**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: Полиморфизм

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Иванов Д. М. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2024

## Цель работы

Изучить основы полиморфизма, наследования. Добавить к игре морской бой дополнения в виде способностей, которые реализуются через один абстрактный класс. Также создать классы-исключения. Затем связать все новые созданные классы и протестировать их работоспособность.

## Задание

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

1)Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).

2)Сканер - позволяет проверить участок поля 2х2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус.

3)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 кораблем

-Атака за границы поля

## Выполнение работы

1) Реализация способностей

Реализуем абстрактный класс способностей игрока Interface\_of\_abilities. Он будет содержать виртуальный метод, который будет иметься у всех дочерних классов:

virtual bool perform\_ability(Playground& playground) = 0 – метод выполнения способности. Принимает в качестве аргумента ссылку на игровое поле, куда нужно будет применить способность.

virtual ~Interface\_of\_abilities() = 0 – виртуальный деструктор

- Способность двойного удара

Напишем класс Double\_atack и унаследуем от нашего интерфейса (по аналогии с другими способностями).

private:

Double\_atack(); - конструктор

bool perform\_ability(Playground& playground); - выполнение способности. Будет изменена обертка в классе Playground на выполнение метода, и при выполнении shoot вызовется метод двойной атаки. После чего этот указатель выполнения удара снова вернется в исходное состояние.

- Способность сканнера

private:

Coords coords; - координаты выполнения способности

Scanner();

Scanner(Coords coords); - конструтор, принимающий координаты, куда необходимо будет применить способность.

bool perform\_ability(Playground& playground) – Пробегаемся циклом по квадрату и через метод поля проверяем, имеет ли данная координата состояние SHIP. Если есть такая точка, метод возвращает true, в противном случае – false.

- Способность обстрела

private:

std::mt19937 generator; - определяем данный генератор для того, чтобы при создании каждого нового объекта появлялась возможность генерировать случайные координаты для обстрела.

Shelling();

bool perform\_ability(Playground& playground); - через алгоритм вычисления рандомных чисел находим случаные координаты поля. Если на них находится корабль, применяется выстрел. И если корабль еще не уничтожен, проводится удар по сегменту. В противном случае координаты генерируется по-новому

2) Реализация “билдеров” способностей

В процессе цикла игра мы можем столкнуться со следующей проблемой. Мы изначально должны инициализировать способности для игрока. Если все оставить, как есть сейчас, игроку придется сразу вводить координаты, для которых он хочет применить ту или иную способность. Даже если пройдет большой промежуток игры и пользователю захочется поменять свое решение – это уже сделать не удастся. Для решения такой проблемы подойдут фабричные классы, “билдеры”. Они будут храниться в очереди во время игры. И когда пользователь захочет применить способность, он будет “дергать” эти классы, а они сразу создадут способность с теми координатами, которые введет пользователь перед самым использованием.

Сделаем абстрактный класс для билдеров.

class Interface\_of\_builders{

public:

virtual std::shared\_ptr<Interface\_of\_abilities> make\_ability() = 0; - метод создания способности.

virtual bool is\_need\_arguments() = 0; - проверка на то, нужны ли аргументы для той или иной способности.

virtual ~Interface\_of\_builders() = 0 – виртуальный деструктор

Для каждой способности реализация билдеров аналогичная. В методе создания способности через std::make\_shared<T> создается указатель на способность.

Однако для сканнера нужно прописать логику добавления координат. Для этого создадим новые классы: Get\_coords, Command\_coords.

В классе Get\_coords будет в поле хранится ссылка на координату, которую будем изменять в процессе игры. Также метод вывода этой координаты get\_coords.

Отнаследован этот класс будет от Command\_coords, который содежит 1 виртуальную команду get\_coords.

