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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Объектно-ориентированное программирование»
Тема: Интерфейсы классов, взаимодействие классов, перезагрузка операций

Студент гр. 8303 _____ Кибардин А.Б.

Преподаватель _____ Филатов Ан.Ю.

Санкт-Петербург
2020

Цель работы.

Разработать и реализовать набор интерфейсов классов, получить навыки в работе с ними и перегрузкой операций.

Задание.

Разработать и реализовать набор классов:

- Класс базы
- Набор классов ландшафта карты
- Набор классов нейтральных объектов поля

Класс базы должен отвечать за создание юнитов, а также учитывать юнитов, относящихся к текущей базе. Основные требования к классу база:

- База должна размещаться на поле
- Методы для создания юнитов
- Учет юнитов, и реакция на их уничтожение и создание
- База должна обладать характеристиками такими, как здоровье, максимальное количество юнитов, которые могут быть одновременно созданы на базе, и.т.д.

Набор классов ландшафта определяют вид поля. Основные требования к классам ландшафта:

- Должно быть создано минимум 3 типа ландшафта
- Все классы ландшафта должны иметь как минимум один интерфейс
- Ландшафт должен влиять на юнитов (например, возможно пройти по клетке с определенным ландшафтом или запрет для атаки определенного типа юнитов)
- На каждой клетке поля должен быть определенный тип ландшафта

Набор классов нейтральных объектов представляют объекты, располагаемые на поле и с которыми могут взаимодействие юнитов. Основные требования к классам нейтральных объектов поля:

- Создано не менее 4 типов нейтральных объектов
- Взаимодействие юнитов с нейтральными объектами, должно быть реализовано в виде перегрузки операций

Классы нейтральных объектов должны иметь как минимум один общий интерфейс

Ход работы.

1) Разработан и реализован класс базы Base (файлы Base.h и Base.cpp).

Класс хранит текущий список юнитов, здоровье, счетчик юнитов, максимальное число юнитов, координаты на поле и указатель на поле, к которому привязана база. Является подписчиком создаваемых юнитов через наследование от класса Observer. Для работы с базой созданы методы класса:

Base(Field*, int, int, int) - конструктор класса, для создания объекта с заданными пользователем координатами базы, ее максимальным здоровьем и максимальным количеством юнитов, одновременно созданных на базе.

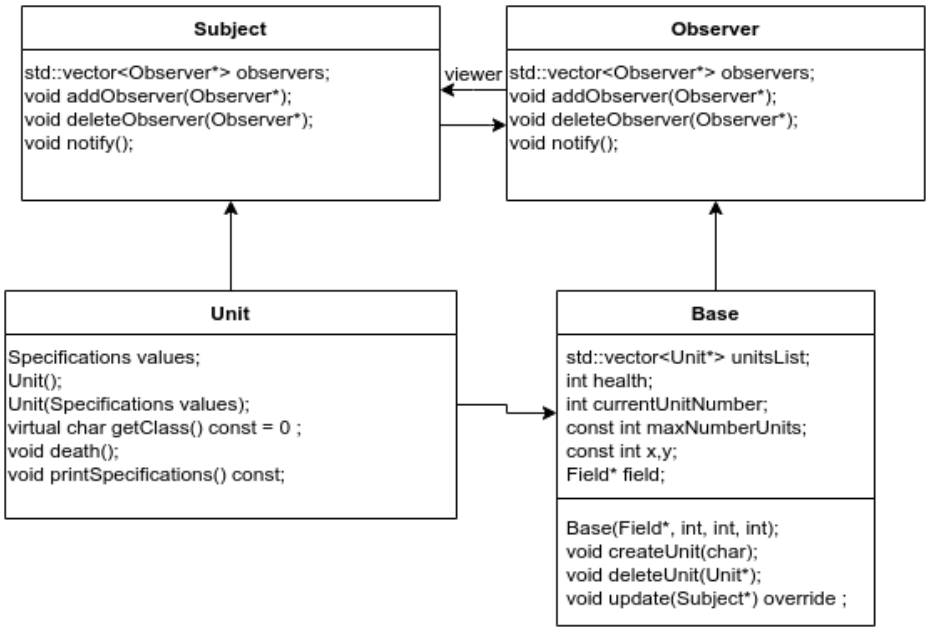
Void createUnit(char) — метод создания юнита определенного класса с помощью паттерна абстрактная фабрика.

