|  |  |  |
| --- | --- | --- |
| Design Patterns | 13th of March  13th of March | |
| Monica Stoica  Rosen Danev  Gaming character | | Decorator pattern |

Table of Contents

[Introduction 2](#_Toc445638597)

[UML Diagram 2](#_Toc445638598)

[User interface 3](#_Toc445638599)

[Reusability 4](#_Toc445638600)

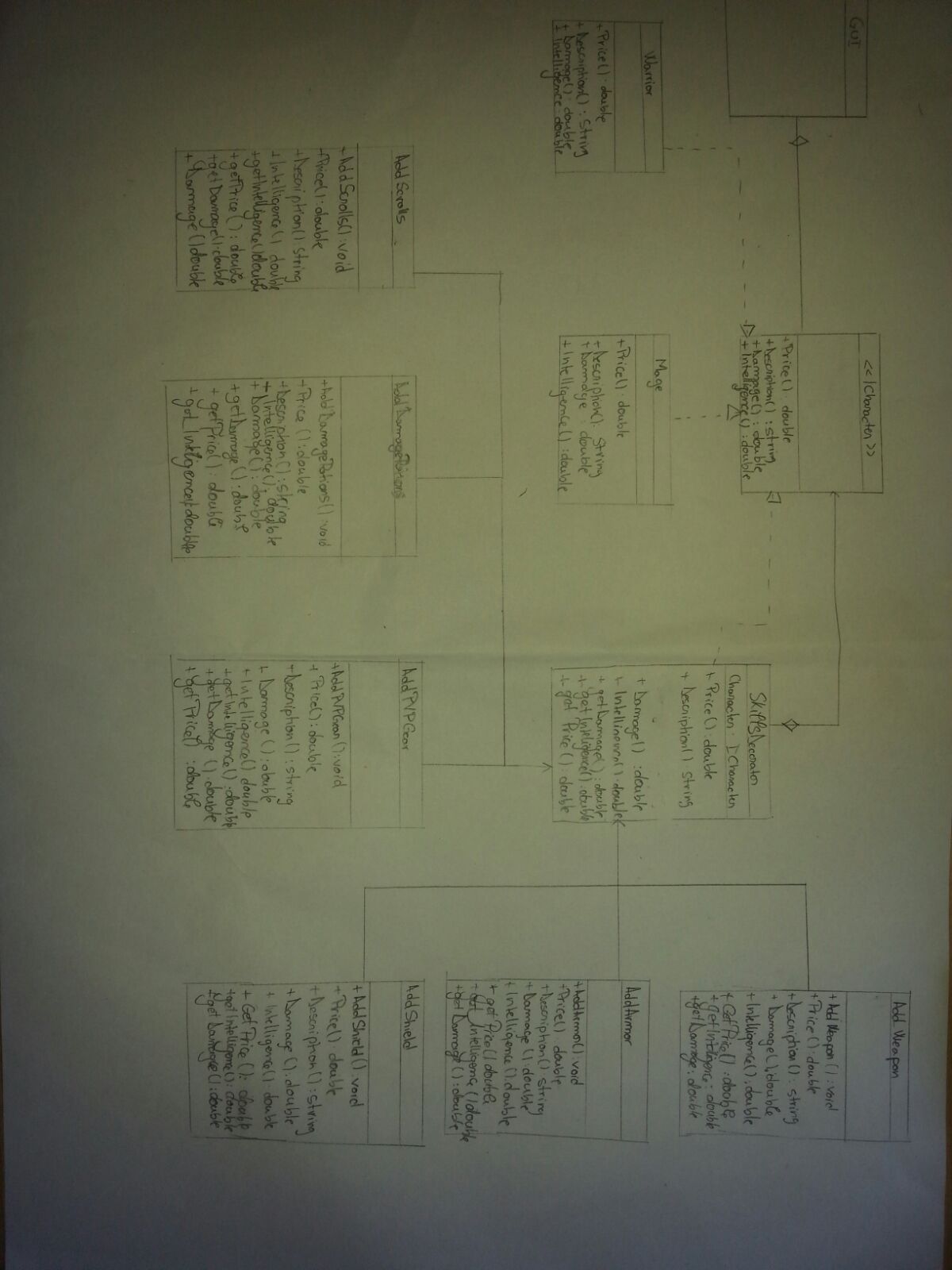
[Maintainability 4](#_Toc445638601)

[Extensibility 4](#_Toc445638602)

# Introduction

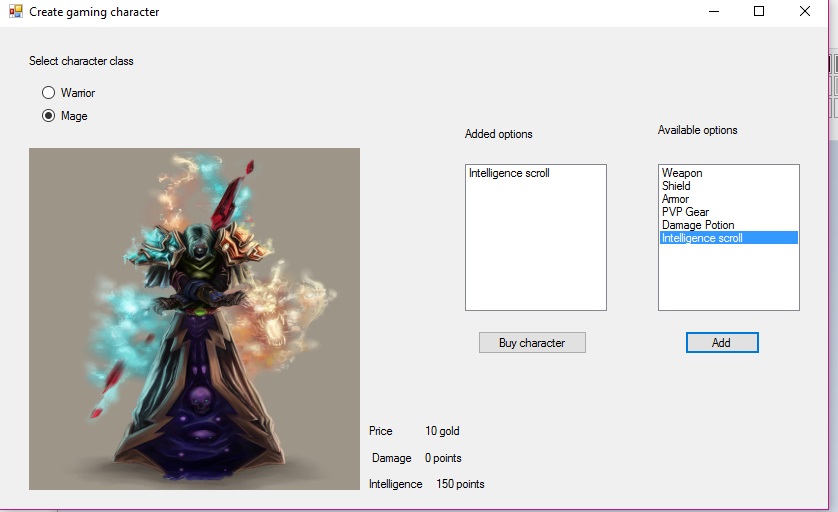
The following document analyses the three main characteristics of the pattern used to develop an application allows the user to create a gaming character. This character can be customized by adding available options that will increase the damage and the intelligence points of the character but also the price. A pattern is general reusable solution to a commonly occurring problem within a giving context. The purpose of using patterns is to speed up the developing process and helps preventing issues that can cause major problems. The purpose of the decorator pattern is to add responsibilities to an object dynamically without changing the user interface. Therefore, the functionality can be extended easily using inheritance.

# UML Diagram

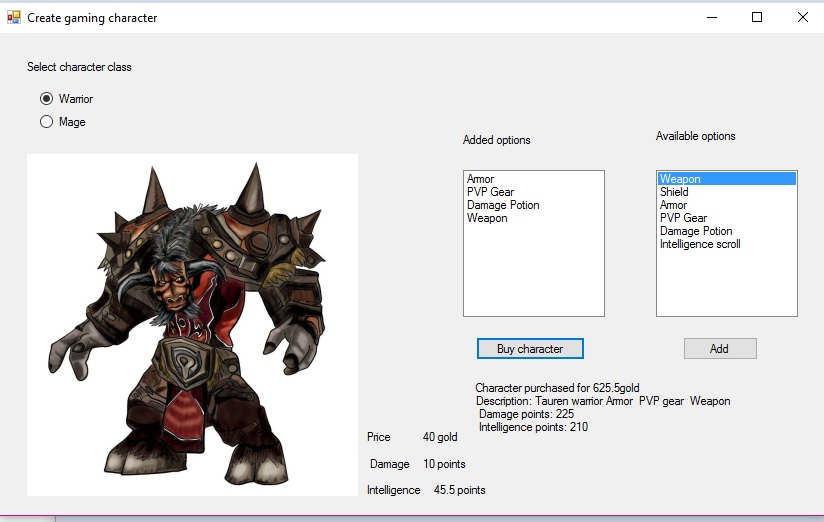


# User interface

The following application allows the use to create and personalize his/hers gaming character. There are two options available: Warrior and Mage. Each one of them has a starting price, damage and intelligence. The user can personalize the character by adding weapons, shields intelligence scroll, etc. When the user adds an option, the specifications will be displayed on the right side of the picture.



If the user decides to buy the character, the program will show the current characteristics of the created character.



# Reusability

One of the main reasons why patterns are useful is because they can be easily reused without changing the code. Considering our above described application, we can attest that the pattern is reusable. The observer pattern can be reused by creating different characters and adding new skills

# Maintainability

Because the classes are not tightly coupled and the pattern is quite small, the system is easily maintained. When a system is easy to maintain it means that new features can be added. However, in this case, if the software grows too much, maintaining the code may take a lot of time.

# Extensibility

New functionality can be provided by adding new code without changing the initial one. Therefore, there is no need to worry about bugs or causing problems. The current pattern can be easily extended because but the original code cannot be changed. There can be added as many skills (in this case) as possible and the characters can differ in their behavior.