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. 3341 _____ Бойцов В.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы.

Целью работы является изучение основ объектно-ориентированного программирования, создание базовых классов для написания игры «Морской бой».

Для выполнения поставленной цели требуется:

- Изучить основные понятия ООП, особенности создания классов;
- Разработать архитектуру базовых классов корабля, поля и менеджера кораблей;
- Реализовать эти классы и связь между ними.

Задание.

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

- 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.

В ходе выполнения работы были разработаны перечисленные ниже классы.

Class BattleshipSegment – вспомогательный, вложенный в *class Battleship* класс. Содержит в себе состояние сегмента корабля *SegmentCondition mSegmentCondition*. Необходим для хранения информации о сегменте корабля, а также для взаимодействия с этим сегментом. Для этого в классе реализованы следующие методы:

- *void takeDamage(const int damage) noexcept* – метод, который принимает на вход количество урона, нанесённого сегменту, в соответствии с которым изменяет состояние поля *mSegmentCondition* (наносит урон сегменту).
- *void repair(const int val) noexcept* – метод, принимающий на вход значение, в зависимости от которого изменяет состояние поля *mSegmentCondition* (чинит сегмент).
- *SegmentCondition getStatus() const noexcept* – метод, возвращающий текущее состояние сегмента.

Class Battleship – класс, отражающий в себе поведение игрового корабля. Содержит в себе вектор сегментов *std::vector<BattleshipSegment> mSegments* для хранения состояния корабля. Этот класс необходим для взаимодействия с сегментами на поле, нанесения урона; количество кораблей является основным свойством состояния игры. Для взаимодействия с классом реализованы следующие методы:

- *bool isAlive() const noexcept* – метод, который считает, есть ли внутри корабля хотя бы один неуничтоженный сегмент, возвращая True в этом случае;
- *SegmentCondition getSegmentCondition(const int index) const* – метод, возвращающий состояние сегмента корабля, находящегося по индексу index. Если индекс некорректен, выбрасывается исключение;
- *std::vector<SegmentCondition> getShipCondition() const* – метод, возвращающий текущее состояние всего корабля в виде векторов объектов *SegmentCondition*;

- *int getLength() const noexcept* – метод, возвращающий длину корабля;
- *void damageSegment(const int index, const int damage)* – метод, который наносит урон *damage* сегменту корабля с индексом *index*. Если индекс некорректен, выбрасывается исключение;
- *void repairSegment(const int index, const int val)* - метод, который чинит на значение *val* сегмент корабля с индексом *index*. Если индекс некорректен, выбрасывается исключение.

class BattlefieldCell – вспомогательный класс, вложенный в *class Battlefield*. Класс хранит в себе указатель на корабль *Battleship* mShipPointer*, сегмент которого можно расположить в клетке (если сегмента корабля в клетке нет, то указатель *nullptr*), индекс сегмента внутри корабля *int mShipSegmentIndex*, а также статус клетки *CellStatus mStatus*. Данный класс необходим для взаимодействия с клетками игрового поля, хранением сегментов кораблей и взаимодействием с ними через игровое поле. Для этого в классе реализованы следующие методы:

- *CellStatus getStatus() const noexcept* – возвращает внутриигровой статус клетки;
- *SegmentCondition getSegmentCondition() const* – возвращает состояние сегмента корабля, если он есть в клетке. Если его нет, выбрасывается исключение;
- *bool hasShip() const* – возвращает *True*, если в клетке есть сегмент корабля, иначе *False*;
- *void setShipSegment(Battleship* const shipPointer, const int shipSegmentIndex) noexcept* – метод, который ставит в клетку сегмент корабля по указателю *shipPointer* с индексом *shipSegmentIndex*;
- *void attackCell(const int damage)* – метод, наносящий урон *damage* сегменту корабля, если таковой есть в клетке. В противном случае выбрасывается исключение.

Class Battlefield – класс игрового поля. Хранит в себе свои горизонтальные и вертикальные размеры, а также «двумерный» вектор клеток поля *std::vector<std::vector<BattlefieldCell>> mBattlefieldArray*. Такое хранение кораблей на поле вызвано простотой и интуитивностью реализации, пускай и несколько проигрывает по памяти по сравнению с другими вариантами, например, *unordered_map*. Данный класс нужен для хранения координат кораблей, взаимодействия с ними через координаты, а также отражения состояния клеток игрового поля, реализации взаимодействия с игровым полем игроком.

