Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»



Звіт

З лабораторної роботи №6

З дисципліни «Об’єктно-орієнтоване програмування »

На тему:

Спадкування

Виконав: Куцевол В.С. KI-109

Прийняв: асис. каф. ЕОМ

Гузинець Н.В

Львів – 2024

**Тема роботи:** Спадкування

**Мета.** : познайомитися із спадкуванням класів.

**Теоретичні відомості**

Спадкування – це механізм, за допомогою якого один клас може одержувати атрибути та функціональність іншого. Спадкування дозволяє створювати ієрархію класів. При створенні нового класу, що повністю дублює існуючий клас і дещо розширяє його новими властивостями і функціональністю програміст може не дублювати існуючий клас і дописувати в нього розширену функціональність, а вказати, що новий клас є спадкоємцем елементів попередньо визначеного класу і визначити у ньому лише необхідну нову функціональність. В цьому випадку існуючий клас, функціональність якого розширюється у новому класі, називається базовим класом (base class). Новостворений клас називається похідним класом (derived class), або спадкоємцем. Кожен похідний клас може бути використаним у ролі базового класу для майбутніх похідних класів створюючи при цьому дерево спадкування, яке ще називають ієрархією спадкування класів (class hierarchy). Спадкування прийнято відображати у вигляді графу (дерева) у напрямку зверху-вниз. При цьому клас, що є у самому верху є самим першим базовим класом і називається кореневим класом або коренем дерева спадкування класів. Похідний клас, через проміжний, може наслідувати характеристики базового класу. У цьому випадку говорять, що базовий клас є непрямим базовим класом (indirect base class) для похідного. Зокрема, корінь дерева наслідувань є непрямим базовим класом для усіх класів, які знаходяться нижче першого рівня ієрархії. Клас, який При одиночному спадкуванні (single inheritance) клас породжується одним базовим класом. При множинному спадкуванні (multiple inheritance) похідний клас успадковує властивості декількох базових класів, причому можлива ситуація коли один базовий клас буде успадкований кілька разів по кількох гілках. При створенні об'єкта похідного класу в пам'яті зберігаються копії усіх класів, які становлять вітку, що породила даний клас.

За допомогою специфікатора доступу можна визначити, яким чином елементи базового класу будуть успадковуватися похідним класом. При відкритому спадкуванні (використанні специфікатора publіc) у похідному класі члени базового класу мають ті ж специфікатори доступу, що й у базовому класі. При захищеному спадкуванні (використанні специфікатора protected) у похідному класі відкриті члени базового класу стають захищеними, а інші зберігають своє початкове значення специфікатора доступу.

У випадку багатократного спадкування наведений порядок виклику і виконання конструкторів зберігається. Глибина вкладень конструкторів при цьому може бути довільною. Порядок виклику конструкторів є послідовним по мірі ієрархічного спадкування і строго фіксується

Конструктор базового типу не спадкується, проте він викликається компілятором на етапі побудови об'єкта похідного класу. Явний виклик конструктора базового класу не допускається. Можна лише за допомогою специфікатора доступу і списку параметрів базового конструктора керувати вибором конструктора базового класу, який запуститься на виконання при створенні об‘кту похідного класу. При визначенні конструктора можна передавати аргументи конструктору лише верхнього сусіднього класу. Конструкторам класів, які не є сусідніми в ієрархії, аргументи можна передавати лише через визначення конструкторів проміжних класів. Після того як об'єкт побудовано конструктор базового класу стає недоступним.

Деструктори мають обов’язково бути віртуальними, якщо тільки клас не має використовуватися як базовий. У протилежному випадку буде застосовуватися статичне зв‘язування при поліморфізмі і оперуванні похідними класами через вказівники чи посилання на базовий клас. Це призведе до того, що при знищенні об‘єкта похідного класу деструктор похідного класу не буде викликатися, що в свою чергу може призводити до помилок функціонування програми.

Класи, що входять до складу дерева спадкування не є розрізненими, а ієрархічно між собою зв'язані. Тому стосовно них легко здійснювати операцію приведення типів визначену правилом приведення класів при простому спадкуванні: класи, які знаходяться ієрархічно нижче, автоматично приводяться до класів, які знаходяться ієрархічно вище на одній вітці дерева спадкування. Наведене правило справджується для явного та неявного приведення типів. Якщо оголошено вказівник на базовий клас, то його можна розглядати як вказівник на будь-який похідний клас породжений від базового і у цьому випадку помилка приведення типів не буде генеруватись. При цьому об'єкт похідного класу, на який посилається вказівник базового класу, автоматично приводиться до типу похідного класу. Після присвоєння адреси об'єкта похідного класу вказівнику на базовий клас, ним можна скористатись як вказівником на цей об'єкт. При цьому забезпечується доступ лише до членів базового класу і членів класів, що знаходяться у їєрархії вище, ніж клас типу якого є вказівник. Доступ до членів похідного класу з-під такого впазівника чи посилання є заборонений. С++ допускає використання вказівника на базовий клас чи посилання на базовий клас з об'єктом похідного класу, але зворотній напрям застосування вказівників і посилань є забороненим (див. лістинг нижче). Це обмеження можна обійти використовуючи явне приведення типів (див. лістинг нижче). Операція інкрементація вказівника базового класу, який вказує на об'єкт 101 похідного класу, приведе до посилання на інший об'єкт базового, а не похідного класу.

**Хід виконання роботи**

**Варіант 12**

Завдання : Створити абстрактний базовий клас і похійдний від нього клас, які реалізують модель предметної області згідно варіанту. Кожен клас має мати мінімум 3 власні елементи даних один з яких створюється динамічно, методи встановлення і читання характеристик елементів-даних класу (Set і Get), та мінімум 2 абстрактні методи обробки даних і мінімум 2 методи обробки даних у похідному класі. Крім цього клас має містити перевантаження оператора присвоєння, конструкторів по замовчуванню і копіювання та віртуальний деструктор. Для розроблених класів реалізувати програму-драйвер, яка демонструє роботу класів

Абстрактний базовий клас: Водойма

Похідний : Море

// file main.cpp

#include <iostream>

#include "BodyOfWater.h"

#include "Sea.h"

using std::string;

using std::cin;

using std::cout;

using std::endl;

int main() {

BodyOfWater Water = BodyOfWater("Black Sea" , 1000 , 50);

cout << "Base class data: " << endl;

cout << "Water name: " << Water.GetName() << endl

<< "Water volume: " << Water.GetVolume() << endl

<< "Water number of countries: " << Water.GetNumberOfCountries() << endl;

cout << "<----------------------------------------------------------------------------------->" << endl;

Sea RedSea = Sea(12.3 , true , 14 , "Red Sea" , 240 , 50);

cout << "Derrived class data: " << endl;

cout << "Sea name: " << RedSea.GetName() << endl

<< "Sea volume: " << RedSea.GetVolume() << endl

<< "Sea number of countries: " << RedSea.GetNumberOfCountries() << endl

<< "Sea seltnes: " << RedSea.GetIsSelt() << endl

<< "Sea avarage temperature: " << RedSea.GetAvarageTemp() << endl

<< "Sea deepness: " << RedSea.GetDeepness() << endl;

cout << "Virtual methods realization: " << endl;

int newNumberOfCountries, temperatureChange;

cout << "Enter new number of countries: "; cin >> newNumberOfCountries;

RedSea.ChangeNumberOfCountries(newNumberOfCountries);

cout << "Number of countries was changed: " << RedSea.GetNumberOfCountries() << endl;

cout << "Enter temperature change: "; cin >> temperatureChange;

RedSea.AddAvarageTemp(temperatureChange);

cout << "Temperature was changed: " << RedSea.GetAvarageTemp() << endl;

return 0;

}

//File BodyOfWater.h

#include "BodyOfWater.h"

#include <iostream>

#include <string>

using std::string;

using std::cout;

using std::cin;

using std::endl;

string BodyOfWater::GetName() {

return BodyOfWater::name;

}

void BodyOfWater::SetName(string name) {

BodyOfWater::name = name;

}

int BodyOfWater::GetVolume() {

return BodyOfWater::volume;

}

void BodyOfWater::SetVolume(int volume){

BodyOfWater::volume = volume;

}

int BodyOfWater::GetNumberOfCountries() {

return BodyOfWater::numberOfCountries;

}

void BodyOfWater::SetNumberOfCountries(int numberOfCountries) {

BodyOfWater::numberOfCountries = numberOfCountries;

}

BodyOfWater::BodyOfWater(string name, int volume, int nunberOfCountries) {

BodyOfWater::name = name;

BodyOfWater::volume = volume;

BodyOfWater::numberOfCountries = nunberOfCountries;

