**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №6**

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

Тема: **Перегрузка операторов / Логирование**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2300 |  | Гаранин Р.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2023

## Цель работы

Реализовать систему классов для отслеживания и вывода сообщений в файл или консоль, не только разработав для этого набор классов “сообщений” для хранения информации о событии, но и перегрузив оператора вывода в поток, чтобы можно было выводить сообщения в различные потоки.

## Задание

а) Реализовать набор классов “сообщений” с общим интерфейсом, который будут срабатывать в определенные моменты и хранить информацию о событии, но не должны хранить сообщение в виде строки. Должны быть реализованы класс для следующих событий:

* Игрок выиграл. Хранится информация о характеристиках игрока
* Игрок проиграл. Хранится информация о координатах клетки на которой событие произошло
* Была запущена новая игра. Хранится информация о размерах поля и стартовой позиции игрока.
* Была введена клавиша и сработала команда. Информация о введенном символе и какая команда сработала.
* Была введена клавиша, но никакая команда не сработала. Информация о введенном символе.

б) Для сообщений перегрузить оператор вывода в поток. Таким образом можно выводить сообщение в различные потоки (cout, файл). При выводе в поток сообщения, должна формироваться строка и подставляться хранимая информация.

в) Разработать систему классов, которые отслеживают сообщения и выводят их в файл и/или консоль. Куда выводить запрашивается у пользователя при запуске программы: никуда, в файл, в консоль, в файл и консоль. Классы, в которых происходит отслеживаемое событие, должны только отправлять сообщение, но не знать куда, то есть только создают сообщение, инициализируя его информацию, и отправляют.

Примечания:

* Система отслеживания должна масштабируема для новых потоков вывода без изменения кода. Для этого вывод в файл и терминал можно обернуть в отдельные классы с общим интерфейсом.
* Для записи в файл придерживайтесь идиомы RAII
* Отслеживаемые сущности не должны знать о том, кто и как их логирует.

## Выполнение работы

В ходе выполнения данной лабораторной работы были реализованы 9 классов: *абстрактный класс Message, класс Message\_log\_win, класс Message\_log\_lose, класс Message\_log\_new\_game, класс Message\_key\_push, класс Message\_key\_not\_work, абстрактный класс Logger, класс Log\_of\_terminal, класс Log\_of\_file.*

*Абстрактный класс Message*: Данный класс является абстрактным, так как содержит только один виртуальный метод *virtual std::string Get\_MESSAGE() = 0,* который и наследуется для классов, генерирующих необходимое сообщение.

*Класс Message\_log\_win*: Данный класс генерирует сообщение о победе игрока и содержит конструктор и метод, который переопределяется через наследования. Конструктор *Message\_log\_win(Player& Value\_player)* – инициализирует приватное поле, а именно игрока и метод std::string *Get\_MESSAGE() override* – данный метод, переопределенный через наследования, генерирует сообщение, которое будет передано далее.

*Класс Message\_log\_lose*: Данный класс генерирует сообщение о проигрыше игрока и содержит конструктор и метод, который переопределяется через наследования. Конструктор *Message\_log\_win(Player& Value\_player)* – инициализирует приватное поле, а именно игрока и метод *std::string Get\_MESSAGE() override* – данный метод, переопределенный через наследования, генерирует сообщение, которое будет передано далее.

*Класс Message\_log\_new\_game*: Данный класс генерирует сообщение о создании игровой сессии и содержит конструктор и метод, который переопределяется через наследования. Конструктор *Message\_log\_new\_game(Field& Value\_field)* – инициализирует приватное поле, а именно игровое поле, для получение и создание сообщения о поле, и метод *std::string Get\_MESSAGE() override* – данный метод переопределен через наследования, и генерирует сообщение, которое будет передано далее.

*Класс Message\_key\_push*: Данный класс генерирует сообщение о нажатой клавиши, которая содержит какое-то действие (команду) и содержит конструктор и метод, который переопределяется через наследования. Конструктор *Message\_key\_push(Interlayer& Value\_layer)* – инициализирует приватное поле, а именно «прослойку», для того чтобы не только получать последнюю нажатую клавишу, но и действие которое она выполнила, кроме того в конструкторе инициализирован вектор, который и хранит действия для клавиш и их сообщение, также был реализован метод *std::string Get\_MESSAGE() override* – данный метод, переопределенный через наследования, генерирует сообщение, которое будет передано далее.

*Класс Message\_key\_not\_work*: Данный класс генерирует сообщение о нажатой «пустой» клавиши, которая не содержит никакого действия, и содержит конструктор и метод, который переопределяется через наследования. Конструктор *Message\_key\_not\_work(Interlayer& Value\_layer)* – инициализирует приватное поле, а именно «прослойку», для того, чтобы получать последнюю нажатую клавишу и метод *std::string Get\_MESSAGE() override* – данный метод, переопределенный через наследования, генерирует сообщение, которое будет передано далее.

*Абстрактный класс Logger*: Данный класс является абстрактным, так как содержит только один виртуальный метод *virtual void log(Message\* message) = 0,* который и наследуется для классов, принимающих и записывающих сообщение.

*Класс Log\_of\_terminal*: Данный класс является наследником *абстрактного класса Logger*, переопределяя метод *void log(Message \*message) override,* данный метод принимает и выводит сообщение в терминал.

*Класс Log\_of\_file*: Данный класс является наследником *абстрактного класса Logger*, переопределяя метод *void log(Message\* message) override,* данный метод принимает и записывает сообщение в выбранный файл. Кроме того, был реализован конструктор *Log\_of\_file()* и деструктор *~Log\_of\_file()*, которые открывают и закрывают файл для следования идиомы RAII.

Разработанную UML-диаграмму см. в приложении А.

## Выводы

В ходе данной лабораторной работы была разработана система классов для отслеживания сообщений и вывода их в файл или в консоль. Так, при запуске игры, у пользователя запрашивается вариант вывода после чего происходят соответствующие действия. Для этого был реализован набор классов “сообщений”, хранящие информацию о событиях, а также был перегружен оператор вывода в поток для вывода сообщений в различные потоки.

# Приложение А UML-ДИАГРАММА РЕАЛИЗОВАННЫХ КЛАССОВ