Данный класс имеет следующие конструкторы:

- *Battlefield(const int horizontalSize, const int verticalSize)* – конструктор, который принимает размеры игрового поля и заполняет его пустыми клетками;
- *Battlefield(const Battlefield& copy)* – конструктор копирования, который копирует размеры игрового поля, но не корабли, располагающиеся на нем. Данное решение вызвано тем, что при копировании кораблей было бы необходимо создавать новые объекты типа *Battleship*, а также отдельный менеджер кораблей, что нельзя сделать из игрового поля;
- *Battlefield(Battlefield&& moved)* – оператор перемещения, который перемещает всё игровое поле, включая корабли, из moved в новосозданное.

Также реализованы следующие методы:

- *void setShip(Battleship* ship, int x, int y, Orientation orientation)* – метод, который принимает указатель на корабль, его координаты на игровом поле, а также его ориентацию на плоскости. Метод проверяет верность введенных координат (как на их соответствие размерам поле, так и на коллизию с другими кораблями), выбрасывая исключения в критических ситуациях, а затем с помощью метода клеток расставляет сегменты корабля по игровому полю;
- *bool hasShipAtCell(int x, int y) const* – метод, проверяющий наличие корабля на игровом поле;

- *CellStatus getCellStatus(int x, int y) const* – метод, возвращающий статус ячейки игрового поля по координатам;
- *SegmentCondition getCellShipCondition(int x, int y) const* – метод, возвращающий состояние сегмента корабля, если таковой есть на игровом поле; в противном случае выбрасывается исключение;
- *void attackCell(int x, int y)* – метод, атакующий ячейку с введенными координатами. Если там есть корабль, то урон будет нанесен кораблю. Статус ячейки изменится в зависимости от наличия там корабля.

Также в классе реализованы копирующий и перемещающий операторы присваивания (при копировании корабли также не копируются).

class ShipManager – класс, ответственный за создание и хранение кораблей. Содержит в себе вектор пар <указатель на корабль, bool> *std::vector<std::pair<Battleship, bool>> mShipsArray*. Класс создаёт корабли в своём конструкторе и хранит их в себе, в связи с чем класс также имеет метод, который вызывает метод игрового поля по размещению кораблей. В классе реализованы следующие методы:

- *int getAliveShipsNumber()* и *int getInactiveShipsNumber()* – методы, возвращающие количество активных и неактивынх кораблей соответственно;
- *std::vector<Battleship> getInactiveShips() const noexcept* и *std::vector<Battleship> ShipManager::getShips() const noexcept* – метод, возвращающий вектор неактивных и вообще всех кораблей в менеджере. Методы нужны для того, чтобы дать интерфейсам понять, какие корабли осталось поставить;
- *void setShipToBattlefield(Battlefield& field, int shipIndex, int x, int y, Orientation orientation)* – метод, принимающий ссылку на поле, индекс корабля в списке неактивных, а также его координаты и ориентацию, в соответствии с которыми необходимо установить корабль на игровом поле. При удачной постановке корабль переходит из списка неактивных кораблей в активные.

Диаграмма классов, разработанных в ходе выполнения лабораторной работы, представлена на рис.1

