МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе №1
по дисциплине «Объектно-ориентированное программирование»
Тема: Создание классов.

Студент гр. 3344 _____ Бубякина Ю.В.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2024

Цель работы.

Научиться проектировать и реализовывать классы с использованием объектно-ориентированного программирования. Освоить принципы инкапсуляции и наследования. Работа включает в себя создание трех ключевых классов: корабля, менеджера кораблей и игрового 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, повреждён, 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.

с) 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:

i. неизвестно (изначально вражеское поле полностью неизвестно),
ii. пустая (если на клетке ничего нет)
iii. 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 *BattleShipSegment*

Приватные поля:

1. *BattleShipSegmentState state* — указывает на текущее состояние ячейки корабля (*Broken* | *Damaged* | *Undamaged*).

Методы класса:

1. *BattleShipSegment()* — конструктор класса, инициализирует *state = Undamaged.*
2. *BattleShipSegmentState getState()* — возвращает текущее состояние ячейки корабля.
3. *void setState(BattleShipSegmentState state)* — присваивает полю *state* значение аргумента *state*.
4. *void getHit()* — наносит урод по сегменту корабля, изменяя его *state*.

Класс *BattleFieldCell*

Приватные поля:

1. *BattleFieldCellState state* — указывает на текущее состояние ячейки поля (*Unknown* | *Empty* | *Ship*).
2. *BattleShipSegment* battleShipSegment* — указатель на расположенный в данной ячейке поля *BattleShipSegment*.

Методы класса:

1. *BattleFieldCell()* — конструктор класса, инициализирует *state = Unknown.*
2. *BattleFieldCellState getState()* — возвращает текущее состояние ячейки поля.
3. *void setBattleShipSegment(BattleShipSegment &)* — устанавливает сегмент корабля в ячейку поля.
4. *bool hasShipSegment()* — возвращает есть ли ячейка корабля в данной ячейке поля.

5. *void hitCell()* — открывает ячейку если она была *Unknown* и наносит урон по сегменту корабля, если он там имеется.

Оператор вывода в консоль*:*

friend std::ostream &operator<<(std::ostream &os, BattleFieldCell &cell) — перегрузка оператора << для класса *std::ostream,* выводит состояние ячейки: «~» — не открыта, «w» — открыта, корабля нет, «O» — открыта, есть корабль и он цел, «o» — открыта, есть корабль и он поврежден, «.» — открыта, есть корабль и он сломан.

Класс *BattleShip*

Приватные поля:

1. *std::vector<BattleShipSegment> segments* — массив сегментов корабля.

Методы класса:

1. *BattleShip(unsigned size)* — конструктор класса, инициализирует *segments* массивом размера *size.*
2. *unsigned size()* — возвращает количество сегментов корабля.
3. *BattleShipSegment &operator[](unsigned index)* — оператор [] используемый для доступа к сегментам корабля по индексу.

Класс *BattleShipManager*

Приватные поля:

1. *std::vector<BattleShip> battleShips* — массив для хранения объектов кораблей.

Методы класса:

1. BattleShipManager(std::vector<unsigned> sizes) — конструктор класса, создает и помещает корабли заданной длины в *battleShips.*
2. *unsigned getNumberOfBattleShips()* — возвращает количество кораблей.
3. BattleShip &operator[](unsigned index)

4. *BattleShip &operator[](unsigned index)* — возвращает ссылку на корабль расположенный по индексу.

Класс *BattleField*

Приватные поля:

1. *std::vector<std::vector<BattleFieldCell>> field* — игровое поле.
2. *unsigned width* - ширина поля.
3. *unsigned height* - высота поля.

