POO - Abstractizare

Abstractizare

Abstractizare - concept

Unul dintre cei 4 piloni ai POO. Abstractizarea e folosită pentru a pune la dispoziție doar elementele esențiale și a le ascunde pe celelalte, ascunde partea de implementare exteriorului.

Noi ca oameni interacționăm foarte mult cu forme abstractizate. Noi știm că avem un puls la inimă ca să funcționăm, dar nu știm cum funcționează inima.

Abstractizarea la bază se realizează folosind specificatorul de access *private*.

Abstractizare - avantaje

Nu trebuie scris cod duplicat.

Scopul principal este de a refolosi codul pentru a realiza o partiționare a codului la nivelul claselor.

Implementarea internă poate fi modificată fără a afecta codul care este folosit pentru a interacționa cu utilizatorul.

Diferențe între încapsulare și abstractizare

Abstractizare	**Încapsulare**
proces de preluare a informației	proces de ascundere a informației
problemele sunt rezolvate la nivel de design	problemele sunt rezolvate la nivel de implementare
prin abstractizare ascundem informația	ascundem date într-o singură entitate și oferim metode de interacțiune cu exteriorul
se realizează prin clase abstracte	prin access modifiers

Abstractizare și Interfețe

Abstractizare și Interfețe

În C++ clasa abstractă și interfața sunt termeni omonimi. O clasă devine abstractă în momentul în care are în interiorul ei cel puțin o funcție virtuală pură (**pure virtual function**).

Exemplu: virtual void function() = 0;

O clasă abstractă nu poate fi instanțiată, deși poate avea constructori.

Dacă o clasă copil nu implementează (nu face overriding) la funcția virtuală pură atunci și aceasta rămâne abstractă.

Exemplu abstractizare

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

class Parent {
   public:

    virtual void hello()  = 0;

    virtual void print() {
        cout << "Parent Class" << endl;
    }
};

class Child : public Parent {
   public:
    void print() {
        cout << "Child Class" << endl;
    }

    void hello(){
        cout << "Not abstract anymore" << endl;
    }
};

int main() {
    Child copil;
    copil.hello();
    copil.print();
    Parent* parinte = &copil;
    parinte->hello();
    parinte->print();
    //Parent parinte2;
    //parinte2.print();
```

Friend Class and Friend Function

Friend Class

O clasă nu poate accesa membrii privați de la o altă clasă, iar dacă nu moștenește o altă clasă nici pe cei protected. Astfel introducem un nou concept *Friend Class*.

Friend Class este o clasă care poate accesa membrii private și protected ai unei clase în care este declarată ca prieten.

```cpp
#include <stdio.h>
#include <string.h>
#include <iostream>
using namespace std;
class Example {
private:
   string name = "Daniel";
   int age = 25;

public:
   friend class Me;
};

class Me {
public:
   void show(Example object){
      cout << object.name << endl;
      cout << object.age << endl;
   }
};

int main() {
   Example object1;
   Me object2;
   object2.show(object1);
   return 0;
}
```

Exemplu friend function

Friend Function

La fel ca Friend Class, Friend function poate accesa membrii private și protected ai clasei.

```
#include <string.h>
#include <iostream>
using namespace std;
class Example {
private:
   string name = "Daniel";
   int age = 25;

public:
    friend class Me;
    friend void hello(Example object);
};

class Me {
public:
   void show(Example object){
      cout << object.name << endl;
      cout << object.age << endl;
   }
};

    void hello(Example object){
        cout << object.age << endl;
        cout << object.name << endl;
    }

int main() {
   Example object1;
   Me object2;
   object2.show(object1);
   hello(object1);
   return 0;
}
```

Summary

De Reținut

Unul dintre cei 4 piloni ai POO. Abstractizarea ascunde implementarea și oferă un șablon pentru noi clase.

Clasele abstracte și interfețele sunt același lucru în C++, în Java sunt lucruri diferite.

O clasă este abstractă când ai o metodă pure virtual function.

Friend class - clasă care poate să apeleze atribute private și protected.

Friend function - funcție globală care poate să apeleze atribute private și protected.

Exerciții

Exerciții

1. **4.5p** Creați o clasă abstractă Country care să aibă 3 metode diferite. Realizați trei clase care să moștenească clasa Country și să implementeze cele 3 metode pure virtual function.

2. **4.5p** Realizați un exemplu legat de cărți în care să implementați:

- Friend class
- Friend function
- Virtual function (see lab 5)