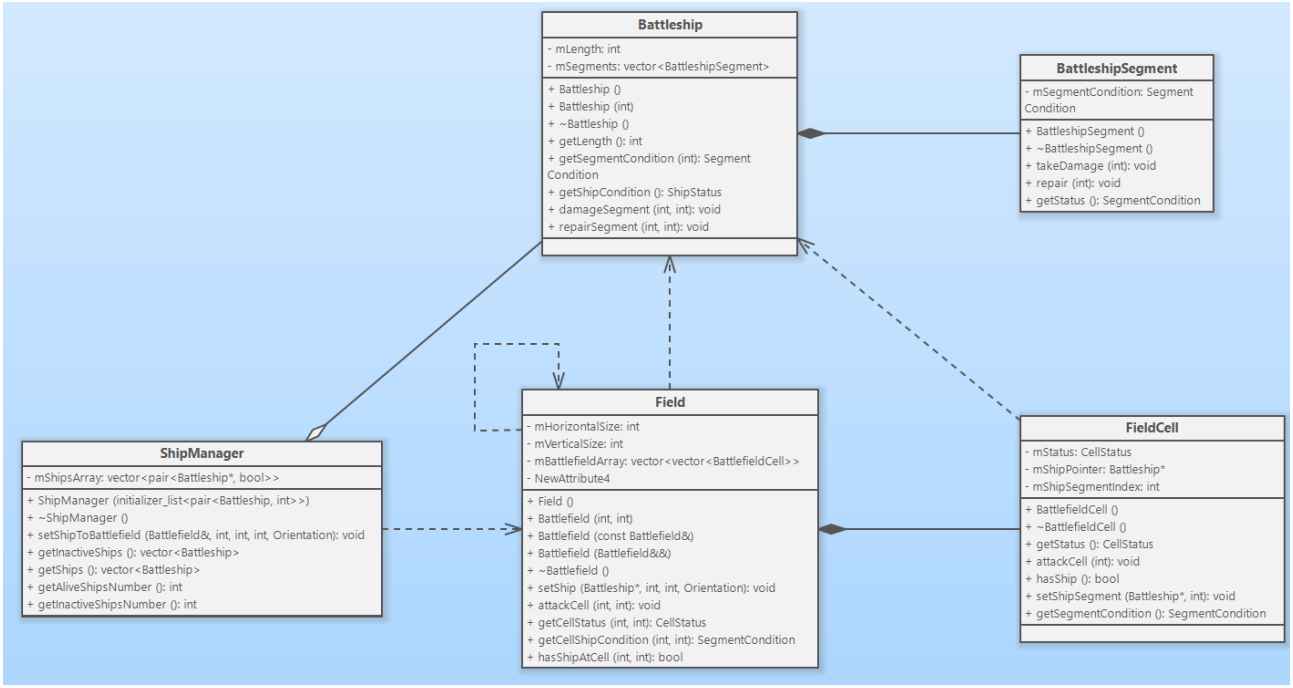

Рисунок 1 – диаграмма классов

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

В результате выполнения лабораторной работы были изучены основы Объектно-ориентированного программирования на языке C++, базовые принципы построения архитектуры программ и создания классов. Были реализованы классы корабля, игрового поля и менеджера кораблей, прописаны методы взаимодействия с этими классами и между этими классами. Была написана программа, проверяющая работоспособность разработанных классов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Battleship.h

```
#ifndef BATTLESHIP
#define BATTLESHIP

#include<vector>

constexpr int minimalShipLength = 1;
constexpr int maximalShipLength = 4;

/**
 * enum, describing current condition of
 * particular ship segment
**/
enum class SegmentCondition
{
    destroyed,
    damaged,
    intact
};

/**
 * Class Battleship contains vector of
 * BattleshipSegment and is used for
 * memorizing taken damage.
 * In future, it will contain
 * own textures for GUI
*/
class Battleship
{
    class BattleshipSegment
    {
        SegmentCondition mSegmentCondition;
    public:
        BattleshipSegment();

        //method to take positive damage for segment
        void takeDamage(const int damage) noexcept;

        //"increase" ship condition by val
        void repair(const int val) noexcept;

        //returns condition of the segment
        SegmentCondition getStatus() const noexcept;
    };

    int mLength;
    std::vector<BattleshipSegment> mSegments;

public:
    Battleship() = default;
    explicit Battleship(int length);
```

```
    //checks if there are not destroyed segments
    bool isAlive() const noexcept;

    //returns condition of <index> segment in ship
    SegmentCondition getSegmentCondition(const int index) const;

    //returns all segments condition in vector
    std::vector<SegmentCondition> getShipCondition() const;

    //returns ships length
    int getLength() const noexcept;

    //damages <index> segment by <damage>
    void damageSegment(const int index, const int damage);

    //repairs ship <index> segment by <val>
    void repairSegment(const int index, const int val);

    ~Battleship() = default;
};

#endif
```

