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

ОТЧЕТ

по практической работе № 1

по дисциплине «ООП»

Тема: Создание классов, конструкторов классов, методов классов, наследование

Студент гр. 8383 _____ Степанов В.Д.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2020

Цель работы.

Научиться работать с классами, конструкторами классов, методами классов, наследованием в языке C++.

Постановка задачи.

Разработать и реализовать набор классов:

- Класс игрового поля
- Набор классов юнитов

Игровое поле является контейнером для объектов представляющим прямоугольную сетку. Основные требования к классу игрового поля:

- Создание поля произвольного размера
- Контроль максимального количества объектов на поле
- Возможность добавления и удаления объектов на поле
- Возможность копирования поля (включая объекты на нем)
- Для хранения запрещается использовать контейнеры из stl

Юнит является объектов, размещаемым на поля боя. Один юнит представляет собой отряд. Основные требования к классам юнитов:

- Все юниты должны иметь как минимум один общий интерфейс
- Реализованы 3 типа юнитов (например, пехота, лучники, конница)
- Реализованы 2 вида юнитов для каждого типа(например, для пехоты могут быть созданы мечники и копейщики)
- Юниты имеют характеристики, отражающие их основные атрибуты, такие как здоровье, броня, атака.
- Юнит имеет возможность перемещаться по карте

Выполнение работы.

Был реализован класс игрового поля и класс клетки. В классе игрового поля был реализован методы: создания поля произвольного размера, удаление и добавление объектов на поле, копирования поля. Так же контролируется максимальное количество объектов на поле. Были реализованы конструкторы

копирования и перемещения. Так же был создан итератор для поля. Код классов приведен в приложении А.

Был реализован общий интерфейс для юнитов. Было реаизовано 3 типа юнитов: Shooter, Runner, Smoker. Для каждого из типов было реализовано два вида: Protected, Strong. Каждый из юнитов имеет характеристики: char name[3] (имя отображающиеся на поле), **int** helth(здоровье), **int** armor (броня), **int** attack (сила атаки) **int** x и **int** y (координаты нахождения на поле). Добавлена возможность перемещения юнитов по карте. Код реализованных классно приведен в приложении Б.

Был реализован класс игры, который объединяет выше перечисленные классы. Код класса приведен в приложении В.

Все методы принимают параметры оптимальным образом. Для атрибутов юнитов созданы свои классы. Для создания юнитов используются паттерн “Абстрактная фабрика”.

Выводы.

В ходе лабораторной работы был получен навык работы с классами, конструкторами классов, методами классов, наследованием в языке C++.

ПРИЛОЖЕНИЕ А
КОД КЛАССА ПОЛЯ

```
class Cell {

    bool isUsed = false;
    char nameUnit[3] = "  ";
public:
    void cellUsed(){
        isUsed = true;
    }

    void cellNotUsed(){
        isUsed = false;
    }

    bool getIsUsed(){
        return isUsed;
    }
    void setUnitName(char name[]){
        nameUnit[0] = name[0];
        nameUnit[1] = name[1];
        nameUnit[2] = name[2];


        isUsed = true;
    }

    void clearCell(){
        isUsed = false;
        nameUnit[0] = ' ';
        nameUnit[1] = ' ';
        nameUnit[2] = '\0';
    }


    char* getUnitName(){
        return nameUnit;
    }
};


class Field {
```

```
    int x, y;
    Cell** place;

public:

    ~Field () {
        for (int i = 0; i < x; i++){

            delete [] place[i];
        }

        delete [] place;
    }


    bool createField(int x, int y){
        if (x < 6){
            std::cout << "X не может быть меньше 6 клеток" <<
std::endl;
            return false;
        } else {
            this->x = x;
        }

            if (y < 6){
                    std::cout << "Y не может быть меньше 6 клеток"
<< std::endl;
                return false;
            } else {
                this->y = y;
            }

               printXY();

               this->place = new Cell*[this->x];


               for (int i = 0; i < this->x; i++){
                   place[i] = new Cell[this->y];

               }
```

```
        return true;
    }



    void printXY(){
        std::cout << "x = " << x << " " << "y = " << y <<
std::endl;
    }

    void printField () {

        for (int i = -1; i <= y; i++){
            for (int j = -1; j <= x; j++){

                if ((i == -1 && j == -1) || (i == -1 && j == x) ||
                    (i == y && j == -1) || (i == y && j == x)){

                    std::cout << " ";

                } else if (i == -1 || i == y) {

                    std::cout << "--";

                } else if (j == -1 || j == x) {

                    std::cout << "|";

                } else {
                    std::cout << place[j][i].getUnitName();

                }

            }
            std::cout << std::endl;
        }
    }

    bool cellIsUsed(int x, int y){
        return place[x][y].getIsUsed();
    }

    bool replaceUnit(int x, int y, int toX, int toY){ // a to b
```

```
        if(place[toX][toY].getIsUsed()) return false;

        Cell &a = place[x][y];
        Cell &b = place[toX][toY];

        b.setUnitName(a.getUnitName());
        a.clearCell();
        return true;

    }

    bool takeUnit(char name[], int x, int y){
        place[x][y].setUnitName(name);
        return true;
    }

    Cell** getCopyField (){

        Cell** cField;

        cField = new Cell*[this->x];

         for (int i = 0; i < this->x; i++){
             cField[i] = new Cell[this->y];
         }

        return cField;
    }

};
```

ПРИЛОЖЕНИЕ Б
КЛАССЫ ЮНИТОВ

```
class Unit {


 public:
    char name[3];
    int helth;
    int armor;
    int attack;
    int x;
    int y;


    void virtual hit () = 0;
    void virtual motion () = 0;
    void virtual setName (char unitNumber) = 0;
    char virtual *getName () = 0;
    void virtual setXY(int x, int y) = 0;
    int virtual getX() = 0;
    int virtual getY() = 0;
};

//-------------------------------------------------------------------
--------------

class Smoker: public Unit {
public:
    void hit () override {
        std::cout << "I'm smoking" << std::endl;
    }

    void motion () override {
        std::cout << "I'm going and smoking" << std::endl;
    }

    void setName (char unitNumber) override{
        if (unitNumber % 2 == 0) {
            name[0] = unitNumber;
            name[1] = '$';
        } else {
```

```
            name[0] = '$';
            name[1] = unitNumber;
        }
        name[2] = '\0';
    }

    char* getName () override {
        return name;
    }

    void setXY(int x, int y) override {
        this->x = x;
        this->y = y;
    }

    int getX () override {
        return x;
    }
    int getY () override {
        return y;
    }

};

class ProtectedSmoker: public Smoker {

public:
    ProtectedSmoker (char num) {
        helth = 15;
        armor = 15;
        attack = 1;
        setName(num);

    }
};

class StrongSmoker: public Smoker {

public:
    StrongSmoker (char num) {
        helth = 10;
        armor = 10;
        attack = 3;
```

```
        setName(num);
    }
};

//-----------------------------------------------------------------
--------------

class Runner: public Unit {
public:
    void hit () override {
        std::cout << "I'm hit and you're losing your points" <<
std::endl;
    }

    void motion () override {
        std::cout << "I'm going very fast" << std::endl;
    }

   void setName (char unitNumber) override{
        if (unitNumber % 2 == 0) {
            name[0] = unitNumber;
            name[1] = '!';
        } else {
            name[0] = '!';
            name[1] = unitNumber;
        }
        name[2] = '\0';
    }

    char* getName () override {
        return name;
    }

    void setXY(int x, int y) override {
        this->x = x;
        this->y = y;
    }

    int getX () override {
        return x;
    }
    int getY () override {
        return y;
```

```
    }


};


class ProtectedRunner: public Runner {
public:

    ProtectedRunner (char num){
        helth = 8;
        armor = 13;
        attack = 5;

        setName(num);
    }
};


class StrongRunner: public Runner {
public:

    StrongRunner (char num){
        helth = 8;
        armor = 8;
        attack = 7;

        setName(num);
    }
};

//-------------------------------------------------------------
--------------

class Shooter: public Unit {
public:
    void hit () override {
          std::cout << "I'm shooting at you" << std::endl;
      }

      void motion () override {
          std::cout << "I'm going and wathching you" << std::endl;
      }
```

```
    void setName (char unitNumber) override{
        if (unitNumber % 2 == 0) {
            name[0] = unitNumber;
            name[1] = '>';
        } else {
            name[0] = '<';
            name[1] = unitNumber;
        }
        name[2] = '\0';
    }

    char* getName () override {
        return name;
    }

    void setXY(int x, int y) override {
        this->x = x;
        this->y = y;
    }

    int getX () override {
        return x;
    }
    int getY () override {
        return y;
    }

};

class ProtectedShooter: public Shooter {

public:
    ProtectedShooter (char num) {
        helth = 5;
        armor = 10;
        attack = 4;


        setName(num);

    }
};
```

```
class StrongShooter: public Shooter {

public:
    StrongShooter (char num) {
        helth = 5;
        armor = 5;
        attack = 6;


        setName(num);


    }
};

//-----------------------------------------------------------------
--------------

class UnitsFactory {
public:


    virtual Unit* CreateUnit (char name, bool isProtected) = 0;

};

class ConcreteUnitShooter: public UnitsFactory {
public:

    Unit* CreateUnit (char num, bool isProtected) override {
        if (isProtected){
            return new ProtectedShooter(num);
        }

        return new StrongShooter( num);

    }

};
```

```
class ConcreteUnitSmoker: public UnitsFactory {
public:

    Unit* CreateUnit (char num, bool isProtected) override {
        if (isProtected){
            return new ProtectedSmoker(num);
        }

        return new StrongSmoker( num);

    }

};


class ConcreteUnitRunner: public UnitsFactory {
public:

    Unit* CreateUnit (char num, bool isProtected) override {
        if (isProtected){
            return new ProtectedRunner( num);
        }

        return new StrongRunner( num);

    }

};


#endif
```

