CS1331 Homework 6

Introduction

In this assignment, you will be using Exceptions, File I/O, and interfaces.

Problem description

Welcome to the Wizarding World of Harry Potter! In this world there are two kinds of people: Muggles and Wizards. Wizards are those born with magical abilities while Muggles are normal people with no magical abilities.

You will be writing seven files for this homework. Let's review them:

House.java

This enum represents the four houses of Hogwarts, a top-notch school of wizardry. There are four values associated with this enum: GRYFFINDOR, HUFFLEPUFF, RAVENCLAW, and SLYTHERIN.

Magical.java

This interface is implemented by all magical beings **including Wizards**.

- `public abstract void castSpell(String spell)`: method, should remain abstract in this interface. When this method is implemented: If the spell taken in is "expecto patronum" the method should print "`name` casts `spell` and a `patronus` patronus appears!". For any other spell the method should print "`name` casts `spell`!".

Person.java

This class represents a general Person. **This class should not be instantiable.**

- `private String name`: variable representing the name of the Person.
- `public Person(String name)`: constructor that takes in and sets the name instance variable.
- `public String toString()`: method that returns "My name is `name`".
- `public String getName()`: method that returns `name`.

Muggle.java

This class represents a Muggle, a person with no magical ability. This is a subclass of Person.

- `private String occupation`: variable representing the occupation of the muggle.
- `public Muggle(String name, String occupation)`: constructor that takes in and sets the name and occupation of the Muggle.
- `public String toString()`: method that returns "My name is `name` and I am a(n) `occupation`.". This method should be written to maximize code reuse (avoid copy&pasting/repeating code).

Wizard.java

This class represents a Wizard, a person with magical abilities. This is a subclass of Person. A wizard is Magical and therefore must implement the interface Magical. **A wizard should also be Comparable to another wizard based on skill.**

- `private House house`: variable representing to which of the four houses this Wizard belongs.
- `private int skill`: variable representing the skill level of this Wizard.
- `private String patronus`: variable representing the patronus of this Wizard.
- `public Wizard(String name, House house, int skill, String patronus)`: constructor that takes in and sets the instance variables.
- `public void duel(Wizard w)`: method that prints "`name1` triumphed over `name2` in a duel!". The name of the wizard with the higher skill level should be `name1` and the other wizard's name should be `name2`. If the skill levels are equal then this method should print "`name3` tied with `name4` in a duel!" where `name3` is the name of **this** wizard and `name4` is the name of the wizard taken in as an argument.
- `public String toString()`: method that returns "My name is `name` and I am a wizard in the `house` house.". This method should be written to maximize code reuse (avoid copy&pasting/repeating code). The house should have only the first letter capitalized.
- properly overridden `equals` method that checks equality based on `skill` and `name`.

ImposterAlertException.java

This class represents a checked exception that is thrown when an imposter is detected in Hogwarts. It should have a constructor that takes in a String and properly stores the value of that String as its message.

Hogwarts.java

This class represents the infamous school of wizardry itself.

- `private ArrayList studentList`: variable representing all of the students of Hogwarts. This should remain an ArrayList of type Object.
- `public void sortingHat(String filePath)`: methods that takes in a String representing the path to a text file. The file contains a list of people in a specific format (see provided `people.txt`). Each line of the file will contain a name concatenated with the first letter of the house the person belongs to or an 'M' if they are a Muggle. If the person is a Muggle, an ImposterAlertException should be thrown. If the person is not a Muggle, the line will **always** be in this format: "HarryG,100,stag". This line would represent a Wizard with `name` "Harry", `House` GRYFFINDOR, `skill` 100, and `patronus` "stag". Every valid Wizard should be added to the `studentList`.
- `public void startCeremony(String filePath)`: method that takes in a String representing the path to a text file and uses it to call sortingHat. If sortingHat throws an ImposterAlertException, it must be handled in this method. In order to handle the exception, the line of the file that caused the exception must be removed (the rest of the file must remain intact). Once that line is removed, sortingHat should be attempted again. This method should **not** go through and remove all of the

lines containing Muggles at once.
- `public String toString()`: method that returns "Welcome to Hogwarts! Meet our students: " + the String representation of `studentList`.