Название файла: Battlefield.h

```
#ifndef BATTLEFIELD
#define BATTLEFIELD


#include"Battleship.h"


constexpr int minimalFieldSize = minimalShipLength;
constexpr int maximalFieldSize = 25;


//enum for plsyers vision of battlefield
enum class CellStatus
{
    unknown,
    empty,
    shipped
};


//enum for ships orientation on the map
enum class Orientation
{
    horizontal,
    vertical
};
```

```
/**
 * Class contains Cells, which have pointers to placed ships
 * if it were placed (for better accessebility),
 * also Cells in-game status.
 * Allows to place ship at map by segments,
 * attack ship by chosen cell.
 *
 * In future, there will be methods for
 * gettng info about the map for drawing one,
 * as some getters for this purpose
*/
class Battlefield
{
    class BattlefieldCell
    {
        //pointer to placed ship if has one
        Battleship* mShipPointer = nullptr;

        //index of segment in the ship (used for getting access to
the ship)
        int mShipSegmentIndex = -1;

        //cells in-game status
        CellStatus mStatus = CellStatus::unknown;

    public:

        BattlefieldCell() = default;

        CellStatus getStatus() const noexcept;

        SegmentCondition getSegmentCondition() const;

        //returns True if there is a ship in cell
        bool hasShip() const;

        //sets ship segment to the cell, adding a pointer to ship
```

```
            void setShipSegment(Battleship* const shipPointer, const
int shipSegmentIndex) noexcept;

            //method to damage a ship by the segment if there is a ship
            void attackCell(const int damage);
        };

        //sizes of the field
        int mHorizontalSize;
        int mVerticalSize;

        //field itself
        std::vector<std::vector<BattlefieldCell>> mBattlefieldArray;

    public:
        Battlefield(const int horizontalSize, const int verticalSize);

        //WARNING: copy constructor does not copy ships, only field
sizes
        Battlefield(const Battlefield& copy);

        //move constructor actually moves stuff
        Battlefield(Battlefield&& moved);

        //set a ship to a shosen cell
        void setShip(Battleship* ship, int x, int y, Orientation
orientation);

        //checks if there if ship in cell at (x, y)
        bool hasShipAtCell(int x, int y) const;

        //gets a status of cell at (x, y)
        CellStatus getCellStatus(int x, int y) const;

        //gets a Condition of segment of ship if has one
        //may throw an exception if there is no ship
        SegmentCondition getCellShipCondition(int x, int y) const;

        //DEBUG method for displaying map to std::cout
```

```
        void display();

        //method for attacking a chosen cell
        void attackCell(int x, int y);

        //WARNING: copy operator does not copy ships, only field sizes
        Battlefield& operator=(const Battlefield& copy);

        //move operator actually moves stuff
        Battlefield& operator=(Battlefield&& moved);
    };

    #endif
```

Название файла: ShipManager.h

```
    #ifndef SHIP_MANAGER
    #define SHIP_MANAGER

    #include<utility>          //for std::pair
    #include<initializer_list>
    #include"Battleship.h"
    #include"Battlefield.h"

    /**
     * This class contains all placed-on-field ships
     * and yet not placed ones, also taking responsibility for
     * calling fields placeShip methods.
     * It will be used to count how many ships there are left,
     * and then to end up the game
    */
    class ShipManager
    {
        std::vector<std::pair<Battleship*, bool>> mShipsArray;

    public:

        //gets list of pairs <shipLength, amount>, places all to
inactive ships
        ShipManager(std::initializer_list<std::pair<int,          int>>
shipList);

        int getAliveShipsNumber() const noexcept;

        int getInactiveShipsNumber() const noexcept;

        //returns vector of all inactive ships
        std::vector<Battleship> getInactiveShips() const noexcept;

        std::vector<Battleship> getShips() const noexcept;
```

```
        //method will be used by player/bot interface to call fields
method
        void setShipToBattlefield(Battlefield& field, int shipIndex,
int x, int y, Orientation orientation);

        ~ShipManager();
    };

    #endif
```

