**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Полиморфизм

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Шаповаленко Е.В. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы

Исследовать и проанализировать концепцию полиморфизма в языке С++, применить его на практике.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Ознакомиться с понятием полиморфизма и изучить его основные типы
2. Научиться проектировать классы с учетом полиморфизма
3. Применить полиморфизм на практике при помощи классов-интерфейсов

## Задание

А) 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.

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

C) Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.

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

1. Попытка применить способность, когда их нет
2. Размещение корабля вплотную или на пересечении с другим кораблем
3. 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

Способности реализованы при помощи класса-интерфейса *IAbility*. В нем есть только один метод – *cast*(). Этот метод использует способность.

У каждой из способностей: *DoubleDamage*, *Scanner*, *Bombardment*, - есть свой конструктор, который принимает на вход необходимые для способности аргументы. Для *DoubleDamage* это ссылка на переменную, в которой хранится урон при атаке. Для *Scanner* это ссылка на игровое поле, координаты верхнего левого угла области сканирования и ссылка на переменную, в которой будет храниться результат сканирования. Для *Bombardment* это ссылка на игровое поле и значение урона при атаке.

Для того, чтобы каждой способности передавать свои параметры реализованы классы настроек способностей при помощи класса-интерфейса *IAbilitySettings*. Каждые настройки принимают в конструкторе параметры, необходимые для соответствующей способности. Метод *getType*() позволяет получить тип способности, а метод *acceptVisitor*(*IVisitor&* *visitor*) позволяет неявно преобразовать *IAbilitySettings*\* к указателю на конкретную способность.

Для неявного преобразования указателей на способности реализован класс *AbilitySettingsVisitor*. Класс является реализацией интерфейса *IVisitor*. Это сделано для решения проблем с кольцевым включением в проекте. *AbilitySettingsVisitor* в конструкторе принимает ссылку на фабрику, которой будут создаваться способности. Кроме того, есть метод *visit* с перегрузками. Когда в классе настроек способностей вызывается метод *acceptVisitor*(*IVisitor&* *visitor*), внутри этого метода вызывается *visitor*.*visit*(*this*). Указатель *this* хранит указатель конкретного типа настроек способностей, то есть нет необходимости его приводить, как это было бы с *IAbilitySettings*\*. Метод *visit* имеет перегрузки под каждый конкретный тип способностей. Внутри этого методы вызывается конкретный метод фабрики *factory*\_, который создает внутри фабрики необходимую способность.

Для того, чтобы не хранить способности напрямую, реализован класс *AbilityFactory*. В поле ability\_ хранится указатель на созданную способность (по умолчанию *nullptr*). Методы *buildDoubleDamage*, *buildScanner*, *buildBombardment*, принимая на вход соответствующие настройки, создают соответствующую способность. Метод *getAbility*() позволяет получить указатель на созданную способность.

Класс *AbilityManager* отвечает за способности игрока. При создании объекта класса в очередь *abilities*\_ добавляется по одной способности каждого вида. *abilities*\_ хранит не сами способности, а только их типы. При вызове метода *castAbility*(*IAbilitySettings*\* *settings*) создается экземпляр способности того вида, который хранится в *abilities\_.front*() (начало очереди), при помощи метода *buildAbility*. *buildAbility* создает способность при помощи фабрики *factory*\_. В качестве аргумента фабрике подается *settings*. После этого вызывается метод *cast*() у созданной способности, после чего она удаляется.

При попытке вызова *castAbility*, когда в очереди нет способностей, или настройки не соответствуют создаваемой способности, бросается соответствующая ошибка.

Метод *size*() возвращает количество способностей в очереди.

Метод *getAbility*() позволяет получить тип способности в начале очереди.

Метод *addAbility*() позволяет добавить в конец очереди случайную способность.

Таким образом, способность создается только при вызове метода *castAbility* класса *AbilityManager*, сразу получая необходимые параметры.

Ниже представлена UML-диаграмма реализованных в данной работе классов:

|  |
| --- |
|  |
| Рисунок 1: UML-диаграмма классов |

Разработанный программный код см. в приложении А.

Тестирование программы см. в приложении Б.

## Выводы

Цель работы - исследовать и проанализировать концепцию полиморфизма в языке С++, применить его на практике – была достигнута.

В ходе выполнения работы были выполнены следующие задачи:

1. Ознакомиться с понятием полиморфизма и изучить его основные типы
2. Научиться проектировать классы с учетом полиморфизма
3. Применить полиморфизм на практике при помощи классов-интерфейсов

# Приложение А Исходный код программы

Название файла: include/ship/ship.h

#ifndef SHIP

#define SHIP

#include <vector>

#include <stdexcept>

enum class ShipOrientation : int

{

horizontal,

vertical

};

enum class ShipStatus : int

{

dead,

alive

};

enum class ShipSegmentStatus : int

{

destroyed,

damaged,

intact

};

class Ship

{

public:

Ship();

explicit Ship(int size);

Ship(const Ship& other);

Ship& operator=(const Ship& other);

Ship(Ship&& other);

Ship& operator=(Ship&& other);

~Ship();

int getSize() const noexcept;

ShipSegmentStatus getSegmentStatus(int index) const;

ShipStatus getShipStatus() const noexcept;

void damageSegment(int index, int damage);

private:

class ShipSegment

{

public:

ShipSegment();

ShipSegment(const ShipSegment& other);

ShipSegment& operator=(const ShipSegment& other);

ShipSegment(ShipSegment&& other);

ShipSegment& operator=(ShipSegment&& other);

~ShipSegment() = default;

void takeDamege(int damage) noexcept;

ShipSegmentStatus getStatus() const noexcept;

private:

int kMaxHealth = 2;

int health\_;