Эту команду мы будем хранить в Scanner\_builder

3) Менеджер способностей

class Ability\_maker:

Command\_coords\* command — указатель на команду получения координат

std::map<Name\_of\_builder, std::shared\_ptr<Interface\_of\_builders>> list\_of\_builders — словарь для получения указателя на билдер

Ability\_maker(Command\_coords\* command) — конструктор, принимающий указатель на команду

std::shared\_ptr<Interface\_of\_builders> get\_builder(Name\_of\_builder name) — возвращает указатель на билдер

В классе Manager\_of\_abilities мы будем хранить очередь способностей и позволять игроку вызывать одну из них.

private:

std::vector<std::shared\_ptr<Interface\_of\_builders>> vector\_of\_abilities – вектор указателей на объект билдера способности, использующийся для хранения очереди способностей во время игры.

Ability\_maker& maker – класс, создающий билдеры

public:

Manager\_of\_abilities(Ability\_maker& maker) – конструктор. Создается вектор, добавляются 3 способности. Потом через команду shuffle элементы переставляются в рандомном порядке. Передается ссылка на объект, создающий указатели на билдеры

void push\_ability(Name\_of\_builder builder) – добавляется в основную очередь указатель на новую способность

std::shared\_ptr<Interface\_of\_builders> get\_ability() – вовращается указатель на билдер способности и после этого удаляется из очереди.

4) Добавление способности при уничтожении корабля

Для этой реализации необходимо связать менеджер способностей и игровое поле. Просто добавить объект менеджера в поле будет небезопасно, слишком много доступа будет у класса игрового поля. Так что воспользуемся таким паттерном, как команда. Это поведенческий шаблон проектирования, позволяющий заворачивать запросы или простые операции в отдельные объекты.

Для этого реализуем абстрактный класс Command\_ability с одним виртуальным методом:

class Command\_ability:

virtual void add\_ability() = 0; - метод добавления способности.

Создадим отдельный класс добавления способности в очередь в наш менеджер. И отнаследуем его от интерфейса Command.

class Add\_ability: public Command\_ability{

private:

Manager\_of\_abilities& manager; - менеджер способностей, куда будем добавлять новую способность

public:

Add\_ability(Manager\_of\_abilities& p\_manager); - конструктор, принимающий менеджер

void add\_ability() - метод добавления.

И в итоге в поле Playground будет добавлено Command\_ability\* command. Передавать этот параметр мы будем в конструкторе. Добавим метод:

void add\_new\_ability\_for\_skills() – функция вызывается в тот момент, когда в методе shoot после попадания был уничтожен корабль. Выполняется command->add\_ability().

5) Классы-исключения

Все классы будут наследоваться от std::runtime\_error, вызываться через throw, а сообщения ошибки выводиться через what().

- Отсутствие способностей

class NoAbilityException

explicit NoAbilitiesException(const std::string& message = "You haven't got any abilities!") - при вызове throw будет вызываться runtime\_error с текстом шибкой, который передается в качестве аргумента. Проверка идет в менеджере способнсостей в методе get\_ability.

- Удар вне поля

class AtackException

private:

std::string constructMessage(int x, int y) const – в этом методе будет через тип данных ostringstream создаваться сообщение ошибки с указанием конкретных неверных координат.

public:

AtackException(int x, int y) – конструктор, вызывающийся через throw в методе shoot класса Playground.

