**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В. И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

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

**Тема: Полиморфизм**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Иванов С.С. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Научиться использовать динамический полиморфизм путём создания класса-интерфейса способности, через наследование создать 3 разные способности, интерфейс события должен быть унифицирован, чтобы их можно было единообразно использовать через интерфейс.

## **Задание**

1. Создать класс-интерфейс способности, которую игрок может применять. Через наследование создать 3 разные способности:
2. Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).
3. Сканер - позволяет проверить участок поля 2х2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус.
4. Обстрел - наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.
5. Создать класс менеджер-способностей. Который хранит очередь способностей, изначально игроку доступно по 1 способности в случайном порядке. Реализовать метод применения способности.
6. Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.
7. Реализуйте набор классов-исключений и их обработку для следующих ситуаций (можно добавить собственные):
8. Попытка применить способность, когда их нет
9. Размещение корабля вплотную или на пересечении с другим кораблем
10. Атака за границы поля

## **Выполнение работы**

Класс Visitor:

Абстрактный класс посетителя. Имплементации этого класса будут посещать имплементации класса Owner (владельцев) и дополнять их функционал.

Класс содержит следующие публичные виртуальные методы:

* virtual void operator()(void \*\_obj) = 0; - принимает владельца и определяет дополняющий его функционал;
* virtual ~Visitor() = 0; - виртуальный деструктор;

Класс Owner:

Абстрактный класс владельца. Имплементации этого класса будут принимать к себе имплементации класса Visitor (посетителей), таким образом расщиряя свой функционал.

Класс содержит следующие публичные методы:

* template <class T, class ...Args\_T>

std::shared\_ptr<Visitor>

accept(Args\_T&& ...\_args) – метод, принимающий в шаблон класс посетителя, создающий его внутри себя и выдающий std::shared\_ptr на него. Тип параметра Args\_T&& ради идеальной передачи аргументов;

Класс содержит следующие публичные виртуальные методы:

* virtual ~Owner() = 0; - виртуальный деструктор;

Класс Field:

Поля начинает имплементить Owner и становится владельцем.

Новые приватные поля:

* std::shared\_ptr<Command> \_react\_of\_destroyed\_ship; - команда, выдающая рандомную способность;

Новые публичные методы:

* void accept\_skill(std::shared\_ptr<iSkill> \_skill); - метод, которые принимает способность-посетителя;

Класс iSkill:

Интерфейс способности. Наследуется от Visitor. Имплементации этого класса будут посещать объекты класса Field и выполнять логику способности.

Класс содержит следующие публичные виртуальные методы:

* virtual inline void use(Field &\_fd) = 0; - принимает поле и использует способность;
* virtual SkillName classname() const noexcept = 0; - возвращает член перечисления, идентифицирующий способность;

Класс DoubleHit:

Класс имплементит iSkill. Реализует двойной урон по определённой клетке на указанном поле.

Класс содержит следующие приватные поля:

* Unit \_unit; - точка, по которой будет совершён выстрел;

Класс содержит следующие публичные методы:

* explicit DoubleHit(const Unit &\_unit); - конструктор, принимающий точку, по которой будет совершён выстрел;
* DoubleHit(std::uint64\_t \_x, std::uint64\_t \_y); - конструктор, принимающий координаты точки, по которой будет совершён выстрел;
* DoubleHit(); - конструктор по умолчанию;
* void operator()(void \*\_obj) override; - метод, содержащий основную логику способности;
* SkillName classname() const noexcept override; - возвращает член перечисления, идентифицирующий способность;
* ~DoubleHit() override; - переопределение деструктора;

Класс RocketAttack:

Класс имплементит iSkill. Реализует удар по случайной клетке на указанном поле.

