Java Programming 2 – Lab Sheet 7

This Lab Sheet contains material based on Lectures 13—14.

The deadline for Moodle submission of this lab exercise is 5pm on Thursday 8 November 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

- Making use of enumerated types
- Making use of the Java streams API
- Additional practice with inheritance and implementing interfaces
- Reading and understanding the documentation of built-in Java library classes
- Implementing **equals()** and **hashCode()** methods

Set up

1. Launch Eclipse as in previous labs (see the Laboratory 3 lab sheet for details)
2. 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**).
3. Choose **Select archive file** (Shortcut: **Alt-A**), click **Browse** (Shortcut: **Alt-R**), go to the location where you downloaded `Laboratory7.zip`, and select that file to open.
4. You should now have one project listed in the Projects window: **Lab7**. 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

In this lab, we will again be working with **Monsters** and **Trainers** – however, several aspects of the class structure will be modified for this lab. The following summarises the changes we will be making this time around.

- The **Monster** class will be made concrete (i.e., non-abstract), and the **Monster** subclasses will be removed
- The type and weaknesses of a **Monster** will be stored in an enumeration called **Type** rather than as Strings.
- In addition, a **name** field will be added to the **Monster** class.
- The method of choosing attack and defense monsters in **Trainer** will be modified:
    - An interface called **MonsterChooser** will be created including the two monster-choosing methods
    - Two concrete implementations of **MonsterChooser** will be created: **ComputerMonsterChooser**, which chooses **Monster**s automatically as before, and

HumanMonsterChooser, which interactively prompts the user for a choice during battle.
    - As part of using a **Trainer** in battle, its **MonsterChooser** must be specified
- In addition, **equals()** and **hashCode()** methods will also be added to the **Trainer** and **Monster** classes

Type

First, you must define an enumerated type called **Type** in the **monster** package. The enumeration should have the following values, in this order: **FIRE, WATER, ELECTRIC, GRASS**. In addition to these four values, you should also define an instance method **isWeakAgainst(Type otherType)** inside **Type** that checks whether the given type is weak against the other type. Weaknesses are defined as follows:

- **FIRE** is weak against { **FIRE**, **WATER** }
- **WATER** is weak against { **WATER**, **GRASS** }
- **ELECTRIC** is weak against { **ELECTRIC**, **GRASS** }
- **GRASS** is weak against { **FIRE**, **GRASS** }

Changes to Monster

The initial **Lab7** code includes a sample solution to Laboratory 6. You must make the following changes to **Monster**:

Changes to the fields and related methods

- Remove the **weaknesses** field
- Change the **type** field to be a **Type** instead of a **String** and change the getter method
- Add a new field **name** of type **String**, and add a getter method
- Change the constructor to have the following signature:
    - `public Monster(String name, Type type, int hitPoints, int attackPoints)`
- Modify the **toString()** method to reflect the new fields

Changes to dodge()

- Make the **Monster** class concrete (i.e., remove the **abstract** modifier)
- Implement the **dodge()** method directly in **Monster** as follows:
    - A **FIRE** monster should dodge every other turn (as before)
    - A **WATER** monster should dodge if the HP is >= 100 (as before)
    - **ELECTRIC** and **GRASS** monsters will never dodge

equals() and hashCode()

- Finally, you must add appropriate **equals()** and **hashCode()** methods to **Monster** – equality should be defined by having all fields identical. You can auto-generate the methods in Eclipse if you wish, or can write them yourselves.

Changes to Trainer

You will make several changes to the **Trainer** class – some to reflect the changes in **Monster**, and some to change the way **Monster**s are selected.

Changes to saving/loading

- You should modify the **loadFromFile()** and **saveToFile()** implementations to reflect the changes in **Monster**. Be sure that all fields are saved and loaded properly in your modified version.

equals() and hashCode()

Implement appropriate **equals()** and **hashCode()** methods in the **Trainer** class – equality should be defined based on the trainer's name and the set of monsters.

Choosing Monsters

First, define an interface **MonsterChooser** in the **battle** package. It should include the following two methods:

- `Monster chooseAttackMonster(Set<Monster> monsters);`
- `Monster chooserDefenseMonster(Set<Monster> monsters);`

Also modify the **Trainer** class to use **MonsterChooser** as follows:

- Add a **public** method **setMonsterChooser(MonsterChooser monsterChooser)** and a field inside the **Trainer** class to store the chooser
- Modify the **chooseAttackMonster** and **chooseDefenseMonster** methods inside **Trainer** to use the **MonsterChooser** instead of the current implementations
    - Be sure to save the original implementations of both methods so that you can re-use them in other classes later on!