- Неправильное расположение кораблей

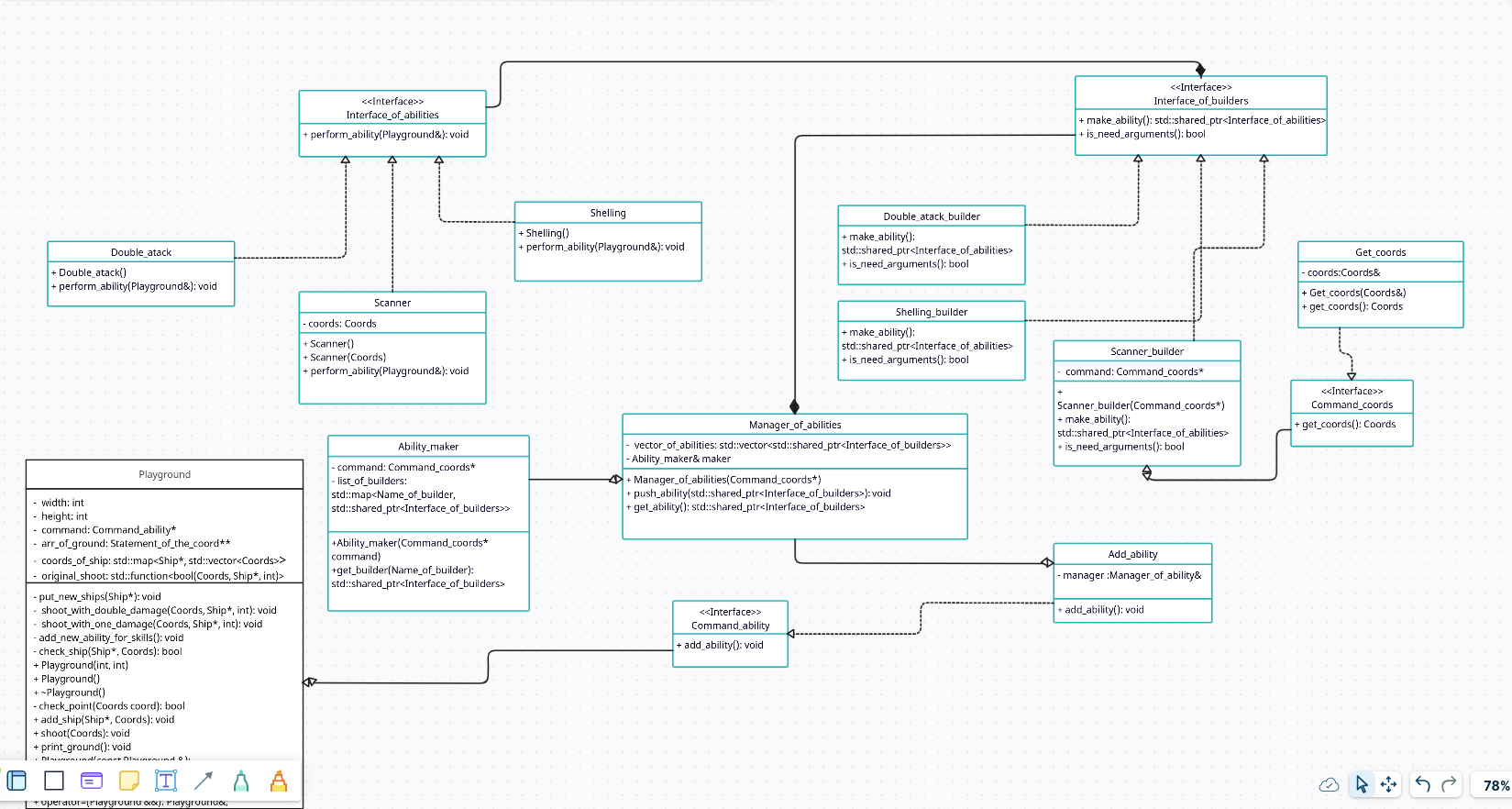
private:

std::string constructMessage(int x, int y) const – по аналогии.

explicit ShipPlacmentException(int x, int y) – по аналогии. Вызывается в методе check\_ship класса Playground.

## Представление в UML-диаграмме

Рисунок 1 — UML диаграмма



## Выводы

Были реализованы интерфейс способностей и 3 класса, наследовающихся от него. Для каждой отдельной способности была прописана своя логика выполнения. Также по аналогии прописаны интерфейс билдеров и сами билдеры. После этого был добавлен менеджер способностей, который связался с классом игрового поля через паттерн “команда”. Их работоспособность была проверена на пробных объектах, к которым применялись различные команды.