};

int kMinSize = 1;

int kMaxSize = 4;

int size\_;

std::vector<ShipSegment> segments\_;

};

#endif

Название файла: source/ship/ship.cpp

#include "ship.h"

#include <vector>

#include <stdexcept>

Ship::Ship() = default;

Ship::Ship(int size)

{

if (size < kMinSize || size > kMaxSize) {

throw std::logic\_error("Ship size can be from 1 to 4");

}

size\_ = size;

for (int i = 0; i < size\_; i++) {

segments\_.push\_back(ShipSegment());

}

}

Ship::Ship(const Ship& other) :

Ship(other.size\_)

{

for (int i = 0; i < size\_; i++) {

segments\_[i] = other.segments\_[i];

}

}

Ship& Ship::operator=(const Ship& other)

{

size\_ = other.size\_;

for (int i = 0; i < size\_; i++) {

segments\_.push\_back(ShipSegment());

segments\_[i] = other.segments\_[i];

}

return \*this;

}

Ship::Ship(Ship&& other) :

size\_(std::move(other.size\_)),

segments\_(std::move(other.segments\_))

{}

Ship& Ship::operator=(Ship&& other)

{

size\_ = std::move(other.size\_);

segments\_ = std::move(other.segments\_);

return \*this;

}

Ship::~Ship()

{

segments\_.clear();

}

int Ship::getSize() const noexcept

{

return size\_;

}

ShipSegmentStatus Ship::getSegmentStatus(int index) const

{

if (index < 0 || index >= size\_) {

throw std::out\_of\_range("Ship segment index out of range");

}

return segments\_[index].getStatus();

}

ShipStatus Ship::getShipStatus() const noexcept

{

int dead\_flag = 1;

for (int i = 0; i < size\_; i++) {

if (segments\_[i].getStatus() != ShipSegmentStatus::destroyed) {

dead\_flag = 0;

break;

}

}

if (dead\_flag) {

return ShipStatus::dead;

} else {

return ShipStatus::alive;

}

}

void Ship::damageSegment(int index, int damage)

{

if (index < 0 || index >= size\_) {

throw std::out\_of\_range("Ship segment index out of range");

}

segments\_[index].takeDamege(damage);

}

Ship::ShipSegment::ShipSegment()

{

health\_ = kMaxHealth;

}

Ship::ShipSegment::ShipSegment(const ShipSegment& other) :

health\_(other.health\_)

{}

Ship::ShipSegment& Ship::ShipSegment::operator=(const ShipSegment& other)

{

health\_ = other.health\_;

return \*this;

}

Ship::ShipSegment::ShipSegment(ShipSegment&& other) :

health\_(std::move(other.health\_))

{}

Ship::ShipSegment& Ship::ShipSegment::operator=(ShipSegment&& other)

{

health\_ = std::move(other.health\_);

return \*this;

}

void Ship::ShipSegment::takeDamege(int damage) noexcept

{

health\_ = std::max(0, health\_ - damage);

}

ShipSegmentStatus Ship::ShipSegment::getStatus() const noexcept

{

if (health\_ == kMaxHealth) {

return ShipSegmentStatus::intact;

} else if (health\_ == 0) {

return ShipSegmentStatus::destroyed;

} else {

return ShipSegmentStatus::damaged;

}

}

Название файла: include/ship/shipManager.h

#ifndef SHIP\_MANAGER

#define SHIP\_MANAGER

#include <iostream>

#include <initializer\_list>

#include <vector>

#include "ship.h"

#include "field.h"

class ShipManager

{

public:

ShipManager();

explicit ShipManager(std::initializer\_list<int> ship\_sizes);

~ShipManager();

void printShips() const noexcept;

void addShip(int ship\_size);

Ship\* getUnusedShip(int index) const;

Ship\* getUsedShip(int index) const;

void makeShipUsed(int index);

int getUnusedShipSize() const noexcept;

int getUsedShipSize() const noexcept;

private:

std::vector<Ship\*> unused\_ships\_;

std::vector<Ship\*> used\_ships\_;

};

#endif

Название файла:source/ship/ shipManager.cpp

#include "shipManager.h"

#include "ship.h"

#include "field.h"

#include <iostream>

#include <initializer\_list>

#include <vector>

#include <utility>

ShipManager::ShipManager() = default;

ShipManager::ShipManager(std::initializer\_list<int> ship\_sizes)

{

for (auto ship\_size : ship\_sizes) {

Ship\* ship = new Ship(ship\_size);

unused\_ships\_.push\_back(ship);

}

}

ShipManager::~ShipManager()

{

for (int i = 0; i < unused\_ships\_.size(); i++) {

delete unused\_ships\_[i];

}

unused\_ships\_.clear();

for (int i = 0; i < used\_ships\_.size(); i++) {

delete used\_ships\_[i];

}

used\_ships\_.clear();

}

void ShipManager::printShips() const noexcept

{

int counter = 0;

std::cout << "Unused ships:" << "\n";

for (auto ship: unused\_ships\_) {

std::cout << "Ship " << counter++ << ": ";

for (int i = 0; i < ship->getSize(); i++) {

if (ship->getSegmentStatus(i) == ShipSegmentStatus::intact) {

std::cout << "[2]";

} else if (ship->getSegmentStatus(i) == ShipSegmentStatus::damaged) {

std::cout << "[1]";

} else {

std::cout << "[0]";

}

}

std::cout << "\n";

}

if (unused\_ships\_.size() == 0) {

std::cout << "None" << "\n";

}

std::cout << "\n";

counter = 0;

std::cout << "Used ships:" << "\n";

for (auto ship: used\_ships\_) {

std::cout << "Ship " << counter++ << ": ";

for (int i = 0; i < ship->getSize(); i++) {

if (ship->getSegmentStatus(i) == ShipSegmentStatus::intact) {

std::cout << "[2]";