Void deleteUnit(Unit*) - метод удаления юнита с базы.

Void update(Subject*) override — метод для получения обновления (его уничтожении) от объекта Subject.

2) Для наблюдения над юнитами в классе базы используется паттерн Наблюдатель(файлы Observer.h, Subject.h Subject.cpp). Класс юнитов наследуется от класса Subject, класс базы наследуется от класса Obserever.

При создании каждого нового юнита, база подписывается на него. При смерти юнита база получает обновление и удаляет юнита с поля и из списка юнитов.



3) Создан набор классов ландшафтов (файлы Landscape.cpp и Landscape.h).

Класс Landscape предоставляет собой виртуальный класс для создания общего интерфейса для новых типов ландафтов

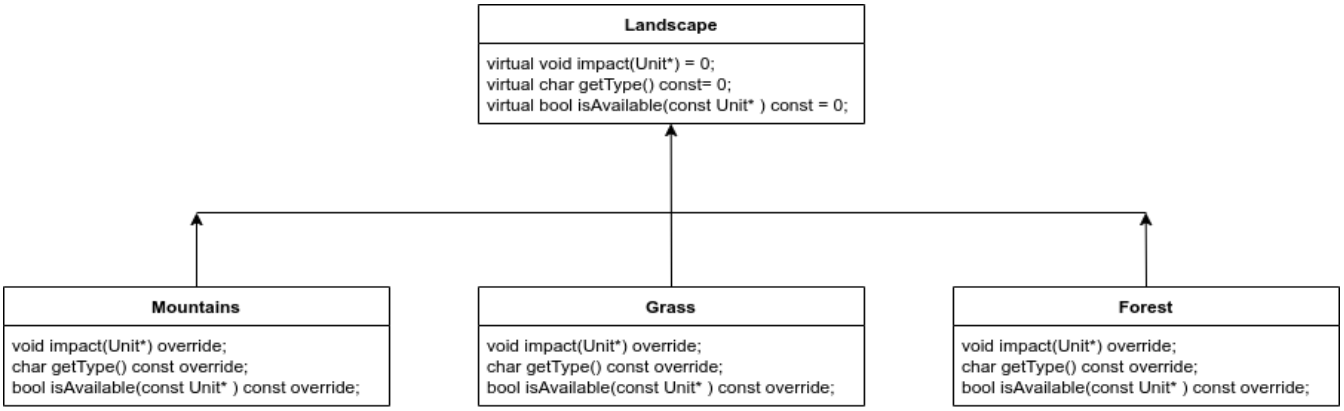
Методы класса Landscape :

virtual int impact(const Unit*) = 0 — метод определяющий влияние на юнита.

Virtual char getType() const = 0 — метод для получения типа ландшафта.

Virtual bool isAvaible() const = 0 — метод определяющий возможность прохождения юнита определенного класса по данному ландшафту.

