

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

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «ООП»**

**Тема: Логирование, перегрузка операций**

Студентка гр. 1383

Седова Э.А

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2022

## **Цель работы.**

Реализовать класс/набор классов отслеживающих изменения состояний в программе.

## **Задание.**

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)

## Выполнение работы.

Создан класс интерфейс *Observer*, содержащий одну виртуальную функцию *update(int lvl, std::string str, Level\* level)*. Его наследует класс *Concrete\_obs*, который наследуют классы *Console\_obs* и *File\_obs*. В них происходит вывод логов в консоль и в файл соответственно.

Создан класс интерфейс *Observable*, содержащий две виртуальные функции *addObs(Observer \* obs)* и *notify(int lvl, std::string str, Level\* level)*. Его наследует класс *Subject*. Метод *addObs()* добавляет в вектор *observers* нового наблюдателя. Метод *notify()* получает на вход уровень логирования, некоторое сообщение и объект типа *Level*. Этот метод уведомляет наблюдателей об изменении, вызывая метод *update()*.

Также реализован класс *Level*. Он имеет поля *info*, *statement*, *error*, *console* и *file* по умолчанию инициализированные булевым значением *false*. Класс имеет методы *setInfo(bool info)*, *setStatement(bool statement)*, *setError(bool error)*, *toFile(bool file)*, *setConsole(bool console)* необходимые для изменения значения полей. Объект класса *Level* используется для хранения информации о необходимости вывода конкретных уровней логирования и способе вывода. Ввод необходимых полей производится в конструкторе *Level*.

Классы *Player*, *Field*, *Even*, *Mediator* наследуются от *Subject*, следовательно, являются наблюдаемыми и могут уведомлять наблюдателей об изменениях с помощью метода *notify()*.

Класс *Message* предназначен для создания сообщения, которое позже выводится в консоль и/или в файл. В нём реализованы два метода: *getMsg()* для взятия значения поля *msg* и *getPrefix(int lvl)* для создания префикса лога. Для данного класса создан перегруженный оператор вывода в поток, который выводит префикс в квадратных скобках и само сообщение лога.

## Тестирование программы.

На рисунках 1 - 6 представлено тестирование программы

```
Include INFO? [y/n]
y
Include STATEMENT? [y/n]
y
Include ERROR? [y/n]
y
Write to the console? [y/n]
y
Write to a file? [y/n]
y
[STATEMENT] Start game
[ERROR] Wrong size
available commands: a d w s exit
P F F . F . . + . #
. K . + F # . . . +
. # E F F . . F F +
+ F + F + + + F F #
# . + . F + + + F F
F F # # + F F F F #
+ + F # + + . . . F
+ + + + F # # . + .
. # + # # F # . # #
# F # . . F + F F +
```

Рисунок 1. – Тестирование программы

```

d
[INFO] Change armor
player armor: 50
[INFO] Player stepped into the fire
. P F . F . . + . #
. K . + F # . . . +
. # E F F . . F F +
+ F + F + + + F F #
# . + . F + + + F F
F F # # + F F F F #
+ + F # + + . . . F
+ + + + F # # . + .
. # + # # F # . # #
# F # . . F + F F +

```

Рисунок 2. – Тестирование программы

```

s
[INFO] Get key
[INFO] Player found key
Key!!!
. . F . F . . + . #
. P . + F # . . . +
. # E F F . . F F +
+ F + F + + + F F #
# . + . F + + + F F
F F # # + F F F F #
+ + F # + + . . . F
+ + + + F # # . + .
. # + # # F # . # #
# F # . . F + F F +

```

Рисунок 3. – Тестирование программы

```

s
[INFO] Player break wall
. . F . F . . + . #
. P . + F # . . . +
. . E F F . . F F +
+ F + F + + + F F #
# . + . F + + + F F
F F # # + F F F F #
+ + F # + + . . . F
+ + + + F # # . + .
. # + # # F # . # #
# F # . . F + F F +

s
. . F . F . . + . #
. . . + F # . . . +
. P E F F . . F F +
+ F + F + + + F F #
# . + . F + + + F F
F F # # + F F F F #
+ + F # + + . . . F
+ + + + F # # . + .
. # + # # F # . # #
# F # . . F + F F +

