Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение   
высшего образования

**Иркутский национальный исследовательский технический университет**

|  |
| --- |
| Институт информационных технологий и анализа данных |
| наименование института |
| Кафедра вычислительной техники |
| наименование кафедры |

|  |
| --- |
| **Отчет** |
| по лабораторной работе №4 по дисциплине «Разработка классов с использованием механизмов наследования, полиморфизма и инкапсуляции» |
| наименование темы  Вариант №20 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил студент |  | АСУб-19-1 |  |  |  | Даниленко А.С. |
|  |  | шифр |  | подпись |  | И.О. Фамилия |
| Проверил |  |  |  |  |  | Маланова Т.В. |
|  |  |  |  | подпись |  | И.О. Фамилия |
| Работа защищена с оценкой | | | |  | | |

Иркутск 2020 г.

**Содержание**

[1 Постановка задачи 3](#_Toc55424417)

[2 Описание структуры пользовательского меню 4](#_Toc55424418)

[3 Проектирование классов 5](#_Toc55424419)

[4 Описание методов классов и спецификация локальных переменных 6](#_Toc55424420)

[5 Таблица тестов 9](#_Toc55424421)

[6 Результаты тестирования 10](#_Toc55424422)

[7 Исходный код 12](#_Toc55424423)

[Список использованных источников 22](#_Toc55424424)

1 Постановка задачи

В соответствии с индивидуальным заданием описать иерархию классов, для каждого класса описать поля и соответствующие методы доступа к ним. В зависимости от задания некоторые из этих классов(как минимум один) являются абстрактными и служат для выделения общих данных и поведения для других классов. Абстрактный класс должен содержать как минимум один абстрактный метод, реализация которого у его наследников должна различаться. Помимо этого в общую часть задания входит разработка класса группирующего объекты описанных в соответствии с заданием классов. Для первого варианта индивидуального задания это может быть класс «кафедра». Для выполнения задания необходимо создать некоторое количество объектов, добавить их в группу используя предусмотренные методы класса-«группы» и для каждого из них вызвать унаследованный метод.

2 Описание структуры пользовательского меню

Меню предоставляет доступ к 5 действиям и завершению работы.  
1. Просмотр набора карт, с которыми начинается игра.  
2. Убрать все начальные карты.  
3. Добавление в начальный набор карты из общего списка.  
4. Создание новой карты.  
5. Переход в режим игры.

Пункты 3, 4 и 5 также представлены в виде меню, соответствующих задачам. Также класс Main хранит некоторые базовые карты, указатели на их списки а также общие внутриигровые параметры вроде текущей и максимальной маны игрока.

3 Проектирование классов

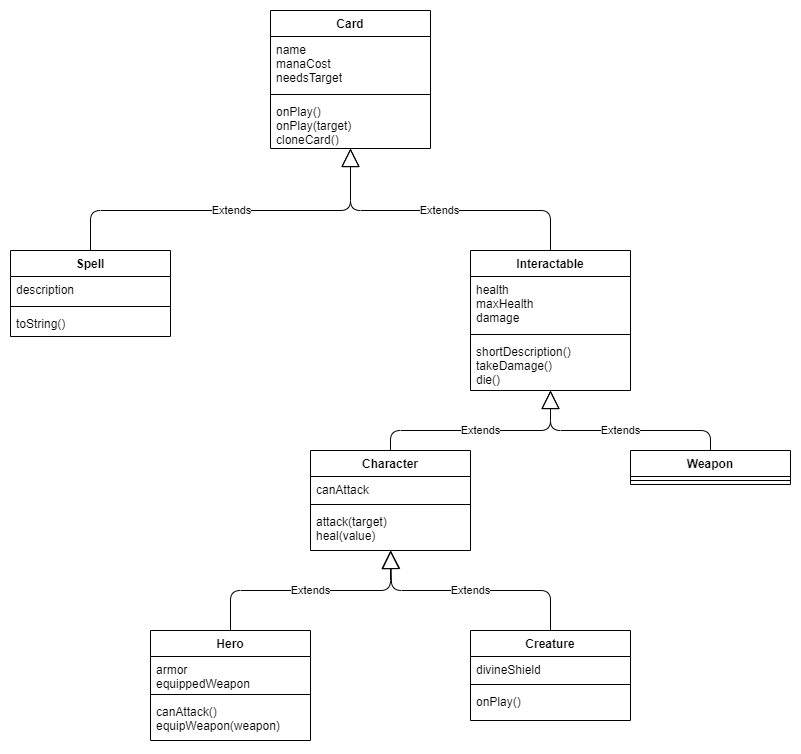


Рисунок 1 – Иерархия и структура классов-карт

Card – абстрактный класс, имеющий имя name, целочисленную стоймость в мане manaCost и указатель на необходимость наличия цели needsTarget. Имеет методы onPlay, различные в зависимости от того, нужна ли для разыгрывания карты цель, а также метод cloneCard, позволяющий вернуть копию карты и переопределяемый в наследниках.

Spell – особый вид карт, имеющих исключительно воздействие, но не сущность, а потому единственным дополнительным свойством является текстовое описание и переопределение метода toString(), позволяющего узнать подробности карты во время игрового процесса. Также этот метод переопределён в большинстве других наследников класса Card.

Interactable – общий класс для всех карт, имеющих сущность, т.е. здоровье и урон. Для всех таких карт существует shortDescription, позволяющий увидеть достаточное, но краткое описание характеристик сущности.

Weapon – особый класс карт-сущностей, единственной ключевой особенностью которого является активное использование сущностями класса Hero. Розыгрыш такой карты экипирует текущего героя этим оружием посредством вызова в нём метода equipWeapon.

Character – общий класс карт, которые существуют на игровом поле как самостоятельные сущности. Имеют поле canAttack, определяющее, может ли сущность атаковать на этом ходу, превращается в истину в начале каждого хода пребывания на поле боя, а также в ложь при успешно совершённой атаке. Также обладает методом атаки attack и восстановления здоровья heal.

Creature – итоговый класс сущностей, обладающих дополнительным полем божественного щита, блокирующего первый полученный урон divineShield. Розыгрыш такой карты призывает её как существо на поле боя.

Hero – класс главной сущности, уничтожение которой определяет победу или поражение. И игрок, и его противник имеют героя. Герой помимо здоровья имеет броню armor, которая поглощает урон и уменьшается вплоть до разрушения наравне со здоровьем. Здоровье не убывает до тех пор, пока герой обладает бронёй. Также герой может носить оружие equippedWeapon и использовать его для усиления своих атак. Оружие разрушается после истощения запаса прочности. Метод canAttack определяет, может ли герой атаковать с учётом наличия оружия. Метод equipWeapon экипирует игрока заданным оружием.

Класс ExceptionHelper позволяет осуществлять ввод с обработкой исключений. Класс CardCollection хранит образцы всех карт и позволяет сохранять и загружать карты в/из текстового файла.

7 Исходный код

Класс ExceptionHelper

package Lab\_3;  
import java.util.Scanner;  
public class ExceptionHelper  
{  
 private static Scanner *input* = new Scanner(System.*in*);  
 public static String readString()  
 {  
 String temp = *input*.nextLine();  
 if (temp != "")  
 return temp;  
 else  
 {  
 System.*out*.print("Пустой ввод. Повторите ввод строки.\n");  
 return *readString*();  
 }  
 }  
 public static int readInt()  
 {  
 try  
 {  
 int temp = Integer.*parseInt*(*readString*());  
 return temp;  
 }  
 catch (Exception e)  
 {  
 System.*out*.print("Ошибка при обработке числа. Повторите ввод целого числа.\n");  
 return *readInt*();  
 }  
 }  
 public static double readDouble()  
 {  
 try  
 {  
 double temp = Double.*parseDouble*(*readString*());  
 return temp;  
 }  
 catch (Exception e)  
 {  
 System.*out*.print("Ошибка при обработке числа. Повторите ввод числа.\n");  
 return *readDouble*();  
 }  
 }  
 public static int[] readIntArray(int size)  
 {  
 int counter = 0;  
 String[] buffer = *readString*().split(" ");  
 for (int i = 0; i < buffer.length; i++)  
 if (buffer[i].length() >= 1)  
 {  
 buffer[counter] = buffer[i];  
 counter++;  
 }  
 if (counter == size)  
 {  
 try  
 {  
 int[] temp = new int[counter];  
 for (int i = 0; i < counter; i++)  
 temp[i] = Integer.*parseInt*(buffer[i]);  
 return temp;  
 }  
 catch (Exception e)  
 {  
 System.*out*.print("Ошибка при обработке значений. Повторите ввод с [" + size + "] целочисленными аргументами.\n");  
 return *readIntArray*(size);  
 }  
 }  
 else  
 {  
 System.*out*.print("Неверное количество значений. Повторите ввод с [" + size + "] целочисленными аргументами.\n");  
 return *readIntArray*(size);  
 }  
 }  
 public static double[] readDoubleArray(int size)  
 {  
 int counter = 0;  
 String[] buffer = *readString*().split(" ");  
 for (int i = 0; i < buffer.length; i++)  
 if (buffer[i].length() >= 1)  
 {  
 buffer[counter] = buffer[i];  
 counter++;  
 }  
 if (counter == size)  
 {  
 try  
 {  
 double[] temp = new double[counter];  
 for (int i = 0; i < counter; i++)  
 temp[i] = Double.*parseDouble*(buffer[i]);  
 return temp;  
 }  
 catch (Exception e)  
 {  
 System.*out*.print("Ошибка при обработке значений. Повторите ввод с [" + size + "] численными аргументами.\n");  
 return *readDoubleArray*(size);  
 }  
 }  
 else  
 {  
 System.*out*.print("Неверное количество значений. Повторите ввод с [" + size + "] численными аргументами.\n");  
 return *readDoubleArray*(size);  
 }  
 }  
}

Класс Main  
package Lab\_4;  
  
import java.util.ArrayList;  
  
public class Main  
{  
 private static CardCollection *collection*;  
 private static Creature *student* = new Creature("Вежливый студент", 2, 1, 8);  
 private static Creature *caredOne* = new Creature("Любимец преподавателей", 3, 4, 1, true);  
 private static Weapon *deadline* = new Weapon("Пылающий дедлайн", 4, 3, 2);  
 private static Hero *debtor* = new Hero("Должник", 0, 0, 10);  
 private static Hero *MinObrRF* = new Hero("Мин.Обр. РФ", 0, 0, 20);  
 private static ArrayList<Character> *characters* = new ArrayList<Character>();  
 public static ArrayList<Creature> *enemySide* = new ArrayList<Creature>();  
 public static ArrayList<Creature> *playerSide* = new ArrayList<Creature>();  
 private static ArrayList<Card> *playerHand* = new ArrayList<Card>();  
 private static ArrayList<Card> *basicPlayerHand* = new ArrayList<Card>();  
 private static int *playerHandSize*;  
 private static int *enemyCreaturesCount*;  
 private static int *playerCreaturesCount*;  
 private static int *allTargetsCount*;  
 public static Hero *enemyHero*;  
 public static Hero *playerHero*;  
 private static int *playerMana*;  
 private static int *playerMaxMana*;  
  
