*Faculty of Natural Science*
Department of Computer Science
Kristianstad University
Course : DA120C
Program : Bachelor in Software Development
Authors : Wael Natafji , Samuel Joseph Mcmurray , Oussama Safi , Lukas Olsson

# Text Adventure Game

# Table of Contents

# 1. Introduction

We were asked to design and implement an application as a project, so we chose to create a text adventure game which is written in java programming language. The user is tasked with escaping from a labyrinth by finding keys and collecting items. While the goal is to escape, points are earned by collecting “treasures” throughout the game that will increase their score.. The user has two choices either fight or flee both options can have deadly consequences! If the user chooses to fight the monster, they may face a grizzly end. Escape is possible if your fast enough, but you won’t discover as much treasure. The user can find items to help along their journey, the items can rejuvenate health and add additional lives. One of the items being keys to open the locked doors and another being hidden treasure which increases the score. The user will be able to save and load game, the user will also be able to save the high score.

## 1.1 Restrictions

The game has some rules that the user must follow. In order to successfully get to the end the player must not let their health or lives be depleted.

The user is not always going to have the advantage when playing the game, as every new player has to choose between three characters each of which have different statistics that can affect gameplay dramatically. The items will be dropped randomly based on a percentage chance and will be based on rarity with items like treasure being the least rare and usable items being more rare.

Our program also uses no Java FX or any graphics for its interface. While this is to expect as the program is a simple word game, it might not satisfy most modern users.

## 2. Requirements

Table 1 below lists all the requirements for this project. The time estimated (EMH) and time spent on a task (AMH) is measured in man hours. The prio column lists the priorities where Prio 1 is highest priority and Prio 10 is lowest priority.

**Table 1 - Requirements**

| Req. No | Req. Name | Req. Description | EMH | AMH | Prio |
|---|---|---|---|---|---|
| 1 | MVC Architecture | Build MVC Architecture within java project | 2 | 3 DONE | 1 |
| 2 | Start Menu | Create start menu with for beginning the game and choosing character | 2 | 1.5 DONE | 2 |
| 3 | Normal game menu | Create game menu for use while within the labyrinth (not encounters) | 3 | 4 DONE | 3 |
| 4 | Implement character class | Add character variables and attributes to character class | 1 | 1.5 DONE | 4 |
| 5 | Implement hero subclass | Add hero variables and attributes to hero subclass | 1.5 | 2 DONE | 5 |
| 6 | Implement monster subclass | Add monster variables and attributes to monster subclass | 0.5 | 0.5 DONE | 6 |

| | | | | | |
|---|---|---|---|---|---|
| 7 | Design Map Outside Java | Create an outline for the map using pen and paper. Place doors and objects within etc. | 0.5 | 1 DONE | 7 |
| 8 | Fill Room Class | Fill room class with appropriate variables | 0.5 | 0.5 DONE | 8 |
| 9 | Implement Designed Map | Within java framework implement the map in the | 2 | 3+ ongoing | 9 |
| 10 | Create initial story elements | Create a story element including goal and necessary info | 0.5 | 1 DONE | 10 |
| 11 | Add Monsters | Create the various possible monsters to encounter | 2 | | 11 |
| 12 | Map navigation | Navigate the map by taking the commands in the normal game menu | 2 | | 12 |
| 13 | Encounter Menu | Add encounter menu with inputs and reactions from players | 3 | | 13 |
| 14 | Printable map screen | Map screen that shows location and where you've been | 2 | | 14 |
| 15 | Death Menu | To give options for what to do upon | 1.5 | | 15 |
| 16 | Treasure Score | Track treasure pickup and list score at end of game | 2 | | 16 |

| | | | | | |
|---|---|---|---|---|---|
| 17 | High Score | Ability to check highest score | 4 | | 17 |
| 18 | Additional Flavor Text | Add more flavor text for entering rooms, attacks and finish other placeholders | 4 | | 18 |
| 19 | Save game function | Add ability to save and quit | 3 | | 19 |
| 20 | Implement Item Functions | Add functionality to items | 4 | | 20 |
| 21 | Key and Doors | Implement doors and keys to function within game | 4 | | 21 |
| 22 | Level System | Use exp value from monsters to increase level providing health benefits etc. | 2 | | 22 |
| 23 | Boss Fight | Add final boss for end game finish | 2 | | 23 |
| 24 | End Screen Menu Win | Add a screen for when you win | 1 | | 24 |
| 25 | Difficulty System | Modify existing enemies and player health etc to create easier/more difficult fights | 4 | | 25 |

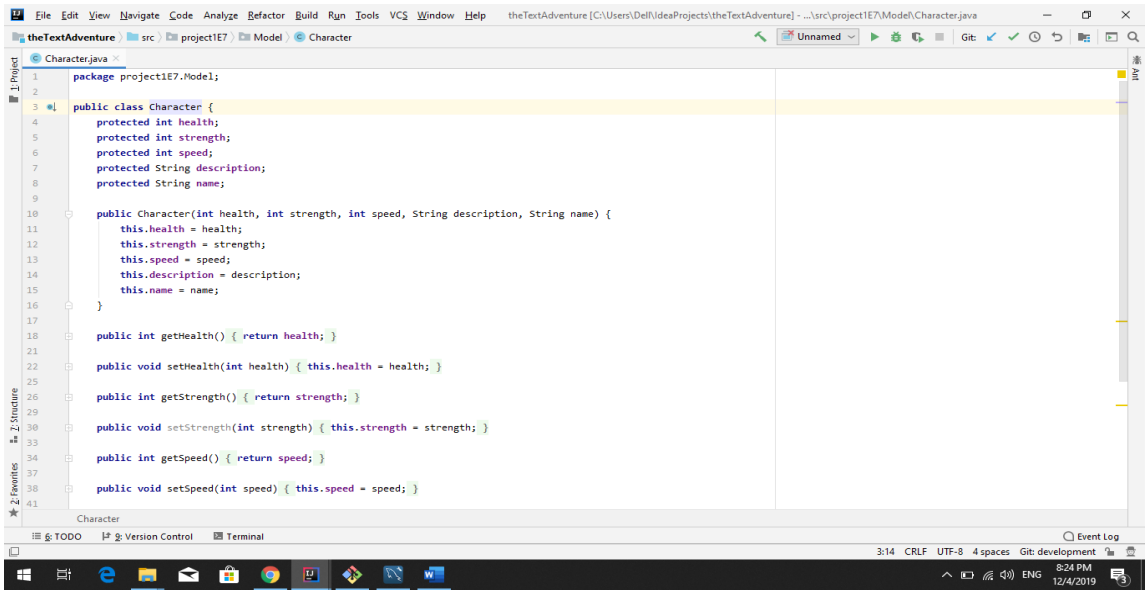
# 3. Design and Implementation

## 3.1 Model Package

The model package holds the data of the objects, the objects are instantiated in the main and then connected to the view and then to the controller.

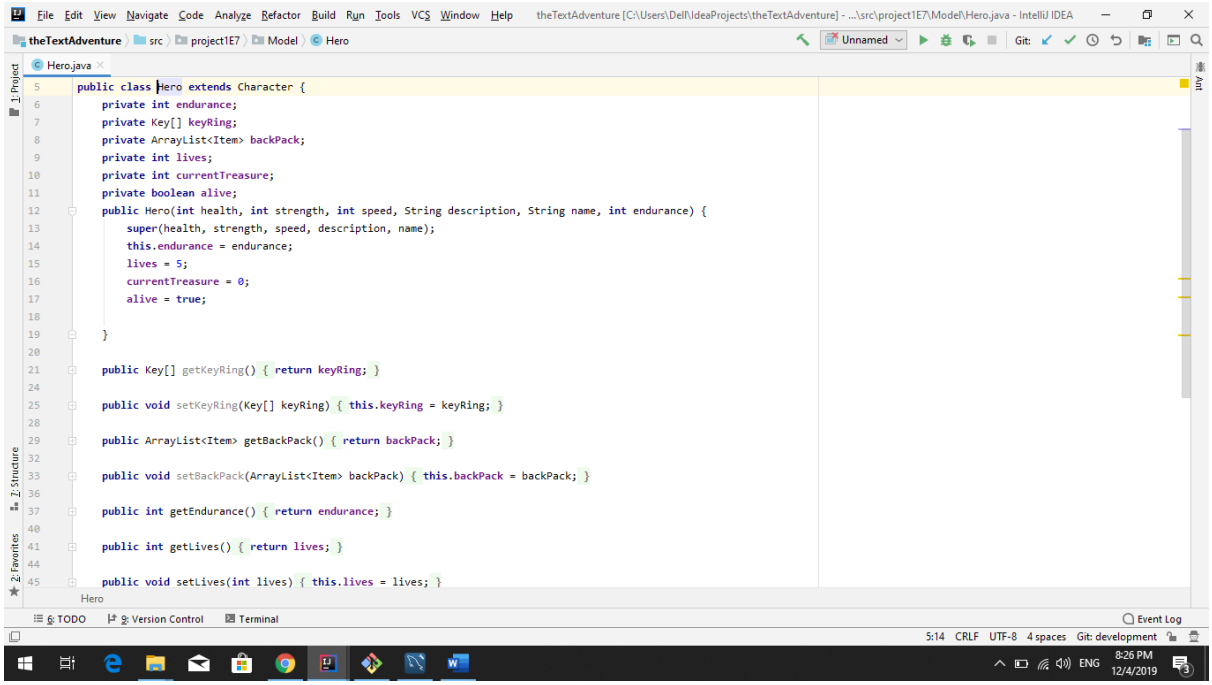
### 3.1.1 Character Class Model

The character class is the super class of the hero class and the monster class.



