**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: Создание классов

|  |  |  |
| --- | --- | --- |
| Студент гр. 3381 |  | Щеглов М.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы.

Цель работы - разработать классы для представления корабля, менеджера кораблей и игрового поля, а также в создании связей между ними. Для этого необходимо изучить концепцию классов и их структуру, реализовать указанные классы в соответствии с требованиями задания и написать программу для проверки их функциональности.

## Задание.

А) Создать класс корабля, который будет размещаться на игровом поле. Корабль может иметь длину от 1 до 4, а также может быть расположен вертикально или горизонтально. Каждый сегмент корабля может иметь три различных состояния: целый, поврежден, уничтожен. Изначально у корабля все сегменты целые. При нанесении 1 урона по сегменту, он становится поврежденным, а при нанесении 2 урона по сегменту, уничтоженным. Также добавить методы для взаимодействия с кораблем.

Б) Создать класс менеджера кораблей, хранящий информацию о кораблях. Данный класс в конструкторе принимает количество кораблей и их размеры, которые нужно расставить на поле.

В) Создать класс игрового поля, которое в конструкторе принимает размеры. У поля должен быть метод, принимающий корабль, координаты, на которые нужно поставить, и его ориентацию на поле. Корабли на поле не могут соприкасаться или пересекаться. Для игрового поля добавить методы для указания того, какая клетка атакуется. При попадании в сегмент корабля изменения должны отображаться в менеджере кораблей.

Каждая клетка игрового поля имеет три статуса:

* неизвестно (изначально вражеское поле полностью неизвестно),
* пустая (если на клетке ничего нет)
* корабль (если в клетке находится один из сегментов корабля).

Для класса игрового поля также необходимо реализовать конструкторы копирования и перемещения, а также соответствующие им операторы присваивания.

Примечания:

* Не забывайте для полей и методов определять модификаторы доступа
* Для обозначения переменной, которая принимает небольшое ограниченное количество значений, используйте enum
* Не используйте глобальные переменные
* При реализации копирования нужно выполнять глубокое копирование
* При реализации перемещения, не должно быть лишнего копирования
* При выделении памяти делайте проверку на переданные значения
* У поля не должно быть методов, возвращающих указатель на поле в явном виде, так как это небезопасно

## Выполнение работы

Реализован **класс Ship,** который представляет собой модель корабля для игры "Морской бой". Класс управляет состоянием корабля, его сегментами и взаимодействием с игровым полем.

Методы класса Ship:

1) Конструктор *explicit Ship(int size):* создаёт корабль заданного размера. Проверяет, что размер находится в допустимом диапазоне (от 1 до 4).

2) *int getSize() const noexcept:* возвращает размер корабля, который был установлен при создании объекта.

3) *ShipStatus getShipStatus() const noexcept*: проверяет состояние всех сегментов корабля. Если все сегменты уничтожены, возвращает статус dead, иначе alive.

4) *ShipOrientation getOrientation() const noexcept*: возвращает ориентацию корабля (горизонтальная или вертикальная).

5) *void damageSegment(int index)*: наносит урон сегменту корабля по индексу. Если индекс выходит за пределы допустимого диапазона, выбрасывается исключение.

Внутренний **класс ShipSegment** и его методы:

1) *Конструктор ShipSegment():* инициализирует сегмент корабля в состоянии Whole (целый).

2) *void takeDamage():* устанавливает статус сегмента как Destroyed при попадании.

3) *ShipSegmentStatus getStatus() const noexcept*: возвращает текущее состояние сегмента: целый (Whole) или уничтоженный (Destroyed).

Реализован **класс ShipManager**, который отвечает за создание и управление несколькими кораблями в игре "Морской бой". Основное назначение класса — хранение коллекции кораблей и предоставление к ним доступа.

Методы класса ShipManager:

1. *Конструктор ShipManager(const std::vector<int>& shipSizes):* принимает вектор размеров кораблей и создаёт объекты Ship на основе этих размеров. Проверяет корректность размеров (от 1 до 4), выбрасывая исключение в случае ошибки.
2. *const Ship& getShip(int index) const*: возвращает ссылку на корабль по указанному индексу. Если индекс выходит за допустимые границы, выбрасывается исключение.
3. *int getShipCount() const noexcept*: возвращает количество кораблей, хранящихся в менеджере. Используется для получения общего числа кораблей в игре.

**Класс GameField** реализует игровое поле для игры "Морской бой". Он отвечает за хранение статуса каждой клетки поля, размещение кораблей и обработку атак. Основное назначение класса — управление клетками поля и взаимодействие с кораблями.

