Java Programming 2 – Lab Sheet 5

This Lab Sheet contains material based on Lectures 9—10.

The deadline for Moodle submission of this lab exercise is 5pm on Thursday 25 October 2018.

Default login details: Username = matric + 1st initial of family name, password = last 8 digits of library barcode (please change your password if you haven't done so already!).

Aims and objectives

- Using code in packages
- Writing code in packages
- Using Java Collections Framework classes: List, Set, Map
- Writing class definitions where only the high-level behaviour is specified

Set up

1. Download **Laboratory5.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 `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 a Monster class; in Lab 4, you refactored that to be a set of classes. In this lab, you will create additional classes that allow monsters to battle with one another. Specifically, you will create a **Trainer** class, which represents a trainer who has a set of Monsters which can battle, and a **Battle** class, which represents a single battle between two trainers until one trainer has no monsters left that are not knocked out. The following sections describe the required behaviour of the two classes in more detail.

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.

Packages

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** and all its subclasses, along with **MonsterException**. 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. Also, create a new package called **battle** to contain the classes you will develop in this lab.

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

- **BattleRunner.java** in the default package
- All other provided classes are now in the **monster** package
- An empty **battle** package

You must not otherwise modify the Monster class and its subclasses, as we will be using our versions of those classes to run our tests.

Trainer class

Create a class in the **battle** package called **Trainer**. A Trainer has a name and a set of Monsters, as well as methods to manage the list of Monsters.

The **Trainer** class should provide the following **public** methods – it is up to you to choose the set of fields and any additional **private** methods that are needed to support this behaviour.

- Constructor: **public Trainer (String name)**
- **public String getName()**
    - Returns the trainer's name
- **public void addMonster (Monster monster)**
    - Adds the given Monster to this trainer's set
- **public Set<Monster> getMonsters()**
    - Returns the current set of Monsters
- **public Map<String, Set<Monster>> getMonstersByType()**
    - Returns a categorised set of all Monsters – see below for more on this method
- **public boolean canFight()**
    - Returns **true** if at least one of this trainer's monsters is not knocked out (i.e., has hit points > 0), and **false** if not
- **public Monster chooseAttackMonster()**
    - Returns the monster from the current trainer's set with the **highest attack points**. The monster must **not be knocked out**. If there is more than one monster with the same attack points, this method can return **any of them**. If there is no such monster, returns **null**.
- **public Monster chooseDefenseMonster()**
    - Returns the monster from the current trainer's set with the **highest hit points**. The monster must **not be knocked out**. If there is more than one monster with the same hit points, this method can return **any of them**. If there is no such monster, returns **null**.

More on getMonstersByType

This method is intended to return a Map, where the keys are the types, and the value for each key is the set of monsters of that type. So if a trainer has the following monsters: One **FireMonster m1**, and two **WaterMonsters m2** and **m3**, then this method should return a Map structured as follows:

- Fire ➔ { m1 }
- Water ➔ { m2, m3}

The test cases will provide examples of the expected output for this method.

Battle class

You should now create another class in the **battle** package, called **Battle**. Again, for this class I will specify only the required behaviour of the public methods – it is up to you to work out any fields and additional private methods that might be needed.

- Constructor: **public Battle (Trainer trainer1, Trainer trainer2)**
  - Initiates a battle between the two **Trainer**s
- **public Trainer doBattle()**
  - Runs a complete battle between the two Trainers and returns the winner, or returns **null** if the battle is a draw. Details of this method are given below.
- **public List<String> getBattleLog()**
  - Returns the complete list of events that occurred during the battle. Details of this method are given below.

More on doBattle()

The **doBattle()** method should run a complete battle between the two trainers. A battle consists of a series of turns, where each Trainer takes turns attacking the other. The battle is finished when one or the other Trainer has no more monsters available (hint: use **Trainer.canFight()** to check this).

Each turn proceeds as follows:

1. The attacking trainer chooses a monster to attack with (use **Trainer.chooseAttackMonster()**)
2. The defending trainer chooses a monster to defend with (use **Trainer.chooseDefenseMonster()**)
3. The attacking monster attacks the defending monster (use **Monster.attack()**)
4. The roles of attacker and defender then switch, and the battle continues.

The first attacker should be **trainer1**.

When the battle finishes, the return value should be computed as follows:

- If neither Trainer can fight, return **null** – the battle is a draw
- Otherwise, return the Trainer for whom **canFight()** is still true

More on getBattleLog()

The **battle log** should contain all important events that happen during the battle. For example, for every turn in the battle, the log should indicate:

- Which trainer is attacking and which is defending

- The selected attacking and defending monsters
- The state of each trainer's monsters after the attack

The format of the log entries is up to you, but you should be sure to include a String recording at least all of the above information. Please see the end of this lab sheet for sample output from my implementation. You can choose a different output format, but this should give some idea of the expected content.

BattleRunner.java

The provided file **BattleRunner.java** includes a **main** method that creates two **Trainer**s and runs a battle between them. You can freely modify this file to run your code once it is completed – note that this will not test every method thoroughly, so you should also be sure to run the JUnit test cases. The sample output from the default version of **BattleRunner** is given at the end of the lab sheet.

Unit tests

The JUnit tests will be provided over the weekend.

This project includes a set of unit tests in the source file **test/TestBattle.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.

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 **H:\eclipse-workspace\Lab5\src\battle** -- and drag only the *two* Java files **Battle.java** and **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 two .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 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** – so please make sure that if you make any changes to the code, you still support the basic functionality.

