Homework 5 – Merlin's Messy Tower

Authors: Nathaniel Gregory, Sarang Desai, Rachna Sahasrabudhe, Allan Nguyen, Sumit Choudhury

Problem Description

This assignment will test your knowledge on interfaces, Comparable, sorting and searching with polymorphism.

Merlin's wizard tower is full of magical items, but over the years it has become cluttered, and Merlin cannot find what he needs anymore. You are Merlin's assistant, who needs to clean the tower and find Merlin the best magical items to help him hustle adventurers. Sadly, you are not magical, so you will have to use your coding skills to make up for it.

Your program will have a Chest class that contains MagicItem objects including Lamp, Carpet, and Wand. Those MagicItems can be invoked to use their abilities at the cost of power, and the Chest will sort and find items within the chest based on their type and power left. Make sure to follow good class design and **encapsulation (make your instance variables private or protected)!**

Solution Description

MagicItem.java

- This class is not concrete, only used to help build other classes.
- Should have the following attributes:
  - Each magic item should have an int variable `'power'` that represents the amount of magic left in the item.
  - Each magic item should also have a string variable `'name'` that represents the name of the object.
- Should have a constructor for MagicItem, which takes in a string `name` and int `power` in that order.
- MagicItem should have a public abstract method `invoke()` which returns nothing and takes in no parameters.
- Printing out a MagicItem should return `"<name> has <power> power left".`
- Should have an equals method that overrides Object's equals. Two MagicItems are equal if they have the same name and power.
- Should implement the `Comparable` interface with correct generics comparing two MagicItems. Only use the power within magic items to compare them.
  - Return positive if the MagicItem has greater power than the item it is being compared to.
  - Return negative if the MagicItem has less power than the item it is being compared to.
- Should have getters for name and power and a setter for power.

Rechargeable.java

- An interface which allows MagicItems to regain power.
- Should have a public method called `recharge()` which takes in an int and returns nothing. This method should add the inputted power back into the item.

Lamp.java

- Is a type of MagicItem.
- Has attributes:
  - boolean variable called `'genieHome'`, represents if a genie is within the lamp.
- Constructor for Lamp should take in name, power, and genieHome in that order.
- When a Lamp is invoked, it should check if the genie is Home. If the genie is not home, print `"Genie away, come back later"`. If the genie is home and there is more than 50 power in the lamp, then print `"Wish granted!"` and subtract 50 power. Otherwise, print `"Not enough power"`.

- Printing out a Lamp should return "`Lamp: <name> has <power> power left`"
- Equals method that overrides Object's equals. Two Lamps are equal if they have the same name, power, and genieHome.

Carpet.java

- Is a type of MagicItem.
- Implements Rechargeable.
- Has attributes:
    - double variable called '`height`', representing its current flying height in meters.
- Constructor for Carpet should take in name, power, and height in that order.
- When a Carpet is invoked, it should check that it has at least 10 power. If it does not, print "`Not enough power`". Otherwise, subtract 10 power and raise the carpet by 10 meters, and print "`We're <meters> meters off the ground!`".
- Should have a public method called `land()` that takes in nothing, sets the height back to zero, prints "`We landed`", and returns nothing.
- Printing out a Carpet should return "`Carpet: <name> has <power> power left`".
- Equals method that overrides Object's equals. Two Carpets are equal if they have the same name, power, and height.

Wand.java

- Is a type of MagicItem.
- Implements Rechargeable.
- Has attributes:
    - String variable called '`owner`', representing the true owner of the wand.
- Constructor for Wand should take in name, power, and owner in that order.
- When a Wand is invoked, it should ask "What is your name?". The user should input a name. If the user inputs a name that is different than the '`owner`' variable, print "`I'm sorry, you are not the true owner`". If they input the correct '`owner`' name and the wand has at least 25 power, print "`FIREBALL!`" and subtract 25 power. Otherwise, print "`Not enough power`".
- Printing out a Wand should return "`Wand: <name> has <power> power left`".
- Equals method that overrides Object's equals. Two Wands are equal if they have the same name, power, and owner.

Chest.java

- Has attributes:
    - an ArrayList of MagicItems called '`itemsInside`', representing the items inside the chest.
- Chest should have a constructor that takes in an ArrayList of MagicItems.
- Chest should also have a no-args constructor that sets '`itemsInside`' to an empty ArrayList.
- The public method `sortChest()` should take in nothing, sort the Chest's contents using **selection sort** and return nothing. After sorting, the most powerful items should be at the top of the Chest (front of the ArrayList). Must use compareTo().
- The public method `addItem()` should take in a MagicItem, place it inside the Chest, and return nothing.
- The public method `findItem()` should take in a MagicItem and return it. If the item doesn't exist, return null. Use **binary search** to implement. Assume the chest is sorted. Print out "`You found a <item>`" when finished. Must use compareTo().
- The public method `removeItem()` should take in a MagicItem and remove it from the Chest and return the item. If the item doesn't exist, return null. Implement using **binary search**. Assume the chest is sorted. Must use compareTo().

- The public method `removeJunk()` should take in nothing, remove the least powerful item in the chest and return it. Print out "`Removed the junk`" when finished. Assume the chest is unsorted. Must use compareTo().
- Use Chest.java as your driver class. In your **main** method, test each of the methods you have created in Chest at least once and invoke each type of MagicItem at least once.

Submitting

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

- `MagicItem.java`
- `Rechargable.java`
- `Lamp.java`
- `Carpet.java`
- `Wand.java`
- `Chest.java`

Make sure you see the message stating, "Homework 5 submitted successfully." We will only grade your last submission be sure to **submit *every file* each time you resubmit.**

When running Checkstyle locally, you will get an error that says "Definition of 'equals()' without corresponding definition of 'hashCode()'". Do not worry, we will not take points off for this error. Any other Checkstyle errors must be fixed.

Import Restrictions

You may only import ArrayList and Scanner.

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 Javadoc

You must run Checkstyle on your submission. (To learn more about Checkstyle, examine our course's external website Java Resources page under "CS 1331 Style Guide".) The Checkstyle cap for this assignment is **15 points**. If you don't have Checkstyle yet, download it from Canvas → files → 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 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

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. Only post code on Piazza in a **private** post.

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
- Make sure to test your program manually based on the assignment instructions and expected outputs.

The Gradescope autograder visible to you is **NOT** comprehensive. A few test cases will be hidden when you submit. This is done intentionally so that you learn to write your own test cases for your assignments.