Методы класса GameField:

1) Конструктор *GameField(int width, int height)*: инициализирует игровое поле заданного размера (ширина и высота). Все клетки поля изначально имеют статус "неизвестно". Проверяет, что размеры поля положительные.

2) Конструктор копирования *GameField(const GameField& other):* создаёт копию другого объекта GameField, выполняя глубокое копирование всех данных поля.

3) Конструктор перемещения *GameField(GameField&& other) noexcept:* перемещает данные из другого объекта GameField, минимизируя копирование.

4) Оператор присваивания копирования *GameField& operator=(const GameField& other):* осуществляет глубокое копирование данных из другого объекта GameField.

5) Оператор присваивания перемещения *GameField& operator=(GameField&& other) noexcept:* выполняет перемещение данных из другого объекта GameField, минимизируя копирование.

6) *bool placeShip(const Ship& ship, int x, int y, ShipOrientation orientation):* размещает корабль на поле с указанными координатами и ориентацией (горизонтальная или вертикальная). Проверяет границы поля и отсутствие пересечений или соприкосновений с другими кораблями.

7) *bool attack(int x, int y, ShipManager& shipManager):* обрабатывает атаку по указанным координатам. Если атака попадает в корабль, статус клетки меняется, а корабль в ShipManager обновляется.

8) *CellStatus getCellStatus(int x, int y) const:* возвращает текущий статус клетки на поле (неизвестно, пусто, корабль).

9) *Деструктор ~GameField():* очищает ресурсы, связанные с игровым полем (в данном случае память освобождается автоматически, так как используются стандартные контейнеры).

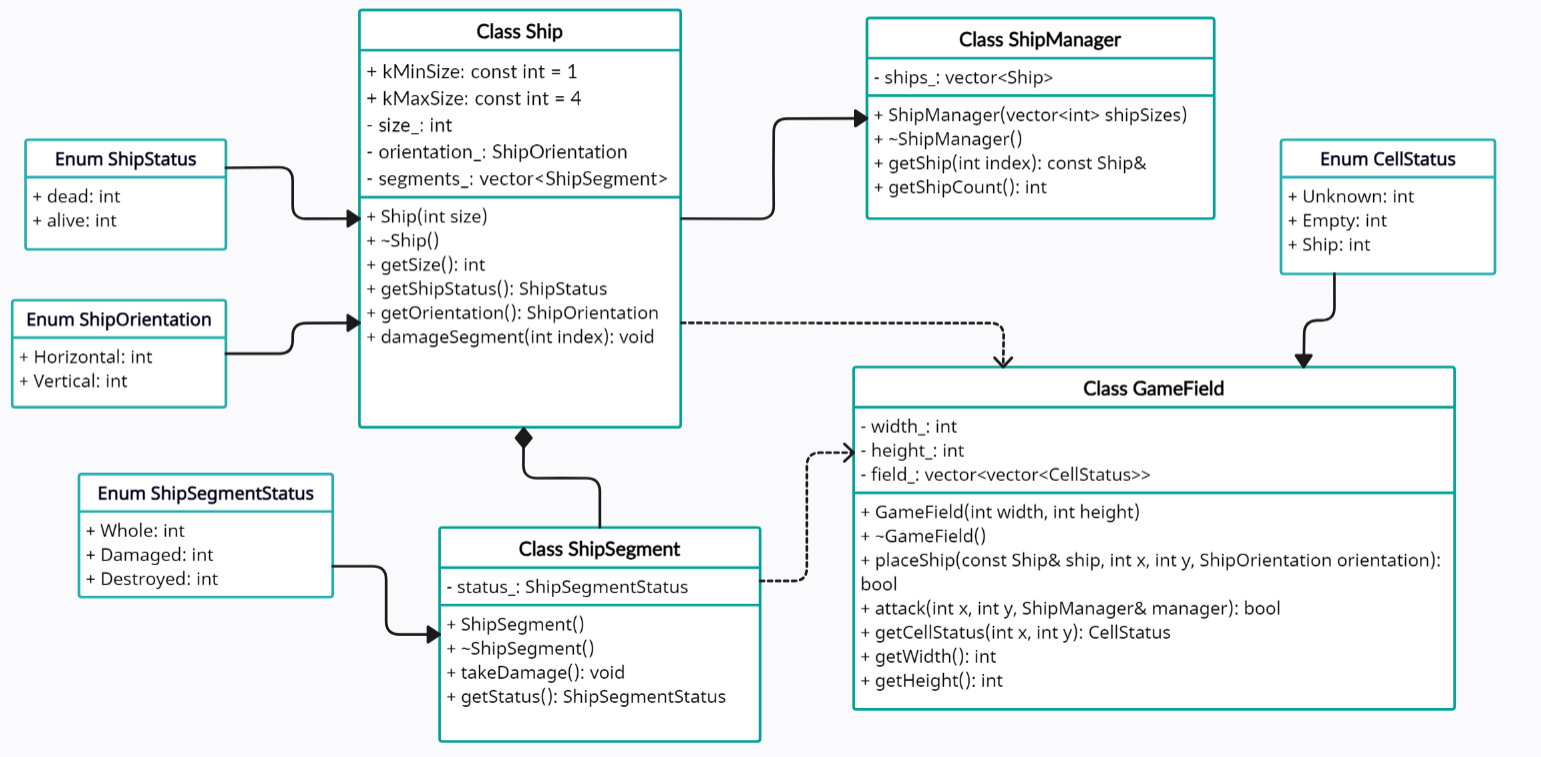
Вспомогательные методы:

1*. bool isWithinBounds(int x, int y) const:* Проверяет, находится ли клетка в пределах поля.

2. *bool canPlaceShip(const Ship& ship, int x, int y, ShipOrientation orientation) const*: Проверяет, можно ли разместить корабль в указанных координатах без пересечений.

3. *void markShipOnField(const Ship& ship, int x, int y, ShipOrientation orientation):* Помечает клетки на поле, занятые кораблём.

Ниже представлена UML-диаграмма реализованных в данной работе классов:



Разработанный программный код см. в приложении А.

Тестирование программы см. в приложении Б.

## Выводы

Цель работы – создание классов для моделирования корабля, менеджера кораблей и игрового поля, а также в реализации их взаимодействия. В ходе выполнения задачи были созданы все необходимые классы, каждый из которых соответствует требованиям задания. Также была реализована программа, которая позволила протестировать корректность работы классов, в частности, проверку размещения кораблей на игровом поле, взаимодействие между кораблями и полем, а также выполнение атак по игровому полю. Таким образом, поставленная цель работы была успешно достигнута.

# Приложение А Исходный код программы

**Название файла: ship.h**

#ifndef SHIP\_H

#define SHIP\_H

#include <vector>

// maybe 'v' and 'h' instead

enum class ShipOrientation : int {

Horizontal = 0,

Vertical = 1

};

enum class ShipStatus : int {

dead = 0,

alive = 1

};

// explicit assignment is optional

enum class ShipSegmentStatus : int {

Whole = 0,

Damaged = 1,

Destroyed = 2

};

class Ship {

public:

explicit Ship(int size); // constructor, disables implicit conversion

~Ship() = default; // destructor

// modifierс const - no change state of object

int getSize() const noexcept;

// ship dead or alive

ShipStatus getShipStatus() const noexcept;

// field orientation horizontal (0) or vertical (1)

ShipOrientation getOrientation() const noexcept;

// cause damage on segment

void damageSegment(int index);

static const int kMinSize = 1;

static const int kMaxSize = 4;

private:

class ShipSegment {

public:

ShipSegment();

~ShipSegment() = default;

// take 1 damage by index

void takeDamage();

// segment whole, damaged, destroyed

ShipSegmentStatus getStatus() const noexcept;

private:

ShipSegmentStatus status\_;

};

int size\_; // ship size 1-4

ShipOrientation orientation\_;

std::vector<ShipSegment> segments\_;

};

#endif // SHIP\_H

**Названиефайла: ship.cpp**

#include "ship.h"

#include <stdexcept>

// constructor

Ship::Ship(int size) : size\_(size), orientation\_(ShipOrientation::Horizontal) {

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

throw std::out\_of\_range("The ship size must be from 1 to 4");

}

// Initialization of industry segments

segments\_.resize(size\_);

}

int Ship::getSize() const noexcept {

return size\_;

}

ShipStatus Ship::getShipStatus() const noexcept {

// Check all segments. If all destroyed, ship is dead.

for (const auto& segment : segments\_) {

if (segment.getStatus() != ShipSegmentStatus::Destroyed) {

return ShipStatus::alive;

}

}

return ShipStatus::dead;

}

ShipOrientation Ship::getOrientation() const noexcept {

return orientation\_;

}

void Ship::damageSegment(int index) {

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

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

}

segments\_[index].takeDamage();

}

// Ship segment constructor

Ship::ShipSegment::ShipSegment() : status\_(ShipSegmentStatus::Whole) {}

void Ship::ShipSegment::takeDamage() {

status\_ = ShipSegmentStatus::Destroyed;