 public static void main(String[] args)  
 {  
 *collection* = CardCollection.*load*();  
 if (false)  
 {  
 *collection* = new CardCollection();  
 *collection*.weapons.add(*deadline*.cloneCard());  
 *collection*.creatures.add(*caredOne*.cloneCard());  
 *collection*.creatures.add(*student*.cloneCard());  
 *collection*.save();  
 }  
 *basicPlayerHand*.add(*deadline*.cloneCard());  
 *basicPlayerHand*.add(*caredOne*.cloneCard());  
 *basicPlayerHand*.add(*student*.cloneCard());  
 *basicPlayerHand*.add(*caredOne*.cloneCard());  
 int choice = -1;  
 while (choice != 0)  
 {  
 System.*out*.println("Что вы хотите сделать?");  
 System.*out*.println("1. Посмотреть начальную руку.");  
 System.*out*.println("2. Очистить начальную руку.");  
 System.*out*.println("3. Добавить карту в начальную руку.");  
 System.*out*.println("4. Создать новую карту.");  
 System.*out*.println("5. Перейти в режим игры(beta).");  
 System.*out*.println("0. Завершить работу.");  
 System.*out*.print("Ваш выбор: ");  
 choice = ExceptionHelper.*readInt*();  
 switch (choice)  
 {  
 case 1:  
 *playerHand* = *basicPlayerHand*;  
 *showHand*();  
 break;  
 case 2:  
 *clearHand*();  
 break;  
 case 3:  
 *cardAddMenu*();  
 break;  
 case 4:  
 *cardCreatorMenu*();  
 break;  
 case 5:  
 *gameMenu*();  
 break;  
 case 0:  
 System.*out*.println("Завершаю работу.");  
 break;  
 default:  
 System.*out*.println("Некорректный ввод.");  
 break;  
 }  
 }  
 }  
 public static void cardAddMenu()  
 {  
 System.*out*.println("Какую карту вы хотите добавить?");  
 System.*out*.println("Оружие");  
 int i = 0;  
 for (Weapon weapon: *collection*.weapons)  
 {  
 i++;  
 System.*out*.println(i + ". " + weapon.toString());  
 }  
 int iWeapon = i;  
 System.*out*.println("\nСущества");  
 for (Creature creature: *collection*.creatures)  
 {  
 i++;  
 System.*out*.println(i + ". " + creature.toString());  
 }  
 System.*out*.println("0. Отмена.");  
 System.*out*.print("Номер добавляемой карты: ");  
 int choice = ExceptionHelper.*readInt*();  
 while (choice < 0 || choice > i)  
 {  
 System.*out*.print("Не существует такой карты. Номер добавляемой карты: ");  
 choice = ExceptionHelper.*readInt*();  
 }  
 if (choice > 0)  
 {  
 if (choice <= iWeapon)  
 *basicPlayerHand*.add(*collection*.weapons.get(choice - 1).cloneCard());  
 else  
 *basicPlayerHand*.add(*collection*.creatures.get(choice - iWeapon - 1).cloneCard());  
 }  
 }  
 public static void clearHand()  
 {  
 *basicPlayerHand* = new ArrayList<Card>();  
 }  
 public static void cardCreatorMenu()  
 {  
 System.*out*.println("1. Создать существо.");  
 System.*out*.println("2. Создать оружие.");  
 System.*out*.println("0. Отмена.");  
 System.*out*.print("Ваш выбор: ");  
 int choice = ExceptionHelper.*readInt*();  
 switch (choice)  
 {  
 case 1:  
 System.*out*.print("Имя: ");  
 String tempName = ExceptionHelper.*readString*();  
 System.*out*.print("Стоймость в мане: ");  
 int tempMana = ExceptionHelper.*readInt*();  
 while (tempMana < 0 || tempMana > 10)  
 {  
 System.*out*.print("Существо не может стоить менее 0 и более 10 маны. Стоймость в мане: ");  
 tempMana = ExceptionHelper.*readInt*();  
 }  
 System.*out*.print("Атака: ");  
 int tempAttack = ExceptionHelper.*readInt*();  
 while (tempAttack < 0)  
 {  
 System.*out*.print("Существо не может иметь менее 0 атаки. Атака: ");  
 tempAttack = ExceptionHelper.*readInt*();  
 }  
 System.*out*.print("Здоровье: ");  
 int tempHealth = ExceptionHelper.*readInt*();  
 while (tempHealth < 1)  
 {  
 System.*out*.print("Существо не может иметь менее 1 здоровья. Здоровье: ");  
 tempHealth = ExceptionHelper.*readInt*();  
 }  
 System.*out*.print("Имеет божественный щит(-/+): ");  
 String tempBubble = ExceptionHelper.*readString*();  
 while (tempBubble != "+" && tempBubble != "-")  
 {  
 System.*out*.print("Введите - или +. Имеет божественный щит(-/+): ");  
 tempBubble = ExceptionHelper.*readString*();  
 }  
 if (tempBubble == "-")  
 *collection*.creatures.add(new Creature(tempName, tempMana, tempAttack, tempHealth));  
 else  
 *collection*.creatures.add(new Creature(tempName, tempMana, tempAttack, tempHealth, true));  
 *collection*.save();  
 break;  
 case 2:  
 System.*out*.print("Имя: ");  
 String tempWeaponName = ExceptionHelper.*readString*();  
 System.*out*.print("Стоймость в мане: ");  
 int tempWeaponMana = ExceptionHelper.*readInt*();  
 while (tempWeaponMana < 0 || tempWeaponMana > 10)  
 {  
 System.*out*.print("Оружие не может стоить менее 0 и более 10 маны. Стоймость в мане: ");  
 tempWeaponMana = ExceptionHelper.*readInt*();  
 }  
 System.*out*.print("Атака: ");  
 int tempWeaponAttack = ExceptionHelper.*readInt*();  
 while (tempWeaponAttack < 0)  
 {  
 System.*out*.print("Оружие не может иметь менее 0 атаки. Атака: ");  
 tempWeaponAttack = ExceptionHelper.*readInt*();  
 }  
 System.*out*.print("Прочность: ");  
 int tempWeaponHealth = ExceptionHelper.*readInt*();  
 while (tempWeaponHealth < 1)  
 {  
 System.*out*.print("Оружие не может иметь менее 1 прочности. Прочность: ");  
 tempWeaponHealth = ExceptionHelper.*readInt*();  
 }  
 *collection*.weapons.add(new Weapon(tempWeaponName, tempWeaponMana, tempWeaponAttack, tempWeaponHealth));  
 *collection*.save();  
 break;  
 case 0:  
 System.*out*.println("Возвращаемся в главное меню.");  
 break;  
 default:  
 System.*out*.println("Действие не распознано. Возвращаемся в главное меню.");  
 break;  
 }  
 }  
 public static void gameMenu()  
 {  
 *playerMaxMana* = 0;  
 *enemyHero* = *MinObrRF*.cloneCard();  
 *enemySide*.add(*student*.cloneCard());  
 *enemySide*.add(*caredOne*.cloneCard());  
 *enemySide*.add(*student*.cloneCard());  
 *setHero*(*debtor*.cloneCard());  
 *playerHand*.clear();  
 for (Card card: *basicPlayerHand*)  
 *playerHand*.add(card.cloneCard());  
 while (*playerHero* != null && *enemyHero* != null)  
 {  
 if (*playerMaxMana* < 10)  
 *playerMaxMana*++;  
 *playerMana* = *playerMaxMana*;  
 for (Creature creature: *playerSide*)  
 creature.canAttack = true;  
 *playerHero*.canAttack = true;  
 int choice = -1;  
 System.*out*.println("Начинается ваш ход. У вас " + *playerMana* + " кристаллов маны. Выберите действие.");  
 System.*out*.println("1. Атаковать существом или героем.");  
 System.*out*.println("2. Разыграть карту из руки.");  
 System.*out*.println("0. Закончить ход.");  
 System.*out*.print("Ваш выбор: ");  
 while (choice != 0)  
 {  
 choice = ExceptionHelper.*readInt*();  
 switch (choice)  
 {  
 case 1:  
 *attackOption*();  
