Object-Oriented Programming
Group Assignment #2
The Middle-Earth Madness

CMP_SC/INFO_TC 3330

Spring 2025

Overview

In this assignment, you will design and implement an object-oriented Java application that manages various characters from Middle-earth. The system must support Create, Read, Update, and Delete (CRUD) operations using simple arrays. The array storing the characters should automatically double in size when it reaches its limit. You will implement concepts such as abstract classes, inheritance, polymorphism, and the **Singleton Design Pattern**. **You must not use Java's Collections Framework (e.g., `ArrayList`, `Set`, `Map`).**

Learning Objectives

- Apply object-oriented principles: abstraction, inheritance, and polymorphism.
- Implement CRUD operations using simple arrays.
- Handle dynamic resizing of arrays manually.
- Design reusable, extendable, and modular code.
- Understand and apply the Singleton Design Pattern.

Background Story

The great realms of Middle-earth—Rohan, Gondor, Mordor, and beyond—are preparing for the coming battles. You have been summoned by the White Council to build a Character Management System that keeps track of heroes, villains, and mythical beings. The system should allow new characters to be added and existing ones to be updated, viewed, or removed as the tides of war shift. However, as we all know, there could be no peace in Middle-earth. Characters will battle and attack each other for the One ring crafted by Sauron at Mount Doom.

The system must recognize different types of characters such as **Elves**, **Dwarves**, **Humans**, **Orcs**, and **Wizards**, each with distinct traits and behaviors.

Requirements

1. Class Design

- **Abstract Class:** `MiddleEarthCharacter`
  - **Fields:** `name (String)`, `health (double)`, `power (double)`
  - **Constructor:** Initializes `name`, `health`, and `power`.
  - **Abstract Methods:**
    - `boolean attack(MiddleEarthCharacter target)` – Return `true` if attack is successful and reduced target's health. Return `false`, attack was ineffective, or attack was against their own kin. Characters that are the same kin cannot do any damage to each other.
    - `String getRace()` – Returns the race of the character.
  - **Concrete Method:** `void displayInfo()` – Prints character details.
- **Subclasses:**
  Must extend `MiddleEarthCharacter` and implement `attack(MiddleEarthCharacter)` and `getRace()`:
  - **Elf**:
    - 1.5 × damage against *Orc*.
    - Ineffective (0 damage) against *Dwarf* and *Elf*.
    - Normal damage against *Human* and *Wizard*.
  - **Dwarf**:
    - 1.5 × damage against *Elf*.
    - Ineffective (0 damage) against *Wizard* and *Dwarf*.
    - Normal damage against *Human* and *Orc*.
  - **Human**:
    - 1.5 × damage against *Wizard*.
    - Ineffective (0 damage) against *Orc* and *Human*.
    - Normal damage against *Elf* and *Dwarf*.
  - **Orc**:
    - 1.5 × damage against *Human*.
    - Ineffective (0 damage) against *Elf* and *Orc*.
    - Normal damage against *Dwarf* and *Wizard*.
  - **Wizard**:
    - 1.5 × damage against *Dwarf*.
    - Ineffective (0 damage) against *Human* and *Wizard*.
    - Normal damage against *Elf* and *Orc*.

  Each subclass should override `attack(MiddleEarthCharacter)` with a unique attack rule.

2. Singleton Pattern Requirement

- **Class:** `MiddleEarthCouncil` (Singleton)
  - **Methods:**
    - `static MiddleEarthCouncil getInstance()` – Returns the single instance.
    - `CharacterManager getCharacterManager()` – Provides access to `CharacterManager`.

3. Character Management System (CMS)

- **Class:** `CharacterManager`
  - **Fields:**
    - `MiddleEarthCharacter[] characters` – Stores `MiddleEarthCharacter` objects.
    - `int size` – Tracks stored characters.
  - **Methods:**
    - `boolean addCharacter(MiddleEarthCharacter c)` – Adds character; doubles array size if full.
    - `MiddleEarthCharacter getCharacter(String name)` – Retrieves character by name.
    - `boolean updateCharacter (MiddleEarthCharacter character, String name, int health, int power)` – Updates character. Returns `true` if there is a change, returns `false` if there is no change/update, or the character does not exist.
    - `boolean deleteCharacter(MiddleEarthCharacter character)` – Deletes character, shifts elements.
    - `void displayAllCharacters()` – Displays all characters. Consider using `displayInfo()`.

4. Main Class:

`MiddleEarthApp`

- Provide a menu-driven interface with options to:
  1. Add a new character.
  2. View all characters.
  3. Update a character.
  4. Delete a character.
  5. Execute all characters' attack actions.
  6. Exit.

- Ideally, you should create a separate class for the "Menu". That is considered to be good practice. It satisfies the 'S' from the SOLID principles, and a menu is very close to a design pattern called "Mediator". In other words, this is real OOP. However, I will keep this note here as a social experiment and see who really pays attention to what I write here and what I say in class. You will not lose or get any extra points if you create a Menu class or not. At least for now ;)

5. Dynamic Array Resizing

Implement resizing in `addCharacter(MiddleEarthCharacter)` to double capacity when full.

6. Additional Requirements

- All methods must include Javadoc comments explaining their functionality, and yes, getter and setter methods, too.
- Ensure polymorphic behavior when invoking `attack(MiddleEarthCharacter)`.
- Provide a `main` method to:
  - Demonstrate all CRUD operations.
  - Simulate a battle sequence illustrating all attack rules.

7. Evaluation Criteria

- Correct implementation of OOP principles.
- Functional Singleton design pattern.
- Proper array resizing logic.
- Comprehensive use of polymorphism and inheritance.
- Clear and logical attack outcomes based on effectiveness rules.

8. Hints

- Plan how characters will be removed from the array (shift elements or manage a dynamic index).
- Carefully manage `null` references after deletion.
- Test each scenario where an attack could be ineffective, normal, or more effective.

Important Notes

- Follow Java naming conventions, or you will lose points.
- Use packages or you will lose points.
- Add Javadoc to your code, or you will lose points.
- Export your project properly, or you will lose points.
- Don't want to 1 commit project, or commit messages like "*Adding Java code*" or "*Update code*", otherwise you will lose points. Commits must be small and meaningful with a commit message that is relevant to the code you pushed. Only I am allowed to do the above, because I am the professor of this class, and I can do whatever I want. This is my class :D
- Write your code considering edge cases. Make sure you have error controls.
- Don't ask how many points will be deducted for the notes above. There is no negotiation here. These are good practices that you must adopt and follow to have a successful career. You can try to violate one of the good practices above and see what happens :) (not recommended).
- Everyone in the group must contribute to the project. Use Git efficiently and communicate!
- If there is a group drama, you have to wait until the next group assignment to split from your group or work alone. See syllabus for details.
- **Due date:** 3/5/2025, 11:59 PM.
- **Submission:** You must submit your GitHub repository and your exported project through Canvas. Submit your GitHub link repository in a file along with your project file submission.

Evaluation Criteria

Component	**Points**
Abstract Class & Inheritance	15
Polymorphism Usage	15
Singleton Pattern Usage	10
CRUD Operations	10
Logic Operations (other methods)	10
Dynamic Resizing Logic	10
Code Style & Documentation	5
Total	**75 Points**

Hints & Tips

- Break the assignment into small, testable tasks.
- Test adding more characters than the initial capacity.

- Ensure Singleton implementation restricts multiple instances.
- Verify overridden `attack(MiddleEarthCharacter)` methods use polymorphism correctly.

Good luck, little Hobbit and may the light of Eärendil guide your code.