### 3.1.2 Hero Class Model

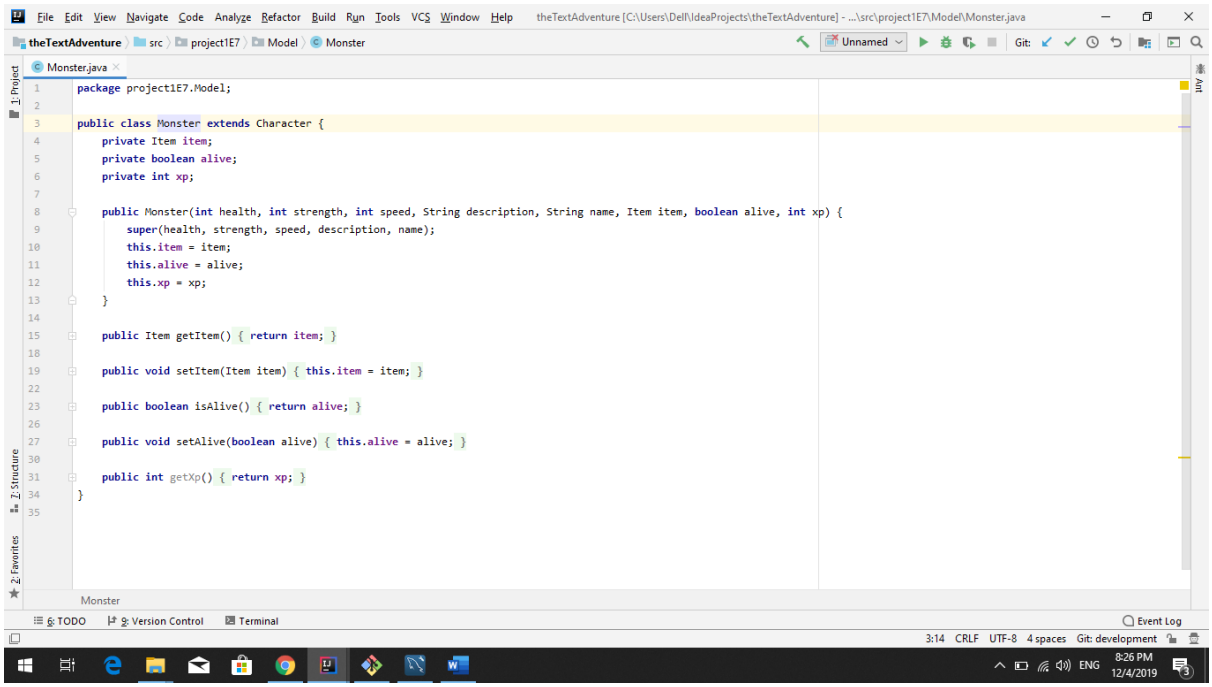
The hero class inherits from the character class has association with main class with aggregation from both item class and key class.



```
public class Hero extends Character {
    private int endurance;
    private Key[] keyRing;
    private ArrayList<Item> backPack;
    private int lives;
    private int currentTreasure;
    private boolean alive;
    public Hero(int health, int strength, int speed, String description, String name, int endurance) {
        super(health, strength, speed, description, name);
        this.endurance = endurance;
        lives = 5;
        currentTreasure = 0;
        alive = true;

    }

    public Key[] getKeyRing() { return keyRing; }

    public void setKeyRing(Key[] keyRing) { this.keyRing = keyRing; }

    public ArrayList<Item> getBackPack() { return backPack; }

    public void setBackPack(ArrayList<Item> backPack) { this.backPack = backPack; }

    public int getEndurance() { return endurance; }

    public int getLives() { return lives; }

    public void setLives(int lives) { this.lives = lives; }
```

### 3.1.3 Monster Class Model

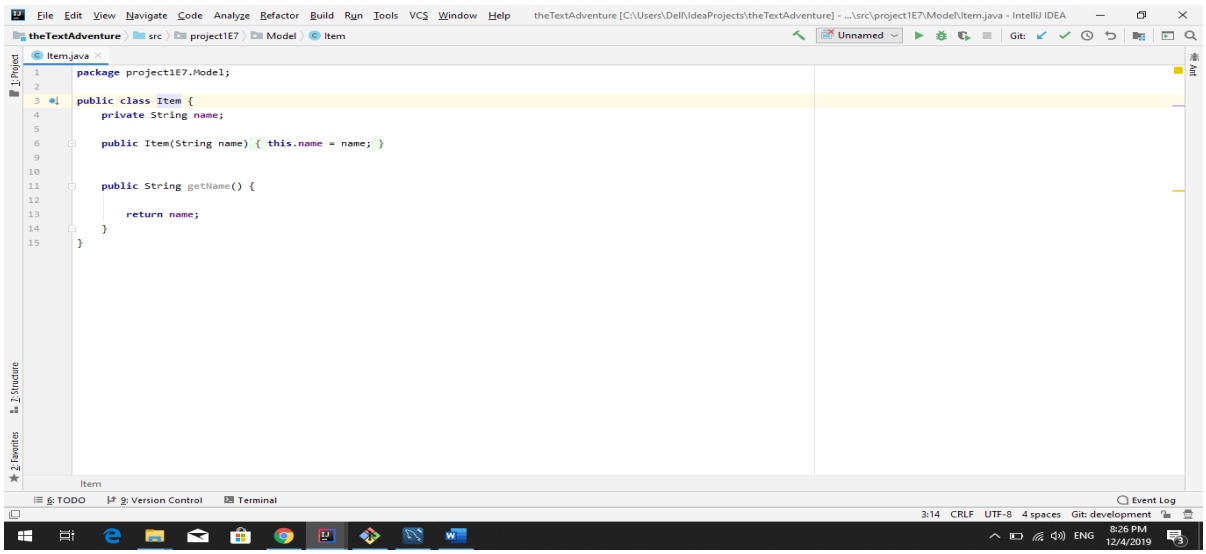
The monster class inherits from the character class with aggregation from item class and association with room class.



```
package project1E7.Model;

public class Monster extends Character {
    private Item item;
    private boolean alive;
    private int xp;

    public Monster(int health, int strength, int speed, String description, String name, Item item, boolean alive, int xp) {
        super(health, strength, speed, description, name);
        this.item = item;
        this.alive = alive;
        this.xp = xp;
    }

    public Item getItem() { return item; }

    public void setItem(Item item) { this.item = item; }

    public boolean isAlive() { return alive; }

    public void setAlive(boolean alive) { this.alive = alive; }

    public int getXp() { return xp; }
}
```

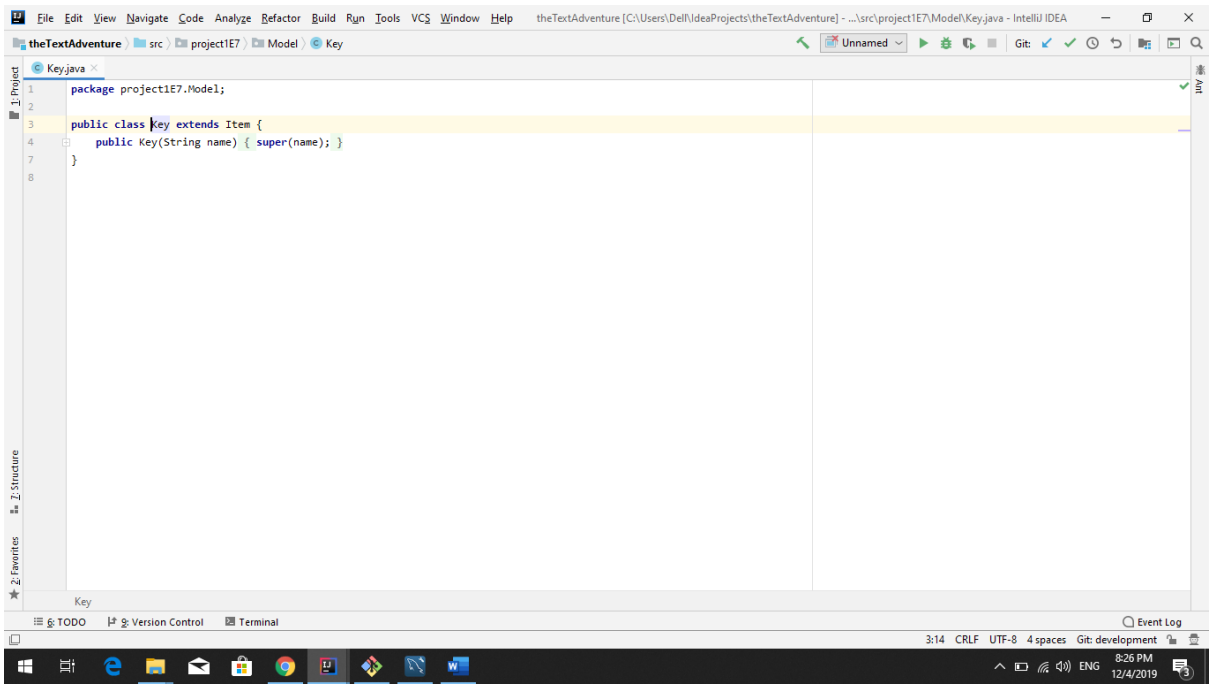
### 3.1.4 Item Class Model

The item class is the super class for key class, treasure class, heart class, healthpotion class, and coffee class. It has an association with room class aggregation with both monsters and players.



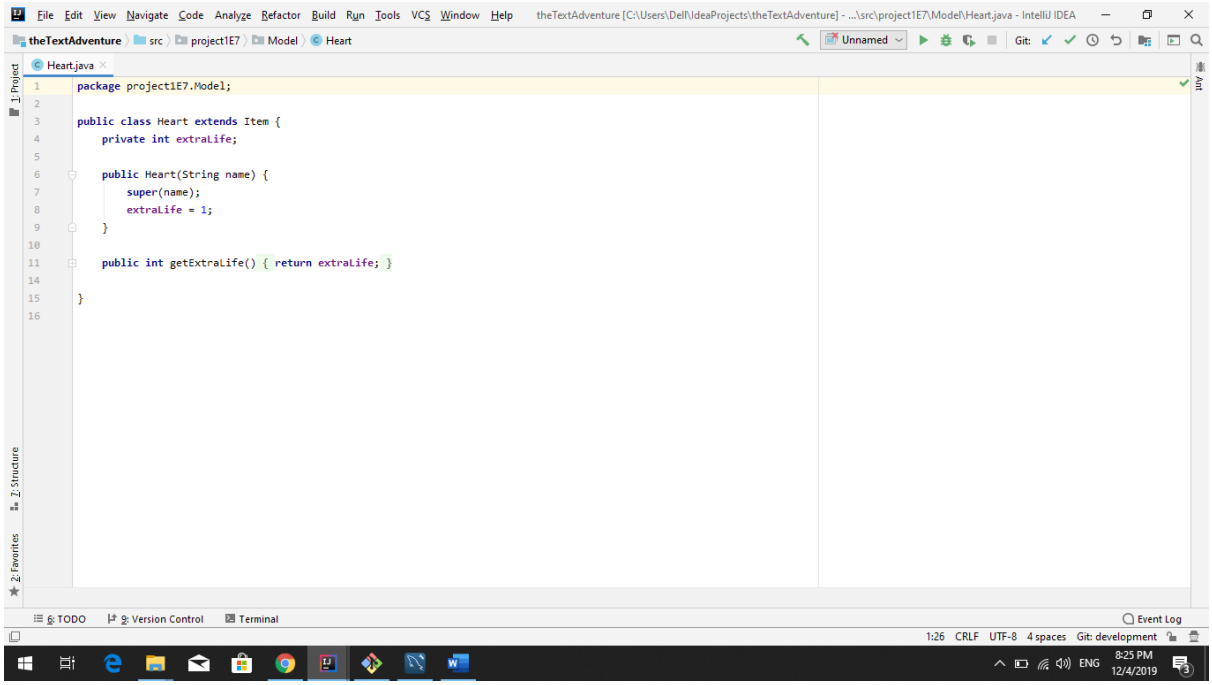

### 3.1.5 Key Class Model

The key class inherits from the item class and has an aggregation with the hero class.



