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

Студент гр. 1381 Возмитель В. Е

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Освоение и практика с логированием, применение на практике перегрузку операций.

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)

Выполнение работы.

1) Определяется класс-интерфейс вывода сообщений *Output,* от

которого наследуются классы *LogCmd и LogFile*, выводящие сообщения в консоль и файл соответственно.

Реализуется виртуальный метод класса с модификатором доступа *public* для вывода сообщений:

- *virtual void print (Message &message) = 0;*

1) Определяется класс *LogCmd*, записывающий сообщений в консоль.

Реализуется метод класса с модификатором доступа *public* для вывода сообщений в консоль:

- *void print (Message &message) override*;

1) Определяется класс *LogFile*, записывающий сообщений в файл.

Реализуется методы класса с модификатором доступа *public* для вывода сообщений в файл:

- *void print (Message &message) override* - вывод сообщений в файл.
- *LogFile(std::string name);* – конструктор класса, открывающий файл с именем *name.*
- *~LogFile()*; - деструктор класса, закрывающий файл с именем name.

1) Определяется класс *Subject,* являющийся абстрактным и базовым классом для наблюдаемых объектов классов в игре (*CommandReader, Controller, Event, Field*).

Реализуются методы класса с модификатором доступа *public*:

- *void attach(Observer *observer);*– метод, добавляющий наблюдателя в свой вектор наблюдателей.
- *void detach(Observer *observer);*– метод, удаляющий наблюдателя из своего вектора наблюдателей.
- *void notify(Message& message);* - метод, оповещающий всех

наблюдателей наблюдаемого объекта.

Реализуются методы класса с модификатором доступа *private*:

- *std::vector<Observer*> observers;* - вектор указателей на наблюдателей.

1) Определяется класс *InfoLog*, хранящий уровни логирования и потоки вывода, выбранные пользователем.

Реализуются методы класса с модификатором доступа *private:*

- *std::vector <OUTPUT> outputs;* - вектор потоков вывода, выбранных пользователем.
- *std::vector <LVL> lvls;* - вектор уровней логирования, выбранных пользователем.

Реализуются методы класса с модификатором доступа *public:*

- *InfoLog(std::vector <OUTPUT> outputs, std::vector <LVL> lvls);* - конструктор класса, где реализована инициализация параметров, а также выбор префикса для лога.
- *std::vector <LVL> get_lvls();* - геттер для уровней.
- *std::vector <OUTPUT> get_outputs();* - геттер для потоков вывода.

1) Определяется класс-интерфейс *Observer,* наследниками которого являются наблюдатели.

Реализуются методы класса с модификатором доступа *public:*

- *virtual void update(Message &message) = 0;* - метод обновления информации для наблюдателя.

1) Определяются классы, наследуемые от *Observer: ObsGame, ObsStatus и ObsError*, объектом которых являются наблюдатели, связанные с первым, вторым и третьим уровнем логирования

соответственно.

Реализуются методы класса с модификатором доступа *public:*

- *void update(Message &message) override;* - метод, в который передаётся объект класса сообщения и который, проверяя необходимый уровень логирования данного сообщения, создаёт объект класса *Logger* и вызывает метод *print*.

1) Определяется класс *Logger*, являющийся логгером сообщений и обёрткой для классов *LogFile* и *LogCmd.*

Реализуются методы класса с модификатором доступа *public:*

- *void print(Message &message);* - метод вывода сообщения, где в зависимости от выбора пользователя вызывается метод *print* классов *LogFile* и/или *LogCmd*.

Реализуются методы класса с модификатором доступа *private:*

- *std::vector<Output *> outs;* - вектор объектов классов, выводящих сообщение в файл и/или консоль.
- *std::vector<LVL> lvls;* - вектор уровней логирования, выбранных пользователем.

1) Определяется класс *Message*, объектом которого сообщение, показывающее определенные события в игре.

Реализуются методы класса с модификатором доступа *public:*

- *friend std::ostream &operator<<(std::ostream &os, Message &message);* - перегрузка оператора вывода в поток для объекта данного класса.
- Геттеры и сеттеры всех нижеперечисленных полей.

Реализуются методы класса с модификатором доступа *private:*

- *LVL lvl;* - уровень логирования.
- *std::string message;* - текст сообщений.
- *InfoLog *info;* - указатель на объект класс с информацией о

выборе логирования и способа вывода логов.

- *std::string previx;* - текст префикса с информацией об уровне логирования.

Архитектура программы.

В проекте для логирования используется паттерн Наблюдатель. В случае логирования изменений поля, игрока и срабатывания событий, наблюдаемыми (наследникам класса *Subject*) являются классы Field (в методе передвижения игрока), Event (при срабатывании события) и Player (при изменении его характеристик). Для логирования состояния игры наблюдаются классы *Controller (*при перемещении игрока*)* и *CommandReader (*при вводе клавиш пользователем*).*

Для вызова метода *notify ()* генерируется и передаётся соответствующее сообщение, далее вызывается метод *update (),* где создаётся объект логгера, выводящий переданное сообщение в консоль или/и файл в зависимости от того, что пользователь ввел с клавиатуры.

UML диаграмма.

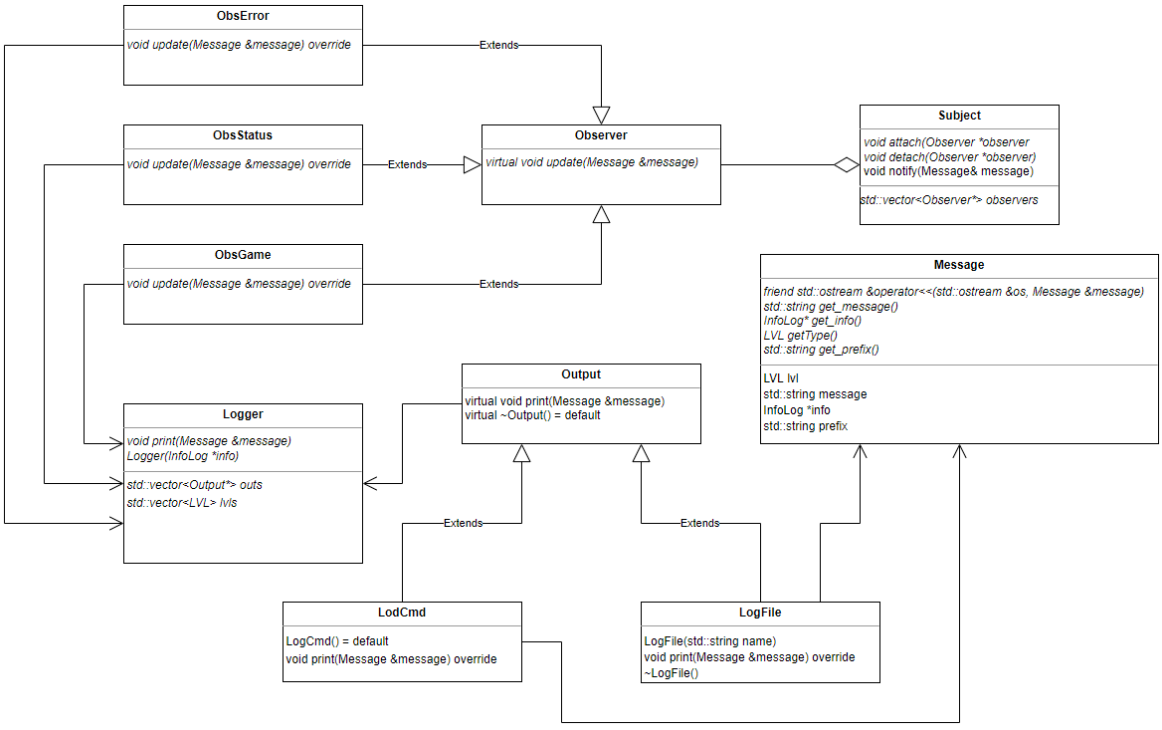

Рис 1 – UML-диаграмма.

Выводы.

Было изучено и логирование, а также отработаны на практике навыки перегрузки операторов.