Homework 04 – Monster Mash

Authors: Brittney, Aanya, Sooraj, Ryan, Jacob

Problem Description

It is time for Dracula's Monster Mash, the hottest monster party of the year! Dracula invites monsters from all over the world to compete in dance battles and have a spooktacular time! How does this involve you? Well, you were snooping around an abandoned castle (or at least you thought it was abandoned) and got caught by a trap. Dracula's trap, to be exact. He was planning to suck your blood, until as a last effort you announced that you were a great programmer! Dracula has been wanting to move some party aspects online this year because of Covid, so he's willing to spare your life if you create an online version of his party.

Solution Description

Your job is to make a `GuestList` class that can manage the dance battles for the evening. This is a party for Monsters, so we need you to create a `Monster` class to represent Dracula's invitees. You will also need to create a `DancingMonster` class for `Monsters` that will be competing in dance battles. You know an object is a dancer if its class implements the `Danceable` interface, which you will be also creating.

`Danceable.java`

This file contains the **interface** for any object that can dance. `Danceable` must be an interface.

- Methods
  - A `dance` method that takes no parameters and returns an `int`.
  - A `wonDance` method that takes no parameters and returns nothing.
  - You should not provide default implementations for any of the methods in this interface.

`Monster.java`

This file contains a concrete class Monster and must implement `Comparable` of type `Monster`.

- Variables
  - `String name`
    - This represents the name of the monster.
  - `int spookiness`
    - This represents the spookiness index of the monster.
  - All instance variables should be private
- Constructors
  - A constructor that takes values in the following order: `name` and `spookiness`
- Methods
  - `compareTo()`
    - Must properly override the `compareTo()` method from the `Comparable` interface.
    - This method takes in a `Monster` as a parameter.
    - One `Monster` is greater than another `Monster` if it has greater `spookiness`.
  - `toString()`
    - Returns a String of the following format:
      `"{name} has a spook rating of {spookiness}"`
      - ◊ The curly braces {} denote values that should be replaced with the corresponding variable(s), do not include the curly braces in your output.

    - Necessary getters and setters.

`DancingMonster.java`

This file represents a DancingMonster and must extend `Monster` and implement `Danceable`.

- Variables
    - `String danceMove`
        - This represents the name of the monster's signature dance move.
    - `int dancesWon`
        - This represents the number of dance battles the monster has won.
    - All instance variables should be private
- Constructors
    - A constructor that takes values in the following order: `name`, `spookiness`, `danceMove`.
        - `dancesWon` should be initialized to 0.
- Methods
    - `dance()`
        - Must properly override the `dance()` method from the `Danceable` interface.
        - This method should print the following with a new line at the end: `"{name} does the {danceMove}!"`
            - ◊ The curly braces {} denote values that should be replaced with the corresponding variable(s), do not include the curly braces in your output.
        - This method also returns a random integer in the range of [0, spookiness] (inclusive).
            - ◊ You <u>must</u> use `Math.random()` to generate this integer.
    - `wonDance()`
        - Must properly override the `wonDance()` method from the `Danceable` interface.
        - It should increment `dancesWon` by 1.
    - `toString()`
        - Returns a String of the following format: `"{name} has a spook rating of {spookiness} and has won {dancesWon} dances thus far"`
            - ◊ The curly braces {} denote values that should be replaced with the corresponding variable(s), do not include the curly braces in your output.
    - Necessary getters and setters.

`GuestList.java`

This file represents a guest list for a party.

- Variables
    - `ArrayList guests`
        - This represents the list of Monsters invited to the party
        - Only Monsters should be allowed to be added to the guest list
        - **Hint:** Don't forget imports!
    - `DancingMonster bestDancer`
        - This represents the best dancer according to the natural ordering of `Monster` objects
    - All instance variables should be private
- Constructors
    - A constructor that takes in an `ArrayList` of `Monsters`.

