# CSE2IOO Semester 1, 2025 Assignment

**Assessment: This assignment is worth 30 % of the final mark for this subject.**

**Due Date: Thursday May 22nd, 2025, at 11.59 pm**

Delays caused by computer downtime cannot be accepted as a valid reason for a late submission without penalty. Students must plan their work to allow for both scheduled and unscheduled downtime. Penalties are applied to late assignments, if you encounter difficulties that lead to late submission or no submission, you should apply for special consideration. Please check SLG for further information on special considerations.

**Individual Assignment:** This is an individual assignment, and the solutions that you submit for all of the tasks must be your own.

**Copying, Plagiarism:** Copying solutions from other students, friends, AI assistants/ChatGPT (Any use of Gen AI must be done responsibly. You must not ask AI to write your assignment and your code for you. You must not submit code that you do not understand), or strangers on the internet is plagiarism and treated as academic misconduct by La Trobe University. The Department of Computer Science and Computer Engineering treats academic misconduct seriously. When it is detected, penalties are strictly imposed and can be as serious as expulsion from the university**.** Plagiarism also includes copying solutions and making superficial changes to the code (such as renaming variables or editing comments). Students who are found to be in breach of this rule may be engaged to provide an oral validation of their understanding of their submitted work (e.g. coding).

**Objectives:** The general aims of this assignment are:

- To analyse a problem in an object-oriented manner, and then design and implement an object-oriented solution that conforms to given specifications.
- To practise using inheritance and polymorphism in Java.
- To practise file input and output in Java.
- To make implementations more robust through mechanisms such as exception handling.

**Submission Details and Marking Scheme:** Please submit only the Java files that you create for the tasks (i.e. please don't submit the class files). Please compress the files into a zip file and submit it through the submission portal on LMS.

If you have not been able to complete a program that compiles and executes containing all functionality, then you should submit a program that compiles and executes with as much functionality as you have completed. You may comment out code that does not compile. However, the commented-out code will not be marked.

**Deployment Platform:** While you are free to develop the code for this assignment on any operating system, your solution must run without any dependencies. We should be able to compile your classes with the simple command **javac *.java**, and execute your programs with a simple command, e.g. **java SomeTester**

## Problem Description

**Monster Battle Quest -** You are tasked with building a turn-based monster battling game. Players can recruit monsters from different species, train them, and engage in battles against other players. Each monster has a type (Fire, Water, Grass), and type advantages influence the damage in battle.

## Task 1: Class Design and Core Implementation

Implement these classes and relationships:

### 1. Monster Class (Abstract Class) – Monster.java

This is the base class for all types of monsters (Fire, Water, Grass). You cannot create a generic Monster directly, but it defines the common features all monsters share.

**Attributes:**

All attributes of this class should be with protected access modifier. This means they are accessible within the class itself and to any subclasses (like FireMonster, WaterMonster, etc.), but not accessible from outside the class hierarchy.

- **protected String name:** The custom nickname given to the monster (e.g., "Blaze").
- **protected MonsterSpecies species:** Reference to the species this monster belongs to (e.g., FireDragon).
- **protected int level:** The monster's level (starts at 1).
- **protected int hp:** The monster's current health points (decreases in battle).
- **protected int attackPower:** How much damage the monster deals when attacking.

**Key Methods:**

- **public abstract void attack(Monster target):** Must be implemented in each subclass. Describes how the monster attacks another.
- **public void takeDamage(int amount):** Reduces the monster's hp by a given amount.
- **public boolean isFainted():** Returns true if the monster's HP is zero or less (fainted).
- **public String toString():** Returns a readable string with monster details (e.g., "Blaze (FireDragon, Lv.1, HP: 90)").

In addition to above methods, you should write appropriate constructor and getter methods to provide controlled access to the attributes where needed.

## 2. FireMonster, WaterMonster, GrassMonster Classes (subclasses of Monster) – FireMonster.java, WaterMonster.java, GrassMonster.java

These are subclasses of Monster. Each subclass has a constructor which calls the super class's constructor.

