Java Programming 2 – Lab Sheet 5

This Lab Sheet is based on lectures up to and including the material on Collections, packages, and Comparable/equals()/hashCode().

The deadline for Moodle submission of this lab exercise is 4:30pm on Thursday 29 October 2020.

Aims and objectives

- Using packages
- Using Java Collections Framework classes and interfaces
- Implementing equals(), hashCode() and compareTo() methods

Set up

1. Download **Laboratory5.zip** from Moodle. **(This file will be provided after the tutorial on Friday 23 October, as it includes a sample solution to Lab 4.)**
2. Launch Eclipse as in the previous lab (see the Eclipse 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 `Laboratory5.zip`, and select that file to open.
5. You should now have one project listed in the Projects window: **Lab5**. 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

This exercise builds on the material that you submitted for Laboratory 4, so it might be worth referring back to your work on that lab before beginning this one.

In Lab 3, you developed Monster and Move classes; in Lab 4, you refactored that to include error checking and to have a full representation of type effectiveness. In this lab, you will create additional classes and methods that allow monsters to battle with one another. Specifically, you will modify the **Monster** class to support battling, and will create two new classes: a **Trainer** class, which represents a trainer who has a collection of Monsters, and a **Battle** class, which represents a situation where two trainers are battling with each other using their own Monster collections.

At the end of the lab sheet are some **optional** extensions to the basic functionality. These extensions are not required, and if you do attempt them, be sure that the classes you submit still support the basic functionality as described in this lab sheet.

Moving code into a package

As a first step, you must create a new package called **monster** and move all of the monster-related classes to this package. This includes **Monster**, **Move**, and **TypedItem**. To create the package in Eclipse, right-click on the **src** directory in your project, choose **New – Package**, and then type the name of the package. You can then drag the classes into the new package, or choose **Refactor – Move …** and specify the destination.

After all of these operations, you should have the following structure in your project:

- **TestLab5.java** in the default package
- All other provided classes are now in the **monster** package

Making Monsters battle-ready

The next step is to add fields and methods to Monsters to allow them to battle each other, as follows.

First, add a new field **hitPoints** to the **Monster** class of type **double**. Add a parameter to all of the **Monster** constructors to allow a value for **hitPoints** to be specified. The constructors should throw an **IllegalArgumentException** if the specified value for **hitPoints** is less than zero.

Then add the following **public** methods to **Monster**:

- **public boolean isFainted()**
  - Returns true if **hitPoints** is zero, and false if **hitPoints** is greater than zero
- **public void removeHP (double damage)**
  - Removes the given amount of hit points from this Monster. If hit points becomes negative, then it should be set to zero.
- **public void attack (Monster defender)**
  - Chooses the best move to use against the given defender (use **chooseMove**), and then uses that move to remove the hit points from the defender (use **removeHP** and **getEffectivePower**).

Overriding equals() and hashCode()

Next, add overridden versions of **equals()** and **hashCode()** to the **Monster** and **Move** classes.

- For **Monster**, equality should be based on the name, the type(s), and the move(s).
- For **Move,** equality should be based on the name, the type, and the power.

You can (and probably should) use Eclipse to automatically generate **equals()** and **hashCode()** methods: use the *Source* menu and then choose *Generate equals() and hashCode()*, and then choose the fields that should be used for the comparison and the methods will be generated.

Implementing Comparable

Also update the **Monster** class so that it implements **Comparable<Monster>**. The **Monster.compareTo** method should compare Monsters based only on their hit points: a Monster with higher hit points should be sorted **before** one with lower hit points. You might want to use **Double.compare()** in your implementation.

Trainer class

Next, create a new class in the **monster** package called **Trainer**. A Trainer should have a name and a way of storing and accessing a collection of Monsters, as well as methods to add and remove Monsters and to battle with them.

The **Trainer** class should provide the following **public** instance methods – it is up to you to choose the fields (be sure to make them all **private**!). You are free to add any other methods that you need to support this behaviour, but be sure that any extra methods are declared **private**.

- Constructor: **public Trainer (String name)**
  - This should initialise all of the fields
- **public String getName()**
  - Returns the trainer's name
- **public String toString()**:
  - You should override the built-in **toString()** method to produce a nice string representation of this Trainer. The details of the implementation are up to you: be sure to include the trainer name, and you can include the other properties if you want.
- **public boolean addMonster (Monster monster)**
  - Adds the given Monster to this trainer's collection. Returns **true** if the monster was not in the collection before (i.e., if the monster was successfully added), and **false** if the trainer already had that monster before it was added
- **public boolean removeMonster (Monster monster)**
  - Removes the given Monster from this trainer's collection. Returns **true** if the monster was in the collection before (i.e., if the monster was successfully removed), and **false** if the monster was not in the collection before the method was called.
- **public Monster chooseBattleMonster()**
  - Chooses and returns a Monster from this Trainer's collection that is capable of battle – that is, one that is not fainted. The only requirement is that the selected Monster should have non-zero hit points (use **isFainted()** to check). If the **Trainer** has no non-fainted **Monster**, then this method should return null..

doBattle method

Finally, you should create one additional **static** method in the **Trainer** class with the following signature:

- **public static Trainer doBattle(Trainer trainer1, Trainer trainer2)**:

This method should simulate a battle between the two Trainers. The trainers should take turns choosing a monster to attack or defend with, until one trainer has no more battle-ready monsters available. The return value should be the winning Trainer.

At a high level, I suggest that the body of **doBattle()** should resemble the following:

- First, set two local **Trainer** variables, **attacker** and **defender**. Initialise **attacker** to **trainer1** and **defender** to **trainer2**.
- Then loop as follows until a winner is found:
    - Choose a battling Monster m1 from the attacker
        - If m1 is null, stop the loop and return defender as the winner
    - Choose a battling Monster m2 from the defender
        - If m2 is null, stop the loop and return attacker as the winner
    - Use m1 to attack m2 (hint: use **Monster.attack()**)
    - Switch the roles of attacker and defender and continue the loop

Unit tests

This project includes a set of unit tests in the source file **TestLab5.java**. Please see the Lab 4 lab sheet for details on how to run the tests.

Recall that **the test cases may not test every single possibility** – just because your code passes all test cases does not mean that it is perfect (although if it fails a test case you do know that there is almost certainly a problem). Also, while you are testing your code, please **make sure that you do not modify the test cases** – we will be testing your code against the original test cases, so if you modify a test case, your code will likely not pass our tests. If your code is failing a test, you must modify your code to fix the failure rather than changing the tests.

You can also write your own class with a **main** method to test things out 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 5 Submission**. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code – probably **.../eclipse-workspace/Lab5/src/monster** -- and drag the *four* Java files **Monster.java, Move.java, TypedItem.java, Trainer.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 four .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 via moodle.

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 exercise correctly with no bugs, and with correct coding style

4*: you completed the exercise correctly, but with serious 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 exercise

2*: you have made some attempt at both classes

1*: minimal effort

Optional extensions

Once you have completed the basic functionality of this lab, here are some additional things to try. Note that these extensions will **not be assessed,** and please make sure not to submit any code with these extensions as you may fail some of the test cases.

- With the current implementation of **Monster.equals**, it is not possible for a Trainer to have two identical monsters in their inventory. Think about how to fix this.
- Another important aspect of the game that is not represented (yet!) is trading: Trainers should be able to exchange Monsters with each other. Think about how to implement this.
- Try making **chooseBattleMonster** more intelligent: e.g., you could try choosing the Monster with the best moveset, or the Monster with the most HP, rather than a random one.