Introduction to Object-Oriented Programming Lab Sheet 7+8

This Lab Sheet contains material based on the Week 7 and 8 topics. You will have two weeks to work on this lab and should submit a solution to all tasks at 16:30 on Thursday 16 November 2022.

Your mark for Lab 7 will be based on your solutions to the tasks in the Lab 7 section; your mark for Lab 8 will be based on your solutions to the tasks in the Lab 8 section. You are free to work on the tasks in any order, although you might want to wait until the Threads content has been covered in the lecture to work on the Lab 8 section.

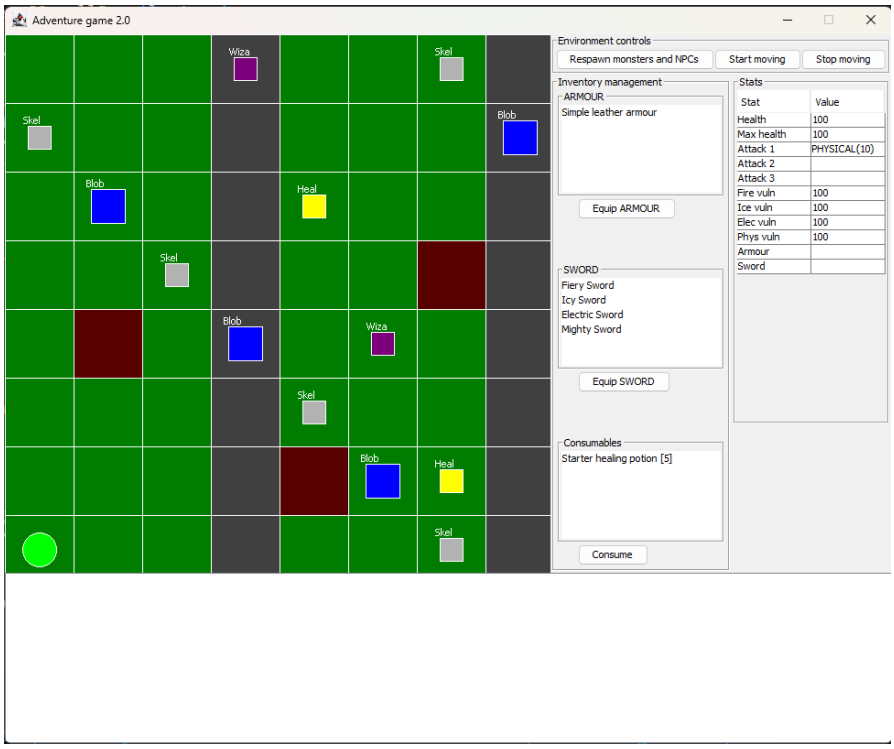
Aims and objectives

- Implementing callback methods for Swing GUI widgets (Lab 7)
- Writing multi-threaded programs (Lab 8)

Running the code

This week's starter code consists of a full solution to Lab 6, with very slight modifications, as well as a new subpackage **openworld.gui** containing code for a Swing version of the adventurer battling game. You should use this week's code as a starting point for the current lab, rather than your own Lab 6 solution, as the GUI code makes assumptions about various aspects of the game code.

When you run the **main** method in the **openworld.gui.GameWindow** class, a window that resembles the following will appear:



The grid on the left represents the game world and is created and managed by the **openworld.gui.GameWorld** class. The terrain is represented by green squares (grass), dark grey squares (mountains), and dark red squares (volcanoes). The other squares represent monsters and NPCs: the large blue squares are blobs, the smaller light grey squares are skeletons, the small yellow squares are healers, and the small purple squares are wizards. The light green circle in the bottom left represents the adventurer.

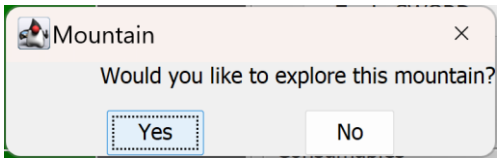
All of the things that appear on the screen are represented by objects of the **openworld.gui.Sprite** class and its subclass **openworld.gui.AdventurerSprite**. If you are curious, you can look at the overridden **GameWorld.paintComponent** method to see how the on-screen objects are drawn.

