Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра “Електронних обчислювальних машин”



**Звіт з лабораторної роботи №6**

на тему:

“Спадкування ”

**Варіант 13**

**Виконав:**

Ст. гр. КІ-15

Когутич В.М.

**Перевірив:**

Викладач

Козак Н.Б.

Львів – 2020

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

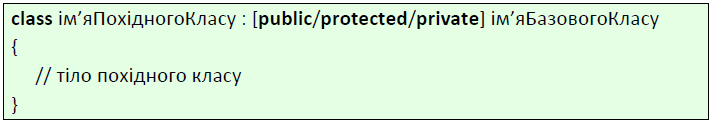
**Теоретичні відомості:**

**Спадкування (ієрархія "іs a")**

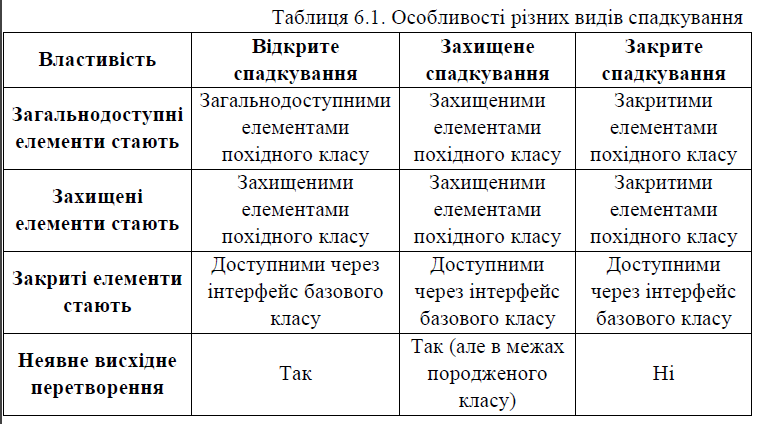
***Спадкування*** – це механізм, за допомогою якого один клас може одержувати атрибути та функціональність іншого. Спадкування дозволяє створювати ієрархію класів.

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:

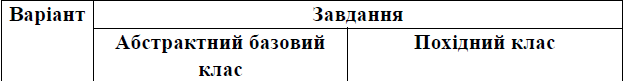


За допомогою специфікатора доступу можна визначити, яким чином елементи базового класу будуть успадковуватися похідним класом. При відкритому спадкуванні (використанні специфікатора publіc) у похідному класі члени базового класу мають ті ж специфікатори доступу, що й у базовому класі. При захищеному 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 cDevice

{

public:

//Constructor's | Destructor's

cDevice()

{

cout << "Constructor of Device" << endl;

this->waranty = rand() / 12;

BatteryCharge = new int;

\*BatteryCharge = rand() / 151-50;

}

cDevice(const cDevice& other)

{

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

this->BatteryCharge = new int;

this->BatteryCharge = other.BatteryCharge;

this->waranty = other.waranty;

this->power = other.power;

}

virtual ~cDevice()

{

cout << "Destructor of Device" << endl;

delete BatteryCharge;

BatteryCharge = 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\* GetBatteryCharge()

{

return this->BatteryCharge;

}

void SetWaranty(int waranty)

{

this->waranty = waranty;

}

void SetPower(double power)

{

this->power = power;

}

void SetGetBatteryCharge(int value)

{

this->BatteryCharge = &value;

}

protected:

bool State;

int waranty;

double power = 1;

int\* BatteryCharge;

};

class Phone : public cDevice

{

private:

string model;

int Innermemory;

int\* DISPLAY;

public:

//Constructor's | Destructor's

Phone(string model, int Innermemory, int mDISPLAY)

{

cout << "Constructor of Phone" << endl;

this->model = model;

this->Innermemory = Innermemory;

DISPLAY = new int;

\*DISPLAY = mDISPLAY;

}

Phone(const Phone& other)

{

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

this->model = other.model;

this->Innermemory = other.Innermemory;

this->DISPLAY = other.DISPLAY;

}

~Phone() override

{

cout << "Destructor of Phone" << endl;

delete this->DISPLAY;

this->DISPLAY = nullptr;

}

//Operator

void operator=(const Phone& other)

{

this->model = model;

this->Innermemory = Innermemory;

}

//Method's

void TurnOn()override

{

cout << endl << "TurnOn Phone" << endl;

this->State = true;

}

void TurnOff()override

{

cout << endl << "TurnOff Phone" << endl;

this->State = false;

}

void takePhoto(int photo)

{

double sizephoto = 0.001;

int mphoto;

mphoto = photo \* sizephoto;

if (!this->State)

{

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

return;

}

if (mphoto > this->Innermemory)

{

cout << "Error: dont enough memory" << endl << endl;

return;

}

cout << "Start making " << photo << " photos." << endl << "Process..." << endl << "Finish!" << endl << endl;

}

void showData() {

cout << endl << "Model of Phone: " << model;

cout << endl << "The amount of battery charge: " << \*BatteryCharge << " %";

cout << endl << "The amount of memory: " << Innermemory << " gigabytes";

cout << endl << "Screen size: " << \*DISPLAY << " inch";

