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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Объектно-ориентированное программирование»

Тема: Логирование, перегрузка операций

Студентка гр.1381 _____ Рымарь М.И.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Изучить логирование и перегрузку операций, реализовать логирование путём добавления набора классов, которые отслеживают изменения состояний в программе, и классов, которые выводят информацию в файл или консоль с перегруженным оператором вывода в поток.

Задание.

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

- Изменения состояния игрока и поля, а также срабатывание событий
- Состояние игры (игра начата, завершена, сохранена, и.т.д.)
- 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

Выполнение работы.

В ходе выполнения лабораторной работы были созданы классы, которые отвечают за сообщение, которое выводится во время логирования, классы, которые отвечают за вывод в разные потоки, и классы, которые отвечают за логирование определённых уровней.

Был создан интерфейс *Logger* с виртуальным методом *logOutput(Message*)*. Классы, отвечающие за его реализацию, - способы представления логов. В данной лабораторной работе есть два таких класса *ConsoleLog* и *FileLog*, которые ответственны за вывод логов в консоль и файл, соответственно.

FileLogger реализует интерфейс *Logger* (с помощью идиомы RAII). Когда создаётся экземпляр, файл создаётся и открывается, а в деструкторе файл закрывается. Метод *logOutput(Message*)* определён на вывод аргумента в файл.

ConsoleLog реализует интерфейс *Logger*. Метод *logOutput(Message*)* определён на вывод аргумента в консоль. У класса переопределён оператор <<, это упрощает работу.

В классе *Levels* содержится перечисление уровней логирования: *GAME, STATUS, ERROR.*

Класс *Message* отвечает за создание лога. В качестве аргументов конструктор класса принимает уровень логирования *level* и текстовое содержание *text*, в конце устанавливает их в поля. Для удобства устанавливается время создания экземпляра в поле *timeMessage*. Для каждого из этих приватных полей были написаны геттеры. Был переопределён оператор <<: он принимает

два аргумента - *out* типа *ostream* и экземпляр *Message*, далее по очереди передаёт в *out* уровень лога, время и содержание сообщения. Объекты, способные работать с экземплярами данного класса могут выводить его содержимое в определённый поток.

Класс *LogPool* обрабатывает экземпляры *Message*. Он реализовать с помощью паттерна *singleton*. В классе есть поле *instance* и метод *getInstance*, который создаёт экземпляр данного класса только в случае его отсутствия (возвращает указатель на этот единственный экземпляр). Метод *setLevels* устанавливает, какие уровни необходимо логировать, а метод *setLoggers* устанавливает, куда будут выводиться логи (файл, консоль, файл и консоль или никуда). Метод *print* принимает в качестве аргумента экземпляр *Message*, проверяет соответствие его уровня требуемому, вызывает для каждого объекта, способного выводить логи, метод *logOutput* с аргументом, который поступил ему.

В каждый из классов, в которых нужно отслеживать их работу, были внесены изменения. Создавалось сообщение с нужным уровнем и содержанием, передавалось в метод *print* у *LogPool*.

Тестирование.

Интерфейс командной строки при запросах на логирование разных уровней в разные потоки представлен на рисунке 1.

```
Do u want to Output Messages in Console?
yes
Do u want to Output Messages in File?
yes
Do u want to Log Errors?
yes
Do u want to Log Status?
yes
Do u want to Log Game?
yes
```

Рисунок 1 – Запрос на логирование

Интерфейс командной строки при логировании уровня Status, который отвечает за статус игры (начало, конец) представлен на рисунках 2 и 3.

```
|status|        Thu Nov 03 09:30:03 2022        Game started
```

Рисунок 2 – Сообщение о начале игры

```
|status|        Thu Nov 03 09:33:01 2022        Game finished
```

Рисунок 3 – Сообщение о конце игры

Логирование уровня Game, самой игры – передвижение игрока, попадание на клетки с событиями – показано на рисунках 4, 5, 6.

```
Please enter 'y' if you want to leave standard value(10, 10):y
|game|     Thu Nov 03 09:38:49 2022        Events have been
 settled.
|game|  Thu Nov 03 09:38:49 2022        Field have been created.
```

Рисунок 4 – Сообщение о создание поля с событиями

```
Please input player's movement direction (L,U,R,D,E):R
|game|     Thu Nov 03 09:38:56 2022        Player moved to 1 0
```

Рисунок 5 – Сообщение о передвижении игрока

```
Please input player's movement direction (L,U,R,D,E):R
|game|     Thu Nov 03 09:39:20 2022        Player moved to 9 0

|game|  Thu Nov 03 09:39:20 2022        Player have been teleported.
```

Рисунок 6 – Сообщение о срабатывании событии

Логирование уровня Errors, отвечающее за вывод сообщений об ошибках представлено на рисунке 7.

```
Please enter 'y' if you want to leave standard value(10, 10):n
Please input width:0
You've enetered wrong value. A standard
value will be assigned.
|error| Thu Nov 03 09:36:07 2022       Incorrect data input
```