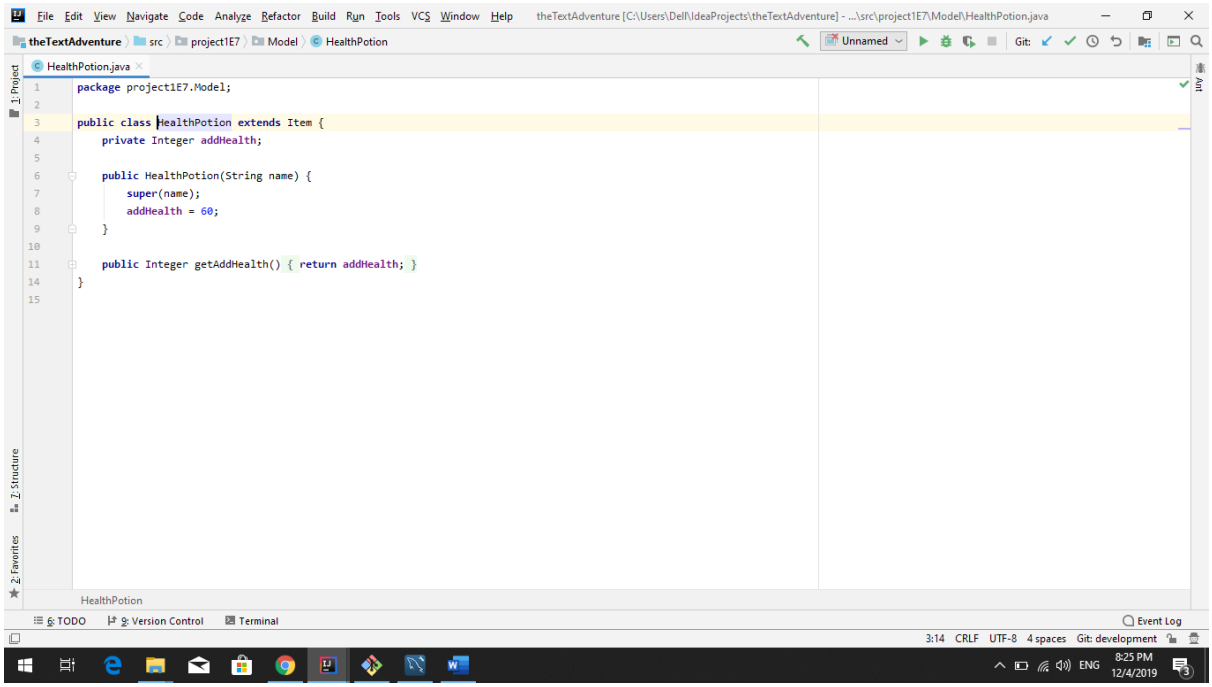
### 3.1.6 Heart Class Model

The heart class inherits from the item class.



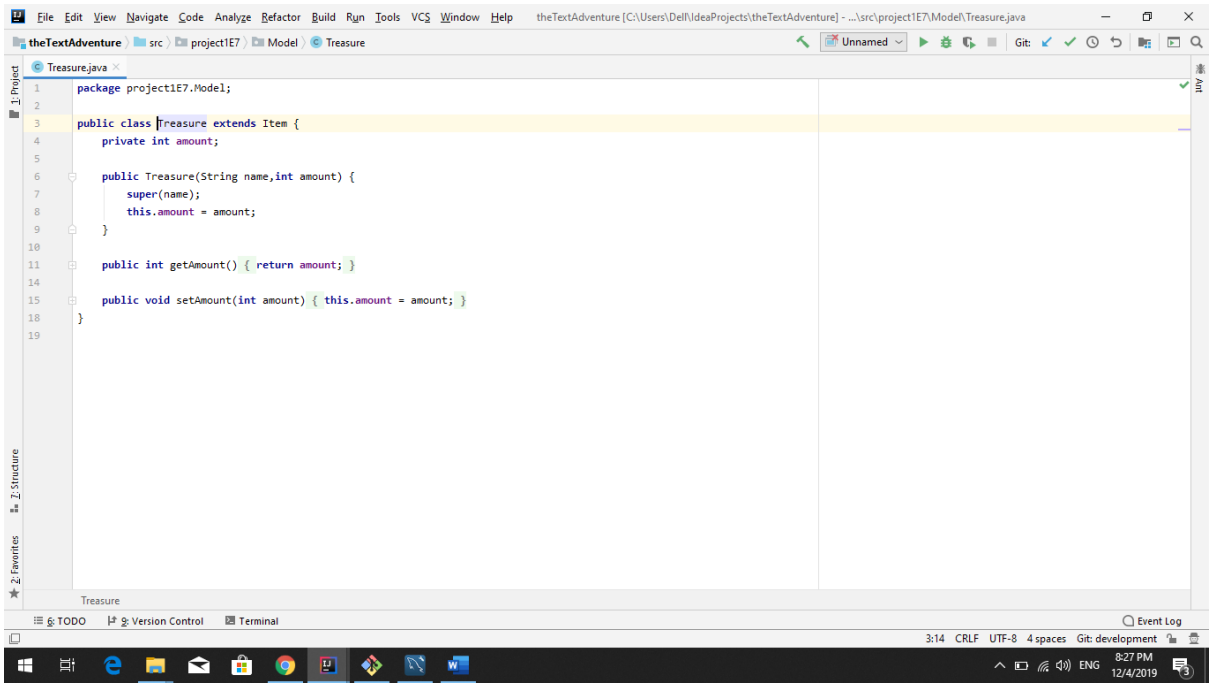

### 3.1.7 HealthPotion Class Model

The healthpotion class inherits from the item class.



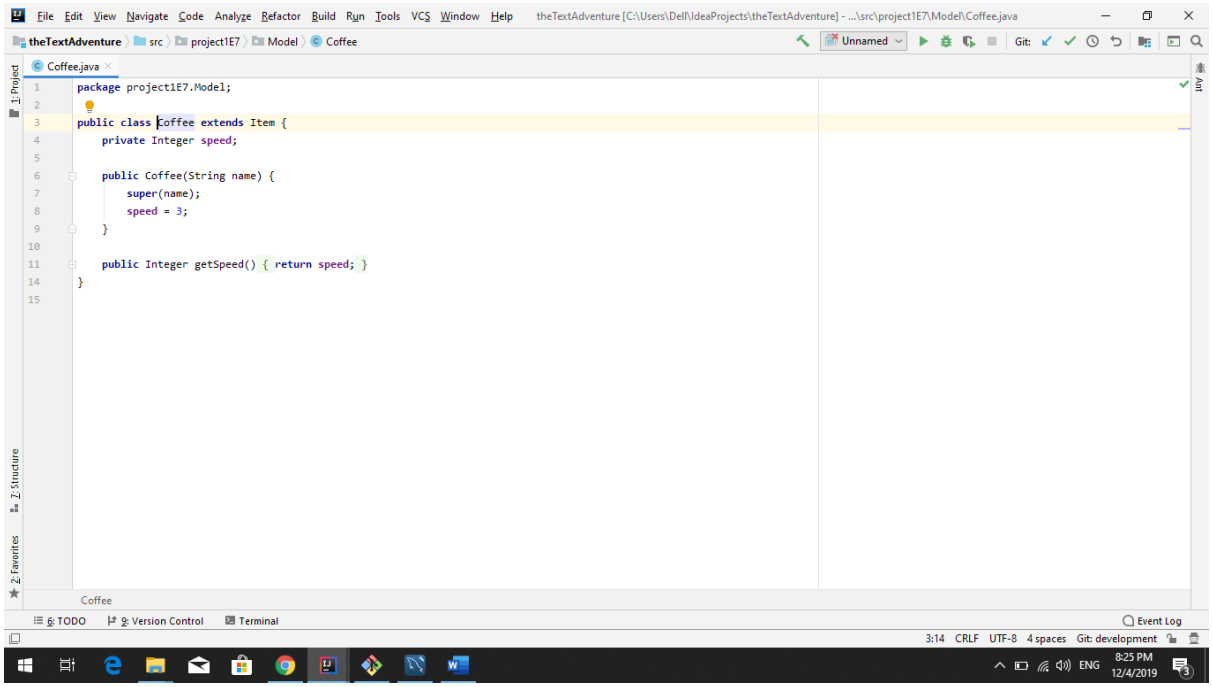
### 3.1.8 Treasure Class Model

The treasure class inherits from the item class.



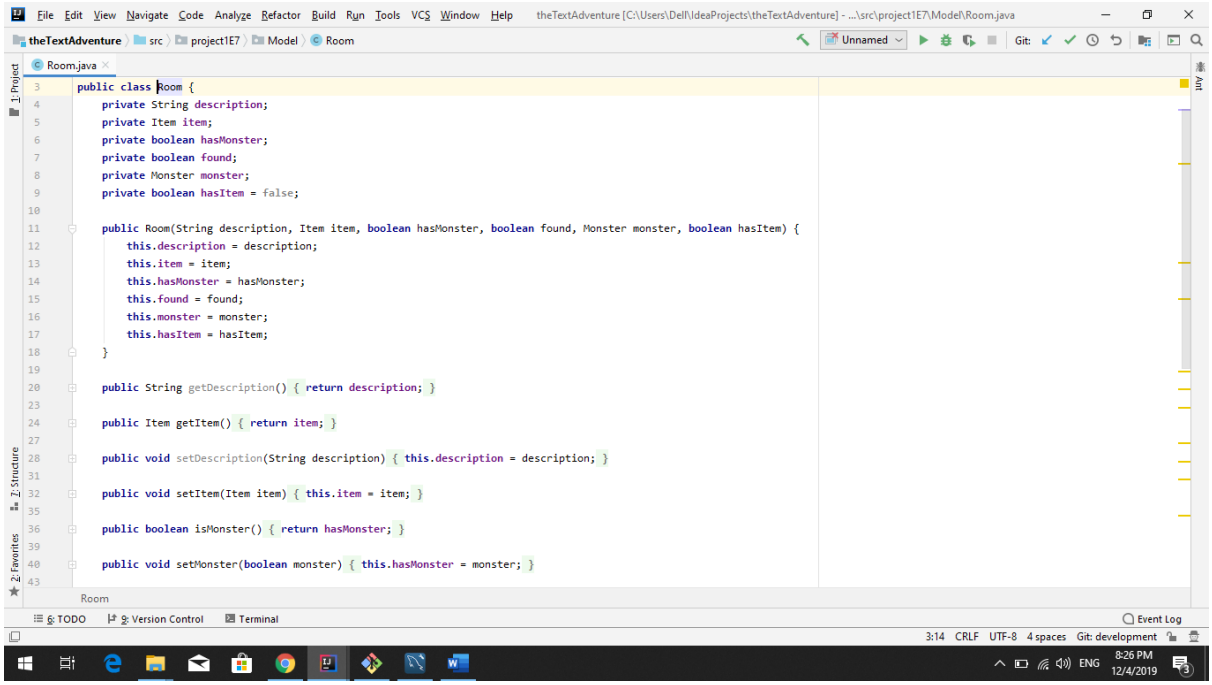

### 3.1.9 Coffee Class Model

The coffee class inherits from the item class.



