МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Объектно-ориентированное программирование»

Тема: Полиморфизм

Студентка гр. 8381 _____ Лисок М.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы.

Реализовать набор классов, для ведения логирования действий и состояний программы.

Задание.

Основные требования:

- Логирование действий пользователя.
- Логирование действий юнитов и базы.
- Реализована возможность записи логов в файл и терминал.
- Взаимодействие с файлами должно быть по идиоме RAII.
- Для логирования состояний перегружен оператор вывода в поток.
- Переключение между разным логированием (логирование в файл, в терминал, без логирования) реализуется при помощи паттерна «Прокси».
- Реализован разный формат записи при помощи паттерна «Адаптер».

Система логирования.

Система логирования в данной работе:

- Классы логирования в файл и терминал(а также прокси класс) обладают возможностью вывода логов в поток.
- Класс адаптер приводит лог в нужный формат для последующего выводу в поток.
- Фасад и команды логируют действия юнитов и базы, затем отправляют соотвествующие логи в адаптер.

Классы логирования в файл, терминал

Классы для логирования в файл и терминал представлены в табл.1.

Таблица 1 - классы для логирования в файл и терминал

Класс	Функция

Logger	Описан в logger/logger.h. Интерфейс класса для логирования в поток. Методы: • printLog(string) выводит в поток переданную строку • setLog(string&) позволяет установить строку • print() позволяет вывести сохраненную строку • getLog() позволяет вернуть сохраненную строку
TerminalLogger	Описан в файлах terminallogger.h и terminallogger.cpp. Предназначен для вывода логов в терминал. • В конструкторе/деструкторе выводится лог о начале/ завершении работы. • В качестве потока используется cout. • Использует **перегрузку оператора вывода в поток.**
FileLogger	Описан в файлах filelogger.h и filelogger.cpp. Предназначен для вывода логов в файл. • В конструкторе файл открывается, в случае ошибки бросается исключение. В деструкторе выводится лог о завершении и файл закрывается. • В качестве потока используется ofstream. • Использует **перегрузку оператора вывода в поток** (та же что и в TerminalLogger).

Переключение между разным логированием.

Переключение между разным логированием (логирование в файл, в терминал, без логирования) по принципу паттерна «Прокси» реализуется в классе ProxyLogger (описан в proxylogger.h, proxylogger.cpp).

- Унаследован от интерфейса Logger.
- В конструктор передается желаемый тип логгера (TOFILE, TOTERMINAL, NOLOG описаны enums.h) и создается соответствующий объект, указатель на который хранится в приватном поле класс. Если же логгирование отключено, указателю передается nullptr

- В деструкторе удаляется логгер, если он был создан.

Адаптер.

Класс Adapter реализован по принципу паттерна «адаптер» (описан в adapter.h и adapter.cpp). В классе реализована обработка информации для формирования строки и вывод её логгером. Логгер хранится в приватном поле класса.

Основные методы класса представлены в табл.2.

Таблица 2 - методы класс Adapter

Метод	Действие
setLogger	Метод для установки логгера
makeLog	Метод вывода лога. Принимаемые параметры: • Тип события (Action act) • Массив параметров map<string, int> В методе происходит вызов одного из методов в зависимости от типа действия и выводится лог с помощью printLog
baseAddLog	• Формирование лога создания базы • Строка лога содержит основную информацию о создании базе: позиция, на которой расположена база или подробный лог ошибки
neutralAddLog	• Формирование лога создания нейтрального объекта • Строка лога содержит основную информацию о создании нейтрального объекта: позиция, на которой расположен нейтральный объект, его тип или подробный лог ошибки

unitAddLog	• Формирование лога создания юнита • Строка лога содержит основную информацию о создании юнита: позиция, на которой расположен юнит, его тип, база в которую он помещен или подробный лог ошибки
attackLog	• Формирование лога атаки • Строка лога содержит основную информацию атаки: позиция, на которой расположен атакуемый, его тип, позиция на которой была атака или подробный лог ошибки
moveLog	• Формирование лога передвижения • Строка лога содержит основную информацию передвижения: позиция, на которой передвигается юнит, его тип или подробный лог ошибки
gameInfoLog	• Формирование лога информация об игре • Строка лога содержит основную информацию об игре: размер поля, количество объектов на поле, количество баз, максимальное количество объектов на поле.
baseInfoLog	• Формирование лога информация о базе • Строка лога содержит основную информацию о базе: позиция, на которой расположена база, номер базы, максимальное количество юнитов, текущее количество юнитов или подробный лог ошибки
userInfoLog	• Формирование лога информация о юните • Строка лога содержит основную информацию о юните: позиция, на которой расположен юнит, его тип, база в которой он находится, основные характеристики или подробный лог ошибки

neutralInfoLog	• Формирование лога информация о нейтральном объекте • Строка лога содержит основную информацию о нейтральном объекте: позиция, на которой расположен нейтральный объект, его тип, основные характеристики или подробный лог ошибки
landCellInfo	• Формирование лога информация о ландшафте • Строка лога содержит основную информацию о ландшафте: позиция, на которой расположена ландшафт, его тип или подробный лог ошибки

Фасад и команды.

Класс фасада хранит объект адаптера в приватном поле. По умолчанию запись логов происходит в терминал, однако тип логгера можно изменить с помощью метода setLogger(LogPlace). При нажатии основных кнопок, вызывается соотвествующий метод команды, в котором записываются параметры для лога, которые в последствии принимает адаптер и формирует лог.

Вывод.

В ходе выполнения лабораторной работы была написана программа, в которой реализованы классы для логирования состояний программы. Были использованы паттерны проектирования, а также принципы объектно-ориентированного программирования.