# Basic #

## Part 1 ##

I have implemented the classes Character and Team following the coursework specification

I have added additional member variables to Character: currentHP, currentAtk, currentDef, curreSpd to store the current attributes, besides the base ones. I also defined maxHP to store the maximum HP of the character on the level he is on. I also aded currentEP and nextLevelEP to store both the characters current EP and the EP required to level up. Last but not least, I also added the variable hisTeam, to store the characters team as a property

Additional methods are also added to Character: getRandomNumber, which is used to calculate which attack is going to be used in the strike method for the Boss and Minion classes. I also added a method called zeroHP(), which automatically sets a characters’ current HP to zero.

I have also added the member variable to the Team class: numberOfCharacters, which counts the number of characters currently present in the team.

The Team class also has some additional methods, such as clearTeam, which is used to remove all members of a team, and reset the counter for numberOfMembers to 0. I have also added returnMembers which returns the members ArrayList.

## Part 2 ##

I have implemented the classes Student, AIStudent, CSStudent, CyberStudent and SEStudent following the coursework specification

I have added the additional methods to the Student class: getKP, getMaxKP, and zeroKP, the last one setting the currentKP to 0.

I have also added new classes: CharacterDeadException, KPException, and SameCharacterException, which are all exceptions thrown. The first is thrown, when a character that should not be dead is dead, such as the character himself, that uses an ability, an enemy that is attacked, or a friend that is being healed. The second one is used, when a character does not have enough KP to use the ability chosen, and the third one is used when two chosen characters, that should not be the same, are the same, such as when a character heals a friend.

## Part 3 ##

I have implemented the interface Monster, and classes Minion and Boss following the coursework specification

I added three new methods to the Monster interface: SyntaxError, NullPointerException and ArrayIndexOutOfBoundException.

Both the Boss and Minion classes have the variables: name, to store the characters name, and chance, where a random number is stored to calculate which attack is used by the character in the strike method

## Part 4 ##

I have implemented the classes MonsterTeam, StudentTeam and Guild following the coursework specification

The strategy for the different students is as folllows.

- \*AIStudent\*: First we check if the character has more than half of his full HP. If not, we heal the character. If the character has maximum KP, we use naturalLanguageProcessing, if not, we use selfStudy. Alternatively, if the student has more than half of his maximum HP, we check if the character has maximum KP. If he does we use machineLearning, if not, we use javaProgramming.

- \*CSStudent\*: First we check if the character has more than half of his full HP. If not, we heal the character with selfStudy. Then we check if the character has maximum KP. If not we use javaProgramming. If the character has both more then half of his maximum HP, and has maximum KP, we check if there are any other characters in the CSStudents’ team that are low on health. If there are, we heal one of them with support. If there is no such character, we use pairWorking.

- \*SEStudent\*: First we count how many students have less than half HP. If more than one character has less than half HP we check if the SEStudent has maximum KP. If he does we use groupDiscussion. If the character does not have maximum KP but more than one character is low on health, we check if the SEStudent himself has less than half HP. If he does, we use selfStudy, if not, we attack with javaProgramming. Alternatively, if one student, or less, have less than half of their maximum HP, we check if it is our character that is low on health. If he is we use selfStudy, if he is not we attack. First we check if our character has maximum KP. If he does, we use groupWork, if he does not, we use javaProgramming

- \*CyberStudent\*: First we check if the character has more than half of his full HP. If not, we heal the character with selfStudy. Alternatively, if the student has more than half of his maximum HP, we check if the character has maximum KP. If he does we use cyberAttack, if not, we use javaProgramming.

The move method also selects the target of the students’ attack. The target is selected by first checking if there are any Boss type characters in the enemy team. If there are, the target is one of the Bosses, if there are no Boss type characters, we attack the minion with the lowest health.

We also select the strongest team mate, by comparing their attack attributes. This is used in the pairWorking method.

The getTeam method in the Guild class works on the following principle: If there are less than six members in the guild we add all of them to the team (lessThanFiveMembers method) . This is counted when members are added to the guild.

If there are more than five members, we check how many of them are alive (membersAlive method) . If there are less than six members with their currentHP greater than 0, we add all of them to the team (lessThanSixAlive method).

