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

ОТЧЕТ

по лабораторной работе № 2

по дисциплине «Объектно-ориентированное программирование»

Тема: «Полиморфизм»

Студент гр. 3343 _____ Жаворонок Д.Н.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2024

Цель работы

Изучить работу классов-интерфейсов, путём усовершенствования программы из предыдущей лабораторной работы. Необходимо создать: класс-интерфейс способности, класс менеджера-способностей и набор классов-исключений для обработки исключительных ситуаций.

Задание

Создать класс-интерфейс способности, которую игрок может применять. Через наследование создать 3 разные способности:

- Двойной урон – следующая атака при попадании по кораблю нанесёт сразу 2 урона (уничтожит сегмент);
- Сканер – позволяет проверить участок поля 2x2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус;
- Обстрел – наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.

Создать класс менеджер-способностей. Который хранит очередь способностей, изначально игроку доступно по 1 способности в случайном порядке. Реализовать метод применения способности.

Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.

Реализуйте набор классов-исключений и их обработку для следующих ситуаций (можно добавить собственные):

- Попытка применить способность, когда их нет;
- Размещение корабля вплотную или на пересечении с другим кораблём;
- Атака за границы поля.

Примечания:

- Интерфейс события должен быть унифицирован, чтобы их можно было единообразно использовать через интерфейс;
- Не должно быть явных проверок на тип данных.

Выполнение работы

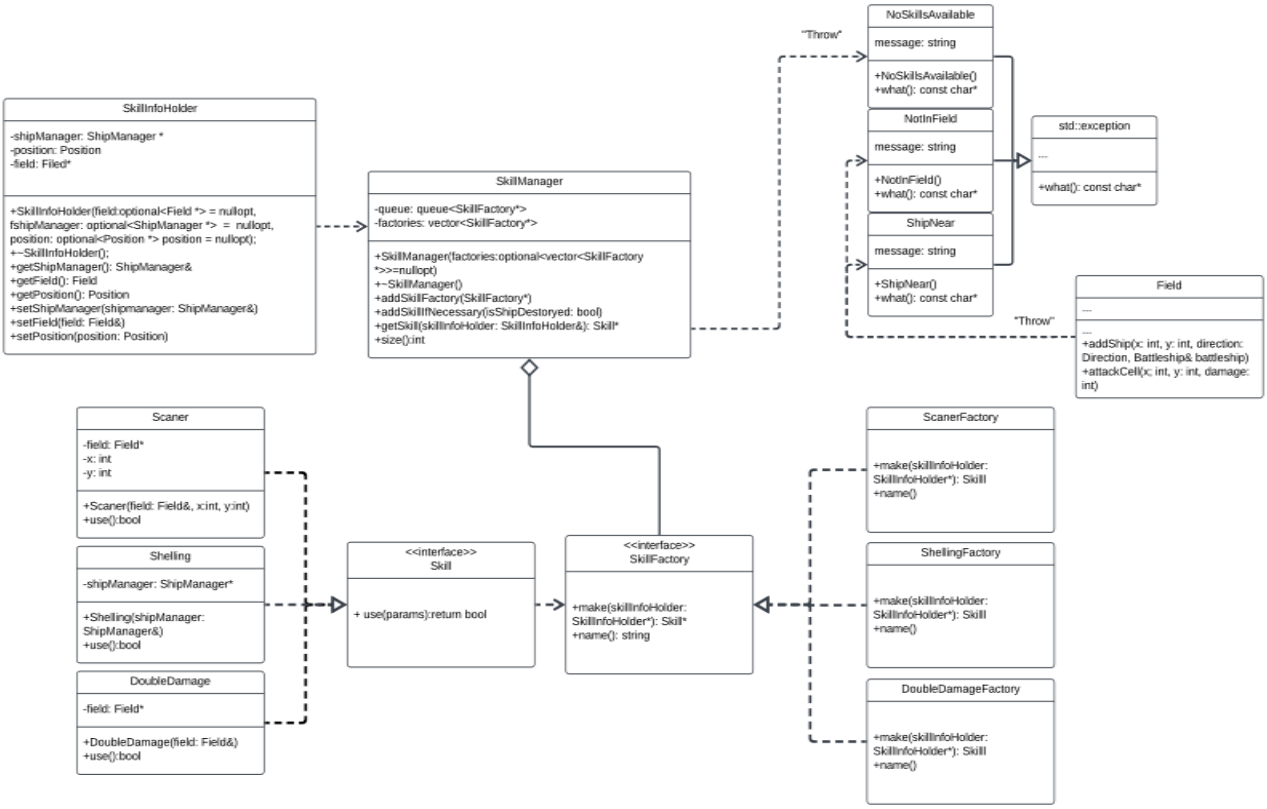


Рисунок 1 – UML-диаграмма классов

Код программы содержит реализацию классов: *Skill, Scaner, DoubleDamage, Shelling, SkillFactory, ScanerFactory, DoubleDamageFactory, ShellingFactory, SkillManager, SkillInfoHolder,* а также классы исключений.

