**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. 3341 |  | Лодыгин И.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Реализовать интерфейс способностей с использованием полиморфизма, а также набор классов-исключений.

## **Задание**

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

* Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).
* Сканер - позволяет проверить участок поля 2х2 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 кораблем
* 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**

В процессе продолжения разработки игры "Морской бой" были реализованы классы для взаимодействия со способностями: Ability, Scanner, DoubleDamage, RandomBombardment, а также набор классов исключений: NoAbilityException, ShipPlacementException, OutOfBoundsException.

**Классы исключений**

1. NoAbilityException

Описание: Этот класс используется для обработки ситуаций, когда в менеджере способностей больше нет доступных способностей для использования. Он наследуется от стандартного класса std::exception, что позволяет использовать его в стандартных блоках обработки исключений.

Методы:

what(): возвращает строку с описанием ошибки.

Роль: Позволяет сигнализировать о том, что игрок больше не может использовать способности, что улучшает обработку ошибок и информирование пользователя о состоянии игры.

1. ShipPlacementException

Описание: Этот класс обрабатывает исключения, возникающие при попытке разместить корабли на игровом поле, если они расположены слишком близко друг к другу или если они перекрываются. Это также наследуется от std::exception.

Методы:

what(): возвращает строку с описанием ошибки.

Роль: Обеспечивает защиту от неправильного размещения кораблей, что помогает поддерживать правила игры и предотвращает некорректное состояние игрового поля.

1. OutOfBoundsException

Описание: Этот класс используется для обработки ошибок, когда игрок пытается выполнить атаку за пределами игрового поля.

Методы:

what(): возвращает строку с описанием ошибки.

Роль: Позволяет обрабатывать ситуации, когда координаты атаки невалидны, что помогает поддерживать целостность игры.

**Классы способностей**

1. Ability

Описание: Это абстрактный класс, который определяет интерфейс для всех способностей в игре. Он содержит чисто виртуальный метод apply, который должен быть реализован в производных классах.

Методы:

apply(SeaBattlePlayground& field, ShipManager& manager): виртуальный метод, который должен быть переопределён в производных классах для выполнения конкретного действия способности.

Роль: Обеспечивает общую структуру для всех способностей, позволяя легко добавлять новые способности в будущем.

1. DoubleDamage

Описание: Этот класс реализует способность, которая позволяет игроку атаковать одну и ту же клетку дважды, нанося двойной урон.

Методы:

apply(SeaBattlePlayground& field, ShipManager& manager): запрашивает у игрока координаты для атаки и вызывает метод attack\_cell на игровом поле дважды.

Роль: Реализует конкретную логику для способности двойного урона, позволяя игроку использовать стратегические возможности для атаки.

1. RandomBombardment

Описание: Этот класс реализует способность, которая случайным образом атакует сегмент корабля противника.

Методы:

apply(SeaBattlePlayground& field, ShipManager& manager): выбирает случайный корабль и случайный сегмент этого корабля для атаки, вызывая метод attack\_cell.

Роль: Вносит элемент случайности и стратегии в игру, позволяя игрокам применять неожиданные тактики.

1. Scanner

Описание: Этот класс реализует способность, которая сканирует определённую область игрового поля и сообщает о местоположении недостающих сегментов кораблей.

Методы:

apply(SeaBattlePlayground& field, ShipManager& manager): запрашивает координаты для сканирования и проверяет состояние клеток, выводя информацию о найденных сегментах кораблей.

Роль: Позволяет игрокам выявлять местоположение кораблей противника, что добавляет элементы стратегии и планирования.

**Архитектурные решения**

Полиморфизм и интерфейсы: Использование абстрактного класса Ability позволяет создать различные способности, следуя единому интерфейсу. Это упрощает управление способностями и добавление новых без изменения основной логики.

Обработка ошибок: Классы исключений, такие как NoAbilityException, ShipPlacementException и OutOfBoundsException, обеспечивают структурированное управление ошибками, что помогает поддерживать корректное состояние игры и информировать пользователя о возникших проблемах.

Разделение ответственности: Каждый класс имеет свою конкретную ответственность:

* Ability и его производные классы управляют способностями.
* Исключения обрабатывают ошибки, связанные с игровой логикой.
* SeaBattlePlayground управляет состоянием игрового поля и размещением кораблей.