Название файла: Battleship.cpp

```
    #include "Battleship.h"
    #include<stdexcept>

    Battleship::BattleshipSegment::BattleshipSegment()
    {
        mSegmentCondition=SegmentCondition::intact;
    }

    void Battleship::BattleshipSegment::takeDamage(const int damage)
noexcept
    {
        if(damage<=0)
            return;
        if(damage>=2)
            mSegmentCondition=SegmentCondition::destroyed;
        else
            if(mSegmentCondition == SegmentCondition::intact)
                mSegmentCondition=SegmentCondition::damaged;
            else if(mSegmentCondition == SegmentCondition::damaged)
                mSegmentCondition=SegmentCondition::destroyed;
    }

    void Battleship::BattleshipSegment::repair(const int val) noexcept
    {
        if(val<=0)
            return;
        if(val==1)
        {
            if(mSegmentCondition==SegmentCondition::destroyed)
                mSegmentCondition=SegmentCondition::damaged;
            else
                mSegmentCondition=SegmentCondition::intact;
        }
        else
            mSegmentCondition=SegmentCondition::intact;
    }

    SegmentCondition Battleship::BattleshipSegment::getStatus() const
noexcept
    {
        return mSegmentCondition;
    }


    Battleship::Battleship(int length):mLength(length)
    {
        if(length<minimalShipLength or length>maximalShipLength)
```

```
            throw std::logic_error("Invalid ship size");
        for(int i=0; i<mLength; i++)
            mSegments.emplace_back();
    }

    bool Battleship::isAlive() const noexcept
    {
        //returns True if at least one segment is not destroyed
        for(auto segment: mSegments)
            if(segment.getStatus()!=SegmentCondition::destroyed)
                return true;
        return false;
    }

    SegmentCondition  Battleship::getSegmentCondition(const  int  index)
const
    {
        if(index<0 or index>=mLength)
            throw std::length_error("Invalid index");
        return mSegments[index].getStatus();
    }

    int Battleship::getLength() const noexcept
    {
        return mLength;
    }

    void Battleship::damageSegment(const int index, const int damage)
    {
        if(index<0 or index>=mLength)
        {
            throw std::length_error("Invalid segment index");
        }
        mSegments[index].takeDamage(damage);
    }

    void Battleship::repairSegment(const int index, const int val)
    {
        if(index<0 or index>=mLength)
            throw std::length_error("Invalid segment index");
        mSegments[index].repair(val);
    }

    std::vector<SegmentCondition> Battleship::getShipCondition() const
    {
        std::vector<SegmentCondition> segments;
        for(auto segment: mSegments)
            segments.push_back(segment.getStatus());
        return segments;
    }
```

Название файла: BattlefieldCell.cpp

```
    #include"Battlefield.h"

    #include<stdexcept>
```

```
    void Battlefield::BattlefieldCell::setShipSegment(Battleship* const
shipPointer, const int shipSegmentIndex) noexcept
    {
        mShipPointer = shipPointer;
        mShipSegmentIndex = shipSegmentIndex;
    }

    void Battlefield::BattlefieldCell::attackCell(const int damage)
    {
        if (mShipPointer == nullptr)
            mStatus = CellStatus::empty;
        else
        {
            mShipPointer->damageSegment(mShipSegmentIndex, damage);
            mStatus = CellStatus::shipped;
        }
    }

    CellStatus Battlefield::BattlefieldCell::getStatus() const noexcept
    {
        return mStatus;
    }

    SegmentCondition  Battlefield::BattlefieldCell::getSegmentCondition()
const
    {
        if (mShipPointer == nullptr)
            throw std::logic_error("No ship handles this cell");
        return mShipPointer->getSegmentCondition(mShipSegmentIndex);
    }

    bool Battlefield::BattlefieldCell::hasShip() const
    {
        return mShipPointer != nullptr;
    }
```

