**Final Project**

**Object Oriented Programming**

Project Name: “Text Based RPG”

Student Name: Abdullah Akmal Sutoyo

Student ID: 2602239320

Class: L2BC

# **Table of Contents**

**Cover Page**

**Contents**

**A. Description**

I. Introduction

II. The function of the program

**B. Design**

I. Parts of the program and requirements

II. Function of each part of the program

**C. Implementation**

I. UML Diagram

II. Extensibility

III. Explanation of the methods made and used

**D. Lessons learned**

I. Learning about the concepts of OOP

II. Error handling and debugging

**E. Evaluation**

I. Does the program work properly

II. Future improvements that can be done

III. Reflection

**F. Evidence of a working program**

# **A. Description**

## **I. Introduction**

After it was announced there was a final project for Object Oriented Programming, I started browsing the internet for simple/intermediate Java projects that uses object-oriented programming principles. I specifically looked for a project that suits my interests and level of understanding in the Java language, and I ended up finding a project that was the perfect fit for me. The project was a text-based RPG game using Swing as the GUI.

<https://www.youtube.com/playlist?list=PL_QPQmz5C6WUMB0xEMZosWbyQo_Kil0Fb>

I used the tutorial above to help me with the basic structure of the code, and then added more things so that the user can have a more immersive experience. I used the IntelliJ IDE for this project and I will be posting all the code along with this report on my repository which can be accessed through my GitHub account here:

<https://github.com/ilovegaming42069/OOPFinalProject>

## **II. The function of this program**

The function that this program serves is that of an RPG game that can provide entertainment for the player. The game is played by reading the story provided by the game, and then the player will choose an action based on the choices given. The objective of the game is to just enjoy the story and world of the game. While this game does not have any practical uses in real life, it is a good time killer if you have free time or when you are bored.

# **B. Design**

## **I. Parts of the program and requirements**

Requirements:

To run the program, you need to download Java and the SDK on your device. And then you can run it on an IDE or any application that can properly run a Java file.