Методы Класса:

1. *BattleField(unsigned width, unsigned height)* — конструктор класса, инициализирует *width* и *height* равными соответствующим аргументам, инициализирует пустое поле *field* размера *width* X *height*.
2. *BattleField::BattleField(const BattleField &other)*: контруктор копирования.
3. *BattleField::BattleField(BattleField &&other) noexcept*: конструктор перемещения.
4. *BattleField &BattleField::operator=(const BattleField &other)*: оператор присваивания копированием.
5. *BattleField &BattleField::operator=(BattleField &&other) noexcept*: оператор присваивания перемещением.
6. *unsigned getWidth()* — возвращает ширину поля.
7. *unsigned getHeight()* — возвращает высоту поля.
8. *void hitCell(Position position)* — атакует ячейку по переданной позиции.
9. *bool placeBattleShip(Position position, BattleShip &battleShip, Orientation orientation)* — устанавливает корабль на поле по заданной координате в нужной ориентации в соответствии с переданными параметрами. Возвращает *true* или *false* в зависимости от того, была ли произведена установка корабля, или нарушены правила расстановки.

Оператор вывода в консоль:

friend std::ostream &operator<<(std::ostream &os, BattleField &field) — перегрузка оператора << для класса *std::ostream,* выводит ячейки поля в консоль.

Тестирование.

```cpp
#include <iostream>
#include "BattleField.h"
#include "BattleShipManager.h"

int main()
{
    // Создание поля
    BattleField battleField(15, 10);
    std::cout << "До расстановки кораблей" << battleField;

    BattleShipManager battleShipManager({1, 1, 1, 1, 2, 2, 2, 3, 3, 4});

    // Добавление кораблей
    battleField.placeBattleShip({5, 6}, battleShipManager[8], Orientation::Up);
    battleField.placeBattleShip({2, 1}, battleShipManager[5], Orientation::Right);

    std::cout << "Попытка поставить корабль рядом с другим\n";
    battleField.placeBattleShip({1, 1}, battleShipManager[7], Orientation::Down);
    std::cout << "\n";

    std::cout << "Попытка поставить корабль вне поля\n";
    battleField.placeBattleShip({1, 3}, battleShipManager[9], Orientation::Left);
    std::cout << "\n";

    std::cout << "После расстановки кораблей" << battleField;

    // Удар по кораблям
    battleField.hitCell({5, 6});
    battleField.hitCell({4, 3});
    battleField.hitCell({4, 4});

    std::cout << "После удара по кораблям" << battleField;
}
```

Вывод в консоль:

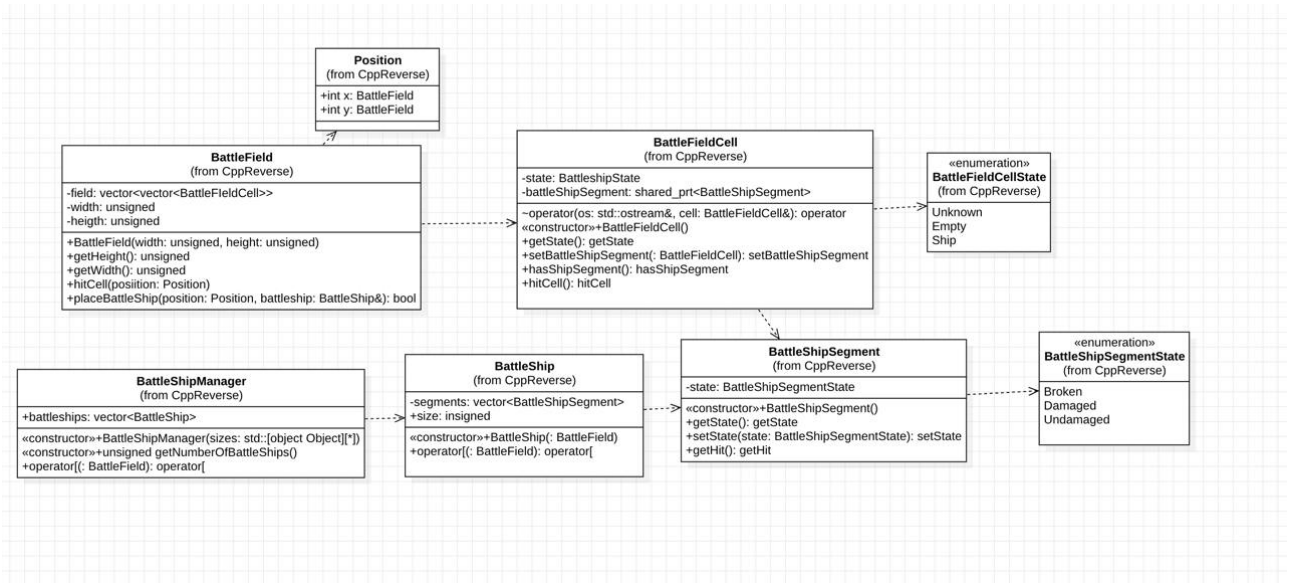
```
До расстановки кораблей
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w

Попытка поставить корабль рядом с другим
There is already a ship located in (2, 1)!

Попытка поставить корабль вне поля
You cant add a ship outside of the field!

После расстановки кораблей
w w w w w w w w w w w w w w
w w 0 0 w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w 0 w w w w w w w w
w w w w w 0 w w w w w w w w
w w w w w 0 w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w

После удара по кораблям
w w w w w w w w w w w w w w
w w 0 0 w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w 0 w w w w w w w w
w w w w w 0 w w w w w w w w
w w w w w o w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w
w w w w w w w w w w w w w w
```

