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-23  
Романюк А.П.

Проверила:  
Демидович А.Г.

Брест 2024

**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: "Автомобили"**

**1 (Множественное наследование).**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <cstring>

// Базовый класс Транспортное средство (Vehicle)

class Vehicle {

protected:

int speed;

char\* fuelType; // Динамическая память

int capacity;

public:

Vehicle(int spd, const char\* fuel, int cap) : speed(spd), capacity(cap) {

fuelType = new char[strlen(fuel) + 1];

strcpy(fuelType, fuel);

}

Vehicle(const Vehicle& other) {

speed = other.speed;

capacity = other.capacity;

fuelType = new char[strlen(other.fuelType) + 1];

strcpy(fuelType, other.fuelType);

}

Vehicle& operator=(const Vehicle& other) {

if (this == &other) return \*this;

delete[] fuelType;

speed = other.speed;

capacity = other.capacity;

fuelType = new char[strlen(other.fuelType) + 1];

strcpy(fuelType, other.fuelType);

return \*this;

}

virtual ~Vehicle() {

delete[] fuelType;

}

virtual void display() const {

std::cout << "Speed: " << speed << " km/h, Fuel: " << fuelType << ", Capacity: " << capacity << std::endl;

}

};

// Класс Автомобиль (Car)

class Car : public Vehicle {

private:

char\* brand;

public:

Car(int spd, const char\* fuel, int cap, const char\* brandName)

: Vehicle(spd, fuel, cap) {

brand = new char[strlen(brandName) + 1];

strcpy(brand, brandName);

}

Car(const Car& other) : Vehicle(other) {

brand = new char[strlen(other.brand) + 1];

strcpy(brand, other.brand);

}

Car& operator=(const Car& other) {

if (this == &other) return \*this;

Vehicle::operator=(other);

delete[] brand;

brand = new char[strlen(other.brand) + 1];

strcpy(brand, other.brand);

return \*this;

}

~Car() {

delete[] brand;

}

void display() const override {

Vehicle::display();

std::cout << "Brand: " << brand << std::endl;

}

};

// Класс Loadable (интерфейс для грузоподъемности)

class Loadable {

protected:

int\* loadCapacity; // Динамическая память

public:

Loadable(int loadCap) {

loadCapacity = new int(loadCap);

}

virtual ~Loadable() {

delete loadCapacity;

}

virtual void load() const {

std::cout << "Load capacity: " << \*loadCapacity << " tons" << std::endl;

}

};

// Класс Грузовик (Truck), множественное наследование

class Truck : public Vehicle, public Loadable {

private:

char\* model;

public:

Truck(int spd, const char\* fuel, int cap, int loadCap, const char\* modelName)

: Vehicle(spd, fuel, cap), Loadable(loadCap) {

model = new char[strlen(modelName) + 1];

strcpy(model, modelName);

}

Truck(const Truck& other) : Vehicle(other), Loadable(other) {

model = new char[strlen(other.model) + 1];

strcpy(model, other.model);

}

Truck& operator=(const Truck& other) {

if (this == &other) return \*this;

Vehicle::operator=(other);

Loadable::operator=(other);

delete[] model;

model = new char[strlen(other.model) + 1];

strcpy(model, other.model);

return \*this;

}

~Truck() {

delete[] model;

}

void display() const override {

Vehicle::display();

Loadable::load();

std::cout << "Model: " << model << std::endl;

}

};

// Меню с динамическим полиморфизмом

void showMenu() {

Vehicle\* vehiclePtr = nullptr;

int choice;

while (true) {

std::cout << "\nВыберите тип транспортного средства:\n";

std::cout << "1. Автомобиль\n";

std::cout << "2. Грузовик\n\n";

std::cin >> choice;

if (choice == 1) {

vehiclePtr = new Car(180, "Gasoline", 5, "Toyota");

}

else if (choice == 2) {

vehiclePtr = new Truck(120, "Diesel", 3, 10, "MAN");

}

else {

break;

}

if (vehiclePtr != nullptr) {

vehiclePtr->display(); // Полиморфизм

delete vehiclePtr;

}

}

}

int main() {

showMenu();

return 0;

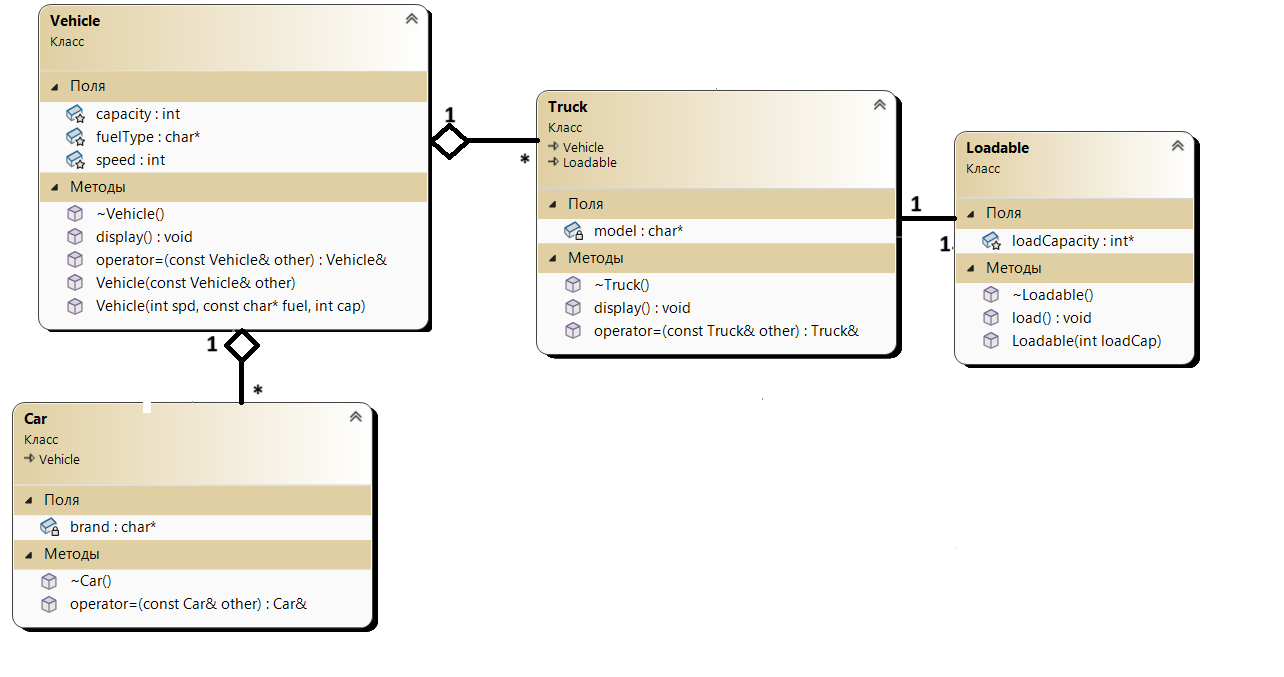
}

**Описание классов:**

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

**Диаграмма классов (уровень ассоциаций и типы отношений):**



**Вывод:**

1. Изучил особенности организации и реализации множественного наследования классов.

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

3. Изучил проектирования иерархий классов с использованием динамического полиморфизма.