POO - Metode, Încapsulare și Specificatori de acces

Metode

Metode

Metodele sunt funcții care aparțin unei clase.

Ca și constructorii, acestea pot fi definite atât în interiorul clasei cât și în afara ei.

Ca și constructori, metodele pot să aibă parametri.

```cpp
#include <iostream>
#include <stdio.h>

using namespace std;


class Teacher {         // The class
  public:               // Access specifier
    void getTeacherName() {  // Method/function defined inside the class
      cout << "Daniel Chis";
    }
};

int main() {
  Teacher oopTeacher;     // Create an object of Teacher
  oopTeacher.getTeacherName() ;  // Call the method
  return 0;
}
```

Exemplu 1 metode

```cpp
#include <iostream>
#include <stdio.h>

using namespace std;


class SquareArea {        // The class
private:
    int area;

  public:              // Access specifier
    void calculateArea(int length, int width); //Method declaration
};

void SquareArea::calculateArea(int length, int width){ //Method definition
    area = length * width;
    cout << area << "\n" ;
}

int main() {
  SquareArea square;     // Create an object of SquareArea
  square.calculateArea(5,40);  // Call the method
  return 0;
}
```

Exemplu 2 metode

Specificatori de acces (Access specifiers)

Specificatori de acces

Specificatorii de acces definesc modul în care membrii unei clase (atribute și metode) pot fi accesați.

În C++ există trei tipuri de specificatori de acces:

- **public**: membrii clasei pot fi accesați din afara ei
- **private**: membrii clasei pot fi accesați doar în interiorul clasei respective
- **protected**: membrii clasei nu pot fi accesați din afara ei decât din interiorul unei clase care o moștenește (mai multe detalii când ajungem la moștenire)

Dacă nu specificăm tipul de specificator, atunci membrul clasei va fi automat de tipul **private**.

```cpp
#include <iostream>
using namespace std;

class Salary {
   public:
      // constructor to set base salary
      Salary(int baseSalary) {
         total = baseSalary;
      }

      // public method to add bonus to the base salary
      void addBonus(int bonus) {
         total = total + total * bonus / 100;
      }

      // public method to get the total salary
      int getTotal() {
         return total;
      };

   private:
      // hidden data where we store the final salary
      int total;
};

int main() {
   Salary salary(1000);

    salary.addBonus(10);
    salary.addBonus(5);

   cout << "Salary with bonuses " << salary.getTotal() <<endl;
   return 0;
}
```

Exemplu 3 - specificatori de access

this Pointer

this Pointer

Toate obiectele din C++ au acces la propria adresă printr-un pointer denumit: **this**. this pointer este un parametru implicit pentru toate funcțiile membre ale unei clase.

Este folosit în interiorul funcțiilor unei clase pentru a referenția obiectul curent.

This este o variabilă de tip pointer utilizată de compilator pentru a ține mereu adresa obiectului curent.

```cpp
#include <iostream>

using namespace std;

class Lab {
    private:
        int number;
    public:
    Lab(int x){
        //this operator
        this->number = x;
    };
    int getLabNumber() {
         return number;
      }
};


int main(void) {
    Lab lab1(1);
    Lab lab2(2);
    cout<<lab1.getLabNumber()<<endl;
    cout<<lab2.getLabNumber()<<endl;
    return 0;
}
```

Exemplu

Static Member

Static Member

Putem defini membrii unei clase și ca **static**. Când declarăm un membru ca **static** acesta va fi unic indiferent câte obiecte instanțiăm la clasa respectivă. Acest atribut este folosit de toate obiectele clasei.

Datele de tipul **static** sunt inițializate cu valoarea zero în momentul în care creăm primul obiect. Nu poate fi definit în interiorul unei clase dar poate fi inițializat în afara ei.

Un membru static poate fi apelat chiar dacă nu există obiecte ale clasei create. Funcțiile statice sunt folosite pentru a accesa membrul static.

