OOP Advanced Exam – ArrPeeGee

You finnaly landed your dream job working in a big game company. All your colleagues are already enjoying the holidays so it is your task to create the rest of the game that will revolutionize the gaming industry. You have to create a RPG game, named ArrPeeGee, totally not copying some of the major ones.

Overview

`ArrPeeGeg` is a game, which lets the user create heroes, participate in different events and fight the forces of evil. Some of the code has already been written by your colleagues, but you have to finish the job.

Structure

The structure of the game circles around **`Heroes and Actions.`**

Heroes

Heroes can be 3 classes:

- **Warrior**
- **Wizard**
- **Necromancer**

All of them have different stats, which you will find in the **Config** class. It's not uncommon to add a new class later in development, so have it in mind.

All heroes should have:

- Name
- Strength - Integer
- Dexterity - Integer
- Intelligence - Integer
- Level - starting from 1
- Health – Calculated with the following formula: **Strength * 10**
- Damage – Calculated with the following formulas:
    - Warrior – (**Strength * 2) + Dexterity**
    - Wizard – (**Intelligence * 5) + Dexterity**
    - Necromancer – **(Intelligence * 2) + (Dexterity * 2) + (Strength * 2)**
- Gold
- A way of knowing if the hero is alive.

Boss

The boss is very similar to the heroes. He only have basic stats and his rewards are different. His health, damage etc. are also in the **Config** class.

The boss has:

- Name
- Health
- Damage
- Gold
- A way of knowing if the boss is alive.

When a boss levels up, he refills his health.

Actions

Actions for now can be only OneVsOne or BossFight:

- **OneVsOne** – exactly two heroes fight to the death.
- **BossFight** – one or more heroes against a **Boss**.

BattleField

Here is where all the action happens. This class is going to be responsible for the creation and usage of the different objects. It should know about all the heroes, bosses and actions and what is happening between them.

Engine

This is where the interaction between the user and the game happens. Make sure that you are not using your battlefield class directly in the main.

Factories

Decouple your classes using factories for creating objects. Singletons won't cut it (they are considered bad practice by some). It would be best if you use reflection for these.

Functionality

You should imlement the functionality of the game. As you see the Heroes, Bosses and Actions are the main entities of the game. They should be dinamically created in the BattleField.

The business logic of the game involves: creating heroes, creating bosses, creating specials for a given hero and creating actions and executing them.

Check below each section, and the functionality it describes.

Targetable

These can attack, take damage and do other fancy stuff. You can find all of their actions in the **Targetable** interface. The main actions are:

- **Attack** – The heroes should be able to attack, dealing damage to the attacked **Target.** When the attack is successful, you should print the following message: **“{hero name} attacked!”.**

If either the attacker or the attacked one is dead, you should print the following message: **"{hero name} is dead! Cannot attack."** or **"{target name} is dead! Cannot be attacked."**

If the attack results in the death of the attacked target, the attacker should take all of his gold and level up. Also you should print the following message: **"{target name} has been slain by {hero name}."**

- **Take damage –** The hero's health should be reduced by the amount of damage he receives.
- **Level Up –** When the hero levels up, all of his stats(STR, DEX, INT) are doubled and he is also fully healed.
- **Give reward –** When someone dies, he should give all of his gold to his killer.
- **Receive reward –** A hero should be able to receive the reward of killing another unit.
- **Check if it is alive**
- Helper methods for names, health, damage etc.

Hero

They have getters and setters for their stats (strength, dexterity, intelligence).

Boss

The boss has similar actions as the heroes, but there are slight differences:

- **Level Up –** When the boss kills a party, he refills his health.
- **ReceiveReward –** When the boss kills a hero, he takes **only 10%** of the hero's gold.

Actions

All actions should be executed immediately after creation. The different actions do different things:

- **OneVsOne –** Two heroes fight to the death. If there are more or less participants you should print the following message: **"There should be exactly 2 participants for OneVsOne!".** The combat mechanics are as follows – **the first entered hero attacks, then the second one attacks** and so on until one of them is dead. When the fight is over, you should print the following message: **"{winner name} is victorious!".**

- **BossFight – X number** of heroes try to defeat the boss. If there are no heroes that try to enter the boss fight, print the following message: **"There should be at least 1 participant for boss fight!".** The combat mechanics are as follows:
    - **First hero attacks, boss attacks first hero**
    - **Second hero attacks, boss attacks second hero and so on…**

  There should not be any attempts to attack a dead target, or a dead hero/boss attacking

  If the heroes manage to kill the boss:

    - All **alive** heroes **level up.**
    - The hero that made the final attack should take the Boss's gold.

- All **alive** heroes should receive an individual reward for participating in the boss fight. (You can find the reward in the **Config** class).
- You should print the following message to the console: **"{boss name} has been slain by:"** and then, each on a new line, all **alive** heroes that killed the boss **ordered by name** in the following format:
  " Name: {name} | Class: {type/class}
  Health: {health} | Damage: {damage}
  {strength} STR | {dexterity} DEX | {intelligence} INT | {gold} Gold"

If the boss defeat all heroes print:

"Boss has slain them all!"

