ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY
KUZEY KIBRIS KAMPUSU • NORTHERN CYPRUS CAMPUS

CNG 443

Assignment 2: A Basic Game Boss-Fight Encounter Manager with Threading

2021-2022 Fall

Important notes:

- Your code will be tested by Moss or similar software against cheating attempts. Any cases suspected of plagiarism will result in a loss of grade and might result in further disciplinary actions.
- Submission will be made via ODTUCLASS. Create a “zip” file named ”Name-Surname.zip” that contains all your source code files inside ”Source” directory, readme, javadoc in ”Docs” directory, and your Jar file. Please provide a readme which describes how to compile and run your code. Please read the **“Extra Requirements”** section for more submission related rules and details.
- No late submissions will be accepted.
- You must follow the ODTUCLASS forum for discussions, possible updates, and corrections.
- You must use Java programming language (No less than version 8).
- The due date for this assignment is 26/12/2021 22:55 (Cyprus time).

Introduction:

In assignment 1, we worked on making a basic encounter manager. In this assignment, we will modify the basic encounter manager, add threading and concurrency, and add new behaviors to its entities. The next section of the report contains a quick revision of assignment 1.

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY
KUZEY KIBRIS KAMPUSU • NORTHERN CYPRUS CAMPUS

Scenario and Background (Revision):

MMORPGs (Massively Multiplayer Online Role-Playing Games) are a genre of video games where you can play with other people in many different scenarios and setups. One very popular scenario is PvE content (Player versus Environment). In this scenario, players come together and create groups. Usually, each group would contain at least 3 players with unique roles. One player would have the “Tank” role (called a tank because this player will keep enemies distracted and will take most of their damage), another player would be “Damage Dealer” (will focus on dealing high damage to enemies who are distracted by the “Tank”), and lastly, we have the “Healer” (will heal players when they take damage).

When this group of 3 players enters a room that has a strong enemy (referred to as boss) an encounter begins. The encounter ends when either all players are dead, or the enemy boss is defeated.

In this assignment, you will make a basic, oversimplified boss-fight encounter manager. This encounter manager will keep track of all entities and their stats. It will handle registering actions like attacks or heals and updates players’ stats and attributes accordingly. The encounter manager starts when the fight is initiated, and it stops when the fight ends.

Please note that this is not a full game. You will not be creating a game loop with game rules etc. Imagine that multiple player clients are sending actions to the game server and in the server, there is a component that eventually handles registering the actions and updating attributes. Meaning that it does not handle main game loop rules.