        - `guests` should be directly assigned the passed-in `ArrayList`
            - ◊ You can assume that the passed-in `ArrayList` will never be `null`
        - `bestDancer` should be initialized with the greatest `DancingMonster` according to the natural ordering for `Monster` objects.
            - ◊ If there is a tie, choose the first dancer in the list with the highest value.
            - ◊ If there are no `DancingMonster` objects in the list, set `bestDancer` to `null`
            - ◊ **Hint:** Take advantage of short-circuit evaluation to handle the case where `bestDancer` is not yet initialized and will error if you try to call methods on it
    - A no-args constructor
        - `guests` should be initialized with an empty ArrayList and `bestDancer` with `null`
- Methods
    - `addGuest()`
        - This method takes in a `Monster` and returns nothing.
            - ◊ You can assume that the passed-in `Monster` will never be `null`
        - The parameter value should be added to the end of `guests`
        - If the parameter is a `DancingMonster` and is *strictly greater than* the current `bestDancer` or if the parameter is a `DancingMonster` and `bestDancer` is null `bestDancer` should be updated.
    - `toString()`
        - Returns a string of the guests' names in the order in which they appear in `guests`, each separated by a comma and a space, in the following format: `"{monster1 name}, {monster2 name}, …, {monster3 name}"`
            - ◊ The curly braces {} denote values that should be replaced with the corresponding variable(s), do not include the curly braces in your output.
    - `danceBattle()`
        - This method takes in two `Monsters` as parameters and returns nothing.
            - ◊ You can assume that the passed-in `Monsters` will never be `null`
        - First, let the competing `Monsters` introduce themselves by printing out the `toString()` method of each `Monster` on separate lines.
        - If one of the two `Monsters` is not a `DancingMonster`, then the `DancingMonster` wins by default. If both `Monsters` are not `DancingMonsters`, then they both forfeit and there is a tie.
        - If both `Monsters` are `DancingMonsters`, then call each monster's `dance()` method and compare the returned ints.
            - ◊ The monster with the higher `dance()` return value should win the dance battle.
            - ◊ If the returned ints are equal then there is a tie.
        - After the battle, increment the winning `Monster`'s `dancesWon` and print out the winner as shown below with a newline at the end: `"{monster name} won the dance battle!"`
            - ◊ The curly braces {} denote values that should be replaced with the corresponding variable(s), do not include the curly braces in your output.
            - ◊ **Hint:** What method can we use to increment `dancesWon`?

    - If there is a tie, do not update either monster's `dancesWon` and print the following with a newline at the end:
      `"And this dance battle is a tie!"`

`MonsterMash.java`

This will be a driver file that you do not need to submit, but we recommend you write to test your homework. This is just a basic check, so please feel free to add to these test cases to more comprehensively test your code.

In the main method:

- Create a `GuestList` of `Monsters`.
- Print out a welcome message for the party and your `GuestList`.
- Call `danceBattle()` 3 times with different combinations of `Monsters`.
- Print out the `bestDancer` of the night

Clarifications and Example Output

Please refer to the pinned Piazza thread for HW04 Clarifications for any other necessary clarifications about the requirements outlined in this pdf. We may also post some example output that you can use to check your code against on the same Piazza post.

Allowed Imports

- `java.util.ArrayList`

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

You must run Checkstyle on your submission. The Checkstyle cap for this assignment is **11 points**. If you don't have Checkstyle yet, download it from Canvas -> Modules -> Checkstyle Resources (Section B & C) or General Information (Section D) -> checkstyle-8.28.jar. 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
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Collaboration

No collaboration is allowed on this assignment. See syllabus for more details.

In addition, 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:

- Danceable.java
- Monster.java
- DancingMonster.java
- GuestList.java

Please only submit the individual files—do not put them into a package or you will risk getting a zero on the assignment. Make sure you see the message stating "HW04 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:

- Prevent upload mistakes (e.g. non-compiling code)
- 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
- Run Checkstyle on your code to avoid losing points
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Feature Restrictions" to avoid losing points
- Check on Piazza for a note containing all official clarifications