Случайные действия: Использование std::rand() и std::shuffle() для случайного выбора способностей и атакующих действий добавляет элемент неожиданности в игру и улучшает взаимодействие с пользователем.

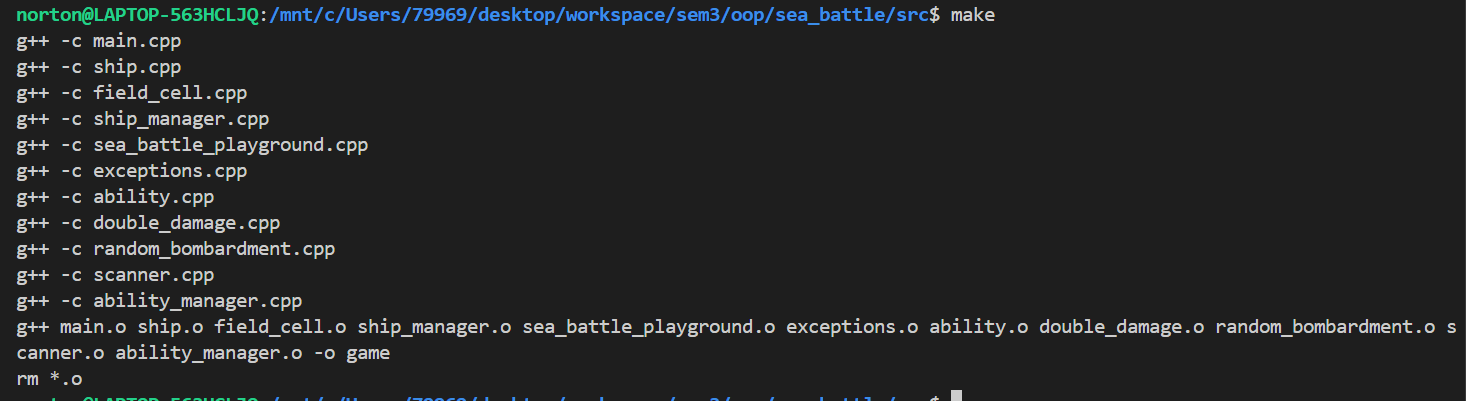
Управление памятью: В классе AbilityManager используется динамическое выделение памяти для объектов способностей, что позволяет управлять их жизненным циклом и избегать утечек памяти.

**Связь между классами**

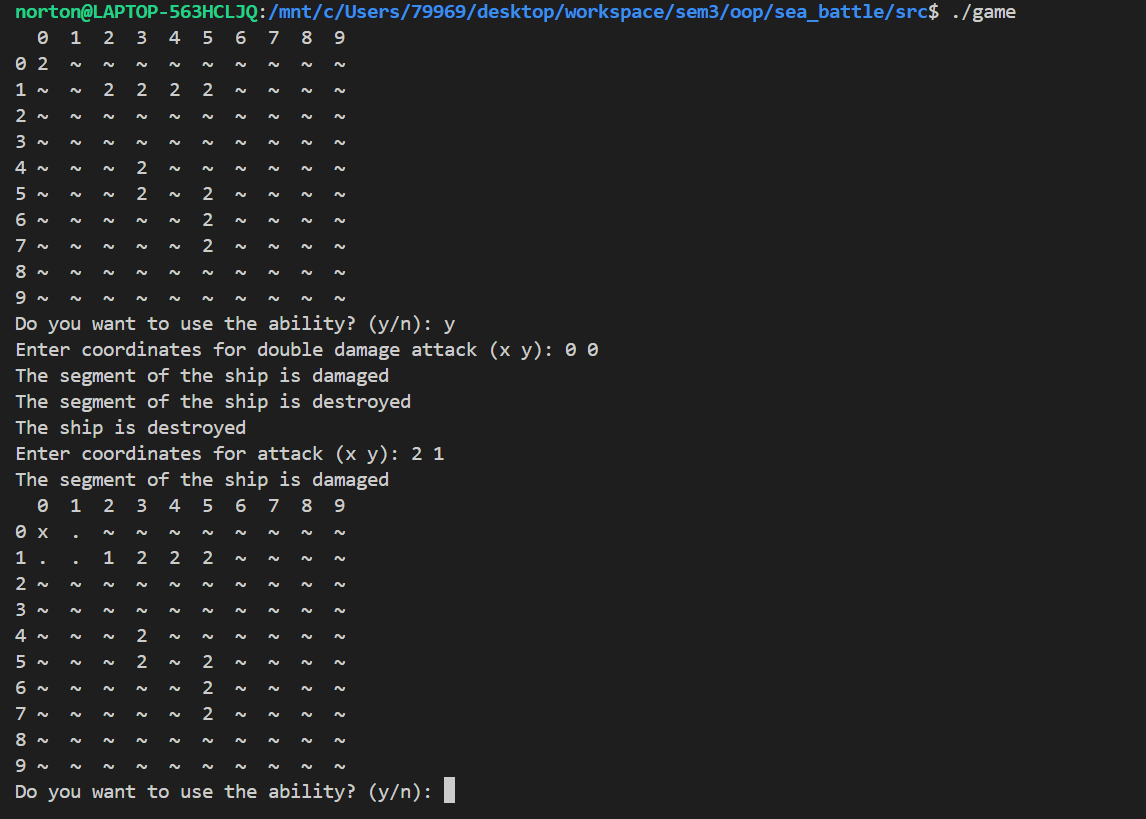
Ability является базовым классом для всех типов способностей, что позволяет AbilityManager легко управлять ими.