}

BodyOfWater::BodyOfWater() {

BodyOfWater::name = "Black sea";

BodyOfWater::volume = 1000;

BodyOfWater::numberOfCountries = 50;

}

BodyOfWater::BodyOfWater(const BodyOfWater& obj) {

BodyOfWater::volume = obj.volume;

BodyOfWater::name = obj.name;

BodyOfWater::numberOfCountries = obj.numberOfCountries;

}

//file BodyOfWater.cpp

#pragma once

#include <string>

using std::string;

class BodyOfWater

{

private:

string name;

int volume;

int numberOfCountries;

public:

string GetName();

void SetName(string);

int GetVolume();

void SetVolume(int);

int GetNumberOfCountries();

void SetNumberOfCountries(int);

virtual void ChangeNumberOfCountries(int) {};

virtual void AddVolume() {};

BodyOfWater(string name, int volume, int nunberOfCountries);

BodyOfWater();

BodyOfWater(const BodyOfWater& obj);

};

//file sea.h

#pragma once

#include "BodyOfWater.h"

#include <iostream>

#include <string>

class Sea :

public BodyOfWater

{

private:

float deepness;

bool isSelt;

int avarageTemp;

public:

void SetDeepness(float);

float GetDeepness();

void SetIsSelt(bool);

bool GetIsSelt();

void SetAvarageTemp(int);

int GetAvarageTemp();

void AddAvarageTemp(int number);

bool isSeaSelt();

Sea& operator = (Sea& obj) {

Sea::deepness = obj.deepness;

Sea::isSelt = obj.isSelt;

Sea::avarageTemp = obj.avarageTemp;

return \*this;

}

Sea(float deepness, bool isSelt, int avarageTemp, string name, int volume, int numberOfCountries);

Sea(const Sea& obj);

void ChangeNumberOfCountries(int newNumberOfCountries) {

Sea::SetNumberOfCountries(newNumberOfCountries);

}

void AddVolume(int number) {

Sea::SetVolume(number);

};

};

//file sea.cpp

#include "Sea.h"

#include <iostream>

void Sea::SetDeepness(float deepness) {

Sea::deepness = deepness;

}

float Sea::GetDeepness() {

return Sea::deepness;

}

void Sea::SetIsSelt(bool isSelt) {

Sea::isSelt = isSelt;

}

bool Sea::GetIsSelt() {

return Sea::isSelt;

}

void Sea::SetAvarageTemp(int avarageTemp) {

Sea::avarageTemp = avarageTemp;

}

int Sea::GetAvarageTemp() {

return Sea::avarageTemp;

}

void Sea::AddAvarageTemp(int number) {

Sea::SetAvarageTemp(number + Sea::avarageTemp);

}

bool Sea::isSeaSelt() {

return Sea::isSelt;

}

Sea::Sea(float deepness, bool isSelt, int avarageTemp, string name, int volume, int numberOfCountries) :

BodyOfWater(name, volume, numberOfCountries) {

Sea::deepness = deepness;

Sea::isSelt = isSelt;

Sea::avarageTemp = avarageTemp;

};

Sea::Sea(const Sea& obj) :

BodyOfWater(obj) {

Sea::deepness = obj.deepness;

Sea::isSelt = obj.isSelt;

Sea::avarageTemp = obj.avarageTemp;

};

**Висновок**

У цій лабораторній роботі я вивчав основні концепції спадкування в мові програмування C++. Спадкування є важливою частиною об'єктно-орієнтованого програмування, яке дозволяє створювати ієрархії класів, використовуючи наявний код з батьківських класів, тим самим забезпечуючи перевикористання коду та покращуючи структуру програми.

Я ознайомився з основними поняттями такими, як базовий клас (батьківський клас) та похідний клас (дочірній клас), та іншими ключовими термінами, такими як публічне, приватне та захищене спадкування. Я дослідив різні аспекти реалізації спадкування, такі як конструктори та деструктори, методи, які можуть бути успадковані, та методи перевантаження.

Крім того, я вивчив важливі концепції, такі як поліморфізм та віртуальні функції, які дозволяють більш гнучко використовувати успадкований код.

Загалом, спадкування в мові програмування C++ є потужним інструментом для організації коду та створення більш простої та ефективної програмної структури. Вивчення цих концепцій допоможе мені створювати більш складні програми з більшою швидкістю та ефективністю.

Top of Form