Alternatively, if there are more than five members alive in the guild we begin the selection process.

This method does not take into consideration the enemy team, instead it tries to create an ideal team (idealTeam method) and if that is unsuccessful, the characters are replced gradually with other characters that are less and less similar to them.

The ideal team consists of two CyberStudents, one AIStudent, one CSStudent and one SEStudent

We first check how many Students are present of each type (presentOfEachType method)

Than we check how many are dead of each type (deadOfEach method)

After that we try to create the ideal team, and if unsuccessful, we start the replacing process (aiNeeded, csNeeded, seNeeded, cyberNeeded). The replacement happens on the following principle: We check if there are more students of a specific type in the guild than there are already in the team plus the number of said type already dead. This basically checks if there are any students of a type that are alive and not in the team already. If a student type satisfies this condition, one of these students is added to the team, replacing a student that did now have enough of that type in the guild.

For example in the ideal team, there is a need for two CyberStudents, however if there is only one CyberStudent in the guild, or only one is alive, we try to see, if there are any AIStudents alive that are not already in the team. If that is the case, we replace the needed CyberStudent with an AIStudent (cyberNeeded = cyberNeeded – 1; aiInTeam = aiInTeam + 1)

The variables: numberOfSE, numberOfCyber, numberOfCS and numberOfAI count how many Students there are of each type. seInTeam, cyberInTeam, csInTeam and aiInTeam count how many students of each type are already in the created team. aiNeeded = 1, cyberNeeded = 2, csNeeded = 1 and seNeeded = 1 show how many of each type are needed to form the ideal team. These numbers are decreased when a character of that type is added, or replaced by another Student of a different type. numberOfMembers counts the number of Students in the guild, while membersInTeam, counts the number of Students already placed in the team. The total number of dead students is stored in the membersDead variable, while the number of dead characters of each type of Student are stored in aiDead, csDead, cyberDead and seDead.

There is also a resetGuild method, that resets every variable and removes all the students in the team. This is needed, to create a new team, after each battle.

## Part 5 ##

I have implemented the class Battle following the coursework specification

I have also added the new class SpeedComparator, which compares characters based on their speed.

The fight method starts by arranging the characters in order in an ArrayList (allCharacters) based on their speed. This is done with the sortTeam method and the SpeedComparator class.

Then a 30 round battle begins, in which time, if the battle is not finished, the method returns null. If the battle is finished in that time, the method returns the winning team, stored in the winningTeam variable

Then each character that is alive moves using the move method implemented in part 5. This only occurs if there are still members alive in the enemy team.

After each character alive moved, we list the currentHP of each character, or if they are dead, we let the user know that a character is dead.

Then, we count the number of enemies and students dead, and if all members of a team are dead, we stop the fight and declare the opposite team the winner. The number of monsters dead is stored in the howManyEnemyDead variable, while the number of students dead is stored in the howManyPlayerDead variable.

## Part 6 ##

I have implemented the class TowerOfMonsters following the coursework specification

The class includes the following methods:

A BufferedReader (reader) is created in the fileReader method.

The fileIsReady method checks whether there are still lines left to read in the text file given. This boolean value is stored in isItReady.

The getLine method gets the next line of the text file, and stores it in the firstLine variable.

The makeEnemyTeam creates an enemy team based on the line read from the text file. There are five minions and five bosses declared in the TowerOfMonsters class. Here we will store the monsters created

There is also a clearTeam method in the TowerOfMonsters class, which gives the null value to all five minions and bosses declared. It also resets the monster team (enemyTeam) based on the method described in part 1 ( Team – clearTeam)

When creating an enemy team, we first check if the file is ready (fileIsReady method) and we use the clearTeam method found in the TowerOfMonsters class.

If the file is ready, we get the next line (getLine method), and we split the line so that we can handle separate monsters. The individual monsteres are stored in: String[] enemies. Then we split each monsters attributes: name, level and type. These are temporarily stored in the String[] enemyName variable until, the name gets stored in name variable, the type in type variable, and the level in strlevel variable.

Then, with the getLevel method, we transform the string (strlevel) into an integer (level)

After that we create the monsters read from the file with the makeMinion and makeBoss methods, depending on their type. These methods store create a new monster and store them in an empty variable declared in the TowerOfMonsters class, and levels them up, to the level given.

In the main method the chickenDinner team is declared, which stores the winning team of each battle, and a levelCount, which counts the number of enemy teams the students have defeated.

The program can be started by giving it a files’ name as an argument, and running TowerOfMonsters. The file should be placed in the same directory as the program, and it should be given as an argument in the following way : „filename.txt”

# Extension #

I have implement the following extension: I added five more Student types AEStudent, which is the Aerospace Electronic Engineering student, the BEStudent, which is the Biomedical Electronic Engineering Student, the MechStudent, which is the Mechatronic Engineering Student, the ElectronicStudent which is the Electronic Engineering Student and the ElectricalStudent which is the Electrical Engineering Student

The \*AEStudent\* has two abilities: spaceInvasion, which deals half the damage of javaProgramming to all enemies, and alienVial, which revives a friendly character and gives them full HP.

The \*BEStudent\* also has two abilities: secondChance, which revives all characters in the BEStudents team, and doubleHeal, which heals the BEStudent to maximum HP.

The \*MechStudent\* also has two abilities: robotInvasion which deals damage to every character in the battle, and automation, which deals damage and gives a small chance to the character to move again.

The \*ElectricalStudent\* also has two abilities: energyTransfer, which deals damage to an enemy, and adds that damage to the character that used the ability, and lifeEnergy, which heals a friend to maximum HP.

The \*ElectronicStudent\* does not have any special abilities, but has all other attributes on maximum, which would be 10.

All of these classes extend to the Student class

The move method also needed to be updated in the StudentTeam with the integer deadMembers, which counts the number of dead members in the team, and the character deadFriend which stores a random dead friend. The characters move as follows:

\*AEStudent\*: First we check how many team mates are dead, and we save a random dead team mate in: deadFriend. Then we check if the character has more than half of his full HP. If not, we heal the character with selfStudy. Alternatively, if the student has more than half of his maximum HP, we check if the character has maximum KP. If not, we use javaProgramming. If the character does have maximum KP and more than half of their HP, we check if there are any dead team mates. If there are we use: alienVial, if not, we use spaceInvasion.

\*BEStudent\*: First we check how many team mates are dead. Then we check if the character has more than half of his full HP. If not, we heal the character. If the character has maximum KP, we use doubleHeal, if not, we use selfStudy. Alternatively, if the student has more than half of his maximum HP, we check if the character has maximum KP. If he does not have maximum KP, we use javaProgramming. If the character does have maximum KP and more than half of their HP, we check how many team mates are dead. If the number of dead team mates is greater than one, we use secondChance, if it is lower than 2 we use javaProgramming.

\*MechStudent\*: First we check how many students are low on health. Then we check if the character himself is low on health. If he is, we heal him with selfStudy. Alternatively, if the student has more than half of his maximum HP, we check if the character has maximum KP. If not, we use javaProgramming. If the character does have maximum KP and more than half of their HP, we check how many team mates have less than half HP. If more than one character is low on health, we use robotInvasion. Alternatively we use: automation.

\*ElectricalStudent\*: First we check if the character has more than half of his full HP. If not, we heal the character. If the character has maximum KP, we use energyTransfer, if not, we use selfStudy. . Alternatively, if the student has more than half of his maximum HP, we check if the character has maximum KP. If not, we use javaProgramming. If the character does have maximum KP and more than half of their HP, we check if there are any characters in his team that need help. If there are such characters we select one and use lifeEnergy. If there are no such characters, we use javaProgramming.

\*ElectronicStudent\*: First we check if the character has more than half of his full HP. If not, we heal the character with selfStudy. Alternatively, if the student has more than half of his maximum HP we use javaProgramming.

The Guild class was also extended with the variables: numberOfMech, numberOfAE, numberOfBE, numberOfElectrical, numberOfElectronic, aeInTeam, beInTeam, mechInTeam, electronicInTeam, electricalInTeam, aeDead, beDead, mechDead, electronicDead and electricalDead, which operate on similar principles as the variables described in part 4.

The resetGuild, deadOfEach, aiNeeded, cyberNeeded, seNeeded, csNeeded and presentOfEachType are also adjusted to take the new characters into consideration.

The extension can be run the same way as the main program.