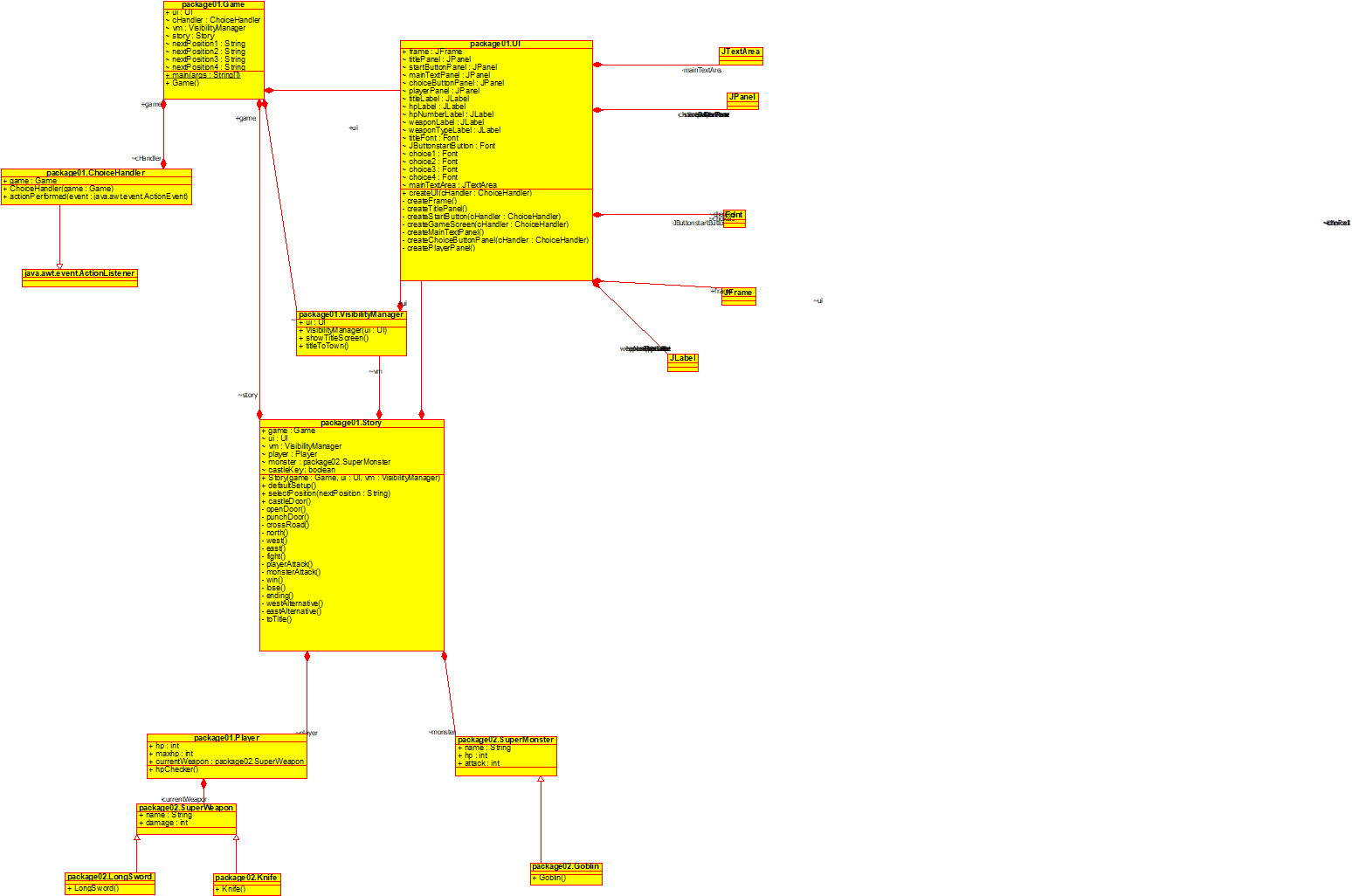
## **II. Function of each part of the program**

In this program I used a window with the size of 800x600. The program starts out with the starting screen which shows the name of the game and one button input to start the game. If the user clicks on the start button, the game will load, and it will immediately start the game.

Now that the user has started the game, the user can now read the story prompt provided by the game. The user can also see four choices in the form of button inputs, and each choice will change the story prompt. The user can also see a health bar, which has a default value of 10. The health can decrease or increase, depending on how the story progresses. If your health bar reaches 0, the prompt will inform you that the user has lost the game and a button input will appear. If you press that button, it will restart the game. The maximum health for the user is 15. The user can also see the weapon that is currently equipped. Each weapon that can be found in the game all have different statistics. If the user founds a new weapon, which also depends on how the story progresses, the user will also see how the weapon text changes to what is currently equipped. In this game, there are enemies to fight. Each enemy that can be fought in the game also have different statistics. If the user defeats an enemy, they might get an item. There are also certain in game events that might increase/decrease your health. If the user has reached a certain part of the story, the game will inform you that the story is to be continued.

# **C. Implementation**

## **I. UML Diagram**



## **II. Extensibility**

1. Comment lines are used to further explain the functions of each function and method and the purpose of each of them to help other programmers.
2. Variables use meaningful identifiers to help other coders to understand easier.
3. Methods are used to make the code easier to understand.
4. UML Diagram to make it easier for others to see how the relationship of the classes in this project.

## **III. Explanation of all the methods made and used**

### *Game.java*

This file contains a public class called Game, and this class has a composition relationship with 4 other classes (UI, ChoiceHandler, VisibilityManager, and Story). This class is the main class of this project, and there are only two methods in this class.

The first method is the constructor to this class, in which it calls the methods that will run the main functions of the game on the other classes like UI, Story and VisibilityManager.

The second method is the main function of the entire project, in which it calls the constructor written above.

### *ChoiceHandler.java*

This file contains a public class called ChoiceHandler, which implements from an inheritance from the Java library called ActionListener. This class has a composition relationship with the class Game. This class handles all the choices of the game from the methods inherited from ActionListener. There are only two methods in this class.

The first method is the constructor to this class, in which it takes the class Game as a parameter, and then initializes it.

The second method is called actionPerformed, which is a method inherited from ActionListener. This method is the main function for all 4 button inputs of the game to change the game scenes.

### *UI.java*

This file contains a public class called UI, and this class has a composition relationship with classes from the Java library such as JFrame, JPanel, JLabel, Font, JButton and JTextArea. From these classes, the UI class contains the methods to create a UI for the game. There are several methods in this class.

The first method is called createFrame, in which it creates the main frame of the game. The background color and the size of the screen is set in this method.

The second method is called createTitlePanel, in which it creates the panel for the title screen. The text for the title screen, the boundaries for the text of the title screen, and the color of the panel as well as the text is set in this method.

The third method is called createStartButton, in which it creates the button for the title screen. This method takes the ChoiceHandler class as a parameter, in which it is used to change the game screen from the start menu to the first scene of the game. The background, text color, and the boundaries are also set in this method.

The fourth method is called createMainTextPanel, in which it creates the panel for the game screen. The text for the game screen, the boundaries for the text of the game, and the color of the panel as well as the text is set in this method.

The fifth method is called createChoiceButtonPanel, in which it creates the buttons for the game screen. This method takes the ChoiceHandler class as a parameter, in which it is used to change the game screen from a scene another scene, depending on the player’s choices. The background, text color, and the boundaries are also set in this method.

The sixth method is called createPlayerPanel, in which it creates the panel for the stats of the players, such as the health bar and the weapon. The text for the statistics, the boundaries for the texts, and the color of the panel as well as the text is set in this method.

The seventh method is called createGameScreen, in which it creates the main game screen by calling the createMainTextPanel, createChoiceButtonPanel, and createPlayerPanel methods. It takes the ChoiceHandler class as a parameter, since the createChoiceButtonPanel needs the ChoiceHandler parameter.

The eight method is called CreateUI, in which it creates the entire UI of the game by calling the createFrame, createTitlePanel, createStartButton, and createGameScreen methods. It takes the ChoiceHandler class as a parameter, since the createStartButton and createGameScreen needs the ChoiceHandler parameter.

### *VisibilityManager.java*

This file contains a public class called VisibilityManager, and this class has a composition relationship with the UI class. This class handles whether the main menu or the game screen is visible in the UI. There are only three methods in this class.

The first method is the constructor to this class, in which it takes the class UI as a parameter, and then initializes it.

The second method is called showTitleScreen, in which it sets the visibility of the titlePanel and startButtonPanel created in the UI class to true, while setting the mainTextPanel, choiceButtonPanel, and the playerPanel’s visibility to false.

The third method is called titleToTown, in which it does the exact opposite of showTitleScreen.

### *Story.java*

This file contains a public class called Story, and this class has a composition relationship with 5 other classes (Game, UI, VisibilityManager, Story, SuperMonster). This class handles each game scene in the game, and there are several methods in this class.

The first method is the constructor of this class, and it takes the Game, UI, and VisibilityManager classes as parameters, and then initializing them.

The second method is called defaultSetup, and this method sets the statistics of the player to how it is supposed to be when you start the game(Player HP is set to 10, weapon equipped is set to knife, and the castle key Boolean is set to false).

There are several methods to determine the current scene of the game, such as castleDoor, openDoor, punchDoor, crossroad, north, west, east, fight, playerAttack, monsterAttack, win, lose, ending, westAlternative, eastAlternative, and toTitle. Each methods have different texts on the buttons and the one shown on the main panel, and all the game logic for each scene is also handled on these methods.

The final method is called selectPosition, and it takes a string called nextPosition as the parameter. Inside the method, there is a switch case, where it takes nextPosition’s string to choose which case to go to. nextPosition’s value depends on the game logic from the methods mentioned above.

### *Player.java*

This file contains a public class called Player, and this class has a composition relationship with the SuperWeapon class. This class contains the statistics of the player, such as the HP and the current weapon. There is only one method in this class, and it is called hpChecker. The method checks whether the health exceeds the maximum HP or the minimum HP, and it will set it so that the HP doesn’t go over the maximum or minimum HP. The maximum HP is set to 15, while the minimum HP is set to 0.

### *SuperWeapon.java*

This file contains a public class called SuperWeapon. This class contains the name of the weapon, as well as the amount of damage the weapon gives.

### *Knife.java*

This file contains a public class called Knife, and this class is an extension of the SuperWeapon class. This class has one method, in which it is the constructor of this class. In the constructor, it sets the name of the weapon to Knife, as well as the damage of the weapon to 3.

### *LongSword.java*

This file contains a public class called LongSword, and this class is an extension of the SuperWeapon class. This class has one method, in which it is the constructor of this class. In the constructor, it sets the name of the weapon to LongSword, as well as the damage of the weapon to 12.

### *SuperMonster.java*

This file contains a public class called SuperMonster. This class contains the name of the enemies in the game, as well as the statistics of the enemy, such as their health and the amount of damage the weapon gives.

### *Goblin.java*

This file contains a public class called Goblin, and this class is an extension of the SuperMonster class. This class has one method, in which it is the constructor of this class. In the constructor, it sets the name of the enemy to Goblin, the health of the enemy to 20, as well as the damage of the enemy to 6.

# **D. Lessons learned**

## **I. Learning about the concepts of OOP**

When Mr Jude started to teach us about the principles of Object-Oriented Programming, I find myself a bit confused at first. After all I suddenly got introduced to new concepts such as encapsulation, abstraction, inheritance, and polymorphism. But, after working at this project, I now understand when to implement these concepts on a program. I feel like I have become a more efficient coder, and I have a better understanding on how classes, packages, attributes, and methods work.

## **II. Error Handling and Debugging**

One of the most frustrating parts when becoming a coder is having to deal with debugging your own program. I feel the same way too, as having to solve a problem that you caused can be quite confusing. I have devised my own way to figure out what went wrong in my code to lead to something not going the way I want to. First, I will check in what line is the error in. Then I will try checking if I have a typo in my code. Maybe I accidentally missed a bracket, comma or a semicolon when writing it. If there is nothing wrong, then there is something wrong with the logic of the code and I will need to check what kind of error is the code having. Usually when running a problematic code, the terminal will give you a description of the error, and usually I will try to fix based on what the problem is based on what the terminal said. If I can’t fix it, I will just copy what the terminal said and paste it on Google to find out more on that type of error. If it is too complicated to fix, I will just try switching the logic of my code.

# **E. Evaluation**

## **I. Does the program work properly?**

This text-based RPG game is essentially just a code I copied from the internet, in which I added further improvements such as fixing the game loop and simplifying the code so that it is more readable. The program works well as a game and has basically no errors that can occur that hasn’t been dealt with. However, the game is unfinished as a story and has no end goal as of now, and I hope in the future I can find the time to further the story and complete the game. Overall, I would say that this program is good in terms that it reaches its goal as a program.

## **II. Future Improvements that can be done**

The improvements I would like to add to this game would be a difficulty system which allows users to choose the difficulty of the game (damage of each weapon and each monsters, as well as the increase/decrease of the health bar from in game events will be set according to the difficulty), a pause menu, a resizable screen, and background music. These improvements aren’t necessarily that hard, and I hope I can do in the future.

## **III. Reflection**

With this project I now know my limitations as a programmer, and I wish that I could improve on these limitations. I am satisfied with how my program ended up being, even though I know that there are many improvements that can still be done with the existing program. I will try to continue to work on this program after submitting this report and I will keep improving it so that it can be something that I am proud of. After all this is one of my first times implementing OOP concepts on Java projects and I will remember the lessons I learnt when making this all throughout my life.

# **F. Evidence of working program**