ПРИЛОЖЕНИЕ В
КЛАСС ИГРЫ

```
class Game {

    Unit* firstPlayer[9];
    Unit* secondPlayer[9];
    Field f;
    int x, y;

    char numberSUnitsFirstPlayer[3] = {'/', '/', '/'};  // '/' ==
47
    char numberSUnitsSecondPlayer[3] = {'0', '0', '0'}; // '0' ==
48
    int countUnitsFirstPlauer;
    int countUnitsSecondPlauer;


public:

    void startGame(){

        int chosenGame;


        do{
            std::cout << "Выберите игру : 1 - стандартная, 2 -
пользовательская, 3 - завершение" << std::endl;
            std::cin >> chosenGame;


            if (std::cin.fail() || chosenGame > 3 || chosenGame <
1 ) // если предыдущее извлечение оказалось неудачным,
            {
                std::cin.clear(); // то возвращаем cin в 'обычный'
режим работы
                std::cin.ignore(32767,'\n'); // и удаляем значения
предыдущего ввода из входного буфера
                std::cout << "Неверный ввод!" << std::endl;
                continue;
            }

            break;
```

```
        }while(true);



        if (chosenGame == 1){
            standartGame();

        } else if (chosenGame == 2){
            modifiedGame();
        }

//        f.createField(x, y);
//        setUnitsToStart();
//        f.printField();
//
//        while (true) {
//
//
//        std::cout << choseUnitToMove () << std::endl;
//
//
//        f.printField();
//        }



    }

private:


    void setUnitsToStart(){

        firstPlayer[0] = doUnits(0, numberSUnitsFirstPlayer[0],
false);
        firstPlayer[1] = doUnits(1, numberSUnitsFirstPlayer[1],
false);
        firstPlayer[2] = doUnits(1, numberSUnitsFirstPlayer[1],
false);
        firstPlayer[3] = doUnits(2, numberSUnitsFirstPlayer[2],
false);
        firstPlayer[4] = doUnits(2, numberSUnitsFirstPlayer[2],
false);
```

```
        firstPlayer[5] = doUnits(2, numberSUnitsFirstPlayer[2],
false);
        countUnitsFirstPlauer = 6;

        secondPlayer[0] = doUnits(0, numberSUnitsSecondPlayer[0],
false);
        secondPlayer[1] = doUnits(1, numberSUnitsSecondPlayer[1],
false);
        secondPlayer[2] = doUnits(1, numberSUnitsSecondPlayer[1],
false);
        secondPlayer[3] = doUnits(2, numberSUnitsSecondPlayer[2],
false);
        secondPlayer[4] = doUnits(2, numberSUnitsSecondPlayer[2],
false);
        secondPlayer[5] = doUnits(2, numberSUnitsSecondPlayer[2],
false);
        countUnitsSecondPlauer = 6;

        f.takeUnit(firstPlayer[0]->getName(), x-1, 0);
        firstPlayer[0]->setXY(x-1, 0);
        f.takeUnit(firstPlayer[1]->getName(), x-2, 0);
        firstPlayer[1]->setXY(x-2, 0);
        f.takeUnit(firstPlayer[2]->getName(), x-1, 1);
        firstPlayer[2]->setXY(x-1, 1);
        f.takeUnit(firstPlayer[3]->getName(), x-1, 2);
        firstPlayer[3]->setXY(x-1, 2);
        f.takeUnit(firstPlayer[4]->getName(), x-2, 1);
        firstPlayer[4]->setXY(x-2, 1);
        f.takeUnit(firstPlayer[5]->getName(), x-3, 0);
        firstPlayer[5]->setXY(x-3, 0);

        f.takeUnit(secondPlayer[0]->getName(), 0, y-1);
        secondPlayer[0]->setXY(0, y-1);
        f.takeUnit(secondPlayer[1]->getName(), 0, y-2);
        secondPlayer[1]->setXY(0, y-2);
        f.takeUnit(secondPlayer[2]->getName(), 1, y-1);
        secondPlayer[2]->setXY(1, y-1);
        f.takeUnit(secondPlayer[3]->getName(), 2, y-1);
        secondPlayer[3]->setXY(2, y-1);
        f.takeUnit(secondPlayer[4]->getName(), 1, y-2);
        secondPlayer[4]->setXY(1, y-2);
        f.takeUnit(secondPlayer[5]->getName(), 0, y-3);
        secondPlayer[5]->setXY(0, y-3);
```

```
    }

    Unit* doUnits(int whatUnit, char &unitNumber, bool
isProtected){ // 2 - runner, 1 - shooter, 0 - smoker

        ConcreteUnitShooter sho;
