**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

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

**Тема: Логирование, перегрузка операций.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1304 |  | Чернякова В.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2022

## Цель работы.

Реализовать класс/набор классов отслеживающих изменения состояний в программе. Отслеживание должно быть 3-х уровней:

1. Изменения состояния игрока и поля, а также срабатывание событий.
2. Состояние игры (игра начата, завершена, сохранена, и.т.д.).
3. Отслеживание критических состояний и ошибок (поле инициализировано с отрицательными размерами, игрок попытался перейти на непроходимую клетку, и.т.д.).

Реализованы классы для вывода информации разных уровней для в консоль и в файл с перегруженным оператором вывода в поток.

## Требования.

Разработан класс/набор классов, отслеживающий изменения разных уровней.

Разработаны классы для вывода в консоль и файл с соблюдением идиомы RAII и перегруженным оператором вывода в поток.

Разработанные классы спроектированы таким образом, чтобы можно было добавить новый формат вывода без изменения старого кода (например, добавить возможность отправки логов по сети).

Выбор отслеживаемых уровней логирования должен происходить в runtime.

В runtime должен выбираться способ вывода логов (нет логирования, в консоль, в файл, в консоль и файл).

## Примечания.

Отслеживаемые сущности не должны ничего знать о сущностях, которые их логируют.

Уровни логирования должны быть заданными отдельными классами или перечислением.

Разные уровни в логах должны помечаться своим префиксом.

Рекомендуется сделать класс сообщения.

Для отслеживания изменений можно использовать наблюдателя.

Для вывода сообщений можно использовать адаптер, прокси и декоратор.

## Описание архитектурных решений и классов.

**Новые классы.**

Интерфейс события *IEvent:* реализован интерфейс события, в котором определены чисто виртуальные методы, от данного интерфейса будут наследоваться конкретные события – абстрактные классы. Виртуальный метод *virtual void reaction(Player& player) = 0* принимает в качестве аргумента игрока по ссылке, так как все изменения будут происходить с полями игрока. Для очищения памяти, удаления объектов, также прописан виртуальный деструктор *virtual ~IEvent().*

Абстрактный класс *GoodEvent:* реализован абстрактный класс-наследник от интерфейса *IEvent*, в котором определен чисто виртуальный метод *virtual void addGood(Player& player) = 0*, который принимает в качестве аргумента игрока по ссылке и будет переопределен в конкретных событиях для воздействия на игрока. Также для того чтобы класс стал абстрактным определен обычный метод *void addPlayerScore(Player& player),* то есть по умолчанию любое хорошее событие будет добавлять игроку 3 очка.

Абстрактный класс *BadEvent:* реализован абстрактный класс-наследник от интерфейса *IEvent*, в котором определен чисто виртуальный метод *virtual void addBad(Player& player) = 0*, который принимает в качестве аргумента игрока по ссылке и будет переопределен в конкретных событиях для воздействия на игрока. Также для того чтобы класс стал абстрактным определен обычный метод *void cutPlayerScore(Player& player),* то есть по умолчанию любое плохое событие будет отнимать у игрока 4 очка.

Класс *Cave:* реализован класс-наследник от абстрактного класса *GoodEvent.* В нем реализованы конструктор по умолчанию *Cave() = default* и деструктор *~Cave()* со спецификатором *final*, в случае создания наследников, чтобы они не смогли переопределить данный метод. Переопределены два метода - *void reaction(Player& player)* интерфейса события и *void addGood(Player& player) final* абстрактного класса, от которого происходит наследование. Первый метод вызывает переопределенный в данном классе метод *addGood(player)*, который добавляет игроку здоровья и еды в количестве 2 и 1 соответственно, и вызывает метод класса-родителя *addPlayerScore(player)* для начисления игроку очков за попадание на хорошее событие.

Класс *Exit:* реализован класс-наследник от абстрактного класса *GoodEvent.* В нем реализованы конструктор по умолчанию *Exit() = default* и деструктор *~Exit()* со спецификатором *final*, в случае создания наследников, чтобы они не смогли переопределить данный метод. Переопределены два метода - *void reaction(Player& player)* интерфейса события и *void addGood(Player& player) final* абстрактного класса, от которого происходит наследование. Первый метод обращается к методу игрока *setWinner()*, для установления игроку статуса победитель, так как он дошел до выхода.

Класс *Resource:* реализован класс-наследник от абстрактного класса *GoodEvent.* В нем реализованы конструктор по умолчанию *Resource() = default* и деструктор *~Resource()* со спецификатором *final*, в случае создания наследников, чтобы они не смогли переопределить данный метод. Переопределены два метода - *void reaction(Player& player)* интерфейса события и *void addGood(Player& player) final* абстрактного класса, от которого происходит наследование. Первый метод вызывает переопределенный в данном классе метод *addGood(player)*, который добавляет игроку ресурса в количестве 3 единиц, и вызывает метод класса-родителя *addPlayerScore(player)* для начисления игроку очков за попадание на хорошее событие. Также это событие условное, в нем происходит проверка: собрал ли игрок ресурсов больше 5(для проверки вызывается соответствующий метод игрока *getResource())* и не открыт ли выход уже(для проверки вызывается соответствующий метод игрока *getOpenExit()).* В случае выполнения двух условий вызывается метод игрока *setOpenExit()* и открывается выход.