### 3.1.10 Room Class Model

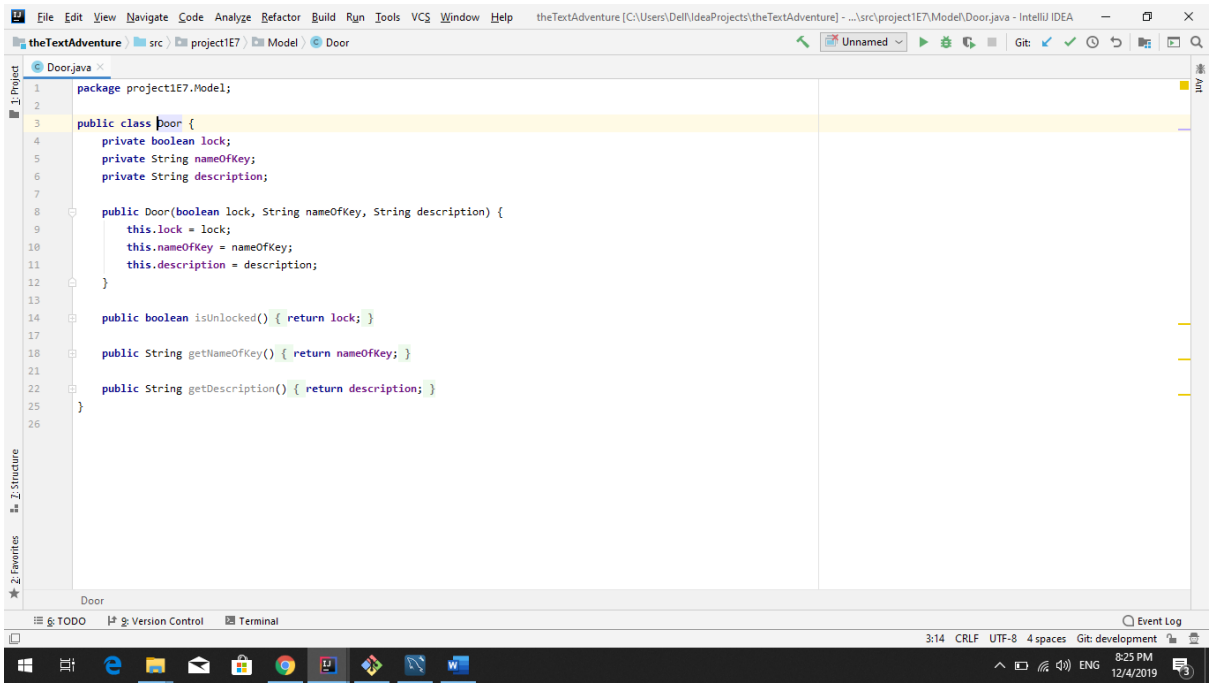
The room class has an association with both item class and monster class, it also has an aggregation with door class and composition with main class.



```java
public class Room {
    private String description;
    private Item item;
    private boolean hasMonster;
    private boolean found;
    private Monster monster;
    private boolean hasItem = false;

    public Room(String description, Item item, boolean hasMonster, boolean found, Monster monster, boolean hasItem) {
        this.description = description;
        this.item = item;
        this.hasMonster = hasMonster;
        this.found = found;
        this.monster = monster;
        this.hasItem = hasItem;
    }

    public String getDescription() { return description; }

    public Item getItem() { return item; }

    public void setDescription(String description) { this.description = description; }

    public void setItem(Item item) { this.item = item; }

    public boolean isMonster() { return hasMonster; }

    public void setMonster(boolean monster) { this.hasMonster = monster; }
```

### 3.1.11 Door Class Model

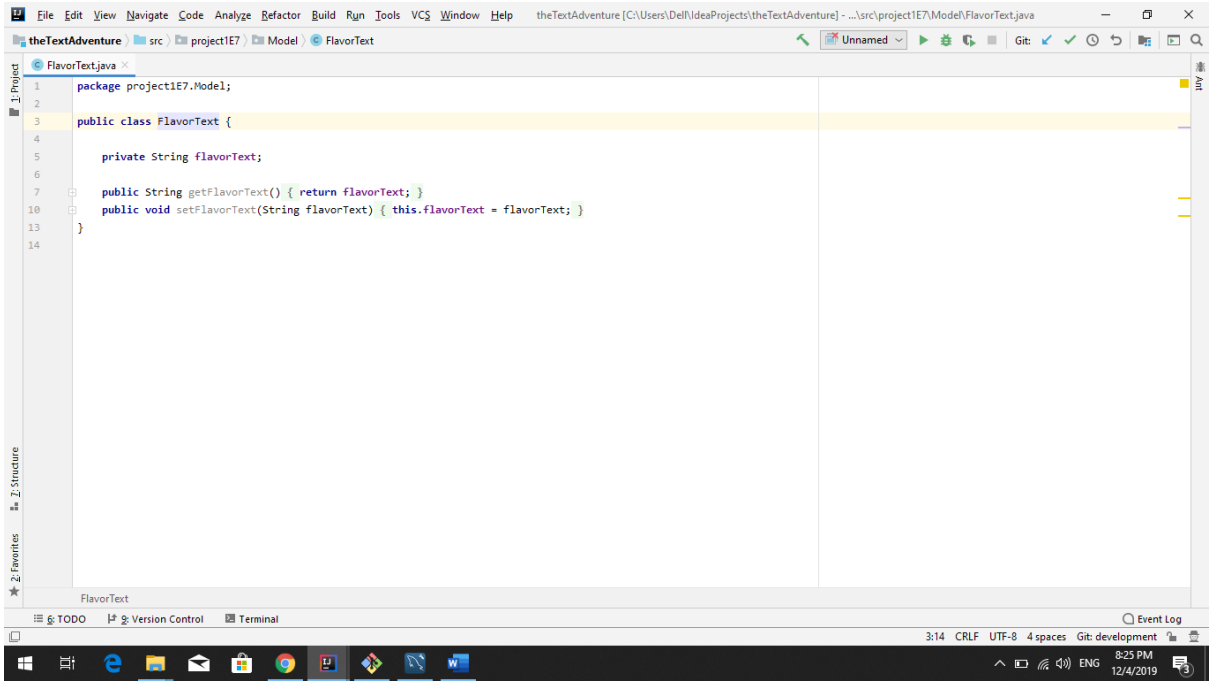
The door class has an aggregation with the room class.



```java
package project1E7.Model;

public class Door {
    private boolean lock;
    private String nameOfKey;
    private String description;

    public Door(boolean lock, String nameOfKey, String description) {
        this.lock = lock;
        this.nameOfKey = nameOfKey;
        this.description = description;
    }

    public boolean isUnlocked() { return lock; }

    public String getNameOfKey() { return nameOfKey; }

    public String getDescription() { return description; }
}
```

### 3.1.11 FlavorText Class Model

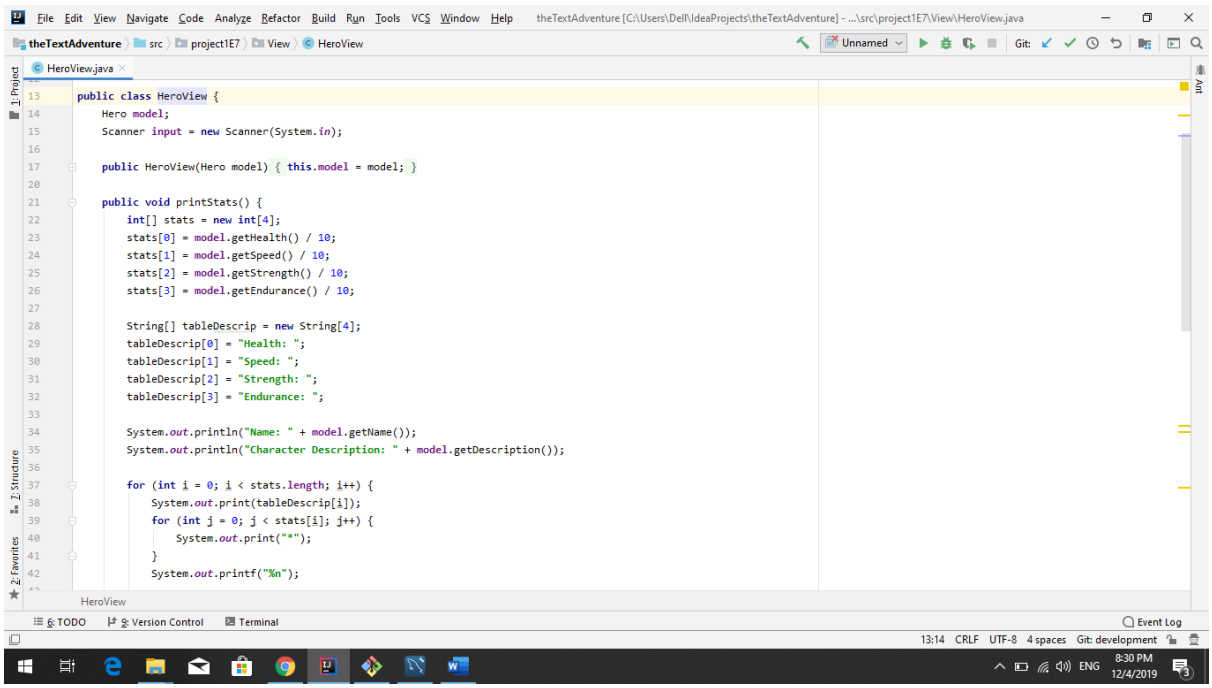
The FlavorText is a temporary class anything done in here will be moved once we figure out where everything will go.



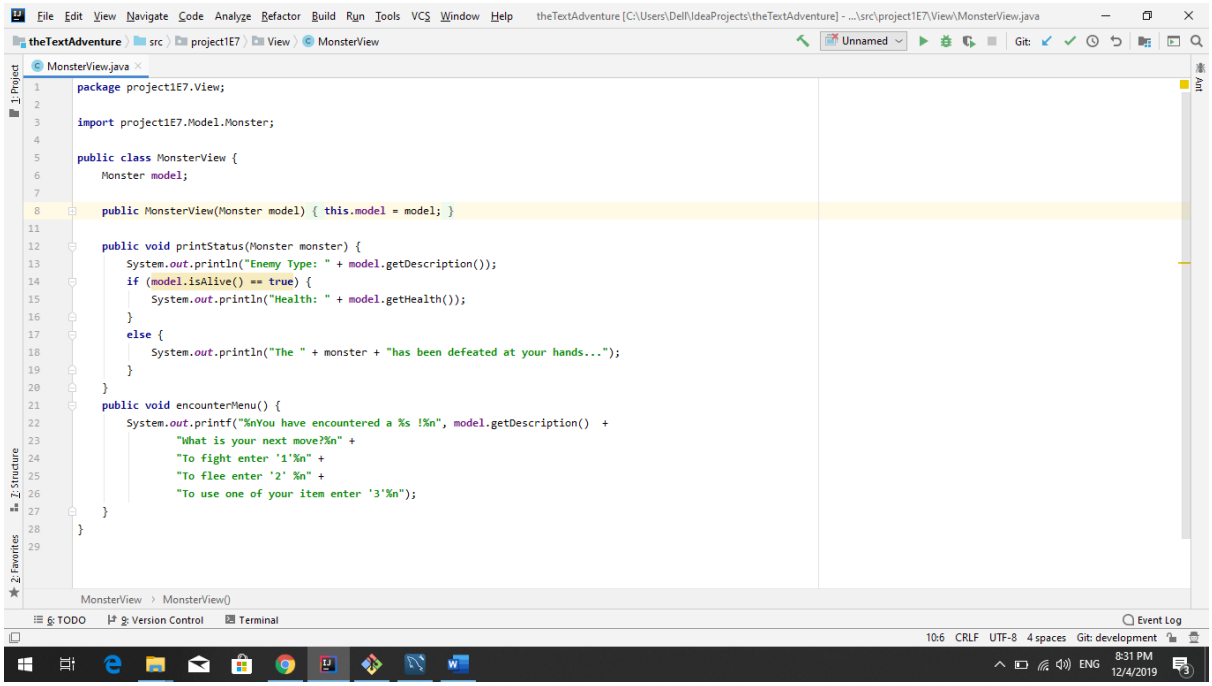

## 3.2 View Package

The view package is what will eventually hold everything that will be printed out to the screen for the user to see sent from the location related to what will be printed takes the model as a parameter.

### 3.2.1 HeroView Class

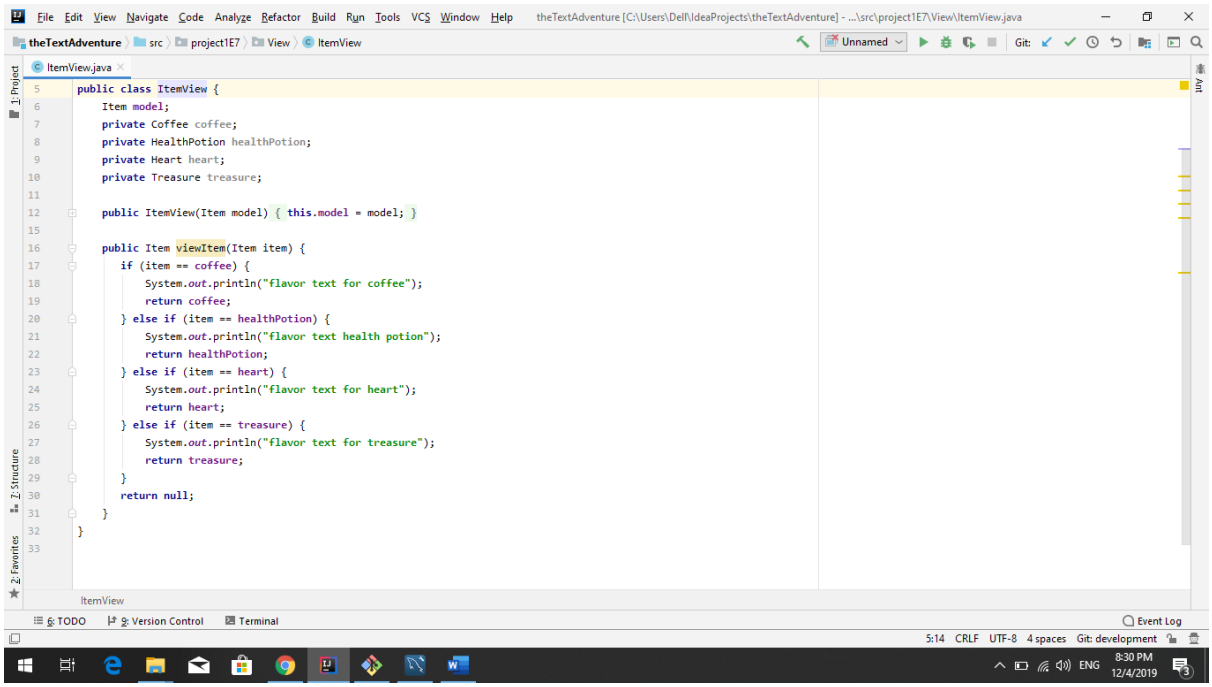


### 3.2.2MonsterView Class



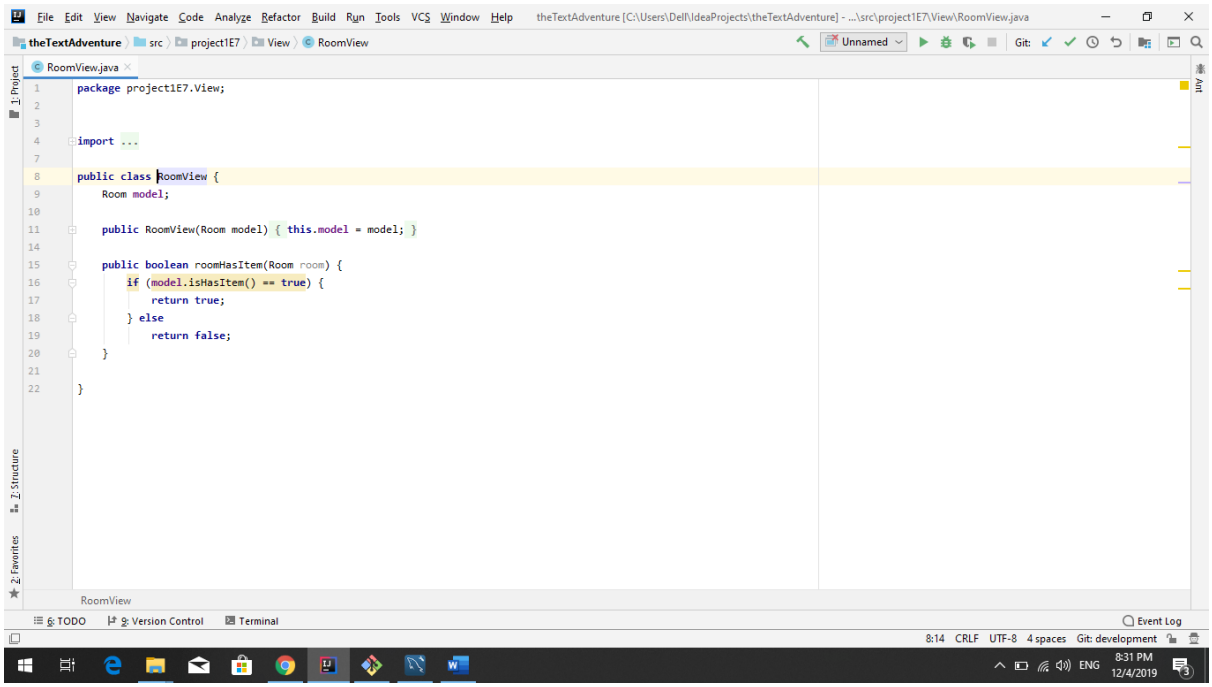
```
package project1E7.View;

import project1E7.Model.Monster;

public class MonsterView {
    Monster model;

    public MonsterView(Monster model) { this.model = model; }

    public void printStatus(Monster monster) {
        System.out.println("Enemy Type: " + model.getDescription());
        if (model.isAlive() == true) {
            System.out.println("Health: " + model.getHealth());
        }
        else {
            System.out.println("The " + monster + "has been defeated at your hands...");
        }
    }
    public void encounterMenu() {
        System.out.printf("%nYou have encountered a %s !%n", model.getDescription()  +
                "What is your next move?%n" +
                "To fight enter '1'%n" +
                "To flee enter '2' %n" +
                "To use one of your item enter '3'%n");
    }
}
```

### 3.2.3 ItemView Class



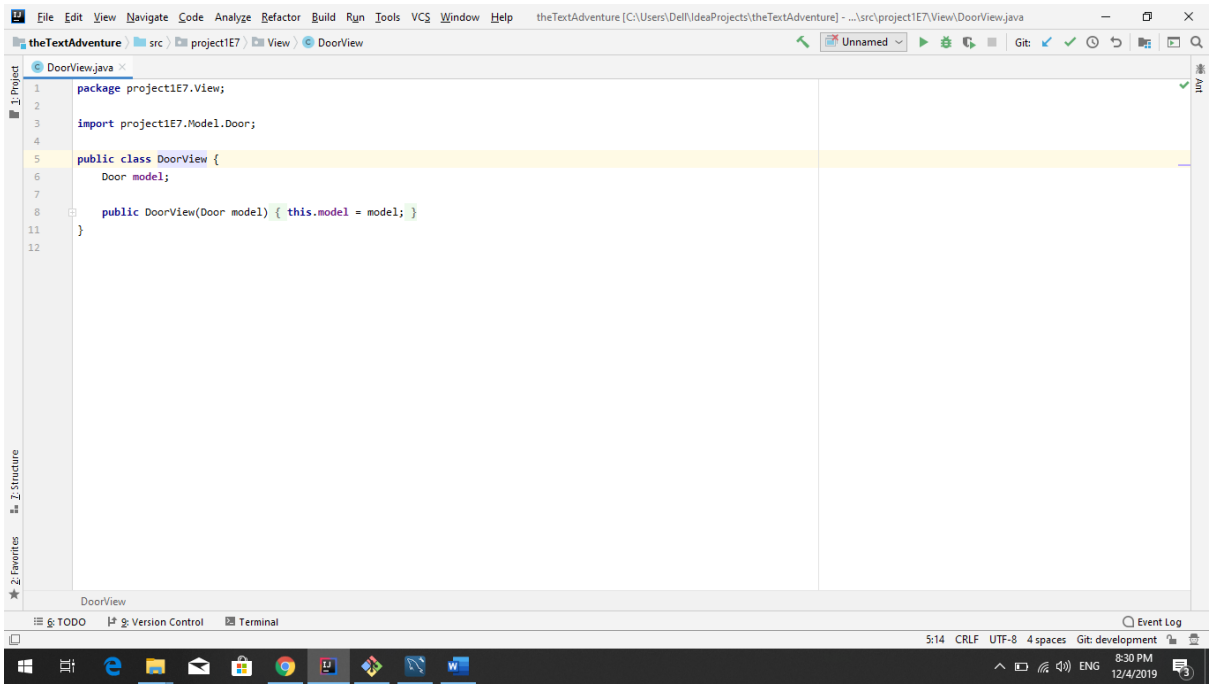
```
public class ItemView {
    Item model;
    private Coffee coffee;
    private HealthPotion healthPotion;
    private Heart heart;
    private Treasure treasure;

    public ItemView(Item model) { this.model = model; }

    public Item viewItem(Item item) {
       if (item == coffee) {
           System.out.println("flavor text for coffee");
           return coffee;
       } else if (item == healthPotion) {
           System.out.println("flavor text health potion");
           return healthPotion;
       } else if (item == heart) {
           System.out.println("flavor text for heart");
           return heart;
       } else if (item == treasure) {
           System.out.println("flavor text for treasure");
           return treasure;
       }
       return null;
    }
}
```