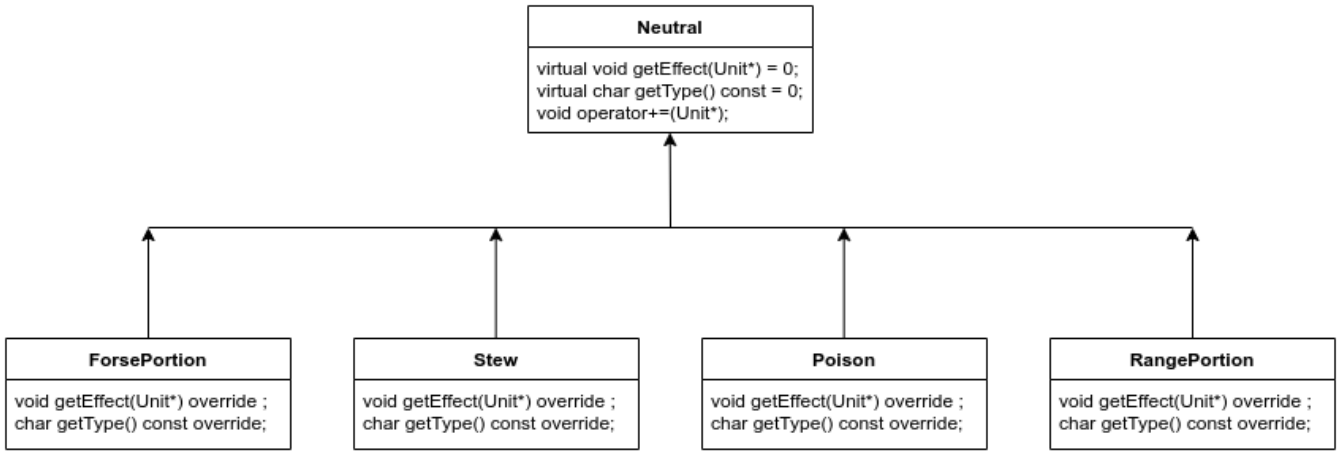
Классы Mountains, Grass, Forest наследуются от класса Landscape и описывают выше перечисленные методы.



4) Создан класс Neutral, реализующий общий интерфейс, и набор классов (forsePortion, Stew, Poison, rangePortion), наследуемых от него и описывающих интерфейс(файлы Neutral.cpp и Neutral.h). Методы для взаимодействия нейтральных объектов с юнитами реализовано в виде перегрузки операции +=.

Методы класса Neutral:

-virtual void getEffect(Unit*) = 0 — метод для применения эффекта нейтрального объекта на юнита.



- virtual char getType() const = 0 — метод для получения типа нейтрального объекта.

Выводы.

В ходе выполнения работы были получены навыки по созданию конструкторов, методов классов и их наследованию. Так же был разработаны и реализованы классы базы, набор классов нейтральных объектов и ландшафтов.

Приложение А.

Исходный код.

Файл bard.cpp:

```
#include "Bard.h"
Bard::Bard(){
    health = 125;
    armor = 0;
    attack = 5;
    attackRange = 10;
}
char Bard::getClass() const {
    return 'B';
}
```

Файл bard.h:

```
#ifndef OOP_1_BARD_H
#define OOP_1_BARD_H
#include "Thief.h"
class Bard: public Thief {
public:
    Bard();
    char getClass() const;
};
#endif //OOP_1_BARD_H
```

Файл cell.cpp:

```
#include "Cell.h"
Cell::Cell():Object(nullptr){}
void Cell::clearCell() {
    Object = nullptr;
}
void Cell::setCharacter(Unit* newUnit) {
    if(Object == nullptr)
        Object = newUnit;
}
Unit* Cell::getCharacter() const{
    return this->Object;
}
Cell::Cell(const Cell& obj):Cell()
{
    this->Object = obj.Object;
}
Cell& Cell::operator=(const Cell& obj) {
    if(this == &obj)
        return *this;
```

```
    this->Object = obj.Object;
    return *this;
}
```

Файл cell.h:

```
#ifndef OOP_1_CELL_H
#define OOP_1_CELL_H
#include "Unit.h"
class Unit;
class Cell {
    Unit* Object;
    public:
    Cell();
    Cell(const Cell& obj);
    Cell& operator=(const Cell& obj);
    void clearCell();
    void setCharacter(Unit* Object);
    Unit* getCharacter() const;
};
#endif //OOP_1_CELL_H
```

Файл dwarf.cpp:

```
#include "Dwarf.h"
Dwarf::Dwarf(){
    health = 250;
    armor = 250;
    attack = 40;
    attackRange = 2;
}
char Dwarf::getClass() const {
    return 'D';
}
```

Файл dwarf.h:

```
#ifndef OOP_1_DWARF_H
#define OOP_1_DWARF_H
#include "Warrior.h"
class Dwarf: public Warrior{
public:
    Dwarf();
    char getClass() const;
};
#endif //OOP_1_DWARF_H
```

Файл elf.cpp:

```
#include "Elf.h"
Elf::Elf()  {
    health = 125;
    armor = 50;
    attack = 30;
    attackRange = 2;
}
char Elf::getClass() const {
    return 'E';
}
```

Файл elf.h:

```
#ifndef OOP_1_ELF_H
#define OOP_1_ELF_H
#include "Thief.h"
class Elf: public Thief {
public:
    Elf();
    char getClass() const;
};
#endif //OOP_1_ELF_H
```

Файл field.cpp:

```
#include <iostream>
#include "Field.h"
Field::Field(int x, int y) :X(x), Y(y), matrix(new Cell*[X]),
objectsCounter(0), maxObjectsOnField(x*y){
    for(int i = 0; i < X; i++) {
        matrix[i] = new Cell[Y];
        for(int j = 0; j < Y; j++)
            matrix[i][j].setCharacter(nullptr);
    }
}
Field::Field():Field(0,0) {}
Field::~Field()
{
    if(matrix)
    {
        for(int i = 0; i < X; i++)
            if(matrix + i)
                delete[] matrix[i];
        delete[] matrix;
    }
}
void Field::addObject(Unit *Object, int x, int y) {
    if(!Object->isOnField())
        Object->replace(this);
```

```
    if(checkPoint(x,y) && !matrix[x][y].getCharacter() &&  objectsCounter <
maxObjectsOnField) {
        matrix[x][y].setCharacter(Object);
        Object->setX(x);
        Object->setY(y);
        objectsCounter++;
    }
}
int Field::getObjectCounter() const{
    return objectsCounter;
}
void Field::deleteObject(Unit *Object) {
    int requiredX = Object->getX();
    int requiredY = Object->getY();
    if(checkPoint(requiredX, requiredY) && matrix[requiredX]
[requiredY].getCharacter() == Object) {
        matrix[requiredX][requiredY].clearCell();
    }
    objectsCounter--;
}
void Field::print() const{
    printf("------------------\n");
    for(int i = 0; i < X;i++){
        for(int j = 0; j < Y; j++)
            if(matrix[i][j].getCharacter() != nullptr)
                std::cout << matrix[i][j].getCharacter()->getClass() << "
";
            else
                std::cout << ". ";
        std::cout << std::endl;
    }
    printf("------------------\n");
}
bool Field::checkPoint(int x, int y) const {
    return x < X && y < Y;
}
void Field::swapCharacters(int x1, int y1, int x2, int y2) {
    if(x1 == x2 && y1 == y2)
        return;
    //std::swap(matrix[x1][y1].Object, matrix[x2][y2].Object);
    Unit* tmp = matrix[x1][y1].getCharacter();
    matrix[x1][y1].clearCell();
    matrix[x1][y1].setCharacter(matrix[x2][y2].getCharacter());
    matrix[x2][y2].clearCell();
    matrix[x2][y2].setCharacter(tmp);
}
int Field::getX() const {
    if(matrix != nullptr)
        return X;
    return -1;
}
int Field::getY() const {
    if(matrix != nullptr)
        return Y;
    return -1;
}
```

```
Field::Field(const Field& obj) : Field(obj.X,obj.Y)
{
    objectsCounter = obj.objectsCounter;
    for(int i = 0; i < X; i++)
        for(int j = 0; j < Y; j++)
            this->matrix[X][Y] = obj.matrix[X][Y];
}
Field::Field(Field&& obj):X(0), Y(0), matrix(nullptr){
    std::swap(matrix, obj.matrix);
    std::swap(X, obj.X);
    std::swap(Y, obj.Y);
    std::swap(objectsCounter, obj.objectsCounter);
    std::swap(maxObjectsOnField, obj.maxObjectsOnField);
}
Field& Field::operator=(const Field& obj) {
    if(this == &obj)
        return *this;
    this->matrix = new Cell*[obj.X];
    this->X = obj.X;
    this->Y = obj.Y;
    this->objectsCounter = obj.objectsCounter;
    this->maxObjectsOnField = obj.maxObjectsOnField;
    for(int i = 0; i < X; i++) {
        this->matrix[i] = new Cell[Y];
        for (int j = 0; j < Y; j++) {
            this->matrix[i][j] = obj.matrix[i][j];
        }
    }
    return *this;
}
Field& Field::operator=(Field &&obj) {
    if(this == &obj)
        return *this;
    std::swap(matrix, obj.matrix);
    std::swap(X, obj.X);
    std::swap(Y, obj.Y);
    std::swap(objectsCounter, obj.objectsCounter);
    std::swap(maxObjectsOnField, obj.maxObjectsOnField);
    return *this;
}
```