AbilityManager хранит и управляет способностями, а также предоставляет механизм для их использования. Он вызывает методы конкретных способностей, передавая необходимые параметры.

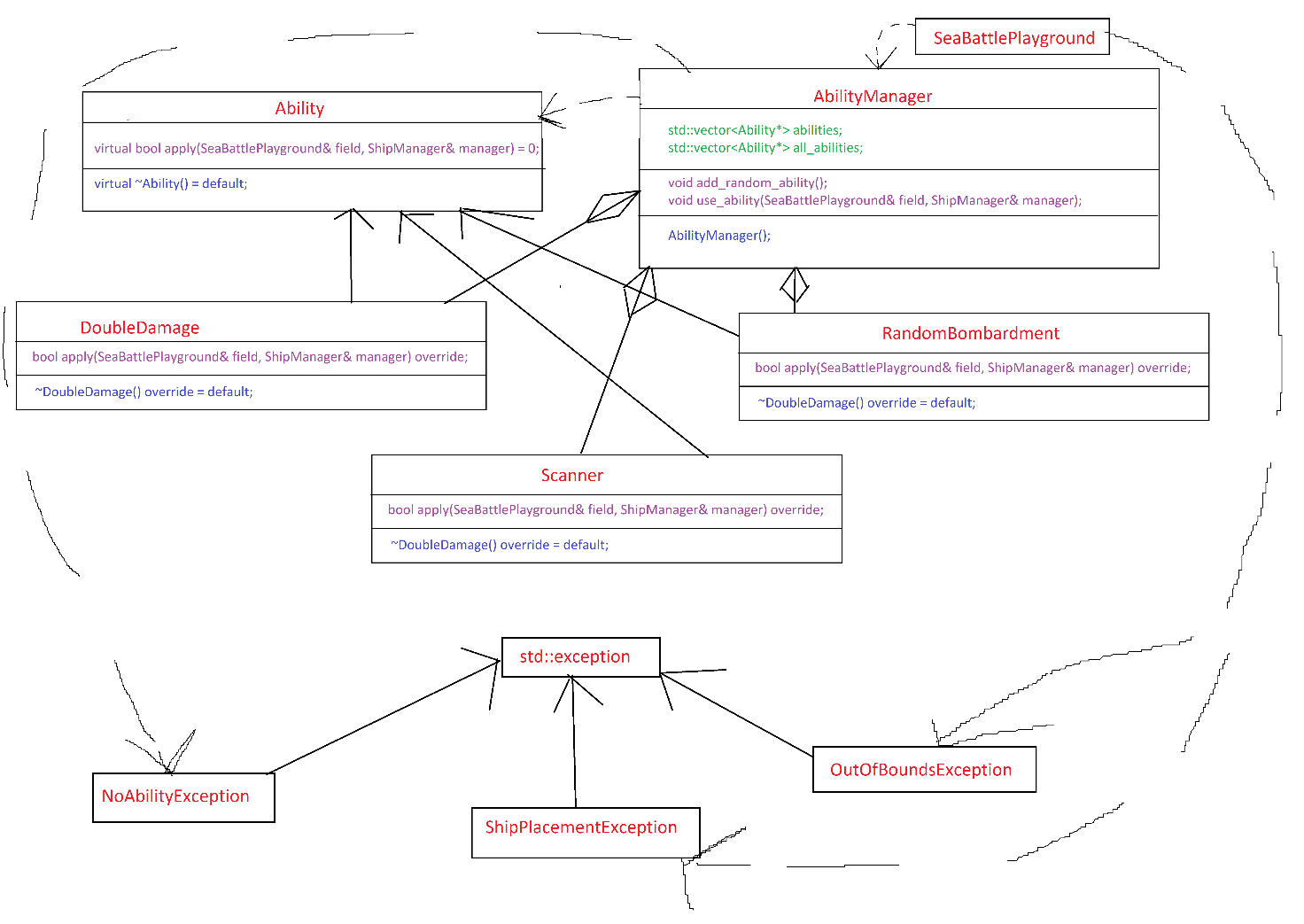
Классы исключений тесно связаны с игровыми процессами, вызывая соответствующие исключения в зависимости от состояния игрового поля, что позволяет игроку получать уведомления о неправильных действиях.