Классы *Skill*, *DoubleDamage*, *Scanner*, *Shelling* и *SkillManager* были добавлены согласно заданию.

Классы *SkillFactory, ScanerFactory, DoubleDamageFactory, BombardmentFactory, SkillManager* были созданы, чтобы реализовать фабричный метод – паттерн, определяющий класс-интерфейс для создания объектов, при этом оставляющий своим подклассам решение, какой класс создавать.

Помимо обозначенных классов, реализованы и интегрированы в код 3 классов-исключений для обработки различных исключительных случаев (применение способности при её отсутствии, выход за границы поля, неправильная расстановка корабля).

Skill является классом-интерфейсом для способностей. Он имеет следующие виртуальные методы:

- *virtual bool use() = 0* – виртуальный метод для применения способности.

Класс *DoubleDamage* является реализацией способности двойного урона. Он имеет следующие поля:

- *Field* field* – указатель на поле.

  и следующие методы:

- *DoubleDamage(Field& field)* – конструктор класса.
- *bool use() override* – использует метод поля *useDoubleDamage,* устанавливающий флаг *useDoubleDamageFlag = true.* После чего очередная атака в поле будет использовать урон равный 2 и переведет флаг в положение *false.*

Класс *Scaner* является реализацией способности сканера, который смотрит участок поля 2x2 на наличие кораблей в нём. Он имеет следующие поля:

- *Field* field* – ссылка на поле.
- *int x , y* – координаты для применения способности.

  И следующие методы:

- *Scaner(Field& field, Pos& coordinate)* – конструктор класса.
- *bool use() override* – производится сканирование поля в области 2x2 по координатам из поля класса, где координаты обозначают левый верхний угол.

Класс *Shelling* является реализацией способности случайного выстрела по случайному сегменту корабля. Он имеет следующие поля:

- *ShipManager* shipManager* – указатель на менеджер кораблей.

  И следующие методы:

- *Shelling(Shipmanager&* shipManager*)* – конструктор класса.

- *bool use() override* – производится выстрел по случайному сегменту случайного корабля не изменяя состояние клетки(способность в любом случае попадёт не по уничтоженному сегменту корабля).

Класс *SkillFactory* является классом-интерфейсом для классов-фабрик способностей. Он имеет следующие виртуальные методы:

- *virtual Skill* make(SkillInfoHolder& skillInfoHolder) = 0* – виртуальный метод, необходимый для создание объекта одной из способностей.

Класс *DoubleDamageFactory* является реализацией создателя способности двойного урона. Он имеет следующие методы:

- *DoubleDamageFactory()* – конструктор класса.
- *Skill* make(SkillInfoHolder& skillInfoHolder) override* – создаёт объект класса *DoubleDamage* и возвращает его указатель.

Класс *ScannerFactory* является реализацией создателя способности сканера. Он имеет следующие методы:

- *ScannerFactory()* – конструктор класса.
- *Skill* make(SkillInfoHolder& skillnfoHolder) override* – создаёт объект класса *Scanner* и возвращает его указатель

Класс *ShellingFactory* является реализацией создателя способности сканера. Он имеет следующие методы:

- *ShellingFactory()* – конструктор класса.
- *Skill* make(SkillInfoHolder& skillInfoHolder) override* – создаёт объект класса Shelling и возвращает его указатель.

Класс *SkillManager* отвечает за контроль над способностями, он хранит в очереди названия способностей, которые используются для создателей способностей. Он имеет следующие поля:

- *queue<SkillFactory*> queue* – очередь имён способностей.

- *vector<SkillFactory*> factories* - вектор возможных способностей из которых при добавлении новой способности после уничтожения корабля рандомом добавляется в queue

И следующие методы:

- *SkillManager(std::optional<std::vector<SkillFactory*>> factories = std::nullopt))* – конструктор класса.
- *int size()* – возвращает размер очереди.
- *void addSkillFactory(SkillFactory*)* – добавляет новую способность в очередь.
- *void addSkillIfNeccessary(bool shipDestroyed)* - в случае если в shipDestoyed == *true*, добавляет новую рандомную способность в очередь способностей
- *Skill *getSkill(SkillInfoHolder &skillInfoHolder)* - возвращает способность находящуюся первой в очереди и удаляет её. Если способностей в очереди нет то выбрасывает ошибку *NoAvailableSkills*

