

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

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Объектно-ориентированное программирование»
Тема: Логическое разделение классов

Студент гр. 8303

Удод М.Н.

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Научиться логическому разделению классов

Задание.

Разработать и реализовать набора классов для взаимодействия пользователя с юнитами и базой. Основные требования:

- Должен быть реализован функционал управления юнитами
- Должен быть реализован функционал управления базой

Ход выполнения работы.

1. Были созданы классы Command и CommandHandler для обработки команд, введенных пользователем.
2. Был создан набор классов, реализующий различные команды пользователя. Все они наследуются от класса Command.
3. Для каждого класса команды был создан соответствующий класс для ее создания из строки, введенной пользователем. Такие классы наследуются от CommandHandler и реализуют паттерн «Цепочка обязанностей»
4. Был создан класс GameFacade, предоставляющий интерфейс для управления игрой.
5. Был реализован класс BaseObserver для взаимодействия поля и юнитов. Так, при создании юнита, сигнал получает база, обрабатывает его, и передает полю. Таким образом реализуется паттерн «Посредник».

Примеры.

```
Now player: 0
|. | |. |
|. | |. |
|. | |. |

create base 0 0
Now player: 1
|B| |. |
|. | |. |
|. | |. |

show base 0 0
Base info:
  HP: 100
  Armor: Armor( Damage Absorption: 3 )
  Max Objects Count: 5
Now player: 0
|B| |. |
|. | |. |
|. | |. |
```

```
Now player: 0
|. | |. |
|. | |. |
|. | |. |

create base 0 0
Now player: 1
|B| |. |
|. | |. |
|. | |. |

create base 2 2
Now player: 0
|B| |. | |
|. | |. |
|. | |. |B|

create unit 0 1 1
Now player: 1
|B| |. | |
|W| |. |
|. | |. |B|
```

```
show unit 0 1
Unit info:
  HP: 100
  Weapon: Weapon( Damage: 10 )
  Armor: Armor( Damage Absorption: 1 )
```

```
Now player: 0
|. | |. |
|. | |. |
|. | |. |

create base 0 0
Now player: 1
|B| |. |
|. | |. |
|. | |. |

create base 2 2
Now player: 0
|B| |. | |
|. | |. |
|. | |. |B|

create unit 1 0 1
Now player: 1
|B| |W| |. |
|. | |. |
|. | |. |B|

move unit 1 0 0 1
Base: Unit 0x562338151cc0 moving
Now player: 0
|B| |. | |
|W| |. |
|. | |. |B|
```

Вывод.

В ходе выполнения лабораторной работы было изучено логическое разделения классов с помощью реализации таких паттернов, как «Команда», «Посредник», «Фасад» и «Цепочка обязанностей».

Приложение А. Исходный код программы

1. GameField.h

```
#ifndef UNTITLED13_GAMEFIELD_H
#define UNTITLED13_GAMEFIELD_H

#include "../Point.h"
#include "../Objects/GameObject.h"
#include "GameFieldIterator.h"
#include "FieldCell.h"

class Unit;

class GameField {
private:
    static GameField *gameField;

    FieldCell **field;

    int fieldHeight;
    int fieldWidth;

    GameField();
    explicit GameField(int fieldSize);
    GameField(int fieldHeight, int fieldWidth);
    GameField(const GameField &other) = delete;
    GameField(GameField &&other) = delete;
    ~GameField();

public:
    static void init( int fieldHeight, int fieldWidth);
    static GameField *getInstance();

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

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

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

    int getHeight() const;
    int getWidth() const;
    FieldCell *getCell(const Point &p) const;

    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::gameField = nullptr;

GameField::GameField():
    fieldHeight(0),
    fieldWidth(0),
    field(nullptr)
{}

GameField::GameField(int fieldSize):
    fieldHeight(fieldSize),
    fieldWidth(fieldSize)
{
    field = new FieldCell* [fieldSize];
    for (int i=0; i<fieldSize; i++){
        field[i] = new FieldCell [fieldSize];
    }
}

GameField::GameField(int fieldHeight, int fieldWidth):
    fieldHeight(fieldHeight),
    fieldWidth(fieldWidth)
{
    field = new FieldCell* [fieldHeight];
    for (int i=0; i<fieldHeight; i++){
        field[i] = new FieldCell [fieldWidth];
    }
}

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

    delete field[y][x].getObject();
    field[y][x].eraseObject();

}

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

    if (object->isOnField){
        std::cout << "GameField: Object " << object << " already on field." <<
std::endl;
        return;
    }

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

```

        if (isInBorder && field[y][x].isEmpty()){

            field[y][x].setObject(object);
            object->position = Point(x, y);
            object->isOnField = true;

        } else{

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

        }
    }

void GameField::deleteObject(GameObject *object) {

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

}

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

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

        field[p2.y][p2.x] = std::move(field[p1.y][p1.x]);
        field[p2.y][p2.x].getObject()->position = p2;

        field[p1.y][p1.x].eraseObject();

    } else{

        std::cout << "GameField: Impossible to move object." << std::endl;

    }

}

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

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

}

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

    deleteObject(point.x, point.y);

}

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

int GameField::getWidth() const{

```

```

        return fieldWidth;
    }

    FieldCell *GameField::getCell(const Point &p) const{

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

    GameField::~GameField() {

        for (int i=0; i<fieldHeight; i++){
            delete []field[i];
        }

        delete []field;
    }

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

        if (!gameField){

            gameField = new GameField(fieldHeight, fieldWidth);

        } else{

            std::cout << "Game field already initialized" << std::endl;

        }

    }

    GameField *GameField::getInstance() {
        if (gameField){
            return gameField;
        } else{
            std::cout << "Game field is not initialized" << std::endl;
            return nullptr;
        }
    }
}

```

3. Main.cpp

```

#include <iostream>
#include "GameField/GameField.h"
#include "Objects/Units/Archer/CrossBowMan.h"
#include "Objects/Base.h"
#include "Landscapes/Ocean.h"
#include "Objects/Neutrals/House.h"
#include "Landscapes/Desert.h"
#include "Landscapes/ZeroSpace.h"
#include "Objects/Units/Archer/BlockBowMan.h"

void example1(){

    GameField::init(3, 3);

```



```

Base *base = new Base();
GameField::getInstance()->addObject(base, 0, 0);

FireMage *fireMage1 = base->createUnit<FireMage>(Point(0, 1));
FireMage *fireMage2 = base->createUnit<FireMage>(Point(0, 2));

std::cout << "FireMage1 Health: " << fireMage1->getHealth() << std::endl;
std::cout << "FireMage2 Health: " << fireMage2->getHealth() << std::endl;

fireMage1->attack(*fireMage2);

std::cout << "FireMage1 Health: " << fireMage1->getHealth() << std::endl;
std::cout << "FireMage2 Health: " << fireMage2->getHealth() << std::endl;

std::cout << std::endl;
for (auto obj : *GameField::getInstance()) {
    std::cout << obj << std::endl;
}
std::cout << std::endl;

GameField::getInstance()->getCell(Point(1, 0))->setLandscape(new
ZeroSpace());
fireMage1->move(Point(1, 0));

std::cout << "FireMage1 Health: " << fireMage1->getHealth() << std::endl;
std::cout << "FireMage2 Health: " << fireMage2->getHealth() << std::endl;

fireMage1->attack(*fireMage2);

std::cout << "FireMage1 Health: " << fireMage1->getHealth() << std::endl;
}

void example2(){

    GameField::init(3, 3);

    Base *base = new Base();
    GameField::getInstance()->addObject(base, 0, 0);

    House *house = new House();
    GameField::getInstance()->addObject(house, 1, 0);

    BlockBowMan *archer = base->createUnit<BlockBowMan>(Point(0, 1));
    std::cout << "BlockBowMan Health: " << archer->getHealth() << std::endl;

    archer->move(Point(1, 0));
    std::cout << "BlockBowMan Health: " << archer->getHealth() << std::endl;

    std::cout << std::endl;
    for (auto obj : *GameField::getInstance()) {
        std::cout << obj << std::endl;
    }
    std::cout << std::endl;
}

```

```

}

void example3(){

    GameField::init(3, 3);

    Base *base = new Base();
    GameField::getInstance()->addObject(base, 0, 0);

    auto *fireMage = base->createUnit<FireMage>(Point(1, 0));

