Introduction of Object Orientation Programming 2: Lab Sheet 5

This Lab Sheet contains material based on Lecture 5.

The deadline for Moodle submission of this lab exercise is 12:00 on Thursday 31 October 2024.

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 and unzip **LB05_WK6_24_starter.zip file**
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 **LB05_WK6_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)**

Submission material

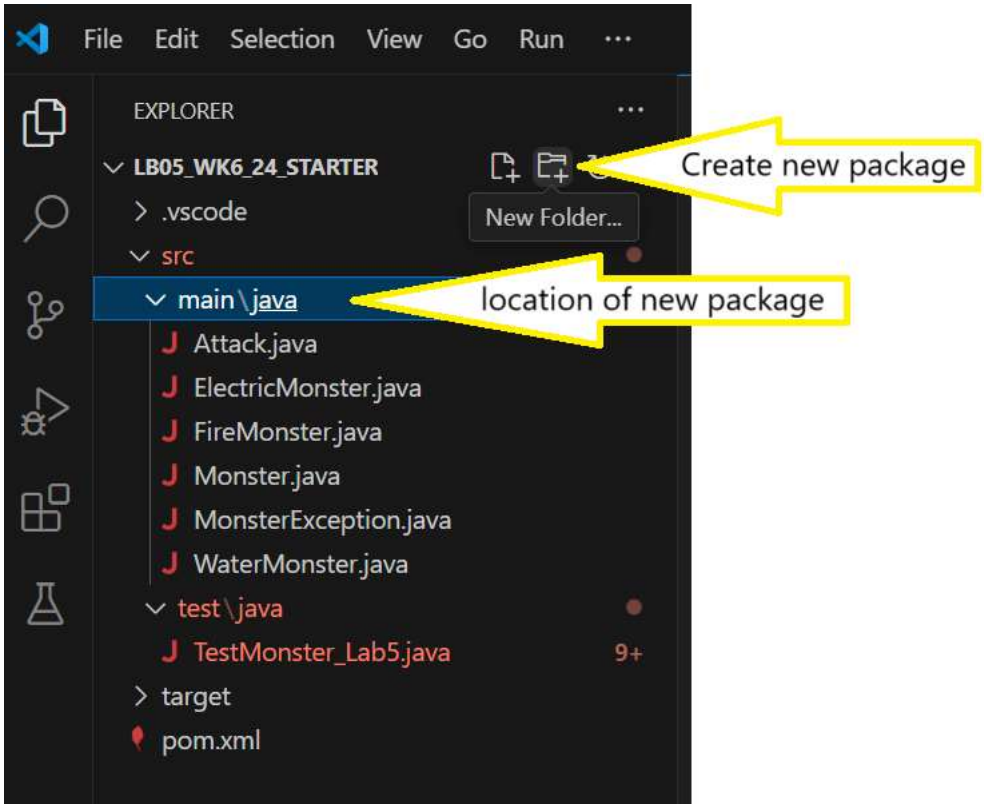
This exercise builds on the material that you submitted for LB04_WK5_24, so it might be worth referring back to your work on that lab before beginning this one.

In LB03_WK4_24, you developed a Monster class; in LB04_WK5_24, 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 collection of Monsters, and a **Trade** class, which represents a situation where two trainers are trading Monster to each other.

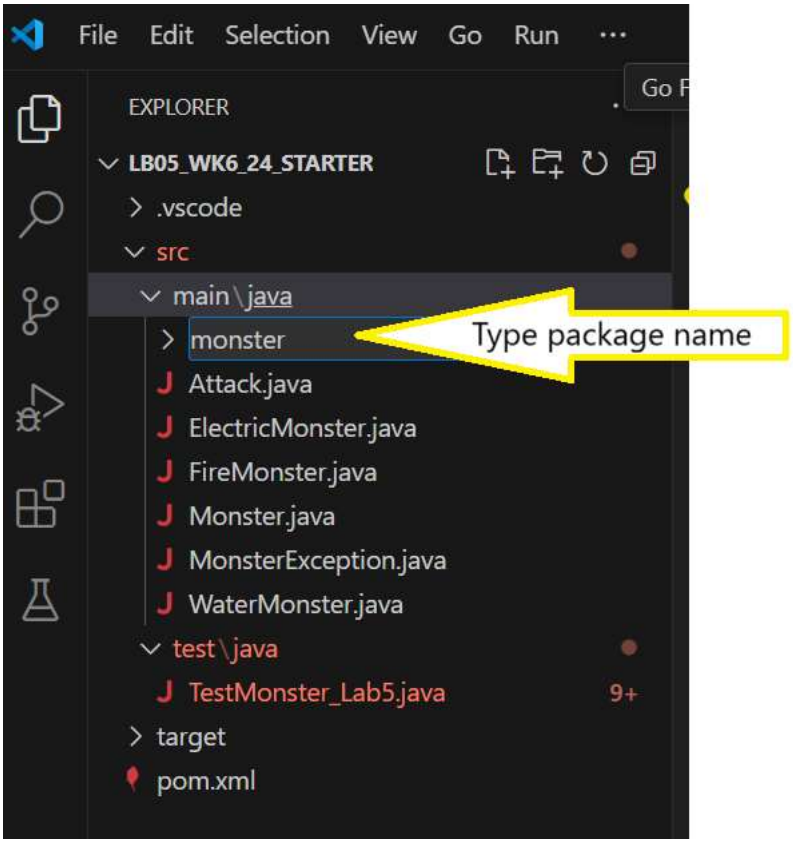
At the end of the lab sheet are some **optional** extensions to the basic functionality. These extensions are not required, and if you do attempt them, be sure that the classes you submit still support the basic functionality as described in this lab sheet.

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 Visual Studio, click on the location where you wish to put the new module **main\java** then click on the **New Folder** icon.



Then type the name of the package. You can then drag the classes into the new package.



Also, create a new package called **trainer** to contain the classes you will develop in this lab.

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

- **TestMonster_Lab5.java** in the **test\java** package
- All other provided classes are now in the **main\java\monster** package
- An empty **main\java\trainer** 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 **trainer** package called **Trainer**. A Trainer has two fields **name** and **monsters** where the latter is a collection 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)**
  - This should initialise any fields as necessary
- **public String getName()**
  - Returns the trainer's name
- **public boolean addMonster (Monster monster)**
  - Adds the given Monster to this trainer's set. Returns **true** if the monster was not in the set 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 set. Returns **true** if the monster was in the set before (i.e., if the monster was successfully removed), and **false** if the monster was not in the set before the method was called.
- **public boolean hasMonster (Monster monster)**
  - Checks whether the given monster is in the trainer's collection and returns **true** if it is and **false** if it is not.
- **public Map<String, Integer> countMonstersByType()**
  - Returns the count of monsters in each category. See below for more about this method.
- **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, but it should include the values of all fields.

More on countMonstersByType

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 ➔ 1
- Water ➔ 2

The test cases provide an example of the expected output for this method.

Trade class

You should now create another class in the **trainer** package, called **Trade**. 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 Trade (Trainer trainer1, Monster monster1, Trainer trainer2, Monster monster2) throws MonsterException**
    - Creates a new **Trade** object indicating that **trainer1** and **trainer2** will trade, with **trainer1** transferring **monster1** to **trainer2**, and **trainer2** transferring **monster2** to **trainer1**.
        - This method should throw a **MonsterException** if **trainer1** does not own **monster1** or **trainer2** does not own **monster2** (hint: use **hasMonster()**)
- **public void doTrade()**: do the actual trade – i.e., transfer **monster1** to **trainer2** and **monster2** to **trainer1** (hint: use **addMonster()** and **removeMonster()**)

Unit tests

- This project includes a set of unit tests in the **test\java\TestMonster_Lab5.java** in the package
-

. Please see the LB04_WK5_24 lab sheet for details on how to run the tests.

Recall that **the test cases do not test the behaviour of the methods** – 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, 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, compress both packages into **LB5_WK6_24_starter\src.zip.** Then go to the IOOP2 moodle site. Click on **LB05_WK6_submission**. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code – **LB5_WK6_24_starter\src.zip**-- 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 **A1** to **H**.

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.

- At the moment, nothing in the specification prevents a Monster from being owned by two Trainers. See if you can add some code to Monster and/or Trainer to catch this situation and throw an Exception when it occurs.
- You could also do some additional checking in **Trade.doTrade()** to make sure that the trade has not become invalid after it was constructed, and only do the trade if both Trainers still have their respective Monsters.