        ConcreteUnitRunner run;
        ConcreteUnitSmoker smo;

        unitNumber += 2;



        if (whatUnit == 0 ) return smo.CreateUnit(unitNumber,
isProtected);

        else if (whatUnit == 1) return sho.CreateUnit(unitNumber,
isProtected);

        else return run.CreateUnit(unitNumber, isProtected);

    }


    bool choseUnitToMove (bool &endGame){

        char stroc[3];

        std::cout << "Введить имя персонажа и в какую сторону
переместить (!5 w1)" << std::endl;
        std::cin.ignore(32767,'\n');
        stroc[0] = std::cin.get();
        stroc[1] = std::cin.get();
        stroc[2] = std::cin.get();
        if (stroc[0] == 'e' && stroc[1] == 'x' && stroc[2] == 'i'
&& std::cin.get() == 't'){
            std::cout << "Конец ввода" << std::endl;
            endGame = false;
            return true;
        }
```

```
        if (stroc[2] != ' ') return false;


        stroc[2] = std::cin.get();

//        if (std::cin.get() != '\n'){
//            while (std::cin.get() != '\n'){}
//            return false;
//        }

        if ('1' <= stroc[0] &&  stroc[0] <='9'){

            return replaceUnit (stroc,secondPlayer ,
countUnitsSecondPlauer);

        } else {
            return replaceUnit (stroc,firstPlayer ,
countUnitsFirstPlauer);
        }

        return true;
    }


    bool replaceUnit (char stroc[3], Unit** units, int
&countUnits){

    Unit* chosenUnit;
    int toX, toY;

        for (int i = 0; i < countUnits; i++){

    if (units[i]->name[0] == stroc[0] && units[i]->name[1] ==
stroc[1]){
                        chosenUnit = units[i];
                        break;
                    }

                    if (i == countUnits-1) return false;
                }

                toX = chosenUnit->getX();
                toY = chosenUnit->getY();
```

```
                switch (stroc[2]) {
                    case 'w':
                        toY--;
                    break;
                    case 'a':
                        toX--;
                    break;
                    case 's':
                        toY++;
                    break;
                    case 'd':
                        toX++;
                    break;

                    default:
                        return false;
                    break;
                }


        if (toX > x || toX < 0 || toY > y || toY < 0) return
false;
        if (f.cellIsUsed(toX, toY)) return false;

        f.replaceUnit(chosenUnit->getX(), chosenUnit->getY(), toX,
toY);
        chosenUnit->setXY(toX, toY);
        return true;
}




    void modifiedGame (){


        do{
            std::cout << "Введите размер поля (не менее 6x6)" <<
std::endl << "Пример: 12 13 (поле 12x13)" <<std::endl;
            std::cin >> x >> y;
```

```
            if (std::cin.fail()) // если предыдущее извлечение
оказалось неудачным,
            {
                std::cin.clear(); // то возвращаем cin в 'обычный'
режим работы
                std::cin.ignore(32767,'\n'); // и удаляем значения
предыдущего ввода из входного буфера
                std::cout << "Неверный ввод!" << std::endl;
                continue;
            }

            if (!(f.createField(x, y))){
                continue;
            }

            f.printField();
            break;
        }while(true);

    }

    void standartGame() {


        bool endGame = true;
        x = 15;
        y = 15;

        f.createField(x, y);
        setUnitsToStart();
        f.printField();

        while (endGame) {


            std::cout << choseUnitToMove (endGame) << std::endl;


            f.printField();
        }
    }

};
```

```
#endif
```