|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
|  |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |
| Институт кибербезопасности безопасности и цифровых технологий |
| Кафедра КБ-2 «Прикладные информационные технологии» |

**Практическая работа № 5**

**Тема «Абстрактная фабрика. Посредник. Строитель. Адаптер»**

по дисциплине: **«Технологии и методы программирования»**.

Подготовил студент 3 курса группы БИСО-01-21 Бондаренко Сергей Александрович

Студенческий билет : 21Б0411

Проверил преподаватель Лесько Сергей Александрович.

Оглавление

[**Абстрактная фабрика (Abstract Factory)** 3](#_Toc169709206)

[**Строитель (Builder)** 6](#_Toc169709207)

[**Адаптер (Adapter)** 11](#_Toc169709208)

[**Посредник (Intermediary, Controller, Mediator)** 15](#_Toc169709209)

# **Абстрактная фабрика (Abstract Factory)**

Листинг программы

#include <iostream>

#include <string>

// Абстрактный класс File

class File {

public:

virtual void open(const std::string& filename) = 0;

virtual void edit() = 0;

virtual void print() = 0;

virtual ~File() {}

};

// Абстрактный класс Highlighter

class Highlighter {

public:

virtual void highlight(const std::string& text) = 0;

virtual ~Highlighter() {}

};

// Абстрактная фабрика

class AbstractFactory {

public:

virtual File\* createFile() = 0;

virtual Highlighter\* createHighlighter() = 0;

virtual ~AbstractFactory() {}

};

// Конкретный класс для обычного текстового файла

class PlainTextFile : public File {

public:

void open(const std::string& filename) override {

std::cout << "Opening plain text file: " << filename << std::endl;

}

void edit() override {

std::cout << "Editing plain text file" << std::endl;

}

void print() override {

std::cout << "Printing plain text file" << std::endl;

}

};

// Конкретный класс для подсветки обычного текста

class PlainTextHighlighter : public Highlighter {

public:

void highlight(const std::string& text) override {

std::cout << "Plain text does not support highlighting: " << text << std::endl;

}

};

// Конкретный класс для XML файла

class XMLFile : public File {

public:

void open(const std::string& filename) override {

std::cout << "Opening XML file: " << filename << std::endl;

}

void edit() override {

std::cout << "Editing XML file" << std::endl;

}

void print() override {

std::cout << "Printing XML file" << std::endl;

}

};

// Конкретный класс для подсветки XML текста

class XMLHighlighter : public Highlighter {

public:

void highlight(const std::string& text) override {

std::cout << "Highlighting XML text: " << text << std::endl;

}

};

// Конкретная фабрика для обычных текстовых файлов

class PlainTextFactory : public AbstractFactory {

public:

File\* createFile() override {

return new PlainTextFile();

}

Highlighter\* createHighlighter() override {

return new PlainTextHighlighter();

}

};

// Конкретная фабрика для XML файлов

class XMLFactory : public AbstractFactory {

public:

File\* createFile() override {

return new XMLFile();

}

Highlighter\* createHighlighter() override {

return new XMLHighlighter();

}

};

// Функция для обработки файла

void processFile(AbstractFactory\* factory, const std::string& filename, const std::string& text) {

File\* file = factory->createFile();

Highlighter\* highlighter = factory->createHighlighter();

file->open(filename);

file->edit();

highlighter->highlight(text);

file->print();

delete file;

delete highlighter;

}

int main() {

AbstractFactory\* plainTextFactory = new PlainTextFactory();

processFile(plainTextFactory, "example.txt", "This is a plain text file.");

delete plainTextFactory;

AbstractFactory\* xmlFactory = new XMLFactory();

processFile(xmlFactory, "example.xml", "<tag>This is an XML file.</tag>");

delete xmlFactory;

return 0;

}

Скриншоты работы программы

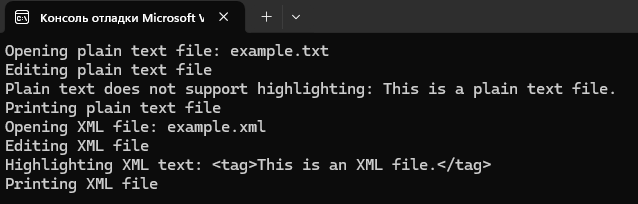


Рисунок 1 – Результат работы программы

Код представляет собой реализацию паттерна проектирования "Абстрактная фабрика" для текстового редактора, который позволяет создавать, редактировать и печатать текстовые файлы, а также поддерживает подсветку ключевых слов для специальных форматов файлов, таких как XML.

Паттерн "Абстрактная фабрика" применяется следующим образом:

1. Абстрактные фабрики (AbstractFactory) определяют интерфейсы для создания абстрактных продуктов (File и Highlighter).
2. Конкретные фабрики (PlainTextFactory и XMLFactory) реализуют интерфейсы абстрактных фабрик для создания конкретных продуктов (PlainTextFile, PlainTextHighlighter, XMLFile, XMLHighlighter).
3. Абстрактные продукты (File и Highlighter) определяют интерфейсы для различных типов файлов и подсветок.
4. Конкретные продукты (PlainTextFile, PlainTextHighlighter, XMLFile, XMLHighlighter) реализуют интерфейсы абстрактных продуктов.
5. Клиентский код использует фабрики для создания объектов без привязки к конкретным классам продуктов. Это позволяет легко добавлять новые форматы файлов и подсветки, создавая новые конкретные классы и фабрики.

# **Строитель (Builder)**

Листинг программы

#include <iostream>

#include <string>

// Абстрактный класс File

class File {

public:

virtual void open(const std::string& filename) = 0;

virtual void edit() = 0;

virtual void print() = 0;

virtual ~File() {}

};

// Абстрактный класс Highlighter

class Highlighter {

public:

virtual void highlight(const std::string& text) = 0;

virtual ~Highlighter() {}

};

// Класс TextEditor, представляющий текстовый редактор

class TextEditor {

private:

File\* file;

Highlighter\* highlighter;

public:

TextEditor() : file(nullptr), highlighter(nullptr) {}

void setFile(File\* f) {

file = f;

}

void setHighlighter(Highlighter\* h) {

highlighter = h;

}

void openFile(const std::string& filename) {

if (file) file->open(filename);

}

void editFile() {

if (file) file->edit();

}

void printFile() {

if (file) file->print();

}

void highlightText(const std::string& text) {

if (highlighter) highlighter->highlight(text);

}

~TextEditor() {

delete file;

delete highlighter;

}

};

// Абстрактный класс Builder

class Builder {

public:

virtual void createFile() = 0;

virtual void createHighlighter() = 0;

virtual TextEditor\* getTextEditor() = 0;

virtual ~Builder() {}

};

// Конкретный класс для обычного текстового файла

class PlainTextFile : public File {

public:

void open(const std::string& filename) override {

std::cout << "Opening plain text file: " << filename << std::endl;

}

void edit() override {

std::cout << "Editing plain text file" << std::endl;

}

void print() override {

std::cout << "Printing plain text file" << std::endl;

}

};

// Конкретный класс для подсветки обычного текста

class PlainTextHighlighter : public Highlighter {

public:

void highlight(const std::string& text) override {

std::cout << "Plain text does not support highlighting: " << text << std::endl;

}

};

// Конкретный класс для XML файла

class XMLFile : public File {

public:

void open(const std::string& filename) override {

std::cout << "Opening XML file: " << filename << std::endl;

}

void edit() override {

std::cout << "Editing XML file" << std::endl;

}

void print() override {

std::cout << "Printing XML file" << std::endl;

}

};

// Конкретный класс для подсветки XML текста

class XMLHighlighter : public Highlighter {

public:

void highlight(const std::string& text) override {

std::cout << "Highlighting XML text: " << text << std::endl;

}

};

// Конкретный строитель для обычных текстовых файлов

class PlainTextBuilder : public Builder {

private:

TextEditor\* editor;

public:

PlainTextBuilder() {

editor = new TextEditor();

}

void createFile() override {

editor->setFile(new PlainTextFile());

}

void createHighlighter() override {

editor->setHighlighter(new PlainTextHighlighter());

}

TextEditor\* getTextEditor() override {

return editor;

}

~PlainTextBuilder() {

delete editor;

}

};

// Конкретный строитель для XML файлов

class XMLBuilder : public Builder {

private:

TextEditor\* editor;

public:

XMLBuilder() {

editor = new TextEditor();

}

void createFile() override {

editor->setFile(new XMLFile());

}

void createHighlighter() override {

editor->setHighlighter(new XMLHighlighter());

}

TextEditor\* getTextEditor() override {

return editor;

}

~XMLBuilder() {

delete editor;

}

};

// Директор, управляющий процессом построения

class Director {

public:

void construct(Builder& builder) {

builder.createFile();

builder.createHighlighter();

}

};

// Главная функция, демонстрирующая использование паттерна "Строитель"

int main() {

Director director;

PlainTextBuilder plainTextBuilder;

director.construct(plainTextBuilder);

TextEditor\* plainTextEditor = plainTextBuilder.getTextEditor();

plainTextEditor->openFile("example.txt");

plainTextEditor->editFile();

plainTextEditor->highlightText("This is a plain text file.");

plainTextEditor->printFile();

delete plainTextEditor;

XMLBuilder xmlBuilder;

director.construct(xmlBuilder);

TextEditor\* xmlEditor = xmlBuilder.getTextEditor();

xmlEditor->openFile("example.xml");

xmlEditor->editFile();

xmlEditor->highlightText("<tag>This is an XML file.</tag>");

xmlEditor->printFile();

delete xmlEditor;

return 0;

}

Скриншоты работы программы

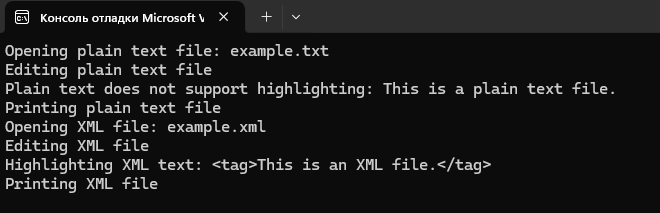


Рисунок 2 – Результат работы программы

Код демонстрирует использование паттерна "Строитель" (Builder) для создания текстового редактора, который поддерживает разные типы файлов и подсветки текста.

Основные компоненты:

1. Абстрактные классы:

* File: Абстрактный класс для файлов.
* Highlighter: Абстрактный класс для подсветки текста.
* Builder: Абстрактный класс строителя для создания объектов TextEditor.

1. Конкретные классы:

* PlainTextFile: Конкретная реализация файла для обычного текста.
* PlainTextHighlighter: Конкретная реализация подсветки для обычного текста.
* XMLFile: Конкретная реализация файла для XML.
* XMLHighlighter: Конкретная реализация подсветки для XML.
* PlainTextBuilder: Конкретный строитель для создания объектов TextEditor с обычным текстом.
* XMLBuilder: Конкретный строитель для создания объектов TextEditor с XML.

1. Класс TextEditor:

* Этот класс представляет текстовый редактор, который может открывать, редактировать, печатать файлы и подсвечивать текст.

1. Класс Director:

* Управляет процессом построения объектов, используя конкретные строители.

Использование:

В main создается объект Director, который использует конкретные строители (PlainTextBuilder и XMLBuilder) для создания различных конфигураций текстового редактора.

Director вызывает методы создания файла и подсветки через конкретные строители.

Созданные редакторы (plainTextEditor и xmlEditor) используются для выполнения операций над файлами, таких как открытие, редактирование, подсветка и печать.

# **Адаптер (Adapter)**

Листинг программы

#include <iostream>

#include <string>

// Интерфейс для работы с файлами

class File {

public:

virtual void open(const std::string& filename) = 0;

virtual void edit() = 0;

virtual void print() = 0;

virtual ~File() {}

};

// Конкретная реализация обычного текстового файла

class PlainTextFile : public File {

public:

void open(const std::string& filename) override {

std::cout << "Opening plain text file: " << filename << std::endl;

}

void edit() override {

std::cout << "Editing plain text file" << std::endl;

}

void print() override {

std::cout << "Printing plain text file" << std::endl;

}

};

// Интерфейс для подсветки текста

class Highlighter {

public:

virtual void highlight(const std::string& text) = 0;

virtual ~Highlighter() {}

};

// Конкретная реализация подсветки для обычного текста (без подсветки)

class PlainTextHighlighter : public Highlighter {

public:

void highlight(const std::string& text) override {

std::cout << "Plain text does not support highlighting: " << text << std::endl;

}

};

// Конкретная реализация XML файла

class XMLFile {

public:

void openXML(const std::string& filename) {

std::cout << "Opening XML file: " << filename << std::endl;

}

void editXML() {

std::cout << "Editing XML file" << std::endl;

}

void printXML() {

std::cout << "Printing XML file" << std::endl;

}

};

// Конкретная реализация подсветки для XML текста

class XMLHighlighter {

public:

void highlightXML(const std::string& text) {

std::cout << "Highlighting XML text: " << text << std::endl;

}

};

// Адаптер для XML файла

class XMLFileAdapter : public File {

private:

XMLFile\* xmlFile;

public:

XMLFileAdapter(XMLFile\* file) : xmlFile(file) {}

void open(const std::string& filename) override {

xmlFile->openXML(filename);

}

void edit() override {

xmlFile->editXML();

}

void print() override {

xmlFile->printXML();

}

};

// Адаптер для XML подсветки

class XMLHighlighterAdapter : public Highlighter {

private:

XMLHighlighter\* xmlHighlighter;

public:

XMLHighlighterAdapter(XMLHighlighter\* highlighter) : xmlHighlighter(highlighter) {}

void highlight(const std::string& text) override {

xmlHighlighter->highlightXML(text);

}

};

// Класс TextEditor, представляющий текстовый редактор

class TextEditor {

private:

File\* file;

Highlighter\* highlighter;

public:

TextEditor() : file(nullptr), highlighter(nullptr) {}

void setFile(File\* f) {

file = f;

}

void setHighlighter(Highlighter\* h) {

highlighter = h;

}

void openFile(const std::string& filename) {

if (file) file->open(filename);

}

void editFile() {

if (file) file->edit();

}

void printFile() {

if (file) file->print();

}

void highlightText(const std::string& text) {

if (highlighter) highlighter->highlight(text);

}

~TextEditor() {

delete file;

delete highlighter;

}

};

// Главная функция, демонстрирующая использование паттерна "Адаптер"

int main() {

TextEditor editor;

// Работа с обычным текстовым файлом

File\* plainTextFile = new PlainTextFile();

Highlighter\* plainTextHighlighter = new PlainTextHighlighter();

editor.setFile(plainTextFile);

editor.setHighlighter(plainTextHighlighter);

editor.openFile("example.txt");

editor.editFile();

editor.highlightText("This is a plain text file.");

editor.printFile();

// Работа с XML файлом через адаптеры

XMLFile\* xmlFile = new XMLFile();

XMLHighlighter\* xmlHighlighter = new XMLHighlighter();

File\* xmlFileAdapter = new XMLFileAdapter(xmlFile);

Highlighter\* xmlHighlighterAdapter = new XMLHighlighterAdapter(xmlHighlighter);

editor.setFile(xmlFileAdapter);

editor.setHighlighter(xmlHighlighterAdapter);

editor.openFile("example.xml");

editor.editFile();

editor.highlightText("<tag>This is an XML file.</tag>");

editor.printFile();

delete plainTextFile;

delete plainTextHighlighter;

delete xmlFileAdapter;

delete xmlHighlighterAdapter;

delete xmlFile;

delete xmlHighlighter;

return 0;

}

Скриншоты работы программы

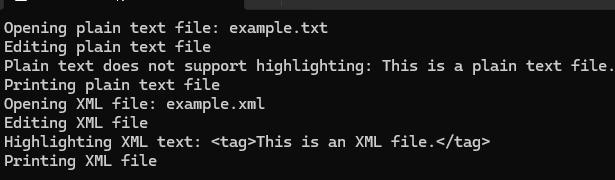


Рисунок 3 – Результат работы программы

Объяснение кода:

1. Интерфейсы и конкретные реализации:

File и Highlighter являются абстрактными интерфейсами для работы с файлами и подсветкой текста.

PlainTextFile и PlainTextHighlighter представляют конкретные реализации для обычного текста.

XMLFile и XMLHighlighter представляют конкретные реализации для XML файлов и подсветки XML текста.

1. Адаптеры:

XMLFileAdapter реализует интерфейс File, позволяя работать с объектами XMLFile через интерфейс File.

XMLHighlighterAdapter реализует интерфейс Highlighter, позволяя работать с объектами XMLHighlighter через интерфейс Highlighter.

1. TextEditor:

TextEditor использует объекты File и Highlighter для выполнения операций открытия, редактирования, печати и подсветки текста.

1. Главная функция:

Создает экземпляры PlainTextFile и PlainTextHighlighter и использует их для работы с обычным текстовым файлом.

Создает экземпляры XMLFile и XMLHighlighter и использует адаптеры XMLFileAdapter и XMLHighlighterAdapter для работы с XML файлом через общий интерфейс.

Преимущества паттерна "Адаптер"

Позволяет использовать существующие классы с несовместимыми интерфейсами.

Повышает повторное использование кода.

Обеспечивает гибкость в расширении функциональности без изменения существующего кода.

# **Посредник (Intermediary, Controller, Mediator)**

Листинг программы

#include <iostream>

#include <string>

#include <vector>

// Абстрактный класс Посредник

class Mediator {

public:

virtual void notify(const std::string& sender, const std::string& event) = 0;

virtual ~Mediator() {}

};

// Абстрактный класс Компонент

class Component {

protected:

Mediator\* mediator;

public:

Component(Mediator\* med = nullptr) : mediator(med) {}

void setMediator(Mediator\* med) {

mediator = med;

}

};

// Конкретный класс File, представляющий файл

class File : public Component {

private:

std::string filename;

public:

File(Mediator\* med = nullptr) : Component(med) {}

void open(const std::string& filename) {

this->filename = filename;

std::cout << "Opening file: " << filename << std::endl;

if (mediator) {

mediator->notify("File", "open");

}

}

void edit() {

std::cout << "Editing file: " << filename << std::endl;

if (mediator) {

mediator->notify("File", "edit");

}

}

void print() {

std::cout << "Printing file: " << filename << std::endl;

if (mediator) {

mediator->notify("File", "print");

}

}

};

// Конкретный класс Highlighter, представляющий подсветку текста

class Highlighter : public Component {

public:

Highlighter(Mediator\* med = nullptr) : Component(med) {}

void highlight(const std::string& text) {

std::cout << "Highlighting text: " << text << std::endl;

if (mediator) {

mediator->notify("Highlighter", "highlight");

}

}

};

// Конкретный класс TextEditor, представляющий текстовый редактор

class TextEditor : public Mediator {

private:

File\* file;

Highlighter\* highlighter;

public:

TextEditor() {

file = new File(this);

highlighter = new Highlighter(this);

}

void notify(const std::string& sender, const std::string& event) override {

if (sender == "File" && event == "open") {

std::cout << "Mediator reacts to File open and triggers highlighter." << std::endl;

highlightText("File content to highlight.");

}

// Другие условия для уведомлений можно добавить здесь

}

void openFile(const std::string& filename) {

file->open(filename);

}

void editFile() {

file->edit();

}

void printFile() {

file->print();

}

void highlightText(const std::string& text) {

highlighter->highlight(text);

}

~TextEditor() {

delete file;

delete highlighter;

}

};

// Главная функция, демонстрирующая использование паттерна "Посредник"

int main() {

TextEditor editor;

editor.openFile("example.txt");

editor.editFile();

editor.printFile();

editor.highlightText("This is some text to highlight.");

return 0;

}

Скриншоты работы программы

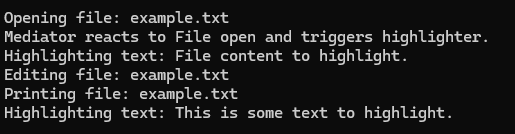


Рисунок 4 – Результат работы программы

Объяснение кода:

1.Посредник (Mediator):

Абстрактный класс Mediator объявляет метод notify, который будет использоваться для уведомления посредника о событиях, происходящих в компонентах.

2.Компоненты (Component):

Абстрактный класс Component хранит ссылку на посредника и может уведомлять его о своих действиях.

File и Highlighter наследуются от Component и взаимодействуют с посредником при выполнении своих операций.

3.Конкретный Посредник (TextEditor):

TextEditor реализует интерфейс Mediator и координирует взаимодействие между File и Highlighter.

В методе notify реализована логика реакции на события. Например, при открытии файла вызывается метод highlightText для подсветки текста.

4.Пример использования:

В главной функции создается объект TextEditor, который координирует работу с файлом и подсветкой.

Вызовы методов openFile, editFile, printFile и highlightText демонстрируют, как компоненты взаимодействуют через посредника.

Преимущества паттерна "Посредник"

Уменьшает связанность между компонентами, делая систему более гибкой и расширяемой.

Центральное управление взаимодействием упрощает изменение поведения системы.

Позволяет легко добавлять новые компоненты и изменять взаимодействие между ними.

Этот паттерн идеально подходит для вашего текстового редактора, так как позволяет легко управлять взаимодействием между различными компонентами, такими как файлы и подсветка текста, улучшая модульность и расширяемость системы.

Вывод: Изучил и реализовал предложенные методы.