The **attack(Monster target)** method is defined in the abstract class Monster. Because it's marked abstract, each subclass (FireMonster, WaterMonster, GrassMonster) **must override** this method and define how the monster attacks another monster. These classes are where the actual battle logic is applied.

In the attack(Monster target) method of each subclass:

- Start with the monster's own attackPower
- Check the type of the target's species
- Apply type advantage rule:
    - (Fire > Grass) Fire monsters deal +10 bonus damage against Grass monsters.
    - (Water > Fire) Water monsters deal +10 bonus damage against Fire monsters.
    - (Grass > Water) Grass monsters deal +10 bonus damage against Water monsters.
- Call target.takeDamage(damage) to reduce the target's HP.

## 3. MonsterSpecies Class – MonsterSpecies.java

This class defines a template or blueprint for monsters of a particular kind (like Pokémon species).

All attributes of this class should be private. These are only accessible within this class, promoting encapsulation.

**Attributes:**

- **private String code:** Unique code for the species (e.g., "FD001").
- **private String name:** Species name (e.g., "FireDragon").
- **private String type:** Fire, Water, or Grass.
- **private int baseHP:** Starting health points for this species.
- **private int baseAttack:** Starting attack value.

You should write appropriate constructor, toString() and any getter methods as needed.

### 4. Player Class – Player.java

This class represents a human player in the game. Each player has a team of up to 5 monsters.

**Attributes:**

- **private String playerId:** Unique identifier (e.g., "P100").
- **private String name:** The player's display name.
- **private Monster[] team:** Array holding up to 5 monsters.
- **private int monsterCount**: Tracks how many monsters are in the team.

**Key Methods:**

- **public void addMonster(Monster monster):** Adds a monster to the team. (checks for full or duplicates).
- **public boolean hasAliveMonster():** Returns true if at least one monster hasn't fainted.
- **public Monster getNextAliveMonster():** Gets the next available monster for battle.
- **public String toString():** Displays all monster details for the player.

You should write appropriate constructor, and any getter methods as needed.

### 5. MonsterBattleSystem Class – MonsterBattleSystem.java

This class is the **main engine** of the game. It manages:

- All species that have been registered
- All players and their monster teams
- Displaying data
- Handling battles

**Attributes:**

- **private MonsterSpecies[] speciesList:** Stores all defined monster species (max 50).
- **private Player[] players:** Stores all registered players (max 100).
- **private int speciesCount, playerCount:** Track how many species and players exist.

**Key Methods:**

- **public void addSpecies(...):** Adds a new species (checks for duplicate codes).
- **public void addPlayer(...):** Registers a new player (checks for duplicate IDs).
- **public void assignMonster(...):** Assigns a monster of a chosen species to a player.

- **public String battle(String player1ID, String player2ID):** Simulates a turn-based battle between two players. Each round involves the first available (not fainted) monster from each side attacking in turn. The battle continues until all monsters from one side faint.

You should write appropriate constructor, toString() and any getter methods as needed. You may also need to add helper methods for this class.

**Battle Rules:**

- **Turn-based rounds** — One monster from each player fights at a time.
- **Each round:**
    - Player 1's monster attacks first
    - Then, if still alive, Player 2's monster attacks
- **A monster faints** if its HP drops to 0 or below.
- Once a monster faints, the next alive monster in the team takes its place.
- The battle continues until all monsters on one side are fainted.
- The player with remaining monster's wins.

**No Interactive Inputs for Task 1**

The **MonsterBattleSystem** class must be implemented in such a way that it can be tested by the program given in the Appendix A without any changes. Thus, it should not be interactive just yet. That is, it should not take any input by the user via the keyboard.

## Task 2 – Testing the MonsterBattleSystem class

Test your **MonsterBattleSystem** class with the test program provided in the appendix.

Your classes should be implemented in such a way that the provided test program can be run without any changes.

## Appendix – A Test Program – MonsterBattleTester.java

As stated earlier, we should be able to run the test program below with your classes, without having to any changes to it. You can also download this .java file from assessment section on LMS.

```java
public class MonsterBattleTester {
    public static void main(String[] args) throws Exception {

        // Create a instance of the battle system
        MonsterBattleSystem system = new MonsterBattleSystem();

        // Add species to the system
        system.addSpecies("FD001", "FireDragon", "Fire", 100, 50);
        system.addSpecies("WD002", "AquaSerpent", "Water", 110, 45);
        system.addSpecies("GD003", "LeafBeast", "Grass", 90, 55);
        // Uncomment the line below to test duplicate species entry
        // Duplicate species should not be added
        //system.addSpecies("FD001", "FireDragon", "Fire", 100, 50);

        // Add players to the system
        system.addPlayer("P100", "John");
        system.addPlayer("P200", "Alice");
        system.addPlayer("P300", "Bob");
        // Uncomment the line below to test duplicate player entry
        // Duplicate players should not be added
        //system.addPlayer("P100", "John");

        // Assign monsters to players
        system.assignMonster("P100", "Blaze", "FD001");
        system.assignMonster("P200", "Sprout", "GD003");
        system.assignMonster("P300", "Aqua", "WD002");

        // A battle between two players and print the result
        System.out.println(system.battle("P100", "P200")); // Blaze vs. Sprout
        // Uncomment below to test a different battle scenario
        // System.out.println(system.battle("P100", "P300")); // Blaze vs. Aqua

        // Print the current state of the system (all players and their monsters)
        System.out.println(system);

    }
}
```

### Sample run – MonsterBattleTester [Blaze vs. Sprout Battle]

```
=== Battle Pre-checks ===

John's Monster [Blaze] HP: 100 Fainted?: false

Alice's Monster [Sprout] HP: 90 Fainted?: false

=== BATTLE START ===

Round 1:

Blaze (Fire) attacks Sprout (Grass) for 50 damage. Sprout now has 30 HP.

Sprout (Grass) attacks Blaze (Fire) for 55 damage. Blaze now has 45 HP.
```

```
Round 2:

Blaze (Fire) attacks Sprout (Grass) for 50 damage. Sprout now has 0 HP.

Sprout has fainted!

Winner: John (P100)

===== BATTLE SUMMARY =====

=== MONSTER SPECIES ===

FD001: FireDragon [Fire] (HP: 100, ATK: 50)

WD002: AquaSerpent [Water] (HP: 110, ATK: 45)

GD003: LeafBeast [Grass] (HP: 90, ATK: 55)

=== PLAYERS ===

P100 - John

  > Blaze (FireDragon, Lv.1, HP: 45)

P200 - Alice

  > Sprout (LeafBeast, Lv.1, HP: 0)

P300 - Bob

  > Aqua (AquaSerpent, Lv.1, HP: 110)
```

## Task 3: Reading from and writing to files

In Task 1, you have implemented the classes to represent monsters, species, and players. You have also implemented a MonsterBattleSystem class which allows us to add monsters, species, players, allocation of monsters to players and player battles.

Building on the work that you have done for Task 1, in this Task 3, you are required to do the tasks described below.

Essentially, you will expand on the functionality of the classes that you have developed in Task 1, and

Besides the information given in the task below, please refer to Task 1 of the Assignment for other information you need.

Implement two methods for the **MonsterBattleSystem** class to save data and to load data.

- **public void saveData() / loadData():** Saves or loads the current game state

**public void saveData() throws Exception** will save the data to three text files (check LMS for these text files):

**MBQ-Species.txt**
This file saves the measurement monster species. A sample is shown below:

```
FD001; FireDragon; Fire; 100; 50

WD002; AquaSerpent; Water; 110; 45

GD003; LeafBeast; Grass; 90; 55
```

Each species is on a separate line. Each line contains SpeciesCode, SpeciesName, Type, BaseHP, and  BaseAttack separated by a semi colon.

**MBQ-Players.txt**
This file saves the player information. A sample is shown below:

```
P100; John

P200; Alice

P300; Liam
```

Each player is on a separate line. Each line contains PlayerID and PlayerName, separated by a semicolon.

**MBQ-Monsters.txt**
This file saves a monster that belongs to a player. A sample is shown below:

```
P100; Blaze; FD001; 1; 100; 50

P200; Sprout; GD003; 1; 90; 55

P300; Frost; IC005; 1; 105; 48
```

Each monster is on a separate line. Each line
contains PlayerID, MonsterNickname, SpeciesCode, Level, HP, and AttackPower, separated by semicolons.

The second method, with the header  **public void loadData() throws Exception** will read the data from the above three text files and save them in a **MonsterBattleSystem** instance by calling methods to add species, add player, add monster etc.

You may need to enhance other classes to support the implementation of the saveData and loadData methods. Also, you should write a small tester to test your methods. The implementation must be such that the sample test program can be run without any change. The sample test program in Appendix A is more or less the bare minimum. You should add more test

## Task 4: Menu interface – MonsterBattleSystemMenu.java

The overall aim of the assignment is to develop a prototype for a **MonsterBattleSystem.**

Write the menu program, called **MonsterBattleSystemMenu.java**, that provides the following interface

```
======================
Monster Battle Quest
======================
1. Add Monster Species
2. Register Player
3. Recruit Monster to Player
4. Display Player Info
5. Display All Data
6. Save Data
7. Load Data
8. Battle Between Two Players
X. Exit
Please enter an option (1-8 or X):
```

The menu program must use the **MonsterBattleSystem** class developed for Task 1.
For each option, the program should ask for the required inputs, each with a separate prompt. A sample run will be provided in a separate file.

The menu program for this task is only required to satisfy the data correctness conditions. It is not required to be robust, i.e. it can crash when certain exceptions occur.

## Task 5- Making the Menu Program Robust – RobustMonsterBattleSystemMenu.java

Implement a class called **RobustMonsterBattleSystemMenu.java** which extends the previous menu and includes enough exception handling features to make it robust. It is required to be robust only in the following sense: when an exception occurs in the execution of an option, the program will display some information about the exception on the screen and return to the main menu.

## Task 6 – Design Your Own Monster – YourOwnMonsterName.java

In addition to implementing the standard monsters (Fire, Water, Grass), you are required to design and implement one original monster species of your own creation. You are required to create a custom monster species from your imagination. This monster must have:

- Create a new subclass of Monster. A unique elemental type (not Fire, Water, or Grass)
- Give it a unique type, baseHP, and attackPower values.
- Override the attack() method with a new mechanic. A distinct custom attack mechanic implemented through attack() method.
- You may also override toString() to personalise output.

## Electronic Submission of the Source Code

- Submit all the Java files that you have developed for the tasks above.
- The code has to run without any external libraries or dependencies requirement.
- You must compress your Java files into a zip file and only submit that zip file, to the submission portal on LMS, the link is underneath where you found the assignment.
- Once deadline is reached, the copy of your submission will be considered the final submission.

## Marking Scheme Overview

### Implementation (Execution of Code) — 90%

- Do all parts of the program execute correctly?
- Note: Your program must compile and run in order to receive marks for implementation. Comment any code that may be partially correct but doesn't compile.
- During the execution test, you may be asked to explain any part of your code. Marks will be reduced if you are unable to explain how your methods or logic work.

### Program Design and Coding Style — 10%

- Does the program conform to the given specifications?
- Is the problem solved in a well-structured and modular way?
- Are good programming practices followed (e.g., proper use of methods, encapsulation, etc.)?
- Is the code well-formatted with consistent indentation?
- Are identifier names meaningful?
- Are comments used appropriately to enhance understanding?

## Return of Assignments

Assignments will be marked **during your allocated lab sessions in Week 12**. It is **mandatory** to attend your face-to-face lab in Week 12. If you are unable to attend your scheduled lab, please email me – Rudri at r.kalaria@latrobe.edu.au to make alternate arrangements.

**Good luck on your quest and remember: debugging is part of the journey! May your monsters be mighty!**