**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №1**

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

# **Тема: Создание классов.**

|  |  |
| --- | --- |
| Студент гр. 3382 | Миллер С. Е. |
| Преподаватель | Жангиров Т. Р. |

Санкт-Петербург

2024

**Цель работы:** Освоить работу с классами на языке С++. Изучить UML диаграммы.

**Задание:**

1. 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, повреждён, уничтожен. Изначально у корабля все сегменты целые. При нанесении 1 урона по сегменту, он становится повреждённым, а при нанесении 2 урона по сегменту, уничтоженным. Также добавить методы для взаимодействия с кораблём.
2. Создать класс менеджера кораблей, хранящий информацию о кораблях. Данный класс в конструкторе принимает количество кораблей и их размеры, которые нужно расставить на поле.
3. 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:

1. неизвестно (изначально вражеское поле полностью неизвестно),
2. пустая (если на клетке ничего нет)
3. 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:*

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

**Теоретические сведения:**

Классы в С ++ - это пользовательский тип данных, который объединяет данные и функции, работающие с этими данными, в одну сущность. Классы используются для моделирования объектов и представляет основу объектно-ориентированного программирования (ООП).

Структура класса:

Класс состоит из полей (данных) и методов (функций). Поля хранят состояние объекта, а методы определяют его поведение.

Каждый класс имеет конструктор - специальный метод класса, который вызывается автоматически при создании объекта. Конструкторы используются для инициализации полей объекта.

Особенности конструктора:

1. Имеет то же имя, что и класс;
2. Не имеет типа возврата;
3. Может иметь параметры (перегрузка конструкторов).

Также класс имеет деструктор - специальный метод класса, который вызывается автоматически при уничтожении объекта. Деструкторы используются для освобождения ресурсов, занимаемых объектом (например, динамической памяти, файловых дескрипторов).

Особенности деструктора:

1. Имеет то же имя, что и класс, но с префиксом ~;
2. Не имеет типа возврата и параметров.
3. Может быть только один деструктор для класса.

Основные компоненты класса

1. Поля (данные): Хранят состояние объекта. Обычно объявляются как private для обеспечения инкапсуляции.
2. Методы (функции): Определяют поведение объекта.
3. Конструкторы и деструкторы: Специальные методы для инициализации и очистки объекта.
4. Модификаторы доступа:
   1. public: Доступные из любого места.
   2. private: Доступные только внутри класса.
   3. protected: Доступные внутри класса и его наследников.
5. Наследование: Возможность создавать новый класс на основе существующего, наследуя его поля и методы.
6. Полиморфизм: Способность объектов разных классов обрабатывать вызовы методов по-разному.
7. Инкапсуляция: Сокрытие внутреннего состояния объекта и предоставление доступа через методы.

**Выполнение работы**

1. ***Класс Ship:***

*Приватные поля класса:*

length\_: длина корабля (целое число от 1 до 4).

orientation\_: ориентация корабля (горизонтальная по умолчанию, может быть определена как ShipOrientation).

segments\_: вектор, представляющий состояние сегментов корабля (каждый сегмент может иметь статус SegmentStatus, который может быть целым, поврежденным или уничтоженным).

index: индекс корабля в массиве кораблей (необязательный атрибут, используется для управления кораблем в игре).

*Публичные методы:*

Конструктор Ship(int length): инициализирует объект Ship с заданной длиной. Если длина не находится в диапазоне от 1 до 4, выводится сообщение об ошибке, и длина не устанавливается. Сегменты инициализируются в состоянии Intact.

int GetLength() const: возвращает длину корабля.

ShipOrientation GetOrientation() const: возвращает ориентацию корабля.

SegmentStatus GetSegmentStatus(int index) const: возвращает статус (состояние) сегмента по указанному индексу.

void DamageSegment(int index): наносит урон сегменту по указанному индексу. Если сегмент целый, он становится поврежденным; если поврежденный — уничтоженным. Если индекс сегмента недопустим, выводится сообщение об ошибке.

bool IsDestroyed() const: проверяет, разрушен ли корабль. Если все сегменты корабля находятся в состоянии Destroyed, метод возвращает true; в противном случае — false.

int GetShipIndex() const: возвращает индекс корабля.

void SetShipIndex(int index): устанавливает индекс корабля.

1. ***Класс ShipManager:***

Класс Ship представляет собой корабль в игре "Морской бой". Он имеет определенную длину, ориентацию и массив сегментов, каждый из которых может находиться в одном из трех состояний: целый, поврежденный или уничтоженный.