Рисунок 7 – Сообщение об ошибке

Содержимое файла с сообщениями разных уровнях логирования представлено на рисунке 8. В начале и конце выводятся сообщения о статусе игры, а между ними сообщения о текущем состоянии игры.

```
|status|    Thu Nov 03 09:38:45 2022    Game started
|game|  Thu Nov 03 09:38:49 2022    Events have been settled.
|game|  Thu Nov 03 09:38:49 2022    Field have been created.
|game|  Thu Nov 03 09:38:56 2022    Player moved to 1 0
|game|  Thu Nov 03 09:38:59 2022    Player moved to 2 0
|game|  Thu Nov 03 09:39:00 2022    Player moved to 3 0
|game|  Thu Nov 03 09:39:01 2022    Player moved to 4 0
|game|  Thu Nov 03 09:39:02 2022    Player moved to 5 0
|game|  Thu Nov 03 09:39:02 2022    Walls have been settled
|game|  Thu Nov 03 09:39:02 2022    Player moved to 6 0
|game|  Thu Nov 03 09:39:04 2022    Player moved to 7 0
|game|  Thu Nov 03 09:39:19 2022    Player moved to 8 0
|game|  Thu Nov 03 09:39:20 2022    Player moved to 9 0
|game|  Thu Nov 03 09:39:20 2022    Player have been teleported.
|status|    Thu Nov 03 09:53:09 2022    Game finished
```

Рисунок 8 – Файл с сообщениями о логировании

UML-диаграмма межклассовых отношений.

На рисунке 6 представлена UML-диаграмма межклассовых отношений.

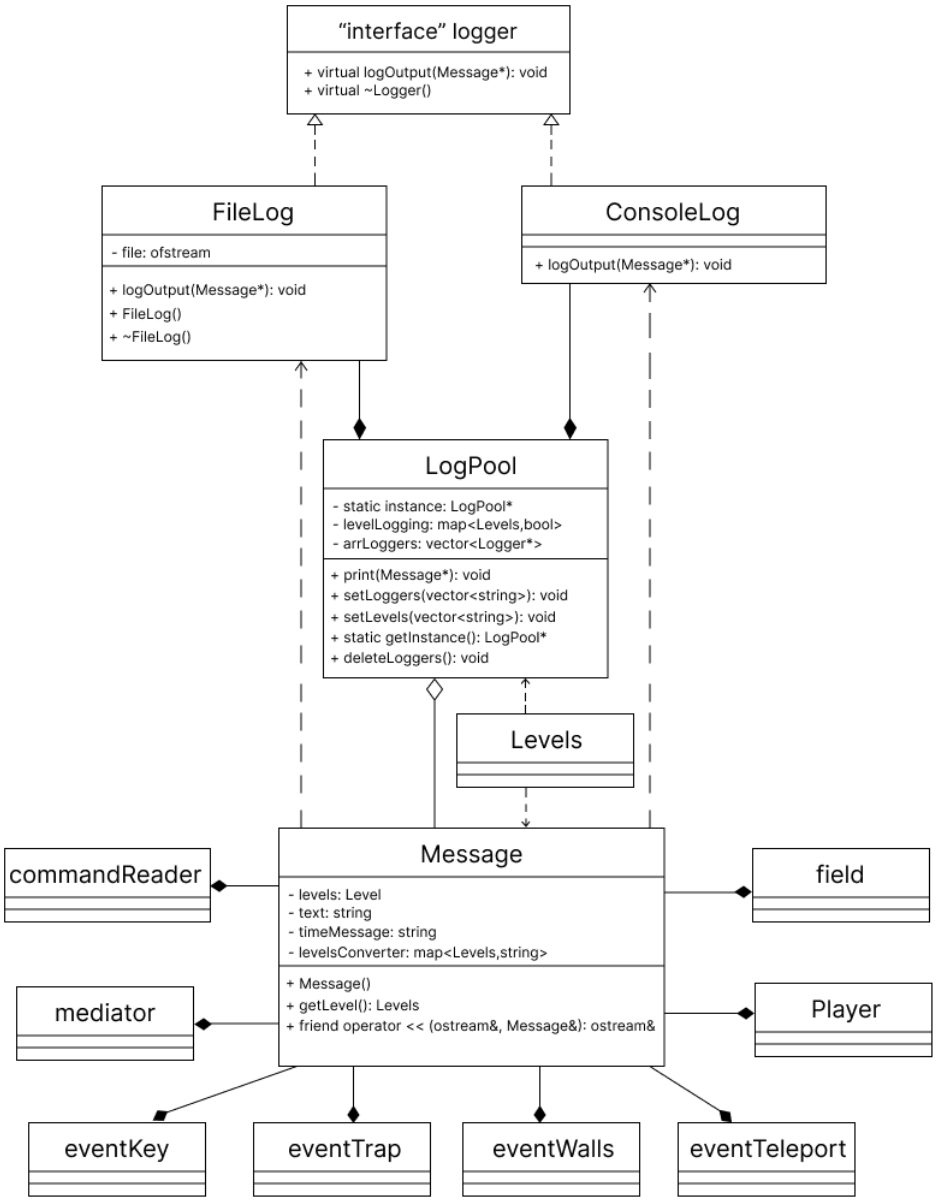

Рисунок 6 – UML-диаграмма

Выводы.

В ходе выполнения лабораторной работы было изучено понятие логирования. Также был реализованы наборы классов, отвечающих за логирование игры и вывод в разные потоки. Был написан класс с перегруженным оператором вывода в поток.