

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. 3342

Львов А.В.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Разработать систему классов для моделирования игры "Морской бой", включая классы кораблей, менеджера кораблей и игрового поля. Классы должны обеспечивать корректное размещение кораблей на поле, обработку атак и отслеживание состояния кораблей и поля.

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.

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

а. 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:

- і. неизвестно (изначально вражеское поле полностью неизвестно),
- і. пустая (если на клетке ничего нет)
- і. 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

Класс `Field` хранит в себе высоту и ширину поля, а также указатели на клетки. Метод `isValidPlace` принимает координаты, на которые следует поставить корабль, его длину и расположение (вертикальное или горизонтальное). Она проверяет, можно ли разместить корабль на заданных координатах и возвращает в зависимости от этого `true` или `false`. Метод `placeShip` принимает координаты, ссылку на корабль и его расположение. Если можно разместить, то размещает его на поле. Метод `attack` принимает координаты, по которым необходимо произвести выстрел. Если в данной клетке находится сегмент, то его здоровье уменьшается, в ином случае печатается “Miss”. После этого клетка становится видимой для игрока, и показывается поле. Метод `setVisibility` делает видимыми все клетки поля. Метод `printField` выводит поле на экран.

Класс `Cell` хранит координаты, значение клетки, поле `visible`, отвечающее за то, является ли эта клетка видимой или нет, и указатель на сегмент, размещенный в этой клетке. Метод `isVisible` возвращает поле `visible`. Метод `setVisibility` устанавливает `visible`, равный `true`. Метод `setValue` устанавливает в поле `value` переданное значение. Метод `setSegment` устанавливает в поле `segment` переданное значение. Метод `getValue` возвращает `value`. Метод `getSegment` возвращает указатель на `segment`.

Класс `ShipSegment` хранит очки здоровья, состояние и координаты. Метод `getHP` возвращает количество очков здоровья. Метод `damage` устанавливает новое состояние сегмента в зависимости от текущего и уменьшает количество очков здоровья. Метод `printInfo` печатает информацию о сегменте. Метод `setCoordinates` устанавливает в поле `coordinates` переданные значения. Метод `setState` устанавливает в поле `State` переданное состояние.

Класс Ship хранит указатели на сегменты, длину, расположение и координаты головного сегмента. Метод `getLength` возвращает длину. Метод `getOrientation` возвращает расположение корабля. Метод `getSegments` возвращает ссылки на сегменты корабля. Метод `setCoordinates` устанавливает координаты головного сегмента. Метод `getCoordinates` возвращает координаты головного сегмента корабля. Метод `printInfoShip` печатает информацию о корабле.

Класс ShipManager хранит вектор кораблей. Метод `getShips` возвращает корабли. Метод `printShips` печатает информацию о каждом корабле.

Разработанный программный код см. в приложении А.

