Introduction to Object Orientation Programming 2: Lab Sheet 6

This Lab Sheet contains material based on Lectures 6.

The deadline for Moodle submission of this lab exercise is 12:00 on Thursday 7 November 2024.

Aims and objectives

- Implementing interfaces, particularly the **Comparable** interface
- Reading and writing files on the file system
- Further practice writing code with less concrete specifications and on finding solutions in the Java library.

Set up

1. Downloadand unzip **LB06_WK7_starter.zip** from Moodle.
2. Launch Visual Studio as in previous lab (see the LB03_WK4_24 lab sheet for details)
3. In Visual Studio, select **File → Open Folder**
4. Go to the location where you unzipped **LB06_WK7_24_starter**, and highlight that folder and click on **Select Folder**
5. You should now be able to see all of your files in the explorer (if not click on **View → Explorer)**
6. 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 **Monster** class (and its subclasses) and **Trainer** class from lab 5.[1]

Your tasks are as follows:

- To update the **Monster** class to ensure that monsters can be sorted, using the **Comparable** interface
- To update the **Trainer** class to allow **Trainer** objects to be saved to and written from files on the file system

Note that as part of implementing this, you may need to add other fields or methods to the **Monster** class – this is fine, as long as everything is properly documented and the test cases pass.

[1] I have added the following method to the **Trainer** class to help with testing:
public Collection<Monster> getMonsters()

Sorting Monsters

Using the **Comparable** interface, implement a comparison method for **Monster** objects that sorts as follows:

1. Monsters should be sorted in **decreasing order of hit points.**
2. If two monsters have the same hit points points, sort in **alphabetical order based on type**.

As a concrete example, consider the following list of **Monster** objects:

- **M1**: Fire monster, hp:150
- **M2**: Water monster, hp:150
- **M3:** Electric monster, hp:200
- **M4**: Electric monster, hp:100
- **M5**: Electric monster, hp:100

After sorting, the above Monsters should be in the either of the following orders:

- **M3, M1, M2, M4, M5**.
- **M3**, **M1, M2, M5, M4**

Both are equally valid because **M4** and **M5** are identical as far as the sorting procedure is concerned.

Saving and loading Trainer

This set of methods should allow **Trainer** objects to be saved to a file on the local file system and loaded from a previously saved file. The methods are as follows:

- **public void saveToFile (String filename) throws IOException** – this **instance** method should save the current **Trainer** to a file at the given location, or throw an appropriate exception if saving is not possible.
- **public static Trainer loadFromFile (String filename) throws IOException** – this **static** method should load a **Trainer** object from the given file (which is assumed to have been created with the **saveToFile()** method above) and return the loaded **Trainer**, or should throw an appropriate exception if loading is not possible.

It is up to you to define a format for the saved trainers – the only requirement is that the **loadFromFile()** method should be able to fully re-create a trainer from the information written by the **saveToFile()** method (i.e., all properties of the loaded trainer should be equal to those of the previous trainer).

Normally when doing this sort of thing in Java, a good option is the **java.io.ObjectInputStream/java.io.ObjectOutputStream** classes and the **Serializable** interface. However, for this lab, **you may not use those classes** – you must use a different method, for example writing a text representation of each **Monster** to a text file, and then parsing that text file to recreate the Monster object when it is read.

As part of your implementation, you might find it useful to **add additional properties and/or methods to the Monster**. This is fine.

Testing your code

As in the previous labs, a set of JUnit test cases are provided to check the fields and method signatures of your classes, in the file **test/java/PreValidation**– please see the lab sheet for LB04_WK5_24 for instructions on using the test cases. You can use the test cases to verify that your code methods and fields are as expected before submitting it. But note that **the test cases may not test if it behaves as expected** – 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).

If you want, as in previous labs, you can also write a class with a **main** method to test your code directly, but **you should not submit this file**.

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 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 IOOP2 moodle site. Click on **LB06_WK7_24_submission**. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code and drag only the two Java files **monster/Monster.java** and **trainer/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 .java file is 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 **A1** to **H**.