Homework 04 – Trick or Treat

Authors: David, Eric
Topics: polymorphism, dynamic binding, interfaces, comparable

Problem Description

Halloween is coming, and the CS 1331 TA team has a treat for you! Or maybe a trick, depending on how well you like these homeworks.

Solution Description

You will write and submit TrickOrTreater.java to define a template for trick-or-treaters, Robbable.java to enable some trick-or-treaters to be robbed, Ghost.java and Witch.java as concrete trick-or-treaters, and HalloweenNight.java to see how the night plays out.

Notes:

1. All variables should be inaccessible from other classes and require an instance to be accessed through, unless specified otherwise.
2. All methods should be accessible from everywhere and must require an instance to be accessed through, unless specified otherwise.
3. Use constructor and method chaining whenever possible! Ensure that your constructor chaining helps you minimize code repetition and maximize code reuse!
4. Helper methods with appropriate visibility are optional.
5. Make sure to add Javadoc comments to your methods and classes!

TrickOrTreater.java

It should not be possible to instantiate TrickOrTreater. A TrickOrTreater should be comparable (***hint**: think interfaces*) to another TrickOrTreater.

Variables:

- `String name` – the trick-or-treater's name
  - Invalid values (null, empty, or blank) should default to "Charlie Brown".
  - If the input is not invalid, you can assume that it does not have leading nor trailing whitespace.
- `int age` – the trick-or-treater's age, in years
  - Must be in the inclusive range [0, 12]. Invalid values should default to 8.
- `int numCandy` – a count of the number of pieces of candy in this trick-or-treater's basket
  - Must never be negative. Invalid values should default to 0.

Constructors:

- A constructor that takes in `name`, `age`, and `numCandy`.

Methods:

- `trickOrTreat`
  - This method should be declared, but not implemented in this class.
  - Where implemented, it should take in no arguments, and return nothing.

- `gainCandy`
  - This method should only be visible to subclasses, and other classes belonging to the same package as TrickOrTreater.java.
  - Takes in an `int` representing the amount of candy to gain.
    - For any positive input, add that amount of candy to the trick-or-treater's candy basket.
  - Return nothing.
- `loseCandy`
  - This method should only be visible to subclasses, and other classes belonging to the same package as TrickOrTreater.java.
  - Takes in an `int` representing the amount of candy to lose.
    - For any positive input, deduct as much of that amount as possible from the trick-or-treater's candy basket, shy of resulting in a negative amount of candy.
  - Return the amount of candy that was successfully deducted.
- `toString`
  - Return a String that adheres to this format (single line, no leading/trailing whitespace, no whitespace preceding/following forward slashes, no quotation marks, no curly braces):

    `"{name}/{age}/{numCandy}"`

- `compareTo`
  - This method should properly override the `compareTo` method from the `Comparable`.
  - Returns an `int` as defined in the Comparable API.
  - You may assume that a null reference will not be passed in.
  - A trick-or-treater is considered greater than the other if it has a greater amount of candy. If both have the same amount of candy, the trick-or-treater of greater age is considered greater.
- Getter for `numCandy`.

Robbable.java

Some kinds of trick-or-treaters can be robbed. Robbable is a functional interface.

Methods:

- `beRobbed`
  - Not defined in Robbable.
  - Takes in no arguments.
  - Returns an `int` representing the number of candy lost during the robbery.

Ghost.java

A Ghost is a TrickOrTreater, and a mischievous one at that.

Variables:

- `int robberiesConducted` – the number of successful robberies conducted by the ghost
  - A robbery is successful if a Ghost gains candy because of it.

Constructors:

- A constructor that takes in `name`, `age`, and `numCandy`, and defaults `robberiesConducted` to 0.
- A no-argument constructor that constructs a Ghost named "Casper the Unfriendly Ghost" who is 12 years old, has no candy, and has conducted no robberies.

Methods:

- `trickOrTreat`
  - When a ghost trick-or-treats, the following is printed to console followed by a newline:

    ```
    "Boo! Trick or treat!"
    ```

  - Upon booing, a ghost acquires two new pieces of candy.
- `rob`
  - Takes in the `TrickOrTreater` to be robbed.
  - Under the cloak of anonymity, a dastardly ghost will rob any trick-or-treater that is robbable, taking some of that trick-or-treater's candy for itself.
  - Ensure that `robberiesConducted` maintains an accurate value.
  - Returns nothing.
- `toString`
  - Return a String that adheres to this format (single line, no leading/trailing whitespace, no whitespace preceding/following forward slashes, no quotation marks, no curly braces):

    ```
    "{name}/{age}/{numCandy}/{robberiesConducted}"
    ```

- `compareTo`
  - This method should properly override the `compareTo` method from `TrickOrTreater`.
  - You may assume that a null reference will not be passed in.
  - This method should usually adhere to the general case of comparing a ghost to any trick-or-treater.
  - If a ghost is compared to another ghost and the basic comparison leads to two ghosts tying one another, a further check is made. The ghost who has conducted more robberies is considered greater than the other.
- No getters or setters.

Witch.java

A Witch is a TrickOrTreater that can be robbed (***hint**: think interfaces*). During a robbery, a witch may be robbed of up to six pieces (not at random) of candy.

Variables:

- `String signatureCackle` – a witch's signature cackle
  - Invalid values (null, empty, or blank) should default to "Bwahaha".
  - If the input is not invalid, you can assume that it does not have leading nor trailing whitespace.

Constructors:

- A constructor that takes in `name`, `age`, `numCandy`, and `signatureCackle`.
- A no-argument constructor that constructs a Witch named "Maleficent" who is 7 years old, has no candy, and has a signature cackle of "Bwahaha".