    GameField::getInstance()->getCell(Point(0, 1))->setLandscape(new Ocean());
    fireMage->move(Point(0, 1));

    auto *house = new House();
    GameField::getInstance()->getCell(Point(1, 1))->setObject(house);
    fireMage->move(Point(1, 1));

    std::cout << std::endl;
    for (auto obj : *GameField::getInstance()) {
        std::cout << obj << std::endl;
    }
}

int main() {

    example2();

    return 0;
}

```

4. Файл Base.h

```

#ifndef UNTITLED13_BASE_H
#define UNTITLED13_BASE_H

#include "../Armor/Armor.h"
#include "GameObject.h"
#include "Units/Wizard/FireMage.h"
#include "Units/Wizard/Voldemort.h"
#include <vector>
#include "../GameField//GameField.h"
#include "../UnitObserver.h"
#include <iostream>
#include <algorithm>

class Base: public GameObject, public UnitObserver {

public:

    Base(): GameObject(ObjectType::BASE) {}
    bool addUnit(Unit *unit, Point position);

    template <typename T>
    T *createUnit(Point position);

```

```

void onUnitAttack(Unit *unit) override;
void onUnitMove(Unit *unit) override;
void onUnitDestroy(Unit *unit) override;
void onUnitDamaged(Unit *unit) override;
void onUnitHeal(Unit *unit) override;

private:

    std::vector<Unit*> units;

    int health;
    int maxObjectsCount = 5;
    Armor armor;

};

template<typename T>
T *Base::createUnit(Point position) {
    if (units.size() < maxObjectsCount) {
        T *unit = new T();
        units.push_back(unit);
        unit->addObserver(this);
        GameField::getInstance()->addObject(unit, position.x, position.y);

        maxObjectsCount++;
        return unit;
    } else{

        std::cout << "Base: Cannot create unit. Limit is exceeded." <<
std::endl;

    }
}

#endif //UNTITLED13_BASE_H

```

5. Файл Base.cpp

```

#include "Base.h"

bool Base::addUnit(Unit *unit, Point position) {
    if (units.size() < maxObjectsCount){

        units.push_back(unit);
        GameField::getInstance()->addObject(unit, position.x, position.y);
        maxObjectsCount++;
        return true;

    } else{

        return false;

    }
}

```

```

void Base::onUnitAttack(Unit *unit) {
    std::cout << "Base: Unit " << unit << " attack" << std::endl;
}

void Base::onUnitMove(Unit *unit) {

    std::cout << "Base: Unit " << unit << " moving" << std::endl;

}

void Base::onUnitDestroy(Unit *unit) {

    auto position = std::find(units.begin(), units.end(), unit);
    if (position != units.end()) {
        GameField::getInstance()->deleteObject(unit);
        units.erase(position); // Тут уже и удаление юнита.
        std::cout << "Base: Unit " << unit << " destroyed" << std::endl;
    } else{
        std::cout << "Called observer of base for unit don't belong to it" <<
std::endl;
    }

}

void Base::onUnitDamaged(Unit *unit) {

    std::cout << "Base: Unit " << unit << " damaged" << std::endl;

}

void Base::onUnitHeal(Unit *unit) {

    std::cout << "Base: Unit " << unit << " healed" << std::endl;

}

```

6. Файл Landscape.h

```

#ifndef UNTITLED13_LANDSCAPE_H
#define UNTITLED13_LANDSCAPE_H

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

class Landscape {

public:

    virtual int getDamageFactor(WeaponType type) = 0;
    virtual int getAbsorptionFactor(ArmorType type) = 0;

};

```

```
#endif //UNTITLED13_LANDSCAPE_H
```

7. Файл NeutralObject.h

```
#ifndef UNTITLED13_NEUTRALOBJECT_H  
#define UNTITLED13_NEUTRALOBJECT_H
```

```
#include "../GameObject.h"
```

```
#include "NeutralObjectStrategy/NeutralObjectStrategy.h"
```

```
class Unit;
```

```
class NeutralObject: public GameObject {
```

```
protected:
```

```
    NeutralObjectStrategy *strategy;
```

```
public:
```

```
    NeutralObject(): GameObject(ObjectType::NEUTRAL_OBJECT){}
```

```
    void setStrategy(NeutralObjectStrategy *strategy) { this->strategy =  
strategy; }
```

```
    virtual void applyTo(Unit &unit)=0;
```

```
    virtual ~NeutralObject(){
```

```
        delete strategy;
```

```
    }
```

```
};
```

```
#endif //UNTITLED13_NEUTRALOBJECT_H
```