HumanMonsterChooser

This class represents a human playing the monster battling game, who should be prompted to choose **Monsters** interactively during the battle. Implement a class **HumanMonsterChooser** in the **battle** package that implements the **MonsterChooser** interface as follows:

- The constructor should take a **java.io.Scanner** instance that will be used to prompt the user to enter their guess. The **Scanner** should be stored in a field to allow the **chooseAttackMonster()** and **chooseDefenseMonster()** methods to operate as specified below.
- The **chooseAttackMonster()** and **chooseDefenseMonster()** methods should both proceed as follows:
    - Show the user the list of non-knocked-out **Monster**s in the given **Set**
    - Prompt the user to choose one of the **Monster**s by number. You can use **Scanner.nextInt()** to get the user's input. You should continue to prompt the user until a valid **Monster** is selected, and then should return that **Monster** as a result.

Full documentation of the **java.io.Scanner** class is available at https://docs.oracle.com/javase/10/docs/api/java/util/Scanner.html. Sample behaviour of this method is shown at the end of the lab sheet.

ComputerMonsterChooser

This class represents a computer playing the battling game. Implement a class **ComputerMonsterChooser** in the **battle** package that implements the **MonsterChooser** interface as follows:

- Both of the methods can essentially be copied from the previous implementation in **Trainer** (this is why you were suggested to save the original implementations above!)
- But see below for some required modifications

Using the Java Streams API

For full marks, **you must modify the implementation of the following methods to use the Java Streams API:**

- `ComputerMonsterChooser.chooseDefenseMonster()`
- `Trainer.canFight()`

Some stream operators you might want to use include **filter()**, **sort()**, **findFirst()**. Other hints:

- To check whether a Stream is empty or not, use **findAny().isPresent()**
- To return the first element in a Stream, or to return null if the Stream is empty, use **findFirst().orElse(null)**

Full documentation of the operations you can call on a Stream is available at https://docs.oracle.com/javase/10/docs/api/java/util/stream/Stream.html

Testing your code

The file **BattleMain.java** contains a class with a **main** method that you can use to test your code – sample output from that method on my version is included at the end of this lab sheet.

As in the previous labs, a set of JUnit test cases will also be provided to check the behaviour of your classes – please see the lab sheet for Lab 3 for instructions on using the test cases. You can use the test cases to verify that your code behaves as expected before submitting it. But note 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). In particular, the test cases will not verify that you have used the Streams API in the above methods (although they will verify correctness of your implementations).

Possible extensions

If you want to try to do a bit more with the game, here are some things you could try:

- Adding the full set of monster types and weaknesses – e.g., see https://pokemondb.net/type for a comprehensive list
- Allowing Monsters to have more than one type, and working out how that interacts with strengths/weaknesses
- Writing more sophisticated versions of **chooseAttackMonster()** and **chooseDefenseMonster()** that take into account the details of the opponent's monsters

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 7 Submission**. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code – probably **H:\eclipse-workspace\Lab7\src** -- and drag only the *six* Java files **Type.java, Monster.java, Trainer.java, MonsterChooser.java, HumanMonsterChooser.java,** and **ComputerMonsterChooser.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 "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