 if (*playerHero* == null || *enemyHero* == null)  
 {  
 choice = 0;  
 break;  
 }  
 System.*out*.print("У вас " + *playerMana* + " из " + *playerMaxMana* + " кристаллов маны. Ваше следующее действие: ");  
 break;  
 case 2:  
 *updateCounters*();  
 if (*playerHandSize* > 0)  
 {  
 *playCardOption*();  
 if (*playerHero* == null || *enemyHero* == null)  
 {  
 choice = 0;  
 break;  
 }  
 }  
 System.*out*.print("У вас " + *playerMana* + " из " + *playerMaxMana* + " кристаллов маны. Ваше следующее действие: ");  
 break;  
 case 0:  
 System.*out*.println("Ваш ход окончен.");  
 break;  
 default:  
 System.*out*.print("Действие не распознано, повторите выбор: ");  
 break;  
 }  
 }  
 }  
 }  
 public static void attackOption()  
 {  
 System.*out*.println("Ваши персонажи:");  
 *updateCounters*();  
 *showPlayerCharacters*();  
 System.*out*.println("0. Отмена");  
 System.*out*.print("Выберите, кем хотите атаковать: ");  
 int choice = ExceptionHelper.*readInt*();  
 while (choice < 0 || choice > *playerCreaturesCount* + 1)  
 {  
 System.*out*.print("Не существует такого персонажа. Повторите ввод: ");  
 choice = ExceptionHelper.*readInt*();  
 }  
 Character attacker = null;  
 if (choice == 1)  
 *continueAttackOption*(*playerHero*);  
 else if (choice > 1)  
 *continueAttackOption*(*playerSide*.get(choice - 2));  
 }  
 public static void continueAttackOption(Character attacker)  
 {  
 if (!attacker.canAttack || attacker.damage <= 0)  
 {  
 if (attacker != *playerHero* || !*playerHero*.canAttack())  
 {  
 System.*out*.println("Выбранный персонаж не может атаковать");  
 return;  
 }  
 }  
 System.*out*.println("Персонажи противника:");  
 *updateCounters*();  
 *showEnemyCharacters*();  
 System.*out*.println("0. Отмена");  
 System.*out*.print("Выберите, кого хотите атаковать: ");  
 int choice = ExceptionHelper.*readInt*();  
 while (choice < 0 || choice > *enemyCreaturesCount* + 1)  
 {  
 System.*out*.print("Не существует такого персонажа. Повторите ввод: ");  
 choice = ExceptionHelper.*readInt*();  
 }  
 if (choice == 1)  
 *attack*(attacker, *enemyHero*);  
 else if (choice > 1)  
 *attack*(attacker, *enemySide*.get(choice - 2));  
 }  
 public static void playCardOption()  
 {  
 System.*out*.println("Ваши карты:");  
 *showHand*();  
 System.*out*.println("0. Отмена");  
 System.*out*.print("Выберите, какую карту хотите разыграть: ");  
 int choice = ExceptionHelper.*readInt*();  
 while (choice < 0 || choice > *playerHandSize*)  
 {  
 System.*out*.print("Не существует такой карты. Повторите ввод: ");  
 choice = ExceptionHelper.*readInt*();  
 }  
 if (choice != 0)  
 *playCard*(*playerHand*.get(choice - 1));  
 }  
 public static void attack(Character attacker, Character target)  
 {  
 attacker.attack(target);  
 attacker.canAttack = false;  
 if (attacker.health < 1)  
 {  
 *playerSide*.remove(attacker);  
 *enemySide*.remove(attacker);  
 if (attacker == *playerHero*)  
 {  
 *playerHero* = null;  
 System.*out*.println("Ваш герой повержен.");  
 }  
 if (attacker == *enemyHero*)  
 {  
 *enemyHero* = null;  
 System.*out*.println("Вы победили.");  
 }  
 }  
 if (target.health < 1)  
 {  
 *playerSide*.remove(target);  
 *enemySide*.remove(target);  
 if (target == *playerHero*)  
 {  
 *playerHero* = null;  
 System.*out*.println("Ваш герой повержен.");  
 }  
 if (target == *enemyHero*)  
 {  
 *enemyHero* = null;  
 System.*out*.println("Вы победили.");  
 }  
 }  
 }  
 public static void setHero(Hero hero)  
 {  
 *playerHero* = hero;  
 }  
 public static void summonMinion(Creature minion)  
 {  
 *playerSide*.add(minion);  
 }  
 private static void playCard(Card card)  
 {  
 if (*playerMana* >= card.manaCost)  
 {  
 if (card.needsTarget)  
 {  
 *playerMana* -= card.manaCost;  
 *updateCounters*();  
 System.*out*.println("Выберите цель.");  
 *showTargets*();  
 int pointer = 0;  
 System.*out*.print("Ваша цель: ");  
 pointer = ExceptionHelper.*readInt*();  
 while (pointer < 0 || pointer > *allTargetsCount*)  
 {  
 System.*out*.println("Не существует такой цели.");  
 System.*out*.print("Ваша цель: ");  
 pointer = ExceptionHelper.*readInt*();  
 }  
 if (pointer == 1)  
 card.onPlay(*enemyHero*);  
 else if (pointer == 2)  
 card.onPlay(*playerHero*);  
 else if (pointer < *enemyCreaturesCount* + 3)  
 card.onPlay(*enemySide*.get(pointer - 3));  
 else  
 card.onPlay(*playerSide*.get(pointer - *enemyCreaturesCount* - 3));  
 }  
 else  
 {  
 *playerMana* -= card.manaCost;  
 card.onPlay();  
 }  
 *playerHand*.remove(card);  
 }  
 else  
 System.*out*.println("Не хватает маны.");  
 }  
 private static void showTargets()  
 {  
 System.*out*.println("1. Герой противника. " + *enemyHero*.shortDescription());  
 System.*out*.println("2. Ваш герой. " + *playerHero*.shortDescription());  
 int i = 3;  
 for (Creature creature: *enemySide*)  
 {  
 System.*out*.println(i + ". Существо противника. " + creature.shortDescription());  
 i++;  
 }  
 for (Creature creature: *playerSide*)  
 {  
 System.*out*.println(i + ". Ваше существо. " + creature.shortDescription());  
 i++;  
 }  
 }  
 private static void showPlayerCharacters()  
 {  
 if (*playerHero*.canAttack)  
 System.*out*.println("1. Ваш герой. " + *playerHero*.shortDescription());  
 else  
 System.*out*.println("1. Ваш герой. " + *playerHero*.shortDescription() + " Не может атаковать на этом ходу.");  
 int i = 2;  
 for (Creature creature: *playerSide*)  
 {  
 if (creature.canAttack)  
 System.*out*.println(i + ". Ваше существо. " + creature.shortDescription());  
 else  
 System.*out*.println(i + ". Ваше существо. " + creature.shortDescription() + " Не может атаковать на этом ходу.");  
 i++;  
 }  
 }  
 private static void showEnemyCharacters()  
 {  
 System.*out*.println("1. Герой противника. " + *enemyHero*.shortDescription());  
 int i = 2;  
 for (Creature creature: *enemySide*)  
 {  
 System.*out*.println(i + ". Существо противника. " + creature.shortDescription());  
 i++;  
 }  
 }  
 private static void showHand()  
 {  
 int i = 1;  
 for (Card card: *playerHand*)  
 {  
 System.*out*.println(i + ". " + card.toString());  
 i++;  
 }  
 }  
 private static void updateCounters()  
 {  
 *enemyCreaturesCount* = 0;  
 for (Creature creature: *enemySide*)  
 *enemyCreaturesCount*++;  
 *playerCreaturesCount* = 0;  
 for (Creature creature: *playerSide*)  
 *playerCreaturesCount*++;  
 *allTargetsCount* = *enemyCreaturesCount* + *playerCreaturesCount* + 2;  
 *playerHandSize* = 0;  
 for (Card card: *playerHand*)  
 *playerHandSize*++;  
 }  
}

