МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Объекто-ориентированное программирование»
Тема: Создание классов

Студент гр. 2382 Ваньков Я.С.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2023

Цель работы

Изучить методы работы с классами. Написать простую программу в стиле ООП.

Задание

а) Создать класс игрока. У игрока должны быть поля, которые определяют его характеристики, например кол-во жизней, очков и.т.д. Также в классе игрока необходимо реализовать ряд методов для работы с его характеристиками. Данные методы должны контролировать значения характеристик (делать проверку на диапазон значений).

б) Создать класс, передвигающий игрока по полю и работу с характеристиками. Данный класс всегда должен знать об объекте игрока, которым управляет, но не создавать класс игрока. В следующих лаб. работах данный класс будет проводить проверку, может ли игрок совершить перемещение по карте.

Примечания:

Не забывайте для полей и методов определять модификатор доступа

Для указания направления движения можно использовать перечисление enum или дополнительную систему классов. Использования чисел или строк является для указания направления является плохой практикой

Делать отдельный метод под каждое направление делает класс перегруженным, и в будущем ограничивает масштабирование класса

Выполнение работы

Player.

Класс игрока. В публичных полях находятся константные указатели на все характеристики игрока. Не имеет методов.

Characterisric.

Главный класс для характеристик: от него наследуются все характеристики. В конструктор класса передается минимальное, максимальное и установленное значение.

Определено три метода:

getValue — возвращает значение

setValue — меняет значение, учитывая ограничение класса.

Check — проеверяет измененное значение, согласно ограничениям класса

Наследники класса характеристик:

Health.

Добавляется приватное поле игрока — is_dead, которое принимает только булевые типы данных.

Также добавляется метод на проверку наличия здоровья у игрока.

Armor

Count

Money

Power

Score

Item.

В приватных полях лежит название предмета и его характеристики(они же и инициализируются в конструкторе).

Методы:

getName — вовзвращает имя

setName — изменяет имя

Weapon

В приватных полях класса лежит урон, наносимый оружием и название оружия.

Методы класса:

getDamage — вовзвращает урон

getName — вовзвращает имя

setDamage — изменяет урон

setName — изменяет имя

use — изменяет значение силы игрока на значение урона

Coordinate

В приватных полях имеет координаты по X и Y.

Дружественный класс для MoveManager

Публичные методы:

getX — возвращает X

getY — возвращает Y

getPair — возвращает пару

MoveManager

Класс изменяющий координаты игрока

Конструктор класса принимает игрока, над которым будут произвоодится действия

В приватных полях находится указатель на игрока, над которым воздействует менеджер.

Методы в зависимости от переданного значения изменяют координату игрока.

Выводы

Были изучены методы работы с классами. Были написаны классы игрока, его характеристик и классы взаимодействия игрока с его координатами

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
class Characteristic
{
    protected:
        int value;
        int max_value;
        int min_value;
    public:
        Characteristic(int min_value,int max_value,int value){
            this->max_value = std::max(min_value,max_value);
            this->min_value = std::min(min_value,max_value);
            setValue(value);
        }
        void setValue(int val){
            int status = check(val);
            switch(status){
                case 0:
                    this->value = val;
                    break;
                case -1:
                    this->value = this->min_value;
                    break;
                case 1:
                    this->value = this->max_value;
                    break;
            }
        }

        int getValue(){
            return this->value;
        }

        int check(int value){
            if (this->min_value > value) return -1;
            if (this->max_value < value) return 1;
            return 0;
        }

};

class Health : public Characteristic
{
    public:
        Health(int min_value,int max_value,int value): Characteristic(min_value,
max_value, value){
```

