SE 116 Project “Cannon Fodder” Report

CAN BAYTEKİN 20190607009

MATTHEW OZAN EANES 20210602201

TAREQ M F ABDULLAH ALHAMMOODI 20200601072

Project Description:

We have created an RPG game where you try to attack enemies and 5 levels. You can pick one of 3 characters to play with A fighter, a Tank or a healer. After picking your weapon you will be able to attack your enemies. You can also pick what type of armor you will wear.

Features:

You have the option of picking from 6 Types of weapons. A fire sword, a light sword, a tower shield, Captain America's shield, Wood wand and a magical wand. You have the option to pick each weapon for each character differently.

```
public void CreateWeapons() {

    // Weapons -> Sword
    Item Sword1 = new Item( name: "Fire Sword", type: "sword", value: 6.0, weight: 4.0);
    items.add(Sword1);

    Item Sword2 = new Item( name: "Light Sword", type: "sword", value: 10.0, weight: 2.0);
    items.add(Sword2);
    // Sword's weight will decrease strength but the Sword itself will increase the strength, so we can ignore it.
    // ----------------------------------------------------------------------------------------------------------

    // Weapons -> Shield
    Item Shield1 = new Item( name: "Tower Shield", type: "shield", value: 5.0, weight: 3.0);
    items.add(Shield1);

    Item Shield2 = new Item( name: "Captain America's Shield", type: "shield", value: 8.0, weight: 5.0);
    items.add(Shield2);
    // Shields decrease strength because of the weight and increase vitality
    // ----------------------------------------------------------------------------------------------------------

    // Weapons -> Wands
    Item Wand1 = new Item( name: "Wood Wand", type: "wand", value: 2.0, weight: 1.0);
    items.add(Wand1);

    Item Wand2 = new Item( name: "Magical Wand", type: "wand", value: 1.0, weight: 0.0);
    items.add(Wand2);
    // Wands increase vitality
    // ----------------------------------------------------------------------------------------------------------
```

While in game you have the option to wear armor. There is wood armor and iron armor.

```
public void CreateClothings() {

    // Clothings -> Armors
    Item Armor1 = new Item( name: "Wood Armor", type: "armor", value: 4, weight: 3);
    items.add(Armor1);

    Item Armor2 = new Item( name: "Iron Armor", type: "armor", value: 7, weight: 6);
    items.add(Armor2);
    // Armors decrease strength because of the weight and increase vitality
    // ----------------------------------------------------------------------------
```

You can see what each fighter has in their inventory such as their weapon and armor.

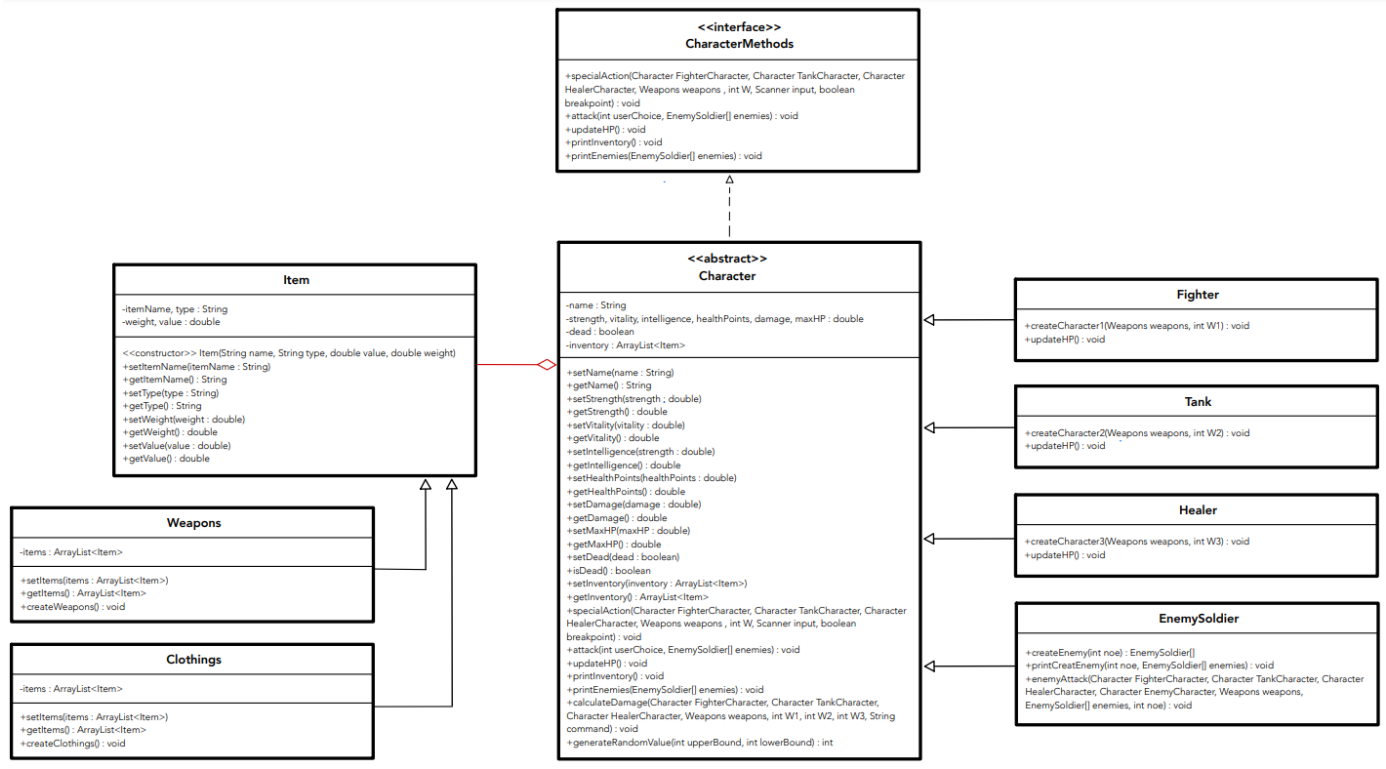
These are for what I have selected for my fighter.

```
inventory
The items owned by the character are :
1 | Fire Sword weight | 4.0
2 | Iron Armor weight | 6.0
```

