**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 |  | Шумов О.Д. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2023

## Цель работы.

Изучить понятия перегрузки операторов и логирование. Реализовать систему классов для вывода сообщений/логов в разные 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

Были созданы 10 классов:

1. *Message* – абстрактный класс сообщений/логов.
2. *MessageWin* – класс наследующий *Message* и реализующий создания сообщения/лога о победе.
3. *MessageLose* – класс наследующий *Message* и реализующий создания сообщения/лога о проигрыше.
4. *MessageNewGame* – класс наследующий *Message* и реализующий создания сообщения/лога о начале новой игры.
5. *MessageKeyCommand* – класс наследующий *Message* и реализующий создания сообщения/лога о верно нажатой клавише и выполненной команде.
6. *MessageKey* – класс наследующий *Message* и реализующий создания сообщения/лога о неверно нажатой клавише.
7. *Logger* – абстрактный класс вывода сообщений/логов в поток.
8. *LoggerTerminal* – класс наследующий *Logger* и реализующий вывод сообщения/лога в терминал.
9. *LoggerFile* – класс наследующий *Logger* и реализующий вывод сообщения/лога в файл.
10. *LoggerTerminalFile* – класс наследующий *Logger* и реализующий вывод сообщения/лога в терминал и файл.

В классе *Message* реализован следующий метод:

1. *virtual std::string get\_message()* – виртуальный метод создания сообщения. Возвращает в качестве результата строку.

Классы *MessageWin*, *MessageLose*, *MessageNewGame*, *MessageKeyCommand*, *MessageKey* наследуют класс Message и реализуют каждый создание своего сообщения.

В классе *MessageWin* реализовано следующее поле:

1. *Player& player\_* – ссылка на объект класса игрока.

Данное поле имеет модификатор доступа private, что защищает его от нежелательного изменения из вне.

В классе *MessageWin* реализованы следующие методы:

1. *std::string get\_message() override* – метод создания сообщения о победе. Выводит данные об игроке.
2. *MessageWin(Player& player)* – конструктор класса сообщения о победе получает на вход ссылку на объект класса игрока.

В классе *MessageLose* реализовано следующее поле:

1. *Player& player\_* – ссылка на объект класса игрока.

Данное поле имеет модификатор доступа private, что защищает его от нежелательного изменения из вне.

В классе *MessageLose* реализованы следующие методы:

1. *std::string get\_message() override* – метод создания сообщения о проигрыше. Выводит координаты клетки на которой умер игрок.
2. *MessageLose(Player& player)* – конструктор класса сообщения о проигрыше получает на вход ссылку на объект класса игрока.

В классе *MessageNewGame* реализовано следующее поле:

1. *Field& field\_* – ссылка на объект класса поля.

Данное поле имеет модификатор доступа private, что защищает его от нежелательного изменения из вне.

В классе *MessageNewGame* реализованы следующие методы:

1. *std::string get\_message() override* – метод создания сообщения о создании новой игры. Выводит размеры поля и координаты стартовой позиции.
2. *MessageNewGame(Field& field)* – конструктор класса сообщения о создании новой игры получает на вход ссылку на объект класса поля.

В классе *MessageKeyCommand* реализованы следующие поля:

1. *Handler& handler\_* – ссылка на объект класса обработчика.
2. *std::unordered\_map <Actions, std::string> command\_* – словарь, где ключ – действие, а значение – команда.

Данные поля имеют модификатор доступа private, что защищает их от нежелательного изменения из вне.

В классе *MessageKeyCommand* реализованы следующие методы:

1. *std::string get\_message() override* – метод создания сообщения о сработанной клавише и выполненном действии. Выводит сработавшую клавишу и выполненной действие.
2. *MessageKeyCommand(Handler& Handler)* – конструктор класса сообщения о сработанной клавише и выполненном действии получает на вход ссылку на объект класса обработчика. Заполняет словарь *command\_*.

В классе *MessageKey* реализовано следующее поле:

1. *Handler& handler\_* – ссылка на объект класса обработчика.

Данное поле имеет модификатор доступа private, что защищает его от нежелательного изменения из вне.

В классе *MessageKey* реализованы следующие методы:

1. *std::string get\_message() override* – метод создания сообщения о сработанной клавише, но невыполненном действии. Выводит сработавшую клавишу.
2. *MessageKey(Handler& Handler)* – конструктор класса сообщения о сработанной клавише, но невыполненном действии получает на вход ссылку на объект класса обработчика.

В классе *Logger* реализован следующий метод:

1. *virtual void log(Message\* message)* – виртуальный метод вывода сообщения/лога в поток. Принимает указатель на объект класса сообщения.

Классы *LoggerTerminal*, *LoggerFile*, *LoggerTerminalFile* наследуют класс *Logger* и реализуют вывод каждый в свой поток.

В классе *LoggerTerminal* реализован следующий метод:

1. *void log(Message\* message) override* – метод вывода сообщения/лога в терминал. Принимает указатель на объект класса сообщения.

В классе *LoggerFile* реализовано следующее поле:

1. *std::ofstream file* – файл для записи сообщений/логов.

Данное поле имеет модификатор доступа private, что защищает его от нежелательного изменения из вне.

В классе *LoggerFile* реализованы следующие методы:

1. *void log(Message\* message) override* – метод вывода сообщения/лога в файл. Принимает указатель на объект класса сообщения.
2. *LoggerFile* – конструктор класса логирования в файла. Открывает файл для записи сообщений/логов.
3. *~LoggerFile* – деструктор класа логирования в файл. Закрывает файл для записи сообщений/файлов.

В классе *LoggerTerminalFile* реализованы следующие поля:

1. *LoggerTerminal logger\_terminal\_* – объект класса логирования в терминал.
2. *LoggerFile logger\_file\_* – объект класса логирования в файл.

Данные поля имеют модификатор доступа private, что защищает их от нежелательного изменения из вне.

В классе *LoggerTerminalFile* реализован следующий метод:

1. *void log(Message\* message) override* – метод вывода сообщения/лога в терминал и в файл. Принимает указатель на объект класса сообщения.

Разработанную UML-диаграмму см. в приложении А.

## Выводы.

Были изучены понятия перегрузки операторов и логирование. Также был реализован набор классов для вывода сообщений/логов в поток.

**Приложение А  
UML-диаграмма**