}

void delateData()

{

if (this->State)

{

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

return;

}

cout << endl << "Start delate data" << endl << "Process..." << endl << "Finish delating" << endl << endl;

}

//Geter's || Setter's

string GetModel()

{

return this->model;

}

double GetCapacity()

{

return this->Innermemory;

}

int\* GetDISPLAY()

{

return DISPLAY;

}

void SetModel(string model)

{

this->model = model;

}

void SetCapacity(int Innermemory)

{

this->Innermemory = Innermemory;

}

void SetDISPLAY(int value)

{

this->DISPLAY = &value;

}

};

int main()

{

double displey;

Phone cPhone("Pixel 4Xl ", 64, 5.7);

displey = \*cPhone.GetDISPLAY();

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

cPhone.takePhoto(1500);

cPhone.TurnOn();

cPhone.takePhoto(1700000);

cPhone.takePhoto(1800);

cPhone.showData();

cPhone.TurnOff();

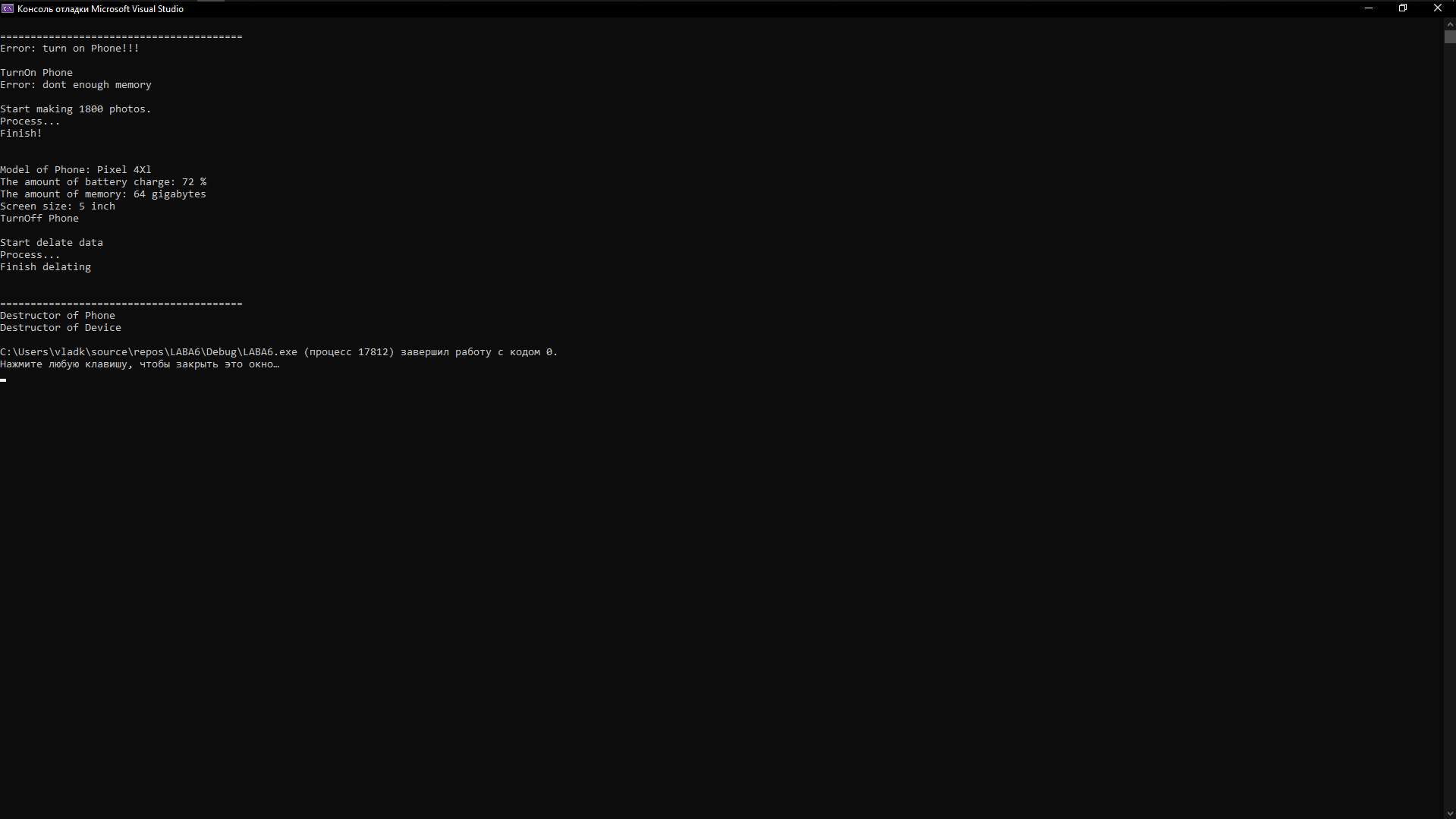
cPhone.delateData();

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

return 0;

}

*Вікно результату:*

**

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