} else if (ship->getSegmentStatus(i) == ShipSegmentStatus::damaged) {

std::cout << "[1]";

} else {

std::cout << "[0]";

}

}

std::cout << "\n";

}

if (used\_ships\_.size() == 0) {

std::cout << "None" << "\n";

}

std::cout << "\n";

return;

}

void ShipManager::addShip(int ship\_size)

{

Ship\* ship = new Ship(ship\_size);

unused\_ships\_.push\_back(ship);

}

Ship\* ShipManager::getUnusedShip(int index) const

{

if (index < 0 || index >= unused\_ships\_.size()) {

throw std::out\_of\_range("Ship index out of range");

}

return unused\_ships\_[index];

}

Ship\* ShipManager::getUsedShip(int index) const

{

if (index < 0 || index >= used\_ships\_.size()) {

throw std::out\_of\_range("Ship index out of range");

}

return used\_ships\_[index];

}

void ShipManager::makeShipUsed(int index)

{

if (index < 0 || index >= unused\_ships\_.size()) {

throw std::out\_of\_range("Ship index out of range");

}

std::move(unused\_ships\_.begin()+index, unused\_ships\_.begin()+index+1, std::back\_inserter(used\_ships\_));

unused\_ships\_.erase(unused\_ships\_.begin()+index);

}

int ShipManager::getUnusedShipSize() const noexcept

{

return unused\_ships\_.size();

}

int ShipManager::getUsedShipSize() const noexcept

{

return used\_ships\_.size();

}

Название файла: source/ship/Makefile

CXX = g++

INCLUDE\_DIR = ../../include/

INCLUDE = $(wildcard $(INCLUDE\_DIR)ship/\*.h $(INCLUDE\_DIR)field/\*.h)

CXXFLAGS = -I $(INCLUDE\_DIR)ship/ -I $(INCLUDE\_DIR)field/ -c

SRC\_DIR = ./

SRC = $(wildcard $(SRC\_DIR)\*.cpp)

BIN\_DIR = ../../bin/

OBJ = $(patsubst $(SRC\_DIR)%.cpp, $(BIN\_DIR)%.o, $(SRC))

all : $(OBJ)

$(BIN\_DIR)%.o : %.cpp $(INCLUDE)

$(CXX) $(CXXFLAGS) $< -o $@

Название файла: include/field/field.h

#ifndef FIELD

#define FIELD

#include <iostream>

#include <vector>

#include <algorithm>

#include <stdexcept>

#include <unordered\_map>

#include "ship.h"

enum class FieldCellStatus : int

{

unknown,

empty,

ship

};

struct Coords

{

int x;

int y;

bool operator==(const Coords& other) const;

};

struct CoordsHash

{

std::size\_t operator()(const Coords& coords) const;

};

class Field

{

public:

Field();

explicit Field(int size\_x, int size\_y);

Field(const Field& other);

Field& operator=(const Field& other);

Field(Field&& other);

Field& operator=(Field&& other);

~Field();

void placeShip(Ship\* ship, Coords coords, ShipOrientation orientation);

bool attackCell(Coords coords, int damage);

void printField() const noexcept;

bool isShip(Coords coords) const;

Coords getSize() const noexcept;

void getRandomShip(Ship\*\* ship, int& index) const noexcept;

private:

class ShipCell

{

public:

ShipCell();

explicit ShipCell(Ship\* ship, int index);

~ShipCell();

void attackCell(int damage);

void setShipSegment(Ship\* ship, int index) noexcept;

ShipSegmentStatus getShipSegmentStatus() const noexcept;

ShipStatus getShipStatus() const noexcept;

void getShipSegment(Ship\*\* ship, int& index) const noexcept;

private:

Ship\* ship\_;

int ship\_segment\_index\_;

};

class FieldCell

{

public:

FieldCell();

explicit FieldCell(FieldCellStatus status);

~FieldCell();

FieldCellStatus getStatus() const noexcept;

void setStatus(FieldCellStatus status) noexcept;

private:

FieldCellStatus status\_;

};

int size\_x\_;

int size\_y\_;

std::unordered\_map<Coords, ShipCell, CoordsHash> ships\_;

std::unordered\_map<Coords, FieldCell, CoordsHash> opened\_cells\_;

};

#endif

Название файла: source/field/field.cpp

#include "field.h"

#include "ship.h"

#include <iostream>

#include <vector>

#include <algorithm>

#include <stdexcept>

#include <unordered\_map>

#include "shipPlacementException.h"

#include "attackOutOfRangeException.h"

bool Coords::operator==(const Coords& other) const

{

return x == other.x && y == other.y;

}

std::size\_t CoordsHash::operator()(const Coords& coords) const

{

return std::hash<int>()(coords.x) ^ (std::hash<int>()(coords.y) << 1);

}

Field::Field() = default;

Field::Field(int size\_x, int size\_y)

{

if (size\_x <= 0 || size\_y <= 0) {

throw std::invalid\_argument("Field size must be grater than 0");

}

size\_x\_ = size\_x;

size\_y\_ = size\_y;

}

Field::Field(const Field& other) :

ships\_(other.ships\_),

opened\_cells\_(other.opened\_cells\_),

size\_x\_(other.size\_x\_),

size\_y\_(other.size\_y\_)

{}

Field& Field::operator=(const Field& other)

{

if (this == &other) {

return \*this;

}

ships\_ = other.ships\_;

opened\_cells\_ = other.opened\_cells\_;

size\_x\_ = other.size\_x\_;

size\_y\_ = other.size\_y\_;

return \*this;