Название файла: Battlefield.cpp

```
#include"Battlefield.h"
#include<iostream>
```

```
    //sets basic field size
    Battlefield::Battlefield(const int horizontalSize, const int
verticalSize):
    mHorizontalSize(horizontalSize), mVerticalSize(verticalSize)
    {
        if(horizontalSize<minimalFieldSize                    or
verticalSize<minimalFieldSize or
        horizontalSize>maximalFieldSize                       or
verticalSize>maximalFieldSize)
            throw std::logic_error("Invalid field size");

        //fulls field with empty cells
        mBattlefieldArray.resize(mVerticalSize);
        for(int y=0;y<mVerticalSize; y++)
        {
            mBattlefieldArray[y].resize(mHorizontalSize);
            for(int x=0; x<mHorizontalSize; x++)
            {
                mBattlefieldArray[y][x] = BattlefieldCell();
            }
        }
    }

    //copies only field sizes, not ships
    Battlefield::Battlefield(const Battlefield& copy):
    Battlefield(copy.mHorizontalSize, copy.mVerticalSize){}

    //moves all stuff
    Battlefield::Battlefield(Battlefield&& moved)
    {
        mHorizontalSize=std::move(moved.mHorizontalSize);
        mVerticalSize=std::move(moved.mVerticalSize);
        mBattlefieldArray=std::move(moved.mBattlefieldArray);
    }

    void Battlefield::setShip(Battleship* ship, int x, int y,
Orientation orientation)
    {
```

```
        if(ship == nullptr)
            throw std::invalid_argument("Ship pointer is nullptr");

        /**
         * Uses offset to calculate the exact ships area as
         * (x+xOffset, y+yOffset)
        */
        int xOffset, yOffset;
        if(orientation == Orientation::horizontal)
        {
            xOffset = ship->getLength()-1;
            yOffset = 0;
        }
        else
        {
            xOffset=0;
            yOffset=ship->getLength()-1;
        }

        //check if ship fits in the field
        if(x<0 or x>=mHorizontalSize-xOffset or y<0 or y>mVerticalSize-
yOffset)
            throw std::logic_error("Invalid ship coordinates");

        //collision check
        for(int j=y-1; j<=y+yOffset+1; j++)
            if(j>=0 and j<mVerticalSize)
                for(int i=x-1; i<=x+xOffset+1; i++)
                    if(i>=0 and i<mHorizontalSize)
                    {
                        if(mBattlefieldArray[j][i].hasShip())
                            throw        std::logic_error("Intersection
between ships occured");
                    }
        //sets ship to the cells by segments
        int segmentIndex=0;
        for(int j=y; j<=y+yOffset; j++)
            for(int i=x; i<=x+xOffset; i++)
            {
```

```
                mBattlefieldArray[j][i].setShipSegment(ship,
segmentIndex++);
            }
        }

        bool Battlefield::hasShipAtCell(int x, int y) const
        {
            if(x<0 or x>=mHorizontalSize or y<0 or y>=mVerticalSize)
                throw std::invalid_argument("Invalid cell indexes");

            return mBattlefieldArray[y][x].hasShip();
        }

        CellStatus Battlefield::getCellStatus(int x, int y) const
        {
            if(x<0 or x>=mHorizontalSize or y<0 or y>=mVerticalSize)
                throw std::invalid_argument("Invalid cell indexes");

            return mBattlefieldArray[y][x].getStatus();
        }

        SegmentCondition  Battlefield::getCellShipCondition(int  x,  int  y)
const
        {
            if(x<0 or x>=mHorizontalSize or y<0 or y>=mVerticalSize)
                throw std::invalid_argument("Invalid cell indexes");

            //may throw an exception if there is no ship
            return mBattlefieldArray[y][x].getSegmentCondition();
        }

        void Battlefield::attackCell(int x, int y)
        {
            mBattlefieldArray[y][x].attackCell(1);
        }

        //DEBUG METHOD
        void Battlefield::display()
        {
```

```
        /**
         *  - stays for unknown,
         *  * stays for empty,
         *  0 stays for shipped,
         *  x stays for damaged,
         *  X stays for destroyed
        */
        std::cout<<"\n";
        for(int y=0; y<mVerticalSize; y++)
        {
            for(int x=0; x<mHorizontalSize; x++)
            {
                if        (mBattlefieldArray[y][x].getStatus()        ==
CellStatus::unknown)
                    std::cout<<"-";
                else      if(mBattlefieldArray[y][x].getStatus()      ==
CellStatus::empty)
                    std::cout<<"*";
                else      if(mBattlefieldArray[y][x].getStatus()      ==
CellStatus::shipped)
                {
                    if(mBattlefieldArray[y][x].getSegmentCondition()  ==
SegmentCondition::intact)
                        std::cout<<"0";
                    else
if(mBattlefieldArray[y][x].getSegmentCondition()                      ==
SegmentCondition::damaged)
                        std::cout<<"x";
                    else
                        std::cout<<"X";
                }
            }
            std::cout<<"\n";
        }
        std::cout<<"\n";
    }

    //WARNING: does not copy ships, only field sizes
    Battlefield& Battlefield::operator=(const Battlefield& copy)
```

```
    {
        if(&copy!=this)
        {
            mHorizontalSize=copy.mHorizontalSize;
            mVerticalSize=copy.mVerticalSize;
            mBattlefieldArray.clear();
            mBattlefieldArray.resize(copy.mVerticalSize);
            for(int y=0;y<copy.mVerticalSize; y++)
            {
                mBattlefieldArray[y].resize(copy.mHorizontalSize);
                for(int x=0; x<copy.mHorizontalSize; x++)
                {
                    mBattlefieldArray[y][x]= BattlefieldCell();
                }
            }
        }
        return *this;
    }

    Battlefield& Battlefield::operator=(Battlefield&& moved)
    {
        if(&moved!=this)
        {
            mHorizontalSize=std::move(moved.mHorizontalSize);
            mVerticalSize=std::move(moved.mVerticalSize);
            mBattlefieldArray=std::move(moved.mBattlefieldArray);
        }
        return *this;
    }
```