Класс Card

package Lab\_4;  
  
public abstract class Card  
{  
 public String name;  
 public int manaCost;  
 public void onPlay(){}  
 public void onPlay(Character target){}  
 public abstract Card cloneCard();  
 public boolean needsTarget;  
}

Класс Spell

package Lab\_4;  
  
public class Spell extends Card  
{  
 private String description;  
 public Spell cloneCard()  
 {  
 return null;  
 }  
 public String toString()  
 {  
 return name + ". Стоймость " + manaCost + ". " + description;  
 }  
}

Класс Interactable

package Lab\_4;  
  
public abstract class Interactable extends Card  
{  
 public int health;  
 public int maxHealth;  
 public int damage;  
 public String shortDescription()  
 {  
 if (health > 0)  
 return "(" + name + " (" + damage + "/" + health + "))";  
 else  
 return "(" + name + " (отчислен))";  
 }  
 public void setHealth(int value)  
 {  
 if (value <= 0)  
 die();  
 else  
 health = value;  
 }  
 public void setMaxHealth(int value)  
 {  
 if (value <= 0)  
 die();  
 else  
 {  
 maxHealth = value;  
 if (health > maxHealth)  
 health = maxHealth;  
 }  
 }  
 public void setDamage(int value)  
 {  
 if (value > 0)  
 damage = value;  
 else  
 damage = 0;  
 }  
 public void takeDamage(int value)  
 {  
 health -= value;  
 if (health <= 0)  
 die();  
 }  
 public void die()  
 {  
 }  
}