}

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

return status\_;

}

**Названиефайла: shipManager.h**

#ifndef SHIP\_MANAGER\_H

#define SHIP\_MANAGER\_H

#include "ship.h"

#include <vector>

#include <stdexcept>

class ShipManager {

public:

// Конструктор, принимающий количество кораблей и их размеры

ShipManager(const std::vector<int>& shipSizes);

// Получить корабль по индексу

const Ship& getShip(int index) const;

// Получить количество кораблей

int getShipCount() const noexcept;

private:

std::vector<Ship> ships\_;

};

#endif // SHIP\_MANAGER\_H}

**Названиефайла: field.h**

#ifndef GAME\_FIELD\_H

#define GAME\_FIELD\_H

#include "shipManager.h"

#include <vector>

enum class CellStatus {

Unknown = 0,

Empty = 1,

Ship = 2

};

class GameField {

public:

// constructor

GameField(int width, int height);

// copy constructor

GameField(const GameField& other);

// move constructor

GameField(GameField&& other) noexcept;

// copy assignment operator

GameField& operator=(const GameField& other);

// move assignment operator

GameField& operator=(GameField&& other) noexcept;

bool placeShip(const Ship& ship, int x, int y, ShipOrientation orientation);

bool attack(int x, int y, ShipManager& shipManager);

CellStatus getCellStatus(int x, int y) const;

int getWidth() const noexcept { return width\_; }

int getHeight() const noexcept { return height\_; }

~GameField();

private:

int width\_;

int height\_;

std::vector<std::vector<CellStatus>> field\_;

// help methods

bool isWithinBounds(int x, int y) const;

bool canPlaceShip(const Ship& ship, int x, int y, ShipOrientation orientation) const;

void markShipOnField(const Ship& ship, int x, int y, ShipOrientation orientation);

void deepCopy(const GameField& other); // Method for deep copying

};

#endif // GAME\_FIELD\_H

**Названиефайла: field.cpp**

#include "field.h"

#include <stdexcept>

// constructor

GameField::GameField(int width, int height) : width\_(width), height\_(height), field\_(height, std::vector<CellStatus>(width, CellStatus::Unknown)) {

if (width <= 0 || height <= 0) {

throw std::invalid\_argument("Invalid field sizes");

}

}

// copy constructor

GameField::GameField(const GameField& other) {

deepCopy(other);

}

// move constructor

GameField::GameField(GameField&& other) noexcept

: width\_(other.width\_), height\_(other.height\_), field\_(std::move(other.field\_)) {

other.width\_ = 0;

other.height\_ = 0;

}

// copy assignment operator

GameField& GameField::operator=(const GameField& other) {

if (this != &other) {

deepCopy(other);

}

return \*this;

}

// move assignment operator

GameField& GameField::operator=(GameField&& other) noexcept {

if (this != &other) {

width\_ = other.width\_;

height\_ = other.height\_;

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

other.width\_ = 0;

other.height\_ = 0;

}

return \*this;

}

// deep copying

// потому что класс поле содержит указатели на динамически выделенную память

void GameField::deepCopy(const GameField& other) {

width\_ = other.width\_;

height\_ = other.height\_;

field\_ = other.field\_;

}

bool GameField::placeShip(const Ship& ship, int x, int y, ShipOrientation orientation) {

if (!canPlaceShip(ship, x, y, orientation)) {

return false;

}

markShipOnField(ship, x, y, orientation);

return true;

}

// Метод для проверки, можно ли разместить корабль

bool GameField::canPlaceShip(const Ship& ship, int x, int y, ShipOrientation orientation) const {

int shipSize = ship.getSize();

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

int currentX = x + (orientation == ShipOrientation::Horizontal ? i : 0);

int currentY = y + (orientation == ShipOrientation::Vertical ? i : 0);

if (!isWithinBounds(currentX, currentY) || field\_[currentY][currentX] != CellStatus::Empty) {

return false;

}

for (int dx = -1; dx <= 1; ++dx) {

for (int dy = -1; dy <= 1; ++dy) {

int neighborX = currentX + dx;

int neighborY = currentY + dy;

if (isWithinBounds(neighborX, neighborY) && field\_[neighborY][neighborX] == CellStatus::Ship) {

return false;

}

}

}

}

return true;

}

// marking cells occupied by ship

void GameField::markShipOnField(const Ship& ship, int x, int y, ShipOrientation orientation) {

int shipSize = ship.getSize();