- You could enhance **chooseAttackMonster()** and/or **chooseDefenseMonster()** to choose a monster that is particularly powerful against the opponent’s monster(s).
- You could add the ability to “revive” knocked-out monsters as part of the battle – you would need to come up with rules for how reviving works and when it is possible.
- You could improve the logging, for example by defining a **LogEvent** class and possibly subclasses to represent different events during the battle

Sample BattleRunner output

```
Winner is: Serena
Battle log:
- Battle between Ash and Serena
- Trainer 1 monsters: [(type:Electric, hp:200, ap:100, wk:[])]
- Trainer 2 monsters: [(type:Fire, hp:100, ap:50, wk:[Water]), (type:Water,
hp:100, ap:50, wk:[Fire, Electric])]
- Ash attacks Serena
- Ash chooses (type:Electric, hp:200, ap:100, wk:[])
- Serena chooses (type:Fire, hp:100, ap:50, wk:[Water])
- (type:Electric, hp:200, ap:100, wk:[]) attacks (type:Fire, hp:100, ap:50,
wk:[Water])
- Attacker monsters now: [(type:Electric, hp:190, ap:100, wk:[])]
- Defender monsters now: [(type:Fire, hp:100, ap:50, wk:[Water]),
(type:Water, hp:100, ap:50, wk:[Fire, Electric])]
- Serena attacks Ash
- Serena chooses (type:Fire, hp:100, ap:50, wk:[Water])
- Ash chooses (type:Electric, hp:190, ap:100, wk:[])
- (type:Fire, hp:100, ap:50, wk:[Water]) attacks (type:Electric, hp:190,
ap:100, wk:[])
- Attacker monsters now: [(type:Fire, hp:100, ap:50, wk:[Water]),
(type:Water, hp:100, ap:50, wk:[Fire, Electric])]
- Defender monsters now: [(type:Electric, hp:140, ap:100, wk:[])]
- Ash attacks Serena
- Ash chooses (type:Electric, hp:140, ap:100, wk:[])
- Serena chooses (type:Fire, hp:100, ap:50, wk:[Water])
- (type:Electric, hp:140, ap:100, wk:[]) attacks (type:Fire, hp:100, ap:50,
wk:[Water])
- Attacker monsters now: [(type:Electric, hp:140, ap:100, wk:[])]
- Defender monsters now: [(type:Fire, hp:0, ap:50, wk:[Water]),
(type:Water, hp:100, ap:50, wk:[Fire, Electric])]
- Serena attacks Ash
- Serena chooses (type:Water, hp:100, ap:50, wk:[Fire, Electric])
- Ash chooses (type:Electric, hp:140, ap:100, wk:[])
- (type:Water, hp:100, ap:50, wk:[Fire, Electric]) attacks (type:Electric,
hp:140, ap:100, wk:[])
- Attacker monsters now: [(type:Fire, hp:0, ap:50, wk:[Water]),
(type:Water, hp:100, ap:50, wk:[Fire, Electric])]
- Defender monsters now: [(type:Electric, hp:90, ap:100, wk:[])]
- Ash attacks Serena
- Ash chooses (type:Electric, hp:90, ap:100, wk:[])
- Serena chooses (type:Water, hp:100, ap:50, wk:[Fire, Electric])
- (type:Electric, hp:90, ap:100, wk:[]) attacks (type:Water, hp:100, ap:50,
wk:[Fire, Electric])
- Attacker monsters now: [(type:Electric, hp:80, ap:100, wk:[])]
- Defender monsters now: [(type:Fire, hp:0, ap:50, wk:[Water]),
(type:Water, hp:100, ap:50, wk:[Fire, Electric])]
- Serena attacks Ash
- Serena chooses (type:Water, hp:100, ap:50, wk:[Fire, Electric])
- Ash chooses (type:Electric, hp:80, ap:100, wk:[])
- (type:Water, hp:100, ap:50, wk:[Fire, Electric]) attacks (type:Electric,
hp:80, ap:100, wk:[])
- Attacker monsters now: [(type:Fire, hp:0, ap:50, wk:[Water]),
(type:Water, hp:100, ap:50, wk:[Fire, Electric])]
- Defender monsters now: [(type:Electric, hp:30, ap:100, wk:[])]
- Ash attacks Serena
- Ash chooses (type:Electric, hp:30, ap:100, wk:[])
- Serena chooses (type:Water, hp:100, ap:50, wk:[Fire, Electric])
- (type:Electric, hp:30, ap:100, wk:[]) attacks (type:Water, hp:100, ap:50,
wk:[Fire, Electric])
```

```
- Attacker monsters now: [(type:Electric, hp:20, ap:100, wk:[])]
- Defender monsters now: [(type:Fire, hp:0, ap:50, wk:[Water]),
(type:Water, hp:100, ap:50, wk:[Fire, Electric])]
- Serena attacks Ash
- Serena chooses (type:Water, hp:100, ap:50, wk:[Fire, Electric])
- Ash chooses (type:Electric, hp:20, ap:100, wk:[])
- (type:Water, hp:100, ap:50, wk:[Fire, Electric]) attacks (type:Electric,
hp:20, ap:100, wk:[])
- Attacker monsters now: [(type:Fire, hp:0, ap:50, wk:[Water]),
(type:Water, hp:100, ap:50, wk:[Fire, Electric])]
- Defender monsters now: [(type:Electric, hp:0, ap:100, wk:[])]
```