homework.

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
```

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 //.

Allowed Collaboration

When completing homeworks and programming exercises for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks or programming exercises
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.

Examples of approved/disapproved collaboration:

- **Approved**: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"
- **Disapproved**: "Hey, it's 10:40 on Thursday... Can I see your code? I won't copy it directly I promise"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Turn-In Procedure

Submission

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

- Superhero.java
- ChargedSuperhero.java
- FlyingSuperhero.java

Make sure you see the message stating "HW02 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. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. 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 basic formatting and usage validation

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 "Allowed Imports" and “Restricted Features” to avoid losing points
- Check on Piazza for a note containing all official clarifications