Класс содержит следующие приватные поля:

* std::random\_device rd; - функтор, выдающий случайные числа;

Класс содержит следующие публичные методы;

* RocketAttack(); - конструктор по умолчанию;
* void operator()(void \*\_obj) override; - метод, содержащий основную логику способности;
* SkillName classname() const noexcept override; - возвращает член перечисления, идентифицирующий способность;
* ~RocketAttack() override; - переопределение деструктора;

Класс содержит следующие приватные методы:

* Unit \_\_get\_random\_unit\_of\_ship(const Field &\_sm) noexcept; - возвращает клетку, на которой расположен верхний левый сегмент рандомного корабля;
* std::uint8\_t \_\_get\_random\_index\_of\_segments(const Ship &\_ship) noexcept; - возвращает индекс рандомного сегмента корабля;

Класс Scanner:

Класс имплементит iSkill. Реализует поиск кораблей на указанном поле.

Класс содержит следующие приватные поля:

* Scanner(const Unit &\_unit, const std::function<void(const Unit&)> &\_funct); - конструктор, принимающий точку и реакцию на обнаружение сегмента корабля;
* Scanner(std::uint64\_t \_x, std::uint64\_t \_y, const std::function<void(const Unit&)> &\_funct); - конструктор, принимающий координаты точки и реакцию на обнаружение сегмента корабля;
* Scanner(); - конструктор по умолчанию;
* void operator()(void \*\_obj) override; - метод, содержащий основную логику способности;
* SkillName classname() const noexcept override; - возвращает член перечисления, идентифицирующий способность;
* ~Scanner() override; - переопределение деструктора;

Класс iSkillFactory:

Интерфейс фабрики способности. Имплементации этого класса будут создавать имплементации класса iSkill.

Класс содержит следующие публичные виртуальные методы:

* virtual std::shared\_ptr<iSkill> create(const Unit &\_unit = Unit(), const std::function<void(const Unit&)> &\_funct = \_\_plug\_for\_react()) = 0; - метод-фабрика способностей;
* virtual SkillName skillname() const noexcept = 0; - возвращает член перечисления, идентифицирующий способность;
* virtual ~iSkillFactory() = 0; - виртуальный деструктор;

Класс DoubleHitFactory:

Класс имплементит iSkillFactory. Создаёт DoubleHit.

Класс содержит следующие публичные методы:

* std::shared\_ptr<iSkill> create(const Unit &\_unit, const std::function<void(const Unit&)> &\_funct = \_\_plug\_for\_react()) override; - метод-фабрика способностей;
* SkillName skillname() const noexcept override; - возвращает член перечисления, идентифицирующий способность;
* ~DoubleHitFactory() override; - переопределение деструктора;

Класс RocketAttackFactory:

Класс имплементит iSkillFactory. Создаёт RocketAttack.

Класс содержит следующие публичные методы:

* std::shared\_ptr<iSkill> create(const Unit &\_unit = Unit(), const std::function<void(const Unit&)> &\_funct = \_\_plug\_for\_react()) override; - метод-фабрика способностей;
* SkillName skillname() const noexcept override; - возвращает член перечисления, идентифицирующий способность;
* ~RocketAttackFactory() override; - переопределение деструктора;

Класс ScannerFactory:

Класс имплементит iSkillFactory. Создаёт Scanner.

Класс содержит следующие публичные методы:

* std::shared\_ptr<iSkill> create(const Unit &\_unit, const std::function<void(const Unit&)> &\_funct) override; - метод-фабрика способностей;
* SkillName skillname() const noexcept override; - возвращает член перечисления, идентифицирующий способность;
* ~ScannerFactory() override; - переопределение деструктора;

Класс SetupReactOfDestroyedShip:

Класс установки реакции на уничтожение корабля у поля. Наследуется от Visitor.

