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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Объектно-ориентированное программирование»

Тема: Полиморфизм

Студент гр. 8303 _____ Удод М.Н.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

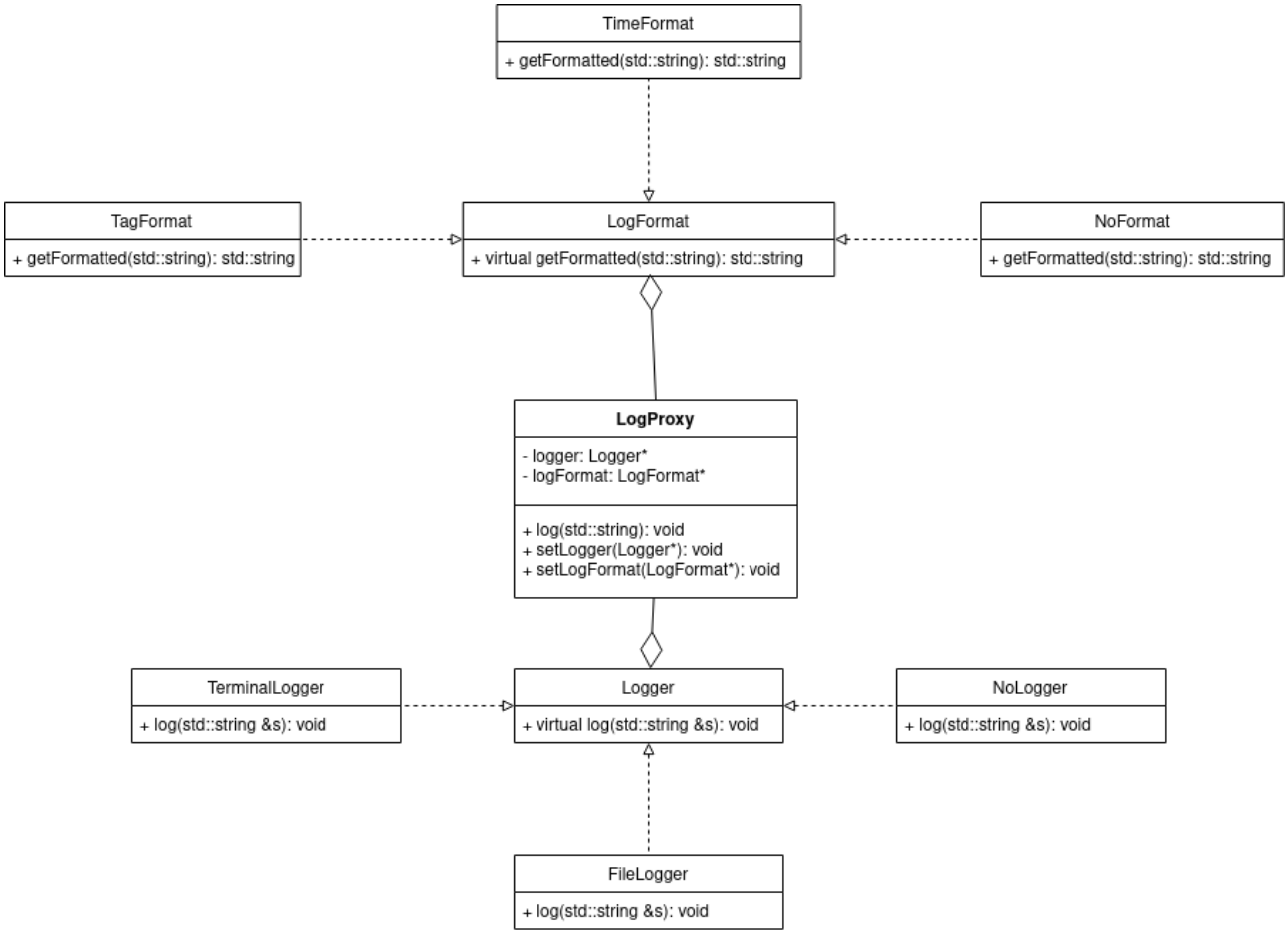
Реализовать набор классов, для ведения логирования действий и состояний программы. Основные требования:

- Логирование действий пользователя
- Логирование действий юнитов и базы

Ход выполнения работы.

1. Реализован набор классов для логирования в файл и терминал. Они наследуются от класса Logger.
2. В классе FileLogger доступ к файлу получается в конструкторе и закрывается в деструкторе. Таким образом, класс соответствует идиоме RAII.
3. У класса Unit был перегружен оператор вывода в поток для более удобной работы с логгированием.
4. Был реализован класс LogProxy, которому можно установить выбранный способом записи логов. Так же, он делегирует все вызовы функций логеру, который содержится в приватном поле.
5. Был реализован набор классов, наследующихся от LogFormat. Все они имеют метод getFormatted, который преборазует исходную строку в отформатированную в соответствии с выбранным форматом вывода.

UML-диаграмма.



Вывод.

В ходе выполнения лабораторной работы были созданы классы для логирования с применением паттернов «Прокси» и «Адаптер».

.

Приложение А. Исходный код программы

1. Logger.h

```
#ifndef UNTITLED13_LOGGER_H
#define UNTITLED13_LOGGER_H

#include <string>

class Logger {

public:

    virtual void log(std::string &s)=0;

};


#endif //UNTITLED13_LOGGER_H
```

2. LogFormat.h

```
#ifndef UNTITLED13_LOGFORMAT_H
#define UNTITLED13_LOGFORMAT_H

#include <string>

class LogFormat {

public:

    virtual std::string getFormatted(std::string &notFormatted)=0;

};


#endif //UNTITLED13_LOGFORMAT_H
```

3. LogProxy.h

```
#include "Loggers/NoLogger.h"
#include "Formats/NoFormat.h"

#include <string.h>
#include <iostream>

class LogProxy {

private:

    Logger *logger;
    LogFormat *logFormat;

public:

    LogProxy(): logger(new NoLogger()), logFormat(new NoFormat()){}
    ~LogProxy(){
        delete logger;
```

```
        delete logFormat;
    }

    friend LogProxy& operator<< (LogProxy &logger, const std::string &s){
        logger.log(s);
        return logger;
    }

    friend LogProxy& operator<< (LogProxy &logger, const int i){
        logger.log(std::to_string(i));
        return logger;
    }

    void log(std::string s){
        std::string formatted = logFormat->getFormatted(s);
        logger->log(formatted);
    }

    void setLogger(Logger *logger1){

        delete logger;
        logger = logger1;
    }

    void setLogFormat(LogFormat *logFormat1){

        delete logFormat;
        logFormat = logFormat1;

    }



};

namespace game{

    static LogProxy log;

}

#endif //UNTITLED13_LOGPROXY_H
```