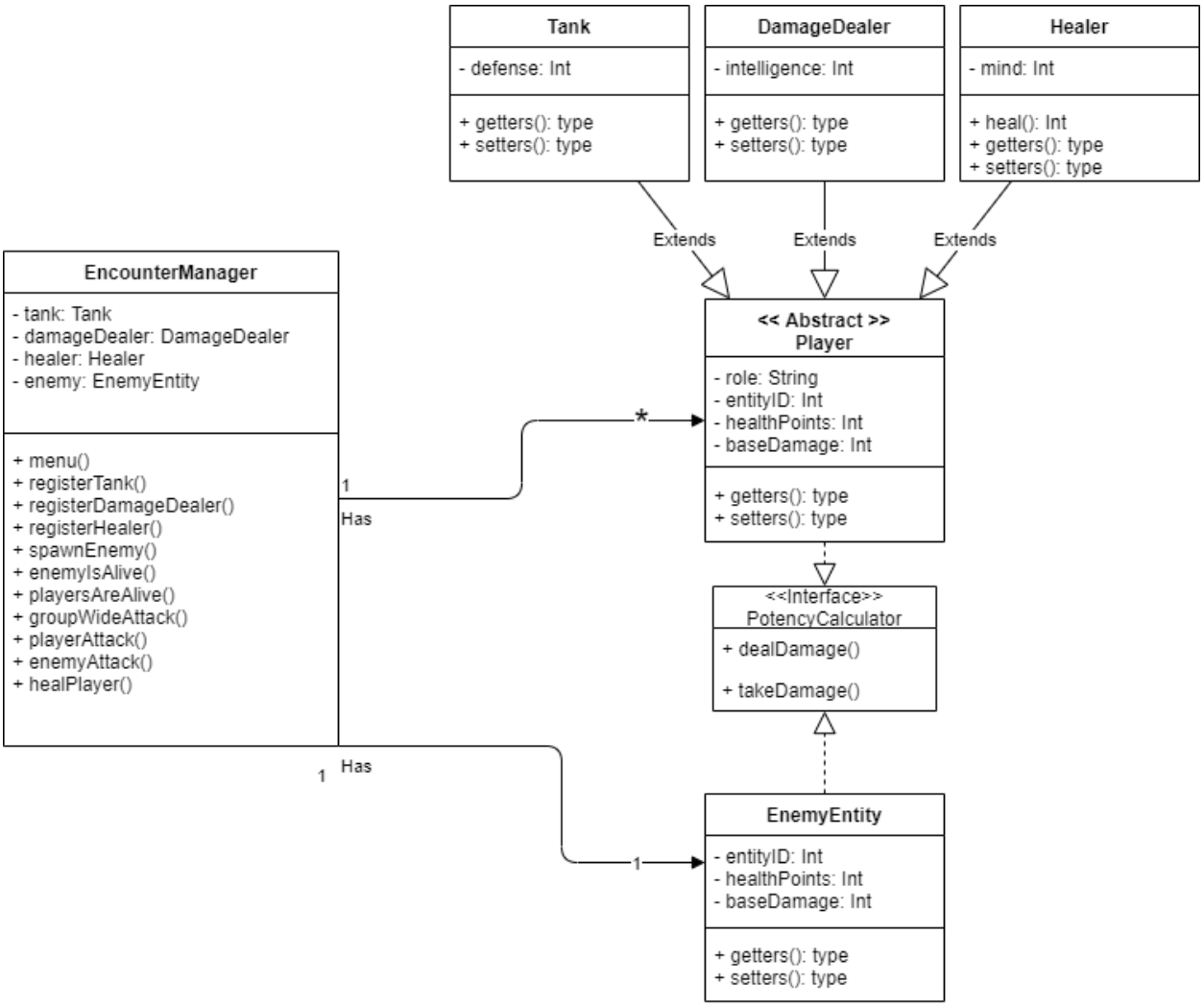

Figure 1: Assignment 1 UML diagram (Old).

Below are some of the function descriptions from **assignment 1**:

enemyIsAlive(): Checks if the enemy is still alive after it took damage. It will be used to control the encounter loop.

playersAreAlive(): When a player dies, this function will check whether all players are dead or not. It will be used to control the encounter loop.

playerAttack(): It handles dealing damage to the enemy based on which player attacked.

healPlayer(): The function will call heal() from the healer and apply the returned heal value to the healed player's HP.

heal(): returns mind+10. Serves as a healing potency calculator.

enemyAttack(): It handles dealing damage to a player.

groupWideAttack(): This function will deal damage to all players.

dealDamage(): The implementation and returned value for this method will depend on the object class:

- DamageDealer: baseDamage + intelligence
- Tank, Healer, EnemyEntity: baseDamage

takeDamage(): The function will take the damageReceived as a parameter and will deduct from HP (healthPoints) the following amounts:

- DamageDealer, Healer, EnemyEntity: damageReceived
- Tank: damageReceived – defense

Please note that you will need to modify the behaviors and parameters of the functions mentioned above to meet the new requirements that will be mentioned in the next section of the report.

In assignment 2, we will change the "Turn-based" nature of the encounter manager where we had to manually enter who takes which action through command line to a "Real-time" status. Our entities will be all performing actions concurrently without having to wait for us to enter instructions through command line. The entities will act as independent agents that try to win the game.

The **Requirements** section of this report talks about the new changes that you need to implement.

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY
KUZEY KIBRIS KAMPUSU • NORTHERN CYPRUS CAMPUS

Requirements:

In this assignment, you will modify assignment 1 (Including the function descriptions that we mentioned earlier) to meet the following requirements:

1) EncounterManager will serve as the singleton class that will manage entities.
2) Player and EnemyEntity classes extend Thread. This means that now, EnemyEntity, Tank, DamageDealer, and Healer will be threads that will be started by the EncounterManager when the encounter starts. They will all have a reference to the EncounterManager from which they will perform the necessary actions.
    - Tank will try to perform an attack every 1 second.
    - DamageDealer will try to perform an attack every 0.5 seconds.
    - Healer will try to perform a heal every 1 second.
    - EnemyEntity will try to perform an attack on the tank every 1 second. However, every 4^{th} attack will be a group wide attack.
3) EncounterManager will ensure that when the healer performs a heal, it will go to the player with the lowest amount of Health Points (HP).
4) EncounterManager will print a message for every action taken (a log to keep track of all changes), for example:

    Damage dealer attacked the enemy (17 damage attack)
    Entities' HP
    Tank: 82
    Damage Dealer: 100
    Healer: 100
    Enemy: 83

    The tank was healed by 18 HP
    Entities' HP
    Tank: 100
    Damage Dealer: 100
    Healer: 100
    Enemy: 83
    …….