}

Field::Field(Field&& other) :

ships\_(std::move(other.ships\_)),

opened\_cells\_(std::move(other.opened\_cells\_)),

size\_x\_(std::move(other.size\_x\_)),

size\_y\_(std::move(other.size\_y\_))

{}

Field& Field::operator=(Field&& other)

{

if (this == &other) {

return \*this;

}

ships\_ = std::move(other.ships\_);

opened\_cells\_ = std::move(other.opened\_cells\_);

size\_x\_ = std::move(other.size\_x\_);

size\_y\_ = std::move(other.size\_y\_);

return \*this;

}

Field::~Field()

{

ships\_.clear();

opened\_cells\_.clear();

}

void Field::placeShip(Ship\* ship, Coords coords, ShipOrientation orientation)

{

if (ship == nullptr) {

throw ShipPlacementException("Ship pointer is nullptr");

}

int offset\_x, offset\_y;

if (orientation == ShipOrientation::horizontal) {

offset\_x = ship->getSize() - 1;

offset\_y = 0;

} else {

offset\_x = 0;

offset\_y = ship->getSize() - 1;

}

if (coords.x < 0 || coords.x >= size\_x\_ - offset\_x || coords.y < 0 || coords.y >= size\_y\_ - offset\_y) {

throw ShipPlacementException("Ship coordinates out of range");

}

for (int i = coords.x - 1; i < coords.x + offset\_x + 1; i++) {

if (i >= 0 && i < size\_x\_) {

for (int j = coords.y - 1; j < coords.y + offset\_y + 1; j++) {

if (j >= 0 && j < size\_y\_) {

if (ships\_.count({i, j}) != 0) {

throw ShipPlacementException("Ships may not contact each other");

}

}

}

}

}

int segment\_index = 0;

for (int i = coords.x; i < coords.x + offset\_x + 1; i++) {

for (int j = coords.y; j < coords.y + offset\_y + 1; j++) {

ships\_[{i, j}] = ShipCell(ship, segment\_index++);

}

}

}

bool Field::attackCell(Coords coords, int damage)

{

if (coords.x < 0 || coords.x >= size\_x\_ || coords.y < 0 || coords.y >= size\_y\_) {

throw AttackOutOfRangeException("Coordinates out of range");

}

if (ships\_.count({coords.x, coords.y}) != 0) {

ships\_[{coords.x, coords.y}].attackCell(damage);

damage = 1;

opened\_cells\_[{coords.x, coords.y}] = FieldCell(FieldCellStatus::ship);

if (ships\_[{coords.x, coords.y}].getShipStatus() == ShipStatus::dead) {

return true;

}

return false;

}

opened\_cells\_[{coords.x, coords.y}] = FieldCell(FieldCellStatus::empty);

return false;

}

void Field::printField() const noexcept

{

for (int y = 0; y < size\_y\_; y++) {

for (int x = 0; x < size\_x\_; x++) {

if (opened\_cells\_.count({x, y}) != 0) {

if (opened\_cells\_.at({x, y}).getStatus() == FieldCellStatus::ship) {

if (ships\_.at({x, y}).getShipSegmentStatus() == ShipSegmentStatus::intact) {

std::cout << "[2]";

} else if (ships\_.at({x, y}).getShipSegmentStatus() == ShipSegmentStatus::damaged) {

std::cout << "[1]";

} else {

std::cout << "[0]";

}

} else {

std::cout << "[.]";

}

} else {

std::cout << "[ ]";

}

}

std::cout << "\n";

}

std::cout << "\n";

}

bool Field::isShip(Coords coords) const

{

if (coords.x < 0 || coords.x >= size\_x\_ || coords.y < 0 || coords.y >= size\_y\_) {

throw std::out\_of\_range("Coordinates out of range");

}

return ships\_.count({coords.x, coords.y}) != 0;

}

Coords Field::getSize() const noexcept

{

return {size\_x\_, size\_y\_};

}

void Field::getRandomShip(Ship\*\* ship, int& index) const noexcept

{

int ship\_index = rand() % ships\_.size();

auto it = ships\_.begin();

std::advance(it, ship\_index);

it->second.getShipSegment(ship, index);

return;

}

Field::ShipCell::ShipCell() = default;

Field::ShipCell::ShipCell(Ship\* ship, int index)

{

ship\_ = ship;

ship\_segment\_index\_ = index;

}

Field::ShipCell::~ShipCell() = default;

void Field::ShipCell::attackCell(int damage)

{

ship\_->damageSegment(ship\_segment\_index\_, damage);

}

void Field::ShipCell::setShipSegment(Ship\* ship, int index) noexcept

{

ship\_ = ship;

ship\_segment\_index\_ = index;

}

ShipSegmentStatus Field::ShipCell::getShipSegmentStatus() const noexcept

{

return ship\_->getSegmentStatus(ship\_segment\_index\_);

}

ShipStatus Field::ShipCell::getShipStatus() const noexcept

{

return ship\_->getShipStatus();

}

void Field::ShipCell::getShipSegment(Ship\*\* ship, int& index) const noexcept

{

\*ship = ship\_;

index = ship\_segment\_index\_;

}

Field::FieldCell::FieldCell()

{

status\_ = FieldCellStatus::unknown;

}

Field::FieldCell::FieldCell(FieldCellStatus status)

{

status\_ = status;

}

Field::FieldCell::~FieldCell() = default;

FieldCellStatus Field::FieldCell::getStatus() const noexcept

{

return status\_;

}

void Field::FieldCell::setStatus(FieldCellStatus status) noexcept

{

status\_ = status;

}

Название файла: include/field/attackOutOfRangeException.h

#ifndef ATTACK\_OUT\_OF\_RANGE\_EXCEPTION

#define ATTACK\_OUT\_OF\_RANGE\_EXCEPTION

#include <string>

class AttackOutOfRangeException

{

public:

AttackOutOfRangeException(std::string message);

~AttackOutOfRangeException();

std::string what() const noexcept;

private:

std::string message\_;

};

#endif

Название файла: source/field/attackOutOfRangeException.cpp

#include "attackOutOfRangeException.h"

#include <string>

AttackOutOfRangeException::AttackOutOfRangeException(std::string message = "") :

message\_(message)

{}

AttackOutOfRangeException::~AttackOutOfRangeException() = default;

std::string AttackOutOfRangeException::what() const noexcept

{

return message\_;

}

Название файла: include/field/shipPlacementException.h

#ifndef SHIP\_PLACEMENT\_EXCEPTION

#define SHIP\_PLACEMENT\_EXCEPTION

#include <string>

class ShipPlacementException

{

public:

ShipPlacementException(std::string message);

~ShipPlacementException();

std::string what() const noexcept;

private:

std::string message\_;

};

#endif

Название файла: source/field/shipPlacementException.cpp

#include "shipPlacementException.h"

#include <string>

ShipPlacementException::ShipPlacementException(std::string message = "") :

message\_(message)

{}

ShipPlacementException::~ShipPlacementException() = default;

std::string ShipPlacementException::what() const noexcept

{

return message\_;

}

Название файла: source/field/Makefile

CXX = g++

INCLUDE\_DIR = ../../include/

INCLUDE = $(wildcard $(INCLUDE\_DIR)ship/\*.h $(INCLUDE\_DIR)field/\*.h)

CXXFLAGS = -I $(INCLUDE\_DIR)ship/ -I $(INCLUDE\_DIR)field/ -c

SRC\_DIR = ./

SRC = $(wildcard $(SRC\_DIR)\*.cpp)

BIN\_DIR = ../../bin/

OBJ = $(patsubst $(SRC\_DIR)%.cpp, $(BIN\_DIR)%.o, $(SRC))

all : $(OBJ)

$(BIN\_DIR)%.o : %.cpp $(INCLUDE)

$(CXX) $(CXXFLAGS) $< -o $@

Название файла: include/ability/abilityFactory.h

#ifndef ABILITY\_FACTORY

#define ABILITY\_FACTORY

#include "iAbility.h"

#include "doubleDamage.h"

#include "doubleDamageSettings.h"

#include "scanner.h"

#include "scannerSettings.h"

#include "bombardment.h"

#include "bombardmentSettings.h"

class AbilityFactory

{

public:

AbilityFactory();

~AbilityFactory();

IAbility\* getAbility() const noexcept;

void buildDoubleDamage(DoubleDamageSettings\* settings);

void buildScanner(ScannerSettings\* settings);

void buildBombardment(BombardmentSettings\* settings);

private:

IAbility\* ability\_ = nullptr;

};

#endif

Название файла: source/ability/abilityFactory.cpp

#include "abilityFactory.h"

#include "iAbility.h"

#include "doubleDamage.h"

#include "doubleDamageSettings.h"

#include "scanner.h"

#include "scannerSettings.h"

#include "bombardment.h"

#include "bombardmentSettings.h"

AbilityFactory::AbilityFactory() = default;

AbilityFactory::~AbilityFactory() = default;

IAbility\* AbilityFactory::getAbility() const noexcept

{

return ability\_;

}

void AbilityFactory::buildDoubleDamage(DoubleDamageSettings\* settings)

{

ability\_ = new DoubleDamage(settings->damage\_);

}

void AbilityFactory::buildScanner(ScannerSettings\* settings)

{

ability\_ = new Scanner(settings->field\_, settings->coords\_, settings->scan\_result\_);

}

void AbilityFactory::buildBombardment(BombardmentSettings\* settings)

{

ability\_ = new Bombardment(settings->field\_, settings->damage\_);

}

Название файла: include/ability/abilityManager.h

#ifndef ABILITY\_MANAGER

#define ABILITY\_MANAGER

#include <queue>

#include <vector>

#include <ctime>

#include "abilityFactory.h"

#include "iAbility.h"

#include "abilitySettingsVisitor.h"

class AbilityManager

{

public:

AbilityManager();

~AbilityManager();

int size() const noexcept;

AbilityType getAbility() const;

void castAbility(IAbilitySettings\* settings);

void addAbility();

private:

IAbility\* buildAbility(IAbilitySettings\* settings);

int kAbilitiesNum = 3;

std::queue<AbilityType> abilities\_;

AbilityFactory factory\_;

AbilitySettingsVisitor visitor\_;

};

#endif

Название файла: source/ability abilityManager.cpp

#include "abilityManager.h"

#include <queue>

#include <vector>

#include <ctime>

#include "abilityFactory.h"

#include "iAbility.h"

#include "abilitySettingsVisitor.h"

#include "noAbilityException.h"

AbilityManager::AbilityManager() :

visitor\_(factory\_)

{

srand(time(0));

std::vector<AbilityType> abilitiesType = {AbilityType::DoubleDamage, AbilityType::Scanner, AbilityType::Bombardment};

int ability\_index;

for (int i = 0; i < kAbilitiesNum; i++) {

ability\_index = rand() % abilitiesType.size();

abilities\_.push(abilitiesType[ability\_index]);

abilitiesType.erase(abilitiesType.begin() + ability\_index);

}

}

AbilityManager::~AbilityManager() = default;

int AbilityManager::size() const noexcept

{

return abilities\_.size();

}

AbilityType AbilityManager::getAbility() const

{

if (this->size() == 0) {

throw NoAbilityException("No abilities avaliable");

}

return abilities\_.front();

}

void AbilityManager::castAbility(IAbilitySettings\* settings)

{

if (this->size() == 0) {

throw NoAbilityException("No abilities avaliable");

