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

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Объектно-ориентированное программирование»
Тема: Создание классов, конструкторов классов, методов классов; наследование

Студент гр. 8303 _____ Удод М.Н.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Научиться создавать классы, их конструкторы и методы. Изучить использование наследования.

Задание.

- 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.

Ход выполнения работы.

1. Был создан класс GameField, использующийся для хранения информации об объектах на поле. Информация хранится в двумерно массиве, без использования контейнеров из stl.
2. Для классы GameField были реализованы методы добавления и удаления объектов на поле, а так же копирование всего поля.

3. Был создан класс Unit, содержащий информацию однотипную информацию для любого объекта в программе.
4. Были созданы классы Armor и Weapon, содержащие информацию о таких характеристиках объекта, как броня и урон.
5. Были созданы различные классы, наследующиеся от класса Armor.
6. Были созданы различные классы, наследующиеся от класса Weapon.
7. Были созданы классы Archer, Infantry и Wizard, являющиеся потомками класса Unit. Эти классы по разному инициализируют поля класса Unit.
8. Были созданы различные классы, наследующиеся от Archer, Wizard и Infantry.

Вывод.

В ходе выполнения лабораторной работы было изучено создание классов путем написание программы с классами, удовлетворяющими условию.

Приложение А. Исходный код программы

1. GameField.h

```
#ifndef UNTITLED13_GAMEFIELD_H
#define UNTITLED13_GAMEFIELD_H

#include "Point.h"
#include "Units/Unit.h"
#include "GameFieldIterator.h"

class Unit;

class GameField {

    friend class Unit;

private:

    Unit ***field;

    int fieldHeight;
    int fieldWidth;

    int maxObjectsCount = 3;

public:

    GameField(int fieldSize);
    GameField(int fieldHeight, int fieldWidth);
    GameField(GameField &other);
    GameField(GameField &&other);

    void deleteObject(int x, int y);
    void deleteObject(Point &point);
    void deleteObject(Unit *object);

    void addObject(Unit *object, int x, int y);

    void moveObject(Point &p1, Point &p2);
    void moveObject(Unit *object, Point &p2);

    Unit ***getField(){ return field; }

    int getHeight();
    int getWidth();
    Unit *getObject(Point &p);

    GameFieldIterator begin(){ return GameFieldIterator(Point(0, 0),
field, fieldHeight, fieldWidth); }
    GameFieldIterator end(){ return GameFieldIterator(Point(0,
fieldHeight), field, fieldHeight, fieldWidth); }

};

#endif //UNTITLED13_GAMEFIELD_H
```

2. GameField.cpp

```
#include <iostream>
#include "GameField.h"

GameField::GameField(int fieldSize){

    field = new Unit** [fieldSize];
    for (int i=0; i<fieldSize; i++){
        field[i] = new Unit* [fieldSize];
        for (int j=0; j<fieldSize; j++){
            field[i][j] = nullptr;
        }
    }

    fieldHeight = fieldSize;
    fieldWidth = fieldSize;

}

GameField::GameField(int fieldHeight, int fieldWidth) {

    field = new Unit** [fieldHeight];
    for (int i=0; i<fieldHeight; i++){
        field[i] = new Unit* [fieldWidth];
        for (int j=0; j<fieldWidth; j++){
            field[i][j] = nullptr;
        }
    }

    this->fieldHeight = fieldHeight;
    this->fieldWidth = fieldWidth;

}

void GameField::deleteObject(int x, int y) {

    if (field[y][x]) {
        maxObjectsCount++;
        delete field[y][x];
        field[y][x] = nullptr;
    }

}

void GameField::addObject(Unit *object, int x, int y) {

    bool isInBorder = x < fieldWidth && y < fieldHeight && x >= 0 && y >=
0;

    if (isInBorder && !field[y][x] && maxObjectsCount && !object-
>isOnField){

        field[y][x] = object;
        maxObjectsCount--;
        object->position = Point(x, y);

    } else{
```

```
        std::cout << "Impossible to add Object " << object << " to
field." << std::endl;

    }
}

void GameField::deleteObject(Unit *object) {

    deleteObject(object->position.x, object->position.y);

}

void GameField::moveObject(Point &p1, Point &p2) {

    if (field[p1.y][p1.x] && !field[p2.y][p2.x]){

        field[p2.y][p2.x] = field[p1.y][p1.x];
        field[p2.y][p2.x]->position = p2;

        field[p1.y][p1.x] = nullptr;

    } else{

        std::cout << "Impossible to move object. Coord " << p1.x << ' '
<< p2.y << " is empty";

    }

}

void GameField::moveObject(Unit *object, Point &p2) {

    Point p1 = object->getPosition();
    moveObject(p1, p2);

}

GameField::GameField(GameField &other) {

    fieldHeight = other.fieldHeight;
    fieldWidth = other.fieldWidth;
    maxObjectsCount = other.maxObjectsCount;

    field = new Unit** [fieldHeight];
    for (int i=0; i<fieldHeight; i++){
        field[i] = new Unit* [fieldWidth];
        for (int j=0; j<fieldWidth; j++){
            if (other.field[i][j])
                field[i][j] = new Unit(other.field[i][j]);
            else
                field[i][j] = nullptr;
        }
    }

}
```

```
void GameField::deleteObject(Point &point) {

    deleteObject(point.x, point.y);

}

int GameField::getHeight() {
    return fieldHeight;
}

int GameField::getWidth() {
    return fieldWidth;
}

Unit *GameField::getObject(Point &p){

    if (p.x < fieldWidth && p.y < fieldHeight)
        return field[p.y][p.x];
    return nullptr;

}

GameField::GameField(GameField &&other) {

    field = other.field;
    fieldHeight = other.fieldHeight;
    fieldWidth = other.fieldWidth;
    maxObjectsCount = other.maxObjectsCount;

}
```