Основные элементы класса Ship:

*Приватные поля класса:*

length\_: длина корабля (целое число от 1 до 4).

orientation\_: ориентация корабля (горизонтальная по умолчанию, может быть определена как ShipOrientation).

segments\_: вектор, представляющий состояние сегментов корабля (каждый сегмент может иметь статус SegmentStatus, который может быть целым, поврежденным или уничтоженным).

index: индекс корабля в массиве кораблей (необязательный атрибут, используется для управления кораблем в игре).

*Публичные методы:*

Конструктор Ship(int length): инициализирует объект Ship с заданной длиной. Если длина не находится в диапазоне от 1 до 4, выводится сообщение об ошибке, и длина не устанавливается. Сегменты инициализируются в состоянии Intact.

int GetLength() const: возвращает длину корабля.

ShipOrientation GetOrientation() const: возвращает ориентацию корабля.

SegmentStatus GetSegmentStatus(int index) const: возвращает статус (состояние) сегмента по указанному индексу.

void DamageSegment(int index): наносит урон сегменту по указанному индексу. Если сегмент целый, он становится поврежденным; если поврежденный — уничтоженным. Если индекс сегмента недопустим, выводится сообщение об ошибке.

bool IsDestroyed() const: проверяет, разрушен ли корабль. Если все сегменты корабля находятся в состоянии Destroyed, метод возвращает true; в противном случае — false.

int GetShipIndex() const: возвращает индекс корабля.

void SetShipIndex(int index): устанавливает индекс корабля

***3.Класс GameField***

Класс GameField представляет игровое поле для игры "Морской бой". Он управляет размещением кораблей, обработкой атак и отображением состояния поля. Класс включает методы для создания поля, размещения кораблей, выполнения атак и отображения состояния игрового поля.

Основные элементы класса GameField:

*Приватные поля класса::*

width: ширина игрового поля.

height: высота игрового поля.

field: двумерный вектор, представляющий состояние каждой клетки на поле (типы клеток, такие как Unknown, Empty, Ship).

occupied\_cells: вектор, содержащий координаты занятых сегментов кораблей.

*Публичные методы:*

Конструктор GameField(int width, int height): инициализирует игровое поле с заданной шириной и высотой. Если размеры недопустимы (меньше 1), выводится сообщение об ошибке. Поле инициализируется значениями Unknown.

Конструктор копирования GameField(const GameField &other): создает новый объект GameField, копируя данные из другого объекта.

Оператор присваивания копированием GameField& operator=(const GameField& other): присваивает значения из одного объекта GameField другому, если они не совпадают.

Конструктор перемещения GameField(GameField &&other) noexcept: перемещает ресурсы из другого объекта GameField, используя std::swap для обмена значениями.