}

if(settings->getType() != abilities\_.front()) {

throw std::logic\_error("Invalid ability settings type");

}

IAbility\* ability = this->buildAbility(settings);

ability->cast();

delete ability;

abilities\_.pop();

}

void AbilityManager::addAbility()

{

int ability\_index = rand() % kAbilitiesNum;

switch (ability\_index)

{

case 0:

{

abilities\_.push(AbilityType::DoubleDamage);

break;

}

case 1:

{

abilities\_.push(AbilityType::Scanner);

break;

}

case 2:

{

abilities\_.push(AbilityType::Bombardment);

break;

}

default:

break;

}

}

IAbility\* AbilityManager::buildAbility(IAbilitySettings\* settings)

{

settings->acceptVisitor(visitor\_);

return factory\_.getAbility();

}

Название файла: include/ability/abilitySettingsVisitor.h

#ifndef ABILITY\_SETTINGS\_VISITOR

#define ABILITY\_SETTINGS\_VISITOR

#include "iVisitor.h"

#include "abilityFactory.h"

class AbilitySettingsVisitor : public IVisitor

{

public:

AbilitySettingsVisitor(AbilityFactory& factory\_);

~AbilitySettingsVisitor();

void visit(class DoubleDamageSettings\* settings);

void visit(class ScannerSettings\* settings);

void visit(class BombardmentSettings\* settings);

private:

AbilityFactory& factory\_;

};

#endif

Название файла: source/ability/abilitySettingsVisitor.cpp

#include "abilitySettingsVisitor.h"

#include "abilityFactory.h"

AbilitySettingsVisitor::AbilitySettingsVisitor(AbilityFactory& factory) :

factory\_(factory)

{}

AbilitySettingsVisitor::~AbilitySettingsVisitor() = default;

void AbilitySettingsVisitor::visit(DoubleDamageSettings\* settings)

{

factory\_.buildDoubleDamage(settings);

}

void AbilitySettingsVisitor::visit(ScannerSettings\* settings)

{

factory\_.buildScanner(settings);

}

void AbilitySettingsVisitor::visit(BombardmentSettings\* settings)

{

factory\_.buildBombardment(settings);

}

Название файла: include/ability/bombardment.h

#ifndef BOMBARDMENT

#define BOMBARDMENT

#include "iAbility.h"

#include "field.h"

class Bombardment: public IAbility

{

public:

Bombardment(Field& field, int damage);

~Bombardment();

void cast() override;

private:

Field& field\_;

int damage\_;

};

#endif

Название файла: source/ability/bombardment.cpp

#include "bombardment.h"

#include "iAbility.h"

#include "field.h"

Bombardment::Bombardment(Field& field, int damage) :

field\_(field),

damage\_(damage)

{}

Bombardment::~Bombardment() = default;

void Bombardment::cast()

{

Ship\* ship;

int index;

field\_.getRandomShip(&ship, index);

ship->damageSegment(index, damage\_);

}

Название файла: include/ability/bombardmentSettings.h

#ifndef BOMBARDMENT\_SETTINGS

#define BOMBARDMENT\_SETTINGS

#include "iAbilitySettings.h"

#include "iVisitor.h"

#include "field.h"

class BombardmentSettings: public IAbilitySettings

{

public:

Field& field\_;

int damage\_;

BombardmentSettings(Field& field, int damage);

~BombardmentSettings();

AbilityType getType() override;

void acceptVisitor(IVisitor& visitor) override;

};

#endif

Название файла: source/ability/bombardmentSettings.cpp

#include "bombardmentSettings.h"

#include "iAbilitySettings.h"

#include "iVisitor.h"

#include "field.h"

BombardmentSettings::BombardmentSettings(Field& field, int damage) :

field\_(field),

damage\_(damage)

{}

BombardmentSettings::~BombardmentSettings() = default;

AbilityType BombardmentSettings::getType()

{

return AbilityType::Bombardment;

}

void BombardmentSettings::acceptVisitor(IVisitor& visitor)

{

visitor.visit(this);

}

Название файла: include/ability/doubleDamage.h

#ifndef DOUBLE\_DAMAGE

#define DOUBLE\_DAMAGE

#include "iAbility.h"

class DoubleDamage: public IAbility

{

public:

DoubleDamage(int& damage);

~DoubleDamage();

void cast() override;

private:

int& damage\_;

};

#endif

Название файла: source/ability/doubleDamage.cpp

#include "doubleDamage.h"

#include "iAbility.h"

DoubleDamage::DoubleDamage(int& damage) :

damage\_(damage)

{}

DoubleDamage::~DoubleDamage() = default;

void DoubleDamage::cast()

{

damage\_ \*= 2;

}

Название файла: include/ability/doubleDamageSettings.h

#ifndef DOUBLE\_DAMAGE\_SETTINGS

#define DOUBLE\_DAMAGE\_SETTINGS

#include "iAbilitySettings.h"

#include "iVisitor.h"

class DoubleDamageSettings: public IAbilitySettings

{

public:

int& damage\_;

DoubleDamageSettings(int& damage);

~DoubleDamageSettings();

AbilityType getType() override;

void acceptVisitor(IVisitor& visitor) override;

};

#endif

Название файла: source/ability/doubleDamageSettings.cpp

#include "doubleDamageSettings.h"

#include "iAbilitySettings.h"

#include "iVisitor.h"

DoubleDamageSettings::DoubleDamageSettings(int& damage) :

damage\_(damage)

{}

DoubleDamageSettings::~DoubleDamageSettings() = default;

AbilityType DoubleDamageSettings::getType()

{

return AbilityType::DoubleDamage;

}

void DoubleDamageSettings::acceptVisitor(IVisitor& visitor)

{

visitor.visit(this);