Класс содержит следующие приватные поля:

* std::shared\_ptr<Command> \_cmd; - команда, испоняемая при уничтожении корабля;

Класс содержит следующие публичные методы:

* explicit SetupReactOfDestroyedShip(std::shared\_ptr<Command> \_cmd); - конструктор, принимающий команду;
* void operator()(void \*\_obj) override; - метод, содержащий основную добавления полю реакции на уничтожение корабля;
* ~SetupReactOfDestroyedShip() override; - переопределение деструктора;

Класс Command:

Абстрактный класс команды. Имплементации этого класса будут превращать запросы/действия/реакции в объекты.

Класс содержит следующие публичные виртуальные методы:

* virtual void exec() = 0; - логика запроса/действия/реакции;
* virtual ~Command() = 0; - виртуальный деструктор;

Класс AddRandomSkill:

Класс имплементит Command. Реализует получение рандомной способности при уничтожении корабля.

Класс содержит следующие приватные поля:

* SkillManager &\_skill\_manager; - ссылка на менеджер скилов;

Класс содержит следующие публичные методы:

* explicit AddRandomSkill(SkillManager &\_sm); - конструктор, принимающий ссылку на менеджер скилов;
* void exec() override; - метод, содержащий основную логику получения рандомной способности;
* ~AddRandomSkill() override; - переопределение деструктора;

Класс Skill:

Абстрактный класс способности, наследующийся от iSkill и содержащий методы, которые необходимы всем способностям:

* inline void use(Field &\_fd) override; - метод, посещающий владельца (в слечае способностей, владелец – это поле);

Класс перечисления SkillName:

Класс перечисления, содержащий в себе константы, идентифицирующие способности:

* DOUBLEHIT = 'D' – идентификатор DoubleHit:
* ROCKETATTACK = 'R' – идентификатор RocketAttack;
* SCANNER = 'S' – идентификатор Scanner;

Класс SkillManager:

Класс менеджера способностей.

Класс содержит следующие приватные поля:

* SkillProduction \_skill\_production; - абстрактная фабрика скилов;
* std::queue<std::shared\_ptr<iSkillFactory>> \_queue\_of\_skills; - очередь фабрик скилов;

Класс содержит следующие публичные методы:

* SkillManager(); - конструктор, в котором в очередь добавляются три различные способности в рандомном порядке;
* void produce\_skill(SkillName \_sname); - метод, добавляющий в очередь определённый скил;
* void produce\_skill(); - метод, добавляющий в очередь рандомный скил;
* SkillName front\_skill() const noexcept; - геттер скила, который находится в начале очереди;
* std::shared\_ptr<iSkillFactory> extract\_skill(); - метод, который извлекает фабрику скилов из очереди;
* bool empty() const noexcept; - метод, который проверяет, есть ли способности в очереди;

Класс содержит следующие приватные методы:

* std::vector<SkillName> \_\_get\_random\_skills() const noexcept; - возвращает вектор из трёх скилов в рандомном порядке;

Также в классе реализованы конструкторы копирования, перемещения, операторы присваивания копированием, перемещением, деструктор.

Класс SkillProduction:

Абстрактная фабрика скилов.

Класс содержит следующие приватные поля:

* std::shared\_ptr<DoubleHitFactory> \_double\_hit\_factory; - фабрика скила DoubleHit;
* std::shared\_ptr<RocketAttackFactory> \_rocket\_attack\_factory; - фабрика скила RocketAttack;
* std::shared\_ptr<ScannerFactory> \_scanner\_factory; - фабрика скила Scanner;

Класс содержит следующие публичные методы:

* SkillProduction(); - конструктор, в котором создаются объекты фабрик скилов;
* std::shared\_ptr<iSkillFactory> get\_factory(SkillName \_sn) const noexcept; - выдаёт копию указателя на указанную фабрику скилов, созданную в конструкторе;

Также в классе реализованы конструкторы копирования, перемещения, операторы присваивания копированием, перемещением, деструктор.

Класс \_\_plug\_for\_react:

Класс-заглушка реакции для скилов, которым эта реакция не нужна.