# Приложение А Исходный код программы

Название файла: Interface\_of\_abilities.h

#ifndef INTERFACE\_OF\_ABILITIES\_H

#define INTERFACE\_OF\_ABILITIES\_H

#include <iostream>

#include "Playground.h"

class Interface\_of\_abilities{

public:

virtual bool perform\_ability(Playground& playground) = 0;

virtual ~Interface\_of\_abilities() = 0;

};

#endif

Название файла: Double\_atack.h

#ifndef DOUBLE\_ATACK\_H

#define DOUBLE\_ATACK\_H

#include "Interface\_of\_abilities.h"

#include "Playground.h"

class Double\_atack: public Interface\_of\_abilities{

private:

Coords coords;

public:

Double\_atack();

Double\_atack(Coords coords);

bool perform\_ability(Playground& playground);

~Double\_atack() = default;

};

#endif

Название файла: Scanner.h

#ifndef SCANNER\_H

#define SCANNER\_H

#include <iostream>

#include "Interface\_of\_abilities.h"

#include "Playground.h"

class Scanner: public Interface\_of\_abilities{

private:

Coords coords;

public:

Scanner();

Scanner(Coords coords);

bool perform\_ability(Playground& playground);

~Scanner() = default;

};

#endif

Название файла: Shelling.h

#ifndef SHELLING\_H

#define SHELLING\_H

#include "Interface\_of\_abilities.h"

#include "Playground.h"

#include <iostream>

#include <ctime>

#include <random>

class Shelling: public Interface\_of\_abilities{

private:

std::mt19937 generator;

public:

Shelling();

bool perform\_ability(Playground& playground);

~Shelling() = default;

};

#endif

Название файла: Manager\_of\_abilities.h

#ifndef MANAGER\_OF\_ABILITIES\_H

#define MANAGER\_OF\_ABILITIES\_H

#include <iostream>

#include <vector>

#include <algorithm>

#include <random>

#include "Interface\_of\_abilities.h"

#include "Double\_atack\_builder.h"

#include "Shelling\_builder.h"

#include "Scanner\_builder.h"

#include "NoAbilitiesException.h"

#include "Command.h"

class Manager\_of\_abilities: public Command{

private:

std::vector<std::shared\_ptr<Interface\_of\_builders>> vector\_of\_abilities;

public:

Manager\_of\_abilities();

void add\_ability();

std::shared\_ptr<Interface\_of\_builders> get\_ability();

};

#endif

Название файла: Interface\_of\_builders.h

#ifndef INTERFACE\_OF\_BUILDERS\_H

#define INTERFACE\_OF\_BUILDERS\_H

#include <iostream>

#include <memory>

#include "Playground.h"

#include "Interface\_of\_abilities.h"

class Interface\_of\_builders{

public:

virtual std::shared\_ptr<Interface\_of\_abilities> make\_ability(Coords coords = {0, 0}) = 0;

virtual bool is\_need\_arguments() = 0;

virtual ~Interface\_of\_builders() = 0;

};

#endif

Название файла: Double\_atack\_builder.h

#ifndef DOUBLE\_ATACK\_BUILDER\_H

#define DOUBLE\_ATACK\_BUILDER\_H

#include <memory>

#include "Interface\_of\_builders.h"

#include "Playground.h"

#include "Double\_atack.h"

class Double\_atack\_builder: public Interface\_of\_builders{

public:

std::shared\_ptr<Interface\_of\_abilities> make\_ability(Coords coords = {0, 0});

bool is\_need\_arguments();

~Double\_atack\_builder() = default;

};

#endif

Название файла: Scanner\_builder.h

#ifndef SCANNER\_BUILDER\_H

#define SCANNER\_BUILDER\_H

#include <memory>

#include "Interface\_of\_builders.h"

#include "Playground.h"

#include "Scanner.h"

class Scanner\_builder: public Interface\_of\_builders{

public:

std::shared\_ptr<Interface\_of\_abilities> make\_ability(Coords coords = {0, 0});

bool is\_need\_arguments();

~Scanner\_builder() = default;

};

#endif

Название файла: Shelling\_builder.h

#ifndef SHELLING\_BUILDER\_H

#define SHELLING\_BUILDER\_H

#include <memory>

#include "Interface\_of\_builders.h"

#include "Playground.h"

#include "Shelling.h"

class Shelling\_builder: public Interface\_of\_builders{

public:

std::shared\_ptr<Interface\_of\_abilities> make\_ability(Coords coords = {0, 0});

bool is\_need\_arguments();

~Shelling\_builder() = default;

};

#endif