Файл field.h:

```
#ifndef OOP_1_FIELD_H
#define OOP_1_FIELD_H
#include "Cell.h"
#include "Unit.h"
class Cell;
class Unit;
class Field{
    int X, Y;
    Cell** matrix;
    int objectsCounter;
    int maxObjectsOnField;
```

```
public:
    Field();
    Field(int x, int y);
    ~Field();
    int getX() const;
    int getY() const;
    bool checkPoint(int x, int y) const;
    void swapCharacters(int x1, int y1, int x2, int y2);
    int getObjectCounter() const;
    void addObject(Unit* Object, int x, int y);
    void deleteObject(Unit* Object);
    void print() const;
    Field(const Field& obj); // copy constructor
    Field(Field&& obj); // move constructor
    Field& operator=(const Field& obj);
    Field& operator=(Field&& obj);
};
#endif //OOP_1_FIELD_H
```

Файл flamen.cpp:

```
#include "Flamen.h"
Flamen::Flamen(){
    health = 125;
    armor = 0;
    attack = 15;
    attackRange = 5;
}
char Flamen::getClass() const {
    return 'F';
}
```

Файл flamen.h:

```
#ifndef OOP_1_FLAMEN_H
#define OOP_1_FLAMEN_H
#include "Wizard.h"
class Flamen: public Wizard {
public:
    Flamen();
    char getClass() const;
};
#endif //OOP_1_FLAMEN_H
```

Файл main.cpp:

```
#include <iostream>
#include "Field.h"
#include "Dwarf.h"
```

```
#include "Paladin.h"
#include "Elf.h"
#include "Bard.h"
#include "Flamen.h"
#include "Necromancer.h"
int main() {
    puts("first example");
    puts("created first field");
    Field gameField(5,8);
    gameField.print();
    printf("add dwarf on field\n");
    Dwarf zoltan;
    gameField.addObject(&zoltan, 2, 2);
    gameField.print();
    printf("move dwarf\n");
    zoltan.move(3,5);
    gameField.print();
    printf("delete dwarf\n");
    zoltan.deleteFromField();
    gameField.print();
    puts("\nsecond example");
    Elf legolas;
    puts("add elf on field");
    gameField.addObject(&legolas, 1, 1);
    gameField.print();
    puts("move first field to second");
    Field newField(std::move(gameField));
    puts("second field");
    newField.print();
    puts("first field");
    gameField.print();
    puts("\nthird example");
    puts("create third field");
    Field third(5,5);
    puts("copy third field to first");
    gameField = third;
    puts("third field");
    third.print();
    puts("first field");
    gameField.print();
    puts("add dwarf to third field");
    third.addObject(&zoltan, 1, 1);
    third.print();
    puts("first field");
    gameField.print();
    return 0;
}
```

Файл necromancer.cpp:

```
#include "Necromancer.h"
Necromancer::Necromancer(){
    health = 100;
    armor = 25;
```

```
    attack = 10;
    attackRange = 5;
}
char Necromancer::getClass() const {
    return 'N';
}
```

