Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет инженерно-экономический

Кафедра экономической информатики

**Отчет к лабораторной работе №10**

**по теме**

**«Особенности использования объектно-ориентированного программирования и паттернов**

**проектирования»**

|  |  |
| --- | --- |
| Выполнил: | Вдовенко Н. Д.  студент группы 272303 |
| Проверила: | Салапура М. Н.  старший преподаватель кафедры ЭИ |

Минск 2023

Цель работы: изучить особенности использования объектно-ориентированного программирования и паттернов проектирования.

Теоретические сведения:

Паттерны проектирования в C++:

Паттерны проектирования представляют собой решения для часто возникающих проблем при проектировании программных систем. В C++, реализация паттернов обычно основана на объектно-ориентированных концепциях языка.

1. Порождающие паттерны (Creational Patterns):

- Абстрактная фабрика (Abstract Factory):

- Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не указывая их конкретных классов.

- Одиночка (Singleton):

- Гарантирует, что у класса есть только один экземпляр, и предоставляет глобальную точку доступа к этому экземпляру.

2. Структурные паттерны (Structural Patterns):

- Адаптер (Adapter):

- Преобразует интерфейс одного класса в интерфейс другого класса, который ожидают клиенты. Позволяет классам совместно работать, не изменяя своих кодов.

- Мост (Bridge):

- Разделяет абстракцию и реализацию так, чтобы они могли изменяться независимо.

3. Поведенческие паттерны (Behavioral Patterns):

- Наблюдатель (Observer):

- Определяет зависимость между объектами так, чтобы при изменении состояния одного объекта все его зависимости были уведомлены и обновлены автоматически.

- Состояние (State):

- Позволяет объекту изменять свое поведение при изменении его внутреннего состояния.

Паттерны проектирования предоставляют эффективные и проверенные решения для типичных проблем в разработке программного обеспечения. Использование объектно-ориентированного программирования и паттернов проектирования в C++ позволяет создавать гибкие, поддерживаемые и расширяемые системы.

Контрольные вопросы:

1. Преимущество объектно-ориентированного проектирования заключается в возможности абстрагирования данных и функциональности в виде объектов, что способствует легкости поддержки, повторного использования кода, а также обеспечивает более понятную и структурированную организацию программы.

2. Пример прототипа, реализации и вызова функции с переменным количеством аргументов:

void exampleFunction(int firstArg, ...);

#include <cstdarg>

void exampleFunction(int firstArg, ...) {

va\_list args;

va\_start(args, firstArg);

// Обработка аргументов

va\_end(args);

}

exampleFunction(1, 2, 3, 4);

3. Пример прототипа, реализации и вызова функции, получающей указатель на функцию:

void functionWithCallback(void (\*callback)(int));

void functionWithCallback(void (\*callback)(int)) {

// Вызов функции обратного вызова

callback(42);

}

void callbackFunction(int value) {

// Обработка значения

}

// Передача указателя на функцию в качестве аргумента

functionWithCallback(callbackFunction);

4. Пример создания и использования перечисления:

#include <iostream>

enum Days { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday };

// Использование

Days currentDay = Tuesday;

5. В языке C++ не разрешены виртуальные конструкторы, потому что создание объекта происходит до вызова конструктора, и виртуальные функции не могут быть вызваны до полного создания объекта. Деструктор делается виртуальным для правильного вызова деструкторов при использовании полиморфизма.

6. Паттерн Фабрика используется для создания объектов без указания конкретных классов, делегируя процесс создания подклассам.

7. Паттерн Фабрика используется для создания эффекта виртуального конструктора, где создание объекта делегируется фабричному методу в интерфейсе или абстрактном классе.

8. Эффект виртуального копирующего конструктора достигается с помощью копирующего конструктора в базовом классе, объявленного как виртуальный, и его переопределения в производных классах.

9. Константа времени компиляции в C++ — это значение, вычисляемое на этапе компиляции, например, через constexpr. Она обеспечивает вычисление значения на этапе компиляции, что может повысить эффективность программы.

10. Имя типа можно узнать с использованием оператора typeid или с использованием механизма шаблонов типов, например, с помощью std::type\_info в сочетании с typeid.

Задание:

Необходимо создать возможность конструирования 5 ОБЪЕКТОВ РАЗЛИЧНЫХ ТИПОВ с использованием подходов «ФАБРИКА», «ФАБРИЧНЫЙ МЕТОД С АРГУМЕНТАМИ», «ФАБРИЧНЫЙ КОНСТРУКТОР», «ПОЛИМОРФНАЯ ФАБРИКА», «ПОЛИМОРФНОЕ КОПИРОВАНИЕ». Для всех типов необходимо реализовать возможность их добавления, удаления, редактирования, вывода содержимого объектов на экран в табличном виде и в файл.

Листинг:

#include <iostream>

#include <vector>

#include <fstream>

#include <iomanip>

class Document {

public:

virtual void display() const = 0;

virtual ~Document() {}

};

class MeetingMinutes : public Document {

public:

void display() const override {

std::cout << "Meeting Minutes Document\n";

}

};

class Report : public Document {

public:

void display() const override {

std::cout << "Report Document\n";

}

};

class DocumentFactory {

public:

virtual Document\* createDocument() const = 0;

virtual ~DocumentFactory() {}

};

class MeetingMinutesFactory : public DocumentFactory {

public:

Document\* createDocument() const override {

return new MeetingMinutes();

}

};

class ReportFactory : public DocumentFactory {

public:

Document\* createDocument() const override {

return new Report();

}

};

class DocumentManager {

private:

std::vector<Document\*> documents;

public:

void addDocument(DocumentFactory\* factory) {

documents.push\_back(factory->createDocument());

}

void displayAllDocuments() const {

std::cout << std::left << std::setw(20) << "Document Type" << std::endl;

std::cout << "-------------------\n";

for (const auto& document : documents) {

document->display();

}

}

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

std::ofstream outFile(filename);

if (outFile.is\_open()) {

for (const auto& document : documents) {

document->display();

outFile << std::endl;

}

outFile.close();

std::cout << "Documents saved to file: " << filename << std::endl;

}

else {

std::cerr << "Unable to open file: " << filename << std::endl;

}

}

~DocumentManager() {

for (auto& document : documents) {

delete document;

}

}

};

void displayMenu() {

std::cout << "==========================\n";

std::cout << "1. Add Meeting Minutes\n";

std::cout << "2. Add Report\n";

std::cout << "3. Display All Documents\n";

std::cout << "4. Save to File\n";

std::cout << "5. Exit\n";

std::cout << "==========================\n";

}

int getUserChoice() {

int choice;

std::cout << "Enter your choice: ";

std::cin >> choice;

while (std::cin.fail()) {

std::cin.clear(); // clear input buffer to restore cin to a usable state

std::cin.ignore(INT\_MAX, '\n'); // ignore last input

std::cout << "Invalid input. Please enter a number: ";

std::cin >> choice;

}

return choice;

}

int main() {

DocumentManager manager;

int choice;

do {

displayMenu();

choice = getUserChoice();

switch (choice) {

case 1:

manager.addDocument(new MeetingMinutesFactory());

std::cout << "Meeting Minutes added.\n";

break;

case 2:

manager.addDocument(new ReportFactory());

std::cout << "Report added.\n";

break;

case 3:

manager.displayAllDocuments();

break;

case 4: {

std::string filename;

std::cout << "Enter filename to save: ";

std::cin >> filename;

manager.saveToFile(filename);

break;

}

case 5:

std::cout << "Exiting program.\n";

break;

default:

std::cout << "Invalid choice. Please try again.\n";

}

} while (choice != 5);

return 0;

}

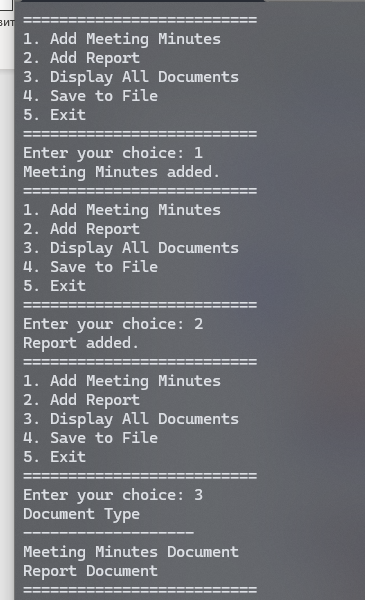


Рисунок 2 – Просмотр информации