Sample output

The following is the sample output of my implementation when I run **BattleMain.java**. User input is shown in green. I have also highlighted (in ***bold italics***) the expected behaviour when the user does not correctly choose a monster.

```
Battle between Ash and Kiawe
Trainer 1 monsters: [Pikachu(type:ELECTRIC, hp:35, ap:55),
Litten(type:FIRE, hp:45, ap:65), Rowlet(type:GRASS, hp:68, ap:55)]
Trainer 2 monsters: [Turtonator(type:FIRE, hp:60, ap:78), Alolan
Marowak(type:FIRE, hp:60, ap:80)]
Ash attacks Kiawe
Choosing an ATTACKING monster
0:Pikachu(type:ELECTRIC, hp:35, ap:55)
1:Litten(type:FIRE, hp:45, ap:65)
2:Rowlet(type:GRASS, hp:68, ap:55)
Please choose a monster from the above list.
50
Invalid input!
0:Pikachu(type:ELECTRIC, hp:35, ap:55)
1:Litten(type:FIRE, hp:45, ap:65)
2:Rowlet(type:GRASS, hp:68, ap:55)
Please choose a monster from the above list.
aaaa
Invalid input!
0:Pikachu(type:ELECTRIC, hp:35, ap:55)
1:Litten(type:FIRE, hp:45, ap:65)
2:Rowlet(type:GRASS, hp:68, ap:55)
Please choose a monster from the above list.
2
Ash chooses Rowlet(type:GRASS, hp:68, ap:55)
Kiawe chooses Alolan Marowak(type:FIRE, hp:60, ap:80)
Rowlet(type:GRASS, hp:68, ap:55) attacks Alolan Marowak(type:FIRE, hp:60,
ap:80)
Attacker monsters now: [Pikachu(type:ELECTRIC, hp:35, ap:55),
Litten(type:FIRE, hp:45, ap:65), Rowlet(type:GRASS, hp:58, ap:55)]
Defender monsters now: [Turtonator(type:FIRE, hp:60, ap:78), Alolan
Marowak(type:FIRE, hp:60, ap:80)]
Kiawe attacks Ash
Kiawe chooses Alolan Marowak(type:FIRE, hp:60, ap:80)
Choosing a DEFENDING monster
0:Pikachu(type:ELECTRIC, hp:35, ap:55)
1:Litten(type:FIRE, hp:45, ap:65)
2:Rowlet(type:GRASS, hp:58, ap:55)
Please choose a monster from the above list.
0
Ash chooses Pikachu(type:ELECTRIC, hp:35, ap:55)
Alolan Marowak(type:FIRE, hp:60, ap:80) attacks Pikachu(type:ELECTRIC,
hp:35, ap:55)
Attacker monsters now: [Turtonator(type:FIRE, hp:60, ap:78), Alolan
Marowak(type:FIRE, hp:60, ap:80)]
Defender monsters now: [Pikachu(type:ELECTRIC, hp:0, ap:55),
Litten(type:FIRE, hp:45, ap:65), Rowlet(type:GRASS, hp:58, ap:55)]
Ash attacks Kiawe
Choosing an ATTACKING monster
0:Litten(type:FIRE, hp:45, ap:65)
1:Rowlet(type:GRASS, hp:58, ap:55)
Please choose a monster from the above list.
1
Ash chooses Rowlet(type:GRASS, hp:58, ap:55)
```

```
Kiawe chooses Alolan Marowak(type:FIRE, hp:60, ap:80)
Rowlet(type:GRASS, hp:58, ap:55) attacks Alolan Marowak(type:FIRE, hp:60,
ap:80)
Attacker monsters now: [Pikachu(type:ELECTRIC, hp:0, ap:55),
Litten(type:FIRE, hp:45, ap:65), Rowlet(type:GRASS, hp:58, ap:55)]
Defender monsters now: [Turtonator(type:FIRE, hp:60, ap:78), Alolan
Marowak(type:FIRE, hp:5, ap:80)]
Kiawe attacks Ash
Kiawe chooses Alolan Marowak(type:FIRE, hp:5, ap:80)
Choosing a DEFENDING monster
0:Litten(type:FIRE, hp:45, ap:65)
1:Rowlet(type:GRASS, hp:58, ap:55)
Please choose a monster from the above list.
1
Ash chooses Rowlet(type:GRASS, hp:58, ap:55)
Alolan Marowak(type:FIRE, hp:5, ap:80) attacks Rowlet(type:GRASS, hp:58,
ap:55)
Attacker monsters now: [Turtonator(type:FIRE, hp:60, ap:78), Alolan
Marowak(type:FIRE, hp:5, ap:80)]
Defender monsters now: [Pikachu(type:ELECTRIC, hp:0, ap:55),
Litten(type:FIRE, hp:45, ap:65), Rowlet(type:GRASS, hp:0, ap:55)]
Ash attacks Kiawe
Choosing an ATTACKING monster
0:Litten(type:FIRE, hp:45, ap:65)
Please choose a monster from the above list.
0
Ash chooses Litten(type:FIRE, hp:45, ap:65)
Kiawe chooses Turtonator(type:FIRE, hp:60, ap:78)
Litten(type:FIRE, hp:45, ap:65) attacks Turtonator(type:FIRE, hp:60, ap:78)
Attacker monsters now: [Pikachu(type:ELECTRIC, hp:0, ap:55),
Litten(type:FIRE, hp:35, ap:65), Rowlet(type:GRASS, hp:0, ap:55)]
Defender monsters now: [Turtonator(type:FIRE, hp:60, ap:78), Alolan
Marowak(type:FIRE, hp:5, ap:80)]
Kiawe attacks Ash
Kiawe chooses Alolan Marowak(type:FIRE, hp:5, ap:80)
Choosing a DEFENDING monster
0:Litten(type:FIRE, hp:35, ap:65)
Please choose a monster from the above list.
0
Ash chooses Litten(type:FIRE, hp:35, ap:65)
Alolan Marowak(type:FIRE, hp:5, ap:80) attacks Litten(type:FIRE, hp:35,
ap:65)
Attacker monsters now: [Turtonator(type:FIRE, hp:60, ap:78), Alolan
Marowak(type:FIRE, hp:0, ap:80)]
Defender monsters now: [Pikachu(type:ELECTRIC, hp:0, ap:55),
Litten(type:FIRE, hp:35, ap:65), Rowlet(type:GRASS, hp:0, ap:55)]
Ash attacks Kiawe
Choosing an ATTACKING monster
0:Litten(type:FIRE, hp:35, ap:65)
Please choose a monster from the above list.
0
Ash chooses Litten(type:FIRE, hp:35, ap:65)
Kiawe chooses Turtonator(type:FIRE, hp:60, ap:78)
Litten(type:FIRE, hp:35, ap:65) attacks Turtonator(type:FIRE, hp:60, ap:78)
Attacker monsters now: [Pikachu(type:ELECTRIC, hp:0, ap:55),
Litten(type:FIRE, hp:35, ap:65), Rowlet(type:GRASS, hp:0, ap:55)]
Defender monsters now: [Turtonator(type:FIRE, hp:0, ap:78), Alolan
Marowak(type:FIRE, hp:0, ap:80)]
Winner is: Ash
```