3. Unit.h

```
#ifndef UNTITLED13_UNIT_H
#define UNTITLED13_UNIT_H


#include "../Armor/Armor.h"
#include "../Weapon/Weapon.h"
#include "../Point.h"

class Unit {

    friend class GameField;

protected:

    Point position;

    Armor armor;
    Weapon weapon;
    int health;

    bool isOnField = false;

public:

    Unit(Unit *other);
    Unit() {}
```

```
    Point& getPosition() { return position; }

};


#endif //UNTITLED13_UNIT_H
```

4. Main.cpp

```
#include <iostream>
#include "GameField.h"
#include "Units/Archer/CrossBowMan.h"
#include "Units/Factory/ArcherFactory.h"
#include "Units/Factory/WizardFactory.h"
#include "Units/Factory/InfantryFactory.h"

void example1(){

    GameField field(3, 3);

    CrossBowMan *crossBowMan = ArcherFactory().createWeack();

    field.addObject(crossBowMan, 0, 0);

    for (auto & it : field) {
        std::cout << (int*)it << std::endl;
    }

    Point nextPoint(2, 2);
    field.moveObject(crossBowMan, nextPoint);

    std::cout << std::endl;
    for (auto & obj : field) {
        std::cout << (int*)obj << std::endl;
    }

    field.deleteObject(crossBowMan);

    std::cout << std::endl;
    for (auto & obj : field) {
        std::cout << (int*)obj << std::endl;
    }

}

void example2(){

    FireMage *mage = WizardFactory().createWeack();
    SwordMan *swordMan = InfantryFactory().createStrong();

    GameField fieldOne(2, 3);

    fieldOne.addObject(mage, 2, 2);
    fieldOne.addObject(mage, 2, 0);
    fieldOne.addObject(mage, 1, 2);

    fieldOne.addObject(swordMan, 1, 1);
```

```
    std::cout << std::endl;
    for (auto & obj : fieldOne) {
        std::cout << (int*)obj << std::endl;
    }

    GameField fieldTwo(fieldOne);
    std::cout << std::endl;
    for (auto & obj : fieldTwo) {
        std::cout << (int*)obj << std::endl;
    }

}

void example3(){

    BlockBowMan *blockBowMan = ArcherFactory().createStrong();
    FireMage *fireMage = WizardFactory().createWeack();
    SpearMan *spearMan = InfantryFactory().createWeack();
    SwordMan *swordMan = InfantryFactory().createStrong();

    GameField field(3);

    field.addObject(fireMage, 0, 0);
    field.addObject(blockBowMan, 0, 1);
    field.addObject(spearMan, 0, 2);
    field.addObject(swordMan, 1, 0);

    std::cout << std::endl;
    for (auto & obj : field) {
        std::cout << (int*)obj << std::endl;
    }

    Point p1(1, 0);
    Point p2(2, 0);
    field.moveObject(fireMage, p1);
    field.moveObject(fireMage, p2);

    std::cout << std::endl;
    for (auto & obj : field) {
        std::cout << (int*)obj << std::endl;
    }

}

int main() {

    example3();

    return 0;
}
```