CS102 – Algorithms and Programming II
Lab Programming Assignment 3
Fall 2022

ATTENTION:

- Compress all of the Java program source files (.java) files into a single zip file.
- The name of the zip file should follow the below convention:
  CS102_Sec1_Asgn3_YourSurname_YourName.zip
- Replace the variables "YourSurname" and "YourName" with your actual surname and name.
- You may ask questions on Moodle.
- Upload the above zip file to Moodle by the deadline (if not significant points will be taken off). You will get a chance to update and improve your solution by consulting to the TA during the lab. You will resubmit your code once you demo your work to the TA.

GRADING WARNING:

- Please read the grading criteria provided on Moodle. The work must be done individually. Code sharing is strictly forbidden. We are using sophisticated tools to check the code similarities. The Honor Code specifies what you can and cannot do. Breaking the rules will result in disciplinary action.

Text-based Fantasy Battle

In this assignment, you are going to implement a text-based Java game where two fantasy characters battle. The battle will take place following a turn-based system. In each turn, characters will use their abilities. The character with the higher speed acts first. When a character's current hit points become zero or less, that character is defeated and the battle ends. Unless the user's character is defeated, we should continue with the next battle.

One of the characters will be controlled by the user. For this character, you will display the available abilities each turn of the battle. The user will choose one of the available abilities. Each ability functions differently, they are described throughout this document. The opposing character's ability will be chosen randomly among his/her abilities.

Each character has a specific level (LV), hit points (HP), mana power (MP), speed, base attack, and base magic. All these variables are going to be integers. For each character, you should also keep the current HP and current MP. Current HP and current MP can change during battle when abilities are used. Current HP is an integer between [0,HP], similarly, current MP is in the range [0,MP].

Unless otherwise mentioned stats of a character are based on their current level and are calculated using the following formulas:

$$HP = 20 + 4 * LV$$
$$MP = \frac{HP}{2}$$
$$Speed = LV$$
$$Base\ Attack = \frac{HP}{10} + 1$$
$$Base\ Magic = \frac{MP}{5} + 2$$

These variables should be updated when LV of the character changes; due to dependencies, you should calculate them in a specific order, i.e., calculate HP before MP, etc. Round the results of divisions to the closest integer.

A battle takes place between two opposing characters. Every character has *attack* and *defend* abilities, regardless of race and class. *Attack* ability causes the base attack of the attacking character to be subtracted from the current HP of the opposing character. *Defend* ability causes the character to recover 10% of his/her HP. For example, if a character's current HP is 37 and HP is 60, after using defend he/she will recover 60 * 10 / 100 = 6 HP, resulting in 37 + 6 = 43 current HP.

Additional abilities are available for the characters based on their race and class.

There are 4 races: Humans, Elves, Dwarfs, and Halflings. <u>You should use inheritance for implementing different races.</u> Each race should be a subclass of the character. Each race introduces new rules for calculating the character stats and make additional abilities available.

Humans: While other stats are calculated regularly, HP of humans is calculated using the following formula:

$$HP = 35 + 3 * LV$$

Humans have the *struggle* ability, which causes 25% of the attacking character's HP to be subtracted from the opposing character's current HP. This ability causes the attacker to lose 10 of his/her current HP as a drawback. This ability cannot be used if losing 10 HP would cause the character's current HP to become zero or less.

Elves: MP and Speed of Elves are calculated using the following formulas:

$$MP = HP - 10$$
$$Speed = LV + 4$$

Elves have *mana restore* ability, which causes the character to restore LV * 2 to their current MP. Using this ability costs 2 HP (i.e., subtract 2 from current HP).

Dwarfs: HP, MP, and speed of dwarfs are calculated using the following formulas:

$$HP = 40 + LV$$
$$MP = 10 + LV$$
$$Speed = \frac{LV}{2} + 1$$

Dwarfs have *rest* ability, that restores 20 HP at the cost of 7 MP.

Halflings: Halflings have two additional abilities: *second breakfast* and *mimic*. *Second breakfast* restores 5 HP and 5 MP. *Mimic* causes the character to use a random ability from the opposing character's ability set. First, choose a random ability from the opposing character's available abilities, then, check for the limitations such as MP requirements, if requirements can be satisfied, this character successfully uses the randomly selected ability. Calculations for the ability are done based on this character's stats. For example, if the opposing character is a human and we select *struggle* ability randomly, the damage is calculated using 25% of this halfling's HP.

