Java Programming 2 – Lab Sheet 8

This Lab Sheet contains material based on the lectures up to and including the material on concurrency.

The deadline for Moodle submission of this lab exercise is 4:30pm on Thursday 19 November 2020.

Aims and objectives

- Using enumerated types
- Writing immutable classes
- Using the Streams API

Set up

1. Download **Laboratory8.zip** from Moodle.
2. Launch Eclipse as in previous labs (see the Laboratory 3 lab sheet for details)
3. In Eclipse, select **File** → **Import** … (Shortcut: **Alt-F, I**) to launch the import wizard, then choose **General** → **Existing Projects into Workspace** from the wizard and click **Next** (Shortcut: **Alt-N**).
4. Choose **Select archive file** (Shortcut: **Alt-A**), click **Browse** (Shortcut: **Alt-R**), go to the location where you downloaded `Laboratory8.zip`, and select that file to open.
5. You should now have one project listed in the Projects window: **Lab8**. Ensure that the checkboxes beside this project is selected (e.g., by pressing **Select All** (Shortcut: **Alt-S**) if it is not), and then press **Finish** (Shortcut: **Alt-F**).

Submission material

Again, this lab builds on the work we have done in previous labs. As part of the starter code, you have been provided with the classes from Lab 5[1] – this lab does not rely on any of the changes from Labs 6 or 7, and does not make use of any of the GUI code from either lab.

[1] Note that a **getHP()** method has been added to Monster.

Part 1: Making Type an enumerated type

Until now, we have used Strings to represent the types of Monsters and Moves. Your first task is to define a new enumerated type **Type** and update the relevant classes to use this type.

First, create a new enumerated type called **Type** (in the **monster** package) with the following five values:

NORMAL, FIRE, WATER, ELECTRIC, GRASS

You should create one instance method in **Type** with the following signature:

- **public double getEffectiveness(Type otherType)**

This method should reproduce the behaviour of the static **TypedItem.getEffectiveness()** method, but using **Type** objects instead of Strings. Note that this new method will only have one parameter, while the original version in **TypedItem** has two. The method should return the effectiveness of the current type (i.e., "this") against the other type. For example:

Type.GRASS.getEffectiveness (Type.WATER) should return 2.0.
Type.WATER.getEffectiveness (Type.GRASS) should return 0.5

Next, you need to update the **TypedItem** interface as follows:

- Remove the **isValidType()** and **getEffectiveness()** methods
- Change the signatures of **hasType** and **getTypes()** to use **Type** objects instead of Strings

Making the above changes to **TypedItem** will have broken the **Monster** and **Move** classes. The final step in this part is to update those two classes so that they support the new version of **TypedItem** instead of the old version.

The provided test cases will test that all of the above has been done correctly.

Don't forget to add appropriate documentation comments to your Type.java file.

Part 2: MonsterCollection

Your next task is to develop a new class to store a collection of monsters.

Basic MonsterCollection behaviour

Define a new class **MonsterCollection** in the **monster** package. This class should have a field to represent the set of monsters in the collection, and single constructor with the following signature to set the value of the field:

- **public MonsterCollection (Set<Monster> monsters)**

You should also define a getter with the following signature:

- **public Set<Monster> getMonsters()**

This class must be **immutable**: it should not be possible to modify the internal state in any way once an object of the class has been constructed. The test cases will test that this has been done correctly.

Adding methods to MonsterCollection

Your final task is to add a set of methods to **MonsterCollection** to access the collection in various ways. **All of these methods must be written using the Java Streams API** – you should not use any other control structures inside any of the methods.

- **public Monster chooseBattleMonster()**
  - This method should choose a Monster from the collection that is able to battle (i.e., one that is not fainted). If there are no such Monsters in the collection, it should return null.
- **public Monster getStrongestMonster()**
  - This method should return the Monster from the collection with the highest HP value. Recall that the default sorting order for **Monster** sorts by decreasing HP value. If there is no such Monster (i.e., if the collection is empty), this method should return null.
- **public double getAverageHP()**
  - This method should return the average HP of all Monsters in the collection. If the collection is empty, it should return 0.
- **public Set<Monster> getMonstersOfType (Type type)**
  - This method should return a Set consisting of all Monsters of the given type.

The test cases will test the behaviour of these methods.

Don't forget to add appropriate documentation comments to your MonsterCollection.java file.

Testing your code

The provided test cases in **TestLab8.java** will test the behaviour of all of the classes. Note that these test do not verify that you have used the Streams API inside MonsterCollection, so you will need to verify that for yourself.

How to submit

You should submit your work before the deadline no matter whether the programs are fully working or not. Before submission, make sure that your code is properly formatted (e.g., by using **Ctrl-Shift-F** to clean up the formatting), and also double check that your use of variable names, comments, etc is appropriate. **Do not forget to remove any "Put your code here" or "TODO" comments!**

When you are ready to submit, go to the JP2 moodle site. Click on **Laboratory 8 Submission**. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code – probably **…/eclipse-workspace/Lab8/src/monster** -- and drag only the *five* Java files **Move.java, Monster.java, TypedItem.java, Type.java, MonsterCollection.java** into the drag-and-drop area on the moodle submission page. **Your markers only want to read your java files, not your class files.** Then click the blue save changes button. Check the .java files are uploaded to the system. Then click **submit assignment** and fill in the non-plagiarism declaration. Your tutor will inspect your files and return feedback to you.

Outline Mark Scheme

Your tutor will mark your work and return you a score in the range "Excellent" (*****) to "Very poor" (*). Example scores might be:

5*: you completed the lab correctly with no bugs, and with correct coding style

4*: you completed the lab correctly, but with stylistic issues – or there are minor bugs in your submission, but no style problems

3*: there are more major bugs and/or major style problems, but you have made a good attempt at the lab

2*: you have made some attempt

1*: minimal effort

Possible (optional) extensions

Here are some additional things to try:

- Rewrite the methods of Move, Monster, and Trainer to use Streams where possible.
- Add an Effectiveness enumeration as well as the Type enumeration (possible values could be SUPER_EFFECTIVE, EFFECTIVE, NOT_VERY_EFFECTIVE). See what changes would be necessary to support this.
- Add **add()** and **remove()** methods to MonsterCollection while keeping it immutable (hint: these methods should return the new MonsterCollection rather than modifying the collection internally – think along the lines of **String.toUpperCase()**)
- Update the Lab 6 and/or 7 classes with the changes necessary to support the new version of the TypedItem interface