Класс Weapon

package Lab\_4;  
  
public class Weapon extends Interactable  
{  
 public Weapon(String name, int cost, int damage, int health)  
 {  
 this.name = name;  
 this.manaCost = cost;  
 this.health = health;  
 this.maxHealth = health;  
 this.damage = damage;  
 }  
 public String toString()  
 {  
 return name + ". Стоймость " + manaCost + ". Атака " + damage + ". Прочность " + health + ".";  
 }  
 public Weapon cloneCard()  
 {  
 return new Weapon(name, manaCost, damage, health);  
 }  
 public void onPlay()  
 {  
 Main.*playerHero*.equipWeapon(this);  
 }  
}

Класс Character

package Lab\_4;  
  
public abstract class Character extends Interactable  
{  
 public boolean canAttack = false;  
 public void attack(Character target)  
 {  
 System.*out*.println(shortDescription() + " -> " + target.shortDescription());  
 target.takeDamage(damage);  
 takeDamage(target.damage);  
 System.*out*.println(shortDescription() + " -- " + target.shortDescription());  
 }  
 public void heal(int value)  
 {  
 health += value;  
 if (health > maxHealth)  
 health = maxHealth;  
 }  
}

Класс Creature

package Lab\_4;  
  
public class Creature extends Character  
{  
 public Creature(String name, int cost, int damage, int health)  
 {  
 this.name = name;  
 this.manaCost = cost;  
 this.health = health;  
 this.maxHealth = health;  
 this.damage = damage;  
 }  
 public Creature(String name, int cost, int damage, int health, boolean shielded)  
 {  
 this.name = name;  
 this.manaCost = cost;  
 this.health = health;  
 this.maxHealth = health;  
 this.damage = damage;  
 divineShield = shielded;  
 }  
 public void onPlay()  
 {  
 Main.*summonMinion*(this);  
 }  
 public boolean divineShield;  
 public void takeDamage(int value)  
 {  
 if (value > 0)  
 {  
 if (divineShield)  
 divineShield = false;  
 else  
 super.takeDamage(value);  
 }  
 }  
 public String toString()  
 {  
 if (divineShield)  
 return name + ". Стоймость " + manaCost + ". Атака " + damage + ". Здоровье " + health + ". Имеет божественный щит.";  
 else  
 return name + ". Стоймость " + manaCost + ". Атака " + damage + ". Здоровье " + health + ".";  
 }  
 public Creature cloneCard()  
 {  
 return new Creature(name, manaCost, damage, health, divineShield);  
 }  
 public String shortDescription()  
 {  
 if (health > 0)  
 {  
 if (divineShield)  
 return "(" + name + " (" + damage + "/" + health + ") Sh)";  
 else  
 return "(" + name + " (" + damage + "/" + health + "))";  
 }  
 else  
 return "(" + name + " (отчислен))";  
 }  
}

Класс Hero

package Lab\_4;  
  