```

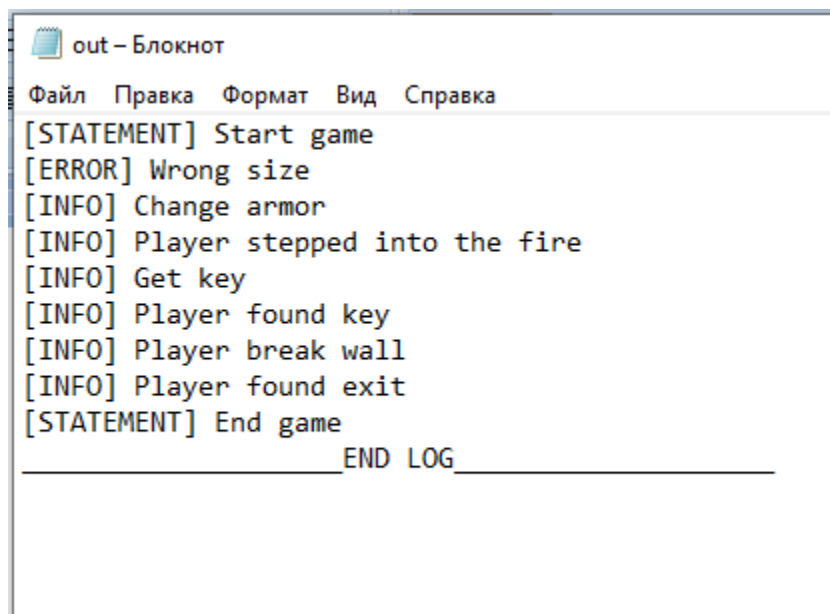
Рисунок 4. – Тестирование программы

```

d
[INFO] Player found exit
[STATEMENT] End game
Win!!!:)

```

Рисунок 5. – Тестирование программы



```
out - Блокнот
Файл  Правка  Формат  Вид  Справка
[STATEMENT] Start game
[ERROR] Wrong size
[INFO] Change armor
[INFO] Player stepped into the fire
[INFO] Get key
[INFO] Player found key
[INFO] Player break wall
[INFO] Player found exit
[STATEMENT] End game
                        END LOG
```

Рисунок 6. – Тестирование программы

## UML-диаграмма межклассовых отношений.

На рисунке 7 изображена UML Диаграмма классов

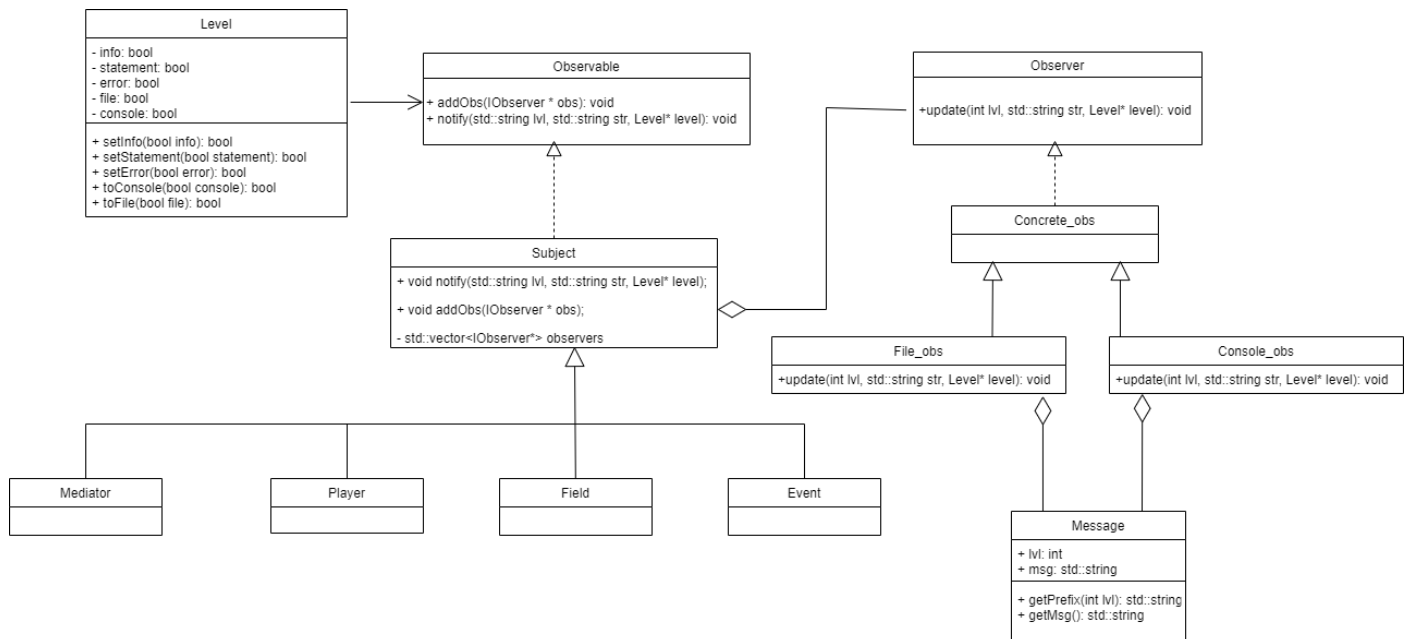


Рисунок 7. – UML Диаграмма классов

### Выводы.

В ходе выполнения работы были реализованы классы, отслеживающие изменения состояний в программе.