Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра “Електронних обчислювальних машин”



**Звіт з лабораторної роботи №7**

на тему:

“Множинне спадкування. Поліморфізм”

Варіант 13

**Виконав:**

Ст. гр. КІ-15

Когутич В. М.

**Перевірив:**

Викладач

Козак Н.Б.

Львів – 2020

**Мета роботи:** познайомитися із множинним спадкуванням класів та поліморфізмом.

***Завдання:***

Спроектувати і реалізувати ієрархію класів, що описують предметну область згідно варіанту, яка реалізується класом 1. Клас 1 в свою чергу утворюється шляхом множинного спадкування класів 2 і 3 кожен з яких в свою чергу успадковує клас 4. Додаткові вимоги:

1. Базовий клас містить мінімум один віртуальний метод, один невіртуальний метод і одну динамічно створювану властивість.

2. Забезпечити механізми коректної роботи конструкторів і деструкторів.

3. Перевантажити оператор присвоєня з метою його коректної роботи.

4. Кожен з класів має містити мінімум одину властивість і 4 методи.

5. Написати main() функцію де створити об‘єкт класу 1 і продемонструвати різницю між статичним і динамічним поліморфізмом.



******

*Код програми:*

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <time.h>

using namespace std;

class CDevice//Класс 4

{

public:

CDevice()

{

cout << "Constructor CDevice" << endl;

batterycharge = new double;

\*batterycharge = rand() / 100;

state = false;

EnergyUsage = 0.0;

}

virtual ~CDevice()

{

cout << "Destructor CDevice" << endl;

delete this->batterycharge;

this->batterycharge = nullptr;

}

virtual void SmartTurnOn(double w\_batterycharge) = 0;

virtual void TurnOff() = 0;

virtual void TimeNeed(double w\_batterycharge) = 0;

void SetBatterycharge(double Batterycharge)

{

\*batterycharge = Batterycharge;

}

void SetEnergyUsage(double energyusage)

{

this->EnergyUsage = energyusage;

}

double GetEnergyUsage()

{

return this->EnergyUsage;

}

double Timework(double minimal\_batterycharge)

{

if (minimal\_batterycharge > \* this->batterycharge)

{

cout << "Error" << endl;

return 0;

}

return \*this-> batterycharge / EnergyUsage;

}

protected:

double\* batterycharge;

bool state;

double EnergyUsage;

};

class CClock :public CDevice //Класс 2

{

public:

CClock()

{

cout << "Constructor CClock" << endl;

EnergyUsage = 0;

}

CClock(double energyusage) : CClock()

{

cout << "#2 Constructor CClock" << endl;

EnergyUsage = energyusage;

}

CClock(double energyusage, int Usage) : CClock(energyusage)

{

cout << "#3 Constructor CClock" << endl;

this->EnergyUsage = EnergyUsage;

}

~CClock() override

{

cout << "Destructor CClock" << endl;

}

//Method's

const std::string currentDateTime() {

time\_t now = time(0);

struct tm tstruct;

char buf[80];

tstruct = \*localtime(&now);

strftime(buf, sizeof(buf), "%Y-%m-%d.%X", &tstruct);

return buf;

}

void TimeNeed(double w\_batterycharge)override

{

if (this->EnergyUsage == 0)

{

cout << "Error" << endl;

return;

}

cout << "For Clock need time: " << EnergyUsage / w\_batterycharge << ";" << endl;

}

void SmartTurnOn(double w\_batterycharge) override

{

if (this->state)

{

cout << "Error: Clock already on!" << endl;

return;

}

if (this->batterycharge == 0)

{

cout << "Error, insert the battery!" << endl;

return;

}

cout << "Turn on clock!" << endl;

state = true;

}

void TurnOff() override

{

cout << "TurnOff clock";

state = false;

}

void SetEnergyUsage(double energyusage)

{

this->EnergyUsage = energyusage;

}

double GetEnergyUsage()

{

return this->EnergyUsage;