Membrii statici nu pot folosi **this->**.

```cpp
#include <iostream>

using namespace std;

class Student {
   public:
      static int studentCount;
         // Increase every time object is created
    Student(){
         studentCount++;
      }
      static int getCount() {
         return studentCount;
      }
};

// Initialize static member of class Student
int Student::studentCount = 20;

int main(void) {
   // Print total number of student before creating a new object
   cout << "Initial number of students " << Student::getCount() << endl;
    Student student1;
    Student student2;
   // Print total number of students after creating other objects
   cout << "Updated number: " << Student::getCount() << endl;

   return 0;
}
```

Exemplu

Încapsulare (Encapsulation)

Încapsularea

Încapsularea este unul dintre pilonii programării orientate pe obiecte. Încapsularea reprezintă procesul de combinare a datelor și a funcțiilor într-o singură structură, ca o capsulă.

Încapsularea este folosită pentru a asigura protecția datelor, date din interiorul unei clase nu pot fi accesate în afara ei. Astfel clasa devine un *black box*.

O bună practică este să declarăm membrii unei clase ca **private**, astfel încât se realizează un control mai bun al datelor, rezultând o securitate mai bună.

Getters & Setters

Procesul de încapsulare al unei clase poate fi împărțit în doi pași:

1. Membri clasei ce reprezintă datele (atributele) vor fi declarați ca **private**
2. Membri clasei ce reprezintă funcțiile care manipulează datele vor fi declarate ca **public**

Astfel, pentru a interacționa cu datele unei clase se folosesc metode de tipul **getters** și **setters**.

Pentru a accesa un atribut **private** se folosesc metodele publice de tipul **get** și **set**.

Exemplu 4

```cpp
#include <iostream>
#include <string.h>

using namespace std;

class Student{ //class name
private: //private members
    string firstName;
    string lastName;
public: //public methods to interact with the members
    //setters:
    void setFirstName(string firstName){
        this->firstName = firstName;
    }
    void setLastName(string lastName){
        this->lastName = lastName;
    }
    //getters:
    string getFirstName(){
        return this->firstName;
    }
    string getLastName(){
        return this->lastName;
    }

};

int main() {
    Student newStudent;
    newStudent.setFirstName("Daniel");
    newStudent.setLastName("Chis");
    cout<<newStudent.getFirstName()<<" "<<newStudent.getLastName()<<endl;
    return 0;
}
```

Summary

De Reținut

3 tipuri de specificatori de acces: public, protected, private

Dacă nu declarăm noi este automat private.

This este o variabilă de tip pointer utilizată de compilator pentru a ține mereu adresa obiectului curent.

Putem defini membrii unei clase și ca **static**. Când declarăm un membru ca **static** acesta va fi unic indiferent câte obiecte instanțiăm la clasa respectivă.

Încapsularea este unul dintre pilonii programării orientate pe obiecte. Încapsularea reprezintă procesul de combinare a datelor și a funcțiilor într-o singură structură, ca o capsulă.

Getters și setters

Exerciții


YOU SHALL NOT
ACCESS MY MEMBERS

Exerciții

1. Să se realizeze o clasă **Money** care are ca atribut **amount**. Atributul amount se setează prin constructor. Valoarea dată la constructor trebuie sa fie una generată random între 0-100. Creați 5 obiecte și afișați numărul de obiecte care au **amount** mai mare de 50. (să se folosească static member doar pentru contor). 4p
2. Să se realizeze o clasă **Student**. Această clasă trebuie să aibă următoarele atribute: nume, prenume, cnp, anul nasterii toate de tipul private, dar și numele facultății și anul înființării acesteia (tot private).

   Creați metodele de get și set pentru **Student**. 1p

   Creați o metodă care să întoarcă sexul studentului în funcție de CNP. 1p

   Creați o metodă care să întoarcă vârsta studentului. 1p

   Creați o metodă care să întoarcă diferența de ani dintre anul înființării facultății și data nașterii studentului. 1. p

   Utilizați this Pointer. 1p