Название файла: ShipManager.cpp

```
    #include"ShipManager.h"
    #include<stdexcept>

    //gets list of pairs <size, amount>
    ShipManager::ShipManager(std::initializer_list<std::pair<int,  int>>
shipList)
    {
        for(auto shipSeries: shipList)
        {
            if(shipSeries.first<minimalShipLength                    or
shipSeries.first>maximalShipLength)
                throw std::invalid_argument("Invalid ship length");
```

```
            if(shipSeries.second<=0)
                throw   std::invalid_argument("Ships   number   must   be
greater than zero");

            for(int i=0; i<shipSeries.second; i++)
            {
                Battleship* newShip = new Battleship(shipSeries.first);
                mShipsArray.push_back({newShip, false});
            }
        }
    }

    std::vector<Battleship>    ShipManager::getInactiveShips()    const
noexcept
    {
        std::vector<Battleship> ships;
        for(auto ship: mShipsArray)
            if(ship.second == false)
                ships.push_back(*ship.first);
        return ships;
    }

    std::vector<Battleship> ShipManager::getShips() const noexcept
    {
        std::vector<Battleship> ships;
        for(auto ship: mShipsArray)
            ships.push_back(*ship.first);
        return ships;
    }

    int ShipManager::getAliveShipsNumber() const noexcept
    {
        int shipNum=0;

        for(auto ship: mShipsArray)
            if(ship.first->isAlive())
                shipNum++;
        return shipNum;
    }

    void   ShipManager::setShipToBattlefield(Battlefield&   field,   int
shipIndex, int x, int y, Orientation orientation)
    {
        if(getInactiveShipsNumber() == 0)
            throw std::logic_error("No ships to be set available");
        if (shipIndex<0 or shipIndex>=mShipsArray.size())
            throw std::invalid_argument("Invalid ship index");
        if(mShipsArray[shipIndex].second)
            throw std::logic_error("Ship was already placed to field");

        //may throw an exception from field, so needs to be processed
once being called
        field.setShip(mShipsArray[shipIndex].first, x, y, orientation);
        mShipsArray[shipIndex].second=true;
    }

    int ShipManager::getInactiveShipsNumber() const noexcept
    {
```

```
    int shipNum=0;

    for(auto ship: mShipsArray)
        if(ship.second==false)
            shipNum++;
    return shipNum;
}

ShipManager::~ShipManager()
{
    for(auto ship: mShipsArray)
    {
        delete ship.first;
    }
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Тестирование написанных классов было выполнено в виде небольшой программы, которая создаёт объекты класса Battleship, Battlefield и ShipManager, добавляет в менеджер кораблей корабли, размещает их на поле, а затем – с помощью отладочных функций – выводит их на экран.

Название файла: main.cpp