Note: sprites monsters and NPCs stay in the game world even if they are not conscious. This is not a problem that you need to fix necessarily, but addressing this issue could be your choice for the open-ended task in Task 8.4.

The buttons on the right represent game controls and are created and managed by the **openworld.gui.ControlPanel** class, with specific components in the **openworld.gui.InventoryPanel**, **openworld.gui.EnvironmentPanel**, and **openworld.gui.AdventurerPanel** classes. The code includes a simple inventory system; details of how that system works are provided in the Lab 7 specification.

When you press the arrow keys, the adventurer moves around the map. Each time the adventurer moves, the **takeTurn()** method of the other monsters and NPCs is also called, so you should also see them move around and attack each other. The window at the bottom shows the log of game actions (which we have until now been printing to System.out).

If the adventurer moves into a mountain location, a dialog box will pop up like this:



The intention is that you would choose whether to explore the Mountain (using the **Mountain.explore()** method) or not. This behaviour is not fully implemented in the starter code, but you will complete it as part of Lab 7.

In the starter code, none of the buttons are operational either. Your other task in Lab 7 will be to make the buttons operational (except for start and stop buttons, which are addressed in Lab 8). The required behaviour of each button is listed in the Lab 7 section, as well as the marks allocated to each button.

In Lab 8, you will make the Monsters move around the screen in response to the “Start Monsters” and “Stop Monsters” button and will also fine-tune other aspects of the gameplay. Details of the required behaviour are listed in the Lab 8 section.

A video will be posted on Moodle showing the expected system behaviour after all parts of both labs have been implemented.

Lab 7

Your task in this lab is to write code to enable the buttons in the control panel to control the game behaviour. Note that the "Start moving" and "Stop moving" buttons are for Lab 8.

Task 7.1: Respawning Monsters and NPCs (1 star)

The **openworld.gui.GameWorld** class provides a method **respawnWorld()** which is called as part of the constructor to populate the game world. You must add code to the **openworld.gui.EnvironmentPanel** class so that this method is also called when the "Respawn all" button is clicked.

Hint: this should be a very simple one or two lines of code once you figure out the correct place to put them. If you want to be sure that your code is working correctly, you should move the adventurer around a few steps to make the monsters and NPCs move around and battle each other; the respawn button should restore the screen to look like it did at the start of the game.

Task 7.2: Exploring Mountains (1 star)

The code to open the "Mountain" dialog box can be found in the **openworld.gui.MoveAdapter.moveAdventurer** method. However, at the moment, the code does not do anything with the result of the dialog box. You need to look at the **openworld.gui.MountainDialog** code to work out how to tell whether the user clicked "yes" or "no", and then write additional code in **moveAdventurer** to implement the exploration if they did click "yes". depending on the type of the entity:

Hints:

- You should call the **Mountain.explore()** method with an appropriate parameter.

Task 7.3: Inventory management (2 stars)

The game includes an **inventory**, which represents items that the adventurer can carry with them and optionally use to help them while exploring the world and battling. All inventory-related classes are in the **openworld.item** package, except for the inventory itself which is in the **openworld.adventurer.Inventory** class.

There are two main types of things that the adventurer can carry in their inventory, **Consumable** items and **Equippable** items.

A **Consumable** item is an item that the adventurer can use to affect their stats – the only type of consumable item included in the starter code is a healing potion. A healing potion can be used a defined number of times – the default healing potion given to the adventurer can be used 5 times. Each time the potion is used, it heals the adventurer, up to their maximum health.

An **Equippable** item is an item that the adventurer can choose to equip in battle – the two main types of equippable items in the starter code are swords and armour. When one of these is equipped, it affects stats such as the damage that the hero can do or their vulnerability to different damage types.

The three lists under "Inventory management" in the main window represent the items in the adventurer's inventory of the three types. Your task is to make the three associated buttons work.