Methods:

- `trickOrTreat`
  - When a witch trick-or-treats, the following is printed to console followed by a newline:

    ```
    "{signatureCackle}! I'll get you my pretty!"
    ```

  - Upon cackling, a witch acquires three new pieces of candy.
- `compareTo`
  - This method should properly override the `compareTo` method from `TrickOrTreater`.
  - You may assume that a null reference will not be passed in.
  - This method should usually adhere to the general case of comparing a witch to any trick-or-treater.
  - If a witch is compared to another witch and the basic comparison leads to two witches tying one another, a further check is made. The witch with the lengthier cackle is considered greater than the other.
- No getters or setters.

HalloweenNight.java

This is it, the big night! Everything comes together here.

Variables:

- `TrickOrTreater[] cryptKickerFive` – array of length 5
- `TrickOrTreater[] ghoulGang` – array of length 5

Constructors:

- A constructor that takes in arrays for `cryptKickerFive` and `ghoulGang` (in this order).
  - You may assume that each input array is not null, and that neither array has null entries. Further, you may assume that each input array is of the expected length. You may shallow copy these arrays.
  - **Note**: Even though this assignment guarantees each team will have five trick-or-treaters, do not hardcode the literal value 5 throughout this class. Write your code so that it would accommodate any pair of positive length arrays that share a length.

Methods:

- `toString`
  - To get a basic idea for the matchup, format a String representing HalloweenNight to this specification (no leading nor trailing whitespace, no angle brackets, no quotation marks, a single space following each colon and comma, and a single space preceding and following "versus"):

    ```
    "cryptKickerFive: <CKF0>, <CKF1>, <CKF2>, <CKF3>, <CKF4>
    versus ghoulGang: <GG0>, <GG1>, <GG2>, <GG3>, <GG4>"
    ```

    - Use the String representations defined for each trick-or-treater.

        - **Note**: CKF represents the cryptKickerFive and GG represents the ghoulGang. The digit following the initials represents the index in the corresponding array.
    - **Note:** It is acceptable to hardcode your format string to anticipate five members on each team for this method.
- `compareTeams`
    - Trick-or-treaters in each group could be off to a head start. Minding that, this method establishes rough criteria for which team begins at an advantage.
    - Compare trick-or-treaters at matching indices of both teams (e.g., 0 and 0). The team with a higher count of greater trick-or-treaters is at an advantage.
    - Print out the appropriate string based upon the conclusion followed by a newline:
        - `"cryptKickerFive is favored."`
        - `"ghoulGang is favored."`
        - `"Neither team is favored."`
- `battle`
    - This method takes in a positive `int` representing the winning candy threshold. A team's total candy count must be at least this amount to win.
        - If the input is nonpositive, default to 60.
    - The instructions below describe the actions that a team should perform in one "turn".
        - Each trick-or-treater on the team will trick-or-treat.
        - Following trick-or-treating, each ghost on the team will attempt to rob the trick-or-treater at the same index (e.g., 0 and 0) of the other team.
            - ***Hint*:** Think of an operator that checks if a given reference has an "is-a" relationship with a class or interface. Although this HW defines only two specific kinds of trick-or-treaters, design your code to hold up even if there were more than two kinds.
    - The battle should begin with the `cryptKickerFive` taking a turn.
    - Each team must take at least one turn.
    - The first team to meet the winning candy threshold after both teams take an equal number of turns wins. If both teams meet this threshold, they tie.
    - Print out the appropriate string based upon the results followed by a newline:
        - `"cryptKickerFive won!"`
        - `"ghoulGang won!"`
        - `"It is a tie!"`

Driver.java

This class will be used to test your code. You do not need to submit your Driver file. We have outlined some suggested tests in the `main` method section to help you get started.

- `main`
    - Create two teams comprised of unique instances of `TrickOrTreater`.
    - Create a `HalloweenNight` object using two valid teams.
    - Print each team.
    - Compare the teams.
    - Pit the teams against each other in battle.

Checkstyle

You must run Checkstyle on your submission (to learn more about Checkstyle, check out cs1331-style-guide.pdf under the Checkstyle Resources module on Canvas). **The Checkstyle cap for this assignment is 25 points.** This means there is a maximum point deduction of 25. If you don't have Checkstyle yet, download it from Canvas → Modules → Checkstyle Resources → checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. 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 number of points we would take off (limited by the Checkstyle cap). In future assignments 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.

Turn-In Procedure

Submission

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

- `TrickOrTreater.java`
- `Robbable.java`
- `Ghost.java`
- `Witch.java`
- `HalloweenNight.java`

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder tests are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

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

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond 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. **We reserve the right to hide any or all test cases, so you should make sure to test your code thoroughly against the assignment's requirements.**

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.

Burden of Testing

You are responsible for thoroughly testing your submission against the written requirements to ensure you have fulfilled the requirements of this assignment.

Be **very careful** to note the way in which text output is formatted and spelled. Minor discrepancies could result in failed autograder cases.

If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

Allowed Imports

To prevent trivialization of the assignment, you may not import any classes for this assignment.

Feature Restrictions

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

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

Collaboration

Only discussion of the assignment at a conceptual high level is allowed. You can discuss course concepts and assignments broadly; that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

The only code you may share are test cases written in a Driver class. If you choose to share your Driver class, they should be posted to the assignment discussion thread on the course discussion forum. We encourage you to write test cases and share them with your classmates, but we will not verify their correctness (i.e., use them at your own risk).

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 Ed Discussion for a note containing all official clarifications and sample outputs.**

It is expected that everyone will follow the Student-Faculty Expectations document and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment deemed by the professor to contain inappropriate offensive language or threats will get a zero.** You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.