Выводы.

Были изучены механизмы инкапсуляции, а также создание классов и их методов. Реализован класс BattleShip (Корабль), который может быть размещён на игровом поле и имеет определённую длину. Корабль может располагаться на поле в любом направлении, и каждый его сегмент имеет три состояния: целый, повреждённый и уничтоженный.

Создан класс BattleShipManager (Менеджер Кораблей), который хранит информацию о кораблях и предоставляет доступ к ним по индексу.

Также реализован класс BattleField (Игровое Поле), принимающий в качестве параметров размеры поля. Этот класс включает метод для размещения корабля на указанных координатах, при этом корабли не могут пересекаться или соприкасаться. Реализован механизм проверки возможности размещения корабля.


Position
(from CppReverse)
+int x: BattleField
+int y: BattleField
BattleField
(from CppReverse)
-field: vector<vector<BattleFIeldCell>>
-width: unsigned
-heigth: unsigned
+BattleField(width: unsigned, height: unsigned)
+getHeight(): unsigned
+getWidth(): unsigned
+hitCell(posiition: Position)
+placeBattleShip(position: Position, battleship: BattleShip&): bool
BattleFieldCell
(from CppReverse)
-state: BattleshipState
-battleShipSegment: shared_prt<BattleShipSegment>
~operator(os: std::ostream&, cell: BattleFieldCell&): operator
«constructor»+BattleFieldCell()
+getState(): getState
+setBattleShipSegment(: BattleFieldCell): setBattleShipSegment
+hasShipSegment(): hasShipSegment
+hitCell(): hitCell
«enumeration»
BattleFieldCellState
(from CppReverse)
Unknown
Empty
Ship
BattleShipManager
(from CppReverse)
+battleships: vector<BattleShip>
«constructor»+BattleShipManager(sizes: std::[object Object][*])
«constructor»+unsigned getNumberOfBattleShips()
+operator[(: BattleField): operator[
BattleShip
(from CppReverse)
-segments: vector<BattleShipSegment>
+size: insigned
«constructor»+BattleShip(: BattleField)
+operator[(: BattleField): operator[
BattleShipSegment
(from CppReverse)
-state: BattleShipSegmentState
«constructor»+BattleShipSegment()
+getState(): getState
+setState(state: BattleShipSegmentState): setState
+getHit(): getHit
«enumeration»
BattleShipSegmentState
(from CppReverse)
Broken
Damaged
Undamaged