You can use your weapons special action. You can use your healers wand to heal other fighters like the tank for example.

```
special action
Which character you want to heal?
Valid values : 'FIGHTER', 'TANK', 'HEALER'
tank
Updating the health point...
Tank health points increased by 3.875
Tank has 11.625 HP now.
```

UML Inheritance Diagram:



Implemented Keywords:

Composition:

When there is a strong contact between one class and another, a composition in Java between two objects related with each other occurs. Without the owner or parent class, no other classes can exist.

For example our Character class has options for an attack, a special action, having inventory, etc. We used composition by using an instance variable that refers to other objects.

```java
@Override
public void attack(int userChoice, EnemySoldier[] enemies, Scanner input) {
    boolean checkpoint = false;
    while (!checkpoint) {
        System.out.println(
                "Which enemy you want to attack?" + "\n" +
                        "Valid values : '1', '2', etc..."
        );
```

```java
@Override
public void specialAction(Character FighterCharacter, Character TankCharacter, Character HealerCharacter, Weapons weapons, int W, Scanner input, boolean breakboint) {
    if (weapons.getItems().get(W).getType() == "wand") {
        System.out.println(
                "Which character you want to heal?" + "\n" +
                "Valid values : 'FIGHTER', 'TANK', 'HEALER'"
        );
```

Inheritance:

Inheritance is a crucial component of OOP (Object-Oriented Programming). In Java, it is the technique that allows one class to inherit the features (fields and methods) of another class.

We used inheritance to reuse code written in class “Character” while defining the characters such as “Fighter”.

```java
public class Tank extends Character {
```

```java
public class Fighter extends Character {
```

```java
public class Healer extends Character {
```

Polymorphism:

Polymorphism allows you to create a single interface with numerous implementations. As you can see we used the same interface “updateHP” in both classes “Healer” and “Fighter”.

```
    @Override
    public void updateHP() {
        super.updateHP();
        System.out.println("Healer health points increased by " + (getHealthPoints()/2));
        setHealthPoints(getHealthPoints() + (getHealthPoints() / 2));
        if (getHealthPoints() > getMaxHP()) {
            setHealthPoints(getMaxHP());
        }
        System.out.println("Healer has " + getHealthPoints() + " HP now." + "\n");
    }
}
```

```
    @Override
    public void updateHP() {
        super.updateHP();
        System.out.println("Fighter health points increased by " + (getHealthPoints()/2));
        setHealthPoints(getHealthPoints() + (getHealthPoints() / 2));
        if (getHealthPoints() > getMaxHP()) {
            setHealthPoints(getMaxHP());
        }
        System.out.println("Fighter has " + getHealthPoints() + " HP now." + "\n");
    }
}
```

Interfaces:

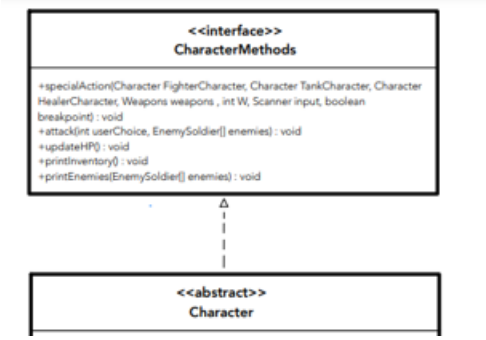
An interface in java is an abstract type that is used to describe a behavior that classes must implement.

We used interfaces to achieve abstraction.

```
public interface characterMethods {
    void specialAction(Character FighterCharacter, Character TankCharacter, Character HealerCharacter, Weapons weapons , int W, Scanner input, boolean breakpoint);
    Integer attack(int userChoice, EnemySoldier[] enemies, Scanner input);
    void updateHP();
    void printInventory();
    void printEnemies(EnemySoldier[] enemies);
```

```
public abstract class Character implements characterMethods {
```

Here it is in our UML diagram:



Abstract Classes:

Data abstraction is the process of concealing some facts from the user and only displaying them what they need to know. They're classes that can't be utilized to make anything (to access it, it must be inherited from another class)

In our case we used it so we could only access it by inheritance with the other characters like fighter healer

```
public abstract class Character implements characterMethods {

    private String name;
    private double strength, vitality, intelligence, healthPoints, damage, maxHP;
    private boolean dead;
    protected ArrayList<Item> inventory;
```

Strings:

```
String option = input.nextLine();
option = option.toUpperCase();
```

Here we have used an Uppercase string method besides the usual string values.

We used it to avoid any errors from user input from different casing. It automatically switches any input into uppercase.

Files:

A file class has several methods for using directories and files like creating directories or files, deleting and renaming them and listing the contents of a directory etc.

In our case we used it to make a text document with the player scores as the kill count.

```
public static void makeinfointotext() throws IOException {
    String results= "The gang name is: " + gangName + " Total dead enemies: " + killcounter;
    File file= new File( pathname: "kod.txt");
    if (!file.exists()){
        file.createNewFile();
    }
    FileWriter fWriter = new FileWriter(file,  append: false);
    BufferedWriter bufferedWriter= new BufferedWriter(fWriter);
    bufferedWriter.write(results);
    bufferedWriter.close();

}
```