Due Date: 29.12.2023 23:55

CENG211 – Programming Fundamentals
Homework #4

In this homework, you are expected to implement a simple **"Turn-Based Video Game Application"** in Java.
The submitted homework should fulfill the following concepts of:

- Inheritance
- Interfaces
- Abstract Classes
- Inner Classes
- Exception Handling
- Generics
- Collections

In this application, you will create a simple turn-based video game that plays out a single battle using player inputs. You should create a TBGame class that will **contain necessary objects, display the menu, and play the game according to inputs**.

TBGame consists of **your characters, randomly generated** opponents**, and a queue** that holds the necessary TurnOrder information.

During the game, each character and opponent *perform an action during their turns while targeting someone from the other side*. Their goal is to reduce the score points of all members of the other side to 0.

TBGame should **randomly initialize opponents** and store them inside an ArrayList. Total number of opponents should also be randomly initialized. This number can vary from 1 to 4.

An **opponent could be a Slime, Goblin, Orc or Wolf**. An opponent cannot exist without having a specific type like these. Randomly generated opponents can be made of any possible combination. Each opponent has the following stats:

- opponentId: A unique id per opponent. (E.g., these can simply be "1, 2, 3, 4" if there are 4 opponents.)
- 50 <= points <= 150
- 5 <= attack <=25
- 1<= speed <= 90

The "opponent ids" exist to be used while displaying the TurnOrder information. An opponent is considered defeated when their "points" stat is reduced to 0 or below. The "attack" stat determines how much damage (to a player character) an opponent deals after performing the "Attack" action. The "speed" stat is used for determining the initial turn order.

When an opponent's turn comes, they perform one of the following actions randomly:

- Attack: Randomly select a specific character and deal damage to them a value that is equal to the opponent's attack stat.
- Guard: Don't do anything this turn and receive %50 less damage until the start of the next turn.
- Special: Perform a special action unique to the opponent's type:

- ***Slime***: absorb (It attacks normally and increases its points equal to the damage dealt. Its points cannot go past 150.)
- ***Goblin***: rushingAttack (It attacks normally and gets another turn immediately after the current one. It deals damage 0.7 × attack stat for both of these turns.)
- ***Orc***: heavyHit (It deals damage for 2 × attack stat and skips its next turn.)
- ***Wolf***: callFriend (It does not attack. This action has 20% chance of success. If successful, an identical wolf is created, and it joins the opponents. If this is unsuccessful, nothing happens.)

When the application starts, the player should **enter the number of player-controlled-characters** to create as input. There could be **3 characters at most**. Then, the stats of the characters should also be **randomly generated**. The newly created "Human<W>" characters should implement the "Character<W>" generic interface.

A human could have one of the following jobs: "Knight", "Hunter", "Squire", "Villager". A human cannot exist without having one of these jobs. The jobs are also determined randomly. Each **human** has the following stats:

- name: A unique name for the character that is taken as an input. If the name already exists after taking the input, a **NotAUniqueNameException** should be thrown.
- 100 <= points <= 150
- stamina (This stat always starts as 10.)
- 20 <= attack <=40
- 10 <= speed <= 99

- weapon: The weapon of the character that is randomly assigned. Each weapon has an additionalAttack stat (from 10 to 20). This stat's value is added to the character's attack stat when an action with the weapon is performed. Each weapon has 2 attack types and one of these must be selected when a character attacks with a weapon. A weapon could be one of the following:
  - ***Sword***: It can slash a selected opponent and directly deal the combined attack stat of the character and the sword. It can also stab an opponent with 25% chance of failure. In case of success, the character deals 2 × combined attack damage.
  - ***Spear***: It can stab an opponent for 1.1 × combined attack damage. It can also be thrown to deal 2 × combined attack damage, but the character skips his next turn as a result.
  - ***Bow***: It can shoot a single arrow to deal an opponent 0.8 × combined attack damage. Also, it can shoot two arrows at the same time to deal an opponent 2.5 × combined attack damage.

A **Character<W>** should be able to do the following:

- punch: The character deals "0.8 × attack stat" to a selected opponent. This action reduces stamina stat by 1.
- attackWithWeapon<W>: The character selects an opponent and one the two attack types of his weapons. The selected opponent is damaged according to the rules of weapons described above. All attacks with sword and spear reduce stamina by 2. Bow's single arrow attack uses 1 stamina, and its double arrow attack reduces stamina by 3.
- guard: The character guards until his next turn and receives 75% reduced damage. This action increases stamina by 3.
- run: All characters run away and leave the battle. The menu displays the remaining points of all opponents. Then, the application terminates. This action can be performed without any limitations.

- specialAction: The character performs a special action on a selected opponent according to his job. Each character can use their **special action once per game**. When they try to use it again a **SpecialAlreadyUsedException** should be thrown.
  - ***Knight*** can skip the current turn and deals 3 × attack on his next turn.
  - ***Hunter*** can attack for 0.5 × attack in the current turn and have two turns back-to-back for his next turn. Note that this does not mean he gets a turn immediately after the current one.
  - ***Squire*** can attack for 0.5 × attack in the current turn and increase his stamina to 10.
  - ***Villager*** has no special action.

Note that, when trying to attack with insufficient stamina, an **InsufficientStaminaException** should be thrown.

TBGame class also has **a queue that contains the turn order**. A Turn has its *owner's id or name*. It also has a *AttackModifier* attribute that can be used for some specific actions described above.

These Turns are ordered inside a queue named TurnOrder. The **order is determined according to the speed stat** of the characters and opponents. The ones with higher speed stat should come first in this order.

Normally, after a character or opponent's turn is over, **they are removed from the queue and added to its tail as their next turn**. According to the special action situations described above, there can be certain changes to the way the queue works. If a character or an opponent completely loses their points, they are completely removed from the queue.

TBGame class should handle tasks like displaying the menu and taking inputs. For these tasks, some helper methods are necessary. These helper methods should be inside an **inner class called Menu**.

Also, initialization of the ArrayLists and Queue should have their own helper methods. These should be inside an **inner class called Initializer**.

Whenever menu asks for an input, it should display all points and stamina values. The order of turns with ids and names should also be displayed. After an action is performed, the values that are **affected** should also be displayed.

The **game ends** when one of the sides are completely wiped out. The game also ends when one of the characters chooses to run.

Example Output:

```
Welcome to TBGame!

These opponents appeared in front of you:
Id: 1, Type: Slime, Points: 62, Attack: 9, Speed: 78
Id: 2, Type: Wolf, Points: 121, Attack: 16, Speed: 65
Id: 3, Type: Slime, Points: 104, Attack: 24, Speed: 83

Please enter the number of characters to create: 1
Enter name for your 1st character: Abcde
```

The stats of your 1st character:
Abcde, Job: Hunter, Points: 136, Stamina: 10, Attack: 33, Speed: 80, Weapon: Sword with +12 attack

The battle starts!

*** Turn Order: Opponent 3, Abcde, Opponent 1, Opponent 2 ***

Move 1 – Opponent 3 attacks Abcde. Deals 24 damage.
Abcde, Job: Hunter, Points: 112, Stamina: 10

Move 2 – It is the turn of Abcde.
[1] Punch
[2] Attack with weapon
[3] Guard
[4] Special Action
[5] Run
Please select an option: 2
Please select weapon attack type ([1] Slash [2] Stab): 1
Please enter an opponent id: 1

Move 2 Result: Abcde attacks Opponent 1. Deals 45 damage.
Abcde, Job: Hunter, Points: 112, Stamina: 8
Opponent 1, Type: Slime, Points: 17

*** Turn Order: Opponent 1, Opponent 2, Opponent 3, Abcde ***

Move 3 – Opponent 1 uses Absorb on Abcde. Deals 9 damage.
Abcde, Job: Hunter, Points: 103, Stamina: 8
Opponent 1, Type: Slime, Points: 26

Move 4 – Opponent 2 starts guarding.

Move 5 – Opponent 3 uses Absorb on Abcde. Deals 24 damage.
Abcde, Job: Hunter, Points: 79, Stamina: 8
Opponent 3, Type: Slime, Points: 128

Move 6 – It is the turn of Abcde.
[1] Punch
[2] Attack with weapon
[3] Guard
[4] Special Action
[5] Run
Please select an option: 5

Your character(s) started running away. The battle ends!
Thanks for playing!

Note that the outputs and results are provided randomly as simple examples. Also, you should not forget about **handling incorrect inputs**.

Important Notes:

1. You must use **List**, **ArrayList and Queue** interfaces in this homework.
2. **When throwing exceptions, you should always handle them. The application should never terminate due to an exception.**
3. You can use standard **java.io** packages to read files. You are also free to use anything inside **java.util** packages. Do NOT use other 3rd party libraries.
4. To support **Turkish characters,** you may need to change your project's text file encoding to UTF8: Right click on your project (in package explorer) → Properties → Text file encoding → Other → UTF8 → Apply.
5. You are expected to write clean, readable, and tester-friendly code. Please try to maximize reusability and prevent from redundancy in your methods.
6. To increase the readability of the code, you are expected to **comment** your code as much as possible.
7. You should try to **adhere to the object-oriented principles as much as possible**. For example, you should place your files inside a proper package structure instead of placing everything into a single package.

Assignment Rules:

1. In this lecture's homework, cheating is not allowed. If cheating has been detected, the homework will be graded as 0 and there will be no further discussion on this.
2. You are expected to submit your homework in groups. Therefore, only one of you will be sufficient to submit your homework.
3. Make sure you export your homework as an Eclipse project. You can use other IDEs as well; however, you must test if it **can be executed** in Eclipse. It is a good idea to check your exported project in another group member's PC.
4. Submit your homework through Cloud-LMS.
5. Your exported Java Project should have the following naming format with your assigned group ID (which will be announced on MS Teams) as the given below:

   G05_CENG211_HW4

   Also, the zip folder that your project in should have the same name.

   G05_CENG211_HW4.zip
6. Please beware that if you do not follow the assignment rules for exporting and naming conventions, you will lose points.
7. Please be informed that your submissions may be anonymously used in software testing and maintenance research studies. Your names and student IDs will be replaced with non-identifying strings. If you do not want your submissions to be used in research studies, please inform the instructor (Dr. Tuğlular) via e-mail.