Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет среднего профессионального образования

Санкт-Петербург 2019

Специальность: 09.02.07 Информационные системы и программирование

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

по теме: Паттерны проектирования

**Цель работы:** Написать программный логгер Базовые требования:

Выполнил:

Студент группы Y2433

Тертень Д. В.

Проверил:

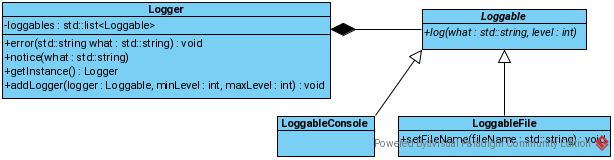
\_\_\_\_\_\_\_\_ Антонов М. Б.

Дата: “\_\_” \_\_\_\_\_\_ 2020г.

Оценка:\_\_\_\_\_\_\_\_\_\_\_

* должен быть класс, реализованный по Singleton, реализуемый методы описывающие уровни ошибок: fatal, critical, error, warning, info, notice;
* должен быть Observer к которому подключаются непосредственно классы, выполняеющие действия логгивания;
* каждый конкретный логгер пишет в свой отдельный поток (файл или консоль);
* каждый логгер реагирует либо: на определённый уровень лога, либо на уровень выше установленного (от notice до fatal);
* все логгеры создаются через Fabric;
* должно быть предусмотрено форматирование сообщений, базовое форматирование: %date% %level% %message%

**UML-диаграммы:**На рисунке 1 представлена UML-диаграмма классов.



*Рисунок 1. Диаграмма классов.*

**Код программы:**

Файл main.cpp:

/\*\*

\* @file main.cpp

\* @author Тертень Даниил

\* @brief Лабораторная работа 1

\*/

#include <iostream>

#include <time.h>

#include <fstream>

#include "Loggable.hpp"

#include "Logger.hpp"

#include "LoggableFactory.hpp"

int main()

{

Logger& logger = Logger::getInstance();

LoggableFileFactory fileFactory;

LoggableConsoleFactory consoleFactory;

logger.addLogger(consoleFactory.createLoggable("console", 1, 5));

logger.addLogger(fileFactory.createLoggable("error", 3, 5));

logger.addLogger(fileFactory.createLoggable("all", 0, 5));

logger.addLogger(fileFactory.createLoggable("critical", 4, 4));

logger.notice("Notice");

logger.warning("Warning");

logger.info("Information");

logger.error("Error");

logger.critical("Critical error");

logger.fatal("Fatal error");

return 0;

}

Файл Logger.hpp:

#include <list>

using namespace std;

/\*\*

\* @file Logger.hpp

\* @author Тертень Даниил

\* @brief Система логгирования

\*/

class Logger

{

private:

Logger() {}

Logger( const Logger&);

Logger& operator=( Logger& );

list<Loggable\*> loggables;

void log(string what, int level)

{

/\*\*

\* @brief Оповестить о событии подписанные средства логгирования

\* @param what Сообщение ошибки

\* @param level Уровень ошибки

\*/

for(list<Loggable\*>::iterator iter = loggables.begin(); iter != loggables.end(); iter++)

(\*iter)->log(what, level);

}

public:

static Logger& getInstance()

{

/\*\*

\* @brief Получить экземпляр класса-одиночки

\* @return Всегда один и тот же экземпляр класса

\*/

static Logger instance;

return instance;

}

void addLogger(Loggable \*logger)

{

/\*\*

\* @brief Подписывает средство логгирования на события

\* @param logger Указатель на средство логгирования

\*/

loggables.push\_back(logger);

}

void notice(string what)

{

/\*\*

\* @brief Уведомление нулевого уровня

\* @param what Сообщение

\*/

log(what, 0);

}

void info(string what)

{

/\*\*

\* @brief Уведомление первого уровня

\* @param what Сообщение

\*/

log(what, 1);

}

void warning(string what)

{

/\*\*

\* @brief Уведомление второго уровня

\* @param what Сообщение

\*/

log(what, 2);

}

void error(string what)

{

/\*\*

\* @brief Уведомление третьего уровня

\* @param what Сообщение

\*/

log(what, 3);

}

void critical(string what)

{

/\*\*

\* @brief Уведомление четвёртого уровня

\* @param what Сообщение

\*/

log(what, 4);

}

void fatal(string what)

{

/\*\*

\* @brief Уведомление пятого уровня

\* @param what Сообщение

\*/

log(what, 5);

}

};

Файл Loggable.hpp:

/\*\*

\* @file Loggable.hpp

\* @author Тертень Даниил

\* @brief Средство логгирования

\*/

using namespace std;

class Loggable

{

protected:

int minLevel, maxLevel;

public:

virtual void log(string what, int level)=0;

void setBounds(int newMinLevel, int newMaxLevel)

{

/\*\*

\* @brief Установка границ, в пределах которых средство будет реагировать

\* @param newMinLevel Минимальный уровень

\* @param newMaxLevel Максимальный уровень

\*/

minLevel=newMinLevel;

maxLevel=newMaxLevel;

}

};

class LoggableConsole: public Loggable

{

void log(string what, int level) override

{

/\*\*

\* @brief Запись сообщения в консоль

\* @param what Сообщение

\* @param level Уровень ошибки

\*/

if(level >= minLevel && level <= maxLevel){

time\_t seconds = time(NULL);

cout << "Datetime: " << asctime(localtime(&seconds)) << " Level: " << level << " Message: " << what << endl;

}

}

};

class LoggableFile: public Loggable

{

private:

string fileName;

ofstream file;

public:

void log(string what, int level) override

{

/\*\*

\* @brief Запись сообщения в файл

\* @param what Сообщение

\* @param level Уровень ошибки

\*/

if(level >= minLevel && level <= maxLevel){

file.open(fileName, std::ios::app);

time\_t seconds = time(NULL);

file << "Datetime: " << asctime(localtime(&seconds)) << " Level: " << level << " Message: " << what << endl;

file.close();

}

}

void setFileName(string newFileName)

{

/\*\*

\* @brief Установка имени файла

\* @param newFileName Новое имя файла

\*/

fileName = newFileName;

}

};

Файл LoggableFactory.hpp:

/\*\*

\* @file LoggableFactory.hpp

\* @author Тертень Даниил

\* @brief Фабрика средств логгирования

\*/

class LoggableFactory

{

public:

virtual Loggable\* createLoggable(string name, int minLevel, int maxLevel)=0;

};

class LoggableConsoleFactory: public LoggableFactory

{

public:

LoggableConsole\* createLoggable(string name, int minLevel, int maxLevel) override

{

/\*\*

\* @brief Создание нового средства ызаписи в консоль

\* @param name Имя средства

\* @param minLevel Минимальный уровень срабатывания

\* @param maxLevel Максимальный уровень срабатывания

\*/

LoggableConsole \*newLoggable = new LoggableConsole();

newLoggable->setBounds(minLevel, maxLevel);

return newLoggable;

}

};

class LoggableFileFactory: public LoggableFactory

{

public:

LoggableFile\* createLoggable(string name, int minLevel, int maxLevel) override

{

/\*\*

\* @brief Создание нового средства записи в файл

\* @param name Имя средства

\* @param minLevel Минимальный уровень срабатывания

\* @param maxLevel Максимальный уровень срабатывания

\*/

LoggableFile \*newLoggable = new LoggableFile();

newLoggable->setBounds(minLevel, maxLevel);

newLoggable->setFileName(name);

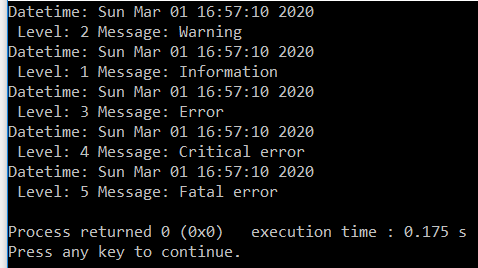
return newLoggable;

}

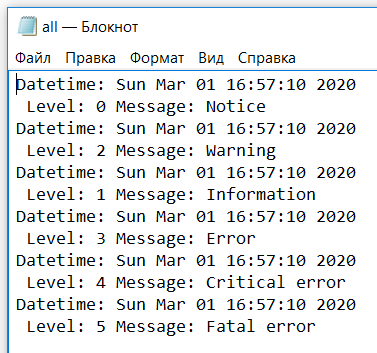
};

**Протокол программы:**

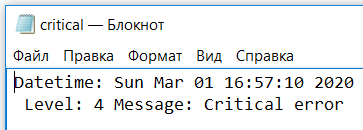
На рисунке 2 представлен консольный вывод программы, на рисунках 3-5 представлен файловый вывод программы.

**

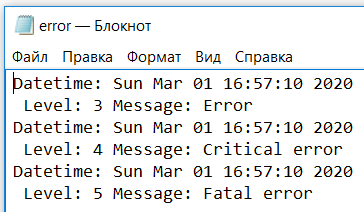
*Рисунок 2 – Консольный вывод программы*

**

*Рисунок 3 – Файл all*

**

*Рисунок 4 – Файл critical*

**

*Рисунок 5 – Файл error*

**Вывод:** в ходе лабораторной работы были реализованы паттерны Singleton, Factory Method, Observer.