5) Since all our entities are now threads that will act independently, we no longer need the interactive mode of EncounterManager where we had to enter data and commands through command line. You may remove that part.

6) You are expected to use threads and threading primitives (for example, “Synchronized” keyword, wait, notify, notifyAll) to achieve the requirements mentioned in this report. You need to keep in mind scenarios where a Tank who is 1 attack away from reaching Zero HP is attacked by the enemy and healed by the healer at the same time. Such race conditions need to be addressed.
7) Remember to do the necessary modifications to the functions from assignment 1 to achieve the behaviors we mentioned in the requirements section.

Remember, the goal here is not to create a balanced scenario or a game loop with rigid rules but to understand how relations and interactions between objects work and to understand threading and concurrency.

The encounter ends when either all players are dead, or the enemy is defeated.

Extra Requirements:

Some extra requirements are listed below:

- Make sure you have a tank, damageDealer, healer, and enemy created and ready so when I run your program, I can directly start the encounter.
- In this course, we have not covered Graphical User Interfaces (GUI), so please provide command-line based interaction.
- For each class, please decide what kind of constructors are required. Unless necessary, do not use public fields. When you use private fields, make sure that you provide getters and setters.
- Pay attention to the overall design, layout, and presentation of your code.
- Make sure that all your methods and fields are commented properly including JavaDoc comments.
- Package all classes into a java-archive package called EncounterManager.jar. With this package, one should be able to run your application by just typing “java -jar EncounterManager.jar”.
- You will Create a “zip” file named ”Name-Surname.zip” that contains all your source code files inside ”Source” directory, readme, javadoc in ”Docs” directory, and the jar file. Please provide a readme which describes how to compile and run your code.

Sample data:

You can use the following sample data to make testing your programs easier when I grade them (If I do not mention an attribute, it means you are free to use any value). This data will be used for the sample run shown in the section below:

tank:

healthPoints: 100
baseDamage: 10
role: Tank
defense: 6

damageDealer:

healthPoints: 100
baseDamage: 10
role: DamageDealer
intelligence: 7

healer:

healthPoints: 100
baseDamage: 10
role: Healer
mind: 8

enemy:

healthPoints: 100
baseDamage: 10

Sample run:

The encounter has started!

Entities' HP
Tank: 100
Damage Dealer: 100
Healer: 100
Enemy: 100

1) Damage dealer attacked the enemy (17 damage attack)

Entities' HP
Tank: 100
Damage Dealer: 100
Healer: 100
Enemy: 83

2) Tank attacked the enemy (10 damage attack)
Entities' HP
Tank: 100
Damage Dealer: 100
Healer: 100
Enemy: 73

3) Enemy attacked the tank (10 damage attack)
Entities' HP
Tank: 90
Damage Dealer: 100
Healer: 100
Enemy: 73

.
.
.
.

13) Enemy attacked all players (10 damage attack)
Entities' HP
Tank: 50
Damage Dealer : 80
Healer: 90
Enemy: 10

14) The tank was healed by 18 HP
Entities' HP
Tank: 68
Damage Dealer : 80
Healer: 90
Enemy: 10

15) tank attacked the enemy (10 damage attack)
Entities' HP
Tank: 68

Damage Dealer : 80
Healer: 90
Enemy: 0

The enemy is dead. The encounter has ended.

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY
KUZEY KIBRIS KAMPUSU • NORTHERN CYPRUS CAMPUS

Assessment Criteria:

This assignment will be marked as follows:

Aspect	Marks (Total 100)
All classes are implemented	10
All class hierarchies are implemented	10
All interfaces are implemented and used	10
For all classes, constructors and data fields are properly implemented	10
All required methods are implemented and work properly	10
Used threading primitives correctly (For example: Synchronized, notify, wait, and etc.)	10
Threads implemented and used correctly	30
Package structure and Jar for invoking the application	10

To get a full mark, your classes should have a constructor with full parameters and JavaDoc comments. Things like code redundancy, methods or classes not working properly or not according to the requirements, not using JavaDoc comments will lead to extra penalties to the items shown in the grading criteria above.