- **If the action doesn't exist, print "Action does not exist."**

BattleField

The battlefield is the class, which controlls all of the game. It should have the following functionality:

- **CreateParticipant** – the battlefield should be able to create heroes and bosses. If the user tries to create a participant that already exists(same name), you should print the following message: **"Participant with that name already exists.".** If the creation is successful, print the following message: **"{participant class} {participant name} entered the battlefield!"**
- **CreateAction –** the battlefield should be able to **create and execute** actions. If the user tries to put heroes that don't exist in the action, print the followint message: **"{participant name} is not on the battlefield! {action name} failed!".**
- **StatisticsParticipants –** print statistics about the heroes on the battlefield in the following format:
  - For heroes
    " Name: {name} | Class: {type/class}
    Health: {health} | Damage: {damage}
    {strength} STR | {dexterity} DEX | {intelligence} INT | {gold} Gold"
  - For bosses
    " Name: {name} | Class: Boss
    Health: {health} | Damage: {damage} | {gold} Gold"

separated with "* * * * * * * * * * * * * * * * * * * *".

If there are no participants on the field, print: **"There are no participants on the battlefield."**

- **Statistics Actions –** print statistics about the actions that ocurred on the battlefield in the following format:

"{action name}", separated by new lines.

- **Remove dead heroes – after each action** you should check for dead participants and remove them from the battlefield and printing: "{participant name} has been removed from the battlefield!". You can then create participants with the same name again.

Bonus Task - Specials

Each hero could have a single special ability (if it is created and assigned to him). Special abilities trigger in certain conditions (different for each hero class) and aid heroes in their adventures.

Exact trigger times:

- Warrior triggers his special **right after he receives damage** and his special's requirements are met
- Wizzard triggers his special **right before dealing damage** and his special's requirements are met
- Necromancer triggers his special **right after dealing damage** and his special's requirements are met

There are **3 special abilities**:

- **Heal** - the hero gains health equal to his points of intelligence every time the ability is triggered. Trigger happens if the hero's health is below or equal to 50%.
- **Toughness** - the hero gains strenght equal to his points of intelligence as long as his health is below or equal to 50%. This effect is **lasts only for the duration of the battle**.
- **Swiftness** - the hero gains dexterity equal to his intelligence points as long as his health is above or equal to 50%. This effect is **lasts only for the duration of the battle**.

NOTE: Bonus task includes second half of I/O tests (15 pts).

Input

The input consists of several commands which will be given in the format, specified below. The program should start receiving commands, until the end command is received.

- `CreateParticipant {hero name} {hero class}`
- `CreateAction {action type} {participant} {participant} … {participant}`
- `CreateSpecial {hero name} {special ability type}`
- `StatParticipants`
- `StatActions`
- `Peace` - ends the program

Output

- All floating point numbers in the output must be rounded to the second symbol after the decimal point.

Constrains

- All **integers** in the input will be in **range [0, 1.000.000.000]**.
- All **input lines** will be **valid and in the format described**.

Examples

Input	Output
CreateParticipant Warr Warrior CreateParticipant Wizz Wizard StatActions StatParticipants CreateAction OneVsOne Warr Wizz StatParticipants Peace	Warrior Warr entered the battlefield. Wizard Wizz entered the battlefield. There are no actions on the battlefield. Name: Warr | Class: Warrior Health: 50.00 | Damage: 14.00 5 STR | 4 DEX | 1 INT | 200.00 Gold * Name: Wizz | Class: Wizard Health: 30.00 | Damage: 23.00 3 STR | 3 DEX | 4 INT | 200.00 Gold * Warr attacked! Wizz attacked! Warr attacked! Wizz attacked! Warr attacked! Wizz has been slain by Warr. Warr is victorious! Name: Warr | Class: Warrior Health: 100.00 | Damage: 28.00 10 STR | 8 DEX | 2 INT | 400.00 Gold Wizz has been removed from the battlefield. Name: Warr | Class: Warrior Health: 100.00 | Damage: 28.00 10 STR | 8 DEX | 2 INT | 400.00 Gold *
CreateParticipant HellRaizor Warrior CreateParticipant Goshko Wizard CreateParticipant DarkWiz Wizard CreateParticipant Autwizard Wizard CreateAction OneVsOne HellRaizor Goshko CreateAction OneVsOne HellRaizor DarkWiz CreateParticipant Diablo Boss CreateParticipant Late Wizard CreateParticipant More Warrior CreateAction BossFight Diablo More HellRaizor Late StatActions	Warrior HellRaizor entered the battlefield. Wizard Goshko entered the battlefield. Wizard DarkWiz entered the battlefield. Wizard Autwizard entered the battlefield. HellRaizor attacked! Goshko attacked! HellRaizor attacked! Goshko attacked! HellRaizor attacked! Goshko has been slain by HellRaizor. HellRaizor is victorious!

StatParticipants Peace	Name: HellRaizor | Class: Warrior Health: 100.00 | Damage: 28.00 10 STR | 8 DEX | 2 INT | 400.00 Gold Goshko has been removed from the battlefield. HellRaizor attacked! DarkWiz attacked! HellRaizor attacked! DarkWiz has been slain by HellRaizor. HellRaizor is victorious! Name: HellRaizor | Class: Warrior Health: 200.00 | Damage: 56.00 20 STR | 16 DEX | 4 INT | 600.00 Gold DarkWiz has been removed from the battlefield. Boss Diablo entered the battlefield. Wizard Late entered the battlefield. Warrior More entered the battlefield. Boss has been slain by: Name: HellRaizor | Class: Warrior Health: 400.00 | Damage: 112.00 40 STR | 32 DEX | 8 INT | 750.00 Gold Name: Late | Class: Wizard Health: 60.00 | Damage: 46.00 6 STR | 6 DEX | 8 INT | 250.00 Gold Name: More | Class: Warrior Health: 100.00 | Damage: 28.00 10 STR | 8 DEX | 2 INT | 250.00 Gold Diablo has been removed from the battlefield. OneVsOne OneVsOne BossFight Name: Autwizard | Class: Wizard Health: 30.00 | Damage: 23.00 3 STR | 3 DEX | 4 INT | 200.00 Gold * Name: HellRaizor | Class: Warrior Health: 400.00 | Damage: 112.00 40 STR | 32 DEX | 8 INT | 750.00 Gold * Name: Late | Class: Wizard Health: 60.00 | Damage: 46.00

	6 STR | 6 DEX | 8 INT | 250.00 Gold * Name: More | Class: Warrior Health: 100.00 | Damage: 28.00 10 STR | 8 DEX | 2 INT | 250.00 Gold *
CreateParticipant Warr Warrior CreateParticipant WizardHealer Wizard CreateSpecial Warr Toughness CreateSpecial WizardHealer Heal CreateParticipant TheBoss Boss CreateAction BossFight TheBoss Warr WizardHealer Peace	Warrior Warr entered the battlefield. Wizard WizardHealer entered the battlefield. Boss TheBoss entered the battlefield. Boss has been slain by: Name: WizardHealer | Class: Wizard Health: 60.00 | Damage: 46.00 6 STR | 6 DEX | 8 INT | 370.00 Gold TheBoss has been removed from the battlefield. Warr has been removed from the battlefield.
CreateParticipant Warr Warrior CreateParticipant Warr Warrior CreateAction BossFight Warr CreateAction BossFight NoOne CreateAction OneVsOne NoOne CreateAction OneVsOne Warr CreateAction NoAction Warr CreateParticipant Boss Boss CreateAction BossFight Warr Boss Peace	Warrior Warr entered the battlefield. Participant with that name already exists. Invalid boss. NoOne is not on the battlefield. BossFight failed. NoOne is not on the battlefield. OneVsOne failed. There should be exactly 2 participants for OneVsOne! Action does not exist. Boss Boss entered the battlefield. Invalid boss.

Tasks

Task 1: High Quality Structure - 60 pts

Refactor anything, which will **improve** the **code quality**.

You can extend from and into all interfaces, but you are not allowed to modify any method signatures.

Achieve good separation of concerns using abstractions and interfaces to decouple classes, while reusing code through inheritance and polymorphism. Your classes should have strong cohesion - have

single responsibility and loose coupling - know about as few other classes as possible. Ensure proper encapsulation.

For this task, submit the whole “app” folder.

Task 2: Correct business logic - 15 pts

You must fix the bugs in the provided skeleton, following the good practices of OOP. The given code provides some functionality, but it does not cover the entire task. Implement the rest of the business logic, using the given code, and implement everything following the requirements specification. Check your solutions in the Judge system.

For this task, submit the whole “app” folder.

Task 3: Unit Testing - 25 pts

Consider starting with this task, so you can build up a solid foundation for your solution.

Test the **`Warrior`** methods**:**

- takeDamage()
- isAlive()
- levelUp()
- giveReward()
- receiveReward()

Test **`OneVsOne`** methods:

- execute()

When testing, **test the classes provided** by the **skeleton, exactly as they are given (you can fix bugs and create private methods, but you can't expose new public methods or constructors and use them in your tests)**.

Submit the **folder** you have put your **tests** into.

Ensure that your test classes are using the **default (empty)** constructors of the tested classes (as they are given in the skeleton).

NOTE: You are **not allowed** to submit **non-test classes** for this task.

Bonus Task: Special Abilities - 15 pts

Implement special abilities for heroes without altering the given interface methods. You are allowed, however, to extend from or into them.

Points for this task are given from I/O tests 6-10.