Класс содержит следующие публичные методы:

* void operator()(const Unit &); - бросает исключение при вызове;

Класс ExtractError:

Класс-исключение, которое бросается при попытке достать скил из пустой очереди. Публично наследуется от std::runtime\_error.

Класс OutOfFieldError:

Класс-исключение, которое бросается при попытке ударить по клетке за пределами поля. Публично наследуется от std::invalid\_argument.

Класс PlacementError:

Класс-исключение, которое бросается при попытке установить корабль вплотную к другому или на перекрестии. Публично наследуется от std::invalid\_argument.

Архитектурные решения:

Ради удобства дальнейшей разработки и масштабирования проекта все сущности были разделены на две основные категории: владельцы и посетители, и одну побочную: команды.

Владельцы содержат в себе основную логику программы, также могут принимать посетителей.

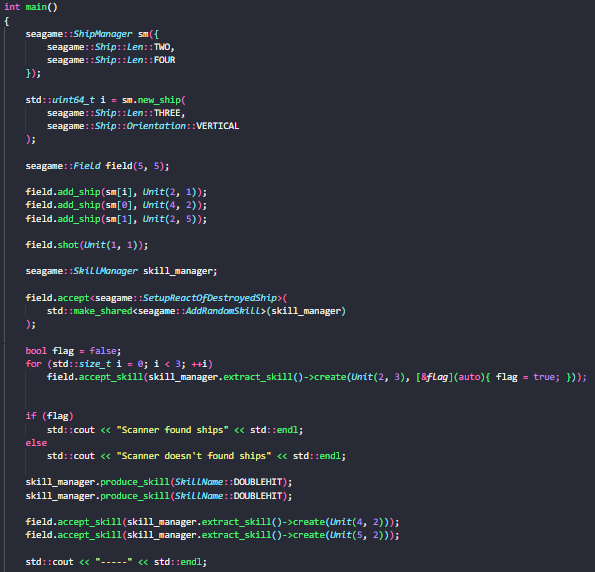
Посетители – сущности, которые посещают владельцев и расширяют их функционал.

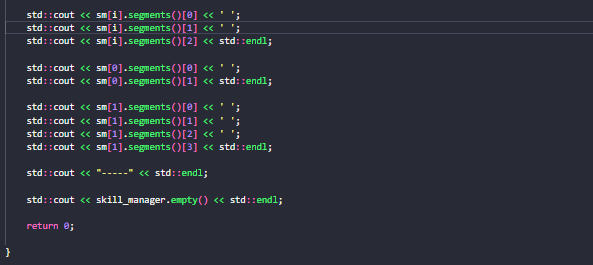
Команды – сущности, превращающие действия/запросы в объекты, что упрощает их эксплуатацию.

По тз необходимо было реализовать три способности, расширяющие возможности пользователя. Эти способности были реализованы как посетители, которые принимались полем и воспроизводили действие над ним.

Также было необходимо реализовать менеджер способностей, который будет представлять из себя обёртку над очередью способностей, доступных к применению. Ради удобства создания способностей (а конкретно: создания способности только в момент, когда она будет применена) были реализованы фабрики способностей. Ради удобства и унификации эксплуатации фабрик способностей была реализована абстрактная фабрика способностей. Её удобство также заключается и в том, что объекты фабрик создаются единожды (при создании абстрактной фабрики), а далее эти объекты просто используются в виде указателей.

Проверка работоспособности написанного кода:

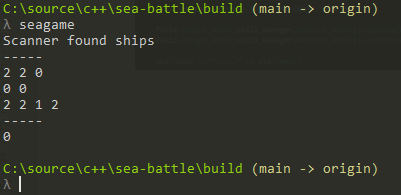




Компиляция:



Запуск:



UML:

См. https://app.creately.com/d/s5KlCAkDtVp/view

## **Выводы**

В ходе выполнения работы было изучено поведение динамического полиморфизма, реализовано несколько интерфейсов, абстрактных классов и их имплементаций, работа с которыми производилась унифицировано через их интерфейсы.