Project Fierce Battle RPG

**Team**: Alexandre Dumas

* Denis Kyashif
* Tsveta Brakalova
* Kristian Rizov
* Kaloyan Peykov
* Martin Petrov – didn’t participate
* Stavri Zoichkin – didn’t participate
* Julian Lazarov – didn’t participate

**Project URL:** [*https://github.com/stinger907/OOPTeamwork*](https://github.com/stinger907/OOPTeamwork)

**Goals**: The goal of this project was to practice the principles of Object Oriented Programming by creating a text based RPG game. We tried to uphold the best practices in OOP and satisfy the project requirements.

**Description**: This is a simple game where characters can be created, equipped with various items and put into battle. The game uses simple text commands that should be entered into the console.

The game meets the following requirements:

* At least **5 interfaces** (with one or more implementations) – ***IBattler, IEquipper, IEvader, INameable, IPriceable***
* At least **15 classes** (implementing the application logic) – the project has three namespaces – respectively ***Characters*** where the characters and their abstract class are stored, ***Interfaces*** where the interfaces lie, and ***Items***, where the items and their abstract class are located. There are also the ***Engine*** class which is responsible for the entire game logic and the ***GameUI*** class which is the executable for this game.
* At least **3 abstract class** (with inheritors) – there are five abstract classes – ***Character, Weapon, MeleeWeapon, MagicalWeapon, RangedWeapon***
* At least **1 exception class** (with usage in your code) - ***InvalidRangeException***
* At least **3 levels of depth in inheritance**
* At least **1 polymorphism** usage – Hero class and its derivatives
* At least **1 structure - *Location***
* At least **1 enumeration – *LocationType***
* At least **1 event** (with subscribers) pops out a message on finished battle and based on the input processes a specific function
* At least **1 design pattern** (e.g. Composite, Singleton, Factory, Wrapper, Bridge, Command, Iterator, …) – ItemShop.cs implemented as Singleton

The code implementation of the game consists of:

Game Implementation

class GameUI – here is where the “Main” method of the game resides. It fires up an instance of the game engine.

class Engine – this is the class where the game logic resides. It consists of several methods. The engine receives a text command and processes it. The game characters can enter a battle, equip or dequip items, revitalize. A game location can also be specifically set, when not it remains neutral e.g. no race gets a specific location bonus. There is a help command that invokes the help method which prints the command manual.

class InvalidRangeException – inherits ApplicationException. It contains properties for minimum and maximum value and also a string constant for default message that can be overridden in the constructors. It is used in the encapsulation of items’ fields.

enum LocationType – an enumeration of all location types.

struct Location – is designed to hold a location with specific name and type. It may not come handy in the current version of the game, but it tolerates further gameplay expansion.

Characters Library

class Character – It implements the INameable interface and is the base class for the main Hero class.

class Hero – this is the main abstract class for the hero types. It inherits Character and also implements IEquipper and IBattler interfaces. Each hero has health points, agility, evasion ratio and gold that can either increase or decrease. The hero class has three constants for starting gold, damage and defense. Every hero has a native location type. Depending on the native location a hero can receive additional bonuses to his agility or health points. Each hero has an inventory. This is a list of items he possesses. The hero can buy, equip and dequip the items. Each hero can battle only one hero enemy at a time. Hero’s Attack method calculates the damage a hero can cause depending on his items, their given bonuses and on the enemy’s characteristics. The hero can also evade an attack. The Hero class spawns a few child classes – Elf, Orc, Undead, and Human, with customized health points, agility and native location type.

Items Library

class Item – each character can possess various items which can improve their attacking and defending capabilities. Each item should have a price which can vary between 0 and 1000. If a price doesn’t meet these requirements an InvalidPriceException is thrown. The items implement two interfaces – INameable and IPriceable. The items divide into 3 main subclasses – Armour, Charm and Weapon. The armor and charm implement IEvader. The Weapon class is a parent for MagicalWeapon, MeleeWeapon and RangedWeapon which also have subclasses – the various types of weapons that can be used. Each weapon differs in its price, damage, attack speed. Specific weapon types differ by their bonuses e.g. only melee weapons give chance of causing a double damage, on the other hand ranged weapons have critical strike ratio and magical weapons can cause direct damage.

class ItemShop – this class has a single instance and therefore implements the Singleton design pattern. It has a set of previously created items that can be bought by the hero (added to the hero’s inventory). The inner method DisplayItems() has been specifically created to suit the gameplay’s console interface.

Interface Library

interface IEvader – requires an implementation of an integer property named EvasionRate

interface IBattler – it assures that whatever implements it must possess integer properties for HealthPoint and Agility and also an Attack() method

interface IEquipper – it assures that whatever implements it must have an Inventory(a collection of items), integer property Gold and Equip() and Dequip() functions for adding and removing items from the inventory

interface INameable – whatever implements it must contain a string property Name

interface IPriceable - whatever implements it must contain an integer property Price



