Ministerul Educaţiei al Republicii Moldova

Universitatea Tehnică a Moldovei

Catedra Informatică Aplicată

**RAPORT**

Lucrarea de laborator nr.4

Programarea Orientata pe Obiecte

A efectuat:

st. gr. C-171 D. Melniciuc

A verificat:

dr., conf.univ. M. Oșovschi

Chişinău 2018

**Tema:** Moştenirea şi compoziţia

**Scopul lucrării:**

* studierea moştenirii, avantajele şi dezavantajele;
* studierea compoziţiei;
* studierea regulilor de definire a moştenirii şi compoziţiei;
* studierea formelor de moştenire;
* studierea iniţializatorilor;
* principiul de substituţie;
* moştenirea şi compoziţia – ce să alegem.

# Noţiuni de bază

Motivarea

Moştenirea reprezintă unul din cele trei mecanisme de bază a moştenirii orientate pe obiecte. Moştenirea poate fi studiată din două puncte de vedere: dezvoltatorului şi utilizatorului clasei.

Din punct de vedere a dezvoltatorului moştenirea înseamnă, că comportarea şi proprietăţile clasei derivate, sunt nişte dezvoltări a clasei de bază. Din punct de vedere a utilizatorului – moştenirea reprezintă existenţa unor particularităţi de schimbare independenta a claselor cu o interfaţă unică (Inginerii TI şi constructorii de automobile, în primul rând sunt inginerii).

Avantajele moştenirii:

* micşorarea cantităţii codului şi folosirea lui repetată;
* micşorarea cantităţii erorilor şi a preţului de cost;
* o interfaţa unică şi substituţia;
* mărirea vitezei de prelucrare;

Cheltuielile moştenirii: unicele cheltuieli la folosirea moştenirii poate fi numită micşorarea vitezei la compilarea codului. Totuşi preocupările de eficacitate nu trebuie să intre in contrazicere cu avantajele, aşa cum cheltuielile, cu adevărat, nu sunt esenţiale, aşa cum unele din ele pot fi ocolite prin utilizarea funcţiilor de substituire.

Definirea şi utilizarea moştenirii:

Moştenirea se desemnează prin indicarea clasei de bază după numele clasei derivate, după două puncte. Ca de exemplu:

class Animal{

int NofLegs;

public:

void Say(){ cout<<”!!!”; }

};

class Dog: public Animal{ // moştenirea

...

};

void main(){

Dog d;

d.Say();

}

Din exemplu se vede, că funcţia şi variabila, sunt definite în clasa *Animal* sunt importate şi pot fi utilizate de obiectul clasei *Dog.* Starea, ce complică percepţia acestui mecanism, consta din aceia că, clasa derivată poate sa-şi definească comportamentul, primit în clasa de bază. (Struţul şi pinguinul nu zboară, deşi sunt păsări, dar ornitoringul depune ouă, deşi este manifer). Ne vom întoarce la aceasta temă în următoarele lucrări de laborator.

Principiul de substituţiei

Adăugând la cele relatate mai sus, putem spune că, moştenirea presupune, că obiectul clasei derivate poate fi utilizat în locul obiectului de bază:

void main(){

Animal \*ptr = new Dog;

}

***Compoziţia***

Compoziţia reprezintă încă un mecanism, legat de programarea orientată pe obiecte, care reprezintă relaţia dintre obiecte, atunci când moştenirea este relaţia dintre clase.

Moştenirea realizează relaţia *a fi „is a ”. Dog* este *mamifer,* iar *mamiferul – animal.*

Compoziţia realizează relaţia de *a conţine „has a ”. Automobilul* conţine *motor* şi *roţi.*

Definirea compoziţiei

Cu adevărat compoziţia este folosită foarte larg, aşa cum variabilele încorporate conţin un tip şi sunt utilizate la definirea clasei. Dar la folosirea variabilelor încorporate nu apar întrebări, ceia ce nu se poate spune despre clasele utilizatorilor. Definirea nu este grea, însă utilizarea constructorilor creează probleme.

class Car{

Engine e;

};

Iniţializatorii

Cum se ştie variabilele închise a clasei de bază devin inaccesibile, deci, ele nu pot fi iniţiate în constructor, mai mult ca atât, aceasta contrazice principiul repetării codului. Soluţia este una – chemarea constructorului clasei de bază. Aşa şi se întâmplă, numai că pentru constructorii impliciţi şi copiilor sunt generate de compilator. Restul trebuie să fie chemate manual. Aceiaşi problemă apare şi la utilizarea compoziţiei, poate fi utilizat numai constructorul implicit şi copiile. Pentru rezolvarea acestor probleme se utilizează iniţializatorii – care permit chemarea constructorilor şi efectuarea oricărei iniţializări.

class Engine{

int power;

public:

Engine(int p){power=p;}

};

class Transport{

...

public:

Transport(char\*);

};

class Car:public Transport{ // наследование

Engine e; // композиция

public:

Car():Transport(“automobile”),e(10){}

};

**Varianta 4:**

а) De creat o ierarhie a claselor *om* şi *colaborator,* care ocupă un post anumit şi primeşte un salariu anumit.Determinaţi fluxul de ieşire şi fluxul de intrare, constructorul de copiere, operatorii de atribuire prin anumite funcţii a clasei de bază.

***Codul programul I:***

*#include <iostream>*

*#include <stdio.h>*

*#include <string.h>*

*#include <math.h>*

*using namespace std;*

*class om{*

*protected:*

*char name[10];*

*char lastname[10];*

*int dob;*

*public:*

*~om(){};*

*om(){};*

*om(char \*n, char \*ln, int d){*

*strcpy(name, n);*

*strcpy(lastname, ln);*

*dob = d;*

*}*

*void display(){*

*std::cout << "Info angajat" << '\n';*

*cout << "nume: " << name << endl;*

*cout << "prenume: " << lastname << endl;*

*cout << "virsta: " << dob << endl;*

*}*

*void intro(){*

*cout << "\nIntroduceti informatia despre angajat\nNume: ";*

*cin >> name;*

*cout << "Prenume: ";*

*cin >> lastname;*

*cout << "Virsta: ";*

*cin >> dob;*

*}*

*};*

*class date{*

*public:*

*int date2;*

*};*

*class colab:public om,public date{*

*protected:*

*char job[10];*

*int wage;*

*date da;*

*public:*

*~colab(){};*

*colab(){};*

*colab(char \*n1, int w, int d2, char \*n, char \*ln, int d):om(n,ln,d),date(da){*

*strcpy(job, n1);*

*wage = w;*

*da.date2 = d2;*

*}*

*void display2(){*

*display();*

*std::cout << "\nInfo meserie\n";*

*cout << "denumirea meseriei: " << job << endl;*

*cout << "salariu: " << wage;*

*cout << "(lei)\nanul angajarii: " << da.date2 << "\n\n";*

*}*

*void intro2(){*

*intro();*

*cout << "Denumirea serviciului: ";*

*cin >> job;*

*cout << "Salariu: ";*

*cin >> wage;*

*cout << "Anul angajarii: ";*

*cin >> da.date2;*

*}*

*};*

*int main(){*

*cout << "Date prestabilite:\n" << endl;*

*colab obj("Actor", 50000000, 1973, "Jason", "Statham", 42);*

*obj.display2();*

*colab obj3;*

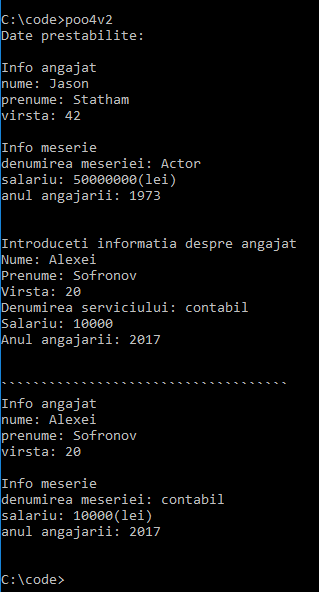
*obj3.intro2();*

*cout << "\n\n````````````````````````````````````\n";*

*obj3.display2();*

*return 0;*

*}*



***ScreenShot:***

***Concluzie:***

*Dupa efectuarea laborotorului am dobintid experienta in limbajul de programere C++ creain clase cu mostenire multipla si compozitie.*