Тестирование:

```
int main()
    auto shipManager = ShipManager({3, 4});

    try
    {
        field.addShip(5, 3, West, shipManager.at(0));
        field.addShip(2, 8, North, shipManager.at(1));
    }       You, 2 часа назад • first commit
    catch (ShipNear &e)
    {
        std::cout << e.what() << '\n';
    }

    try
    {
        field.attack(-100, 100, 1);
    }
    catch (NotInField &e)
    {
        std::cout << e.what() << '\n';
    }

    field.attack(3, 3, 1);
    field.attack(4, 3, 2);
    field.attack(4, 4, 2);

    std::cout << "\n";
    std::cout << field;
    std::cout << "\n";
```

```
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .

Координаты не в поле

. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . 1 0 2 . . . .
. . . . . . . . . .
. . 2 . . . . . . .
. . 2 . . . . . . .
. . 2 . . . . . . .
. . 2 . . . . . . .
. . . . . . . . . .
```

Рисунки 2-3 - Код для проверки выброса и исключения и двойного урона и результат работы программы

```
Position position(3, 2);
auto skillInfoHolder = SkillInfoHolder(&field, &shipManager, &position);
auto skillManager = SkillManager(std::vector<SkillFactory *>());
std::cout << skillManager << '\n';

try
{
    std::cout << "Пробуем взять скилл когда их нет.\n";
    skillManager.getSkill(skillInfoHolder);
}
catch (NoSkillsAvailable &e)
{
    std::cout << e.what() << '\n';
}

// Shelling test
skillManager.addSkillFactory(new ShellingFactory);
skillManager.addSkillIfNeccessary(true);
auto skill = skillManager.getSkill(skillInfoHolder);
skillManager.addSkillIfNeccessary(skill->use());
free(skill);
```

```
Список доступных скиллов пуст :(

Пробуем взять скилл когда их нет.
Нет доступных скиллов
Skill: Shelling

. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . 0 0 2 . . . .
. . . . . . . . . .
. . 2 . . . . . . .
. . 2 . . . . . . .
. . 2 . . . . . . .
. . 2 . . . . . . .
. . . . . . . . . .
```

Рисунки 4 - 5 - тестирование выброса исключения при отсутсвии способностей в менеджере и способности - бомбардировки

```
// DoubleDamage test
skillManager.addSkillFactory(new DoubleDamageFactory);
skillManager.addSkillIfNeccessary(true);
auto skill = skillManager.getSkill(skillInfoHolder);
skill->use();
free(skill);
std::cout << "\n";
skillManager.addSkillIfNeccessary(field.attack(5, 3, 2));
std::cout << skillManager;
skillManager.addSkillIfNeccessary(field.attack(3, 3, 2));
std::cout << skillManager;
```

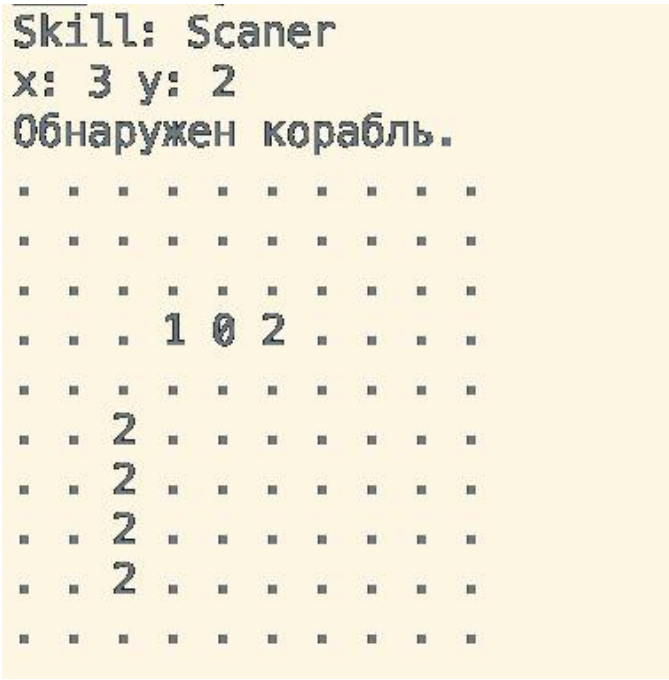
```
Skill: DoubleDamage

Список доступных скиллов пуст :(
DoubleDamage

. . . . . . . . . . .
. . . . . . . . . . .
. . . . . . . . . . .
. . . 0 0 0 . . . . .
. . . . . . . . . . .
. . 2 . . . . . . . .
. . 2 . . . . . . . .
. . 2 . . . . . . . .
. . 2 . . . . . . . .
. . . . . . . . . . .
```

Рисунки 6-7 - тестирование способности “Двойной урон”

```
// Scaner test
skillManager.addSkillFactory(new ScanerFactory);
skillManager.addSkillIfNeccessary(true);
auto skill = skillManager.getSkill(skillInfoHolder);
std::cout << (skill->use() ? "Обнаружен корабль." : "Кораблей нет.");
free(skill);
```



Рисунки 8-9 - тестирование способности Сканер

Выводы

Во время выполнения лабораторной работы, была изучена работа классов-интерфейсов и созданы: класс-интерфейс способности, класс менеджера-способностей и набор классов-исключений для обработки исключительных ситуаций