Тестирование

Для проверки работоспособности программы, был инициализирован менеджер кораблей с размерами кораблей 4, 3, 2 и 1. Затем эти корабли были размещены на поле и по некоторым сегментам была произведена атака.

```
int main() {
    std::vector<int> lengths = {4, 3, 2, 1};
    try {
        ShipManager manager( sizes: lengths);
        Field field;
        auto ships : vector<shared_ptr<Ship>> = manager.getShips();
        try {
            field.placeShip( coordinates: { .x: 1, .y: 1}, ship: ships[0], orientation: Orientation::Horizontal);
            field.placeShip( coordinates: { .x: 3, .y: 4}, ship: ships[1]);
            field.placeShip( coordinates: { .x: 6, .y: 8}, ship: ships[2], orientation: Orientation::Horizontal);
            field.placeShip( coordinates: { .x: 1, .y: 7}, ship: ships[3]);
            field.printField();
        }
        catch (std::invalid_argument) {
            std::cout << "Cannot place this segments!" << std::endl;
        }
        field.attack( coordinates: { .x: 2, .y: 2});
        field.attack( coordinates: { .x: 1, .y: 1});
        field.attack( coordinates: { .x: 2, .y: 1});
        field.printField();
    }
    catch (std::invalid_argument) {
        std::cout << "Invalid length of ship!";
    }
}
```

```

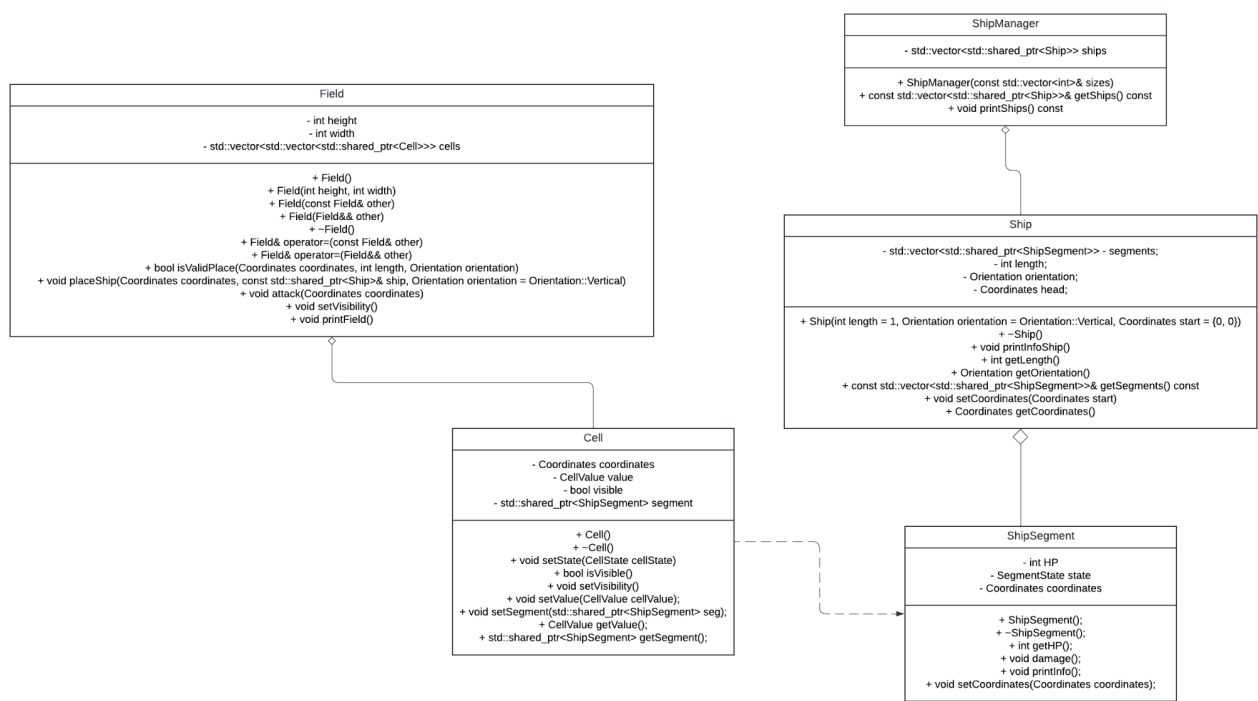
./Battleship

~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~

Miss
Hit
Hit

~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ W W ~ ~ ~ ~ ~ ~
~ ~ . ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~

```



Выводы

В ходе выполнения лабораторной работы, был реализован функционал добавления кораблей заданных размеров на игровое поле, создания игрового поля с заданными размерами, а также атаку по клетке поля и повреждение сегмента. Программа была успешно протестирована. Была реализована UML-диаграмма классов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "../include/ShipSegment.h"
#include "../include/Ship.h"
#include "../include/Field.h"
#include "../include/ShipManager.h"

int main() {
    std::vector<int> lengths = {4, 3, 2, 1};
    try {
        ShipManager manager(lengths);
        Field field;
        auto ships = manager.getShips();
        try {
            field.placeShip({1, 1}, ships[0], Orientation::Horizontal);
            field.placeShip({3, 4}, ships[1]);
            field.placeShip({6, 8}, ships[2], Orientation::Horizontal);
            field.placeShip({1, 7}, ships[3]);
            field.printField();
        }
        catch (std::invalid_argument) {
            std::cout << "Cannot place this segments!" << std::endl;
        }
        field.attack({2, 2});
        field.attack({1, 1});
        field.attack({2, 1});
        field.printField();
    }
    catch (std::invalid_argument) {
        std::cout << "Invalid length of ship!";
    }
}
```

Название файла: Cell.cpp