All of the relevant GUI code is in the **openworld.gui.InventoryPanel** class, while the methods that you will need to call are in the **openworld.item.Item** and **openworld.adventurer.Inventory** classes.

Consumable items (1 star)

When the "Consume" button is clicked, the program should check if something is selected in the "Consumables" list. If there is a selection, the adventurer should consume the item using the **Inventory.useItem()** method; if there is no selection, your code should do nothing. You should see changes to the adventurer's stats based on the properties of the item, and the use count for the item will also go down in the list.

Equippable items (1 star)

When either of the "Equip" buttons is clicked (sword or armour), your code should check if something is selected in the corresponding list. If there is a selection, you should equip the item using the **Inventory.equipItem()** method; if there is no selection, your code should do nothing. You should see changes to the adventurer's stats based on the properties of the item.

Hints:

- The live-coding example for "Delete Employee" should provide a very useful example of the code that you should write to implement the above tasks
- **Do not remove the calls to updateLists() or to ControlPanel.getAdventurerPanel().update()** – these calls are very important to keep the displayed information in sync with the inventory

Lab 8

Your main task in this lab is to write code to make the monsters and NPCs move around the map in response to a click on the "Start Monsters" button, and to stop when the "Stop Monsters" button is clicked; you will also implement some other gameplay enhancements.

Note that the instructions for this lab are deliberately less detailed than in previous weeks. The live-coding example from the Week 8 lectures should provide all of the tools that you need to get this working. In our sample solution, the complete implementation of all three tasks requires less than 30 lines of code.

Task 8.1: Starting the Monsters and NPCs (1 star)

The first task is to make all of the Monsters and NPCs move around when the "Start Moving" button is clicked. Concretely, you should ensure that the **openworld.World.nonPlayerCharactersMove()** and **openworld.World.monsterMove()** methods are called once per second (i.e., once every 1000 milliseconds), repeatedly.

Hints:

- In the starter code, the above two methods are called in **MovementAdapter.keyPressed()** – you should remove those method calls.
- Note that you will need to be able to stop the Monsters and NPCs as part of Task 8.2 (below) – this should also inform your design of the Monster movement code.
- Make sure that your code also works (i.e., doesn't crash or make the movement fast or strange) if "Start Moving" is clicked repeatedly

Task 8.2: Stopping the Monsters (1 star)

Now that the Monsters and NPCs are able to move, the next step is to make them stop moving when the "Stop Moving" button is clicked.

Hints:

- You will likely need to add some more code to **openworld.gui.GameWorld** to be able to stop the characters from moving on demand.
- Take care that clicking the button several times in a row does not result in a crash or any other unintended behaviour.

Task 8.3: Fixing the adventurer movement (1 star)

You may have noticed that it is possible to move the adventurer off the map (e.g., by pressing left or down at the start of the game). The problem here is in the **openworld.gui.MoveAdapter.moveAdventurer** method. You need to find the bug and fix it so that the adventurer cannot move off the screen.

Hint:

- There is code in the **openworld.Coordinates** class that can help with solving this problem.

Task 8.4: Gameplay enhancements (1 star)

Once you have the core threading and GUI tasks working, you should add an additional feature to the game for the final mark. If you added a feature in your Lab 6, you could port that to the GUI version; you could deal with fainted NPCs and monsters (either by removing them from the game, or by making their sprites look different, or even by creating a method for them to come back to life); you could also add other NPC, Monster, Terrain, or Item types, add sound effects or fancier visual effects, or anything else. You can check with the lecturers and tutors about your idea.

You must include a text file with your submission indicating your enhancement or you will not receive marks for this task.

What to submit

You should submit code for both tasks on Thursday 16 November through Moodle – there is no submission on Thursday 9 November.

You should submit a zip file consisting of all of your .java files in all packages, along with a text document indicating the modification that you made as part of Task 8.4.

You will receive two marks through Moodle: the mark for Lab 7 will be based on Tasks 7.1—7.3, while the mark for Lab 8 will be based on Tasks 8.1—8.4. The code will not be auto-marked; the tutors will test your code by running your game and clicking on the various buttons to check the behaviour.