> Note: You should not add any additional public methods to any class other than the ones listed above or those required for compilation. You may not change any of the listed method headers. You may use as many private helper methods as you wish and add additional variables as needed.

Testing

- You have been provided with two text files for testing your Hogwarts class and a Tester.java.
- The Tester will reset the provided text files each time before it runs.
- The output from running the Tester class should be:
```
Welcome to Hogwarts! Meet our students: [My name is Harry and I am a wizard in the Gryffindor house., My name is Ron and I am a wizard in the Gryffindor house., My name is Hermione and I am a wizard in the Gryffindor house.]
Welcome to Hogwarts! Meet our students: [My name is Harry and I am a wizard in the Gryffindor house., My name is Ron and I am a wizard in the Gryffindor house., My name is Hermione and I am a wizard in the Gryffindor house.]
```
  - Note that the contents of people.txt and people1.txt should be identical after running the Tester. people.txt should not change and one line should be removed from people1.txt.
- Running our Tester does not guarantee you a 100. We encourage you to write your own test cases and text files to fully test your homework files.

Javadocs

You will need to write Javadoc comments and watch for checkstyle errors with your submission.

- Every class should have a class level Javadoc that includes `@author <GT Username>` and `@version <version number>`.
- Every public method should have a Javadoc explaining what the method does and includes any of the following tags if applicable:
  - `@param <parameter name> <brief description of parameter>`
  - `@return <brief description of what is returned>`

See the CS 1331 Style Guide on Canvas for details.

Checkstyle

For each of your homework assignments we will run checkstyle and deduct one point for every checkstyle error, with the points deducted being capped.

For this homework the **checkstyle cap is 85**, meaning you can lose up to 85 points on this assignment due to style errors. This limit will increase with each homework.

- If you encounter trouble running checkstyle, check Piazza for a solution and/or ask a TA as soon as you can!
- You can run checkstyle on your code by using the jar file found on Canvas that includes xml configuration file specifying our checks. To check the style of your code run `java -jar checkstyle-6.2.2.jar *.java`.
- To check your Javadocs run `java -jar checkstyle-6.2.2.jar -j *.java`.
- Note that the command for checking code and the command for checking Javadocs are different. You will have to run both commands to fully test for style errors.
- Javadoc errors are the same as checkstyle errors, as in each one is worth a single point and they are counted towards the checkstyle cap.
- **You will be responsible for running checkstyle on *ALL* of your code.**
- Depending on your editor, you might be able to change some settings to make it easier to write style-compliant code. See the Customization Tips on Canvas for more information.

Collaboration

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

- What general strategies or algorithms you used to solve problems in the homework
- Parts of the homework specification 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:

OKAY: "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?"

BY NO MEANS OKAY: "Hey... the homework is due in like 20 minutes... Can I see your code? I *promise* won't copy it directly!"

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.

Submission

- Submit **ALL** of your Java files for this homework (you do not need to submit the Tester) as attachments to the `Homework 6` assignment on Canvas. You can submit as many times as you want, so feel free to submit as you make substantial progress on the homework. We only grade your **last** submission, meaning we will ignore any previous submissions.
- If you submit multiple times Canvas will append a number to your zip file (`Wizard.java` becomes `Wizard-1.java`). **Do not worry about this, we will fix the file name before compiling and running your code.**
- Non-compiling code will be given a score of 0. For this reason, we recommend submitting early and then confirming that you submitted ALL of the necessary files by re-downloading your file(s) and compiling/running them.
- Files that contain inappropriate language and/or profanity will receive a 0.

Good Luck! `\(°□°)/`