**Laborator 3.** Programare bazata pe clase

**Incapsularea folosind clase in limbjele c++, python si java**

**C++**

Incapsulare = mecanismul oferit de limbaj pentru a lega functii de date. Accesorii spre campuri si spre metode se face ca la structuri (in C cu punct). Variabilele de tipul clasa la nastere genereaza apelul de constructor, iar la moarte, destructorul.

In POO o **clasa este un tip de date,** iar **obiectele sunt instante**ale acestui tip de date.

**Incapsularea in C++**nu reprezinta altceva decat proprietatea claselor de obiecte de a grupa sub aceeasi structura datele si metodele aplicabile asupra datelor. In sensul programarii orientata pe obiecte, incapsularea defineste, de asemenea, modalitatea in care diversele obiecte si restul programului se pot referi la datele specifice obiectelor. Casele  din C++ permit separarea datelor in date private si date publice. Programele utilizatorilor pot accesa si utiliza datele unui obiect, declarate private, numai prin utilizarea unor metode definite public. Separarea datelor in sectiuni de date publice si private datermina protejarea datelor fata de utilizarea lor eronata in programe

**JAVA**

Pe scurt, încapsularea este procesul prin care “înfăşurăm” câmpurile unei clase prin nişte metode cu scopul de a avea mai mult control asupra valorilor acestora. În mod normal, acest procedeu este folosit pentru câmpurile declarate private.

exemplu:

**public class Caine{**

**private String nume;**

**private int lungimeCoada;**

**public String getNume(){**

**return nume;**

**}**

**public void setNume(String nume){**

**this.nume = nume;**

**}**

**public void getLungimeCoada(){**

**return lungimeCoada**

**}**

**public int setLungimeCoada(int lungimeCoada){**

**this.lungimeCoada = lungimeCoada;**

**}**

**}**

Observăm că ambele câmpuri din clasa de mai sus sunt declarate folosind modificatorul de acces private, prin urmare, acestea nu pot fi accesate din afara clasei curente.

Cu toate acestea, ne-am dori să avem totuşi o cale de a accesa şi modifica aceste câmpuri şi din exteriorul clasei, fără a modifica, totuşi, modificatorul de acces ales.

Mai mult, să presupunem că ne-am hotărât ca numele câinelui să fie setat doar o singură dată, în constructor, iar apoi să nu mai poată fi modificat. Cu alte cuvinte, vrem ca numele să fie read-only. Soluţia este foarte simplă: ştergem setterul pentru câmpul nume şi adăugăm codul necesar atribuirii în constructor. În felul acesta ne asigurăm că nimeni nu mai poate modifica această variabilă, dar o poate accesa oricând folosind getterul.

Folosind acest concept căpătăm un control sporit asupra datelor din programul nostru. Ba, mai mult, este recomandat şi este considerată o practică foarte bună, ca toate câmpurile să fie private, iar accesul să se facă numai prin intermediul getterilor şi setterilor.

**Python**

In limbajul Python, definirea unei clase se realizeaza prin intermediul cuvantului cheie class, dupa care este precizata denumirea clasei. Corpul clasei cuprinde instructiuni care permit definirea membrilor clasei, atribute de tip data sau metode.

Am avut in vedere descrierea clasei Persoana prin intermediul atributelor de instanta nume si varsta si a atributului de clasa numar\_persoane. Clasa introduce si o metoda constructor, \_\_init\_\_(), necesara in momentul instantierii clasei (permite crearea de obiecte instanta de tip Persoana).

|  |
| --- |
| classPersoana(): |
|  | """Clasa Persoana""" |
|  |  |
|  | numar\_persoane =0; |
|  |  |
|  | def\_\_init\_\_(self, nume, varsta): |
|  | self.\_\_nume = nume |
|  | self.\_\_varsta = varsta |
|  | \_\_class\_\_.numar\_persoane +=1 |
|  |  |
|  | def\_\_str\_\_(self): |
|  | return"{0}, {1}, {2}".format(self.\_\_nume, self.\_\_varsta, \_\_class\_\_.numar\_persoane) |
|  |  |
|  | defse\_prezinta(self): |
|  | print("Buna! Numele meu este {} si am {} ani.".format(self.get\_nume(), self.\_\_varsta)) |
|  |  |
|  | defset\_nume(self, nume): |
|  | self.\_\_nume = nume |
|  |  |
|  | defget\_nume(self): |
|  | return self.\_\_nume |

Incapsularea exprima proprietatea de opacitate a obiectelor cu privire la structura lor interna si modul de implementare a metodelor. Fiecare element al sistemului poate executa actiuni, isi poate modifica starea si poate comunica cu alte elemente fara a dezvalui facilitatile detinute.