```
#include "../include/Cell.h"

Cell::Cell() : coordinates({0, 0}), visible(false), value(CellValue::Empty), segment(nullptr) {}

Cell::~~Cell() = default;

void Cell::setValue(CellValue cellValue) {
    value = cellValue;
}

bool Cell::isVisible() {
```

```

    return visible;
}

void Cell::setVisibility() {
    visible = true;
}

void Cell::setSegment(std::shared_ptr<ShipSegment> seg) {
    segment = seg;
}

CellValue Cell::getValue() {
    return value;
}

std::shared_ptr<ShipSegment> Cell:: getSegment() {
    return segment;
}

```

Название файла: Field.cpp

```

#include "../include/Field.h"
#include <iostream>

Field::Field() : height(10), width(10) {
    cells.resize(height);
    for (int i = 0; i < height; i++) {
        cells[i].resize(width);
        for (int j = 0; j < height; j++) {
            cells[i][j] = std::make_shared<Cell>();
        }
    }
}

Field::Field(int height, int width) : height(height), width(width){
    cells.resize(height);
    for (int i = 0; i < height; i++) {
        cells[i].resize(width);
        for (int j = 0; j < height; j++) {
            cells[i][j] = std::make_shared<Cell>();
        }
    }
}

Field::Field(const Field &other)
    : height(other.height),
      width(other.width),
      cells(other.cells){
}

```

```

Field::Field(Field &&other)
    : height(other.height), width(other.width),
      cells(std::move(other.cells)){}

Field &Field::operator=(const Field &other) {
    if (this != &other) {
        height = other.height;
        width = other.width;
        cells = other.cells;
    }
    return *this;
}

Field &Field::operator=(Field &&other) {
    if (this != &other) {
        height = other.height;
        width = other.width;
        cells = std::move(other.cells);
    }
    return *this;
}

void Field::printField() {
    const char* resetColor = "\033[0m";
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            CellValue value = cells[i][j]->getValue();
            if (cells[i][j] -> isVisible()) {
                std::cout << "\033[32m" << static_cast<char>(value) << resetColor << " ";
            }
            else {
                std::cout << "\033[34m" << "~" << resetColor << " ";
            }
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

bool Field::isValidPlace(Coordinates coordinates, int length, Orientation orientation) {
    int startX = coordinates.x;
    int startY = coordinates.y;

    if (orientation == Orientation::Vertical) {
        if (startY + length > height) {
            return false;
        }
    }
    else {
        if (startX + length > width) {
            return false;
        }
    }
}

```

```

    }
}

for (int segmentIndex = 0; segmentIndex < length; ++segmentIndex) {
    for (int yOffset = -1; yOffset <= 1; ++yOffset) {
        for (int xOffset = -1; xOffset <= 1; ++xOffset) {
            int checkY = startY + (orientation == Orientation::Vertical ? segmentIndex : 0) +
yOffset;
            int checkX = startX + (orientation == Orientation::Horizontal ? segmentIndex : 0)
+ xOffset;

            if (checkY >= 0 && checkY < height && checkX >= 0 && checkX < width) {
                if (cells[checkY][checkX]->getValue() != CellValue::Empty) {
                    return false;
                }
            }
        }
    }
}

return true;
}

```

```

void Field::placeShip(Coordinates coordinates, const std::shared_ptr<Ship>& ship,
Orientation orientation) {
    int length = ship->getLength();
    int x = coordinates.x;
    int y = coordinates.y;
    auto segments = ship->getSegments();
    if (isValidPlace(coordinates, length, orientation)) {
        ship ->setCoordinates(coordinates);
        for (int i = 0; i < length; i++) {
            if (orientation == Orientation::Vertical) {
                cells[y+i][x]->setValue(CellValue::Segment);
                cells[y+i][x]->setSegment(segments[i]);
            }
            else {
                cells[y][x+i]->setValue(CellValue::Segment);
                cells[y][x+i]->setSegment(segments[i]);
            }
        }
    }
    else {
        throw std::invalid_argument("Cannot place this ship!");
    }
}

```

```

void Field::attack(Coordinates coordinates) {
    int x = coordinates.x;
    int y = coordinates.y;

```

```

if (x < 0 || x > 9 || y < 0 || y > 9) {
    throw std::invalid_argument("Invalid coordinates!");
}

auto cell = cells[y][x];
auto value = cell->getValue();
auto segment = cell->getSegment();

switch (value) {
    case CellValue::Empty:
        std::cout << "Miss" << std::endl;
        cell -> setVisibility();
        break;
    case CellValue::Segment:
        std::cout << "Hit" << std::endl;
        cell->setValue(CellValue::Hit);
        cell -> setVisibility();
        segment->damage();
        break;
    case CellValue::Hit:
        std::cout << "Destroyed" << std::endl;
        cell->setValue(CellValue::Destroyed);
        segment->damage();
        break;
    case CellValue::Destroyed:
        std::cout << "Segment is already destroyed!";
        break;
}
}

void Field::setVisibility() {
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < height; j++) {
            cells[i][j]->setVisibility();
        }
    }
}

Field::~Field() = default;

```

Название файла: Ship.cpp

```
#include "../include/Ship.h"
```

```

Ship::Ship(int length, Orientation orientation, Coordinates start) : length(length),
orientation(orientation), head(start){
    for (int i = 0; i < length; i++) {
        segments.push_back(std::make_shared<ShipSegment>());
    }
}

```

```

}

int Ship::getLength() {
    return length;
}

const std::vector<std::shared_ptr<ShipSegment>>& Ship::getSegments() const {
    return segments;
}

Orientation Ship::getOrientation() {
    return orientation;
}

Coordinates Ship::getCoordinates() {
    return head;
}

void Ship::setCoordinates(Coordinates start) {
    head = start;
}

void Ship::printInfoShip() {
    for (const auto segment : segments) {
        segment->printInfo();
    }
}

Ship::~Ship() = default;

```

Название файла: ShipManager.cpp

```

#include "../include/ShipManager.h"

ShipManager::ShipManager(const std::vector<int>& lengths) {
    for (const auto& length : lengths) {
        if (length < 1 || length > 4) {
            throw std::invalid_argument("Invalid length of ship!");
        }
        ships.push_back(std::make_shared<Ship>(length));
    }
}

const std::vector<std::shared_ptr<Ship>>& ShipManager::getShips() const {
    return ships;
}

void ShipManager::printShips() const {
    for (const auto &ship: ships) {
        ship->printInfoShip();
    }
}

```

```
}  
}
```

Название файла: ShipSegment.cpp

```
#include "../include/ShipSegment.h"
```

```
ShipSegment::ShipSegment() :  
    HP(2), state(SegmentState::Undamaged), coordinates(Coordinates{0, 0})  
{
```

```
int ShipSegment::getHP() {  
    return HP;  
}
```

```
void ShipSegment::damage() {  
    switch (state) {  
        case SegmentState::Undamaged:  
            setState(SegmentState::Damaged);  
            break;  
        case SegmentState::Damaged:  
            setState(SegmentState::Destroyed);  
            break;  
        case SegmentState::Destroyed:  
            throw std::runtime_error("Segment is already destroyed!");  
    }  
    HP--;  
}
```

```
void ShipSegment::printInfo() {  
    std::cout << "SEGMENT INFO" << std::endl;  
    std::cout << "HP: " << HP << std::endl;  
    std::cout << "Coordinates" << coordinates.x << " " << coordinates.y << std::endl;  
}
```

```
void ShipSegment::setCoordinates(Coordinates coords) {  
    coordinates = coords;  
}
```

```
void ShipSegment::setState(SegmentState segmentState) {  
    state = segmentState;  
}
```

```
ShipSegment::~ShipSegment() = default;
```

Название файла: Cell.h

```
#ifndef OOP_CELL_H  
#define OOP_CELL_H
```

```
#include "structures.h"
```



```
#include "ShipSegment.h"
#include <memory>
```

```
class Cell{
public:
    Cell();
    ~Cell();
    bool isVisible();
    void setVisibility();
    void setValue(CellValue cellValue);
    void setSegment(std::shared_ptr<ShipSegment> seg);
    CellValue getValue();
    std::shared_ptr<ShipSegment> getSegment();
private:
    Coordinates coordinates;
    CellValue value;
    bool visible;
    std::shared_ptr<ShipSegment> segment;
};
```

```
#endif //OOP_CELL_H
    Название файла: Field.h
```

```
#ifndef OOP_FIELD_H
#define OOP_FIELD_H
```

```
#include "structures.h"
#include "Ship.h"
#include <memory>
#include <vector>
#include "Cell.h"
```

```
class Field {
public:
    Field();
    Field(int height, int width);
    Field(const Field& other);
    Field(Field&& other);
    ~Field();

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

    bool isValidPlace(Coordinates coordinates, int length, Orientation orientation);
    void placeShip(Coordinates coordinates, const std::shared_ptr<Ship>& ship, Orientation
orientation = Orientation::Vertical);
    void attack(Coordinates coordinates);

    void setVisibility();
    void printField();
```

```

private:
    int height;
    int width;
    std::vector<std::vector<std::shared_ptr<Cell>>> cells;
};

#endif //OOP_FIELD_H

    Название файла: Ship.h

#ifndef OOP_SHIP_H
#define OOP_SHIP_H

#include "structures.h"
#include "ShipSegment.h"

#include <vector>
#include <memory>
#include <iostream>

class Ship {
public:
    Ship(int length = 1, Orientation orientation = Orientation::Vertical, Coordinates start = {0,
0});
    ~Ship();
    void printInfoShip();
    int getLength();
    Orientation getOrientation();
    const std::vector<std::shared_ptr<ShipSegment>>& getSegments() const;
    void setCoordinates(Coordinates start);
    Coordinates getCoordinates();
private:
    std::vector<std::shared_ptr<ShipSegment>> segments;
    int length;
    Orientation orientation;
    Coordinates head;
};

#endif //OOP_SHIP_H

    Название файла: ShipManager.h

#ifndef OOP_SHIPMANAGER_H
#define OOP_SHIPMANAGER_H

#include <memory>
#include <vector>
#include "Ship.h"

class ShipManager {

```

```

public:
    ShipManager(const std::vector<int>& sizes);
    const std::vector<std::shared_ptr<Ship>>& getShips() const;
    void printShips() const;
private:
    std::vector<std::shared_ptr<Ship>> ships;
};

#endif //OOP_SHIPMANAGER_H
    Название файла: ShipManager.h

#ifndef OOP_SHIPSEGMENT_H
#define OOP_SHIPSEGMENT_H

#include "structures.h"
#include <iostream>

class ShipSegment {
public:
    ShipSegment();
    ~ShipSegment();
    int getHP();
    void damage();
    void printInfo();
    void setCoordinates(Coordinates coordinates);
    void setState(SegmentState segmentState);
private:
    int HP;
    SegmentState state;
    Coordinates coordinates;
};

#endif //OOP_SHIPSEGMENT_H
    Название файла: structures.h

#ifndef OOP_STRUCTURES_H
#define OOP_STRUCTURES_H

#include <iostream>

enum class SegmentState{Undamaged, Damaged, Destroyed};

enum class Orientation{Vertical, Horizontal};

enum class CellValue : char {
    Empty = '.',
    Segment = 'S',
    Hit = 'W',
    Destroyed = 'X'
};

```

```
struct Coordinates{
    int x;
    int y;

    bool operator==(const Coordinates &other) const {
        return x == other.x && y == other.y;
    }
};

#endif //OOP_STRUCTURES_H
```