**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. 0303 |  | Калмак Д.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2021

## **Цель работы.**

Создать классы для логирования информации об объектах игры.

## **Задание.**

Необходимо проводить логирование того, что происходит во время игры.

Требования:

* Реализован класс логгера, который будет получать объект, который необходимо отслеживать, и при изменении его состоянии записывать данную информацию.
* Должна быть возможность записывания логов в файл, в консоль или одновременно в файл и консоль.
* Должна быть возможность выбрать типа вывода логов
* Все объекты должны логироваться через перегруженный оператор вывода в поток.
* Должна соблюдаться идиома RAII

Потенциальные паттерны проектирования, которые можно использовать:

* Адаптер (Adapter) - преобразование данных к нужному формату логирования
* Декоратор (Decorator) - форматирование тВВЦекста для логирования
* Мост (Bridge) - переключение между логированием в файл/консоль
* Наблюдатель (Observer) - отслеживание объектов, которые необходимо логировать
* Синглтон (Singleton) - гарантия логирования в одно место через одну сущность
* Заместитель (Proxy) - подстановка и выбор необходимого логирования

## **Выполнение работы.**

Реализован класс Logger, который является основой для логирования. У класса реализованы поля info\_begin и info\_end для записей в начало и конец лога конкретизирующей информации. В классе Logger переопределен метод вывода в поток и определен виртуальный метод log() для логирования информации.

Класс Logger\_console наследуется от Logger. В классе определяется метод log() для вывода лога в консоль с учетом конкретизирующей информации в начале и конце.

Класс Logger\_file наследуется от Logger. Реализовано поле file для записи в файл лога. Определен метод log() для записи лога в файл с учетом конкретизирующей информации в начале и конце.

Реализован класс Loggers. В нем определено поле вектора loggers\_l для хранения всех логгеров. Определены виртуальные методы add\_logger() и notify().

Реализован класс Loggable. Он наследуется от Loggers. В нем определен метод add\_logger() для добавления логгера, определен метод notify() для добавления информации в логгеры. Игровой объект наследуется от класса Loggable.

Разработанный код см. в Приложении А.

Разработанную диаграмму классов UML см. в Приложении Б.

## **Тестирование.**

Результаты тестирования представлены в Таблице 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 1 – Результаты тестирования | | | |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 1 | 15 15 | P: Wearing an Armor. Its value: 20  P: Changing health. Its change: 20  P: Interaction with the creature!  B: Changing health. Its change: -15  B: Interaction with the creature!  B: Attack without damage!  B: Changing health. Its change: 20 | Все верно. |

## **Выводы.**

Таким образом, были реализованы классы для логирования информации об объектах игры.

# **ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ**

**Файл Logger.h:**

#ifndef LOGGER\_H

#define LOGGER\_H

#include <iostream>

using namespace std;

class *Logger*{

    protected:

*string* info\_begin;

*string* info\_end;

    public:

        Logger(*string* *info\_begin* = "", *string* *info\_end* = ""): info\_begin(*info\_begin*), info\_end(*info\_end*) {}

        virtual void log(*string* *logs*) = 0;

*Logger* & operator<<(*string* *logs*){

            log(*logs*);

            return \**this*;

        }

};

#endif

**Файл Logger\_console.h:**

#ifndef LOGGER\_CONSOLE\_H

#define LOGGER\_CONSOLE\_H

#include "Logger.h"

class *Logger\_console*: public *Logger*{

    public:

        Logger\_console(*string* *info\_begin* = "", *string* *info\_end* = ""):

*Logger*(*info\_begin*, *info\_end*) {}

        void log(*string* *logs*) override{

            cout << info\_begin << *logs* << info\_end << endl;

        }

};

#endif

**Файл Logger\_file.h:**

#ifndef LOGGER\_FILE\_H

#define LOGGER\_FILE\_H

#include "Logger.h"

#include <fstream>

class *Logger\_file*: public *Logger*{

    private:

*ofstream* file;

    public:

        Logger\_file(*string* *f\_name*, *string* *info\_begin* = "", *string* *info\_end* = ""): *Logger*(*info\_begin*, *info\_end*) {

            file.open(*f\_name*);

        }

        void log(*string* *logs*) override{

            file << info\_begin << *logs* << info\_end << endl;

        }

        ~Logger\_file() {

            file.close();

        }

};

#endif

**Файл Loggers.h:**

#ifndef LOGGERS\_H

#define LOGGERS\_H

#include "Logger.h"

#include <vector>

class *Loggers*{

    protected:

        vector<*Logger* \*> loggers\_l;

    public:

        virtual void add\_logger(*Logger*\* *Logger*) = 0;

        virtual void notify(*string* *logs*) = 0;

};

#endif

**Файл Loggable.h:**

#ifndef LOGGABLE\_H

#define LOGGABLE\_H

#include "Loggers.h"

class *Loggable*: public *Loggers*{

    public:

        void add\_logger(*Logger*\* *Logger*){

            loggers\_l.push\_back(*Logger*);

        }

        void notify(*string* *logs*){

            for (auto& a: loggers\_l)

                \*(a) << *logs*;

        }

};

#endif

# **ПРИЛОЖЕНИЕ Б UML ДИАГРАММА КЛАССОВ**