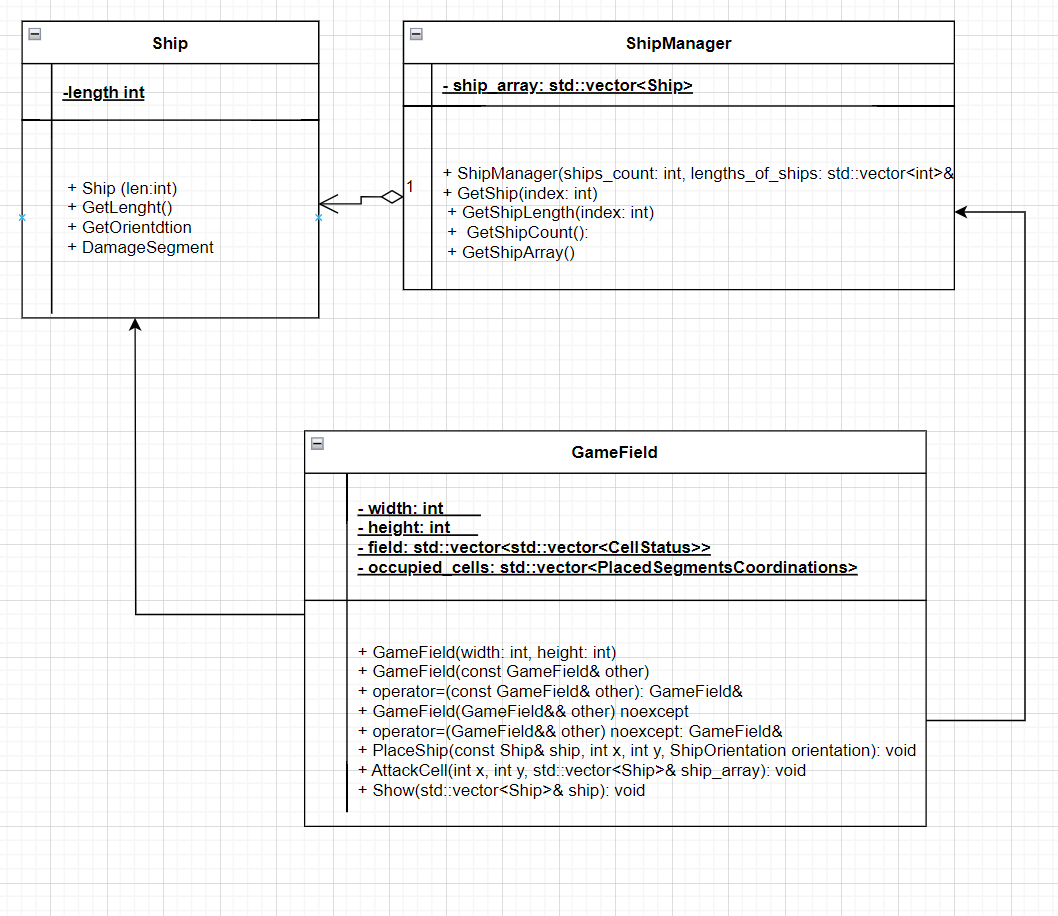
Оператор присваивания перемещением GameField& operator=(GameField&& other) noexcept: присваивает ресурсы из одного объекта GameField другому, используя перемещение.

void PlaceShip(const Ship& ship, int x, int y, ShipOrientation orientation): размещает корабль на игровом поле. Проверяет, не выходит ли корабль за границы поля и не пересекается ли с другими кораблями. Если размещение невозможно, выводит соответствующее сообщение.

void AttackCell(int x, int y, std::vector<Ship> &ship\_array): обрабатывает атаку по указанным координатам. Если координаты выходят за границы поля, выводится сообщение об ошибке. Если атака попадает в занятый сегмент, вызывается метод DamageSegment у соответствующего корабля.

void Show(std::vector<Ship> &ships): отображает текущее состояние игрового поля. Каждая клетка отображается в зависимости от ее состояния: пустая клетка — [ ], неизвестная клетка — [?], поврежденный сегмент — [#], уничтоженный сегмент — [X], целый сегмент — [W].

***UML - диаграмма:***

******

`

**Вывод**

В ходе лабораторной работы удалось изучить классы на С++ и разобраться с UML - диаграммами. Также были созданы классы FlieldSegment, Ship, ShipManager, Flield, которые помогут в дальнейшем реализовать игру «Морской бой».

**Приложение 1**

*Файл GameField.cpp:*

##include "GameField.h"

#include <iostream>

GameField::GameField(int width, int height) : width(width), height(height) {

if (width < 1 || height < 1) {

std::cout << "Field width and height must be greater than 0.";

return;

}

field.resize(height, std::vector<CellStatus>(width, CellStatus::Unknown));

}

//конструктор копирования

GameField::GameField(const GameField &other)

{

width = other.width;

height = other.height;

field = other.field;

occupied\_cells = other.occupied\_cells;

}

//оперотор присваивания копирования

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

{

if (this != &other)

{

width = other.width;

height = other.height;

field = other.field;

occupied\_cells = other.occupied\_cells;

}

return \*this;

}

// Конструктор перемещения

GameField::GameField(GameField &&other) noexcept

{

std::swap(width, other.width);

std::swap(height, other.height);

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

occupied\_cells = std::move(other.occupied\_cells);

}

// Оператор присваивания перемещением

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

{

if (this != &other)

{

std::swap(width, other.width);

std::swap(height, other.height);

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

occupied\_cells = std::move(other.occupied\_cells);

}

return \*this;

}

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

{

//проверка на выход за границы

if (x < 0 || y >= height || x >= width || y < 0

|| (orientation == ShipOrientation::Horizontal && x + ship.GetLength() >= width)

|| (orientation == ShipOrientation::Vertical && y + ship.GetLength() >= height))

{

std::cout << ship.GetShipIndex() << " The ship out of bounds" << '\n';

return;