### 3.2.4 RoomView Class



```
package project1E7.View;


import ...

public class RoomView {
    Room model;

    public RoomView(Room model) { this.model = model; }

    public boolean roomHasItem(Room room) {
        if (model.isHasItem() == true) {
            return true;
        } else
            return false;
    }

}
```

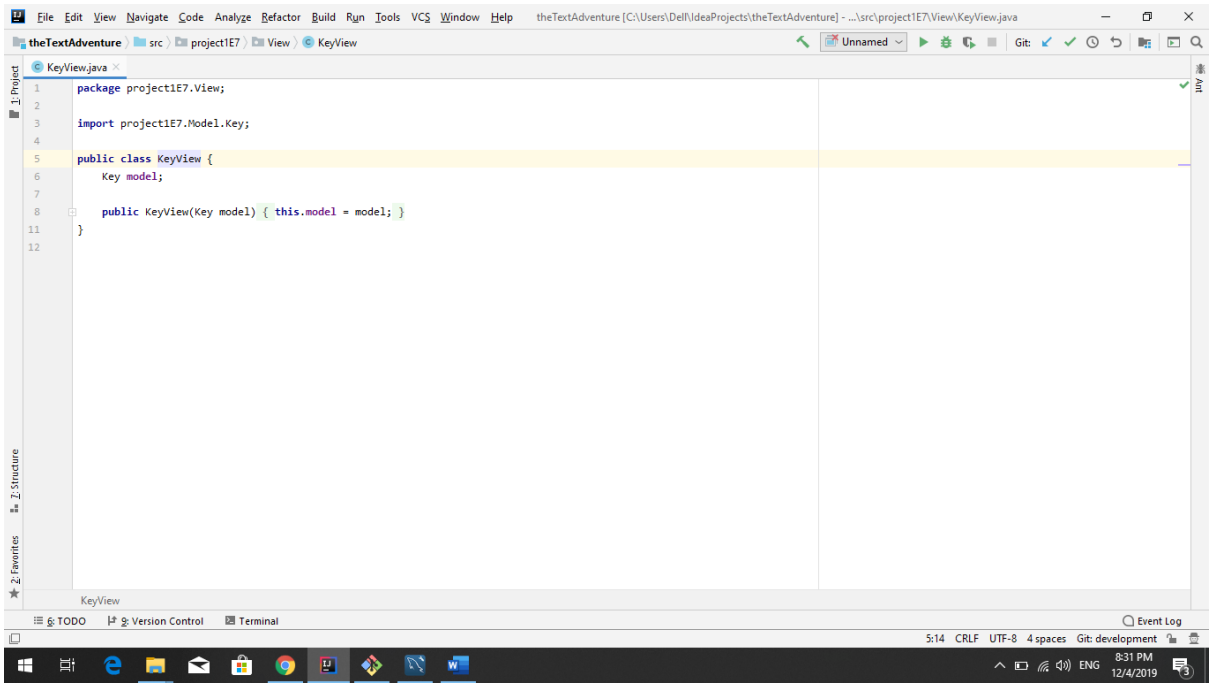
### 3.2.5 DoorView Class



```
package project1E7.View;

import project1E7.Model.Door;

public class DoorView {
    Door model;

    public DoorView(Door model) { this.model = model; }
}
```

### 3.2.6 KeyView Class

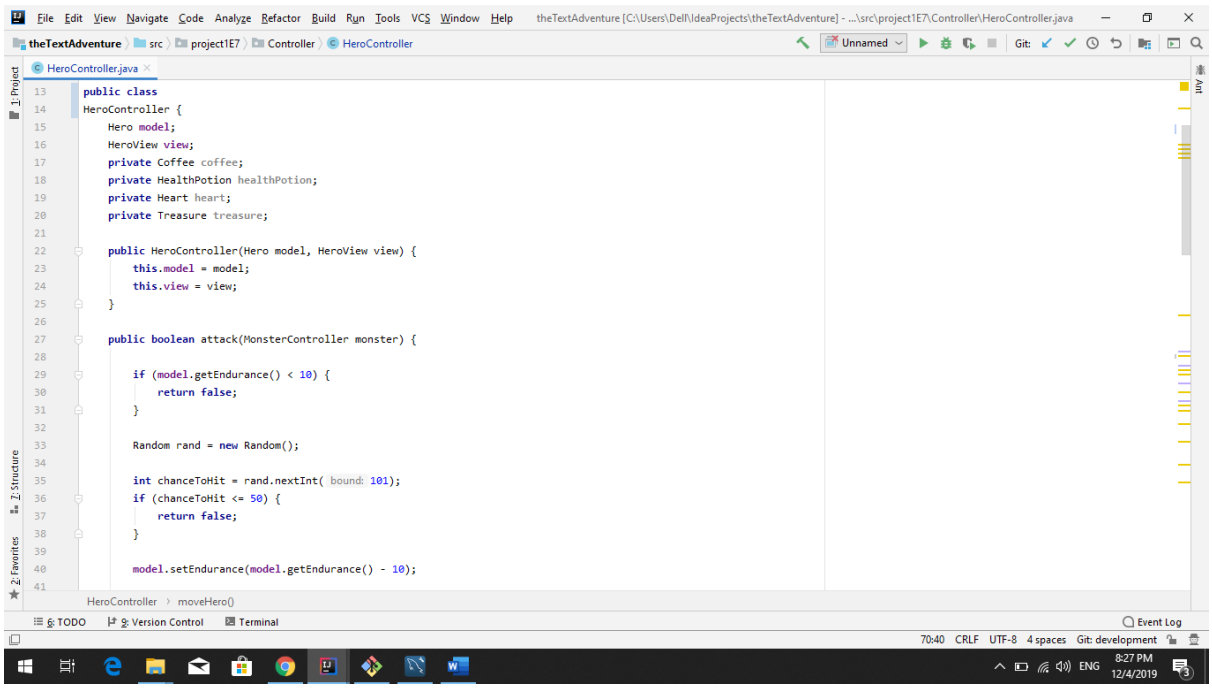
This class may be leaving.



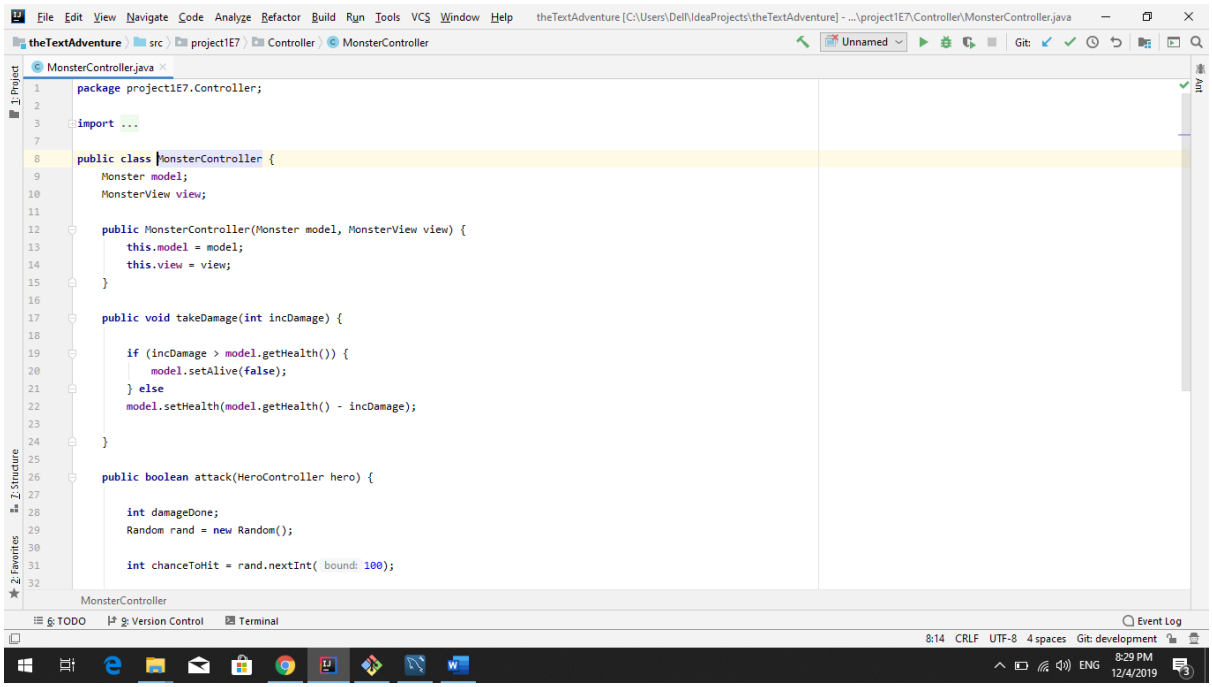

## 3.3 Controller Package

The controller package holds the logic operations of the game to the respective classes the controller takes the model and the view as a parameter when created, the controller carries out the the methods and logic of the model.

### 3.3.1 HeroController Class

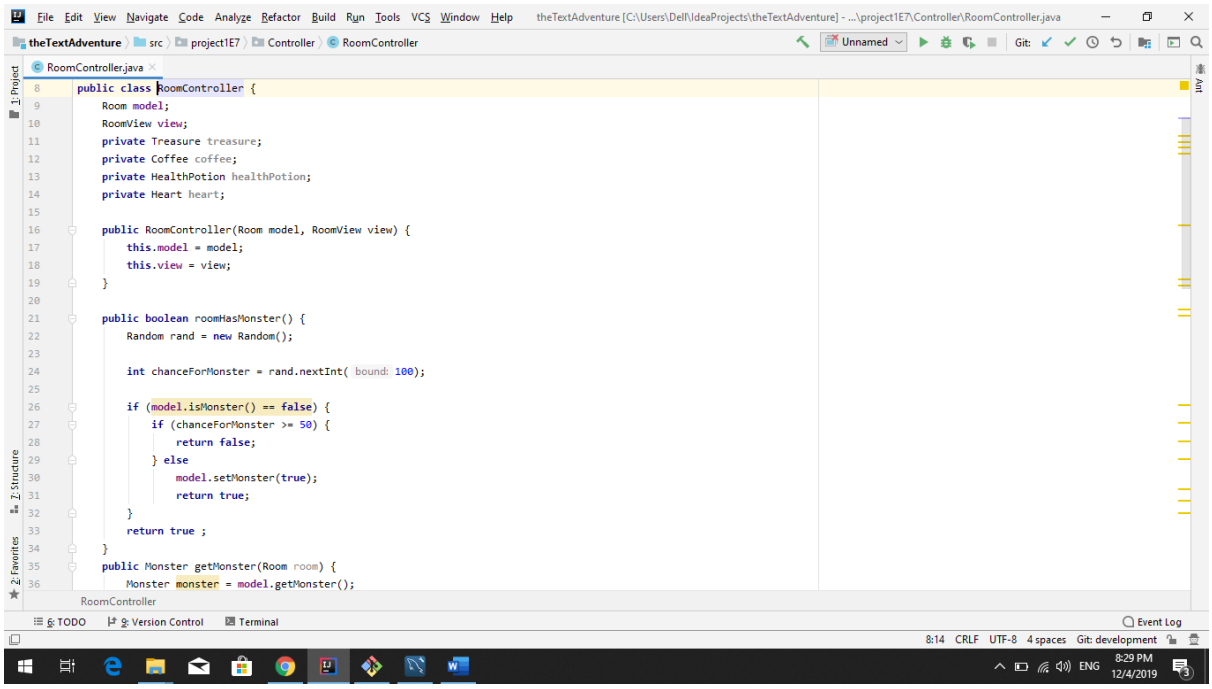


### 3.3.2 MonsterController Class



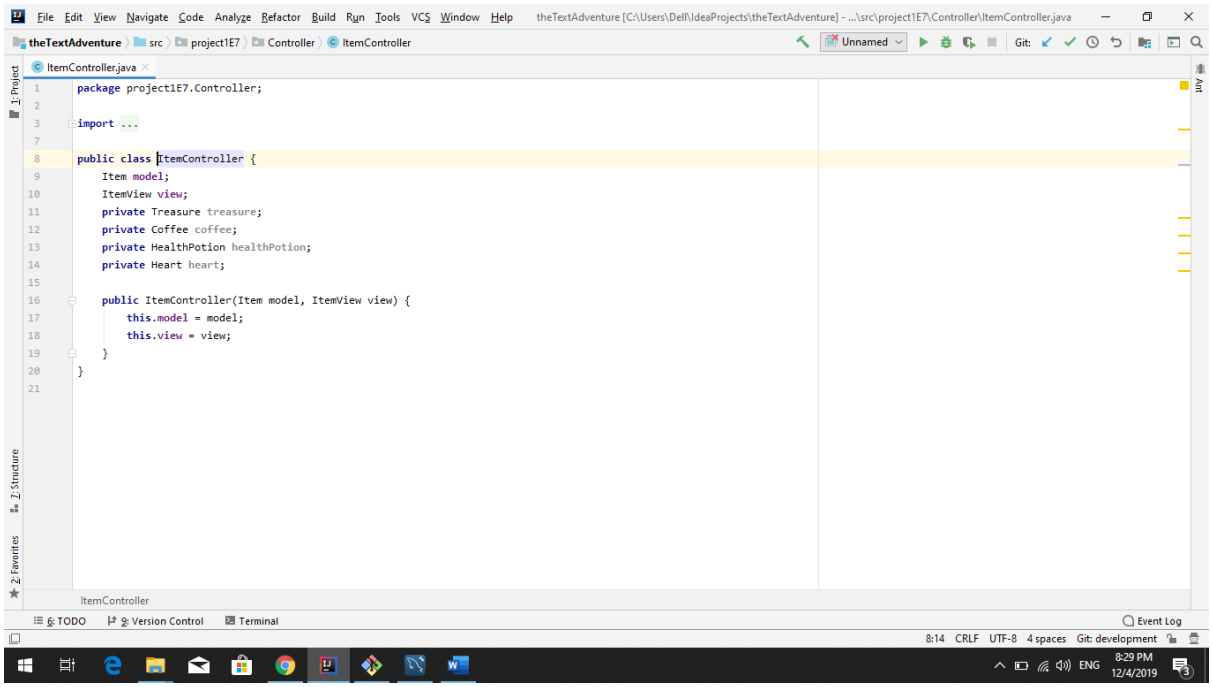
```
package project1E7.Controller;

import ...

public class MonsterController {
    Monster model;
    MonsterView view;

    public MonsterController(Monster model, MonsterView view) {
        this.model = model;
        this.view = view;
    }

    public void takeDamage(int incDamage) {

        if (incDamage > model.getHealth()) {
            model.setAlive(false);
        } else
        model.setHealth(model.getHealth() - incDamage);

    }

    public boolean attack(HeroController hero) {

        int damageDone;
        Random rand = new Random();

        int chanceToHit = rand.nextInt( bound: 100);
```

### 3.3.3 RoomController Class

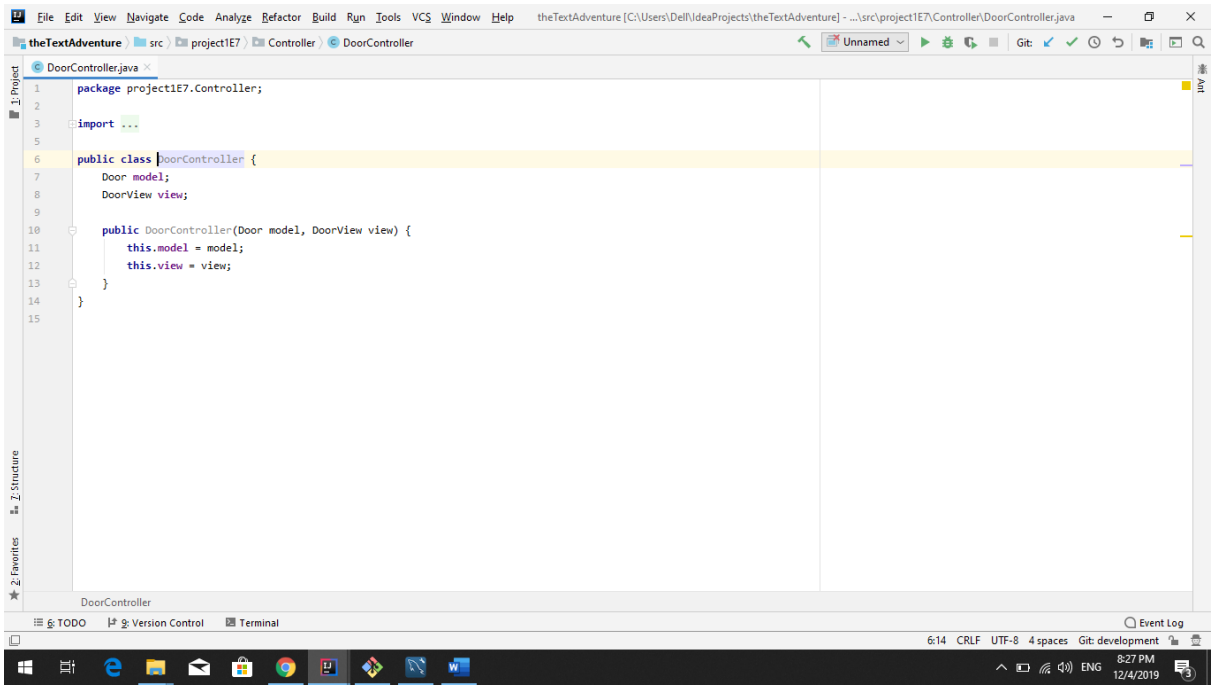


```
public class RoomController {
    Room model;
    RoomView view;
    private Treasure treasure;
    private Coffee coffee;
    private HealthPotion healthPotion;
    private Heart heart;

    public RoomController(Room model, RoomView view) {
        this.model = model;
        this.view = view;
    }

    public boolean roomHasMonster() {
        Random rand = new Random();

        int chanceForMonster = rand.nextInt( bound: 100);

        if (model.isMonster() == false) {
            if (chanceForMonster >= 50) {
                return false;
            } else
                model.setMonster(true);
                return true;
        }
        return true ;
    }
    public Monster getMonster(Room room) {
        Monster monster = model.getMonster();
```

### 3.3.4 ItemController Class

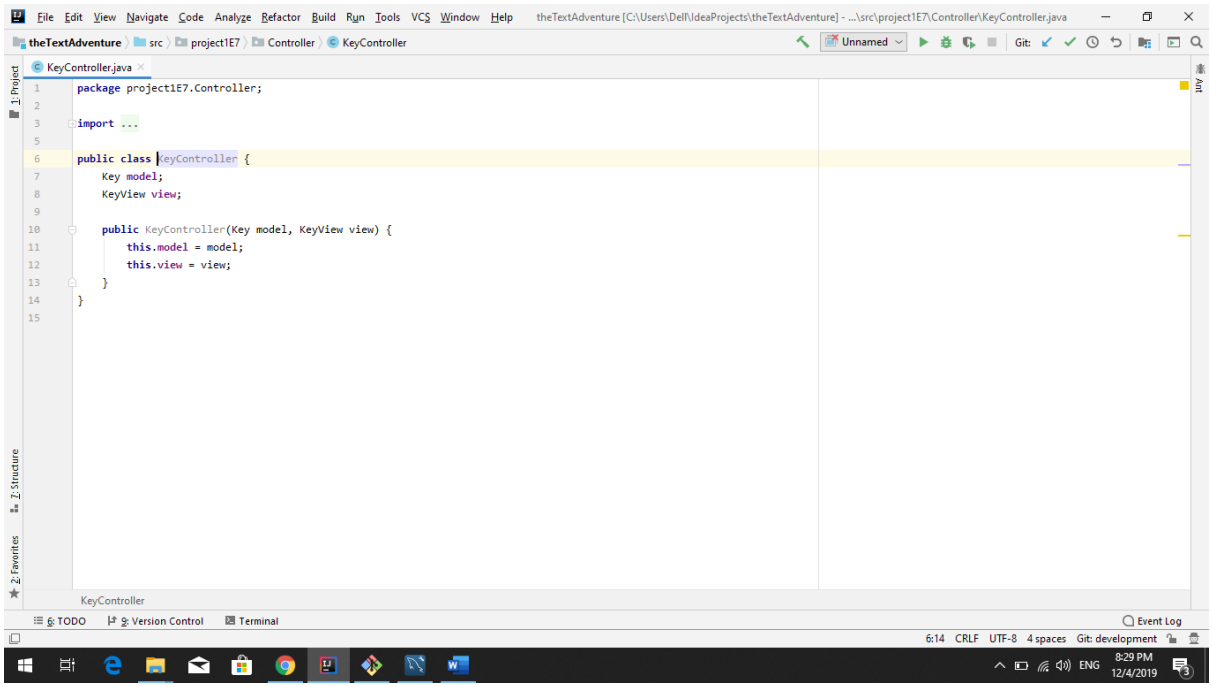



### 3.3.5 DoorController Class



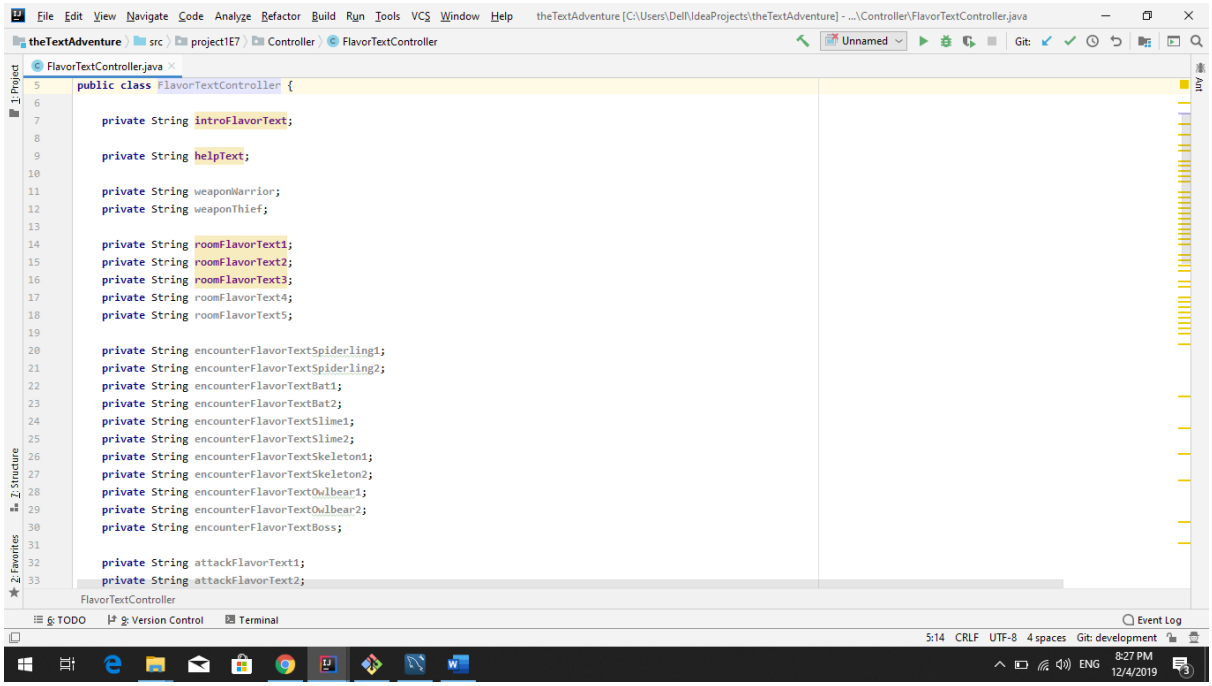
### 3.3.6 KeyController Class

May be leaving this class behind.



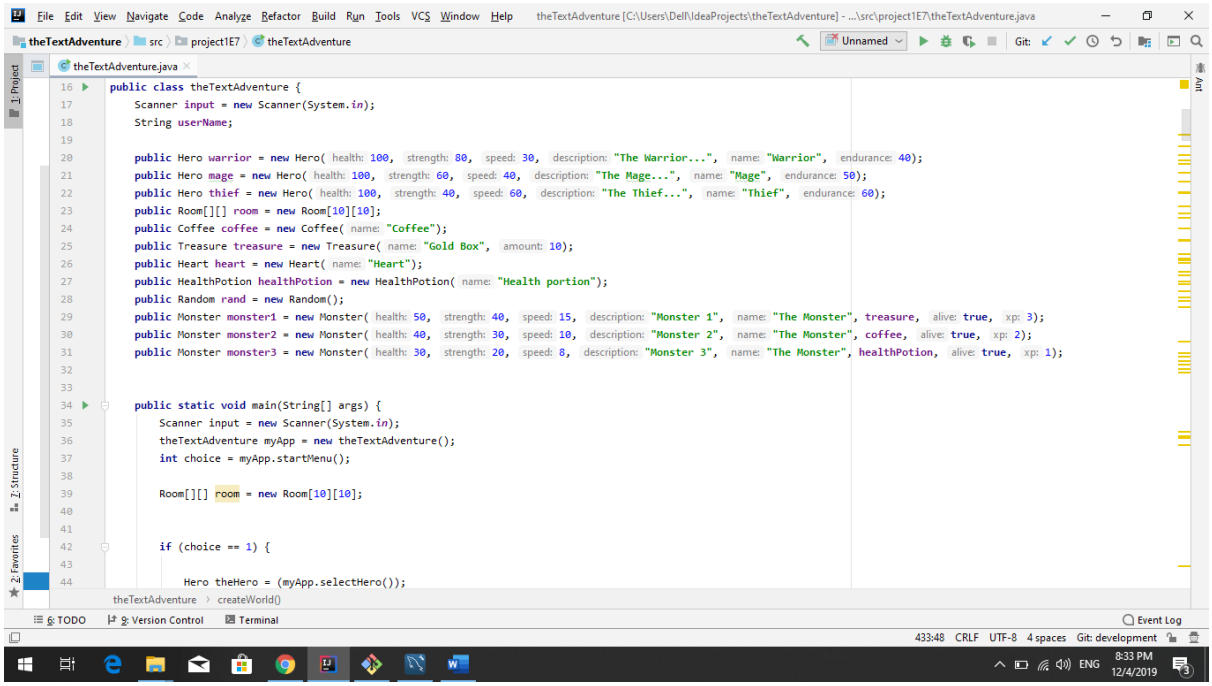

### 3.3.6 FlavorTextController Class

This is a temporary class and will be leaving to the respective view classes.



## 3.4 The Text Adventure Main Class

The text adventure class is the main it contains some methods that may end up in new classes we create down the line but runs the main game.



# 4. Test Results

Table 2 below contains the current status of implemented and tested requirements.

**Table 2 - Test Results**

| Req. No | Req. Name | Test Result |
|---|---|---|
| 1 | MVC Architecture | PASSED |
| 2 | Start Menu | PASSED |
| 3 | Normal game menu | NOT IMPLEMENTED |
| 4 | Implement character class | PASSED |
| 5 | Implement hero subclass | PASSED |
| 6 | Create initial story elements | PASSED |
| 7 | Design Map Outside Java | PASSED |
| 8 | Fill Room Class | NOT IMPLEMENTED |
| 9 | Implement Designed Map | FAILED |
| 10 | Create initial story elements | PASSED |
| 11 | Add Monsters | NOT IMPLEMENTED |
| 12 | Map navigation | NOT IMPLEMENTED |
| 13 | Encounter Menu | NOT IMPLEMENTED |
| 14 | Printable map screen | NOT IMPLEMENTED |
| 15 | Death Menu | NOT IMPLEMENTED |
| 16 | Treasure Score | NOT IMPLEMENTED |
| 17 | High Score | NOT IMPLEMENTED |
| 18 | Additional Flavor Text | NOT IMPLEMENTED |
| 19 | Save game function | NOT IMPLEMENTED |
| 20 | Implement Item Functions | NOT IMPLEMENTED |
| 21 | Key and Doors | NOT IMPLEMENTED |
| 22 | Level System | NOT IMPLEMENTED |
| 23 | Boss Fight | NOT IMPLEMENTED |
| 24 | End Screen Menu Win | NOT IMPLEMENTED |
| 25 | Difficulty System | NOT IMPLEMENTED |

# 5. Summary and Conclusion

The text adventure game is coming along it is our belief we have finished all the classes we need for the game to operate as far as we know now. The start menu has been implemented currently you can only start a game and exit as we have not finished the view high score and load game functions. The choose hero menu has been implemented with a star chart of the heroes stats, you can also choose your hero. The world has been created using a object multidimensional array for Room we have filled the rooms with items and monsters where we desire them, we also have been developing the functionality for random item and monster drops in rooms. That feature is not yet tested we are still trying to figure out how to move around the map.

## 5.1 Weekly Progress

Below is a short summary of what was done each week.

### 5.1.1 Week 1

During this week we spent several hours planning the structure of our text based game. This included deciding which features were to be implemented and which were to be cut. This included deciding how to structure the classes and methods. There is still work to do on some aspects of structure and decision such as how we will implement the classes within the game and what abilities each will have as well as how they will be used through methods, classes, or some specific condition. This is shown on our requirements list as the least prioritized as it was decided to be worked on later when we choose the different abilities for each character. Our biggest challenge was talking through exactly which features of the game we were comfortable in completing in the alloted time and which ones we decided not to. Luckily being the optimistic group we are most features are staying in the plan right now. We learned it is a time consuming process to plan a game, even as simple as this one. After this week we are feeling much more confident and on the same page now.

### 5.1.2 Week 2

Most of time has been spent on figuring out git as it was a new experience for a majority of the group and issues popped up frequently. These issues have continuously have costed hours of valuable programming time our solution will be creating a branch for each of us after we merge to the master and merging through out the week when we meet up to discuss progress and talk game plan. As a group we decided we wanted to take the next step in learning about development architecture so we have converted our text adventure game to a MVC architecture. We have had some trial and error figuring out exactly how we go about implementing but I believe we are starting to get the hang of it. We have implemented the Start game menu which only is capable of quiting and starting a game. We have also implemented the choosing of the hero menu where we also print out a bar chart made of stars as a visual representation of the heroes skills. Neither of these features are yet to be break proof yet, we need to learn more about try catch or other features that can combat the wrong variable type. We have also implemented the starting story description but the place at which it is located is subject to change. Due to set backs with git we were not able to get the map out in a timely manner and have just gotten within the past 2 days and needs some changes as to how it builds the map. We have started on developing the fighting logic but have yet to be able to test as we are still learning the MVC figuring out how to get a specific monster from a list of monsters or even have the room return the monster has proven difficult. Although we are confident these things can be worked out and we believe we are headed in the right direction.

### 5.1.3 Week 3

TODO.

#### 5.1.4 Week 4
TODO.

#### 5.1.5 Week 5
TODO.

#### 5.1.6 Week 6
TODO.

#### 5.1.7 Week 7
TODO.

## 5.2 Difficulties and challenges
Below is a list of notable challenges that came up during this project and that took a long time to solve.

#### 5.2.1 Getting Started with GIT Difficulty 1
Due to the inexperience with git we have run into numerous issues and difficulties costing us hours of potential coding time. Our solution is to make separate branches for each member of the group and merge when we meet during the week.

#### 5.2.2 Hero Navigation Between Rooms Difficulty 2
We had thought we solved it and moved on but turns out not to be the case. We have the basic idea as to how it must be done but solving has given us the most trouble in how we can give it the necessary functionality, without having to write long code. We decided since it is close to deadline to prioritize that above all for the next week and moved on fine tuning our current code.

#### 5.2.3 Learning MVC architecture 3
Although it's not required we want to implement this architecture design to our game. It has probably set us back a bit but we believe we are finally getting the hang of it and are sure we can make it work for the better of our project. It is also a very important lesson to be learned.

## 5.3 Correctness of time estimates
We did not have the correct requirements during our first meeting and also didn't have the correct times  on the last report so although I'd like to say we were accurate I can't say for sure but I believe we would probably be ahead of schedule had we not had issues with our git setup. So we have a solution in mind for after we merge to the master.

## 5.4 Priority decisions

Our priorities may have been  a little out of order but our primary problem was trying to do to much quickly we should have developed slowly making improvements 1 step at a time instead of trying to do everything at once. We should have attempted to get a minimum viable product out rather than what we have, but there is hope for us yet we can definitely improve on what we have and produce a high quality text adventure game.

## 5.5 Conclusion

<Instructions: Look back on the whole project. Here you can write a bit more freely about your thoughts on this project. What was your overall experience? How was the teamwork? What did you learn? Can you list some points that you will do better in next project? Other thoughts. >

# 6 References