Файл necromancer.h:

```
#ifndef OOP_1_NECROMANCER_H
#define OOP_1_NECROMANCER_H
#include "Wizard.h"
class Necromancer: public Wizard {
public:
    Necromancer();
    char getClass() const;
};
#endif //OOP_1_NECROMANCER_H
```

Файл paladin.cpp:

```
#include "Paladin.h"
Paladin::Paladin(){
    health = 300;
    armor = 150;
    attack = 35;
    attackRange = 2;
}
char Paladin::getClass() const {
    return 'P';
}
```

Файл paladin.h:

```
#ifndef OOP_1_PALADIN_H
#define OOP_1_PALADIN_H
#include "Warrior.h"
class Paladin: public Warrior{
public:
    Paladin();
    char getClass() const;
};
#endif //OOP_1_PALADIN_H
```

Файл thief.cpp:

```
#include "Thief.h"
```

Файл thief.h:

```
#ifndef OOP_1_THIEF_H
#define OOP_1_THIEF_H
#include "Unit.h"
class Thief: public Unit{
    friend class Bard;
    friend class Elf;
};
#endif //OOP_1_THIEF_H
```

Файл unit.cpp:

```
#include <cstdio>
#include "Unit.h"
Unit::Unit():health(200), armor(200), attack(20), attackRange(2),
gameField(nullptr)
{}
int Unit::getX() const {
    if(gameField != nullptr)
        return this->x;
    return -1;
}
int Unit::getY() const {
    if(gameField != nullptr)
        return this->y;
    return -1;
}
void Unit::setX(int x) {
    this->x = x;
}
void Unit::setY(int y) {
    this->y = y;
}
void Unit::move(int X, int Y){
    if(gameField != nullptr && gameField->checkPoint(X, Y))
    {
        printf("start coord:%d %d\n", this->x, this->y);
        gameField->swapCharacters(this->x,this->y, X, Y);
        this->y = Y;
        this->x = X;
    }
}
bool Unit::isOnField() {
    return gameField != nullptr;
}
 void Unit::deleteFromField() {
    gameField->deleteObject(this);
    gameField = nullptr;
}
void Unit::replace(Field *gameField) {
    this->gameField = gameField;
```

```
}
```

Файл unit.h:

```
#ifndef OOP_1_UNIT_H
#define OOP_1_UNIT_H
#include "Cell.h"
#include "Field.h"
class Field;
class Cell;
class Unit {
    int health;
    int armor;
    int attack;
    int attackRange;
    Field* gameField;
    int x;
    int y;
public:
    Unit();
    bool isOnField();
    int getX() const;
    int getY() const;
    void setX(int x);
    void setY(int y);
    void move(int X, int Y);
    void replace(Field* gameField);
    virtual char getClass() const = 0 ;
    void deleteFromField();
    friend class Wizard;
    friend class Warrior;
    friend class Thief;
    friend class Elf;
    friend class Bard;
    friend class Dwarf;
    friend class Paladin;
    friend class Flamen;
    friend class Necromancer;
    };
#endif //OOP_1_UNIT_H
```

Файл warrior.cpp:

```
#include "Warrior.h"
```

Файл warrior.h:

```
#ifndef OOP_1_WARRIOR_H
#define OOP_1_WARRIOR_H
#include "Unit.h"
```

```
class Warrior: public Unit {
public:
    friend class Dwarf;
    friend class Paladin;
};
#endif //OOP_1_WARRIOR_H
```

Файл wizard.cpp:

```
#include "Wizard.h"
char Wizard::getClass() const {
    return 'W';
}
```

Файл wizard.h:

```
#ifndef OOP_1_WIZARD_H
#define OOP_1_WIZARD_H
#include "Unit.h"
class Wizard: public Unit {
public:
    char getClass() const;
    friend class Flamen;
    friend class Necromancer;
};
#endif //OOP_1_WIZARD_H
```