Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**Звіт**

Про виконання лабораторної роботи №6

**На тему:**

“Наслідування. Створення та використання ієрархії класів”

**Лектор:**

доц. каф. ПЗ

Коротєєва Т.О.

**Виконав:**

ст. гр. ПЗ-18

Юшкевич А.І.

**Прийняв:**

асис. каф. ПЗ

Далявський В.С.

« … » … 2023 р.

∑ = \_\_\_\_\_\_\_\_\_\_\_\_

Львів – 2023

**Тема**: Наслідування. Створення та використання ієрархії класів.

**Мета**: Навчитися створювати базові та похідні класи, використовувати наслідування різного типу доступу, опанувати принципи використання множинного наслідування. Навчитися перевизначати методи в похідному класі, освоїти принципи такого перевизначення.

Теоретичні відомості

1. **Наслідування**

Наслідуванням називається процес визначення класу на основі іншого класу. На новий (дочірній) клас за замовчуванням поширюються всі визначення змінних екземпляра і методів зі старого (батьківського) класу, але можуть бути також визначені нові компоненти або «перевизначені» визначення батьківських функцій і дано нові визначення. Прийнято вважати, що клас А успадковує свої визначення від класу В, якщо клас А визначений на основі класу В зазначеним способом.

Класи можуть бути пов'язані один з одним різними відношеннями. Одним з основних є відношення клас‑підклас, відоме в об'єктно-орієнтованому програмуванні як наслідування. Наприклад, клас автомобілів Audi 6 є підкласом легкових автомобілів, який в свою чергу входить у більший клас автомобілів, а останній є підкласом класу транспортних засобів, який крім автомобілів включає в себе літаки, кораблі, потяги і т.д. Прикладом подібних відношень є системи класифікації в ботаніці та зоології.

При наслідуванні всі атрибути і методи батьківського класу успадковуються класом-нащадком. Наслідування може бути багаторівневим, і тоді класи, що знаходяться на нижніх рівнях ієрархії, успадкують всі властивості (атрибути і методи) всіх класів, прямими або непрямими нащадками яких вони є.

Крім одиничного, існує і множинне наслідування, коли клас наслідує відразу кілька класів. При цьому він успадкує властивості всіх класів, нащадком яких він є.

При наслідуванні одні методи класу можуть заміщатися іншими. Так, клас транспортних засобів буде мати узагальнений метод руху. У класах-нащадках цей метод буде конкретизований: автомобіль буде їздити, літак – літати, корабель – плавати. Така зміна семантики методу називається поліморфізмом. Поліморфізм – це виконання методом з одним і тим же ім'ям різних дій залежно від контексту, зокрема, від приналежності до того чи іншого класу. У різних мовах програмування поліморфізм реалізується різними способами.

Завдання

1. Розробити ієрархію класів відповідно до варіанту. Набір полів і методів, необхідних для забезпечення функціональної зручності класів, визначити самостійно.

2. Створити базовий (відповідає сутності) та похідні класи (типи сутностей).

3. Використати public, protected наслідування.

4. Використати множинне наслідування (за необхідності).

5. Визначити функцію PrintName() в базовому класі, яка друкує назву відповідного класу.

Перевизначити її в похідних класах.

6. Реалізувати методи варіанта та результати вивести у файл.

7. Продемострувати роботу класів таким чином (на вибір):

a. За допомогою тестів Google Test, які перевірятимуть роботу ієрархії класів.

b. За допомогою інтерактивної програми із демострацією її роботи.

8. Оформити звіт до лабораторної роботи. Включити у звіт UML-діаграму розробленої ієрархії класів та результат роботи тестів.

Варіант №12:

|  |  |
| --- | --- |
| 13 | Розробити ієрархію класів для сутності: **транспортний засіб** (CVehicle). Розробити такі типи транспортних засобів:   * автомобіль (CCar) * вантажівка (CTruck) * мотоцикл (CMotorcycle)   Кожен клас повинен вміти розраховувати:   * відстань (км), яку може проїхати за 1 літр палива із корисним навантаженням (кг); - витрати на обслуговування (за 1000 км). * максимальну корисну масу перевезення. |

# Код програми

## Lab\_06\_OOP:

Header\_Lab\_06.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class CVehicle {

public:

void PrintName() {

cout << "CVehicle\n\n";

}

double GetDistanceWithWeight() {

return distance\_with\_weight;

}

double GetPriceForService() {

return price\_for\_service;

}

double GetMaxWeight() {

return max\_weight;

}

void PrintDistanceWithWeight() {

cout << "Max distance with weight of " << real\_weight << "kg is: " << distance\_with\_weight << "km" << endl << endl;

}

void PrintPriceForService() {

cout << "Price of service for 1000km is: $" << price\_for\_service << endl << endl;

}

void PrintMaxWeight() {

cout << "Max weight Vehicle can handle is: " << max\_weight << "kg" << endl << endl;

}

CVehicle(double crub\_weight\_distance, double base\_weight, double max\_weight, double price\_for\_service, double real\_weight) {

if (real\_weight > base\_weight && real\_weight < max\_weight) {

this->real\_weight = real\_weight;

this->distance\_with\_weight = CalculateDistanceWithWeight(real\_weight, crub\_weight\_distance);

this->price\_for\_service = CalculatePriceForService(price\_for\_service);

this->max\_weight = CalculateMaxWeight(max\_weight);

}

else {

this->real\_weight = base\_weight;

this->distance\_with\_weight = CalculateDistanceWithWeight(this->real\_weight, crub\_weight\_distance);

this->price\_for\_service = CalculatePriceForService(price\_for\_service);

this->max\_weight = CalculateMaxWeight(max\_weight);

}

}

protected:

double real\_weight;

double distance\_with\_weight;

double price\_for\_service;

double max\_weight;

double base\_weight;

string vehicle\_name;

private:

double CalculateDistanceWithWeight(double weight, double crub\_weight\_distance) {

double loss\_per\_hundred\_kg = crub\_weight\_distance / 1000;

double distance\_with\_weight = crub\_weight\_distance - weight / 100 \* loss\_per\_hundred\_kg;

return distance\_with\_weight;

}

double CalculatePriceForService(double price\_for\_service) {

return price\_for\_service;

}

double CalculateMaxWeight(double max\_weight) {

return max\_weight;

}

};

class CCar : public CVehicle {

public:

void PrintName() {

cout << "CCar\n\n";

}

CCar() = delete;

CCar(string vehicle\_name, double additional\_weigh = 0) : CVehicle(crub\_weight\_distance, base\_weight, max\_weight, price\_for\_service, additional\_weigh) {

this->vehicle\_name = vehicle\_name;

}

private:

static double crub\_weight\_distance;

static double max\_weight;

static double base\_weight;

static double price\_for\_service;

};

class CTruck : public CVehicle {

public:

void PrintName() {

cout << "CTruck\n\n";

}

CTruck() = delete;

CTruck(string vehicle\_name, double additional\_weigh = 0) : CVehicle(crub\_weight\_distance, base\_weight, max\_weight, price\_for\_service, additional\_weigh) {

this->vehicle\_name = vehicle\_name;

}

private:

static double crub\_weight\_distance;

static double max\_weight;

static double base\_weight;

static double price\_for\_service;

};

class CMotorcycle : public CVehicle {

public:

void PrintName() {

cout << "CMotorcycle\n\n";

}

CMotorcycle() = delete;

CMotorcycle(string vehicle\_name, double additional\_weigh = 0) : CVehicle(crub\_weight\_distance, base\_weight, max\_weight, price\_for\_service, additional\_weigh) {

this->vehicle\_name = vehicle\_name;

}

private:

static double crub\_weight\_distance;

static double max\_weight;

static double base\_weight;

static double price\_for\_service;

};

Lab\_06\_OOP.cpp:

#include "Header\_Lab\_06.h"

double CCar::max\_weight = 2'510;

double CCar::base\_weight = 1'100;

double CCar::crub\_weight\_distance = 288;

double CCar::price\_for\_service = 137.41;

double CTruck::max\_weight = 27'215.5;

double CTruck::base\_weight = 18'000;

double CTruck::crub\_weight\_distance = 800;

double CTruck::price\_for\_service = 659;

double CMotorcycle::max\_weight = 292;

double CMotorcycle::base\_weight = 216;

double CMotorcycle::crub\_weight\_distance = 90;

double CMotorcycle::price\_for\_service = 346;

### pch.h:

#pragma once

#include "gtest/gtest.h"

#include "../Lab\_06\_OOP/Header\_Lab\_06.h"

### test.cpp:

#include "pch.h"

TEST(CVehicleTest, CalculateDistanceWithWeightTest) {

CVehicle v(18000, 2000, 5000, 50, 3500);

double expected\_distance = 17370;

double actual\_distance = v.GetDistanceWithWeight();

EXPECT\_DOUBLE\_EQ(expected\_distance, actual\_distance);

}

TEST(CVehicleTest, CalculatePriceForServiceTest) {

CVehicle v(18000, 2000, 5000, 50, 3500);

double expected\_price = 50;

double actual\_price = v.GetPriceForService();

EXPECT\_DOUBLE\_EQ(expected\_price, actual\_price);

}

TEST(CVehicleTest, CalculateMaxWeightTest) {

CVehicle v(18000, 2000, 5000, 50, 3500);

double expected\_max\_weight = 5000;

double actual\_max\_weight = v.GetMaxWeight();

EXPECT\_DOUBLE\_EQ(expected\_max\_weight, actual\_max\_weight);

}

TEST(CCarTest, PrintNameTest) {

CCar toyota("Toyota Corolla");

testing::internal::CaptureStdout();

toyota.PrintName();

string output = testing::internal::GetCapturedStdout();

EXPECT\_EQ("CCar\n\n", output);

}

TEST(CCarTest, ConstructorTest) {

CCar toyota("Toyota Corolla", 100);

double expected\_distance = 284.83199999999999;

double actual\_distance = toyota.GetDistanceWithWeight();

EXPECT\_DOUBLE\_EQ(expected\_distance, actual\_distance);

double expected\_max\_weight = 2510;

double actual\_max\_weight = toyota.GetMaxWeight();

EXPECT\_DOUBLE\_EQ(expected\_max\_weight, actual\_max\_weight);

}

TEST(CTruckTest, PrintNameTest) {

CTruck volvo("Volvo");

testing::internal::CaptureStdout();

volvo.PrintName();

string output = testing::internal::GetCapturedStdout();

EXPECT\_EQ("CTruck\n\n", output);

}

TEST(CTruckTest, ConstructorTest) {

CTruck volvo("Volvo", 1000);

double expected\_distance = 656;

double actual\_distance = volvo.GetDistanceWithWeight();

EXPECT\_DOUBLE\_EQ(expected\_distance, actual\_distance);

double expected\_max\_weight = 27215.5;

double actual\_max\_weight = volvo.GetMaxWeight();

EXPECT\_DOUBLE\_EQ(expected\_max\_weight, actual\_max\_weight);

}

TEST(CMotorcycleTest, PrintNameTest) {

CMotorcycle kawasaki("Kawasaki H2");

testing::internal::CaptureStdout();

kawasaki.PrintName();

string output = testing::internal::GetCapturedStdout();

EXPECT\_EQ("CMotorcycle\n\n", output);

}

TEST(CMotorcycleTest, ConstructorTest) {

CMotorcycle kawasaki("Kawasaki H2", 100);

double expected\_distance = 89.805599999999998;

double actual\_distance = kawasaki.GetDistanceWithWeight();

EXPECT\_DOUBLE\_EQ(expected\_distance, actual\_distance);

double expected\_max\_weight = 292;

double actual\_max\_weight = kawasaki.GetMaxWeight();

EXPECT\_DOUBLE\_EQ(expected\_max\_weight, actual\_max\_weight);

}

# Результат виконання

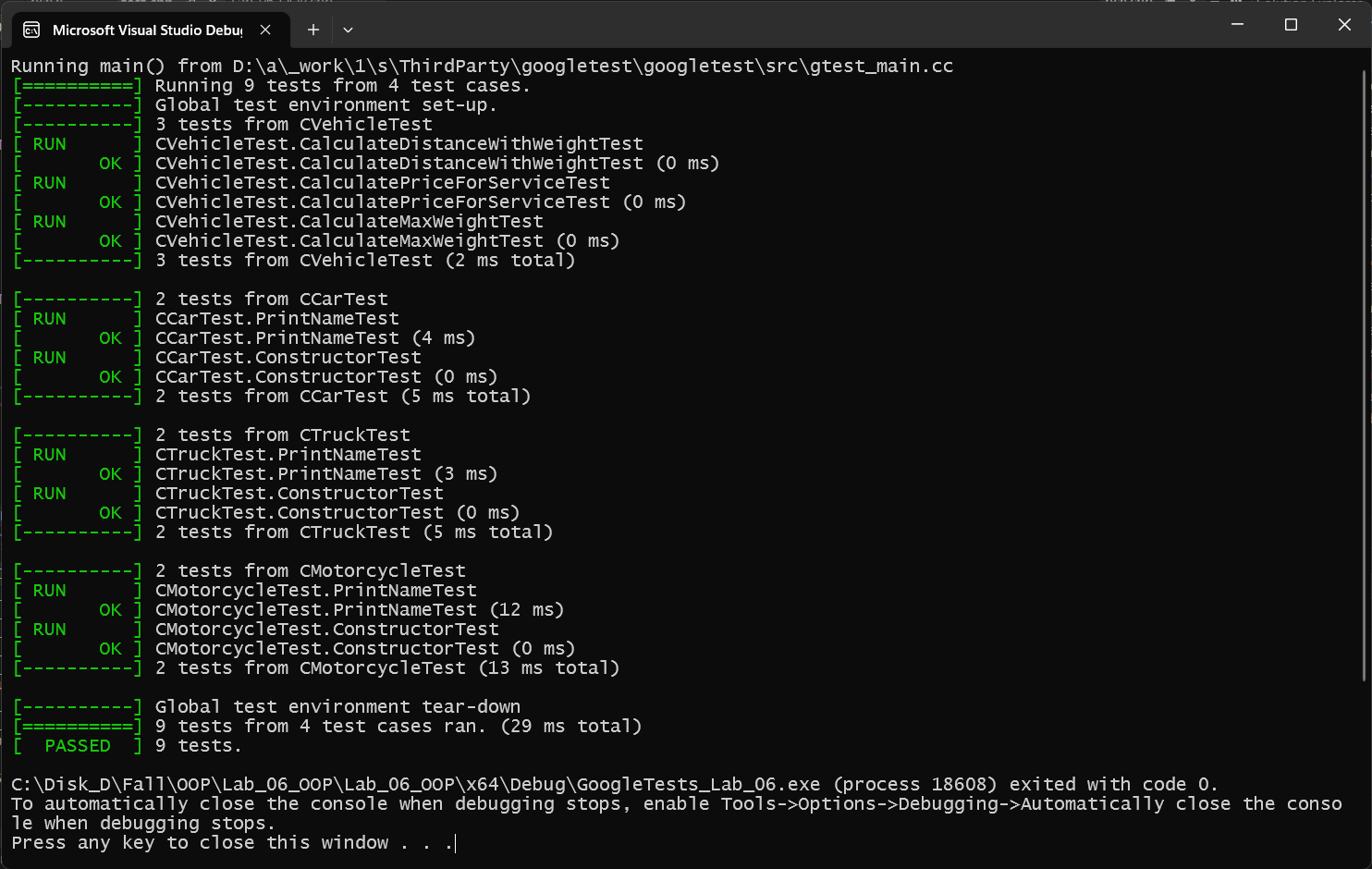


Рис. 1. Виконання тестів для функцій класів CVehicle, CCar, CTruck, CMotorcycle.

# Висновок

Під час виконання лабораторної роботи я навчився створювати базові та похідні класи, використовувати наслідування різного типу доступу, опанував принципи використання множинного наслідування, навчився перевизначати методи в похідному класі, освоїв принципи такого перевизначення.