Все способности работают с классами SeaBattlePlayground и ShipManager, обеспечивая взаимодействие с состоянием игры и кораблями, что позволяет им быть взаимозависимыми и связанными в логике игры. 





UML диаграмма классов отображена ниже. На ней зеленым цветом отображены поля классов, синим цветом конструкторы/деструкторы, а фиолетовым – методы. Также на диаграмме отображены связи между классами.



Разработанный программный код см. в приложении А.

## **Выводы**

В ходе разработки были созданы классы, которые помогли реализовать менеджер способностей, а также набор классов-исключений.

# **Приложение А Исходный код программы**

Название файла: main.cpp

#include <iostream>

#include <exception>

#include <vector>

#include <ctime>

#include "sea\_battle\_playground.h"

#include "ship\_manager.h"

#include "ability\_manager.h"

int main(){

std::srand(std::time(0));

try {

AbilityManager ability\_manager;

SeaBattlePlayground my\_field(10, 10);

int ships\_quantity = 4;

std::vector<int> ship\_sizes = { 1, 2, 3, 4 };

ShipManager ships(ships\_quantity, ship\_sizes);

my\_field.place\_ship(ships.ships[0], 0, 0, false);

my\_field.place\_ship(ships.ships[1], 3, 4, true);

my\_field.place\_ship(ships.ships[2], 5, 5, true);

my\_field.place\_ship(ships.ships[3], 2, 1, false);

my\_field.display\_your\_playing\_field();

my\_field.ability\_attack\_cell(ability\_manager, ships);

my\_field.display\_your\_playing\_field();

my\_field.ability\_attack\_cell(ability\_manager, ships);

my\_field.display\_your\_playing\_field();

}

catch (const std::exception& e) {

std::cout << e.what() << std::endl;

}

return 0;

}

Название файла: ability.cpp

#include "ability.h"

Название файла: ability.h

#ifndef ABILITY\_H

#define ABILITY\_H

#include "sea\_battle\_playground.h"

#include "ship\_manager.h"

class Ability {

public:

virtual bool apply(SeaBattlePlayground& field, ShipManager& manager) = 0;

virtual ~Ability() = default;

};

#endif

Название файла: scanner.cpp

#include <iostream>

#include "sea\_battle\_playground.h"

#include "ship\_manager.h"

#include "ability.h"

#include "scanner.h"

enum CellState {

EMPTY\_CELL = '.',

UNKNOWN\_CELL = '~',

DAMAGED\_SHIP = '1',

UNDAMAGED\_SHIP = '2',

DESTROYED\_SHIP = 'x'

};

bool Scanner::apply(SeaBattlePlayground& field, ShipManager& manager){

std::cout << "Enter coordinates for scan (x y): ";

int x, y;

std::cin >> x >> y;

for(int i = x; i < x + 2; ++i) {

for(int j = y; j < y + 2; ++j) {

if(field.point\_exists(i, j) && field.field[i][j].cell\_state == UNDAMAGED\_SHIP) {

std::cout << "Ship segment detected at (" << i << ", " << j << ")" << std::endl;

}

}

}

return false;

}

Название файла: scanner.h

#ifndef SCANNER\_H

#define SCANNER\_H

#include "sea\_battle\_playground.h"

#include "ship\_manager.h"

#include "ability.h"

class Scanner : public Ability {

public:

bool apply(SeaBattlePlayground& field, ShipManager& manager) override;

~Scanner() override = default;

};

#endif

Название файла: double\_damage.cpp

#include <iostream>

#include "sea\_battle\_playground.h"

#include "ship\_manager.h"

#include "double\_damage.h"

bool DoubleDamage::apply(SeaBattlePlayground& field, ShipManager& manager){

int x, y;

std::cout << "Enter coordinates for double damage attack (x y): ";

std::cin >> x >> y;

if(field.attack\_cell(x, y)) return true;

if(field.attack\_cell(x, y)) return true;

return false;

}

Название файла: double\_damage.h

#ifndef DOUBLE\_DAMAGE\_H

#define DOUBLE\_DAMAGE\_H

#include "sea\_battle\_playground.h"

#include "ship\_manager.h"

#include "ability.h"

class DoubleDamage : public Ability {

public:

bool apply(SeaBattlePlayground& field, ShipManager& manager) override;

~DoubleDamage() override = default;

};

#endif

Название файла: random\_bombardment.сpp

#include <iostream>

#include "sea\_battle\_playground.h"

#include "ship\_manager.h"

#include "ability.h"

#include "random\_bombardment.h"

bool RandomBombardment::apply(SeaBattlePlayground& field, ShipManager& manager){

int ship\_index = rand() % manager.quantity;

Ship& target\_ship = manager.get\_ship(ship\_index);

int segment = rand() % target\_ship.length;

for (int x = 0; x < field.width; x++) {

for (int y = 0; y < field.length; y++) {

FieldCell& cell = field.field[x][y];

if (cell.ship == &target\_ship && cell.ship\_part\_index == segment) {

std::cout << "Random bombardment applied to ship " << ship\_index

<< " at coordinates (" << x << ", " << y << ")" << std::endl;

return(field.attack\_cell(x, y));

}

}

}

throw std::runtime\_error("Segment not found on the field");

}

Название файла: random\_bombardment.h

#ifndef RANDOM\_BOMBARDMENT\_H

#define RANDOM\_BOMBARDMENT\_H

#include "sea\_battle\_playground.h"

#include "ship\_manager.h"

#include "ability.h"

class RandomBombardment : public Ability {

public:

bool apply(SeaBattlePlayground& field, ShipManager& manager) override;

~RandomBombardment() override = default;

};

#endif

Название файла: exceptions.cpp

#include <exception>

#include "exceptions.h"

const char\* NoAbilityException::what() const noexcept{

return "No abilities available to use.";

}

const char\* ShipPlacementException::what() const noexcept{

return "Ships cannot be placed too close to each other or overlap.";

}

const char\* OutOfBoundsException::what() const noexcept{

return "Attack is out of bounds.";

}

Название файла: exceptions.h

#ifndef EXCEPTIONS\_H

#define EXCPETIONS\_H

#include <exception>

class NoAbilityException : public std::exception {

public:

const char\* what() const noexcept override;

};

class ShipPlacementException : public std::exception {

public:

const char\* what() const noexcept override;

};

class OutOfBoundsException : public std::exception {

public:

const char\* what() const noexcept override;

};

#endif

Название файла: Makefile

all : game

main.o : main.cpp

g++ -c main.cpp

ship.o : ship.cpp

g++ -c ship.cpp

field\_cell.o : field\_cell.cpp

g++ -c field\_cell.cpp

ship\_manager.o : ship\_manager.cpp

g++ -c ship\_manager.cpp

sea\_battle\_playground.o : sea\_battle\_playground.cpp

g++ -c sea\_battle\_playground.cpp

exceptions.o : exceptions.cpp

g++ -c exceptions.cpp

ability.o : ability.cpp

g++ -c ability.cpp

double\_damage.o : double\_damage.cpp

g++ -c double\_damage.cpp

random\_bombardment.o : random\_bombardment.cpp

g++ -c random\_bombardment.cpp

scanner.o : scanner.cpp

g++ -c scanner.cpp

ability\_manager.o : ability\_manager.cpp

g++ -c ability\_manager.cpp

game : main.o ship.o field\_cell.o ship\_manager.o sea\_battle\_playground.o exceptions.o ability.o double\_damage.o random\_bombardment.o scanner.o ability\_manager.o

g++ main.o ship.o field\_cell.o ship\_manager.o sea\_battle\_playground.o exceptions.o ability.o double\_damage.o random\_bombardment.o scanner.o ability\_manager.o -o game

rm \*.o