}

//проерка на пересечение или соприкосновение кораблей

for (int i = 0; i < ship.GetLength(); i++){

if (orientation == ShipOrientation::Horizontal){

if(field[y][x + i] == CellStatus::Ship

|| (y + 1 < height && field[y + 1][x + i] == CellStatus::Ship)

|| (y - 1 >= 0 && field[y - 1][x + i] == CellStatus::Ship))

{

std::cout << ship.GetShipIndex() << " Ship intersects or touches with other ships" << '\n';

return;

}

}

else{

if (field[y + i][x] == CellStatus::Ship

|| (x + 1 < width && field[y + i][x + 1] == CellStatus::Ship)

|| (x - 1 >= 0 && field[y + i][x - 1] == CellStatus::Ship))

{

std::cout << "Ship intersects or touches with other ships" << '\n';

return;

}

}

}

for (int i = 0; i < ship.GetLength(); i++)

{

if (orientation == ShipOrientation::Horizontal)

{

occupied\_cells.emplace\_back(PlacedSegmentsCoordinations{.x = x + i, .y = y, .ship\_index = ship.GetShipIndex(), .segment\_index = i});

field[y][x + i] = CellStatus::Ship;

}

else

{

occupied\_cells.emplace\_back(PlacedSegmentsCoordinations{.x = x, .y = y + i, .ship\_index = ship.GetShipIndex(), .segment\_index = i});

field[y + i][x] = CellStatus::Ship;

}

}

}

void GameField::AttackCell(int x, int y,std::vector<Ship> &ship\_array)

{

if (x >= width || x < 0 || y >= height || y < 0)

{

std::cout << "You can't attack coordinates out of bounds" << '\n';

return;

}

for (auto &cell : occupied\_cells)

{

if (cell.x == x && cell.y == y)

{

ship\_array[cell.ship\_index].DamageSegment(cell.segment\_index);

return;

}

}

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

}

void GameField::Show(std::vector<Ship> &ships)

{

for (int y = 0; y < height ; y++)

{

for (int x = 0; x < width; x++)

{

if(field[y][x] == CellStatus::Empty){

std::cout<<"[ ] ";

}

else if (field[y][x] == CellStatus::Unknown)

{

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

}

else{

for (int i = 0; i < (int) occupied\_cells.size(); i++){

if (occupied\_cells[i].x == x && occupied\_cells[i].y == y){

SegmentStatus segstatus = ships[occupied\_cells[i].ship\_index].GetSegmentStatus(occupied\_cells[i].segment\_index);

if (segstatus == SegmentStatus::Damaged){

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

}

else if (segstatus == SegmentStatus::Destroyed){

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

}

else{

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

}

}

}

}

}

std::cout << std::endl;

}

}

*Файл GameField.h:*

##pragma once

#include <vector>

#include <stdexcept>

#include "Ship.h"

#include "ShipManager.h"

struct PlacedSegmentsCoordinations

{

int x;

int y;

int ship\_index;

int segment\_index;

};

class GameField {

public:

GameField(int width, int height);

GameField(const GameField& other);

GameField& operator = (const GameField& other);

GameField(GameField&& other) noexcept;

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

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

void AttackCell(int x, int y, std::vector<Ship> &ship\_array);

void Show(std::vector<Ship> &ship);

private:

int width;

int height;

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

std::vector<PlacedSegmentsCoordinations> occupied\_cells;

};

*Файл Ship.cpp:*

#include <iostream>

#include "Ship.h"

Ship::Ship(int length)

: length\_(length), orientation\_(ShipOrientation::Horizontal) {

if (length\_ > 4 ||length\_ < 1) {

std::cout << "Ship length must be between 1 and 4 inclusive." << '\n';

return;

}

segments\_.resize(length, SegmentStatus::Intact);

}

int Ship::GetLength() const { return length\_; }

ShipOrientation Ship::GetOrientation() const { return orientation\_; }

SegmentStatus Ship::GetSegmentStatus(int index) const {

return segments\_[index];

}

void Ship::DamageSegment(int index) {

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

std::cout << "Invalid segment index." << '\n';

return;

}

if (segments\_[index] == SegmentStatus::Intact) {

segments\_[index] = SegmentStatus::Damaged;

}

else if (segments\_[index] == SegmentStatus::Damaged) {

segments\_[index] = SegmentStatus::Destroyed;

}

}

bool Ship::IsDestroyed() const {

for (auto status : segments\_) {

if (status != SegmentStatus::Destroyed) {

return false;

}

}

return true;

}

int Ship::GetShipIndex() const { return index; }

void Ship::SetShipIndex(int index) { this->index = index; }

*Файл Ship.h:*

#pragma once

#include <vector>

#include <stdexcept>

#include <string>

enum class CellStatus { Empty, Unknown, Ship };

enum class ShipOrientation { Horizontal, Vertical };

enum class SegmentStatus { Intact, Damaged, Destroyed };

class Ship {

public:

Ship(int length);

// Возвращает длину корабля

int GetLength() const;

// Возвращает ориентацию корабля

ShipOrientation GetOrientation() const;

// Возвращает статус сегмента по индексу

SegmentStatus GetSegmentStatus(int index) const;

// Наносит урон сегменту корабля по индексу

void DamageSegment(int index);

// Проверяет, уничтожен ли корабль

bool IsDestroyed() const;

int GetShipIndex() const;

void SetShipIndex(int index);

private:

int length\_;

int index;

ShipOrientation orientation\_;

std::vector<SegmentStatus> segments\_;

};

*Файл ShipManager.cpp:*

##include <iostream>

#include "ShipManager.h"

ShipManager::ShipManager(int ships\_count, std::vector<int>& lengths\_of\_ships)

{

for (int i = 0; i < ships\_count; i++) {

ship\_array.emplace\_back(lengths\_of\_ships[i]);

ship\_array[i].SetShipIndex(i);

}

}

Ship& ShipManager::GetShip(int index)

{

return ship\_array[index];

}

// Возвращает длину корабля по индексу

int ShipManager::GetShipLength(int index) const {

if (index < 0 || index >= (int) ship\_array.size()) {

std::cout<< "Invalid ship index.";

return 0;

}

return ship\_array[index].GetLength();

}

int ShipManager::GetShipCount() const

{

return ship\_array.size();

}

std::vector<Ship>& ShipManager::GetShipArray()

{

return ship\_array;

}

*Файл ShipManager.h:*

#pragma once

#include <vector>

#include "Ship.h"

class ShipManager {

public:

ShipManager(int ships\_count, std::vector<int>& lengths\_of\_ships);

// Возвращает ссылку на корабль по индексу

Ship& GetShip(int index);

// Возвращает длину корабля по индексу

int GetShipLength(int index) const;

int GetShipCount() const;

std::vector<Ship>& GetShipArray();

private:

std::vector<Ship> ship\_array;

};

*Файл main.cpp:*

#// Функция main

#include <iostream>

#include <vector>

#include <string>

#include "GameField.h"

#include "ShipManager.h"

int main()

{

// Пример использования классов

std::vector<int> ship\_lengths = {3, 2, 3, 2, 7};

ShipManager ship\_manager(5, ship\_lengths);

GameField my\_field(10, 10);

// Размещение кораблей на поле

my\_field.PlaceShip(ship\_manager.GetShip(0), 1, 1, ShipOrientation::Vertical);

my\_field.PlaceShip(ship\_manager.GetShip(1), 3, 3, ShipOrientation::Horizontal);

my\_field.PlaceShip(ship\_manager.GetShip(2), 9, 8, ShipOrientation::Vertical);

my\_field.PlaceShip(ship\_manager.GetShip(3), 1, 2, ShipOrientation::Horizontal);

my\_field.PlaceShip(ship\_manager.GetShip(4), 0, 5, ShipOrientation::Horizontal);

// Пример атаки

my\_field.AttackCell(1, 1, ship\_manager.GetShipArray());

my\_field.AttackCell(1, 1, ship\_manager.GetShipArray());

my\_field.AttackCell(3, 3, ship\_manager.GetShipArray());

my\_field.Show(ship\_manager.GetShipArray());

return 0;

}

*Файл Makefile:*

# Compiler and flags

CXX = g++

CXXFLAGS = -std=c++20 -Wall -pedantic

# Object files

OBJECTS = main.o Ship.o ShipManager.o GameField.o

# Target

TARGET = game

# Rules

all: $(OBJECTS)

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

main.o: main.cpp Ship.h ShipManager.h GameField.h

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

ship.o: Ship.cpp Ship.h

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

ship\_manager.o: ShipManager.cpp ShipManager.h Ship.h

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

game\_field.o: GameField.cpp GameField.h Ship.h

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

clean:

rm -f \*.o $(TARGET)