Characters can be in one of the following three classes: Fighter, Rogue, and Mage. These should be implemented as interfaces, as they will appear together with the character's race. Note that you cannot implement the method's body inside the interface classes, you will determine the method signature in the interface and the body will be implemented inside the classes that implement these interfaces.

Each class enables additional abilities for the character.

Fighter: Being a fighter enables *slash* and *burst* abilities. *Slash* causes a damage of Base Attack * 2 on the opposing character at the cost of 5 MP. **Note that, any ability that costs MP cannot be used if the character's current MP is not enough to cover the ability's MP cost.** *Burst* spends all the current MP of the character to damage the opposing character for Spent MP * 3. For example, if the character has currently 10 MP, he/she will damage the opposing character for 30 and will have 0 MP afterward.

Rogue: Being a rogue enables *quick attack* and *shoot arrow* abilities. *Quick attack* is similar to the attack ability but always hits first at the cost of 2 MP. *Quick attack* ignores the speeds of the characters unless both characters are using *quick attack*, in this case the character with higher speed acts first. *Shoot arrow* ability damages the opposing character for base attack + speed + 2, at the cost of 3 HP. If *shoot arrow* defeats the opposing character, do not subtract 3 HP from this character to avoid both sides being defeated.

Mage: Being a mage enables *fire*, *thunder,* and *blizzard* abilities. For *fire*, first, find the maximum of the two characters' speeds, then damage the opposing character for max speed * 3. Fire costs

2 MP, or in case there is not enough MP, it costs 3 HP. If *fire* defeats the opposing character, do not subtract 3 HP from this character to avoid both sides being defeated. *Thunder* damages the opposing character for base magic * 2 at the cost of 8 MP. Blizzard damages both characters for 12 HP. If this mage has non-zero current MP, current MP is decreased instead of current HP. For example, if the mage has 20 current HP and 7 current MP, after using *blizzard*, he/she will have 15 current HP and 0 current MP. In this example, 7 of the damage is subtracted from the current MP, and the remaining 5 is subtracted from the current HP. Attack ability of the mage class damages for base magic, instead of base attack.

When the program starts, let the user choose a race and a class. Various classes are only available to certain races, this would be realized by which interfaces each class implements. While choosing the class of the character you should limit the options based on the character's race respecting the following table:

Class \ Race	**Human**	**Elf**	**Dwarf**	**Halfling**
Fighter	Available	-	Available	Available
Rogue	Available	Available	-	Available
Mage	Available	Available	Available	-

The new character should be LV 1. Display the initial stats of the character. Then, the user's character will enter a series of battles until he/she is defeated. For each battle, create a random opposing character. LV of this opposing character should be determined randomly based on LV of the user's character. The opposing character is either at the same LV as the user's character or 1 LV below it, unless the user's character is LV 1, in this case opposing character should also be LV 1.

After defeating each opposing character, increase LV of the user's character by 1. In this case, stats of the character should be recalculated, but current HP and current MP is not reset. Restore 10% of the new HP and MP to the current HP and current MP. For example, suppose a character's current HP is 27 after the battle and his/her new HP is calculated as 50. In this case, the current HP should be increased by 50 * 10 / 100 = 5, which results in a current HP of 32. Since the stats are recalculated, display the new stats of the user's character before starting a new battle. The battles should continue with new random opposing characters.

During battle, you should display the current HP and current MP of both characters. Display the available abilities so that the user can choose. Whenever a character uses an ability, print out a message giving the details of what happened, i.e., the ability name and any of the influenced variables.

When the user's character is defeated, display the number of enemies defeated and ask whether the user wants to play again.

For the implementation, you may include class and race information as variables in your character class. Make sure you make use of inheritance for different races, and interfaces for different classes. Make use of polymorphism for different characters in your game loop. To this end, you

may include methods to list the available actions, use the selected ability, increase/decrease LV, current HP or current MP, update/print the stats, etc. into the parent class, so that any child extending the character class would support these functions. Each different ability can be implemented as a different method. Abilities may take the opposing character as a parameter. You are free to include any variables or methods to support the required functionality.

Preliminary Submission: You will submit an early version of your solution before the lab. This version should at least include the following:

- The main logic of the game should be complete, this includes character selection, random enemy creation, the turn-based battles with the user being able to choose the ability to use. We should be able to continue the game by defeating the random opponents.
- You should also complete the character class and at least two of the races. The races you implement should be available for the user to choose from.

You will have time to complete your solution in the lab. Do not forget to make your final submission at the end. Even if you finish the assignment in the preliminary submission, you should submit for the final submission on Moodle.

Not completing the preliminary submission on time results in 50% reduction of this assignment's final grade.