La nivelul clasei Persoana putem implementa incapsularea prin prefixarea numelui atributelor de instanta (nume, varsta)cu underscore. Limbajul Python protejeaza membrii declarati cu dublu underscore prin modificarea numelor acestora cu numele clasei.

**2. Definirea unei clase cu 2 constructori, cu un destructor, cu 2 accesori,**

**2 modificatori si inca 1-2 operatii.**

**C++**

#include <string>

#include <vector>

#include <iostream>

namespace ro::usv::rf

{

class Person

{

private:

std::string nume;

std::string preNume;

std::string gen;

public:

Person(const std::string &nume, const std::string &preNume, const std::string &gen);

Person(const std::string &nume, const std::string &preNume);

std::string getNume();

void setNume(const std::string &nume);

std::string getPreNume();

void setPreNume(const std::string &preNume);

std::string getGen();

void setGen(const std::string &gen);

std::string getCompletName();

std::string toString();

static void main(std::vector<std::string> &args);

};

}

#include <string>

#include <vector>

int main(int argc, char \*\*argv)

{

std::vector<std::string> args(argv + 1, argv + argc);

ro::usv::rf::Person::main(args);

}

namespace ro::usv

{

Person::Person(const std::string &nume, const std::string &preNume, const std::string &gen)

{

this->nume = nume;

this->preNume = preNume;

this->gen = gen;

}

Person::Person(const std::string &nume, const std::string &preNume)

{

this->nume = nume;

this->preNume = preNume;

this->gen = "neutru";

}

std::string Person::getNume()

{

return nume;

}

void Person::setNume(const std::string &nume)

{

this->nume = nume;

}

std::string Person::getPreNume()

{

return preNume;

}

void Person::setPreNume(const std::string &preNume)

{

this->preNume = preNume;

}

std::string Person::getGen()

{

return gen;

}

void Person::setGen(const std::string &gen)

{

this->gen = gen;

}

std::string Person::getCompletName()

{

return nume + L" " + preNume;

}

std::string Person::toString()

{

return L"Person [nume=" + nume +", preNume=" + preNume + L", gen=" + gen +"]";

}

void Person::main(std::vector<std::string> &args)

{

Person \*person1 = new Person("Ionescu", "Ion");

Person \*person2 = new Person("Popescu", "Ion", "masculin");

std::wcout << person1 << std::endl;

std::wcout << person2 << std::endl;

person1->setGen("masculin");

person2->setNume("Pop");

std::wcout << person1->getCompletName() << std::endl;

std::wcout << person2->getCompletName() << std::endl;

std::wcout << person1 << std::endl;

std::wcout << person2 << std::endl;

delete person2;

delete person1;

}

}

**JAVA**

package ro.usv.rf;

public class Person {

private String nume;

private String preNume;

private String gen;

public Person(String nume, String preNume, String gen) {

super();

this.nume = nume;

this.preNume = preNume;

this.gen = gen;

}

public Person(String nume, String preNume) {

super();

this.nume = nume;

this.preNume = preNume;

this.gen = "neutru";

}

public String getNume() {

return nume;

}

public void setNume(String nume) {

this.nume = nume;

}

public String getPreNume() {

return preNume;

}

public void setPreNume(String preNume) {

this.preNume = preNume;

}

public String getGen() {

return gen;

}

public void setGen(String gen) {

this.gen = gen;

}

public String getCompletName() {

return nume + " " + preNume;

}

@Override

public String toString() {

return "Person [nume=" + nume + ", preNume=" + preNume + ", gen=" + gen + "]";

}

public static void main(String[] args) {

Person person1 = new Person("Ionescu", "Ion");

Person person2 = new Person("Popescu", "Ion", "masculin");

System.out.println(person1);

System.out.println(person2);

person1.setGen("masculin");

person2.setNume("Pop");

System.out.println(person1.getCompletName());

System.out.println(person2.getCompletName());

System.out.println(person1);

System.out.println(person2);

}

}