}

protected:

double EnergyUsage;

char buf[80];

};

class CRadio :public CDevice //Класс 3

{

public:

CRadio()

{

cout << "Constructor CRadio" << endl;

volume = 0;

EnergyUsage = 0;

}

CRadio(double Volume) :CRadio()

{

cout << "#2 Constructor CRadio" << endl;

volume = Volume;

}

CRadio(double Volume, int Usage) :CRadio(volume)

{

cout << "#3 Constructor CRadio" << endl;

this->EnergyUsage = Usage;

this->volume = Volume;

}

~CRadio() override

{

cout << "Destructor CRadio" << endl;

}

//Method's

void TimeNeed(double w\_batterycharge)override

{

if (this->EnergyUsage == 0)

{

cout << "Error" << endl;

return;

}

cout << "For Radio need time: " << EnergyUsage / w\_batterycharge << ";" << endl;

}

void SmartTurnOn(double w\_batterycharge) override

{

if (this->state)

{

cout << "Error: Radio already on!" << endl;

return;

}

if (this->batterycharge == 0)

{

cout << "Error, insert the battery! !" << endl;

return;

}

if (\*this->batterycharge < w\_batterycharge)

{

cout << "Error: not enough battery!" << endl;

return;

}

cout << "Turn on Radio!" << endl;

state = true;

}

void TurnOff() override

{

cout << "TurnOff Radio";

state = false;

}

void Setvolume(double Volume)

{

this->volume = Volume;

}

protected:

double volume ;

};

class Cradioclock :public CClock, public CRadio //Класс 1

{

public:

Cradioclock()

{

cout << "Constructor Cradioclock" << endl;

power = 0;

}

Cradioclock(double power) :Cradioclock()

{

cout << "Constructor Cradioclock" << endl;

this->power = power;

}

~Cradioclock() override

{

cout << "Destructor Cradioclock" << endl;

}

Cradioclock operator=(const Cradioclock& other)

{

this->power = other.power;

this->volume = other.volume;

this->::CDevice::EnergyUsage = other.::CDevice::EnergyUsage;

this->::CDevice::batterycharge = other.::CDevice::batterycharge;

}

void TimeNeed(double w\_batterycharge)override

{

if (this->::CDevice::EnergyUsage == 0)

{

cout << "Error" << endl;

return;

}

cout << "For RadioClock need time: " << ::CDevice::EnergyUsage / w\_batterycharge << ";" << endl;

}

void SetPower(double power)

{

this->power = power;

}

double GetPower()

{

return this->power;

}

void SmartTurnOn(double w\_batterycharge)override

{

if (this->::CDevice::state)

{

cout << "Error: RadioClock already on!" << endl;

return;

}

if (this->::CDevice::batterycharge == 0)

{

cout << "Error, insert battery!" << endl;

return;

}

if (\*this->::CDevice::batterycharge < w\_batterycharge)

{

cout << "Error: not enough battery!" << endl;

return;

}

if (this->volume == 0)

{

cout << "Error: no sound!" << endl;

return;

}

cout << "RadioClock start!" << endl;

::CDevice::state = true;

}

void TurnOff() override

{

cout << "RdioClock turn off!" << endl;

::CDevice::state = false;

}

private:

double power;

};

int main()

{

Cradioclock rc;

rc.Setvolume (50);

rc.SetPower(8.0);

//Поліморфізм

CClock cl;

CRadio rd;

//Статичний поліморфізм

cl.SetEnergyUsage(1);

cl.TimeNeed(5.6);

rd.SetEnergyUsage(3);

rd.TimeNeed(5.6);

//Динамічний Поліморфізм

CDevice\* m;

m = &cl;

m->SetEnergyUsage(1.5);

m->TimeNeed(5.6);

m = &rd;

m->SetEnergyUsage(2.5);

m->TimeNeed(5.6);

}

*Вікно результату:*

**

*Висновок:* я познайомився з множинним спадкуванням класів та з поліморфізмом.