Класс *Clan:* реализован класс-наследник от абстрактного класса *BadEvent.* В нем реализованы конструктор по умолчанию *Clan() = default* и деструктор *~Clan()* со спецификатором *final*, в случае создания наследников, чтобы они не смогли переопределить данный метод. Переопределены два метода - *void reaction(Player& player)* интерфейса события и *void addBad(Player& player) final* абстрактного класса, от которого происходит наследование. Первый метод вызывает переопределенный в данном классе метод *addBad(player)*, который отнимает у игрока здоровье в количестве 2 единиц, и вызывает метод класса-родителя *CutPlayerScore(player)* для убавления игроку очков за попадание на плохое событие. Также вызывается метод игрока *setDead()* для проверки, не умер ли игрок, чтобы завершить игру в данном случае.

Класс *Hungry:* реализован класс-наследник от абстрактного класса *BadEvent.* В нем реализованы конструктор по умолчанию *Hungry() = default* и деструктор *~Hungry()* со спецификатором *final*, в случае создания наследников, чтобы они не смогли переопределить данный метод. Переопределены два метода - *void reaction(Player& player)* интерфейса события и *void addBad(Player& player) final* абстрактного класса, от которого происходит наследование. Первый метод вызывает переопределенный в данном классе метод *addBad(player)*, который отнимает у игрока еду в количестве 2 единиц, и вызывает метод класса-родителя *CutPlayerScore(player)* для убавления игроку очков за попадание на плохое событие. Также вызывается метод игрока *setDead()* для проверки, не умер ли игрок, чтобы завершить игру в данном случае.

**Порождающий паттерн. Абстрактная фабрика.**

Интерфейс фабрики *FactoryEvent*: реализован интерфейс фабрики, в котором определены чисто виртуальные методы, от данного интерфейса будут наследоваться конкретные фабрики-классы. Виртуальный метод *virtual IEvent\* createEvent() = 0* будет переопределён в классах наследниках и будет возвращать указатель на *IEvent*, так как на данном этапе мы не работаем с конкретным событием. Для очищения памяти, удаления объектов, также прописан виртуальный деструктор *virtual ~FactoryEvent().*

Класс *FactoryCave*: реализован класс-наследник от интерфейса *FactoryEvent*, в котором переопределен виртуальный метод *IEvent\* createEvent*() со спецификатором *final*, чтобы в дальнейшем его переопределение было невозможно. Реализация метода заключается в том, что создается объект класса *Cave.*

Класс *FactoryClan*: реализован класс-наследник от интерфейса *FactoryEvent*, в котором переопределен виртуальный метод *IEvent\* createEvent*() со спецификатором *final*, чтобы в дальнейшем его переопределение было невозможно. Реализация метода заключается в том, что создается объект класса *Clan.*

Класс *FactoryExit*: реализован класс-наследник от интерфейса *FactoryEvent*, в котором переопределен виртуальный метод *IEvent\* createEvent*() со спецификатором *final*, чтобы в дальнейшем его переопределение было невозможно. Реализация метода заключается в том, что создается объект класса *Exit.*

Класс *FactoryHungry*: реализован класс-наследник от интерфейса *FactoryEvent*, в котором переопределен виртуальный метод *IEvent\* createEvent*() со спецификатором *final*, чтобы в дальнейшем его переопределение было невозможно. Реализация метода заключается в том, что создается объект класса *Hungry.*

Класс *FactoryResource*: реализован класс-наследник от интерфейса *FactoryEvent*, в котором переопределен виртуальный метод *IEvent\* createEvent*() со спецификатором *final*, чтобы в дальнейшем его переопределение было невозможно. Реализация метода заключается в том, что создается объект класса *Resource.*

**Изменения в классах.**

Класс *Player*: добавлены методы *addHealth(), addFood(), addScore(), addResource(),* которые увеличивают соответствующие поля игрока на определенное значение. Методы *cutHealth(), cutFood(), cutScore(),* которые уменьшают соответствующие поля на определённое значение. Добавлены поля *winner, dead и open\_exit* и соответственные методы для них. Поля добавлены в конструктор и их значение по умолчанию – *false*. Геттеры, для получения значения этих полей, и сеттеры, для установления значения. Для установления значения поля *winner* в *true* происходит проверка: набрано ли ресурсов больше 5. Для установления значения поля *dead* в *true* происходит проверка: значение полей здоровья или еды не меньше ли 0. Сеттер для поля *open\_exit* устанавливает его значение в *true*.

Класс *Cell*: