МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ образовательное учреждениевысшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации



**ОТЧЁТ по Лабораторной работе №5**

**по дисциплине «ПРОГРАММИРОВАНИЕ»**

**«Наследование. Полиморфизм»**

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АБс-222  Студент: Гатауллин Д.Р. | Преподаватель: Архипова А.Б. |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

**Цели и задачи работы**: Написать программу с использованием принципов ООП и паттернов программирования.

**Задание к работе**:

Паттерн «Декоратор» (Decorator)

Необходимо реализовать программу, которая будет преобразовывать С++ код (или других языков программирования) в HTML. Добавить цвет для синтаксических структур.

**Текст программы**

==========================================================

LB5.cpp

==========================================================

#include <iostream>

#include <fstream>

#include <string>

#include <regex>

// Базовый класс для компонентов кода

class CodeComponent {

public:

// Чисто виртуальная функция для преобразования в HTML

virtual std::string toHtml() = 0;

};

// Класс для представления кода

class Code : public CodeComponent {

private:

std::string code;

public:

// Конструктор принимает строку с кодом

Code(const std::string& code) : code(code) {}

// Преобразование кода в HTML

std::string toHtml() override {

std::string htmlCode = "";

for (char c : code) {

if (c == '<') {

htmlCode += "&lt;";

}

else if (c == '>') {

htmlCode += "&gt;";

}

else {

htmlCode += c;

}

}

return htmlCode;

}

};

// Базовый класс для декораторов кода

class CodeDecorator : public CodeComponent {

protected:

CodeComponent\* component; // Компонент, который декорируется

public:

// Конструктор принимает компонент для декорирования

CodeDecorator(CodeComponent\* component) : component(component) {}

// Преобразование кода в HTML

std::string toHtml() override {

return component->toHtml();

}

};

// Декоратор для подсветки ключевых слов

class KeywordDecorator : public CodeDecorator {

public:

// Конструктор принимает компонент для декорирования

KeywordDecorator(CodeComponent\* component) : CodeDecorator(component) {}

// Преобразование кода в HTML с подсветкой ключевых слов

std::string toHtml() override {

std::string html = CodeDecorator::toHtml();

// Однострочные комментарии

html = std::regex\_replace(html, std::regex("//.\*"), "<span style='color:#57A64A;'>$&</span>");

// Многострочные комментарии

html = std::regex\_replace(html, std::regex("/\\\*(.|[\r\n])\*?\\\*/"), "<span style='color:#57A64A;'>$&</span>");

// Ключевые слова

html = std::regex\_replace(html, std::regex("\\b(int|float|double|char|void|bool|short|long|signed|unsigned|const|static|volatile|if|else|for|while|do|break|continue|return|switch|case|default|class|struct|typedef|enum|namespace|template|this|public|private|protected|virtual|operator|sizeof|dynamic\_cast|static\_cast|reinterpret\_cast|const\_cast|new|delete|throw|try|catch|explicit|friend|inline|register|extern|mutable|using|namespace|std)\\b"), "<span style='color:#569CD6;'>$&</span>");

// Строки

html = std::regex\_replace(html, std::regex("\"(.\*?)\""), "<span style='color:#D69D85;'>$&</span>");

// Числа

html = std::regex\_replace(html, std::regex("\\b(\\d+\\.?\\d\*|\\.\\d+)\\b"), "<span style='color:#B5CEA8;'>$&</span>");

// Препроцессорные директивы

html = std::regex\_replace(html, std::regex("^#\\w+"), "<span style='color:#9CDCFE;'>$&</span>");

// Типы данных

html = std::regex\_replace(html, std::regex("\\b(std::\\w+|std::string|std::regex)\\b"), "<span style='color:#4EC9B0;'>$&</span>");

// Функции

html = std::regex\_replace(html, std::regex("\\b(main|printf|cin|cout|cerr|ifstream|ofstream|istream|ostream|getline|strncpy|strcmp|strlen|strcat|memcpy|malloc|free|new|delete|throw|catch|try|std::\\w+|std::ifstream|std::ofstream|std::cerr|std::cout)\\b"), "<span style='color:#DCDCAA;'>$&</span>");

return html;

}

};

int main(int argc, char\* argv[]) {

setlocale(LC\_ALL, "Rus");

if (argc != 2) {

std::cerr << "Использование: " << argv[0] << " <имя\_файла>" << std::endl;

return 1;

}

std::string inputFileName = argv[1];

std::ifstream file(inputFileName);

if (!file.is\_open()) {

std::cerr << "Не удалось открыть файл: " << inputFileName << std::endl;

return 1;

}

std::string code((std::istreambuf\_iterator<char>(file)), std::istreambuf\_iterator<char>());

file.close();

CodeComponent\* rawCode = new Code(code);

CodeComponent\* decoratedCode = new KeywordDecorator(rawCode);

std::string htmlCode = decoratedCode->toHtml();

size\_t lastdot = inputFileName.find\_last\_of(".");

std::string baseName = (lastdot == std::string::npos) ? inputFileName : inputFileName.substr(0, lastdot);

std::string outputFileName = baseName + ".html";

std::ofstream htmlFile(outputFileName);

if (!htmlFile.is\_open()) {

std::cerr << "Не удалось создать HTML файл: " << outputFileName << std::endl;

return 1;

}

htmlFile << "<!DOCTYPE html>\n<html>\n<head>\n<style>\n"

"body { background-color: #1E1E1E; color: #D4D4D4; font-family: 'Consolas', monospace; white-space: pre; }\n"

"span { font-weight: normal; }\n"

".keyword { color: #569CD6; }\n"

".string { color: #CE9178; }\n"

".comment { color: #6A9955; }\n"

".number { color: #B5CEA8; }\n"

".preprocessor { color: #9B9B9B; }\n"

".datatype { color: #4EC9B0; }\n"

".function { color: #DCDCAA; }\n"

"</style>\n</head>\n<body>\n"

<< htmlCode

<< "\n</body>\n</html>";

htmlFile.close();

delete decoratedCode;

delete rawCode;

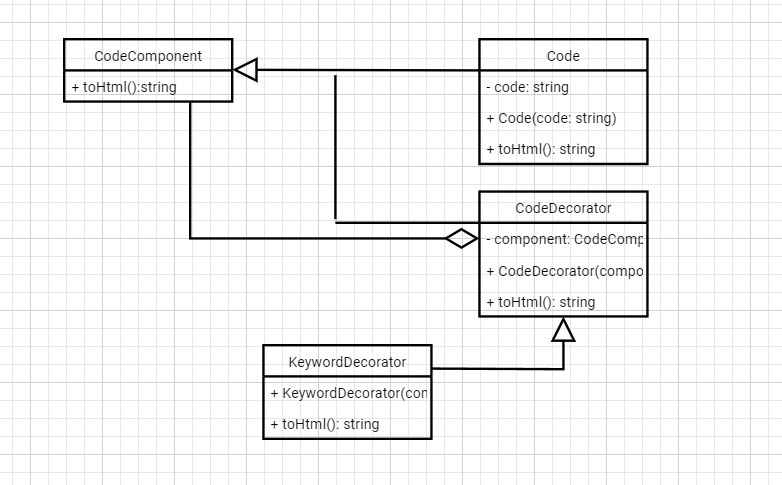
std::cout << "HTML файл успешно создан: " << outputFileName << std::endl;

return 0;

}

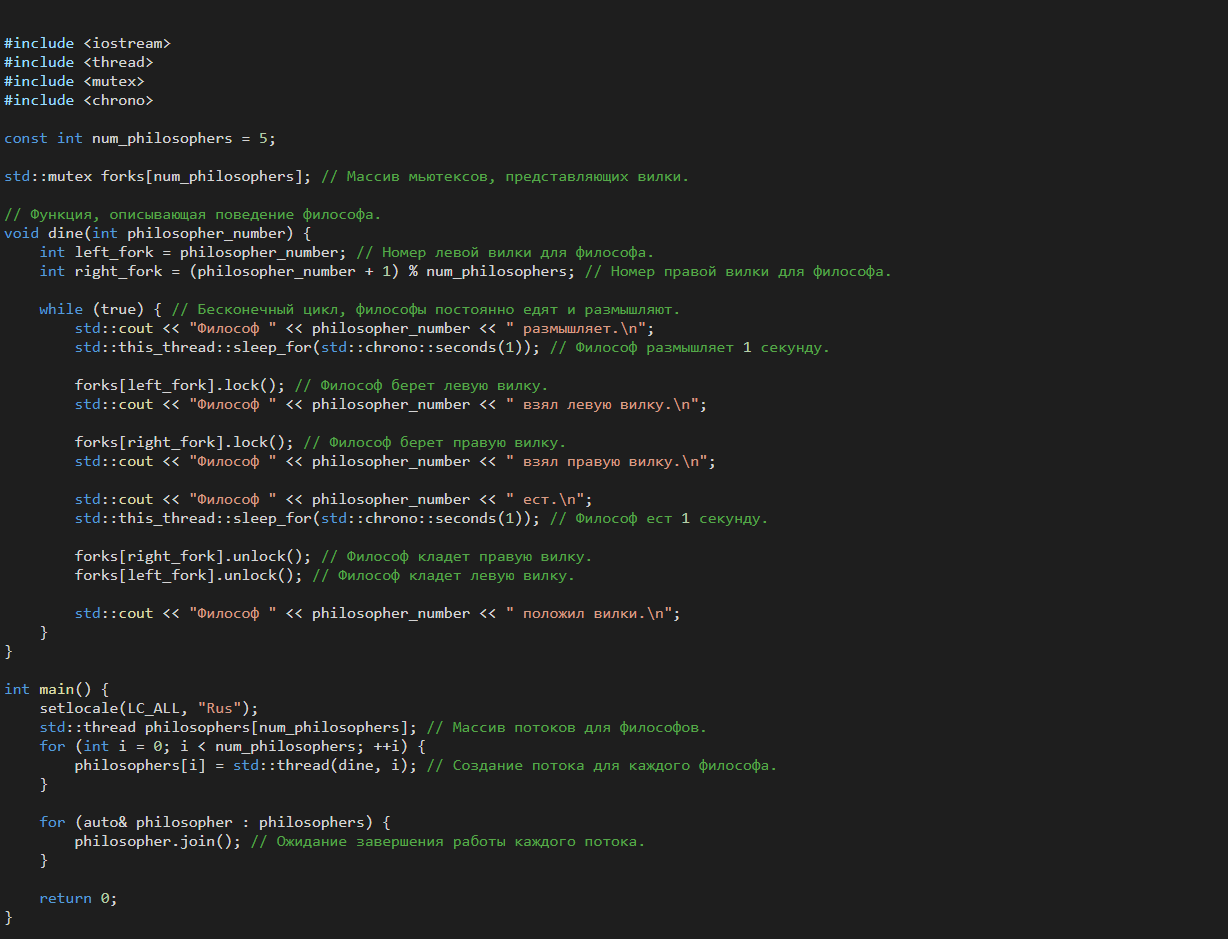
==========================================================

**Диаграмма классов в нотации UML**



**Результат работы программы**





**Вывод**: Программа работает корректно: предоставлен результат выполнения. В своём коде я использовал паттерн Декоратор, структурный паттерн проектирования, который позволяет динамически добавлять новую функциональность к объектам, оборачивая их в удобные обёртки (+: гибкость, модульность, повторное использование кода, -: сложность, трудность в откладке, производительность). Также я использовал такие технологии ООП: наследование: Классы Code, CodeDecorator и KeywordDecorator наследуют от класса CodeComponent. Это позволяет им переопределить метод toHtml() и добавить свою собственную логику преобразования кода в HTML, абстракция: CodeComponent является абстрактным базовым классом, который определяет общий интерфейс для всех компонентов кода. Это позволяет обрабатывать все компоненты кода единообразно, полиморфизм: Метод toHtml() в CodeComponent является виртуальным, что позволяет его переопределить в производных классах. Это означает, что вызов toHtml() будет вызывать соответствующую реализацию в зависимости от типа объекта во время выполнения, инкапсуляция: Данные (атрибуты) каждого класса скрыты внутри класса и доступны только через методы этого класса.