```
                isDead();
            }
            bool isDead(){
                if (this->value <= this->min_value){
                    this->is_dead = true;
                    std::cout << "Поздравляю, Вы мертвы!" << '¥n';
                }
                return is_dead;
            }
        private:
            bool is_dead = false;
    };

    class Power: public Characteristic
    {
        public:
            Power(int min_value,int max_value,int value): Characteristic(min_value,
max_value, value){}
    };

    class Armor: public Characteristic
    {
        public:
            Armor(int min_value,int max_value,int value): Characteristic(min_value,
max_value, value){}
    };

    class Money: public Characteristic
    {
        public:
            Money(int min_value,int max_value,int value): Characteristic(min_value,
max_value, value){}
    };

    class Score: public Characteristic
    {
        public:
            Score(int min_value,int max_value,int value): Characteristic(min_value,
max_value, value){}
    };

    class Count: public Characteristic
    {
        public:
            Count(int min_value,int max_value,int value): Characteristic(min_value,
max_value, value){}
    };

    enum class Direction{right,up,left,down};
    class MoveManager;
```

```
class Coordinate
{
    friend MoveManager;
    int x;
    int y;
    public:
        Coordinate(int x, int y){
            this->x = x;
            this->y = y;
        }
        int getX(){
            return this->x;
        }
        int getY(){
            return this->y;
        }
        std::pair<int,int> getPair(){
            std::pair pair = std::make_pair(this->x,this->y);
            return pair;
        }
};
```

```
class Weapon
{
    int damage;
    std::string name;
    public:
        Weapon(std::string name, int damage, Power* power){
            this->name = name;
            this->damage = damage;
            use(power);
        }
        int getDamage(){return this->damage;}
        void setDamage(int damage){this->damage = damage;}
        std::string getName(){return this->name;}
        void setName(std::string name){this->name = name;}
        void use(Power* power){
            power->setValue(this->damage);
        }
};

class Item
{
    std::string name;
    Health* health;
```

```
    Power* power;
    Armor* armor;
    Count* count;
    public:
        Item(std::string name,int health=0, int power=0, int armor=0, int count=0){
            this->name = name;
            this->health = new Health(0,250,health);
            this->power = new Power(0,50,power);
            this->armor = new Armor(0,50,armor);
            this->count = new Count(0,16,count);
        }
        std::string getName(){return this->name;}
        void setName(std::string name){this->name = name;}

};



class Player
{
    public:
        Health* const health = new Health(0,1000,1000);
        Power* const power = new Power(0,100,0);
        Armor* const armor = new Armor(0,100,0);
        Money* const money = new Money(0,10000,600);
        Score* const score = new Score(0,1000,0);
        Weapon* const weapon = new Weapon("К у л а к и", 2, power);
        Item* const item = new Item("Ф л а с к а", 100, 0, 0, 1);
        Coordinate* const coordinate = new Coordinate(0,0);
};

class MoveManager{
    Player* player;
    public:
        MoveManager(Player* player){
            this->player = player;
        }
        void move(Direction direction){
            switch (direction){
                case Direction::right:
                    this->player->coordinate->x += 1;
                    break;
                case Direction::left:
                    this->player->coordinate->x -= 1;
                    break;
                case Direction::up:
                    this->player->coordinate->y += 1;
                    break;
                case Direction::down:
                    this->player->coordinate->y -= 1;
```

```
                        break;
                }
            }
    };



    int main()
    {
        Player player;
        MoveManager playerContol(&player);
        std::cout << "Оружие: " << player.weapon->getName() << std::endl;
        std::cout << "Расходники: " << player.item->getName() << std::endl;
        std::cout << "Характеристики: " << std::endl;
        std::cout << "Здоровье: " << player.health->getValue() << std::endl;
        std::cout << "Сила: " << player.power->getValue() << std::endl;
        std::cout << "Броня: " << player.armor->getValue() << std::endl;
        std::cout << "Монеты: " << player.money->getValue() << std::endl;
        std::cout << "Очки: " << player.score->getValue() << std::endl;
        std::cout << "Координата: " << player.coordinate->getX() << " " <<
player.coordinate->getY() << std::endl;
        playerContol.move(Direction::up);
        std::cout << "Координата: " << player.coordinate->getX() << " " <<
player.coordinate->getY() << std::endl;
        return 0;
    }
```