}

Название файла: include/ability/scanner.h

#ifndef SCANER

#define SCANER

#include "iAbility.h"

#include "field.h"

class Scanner: public IAbility

{

public:

Scanner(Field& field, Coords coords, bool& scan\_result);

~Scanner();

void cast() override;

private:

int kScanRadius = 2;

Field& field\_;

Coords coords\_;

bool& scan\_result\_;

};

#endif

Название файла: source/ability/scanner.cpp

#include "scanner.h"

#include "iAbility.h"

#include "field.h"

Scanner::Scanner(Field& field, Coords coords, bool& scan\_result) :

field\_(field),

coords\_(coords),

scan\_result\_(scan\_result)

{}

Scanner::~Scanner() = default;

void Scanner::cast()

{

int size\_x = field\_.getSize().x;

int size\_y = field\_.getSize().y;

for (int i = coords\_.x; i < coords\_.x + kScanRadius; i++) {

for (int j = coords\_.y; j < coords\_.y + kScanRadius; j++) {

if (i >= 0 && i < size\_x && j >= 0 && j < size\_y) {

if (field\_.isShip({i, j})) {

scan\_result\_ = true;

return;

}

}

}

}

scan\_result\_ = false;

return;

}

Название файла: include/ability/scannerSettings.h

#ifndef SCANER\_SETTINGS

#define SCANER\_SETTINGS

#include "iAbilitySettings.h"

#include "iVisitor.h"

#include "field.h"

class ScannerSettings: public IAbilitySettings

{

public:

Field& field\_;

Coords coords\_;

bool& scan\_result\_;

ScannerSettings(Field& field, Coords coords, bool& scan\_result);

~ScannerSettings();

AbilityType getType() override;

void acceptVisitor(IVisitor& visitor) override;

};

#endif

Название файла: source/ability/scannerSettings.cpp

#include "scannerSettings.h"

#include "iAbilitySettings.h"

#include "iVisitor.h"

#include "field.h"

ScannerSettings::ScannerSettings(Field& field, Coords coords, bool& scan\_result) :

field\_(field),

coords\_(coords),

scan\_result\_(scan\_result)

{}

ScannerSettings::~ScannerSettings() = default;

AbilityType ScannerSettings::getType()

{

return AbilityType::Scanner;

}

void ScannerSettings::acceptVisitor(IVisitor& visitor)

{

visitor.visit(this);

}

Название файла: include/ability/iAbility.h

#ifndef ABILITY

#define ABILITY

class IAbility

{

public:

virtual void cast() = 0;

};

#endif

Название файла: include/ability/iAbilitySettings.h

#ifndef ABILITY\_SETTINGS

#define ABILITY\_SETTINGS

#include "iVisitor.h"

enum class AbilityType : int

{

DoubleDamage,

Scanner,

Bombardment

};

class IAbilitySettings

{

public:

virtual AbilityType getType() = 0;

virtual void acceptVisitor(IVisitor& visitor) = 0;

};

#endif

Название файла: include/ability/iVisitor.h

#ifndef VISITOR

#define VISITOR

class IVisitor

{

public:

virtual void visit(class DoubleDamageSettings\* settings) = 0;

virtual void visit(class ScannerSettings\* settings) = 0;

virtual void visit(class BombardmentSettings\* settings) = 0;

};

#endif

Название файла: include/ability/noAbilityException.h

#ifndef NO\_ABILITY\_EXCEPTION

#define NO\_ABILITY\_EXCEPTION

#include <string>

class NoAbilityException

{

public:

NoAbilityException(std::string message);

~NoAbilityException();

std::string what() const noexcept;

private:

std::string message\_;

};

#endif

Название файла: source/ability/noAbilityException.cpp

#include "noAbilityException.h"

#include <string>

NoAbilityException::NoAbilityException(std::string message = "") :

message\_(message)

{}

NoAbilityException::~NoAbilityException() = default;

std::string NoAbilityException::what() const noexcept

{

return message\_;

}

Название файла: source/main.cpp

#include <iostream>

#include "ship.h"

#include "shipManager.h"

#include "field.h"

#include "abilityManager.h"

#include "noAbilityException.h"

#include "shipPlacementException.h"

#include "attackOutOfRangeException.h"

int main()

{

int playerDamage = 1;

bool scan\_result;

ShipManager ship\_manager({1, 2, 3, 4});

Field field(5, 5);

AbilityManager ability\_manager;

field.placeShip(ship\_manager.getUnusedShip(2), {0, 0}, ShipOrientation::horizontal);

ship\_manager.makeShipUsed(2);

field.placeShip(ship\_manager.getUnusedShip(0), {4, 4}, ShipOrientation::horizontal);

ship\_manager.makeShipUsed(0);

field.attackCell({0, 0}, 0);

field.attackCell({1, 0}, 0);

field.attackCell({2, 0}, 0);

std::cout << "Ability count: " << ability\_manager.size() << "\n\n";

if (field.attackCell({4, 4}, 2)) {

ability\_manager.addAbility();

}

ship\_manager.printShips();

field.printField();

std::cout << "Ability count: " << ability\_manager.size() << "\n\n";

AbilityType a\_type;

while (ability\_manager.size() > 0) {

a\_type = ability\_manager.getAbility();

if (a\_type == AbilityType::DoubleDamage) {

std::cout << "Double Damage:\n";

DoubleDamageSettings settings(playerDamage);

ability\_manager.castAbility(&settings);

field.attackCell({0 ,0}, playerDamage);

ship\_manager.printShips();

field.printField();

} else if (a\_type == AbilityType::Scanner) {

std::cout << "Scanner:\n";

ScannerSettings settings(field, {0, 0}, scan\_result);

ability\_manager.castAbility(&settings);

if (scan\_result) {

std::cout << "Ship\n\n";

