Министерство образования Республики Беларусь  
Учреждение образования  
«Брестский государственный технический университет»  
Кафедра ИИТ

Лабораторная работа №3

«Иерархии наследования, динамический полиморфизм в приложениях на языках С++, С#  
По дисциплине: «Объектное моделирование интеллектуальных систем»

Выполнил:  
Студент 3 курса  
Группы ИИ-21  
Кирилович А. А.

Проверила:  
Демидович А.Г.

Брест 2023

**Цель работы:**

1. Изучить особенности организации и реализации множественного наследования классов.

2. Изучить варианты организации иерархии классов с использованием отношений агрегации, композиции, использования.

3. Изучить проектирование иерархий классов с использованием динамического полиморфизма.

Множественное наследование

Анализировать предметную область и разработать классы для демонстрации всех видов наследования. Для этого:

- описать исходный класс. Состав минимальный, включая конструкторы, деструктор, перегруженный оператор. Включить в классы члены-данные, требующие динамического распределения памяти;

- для заданной предметной области разработать дополнительные классы для демонстрации всех видов наследования;

- описать классы на UML (язык - русский);

- описать диаграмму классов

(уровень ассоциаций) на UML (язык - русский) без раскрытия классов;

- уточнить диаграмму классов (типы отношений) на UML (язык - русский) без раскрытия классов;

- описать уточненную диаграмму классов UML с раскрытием классов;

- специфицировать, прокомментировать при необходимости данные и методы класса.

Организация иерархий классов

Для указанной предметной области:

– реализовать иерархии классов, используя наследование и используя агрегацию (состав минимальный, включая конструкторы, деструктор, перегруженный оператор):

- для каждого описать диаграмму UML (язык - русский);

- описать диаграмму UML (язык - английский);

- специфицировать данные и методы класса;

- разработать диаграмму классов (уровень ассоциаций);

- уточнить диаграмму классов (типы отношений). Программировать.

Реализовать приложение для обработки объектов предметной области MIX (целое и вещественное), использовав агрегацию и наследование для организации иерархии классов.

Динамический полиморфизм

1. Продемонстрировать для своей предметной области (без применения виртуальных функций) использование собственных указателей классов и указателей базовых классов для доступа к членам классов иерархии со стороны объектов.
2. Повторить предыдущий пункт с применением виртуальных функций.
3. Реализовать приложение с использованием динамического полиморфизма. Вариант реализации – консольный с управлением через простейшее меню.

**Предметная область: "Университет"**

**1 (Множественное наследование).**

Описание исходного класса "Библиотека":

class Library {

private:

wstring name;

int numBooks;

Book\* books;

public:

Library() {

name = L"";

numBooks = 0;

books = nullptr;

}

Library(wstring \_name, int \_numBooks) {

name = \_name;

numBooks = \_numBooks;

books = new Book[numBooks];

}

~Library() {

delete[] books;

}

Library& operator=(const Library& other) {

if (this != &other) {

name = other.name;

numBooks = other.numBooks;

delete[] books;

books = new Book[numBooks];

for (int i = 0; i < numBooks; ++i) {

books[i] = other.books[i];

}

}

return \*this;

}

};

Класс "Библиотека" содержит данные о названии библиотеки, количестве книг и массиве объектов класса "Книга". Требуется динамическое распределение памяти для хранения массива студентов.

Дополнительные классы для демонстрации всех видов наследования:

a) Класс "Книга":

class Book {

private:

wstring title;

wstring author;

public:

Book() {

title = L"";

author = L"";

}

Book(wstring \_title, wstring \_author) {

title = \_title;

author = \_author;

}

wstring getTitle() const {

return title;

}

void setTitle(wstring \_title) {

title = \_title;

}

wstring getAuthor() const {

return author;

}

void setAuthor(wstring \_author) {

author = \_author;

}

};

б) Класс "Библиотекарь":

class Librarian {

private:

wstring name;

wstring department;

public:

Librarian() {

name = L"";

department = L"";

}

Librarian(wstring \_name, wstring \_department) {

name = \_name;

department = \_department;

}

wstring getName() const {

return name;

}

void setName(wstring \_name) {

name = \_name;

}

wstring getDepartment() const {

return department;

}

void setDepartment(wstring \_department) {

department = \_department;

}

};

в) Класс "Секция":

class LibrarySection {

private:

wstring name;

Librarian\* librarian;

public:

LibrarySection() {

name = L"";

librarian = nullptr;

}

LibrarySection(wstring \_name, Librarian\* \_librarian) {

name = \_name;

librarian = \_librarian;

}

wstring getName() const {

return name;

}

void setName(wstring \_name) {

name = \_name;

}

Librarian\* getLibrarian() {

return librarian;

}

void setLibrarian(Librarian\* \_librarian) {

librarian = \_librarian;

}

};

**Описание классов:**

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Автоматически созданное описание

**2 (Организация иерархий классов).**

Так как структура всех UML будет идентична предыдущему пункту, нет нужды повторно их повторно рисовать.

**Код программы:**

#include <iostream>

#include <string>

using namespace std;

class Book {

private:

string title;

string author;

public:

Book() : title(""), author("") {}

Book(string \_title, string \_author) : title(\_title), author(\_author) {}

string getTitle() const {

return title;

}

void setTitle(string \_title) {

title = \_title;

}

string getAuthor() const {

return author;

}

void setAuthor(string \_author) {

author = \_author;

}

};

class Librarian {

private:

string name;

string department;

public:

Librarian() : name(""), department("") {}

Librarian(string \_name, string \_department) : name(\_name), department(\_department) {}

string getName() const {

return name;

}

void setName(string \_name) {

name = \_name;

}

string getDepartment() const {

return department;

}

void setDepartment(string \_department) {

department = \_department;

}

};

class LibrarySection {

private:

string name;

Librarian\* librarian;

public:

LibrarySection() : name(""), librarian(nullptr) {}

LibrarySection(string \_name, Librarian\* \_librarian) : name(\_name), librarian(\_librarian) {}

string getName() const {

return name;

}

void setName(string \_name) {

name = \_name;

}

Librarian\* getLibrarian() {

return librarian;

}

void setLibrarian(Librarian\* \_librarian) {

librarian = \_librarian;

}

};

class Library {

private:

string name;

int numBooks;

Book\* books;

public:

Library() : name(""), numBooks(0), books(nullptr) {}

Library(string \_name, int \_numBooks, Book\* \_books) : name(\_name), numBooks(\_numBooks), books(\_books) {}

Library& operator=(const Library& other) {

if (this != &other) {

name = other.name;

numBooks = other.numBooks;

delete[] books;

books = new Book[numBooks];

for (int i = 0; i < numBooks; ++i) {

books[i] = other.books[i];

}

}

return \*this;

}

void displayBooks() const {

cout << "Books in the library:" << endl;

for (int i = 0; i < numBooks; ++i) {

cout << "Title: " << books[i].getTitle() << ", Author: " << books[i].getAuthor() << endl;

}

}

};

int main() {

Librarian\* librarian = new Librarian("Петрова", "Научная литература");

Book books[2] = { Book("Physics Textbook", "John Doe"), Book("Mathematics Handbook", "Jane Smith") };

LibrarySection\* section = new LibrarySection("Physics", librarian);

Library library("Central Library", 2, books);

cout << section->getName() << " " << section->getLibrarian()->getName() << endl;

cout << librarian->getName() << " " << librarian->getDepartment() << endl;

library.displayBooks();

delete section;

delete librarian;

return 0;

}

**3 (Динамический полиморфизм).**

1. Демонстрация использования собственных указателей классов и указателей базовых классов для доступа к членам классов иерархии со стороны объектов (без применения виртуальных функций):

**Код программы:**

#include <iostream>

using namespace std;

class BaseClass {

public:

int baseVariable;

void baseMethod() {

cout << "Вызван метод базового класса" << endl;

}

};

class DerivedClass : public BaseClass {

public:

int derivedVariable;

void derivedMethod() {

cout << "Вызван метод производного класса" << endl;

}

};

int main() {

// Создание объекта производного класса

DerivedClass derivedObj;

derivedObj.baseVariable = 42;

derivedObj.derivedVariable = 10;

// Использование указателей базового класса для доступа к объекту производного класса

BaseClass\* basePtr = &derivedObj;

basePtr->baseVariable = 21;

cout << "Значение baseVariable: " << derivedObj.baseVariable << endl;

cout << "Значение derivedVariable: " << derivedObj.derivedVariable << endl;

// Использование указателя производного класса для доступа к объекту производного класса

DerivedClass\* derivedPtr = &derivedObj;

derivedPtr->baseVariable = 33;

derivedPtr->derivedVariable = 99;

cout << "Значение базового класса: " << derivedObj.baseVariable << endl;

cout << "Значение производного класса: " << derivedObj.derivedVariable << endl;

return 0;

}

2. Демонстрация использования виртуальных функций:

**Код программы:**

#include <iostream>

using namespace std;

class BaseClass {

public:

int baseVariable;

virtual void method() {

cout << "Вызвана виртуальная функция базового класса" << endl;

}

};

class DerivedClass : public BaseClass {

public:

int derivedVariable;

void method() override {

cout << "Вызвана виртуальная функция производного класса" << endl;

}

};

int main() {

// Создание объекта производного класса

DerivedClass derivedObj;

derivedObj.baseVariable = 42;

derivedObj.derivedVariable = 10;

// Использование указателей базового класса для вызова виртуальной функции

BaseClass\* basePtr = &derivedObj;

basePtr->method();

return 0;

}

3. Реализовать приложение с использованием динамического полиморфизма через консольное меню:

**Код программы:**

#include <iostream>

using namespace std;

class Shape {

public:

virtual void draw() = 0;

};

class Circle : public Shape {

public:

void draw() override {

cout << "Рисуем круг" << "◯" << endl;

}

};

class Square : public Shape {

public:

void draw() override {

cout << "Рисуем квадрат" << "☐" << endl;

}

};

int main() {

Shape\* shape;

int option;

do {

cout << "Выберите фигуру для отрисовки:" << endl;

cout << "1. Круг" << endl;

cout << "2. Квадрат" << endl;

cout << "0. Выход" << endl;

cout << "Введите номер: ";

cin >> option;

switch (option) {

case 1:

shape = new Circle();

shape->draw();

delete shape;

break;

case 2:

shape = new Square();

shape->draw();

delete shape;

break;

case 0:

cout << "Выход из программы." << endl;

break;

default:

cout << "Ошибка: некорректный выбор." << endl;

break;

}

} while (option != 0);

return 0;

}

**Вывод:** Изучил особенности организации и реализации множественного наследования классов. Изучил варианты организации иерархии классов с использованием отношений агрегации, композиции, использования. Изучил проектирования иерархий классов с использованием динамического полиморфизма.