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

int currentX = x + (orientation == ShipOrientation::Horizontal ? i : 0);

int currentY = y + (orientation == ShipOrientation::Vertical ? i : 0);

field\_[currentY][currentX] = CellStatus::Ship;

}

}

bool GameField::attack(int x, int y, ShipManager& shipManager) {

if (!isWithinBounds(x, y)) {

throw std::out\_of\_range("Attack outside the field");

}

if (field\_[y][x] == CellStatus::Ship) {

field\_[y][x] = CellStatus::Empty;

return true;

}

// miss atack

return false;

}

// cell is in field

bool GameField::isWithinBounds(int x, int y) const {

return x >= 0 && x < width\_ && y >= 0 && y < height\_;

}

CellStatus GameField::getCellStatus(int x, int y) const {

if (!isWithinBounds(x, y)) {

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

}

return field\_[y][x];

}

// destructor

GameField::~GameField() {}

**Файл ShipManager.cpp**

#include " shipManager.h"

#include " ship.h"

// constructor

ShipManager::ShipManager(const std::vector<int>& shipSizes) {

for (int size = 0; size < shipSizes.size(); size++) {

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

throw std::invalid\_argument("Ship size must be from 1 to 4");

}

Ship ship(size);

ships\_.push\_back(ship);

}

}

// get ship by index

const Ship& ShipManager::getShip(int index) const {

if (index < 0 || index >= static\_cast<int>(ships\_.size())) {

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

}

return ships\_[index];

}

// get count of ship

int ShipManager::getShipCount() const noexcept {

return static\_cast<int>(ships\_.size());

}

# Приложение Б Тестирование программы

Название файла: main.cpp

#include "ship.h"

#include "shipManager.h"

#include "field.h"

#include <iostream>

int main() {

// cоздаем менеджер кораблей с заданными размерами

std::vector<int> shipSizes = {4, 3, 2, 1, 3};

ShipManager manager(shipSizes);

// test

std::cout << "count of ship: ", manager.getShipCount() << '\n';

std::cout << manager.getShip(0).getSize() << '\n';

std::cout << manager.getShip(1).getSize() << '\n';

std::cout << manager.getShip(2).getSize() << '\n';

std::cout << manager.getShip(3).getSize() << '\n';

std::cout << manager.getShip(4).getSize() << '\n';

// cоздаем игровое поле размером 10x10

GameField field(10, 10);

// Размещаем корабли на поле

std::cout << "Размещение кораблей на поле..." << std::endl;

// Размещаем корабль с размером 4 на позиции (1, 1) горизонтально

if (field.placeShip(manager.getShip(1), 1, 1, ShipOrientation::Horizontal)) {

std::cout << "Ok (1, 1)" << std::endl;

} else {

std::cout << "Error (1, 1)" << std::endl;

}

// // Размещаем корабль с размером 3 на позиции (3, 5) вертикально

if (field.placeShip(manager.getShip(1), 3, 5, ShipOrientation::Vertical)) {

std::cout << "Ok (3, 5)" << std::endl;

} else {

std::cout << "Error (3, 5)" << std::endl;

}

// Проводим атаку по полю

std::cout << "\nАтака по полю..." << std::endl;

// (1, 1) — это попадание

if (field.attack(1, 1, manager)) {

std::cout << "hit (1, 1)" << std::endl;

} else {

std::cout << "mis (1, 1)" << std::endl;

}

// (0, 0) — это промах

if (field.attack(0, 0, manager)) {

std::cout << "hit (0, 0)" << std::endl;

} else {

std::cout << "mis (0, 0)" << std::endl;

}

// (3, 5) — это попадание

if (field.attack(3, 5, manager)) {

std::cout << "hit (3, 5)" << std::endl;

} else {

std::cout << "mis (3, 5)" << std::endl;

}

// (2, 1) — это попадание

if (field.attack(2, 1, manager)) {

std::cout << "hit (2, 1)" << std::endl;

} else {

std::cout << "mis (2, 1)" << std::endl;

}

// Проверяем статус клеток

std::cout << "\nПроверка статуса клеток на поле..." << std::endl;

std::cout << "Клетка (1, 1): " << static\_cast<int>(field.getCellStatus(1, 1)) << std::endl;

std::cout << "Клетка (0, 0): " << static\_cast<int>(field.getCellStatus(0, 0)) << std::endl;

std::cout << "Клетка (3, 5): " << static\_cast<int>(field.getCellStatus(3, 5)) << std::endl;

std::cout << "Клетка (2, 1): " << static\_cast<int>(field.getCellStatus(2, 1)) << std::endl;

return 0;

}

Результат работы программы:

