

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**gr. IA-231, Chistol Maxim**

**Raport**

**pentru lucrarea de laborator Nr.4**

***la cursul de “Programarea Orientată pe Obiecte”***

Verificat:

Kulev Mihail dr., conf. univ.

Departamentul Informatică şi IS,

Facultatea FCIM, UTM

**Chișinău 2024**

**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

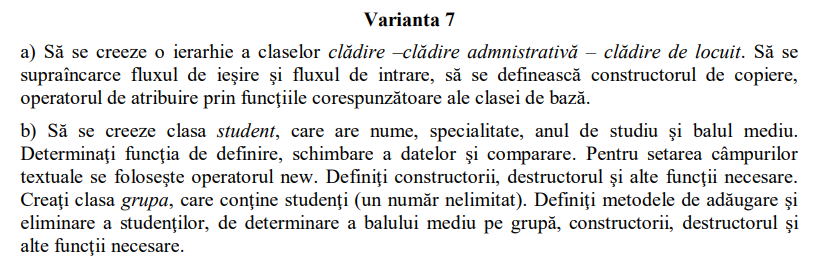
• moştenire şi compoziţie

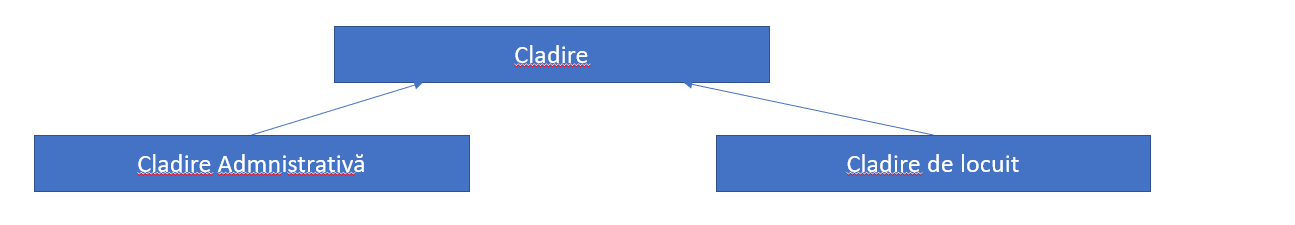
• Studierea formelor de supraîncărcare;

Noţiuni de bază

Derivarea permite definirea într-un mod simplu, eficient şi flexibil a unor clase noi prin adăugarea unor funcţionalităţi claselor deja existente, fără să fie necesară reprogramarea sau recompilarea acestora. Clasele derivate exprimă relaţii ierarhice între conceptele pe care acestea le reprezintă şi asigură o interfaţă comună pentru mai multe clase diferite. Moştenirea poate fi abordată din două puncte de vedere: al elaboratorului şi al utilizatorului clasei.

Din punctul de vedere al elaboratorului, moştenirea abordează comportarea şi proprietăţile clasei derivate ca extensie a proprietăţilor clasei de bază. Din punct de vedere al utilizatorului – moştenirea semnifică existenţa unui şir de clase parţial intershimbabile cu o interfaţă unică. (Inginerii TI şi inginerii mecanici sunt ambii, în primul rînd, ingineri).





**Figura 1:**Schema pentru [a]

|  |  |
| --- | --- |
| **Codul:** | **Output:** |
| **#include <iostream>**  **#include <string>**  **#include <vector>**  **using namespace std;**  **class Cladire {**  **protected:**  **string adresa;**  **int suprafata;**  **public:**  **Cladire(string adresa = "", int suprafata = 0)**  **: adresa(adresa), suprafata(suprafata) {}**  **Cladire(const Cladire& other)**  **: adresa(other.adresa), suprafata(other.suprafata) {}**  **Cladire& operator=(const Cladire& other) {**  **if (this != &other) {**  **adresa = other.adresa;**  **suprafata = other.suprafata;**  **}**  **return \*this;**  **}**  **~Cladire() {}**  **void citire(istream& in) {**  **cout << "Introduceți adresa: ";**  **in >> ws;**  **getline(in, adresa);**  **do {**  **cout << "Introduceți suprafața (pozitivă): ";**  **in >> suprafata;**  **} while (suprafata <= 0);**  **}**  **void afisare(ostream& out) const {**  **out << "Adresa: " << adresa << ", Suprafața: " << suprafata << " mp";**  **}**  **friend istream& operator>>(istream& in, Cladire& cladire) {**  **cladire.citire(in);**  **return in;**  **}**  **friend ostream& operator<<(ostream& out, const Cladire& cladire) {**  **cladire.afisare(out);**  **return out;**  **}**  **};**  **class CladireAdministrativa : public Cladire {**  **protected:**  **int nrBirouri;**  **public:**  **CladireAdministrativa(string adresa = "", int suprafata = 0, int nrBirouri = 0)**  **: Cladire(adresa, suprafata), nrBirouri(nrBirouri) {}**  **CladireAdministrativa(const CladireAdministrativa& other)**  **: Cladire(other), nrBirouri(other.nrBirouri) {}**  **CladireAdministrativa& operator=(const CladireAdministrativa& other) {**  **if (this != &other) {**  **Cladire::operator=(other);**  **nrBirouri = other.nrBirouri;**  **}**  **return \*this;**  **}**  **void citire(istream& in) {**  **Cladire::citire(in);**  **do {**  **cout << "Introduceți numărul de birouri (pozitiv): ";**  **in >> nrBirouri;**  **} while (nrBirouri <= 0);**  **}**  **void afisare(ostream& out) const {**  **Cladire::afisare(out);**  **out << ", Număr de birouri: " << nrBirouri;**  **}**  **};**  **class CladireDeLocuit : public Cladire {**  **protected:**  **int nrApartamente;**  **public:**  **CladireDeLocuit(string adresa = "", int suprafata = 0, int nrApartamente = 0)**  **: Cladire(adresa, suprafata), nrApartamente(nrApartamente) {}**  **CladireDeLocuit(const CladireDeLocuit& other)**  **: Cladire(other), nrApartamente(other.nrApartamente) {}**  **CladireDeLocuit& operator=(const CladireDeLocuit& other) {**  **if (this != &other) {**  **Cladire::operator=(other);**  **nrApartamente = other.nrApartamente;**  **}**  **return \*this;**  **}**  **void citire(istream& in) {**  **Cladire::citire(in);**  **do {**  **cout << "Introduceți numărul de apartamente (pozitiv): ";**  **in >> nrApartamente;**  **} while (nrApartamente <= 0);**  **}**  **void afisare(ostream& out) const {**  **Cladire::afisare(out);**  **out << ", Număr de apartamente: " << nrApartamente;**  **}**  **};**  **int main() {**  **vector<Cladire> cladiri;**  **cladiri.emplace\_back(CladireAdministrativa("Str. Administrativa 1", 500, 10));**  **cladiri.emplace\_back(CladireDeLocuit("Str. Locuintei 2", 300, 8));**  **for (const auto& cladire : cladiri) {**  **cout << cladire << endl;**  **}**  **cout << "\nIntroduceți o clădire administrativă nouă:\n";**  **CladireAdministrativa adminNoua;**  **cin >> adminNoua;**  **cout << "Clădire administrativă nouă: " << adminNoua << endl;**  **return 0;**  **}** |  |