```
#include"ShipManager.h"
#include<iostream>

//no OOP only as debug func
void printShip(const Battleship& ship)
{
    auto segments = ship.getShipCondition();
    std::cout<<"Ship "<<ship.getLength()<<" condition:\n\t";
    for(auto segment: segments)
        std::cout<<int(segment)<<' ';
    std::cout<<'\n';
}

//no OOP only as debug func
void printShipsInManager(const ShipManager& manager)
{
    auto ships = manager.getInactiveShips();
    std::cout<<"Inactive ships in manager:
"<<manager.getInactiveShipsNumber()<<'\n';
    for(auto ship: ships)
        printShip(ship);
}

int main()
{
    Battleship myShip(3);
    printShip(myShip);
    myShip.damageSegment(1, 1);
    printShip(myShip);
```

```
        ShipManager myManager({{1, 2}, {2, 3}, {1, 2}});
        printShipsInManager(myManager);

        Battlefield myField(20, 15);
        myManager.setShipToBattlefield(myField,      0,      1,      2,
Orientation::horizontal);
        printShipsInManager(myManager);
        Battlefield newField(24, 24);
        newField.display();
        newField=myField;
        myManager.setShipToBattlefield(newField,     1,     10,     1,
Orientation::vertical);
        printShipsInManager(myManager);
        myField.display();
        newField.display();
        myField.attackCell(1, 1);
        myField.attackCell(2, 1);
        myField.attackCell(2, 2);
        myField.attackCell(1, 2);

        myField.attackCell(10, 1);
        myField.attackCell(10, 2);
        myField.attackCell(10, 3);
        myField.attackCell(10, 4);
        myField.attackCell(10, 5);

        newField.attackCell(1, 1);
        newField.attackCell(2, 1);
        newField.attackCell(2, 2);
        newField.attackCell(1, 2);

        newField.attackCell(10, 1);
        newField.attackCell(10, 2);
        newField.attackCell(10, 3);
        newField.attackCell(10, 4);
        newField.attackCell(10, 5);

        std::cout<<"Before                                    attack:
"<<myManager.getAliveShipsNumber()<<'\n';
```

```
        newField.attackCell(10, 1);
        newField.attackCell(10, 2);
        newField.attackCell(10, 3);
        newField.attackCell(10, 4);
        newField.attackCell(10, 5);

        std::cout<<"After                                    attack:
"<<myManager.getAliveShipsNumber()<<'\n';

        std::cout<<int(myField.getCellStatus(10, 1))<<'\n';
        if(newField.hasShipAtCell(10, 1))
            std::cout<<int(newField.getCellShipCondition(10, 1))<<'\n';

        myField.display();
        newField.display();
    }
```

Результаты тестирования:

Ship 3 condition:

2 2 2

Ship 3 condition:

2 1 2

Inactive ships in manager: 7

Ship 1 condition:

2

Ship 1 condition:

2

Ship 2 condition:

2 2

Ship 2 condition:

2 2

Ship 2 condition:

2 2

Ship 1 condition:

2

Ship 1 condition:

2

Inactive ships in manager: 6

Ship 1 condition:

2

Ship 2 condition:

2 2

Ship 2 condition:

2 2

Ship 2 condition:

2 2

Ship 1 condition:

2

Ship 1 condition:

2

Inactive ships in manager: 5

Ship 2 condition:

2 2

Ship 2 condition:

2 2

Ship 2 condition:

2 2

Ship 1 condition:

2

Ship 1 condition:

2

```
--------------------
--------------------
--------------------
--------------------
--------------------
--------------------
--------------------
--------------------
--------------------


--------------------
--------------------
--------------------
--------------------
--------------------
--------------------
--------------------
--------------------
--------------------
--------------------
--------------------
--------------------
--------------------
--------------------
--------------------
```

Before attack: 7

After attack: 6

1

0

--------*---------

-x*-------*---------

----------*---------

----------*---------

----------*---------

--------X---------

-**-------*---------

----------*---------

----------*---------

----------*---------