public class Hero extends Character  
{  
 private int armor;  
 private Weapon equippedWeapon;  
 public Hero cloneCard()  
 {  
 Hero hero = new Hero(name, manaCost, damage, maxHealth, armor);  
 if (equippedWeapon != null)  
 hero.equipWeapon(equippedWeapon.cloneCard());  
 return hero;  
 }  
 public Hero(String name, int cost, int damage, int health)  
 {  
 this.name = name;  
 this.manaCost = cost;  
 this.health = health;  
 this.maxHealth = health;  
 this.damage = damage;  
 }  
 public Hero(String name, int cost, int damage, int health, int armor)  
 {  
 this.name = name;  
 this.manaCost = cost;  
 this.health = health;  
 this.maxHealth = health;  
 this.damage = damage;  
 this.armor = armor;  
 }  
 public void takeDamage(int value)  
 {  
 if (armor > value)  
 armor -= value;  
 else  
 {  
 health -= value - armor;  
 armor = 0;  
 }  
 }  
 public boolean canAttack()  
 {  
 if (equippedWeapon != null)  
 return canAttack && equippedWeapon.damage > 0;  
 else  
 return false;  
 }  
 public void equipWeapon(Weapon weapon)  
 {  
 equippedWeapon = weapon;  
 }  
 public void attack(Character target)  
 {  
 System.*out*.println(shortDescription() + " -> " + target.shortDescription());  
 if (equippedWeapon != null)  
 {  
 target.takeDamage(damage + equippedWeapon.damage);  
 equippedWeapon.takeDamage(1);  
 if (equippedWeapon.health < 1)  
 equippedWeapon = null;  
 }  
 else  
 target.takeDamage(damage);  
 takeDamage(target.damage);  
 System.*out*.println(shortDescription() + " -- " + target.shortDescription());  
 }  
 public void onPlay()  
 {  
 Main.*setHero*(this);  
 }  
 public String toString()  
 {  
 if (equippedWeapon == null)  
 {  
 if (armor < 1)  
 return name + ". Стоймость " + manaCost + ". Атака " + damage + ". Здоровье " + health + ".";  
 else  
 return name + ". Стоймость " + manaCost + ". Атака " + damage + ". Здоровье " + health + ". Броня " + armor + ".";  
 }  
 else  
 {  
 if (armor < 1)  
 return name + ". Стоймость " + manaCost + ". Атака " + damage + ". Экипировано оружие" + equippedWeapon.shortDescription() + ". Здоровье " + health + ".";  
 else  
 return name + ". Стоймость " + manaCost + ". Атака " + damage + ". Экипировано оружие" + equippedWeapon.shortDescription() + ". Здоровье " + health + ". Броня " + armor + ".";  
 }  
 }  
 public String shortDescription()  
 {  
 if (health > 0)  
 {  
 if (equippedWeapon == null)  
 {  
 if (armor > 0)  
 return "(" + name + " (" + damage + "/" + health + "+" + armor + "))";  
 else  
 return "(" + name + " (" + damage + "/" + health + "))";  
 }  
 else  
 {  
 if (armor > 0)  
 return "(" + name + " (" + damage + "/" + health + "+" + armor + ")" + " + W(" + equippedWeapon.damage + "/" + equippedWeapon.health + "))";  
 else  
 return "(" + name + " (" + damage + "/" + health + ")" + " + W(" + equippedWeapon.damage + "/" + equippedWeapon.health + "))";  
 }  
 }  
 else  
 return "(" + name + " (отчислен))";  
 }  
}

Класс CardCollection

package Lab\_4;  
  
import java.io.\*;  
import java.util.ArrayList;  
  
public class CardCollection  
{  
 public ArrayList<Hero> heroes = new ArrayList<Hero>();  
 public ArrayList<Creature> creatures = new ArrayList<Creature>();  
 public ArrayList<Weapon> weapons = new ArrayList<Weapon>();  
 public ArrayList<Spell> spells = new ArrayList<Spell>();  
 public void save()  
 {  
 try(FileWriter writer = new FileWriter("test.txt", false))  
 {  
 String text = "Kitty 1 1 1 true/";  
 for (Creature creature: creatures)  
 text = text + creature.name.replace(" ", "\_") + " " + creature.manaCost + " " + creature.damage + " " + creature.maxHealth + " " + creature.divineShield + "/";  
 text = text + "Mark:Bone 2 2 2/";  
 for (Weapon weapon: weapons)  
 text = text + weapon.name.replace(" ", "\_") + " " + weapon.manaCost + " " + weapon.damage + " " + weapon.maxHealth + "/";  
 writer.write(text);  
 writer.flush();  
 }  
 catch(IOException ex)  
 {  
 System.*out*.println(ex.getMessage());  
 }  
 }  
 public static CardCollection load()  
 {  
 try(FileReader reader = new FileReader("test.txt"))  
 {  
 CardCollection tempCC = new CardCollection();  
 int symbol = reader.read();  
 String line = "";  
 while(symbol != -1)  
 {  
 line = line + (char)symbol;  
 symbol = reader.read();  
 }  
 String[] line1 = line.split("Mark:");  
 String[] line2 = line1[0].split("/");  
 String[] line2x;  
 Creature tempCreature;  
 for (int i = 0; i < line2.length; i++)  
 {  
 line2x = line2[i].split(" ");  
 tempCreature = new Creature(line2x[0].replace("\_", " "), Integer.*parseInt*(line2x[1]), Integer.*parseInt*(line2x[2]), Integer.*parseInt*(line2x[3]), Boolean.*parseBoolean*(line2x[4]));  
 tempCC.creatures.add(tempCreature);  
 }  
 String[] line3 = line1[1].split("/");  
 String[] line3x;  
 Weapon tempWeapon;  
 for (int i = 0; i < line3.length; i++)  
 {  
 line3x = line3[i].split(" ");  
 tempWeapon = new Weapon(line3x[0].replace("\_", " "), Integer.*parseInt*(line3x[1]), Integer.*parseInt*(line3x[2]), Integer.*parseInt*(line3x[3]));  
 tempCC.weapons.add(tempWeapon);  
 }  
 return tempCC;  
 }  
 catch(IOException ex)  
 {  
 System.*out*.println(ex.getMessage());  
 return new CardCollection();  
 }  
 }  
}

Список использованных источников