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 _____ Смирнов Д. Ю.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2022

Цель работы.

Изучить основы логирования программ, научиться перегружать операторы классов.

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).

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.

Для выполнения лабораторной работы были созданы классы, отвечающие за сообщение, создание сообщения, вывод, логику логирования.

Новые классы:

СООБЩЕНИЕ: Класс *Message*, хранит в себе уровень логирования, текст сообщения, умеет по уровню логирования взять префикс, а также хранит точное время создания сообщения. Данный класс имеет перегруженный оператор вывода в поток, из-за чего именно его и будем передавать в классы для вывода, которые были заданы пользователем.

“ФАБРИКА” СООБЩЕНИЙ: Класс *MessageFactory* нужна для созданий сообщений в любом классе программы. Для создания сообщения нужен уровень и текст сообщения. За данным классом наблюдает класс *LogPool*.

ЛОГГЕР: Класс *LogPool* наблюдает за фабрикой сообщений, если она создала сообщение, которое соответствует уровням текущего логирования, то передает его в классы для вывода.

КЛАССЫ ВЫВОДА: Создан интерфейс *ILogger* – имеет виртуальный метод *print()*, принимающий объект класса Message, так сделано, чтобы можно

было добавить новый формат вывода без изменения старого кода. От данного интерфейса будут реализовываться все классы вывода. Класс *ConsoleLog* – метод *print()* выводит сообщение в консольный поток вывода. Класс *FileLog* – написан с соблюдением идиомы *RAII*, создание объекта данного класса происходит с захватом текстового ресурса, а освобождение происходит в деструкторе данного класса. Метод *print()* выводит сообщение в поток файла, если файл открыт.

ПАТТЕРНЫ:

В данной лабораторной работе был использован паттерн *Singleton* для класса *MessageFactory*, так как сущность данного класса должна существовать в единственном экземпляре и к ней должен быть из любой части программы.

UML диаграмма классов представлена в приложение А.

Выводы.

Были изучены основы логирования программ. В ходе лабораторной работы были написаны классы отвечающие за логирование игры, а также был перегружен оператор вывода в поток для одного из классов.

ПРИЛОЖЕНИЕ А

UML диаграмма классов