****

**Figura 2:**Schema pentru [b]

|  |  |
| --- | --- |
| **Codul:** | **Output:** |
| **#include <iostream>**  **#include <string>**  **#include <vector>**  **using namespace std;**  **class Student {**  **string\* nume;**  **string\* specialitate;**  **int an;**  **float bal;**  **public:**  **// Constructor implicit**  **Student(const string& nume = "", const string& specialitate = "", int an = 1, float bal = 0.0) {**  **this->nume = new string(nume);**  **this->specialitate = new string(specialitate);**  **this->an = an;**  **this->bal = bal;**  **}**  **// Constructor de copiere**  **Student(const Student& other) {**  **nume = new string(\*other.nume);**  **specialitate = new string(\*other.specialitate);**  **an = other.an;**  **bal = other.bal;**  **}**  **// Operator de atribuire**  **Student& operator=(const Student& other) {**  **if (this != &other) {**  **\*nume = \*other.nume;**  **\*specialitate = \*other.specialitate;**  **an = other.an;**  **bal = other.bal;**  **}**  **return \*this;**  **}**  **// Destructor**  **~Student() {**  **delete nume;**  **delete specialitate;**  **}**  **// Metode pentru schimbarea datelor (setteri)**  **void setNume(const string& numeNou) {**  **\*nume = numeNou;**  **}**  **void setSpecialitate(const string& specialitateNoua) {**  **\*specialitate = specialitateNoua;**  **}**  **void setAn(int anNou) {**  **an = anNou;**  **}**  **void setBal(float balNou) {**  **bal = balNou;**  **}**  **// Metode pentru comparare**  **bool operator<(const Student& other) const {**  **return bal < other.bal;**  **}**  **bool operator==(const Student& other) const {**  **return \*nume == \*other.nume && \*specialitate == \*other.specialitate &&**  **an == other.an && bal == other.bal;**  **}**  **// Metode pentru citire și afișare**  **void citire() {**  **cout << "Introduceți numele: ";**  **cin >> \*nume;**  **cout << "Introduceți specialitatea: ";**  **cin >> \*specialitate;**  **cout << "Introduceți anul: ";**  **cin >> an;**  **cout << "Introduceți balul mediu: ";**  **cin >> bal;**  **}**  **void afisare() const {**  **cout << "Nume: " << \*nume << ", Specialitate: " << \*specialitate**  **<< ", An: " << an << ", Bal mediu: " << bal;**  **}**  **// Metodă pentru a obține balul mediu**  **float getBal() const { return bal; }**  **};**  **class Grupa {**  **vector<Student> studenti;**  **public:**  **// Constructor implicit**  **Grupa() = default;**  **// Constructor cu listă inițială de studenți**  **Grupa(const vector<Student>& studentiInitiali) : studenti(studentiInitiali) {}**  **// Metode pentru adăugare și eliminare**  **void adaugaStudent(const Student& student) {**  **studenti.push\_back(student);**  **}**  **void eliminaStudent(int index) {**  **if (index >= 0 && index < studenti.size()) {**  **studenti.erase(studenti.begin() + index);**  **} else {**  **cout << "Index invalid!" << endl;**  **}**  **}**  **float calculeazaBalMed() const {**  **if (studenti.empty()) return 0.0;**  **float suma = 0.0;**  **for (const auto& student : studenti) {**  **suma += student.getBal();**  **}**  **return suma / studenti.size();**  **}**  **void afiseazaStudenti() const {**  **for (const auto& student : studenti) {**  **student.afisare();**  **cout << endl;**  **}**  **}**  **};**  **int main() {**  **// Creare grupă**  **Grupa grupa;**  **Student s1("Ion", "Informatica", 2, 8.5);**  **Student s2("Ana", "Matematica", 1, 9.0);**  **grupa.adaugaStudent(s1);**  **grupa.adaugaStudent(s2);**  **cout << "Studenți în grupă:" << endl;**  **grupa.afiseazaStudenti();**  **cout << "Balul mediu pe grupă: " << grupa.calculeazaBalMed() << endl;**  **s1.setBal(9.2);**  **grupa.eliminaStudent(0);**  **grupa.adaugaStudent(s1);**  **cout << "Studenți după actualizare:" << endl;**  **grupa.afiseazaStudenti();**  **return 0;**  **}** |  |

# **Concluzii:**

Moștenirea și compoziția sunt două concepte esențiale în programarea orientată pe obiecte, fiecare având avantaje și dezavantaje distincte. Moștenirea facilitează reutilizarea codului și construirea de ierarhii între clase, oferind o interfață comună și posibilitatea utilizării principiului de substituție. Pe de altă parte, compoziția promovează flexibilitatea, permițând construcția de clase prin combinarea comportamentelor și proprietăților altor clase fără a crea relații ierarhice rigide.

Alegerea între moștenire și compoziție depinde de necesitățile specifice ale aplicației. Moștenirea este potrivită atunci când există o relație clară de tipul "este un" între clase, iar compoziția este preferată atunci când relația este de tipul "are un". Supraîncărcarea și inițializatorii contribuie la extinderea funcționalității, oferind metode flexibile de adaptare a claselor.

Studiul acestor concepte subliniază importanța unei înțelegeri aprofundate a regulilor și formelor de implementare, astfel încât soluțiile software să fie eficiente, robuste și ușor de întreținut. Astfel, printr-o utilizare corectă a moștenirii și compoziției, se poate crea un design orientat pe obiecte bine structurat, care să răspundă nevoilor variate ale utilizatorilor și dezvoltatorilor.

# **Bibliografie:**

1. Lucrare de laborator 4: <https://else.fcim.utm.md/pluginfile.php/37341/mod_resource/content/1/lab4Rom.pdf> Accesat pe 25.11.2024
2. POO:<https://www.youtube.com/watch?v=rZcTaRU7AAw&list=PLQOaTSbfxUtBm7DxblJZShqBQnBAVzlXX> Acesat pe 25.11.2024
3. Intelegență artificială: <https://chatgpt.com/c/6749b566-c1e8-8001-b9e4-027fa5295f07Accesat> pe 25.11.2024