} else {

std::cout << "No ship\n\n";

}

} else {

std::cout << "Bombardment:\n";

BombardmentSettings settings(field, playerDamage);

ability\_manager.castAbility(&settings);

ship\_manager.printShips();

field.printField();

playerDamage = 1;

}

}

std::cout << "Ability count: " << ability\_manager.size() << "\n\n";

try

{

DoubleDamageSettings settings(playerDamage);

ability\_manager.castAbility(&settings);

}

catch(const NoAbilityException& e)

{

std::cout << e.what() << '\n';

}

try

{

field.placeShip(ship\_manager.getUnusedShip(0), {0, 1}, ShipOrientation::vertical);

ship\_manager.makeShipUsed(0);

}

catch(const ShipPlacementException& e)

{

std::cout << e.what() << '\n';

}

try

{

field.placeShip(ship\_manager.getUnusedShip(0), {0, 4}, ShipOrientation::vertical);

ship\_manager.makeShipUsed(0);

}

catch(const ShipPlacementException& e)

{

std::cout << e.what() << '\n';

}

try

{

field.attackCell({5, 5}, playerDamage);

}

catch(const AttackOutOfRangeException& e)

{

std::cout << e.what() << '\n';

}

return 0;

}

Название файла: Makefile

TARGET = ./main

CXX = g++

INCLUDE\_DIR = ./include/

INCLUDE = $(wildcard $(INCLUDE\_DIR)\*/\*.h)

CXXFLAGS = -I $(INCLUDE\_DIR)ship/ -I $(INCLUDE\_DIR)field/ -I $(INCLUDE\_DIR)ability/

SRC\_DIR = ./source/

SRC = $(wildcard $(SRC\_DIR)\*/\*.cpp)

BIN\_DIR = ./bin/

OBJ = $(addprefix $(BIN\_DIR), $(patsubst %.cpp, %.o, $(notdir $(SRC))))

all : create\_bin\_dir ability field ship $(TARGET)

ability :

make -C $(SRC\_DIR)ability/

field :

make -C $(SRC\_DIR)field/

ship :

make -C $(SRC\_DIR)ship/

$(BIN\_DIR)main.o : $(SRC\_DIR)main.cpp $(INCLUDE\_DIR)

$(CXX) $(CXXFLAGS) -c $< -o $@

$(TARGET) : $(OBJ) $(BIN\_DIR)main.o

$(CXX) $(CXXFLAGS) $^ -o $@

create\_bin\_dir :

@mkdir -p $(BIN\_DIR)

clean:

@rm $(TARGET)

@rm -rf $(BIN\_DIR)

.

# Приложение Б Тестирование программы

Название файла: source/main.cpp

#include <iostream>

#include "ship.h"

#include "shipManager.h"

#include "field.h"

#include "abilityManager.h"

#include "noAbilityException.h"

#include "shipPlacementException.h"

#include "attackOutOfRangeException.h"

int main()

{

int playerDamage = 1;

bool scan\_result;

ShipManager ship\_manager({1, 2, 3, 4});

Field field(5, 5);

AbilityManager ability\_manager;

field.placeShip(ship\_manager.getUnusedShip(2), {0, 0}, ShipOrientation::horizontal);

ship\_manager.makeShipUsed(2);

field.placeShip(ship\_manager.getUnusedShip(0), {4, 4}, ShipOrientation::horizontal);

ship\_manager.makeShipUsed(0);

field.attackCell({0, 0}, 0);

field.attackCell({1, 0}, 0);

field.attackCell({2, 0}, 0);

std::cout << "Ability count: " << ability\_manager.size() << "\n\n";

if (field.attackCell({4, 4}, 2)) {

ability\_manager.addAbility();

}

ship\_manager.printShips();

field.printField();

std::cout << "Ability count: " << ability\_manager.size() << "\n\n";

AbilityType a\_type;

while (ability\_manager.size() > 0) {

a\_type = ability\_manager.getAbility();

if (a\_type == AbilityType::DoubleDamage) {

std::cout << "Double Damage:\n";

DoubleDamageSettings settings(playerDamage);

ability\_manager.castAbility(&settings);

field.attackCell({0 ,0}, playerDamage);

ship\_manager.printShips();

field.printField();

} else if (a\_type == AbilityType::Scanner) {

std::cout << "Scanner:\n";

ScannerSettings settings(field, {0, 0}, scan\_result);

ability\_manager.castAbility(&settings);

if (scan\_result) {

std::cout << "Ship\n\n";

} else {

std::cout << "No ship\n\n";

}

} else {

std::cout << "Bombardment:\n";

BombardmentSettings settings(field, playerDamage);

ability\_manager.castAbility(&settings);

ship\_manager.printShips();

field.printField();

playerDamage = 1;

}

}

std::cout << "Ability count: " << ability\_manager.size() << "\n\n";

try

{

DoubleDamageSettings settings(playerDamage);

ability\_manager.castAbility(&settings);

}

catch(const NoAbilityException& e)

{

std::cout << e.what() << '\n';

}

try

{

field.placeShip(ship\_manager.getUnusedShip(0), {0, 1}, ShipOrientation::vertical);

ship\_manager.makeShipUsed(0);

}

catch(const ShipPlacementException& e)

{

std::cout << e.what() << '\n';

}

try

{

field.placeShip(ship\_manager.getUnusedShip(0), {0, 4}, ShipOrientation::vertical);

ship\_manager.makeShipUsed(0);

}

catch(const ShipPlacementException& e)

{

std::cout << e.what() << '\n';

}

try

{

field.attackCell({5, 5}, playerDamage);

}

catch(const AttackOutOfRangeException& e)

{

std::cout << e.what() << '\n';

}

return 0;

}

Результат работы программы:

