IOOP2 2025 Lab Sheet D
Battle in the Library!

This Lab Sheet contains material based on Week 4 topics.

The deadline for Moodle submission of this lab exercise is 12:00 noon on Thursday 23 October.

Aims and objectives

- Using and extending abstract classes and methods
- Using enumerated types as a Clean Code feature
- Finding and fixing bugs in provided code

Starter code

Download the file **LabD.zip** from the Moodle page and unzip it somewhere in your own file space. Open the **LabD** folder in VSCode to access the starter code.

This week, the starter code already contains a near-complete solution to this week's tasks, but with several bugs. Your main task will be to identify and fix these bugs and to summarise the bug-fixing process. Please make careful note of all of the bugs you identify and fix, as your main submission will be a report about the bugs that you found and how you fixed them.

Tasks

The Library is no longer a quiet place of study: strange creatures have begun to wander its halls. Your hero must be ready to defend themselves by fighting the enemy creatures.

Task 0 (already done for you): Removing "magic strings", making Book abstract

Until now, we have represented the hero's skills with strings; this is potentially error prone, so for this lab we have changed the implementation to use the enumerated type: **Skill**. Note how the changes have been made across the **Hero**, **SkillBook**, and **GameHarness** classes.

We have also updated the **Book** class to make it abstract, and the **doRead()** method has also been made abstract.

You do not need to make any changes for this part. But do make sure that you understand the changes that have been made so that you are able to carry out the other tasks for this week.

Task 1: Character class and subclasses

To represent the characters in the library – heroes and enemies – and to allow them to battle each other, we have added an abstract **Character** class to the project. Every character has a name and a set of hit points, and the character is considered "alive" as long as their hit point value is greater than zero. A character with zero hit points is "dead". A character should never have hit points below zero. The **Character** class also provides an abstract **attack** method that should be used to decrease the hit points of the targeted character.

We have provided a **Goblin** class giving a simple concrete implementation of **Character**, showing how the **attack** method can be implemented. We have also refactored the **Hero** class so that it is also a subclass of **Character**.

There are several bugs in this implementation! You should test the code thoroughly and make note of all the bugs that you find and fix. Some of the bugs will show up as errors (squiggly red underlines) in VSCode, while others reflect coding errors where the code will compile and run, but will produce incorrect results according to the intended behaviour as specified above. Note that some of the bugs may not be fully apparent until you also find and fix the bugs in **GameHarness** as part of Task 2.

For each bug, take note of these facts which you will need to include in your report:

- The source file where the bug was found
- The details of the bug, that is, what the issue was and the effect on the code
- A description of the change(s) that you made to fix the bug.

Sample bug description

We will give you one of the bugs for free, to show you what is expected in the bug reporting.

As you will probably have noticed, there is a bug in the **Hero.java** constructor: it does not correctly call the super-class **Character** constructor, so the code shows an error in VSCode, and to fix it requires calling the correct parent constructor. Here is how to report the bug and the required fix.

Bug 1: error in **Hero** constructor

- Source file: Hero.java (the constructor)
- Issue: **Hero** constructor does not explicitly call the super-class constructor, and since there is no suitable constructor available, the code does not compile
- Fix: replace the first two lines of the **Hero** constructor with the following:
    - **super(name, 12);**

Bugs to find

In addition to the sample bug identified above, there are **six more** intentional bugs to identify and fix across **Character.java** and **Hero.java.** There are **no** (intentional) bugs in **Goblin.java**. You should record each bug and how you fixed it in a format like what is shown above.

Task 2: Battling in GameHarness

A new option has been added to **GameHarness** to allow the hero to battle an enemy. The hero should always attack first, and then the hero and enemy take turns attacking each other, using the **Character.attack()** method, until one is no longer alive. If the hero loses the battle, the game is over.

There are bugs in this implementation too! Again, test your code thoroughly and make note of the bugs that you find so that you can include the description in your submission.

There are **two** intentional bugs in **GameHarness.**

Task 3: Adding a new enemy type

Once you have completed the above debugging tasks, your final task is to define a new enemy type. You can use the provided **Goblin** as an inspiration, but your enemy must have a different name and different battle mechanics: for example, it might do an increased amount of damage on each successive turn, might do more damage as its own HP decreases, or could implement something else. Be creative!

To test your enemy type, you can update **GameHarness.java** to replace the use of the **Goblin** class with your new enemy class.

Testing your code

You can test your code using the **main** method of the provided **GameHarness.java** file. **This week, don't forget that there are bugs in GameHarness as well as in the other provided files.**

What to submit

Before the deadline, you should submit the source file containing your new enemy type from Task 3. **You do not need to submit any other source files.**

You should also submit a text document (MS Word or plain text) including a description of all the bugs that you found in the code and how you fixed them. You should be specific, for example referring to specific methods in the original source files.

For each bug, make sure to note the details as shown in the example:

- The source file where the bug was found
- The details of the bug -- that is, what the issue was and how it would affect the code
- A description of the change(s) that you made to fix the bug.

Marking scheme

This lab will be marked out of 6, as follows:

- 1 mark for submission of source file and bug document, whether correct or not
- 3 marks for bug fixes in **Character** and **Hero**
- 1 mark for bug fixes in **GameHarness**
- 1 mark for submission of new enemy type

Optional tasks

If you have completed the lab tasks, here are some additional things you might want to try. Note that we will continue to extend these classes significantly over the next several weeks, but if you want to try something new in the meantime, here are some ideas.

1. Come up with some appropriate behaviour if the current skill is HEALING – for example, this could increase the user's HP, or even the target's HP. You might also want to make sure that the HP can't go above a certain value.
2. Implement "critical hits", where a hit might have a small chance of doing extra damage (hint: you can use the **java.util.Random** class to help implement this).
3. Create a new enemy type with more complex attack mechanics, such as a stun mechanism that skips the next turn or a burning/poison mechanism that does damage over time.