Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра “Електронних обчислювальних машин”



**Звіт з лабораторної роботи №6**

на тему:

“Спадкування ”

**Виконав:**

Ст. гр. КІ-15

Залевський К.С.

**Перевірив:**

Викладач

Козак Н.Б.

Львів – 2020

**Мета роботи:** познайомитися із перевантаженням операторів.

**Теоретичні відомості:**

**Спадкування (ієрархія "іs a")**

***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***.

Похідний клас наслідує характеристики базового через ***специфікатор доступу (acces specifier) : "довкрапка"****.* Нижче наведено синтаксис спадкування базового класу:



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. Нарешті, при закритому спадкуванні (використанні специфікатора prіvate) у похідному класі всі члени базового класу стають закритими.



***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.



*Код програми:*

#include <iostream>

#include <string>

using namespace std;

class Boiler

{

public:

//Constructor's | Destructor's

Boiler()

{

cout << "Constructor of Boiler" << endl;

this->waranty = rand()/12;

BoilTime = new int;

\*BoilTime = rand()/10;

}

Boiler(const Boiler &other)

{

cout << "Constructor of Boiler (copy)" << endl;

this->BoilTime = new int;

this->BoilTime = other.BoilTime;

this->waranty= other.waranty;

this->power= other.power;

}

virtual ~Boiler()

{

cout << "Detructor of Boiler" << endl;

delete BoilTime;

BoilTime = nullptr;

}

//Method's

virtual void TurnOn() = 0;

virtual void TurnOff() = 0;

//Geter's || Setter's

int GetWaranty()

{

return this->waranty;

}

double GetPower()

{

return this->power;

}

int\* GetBoilTime()

{

return this->BoilTime;

}

void SetWaranty(int waranty)

{

this->waranty=waranty;

}

void SetPower(double power)

{

this->power=power;

}

void SetBoilTime(int value)

{

this->BoilTime=&value;

}

protected:

bool State;

int waranty;

double power=1.6;

int\* BoilTime;

};

class Teapot : public Boiler

{

private:

string model;

int capacity;

int\* Cable\_Lenth;

public:

//Constructor's | Destructor's

Teapot (string model,int capacity, int cablelenth)

{

cout << "Constructor of Teapot" << endl;

this->model = model;

this->capacity = capacity;

Cable\_Lenth = new int;

\*Cable\_Lenth = cablelenth;

}

Teapot(const Teapot& other)

{

cout << "Constructor of Teapot (copy)" << endl;

this->model = other.model;

this->capacity = other.capacity;

this->Cable\_Lenth = other.Cable\_Lenth;

}

~Teapot() override

{

cout << "Detructor of Teapot" << endl;

delete this->Cable\_Lenth;

this->Cable\_Lenth = nullptr;

}

//Operator

void operator=(const Teapot& other)

{

this->model = model;

this->capacity = capacity;

}

//Method's

void TurnOn()override

{

cout <<endl<< "TurnOn Teapot" << endl;

this->State = true;

}

void TurnOff()override

{

cout <<endl<< "TurnOff Teapot" << endl;

this->State = false;

}

void StartBoiling(int water\_capacity)

{

if (!this->State)

{

cout << "Error: turn on Teapot!!!" << endl;

return;

}

if (water\_capacity > this->capacity)

{

cout << "Error: water capacity more than Teapot's capacity"<<endl << endl;

return;

}

cout << "Start boiling with "<<water\_capacity<<" litr of water." << endl << "Process..." << endl << "Finish boiling" <<endl<< endl;

}

void CleanTeapot()

{

if (this->State)

{

cout << "Error: turn off Teapot!!!" << endl;

return;

}

cout <<endl<< "Start cleaning" << endl << "Process..." << endl << "Finish cleaning" <<endl<< endl;

}

//Geter's || Setter's

string GetModel()

{

return this->model;

}

double GetCapacity()

{

return this->capacity;

}

int\* GetCableLenth()

{

return Cable\_Lenth;

}

void SetModel(string model)

{

this->model = model;

}

void SetCapacity(int capacity)

{

this->capacity = capacity;

}

void SetCablelenth(int value)

{

this->Cable\_Lenth= &value;

}

};

int main()

{

int lenth;

Teapot teapot("Teapot 2k", 16, 4);

lenth = \*teapot.GetCableLenth();

cout << endl << "========================================" << endl;

teapot.StartBoiling(17);

teapot.TurnOn();

teapot.StartBoiling(17);

teapot.StartBoiling(14);

teapot.CleanTeapot();

teapot.TurnOff();

teapot.CleanTeapot();

cout << endl << "========================================" << endl;

return 0;

}

*Вікно результату:*



*Висновок:* я познайомився з спадкуванням класів, з абстрактними класами, та віртуальними методами класу.