POO - Polimorfism

Polimorfism

Polimorfism - concept

Unul dintre cei 4 piloni ai POO. Polimorfism înseamnă mai multe forme. Polimorfismul poate fi asemuit cu abilitatea unui mesaj de a fi afișat în mai multe forme.

În viața reală un om este poliform, la muncă se comportă într-un fel, acasă în alt fel și când iese în oraș cu prietenii în alt fel. Astfel, omul e polimorf în funcție de situație.

Polimorfismul poate fi de două tipuri:

- La compile time - Supraîncărcare (Overloading)
- La run time - Suprascriere (Overriding)

Overloading

Overloading - Metode

Supraîncărcarea reprezintă posibilitatea de a avea în cadrul unei clase mai multe metode denumite la fel, dar care diferă prin implementarea lor. Acestea se disting prin semnătura lor, și anume:

- Numele lor (nu reprezintă overloading)
- Numărul și tipul parametrilor

Supraîncărcarea are loc la compilare, ea mai purtând numele și de polimorfism static. Compilatorul decide ce metodă este apelată în funcție de tipul referinței, a numelui și a parametrilor. Astfel la runtime deja se știe ce metodă se execută.

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

using namespace std;
class Greeting
{
    public:

    // default method
    void hello()
    {
        cout << "Buna!" << endl;
    }

    // method with 1 parameter
    void hello(int x)
    {
        cout << "Ziua buna voua " << x << " oameni" << endl;
    }

    //method with 1 strig parameter
    void hello(string x)
    {
        cout << "Buna ziua " << x << endl;
    }

    //method with 2 parameters
    void hello(int x, string y)
    {
        cout << "Celor " << x << " va transmit " << y << endl;
    }
};
```

```cpp
int main() {

    Greeting salut;
    salut.hello();
    salut.hello(10);
    salut.hello("tuturor");
    salut.hello(30, "buna dimineata");
    return 0;
}
```

Exemplu function overloading

Reguli overloading

1. Metode care diferă doar în return type (int function() vs char function()) - crapă la compilare
2. Dacă 2 metode au același parametru din care unul e pointer nu o să meargă
3. Dacă 2 metode au aceeași parametrii iar la una sunt setați default nu o să meargă
4. Dacă 2 metode au același parametru iar unul e de tipul const nu o sa meargă

Overloading - Operatori

În C++ putem face overloading și la operatori. Aceștia vor fi specifici la nivel de clasă.

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

class Family {
   private:
    int members;

   public:

    Family(){
        members = 2;
    }

    void operator +() {
        this->members = this->members  + 1;
    }

    void operator -() {
        this->members = this->members - 1;
    }

    void operator ++(){
        this->members = this->members +2;
    }

    void getMembers() {
        cout << "This family has " << this->members << " members" << endl;
    }
};
```

```cpp
int main() {
    Family familie;
    familie.getMembers();
    +familie;
    ++familie;
    familie.getMembers();
    -familie;
    familie.getMembers();
    return 0;
}
```

Exemplu overloading operators

Reguli overloading operatori

Operatori care pot fi supraîncărcați

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

Operatori care nu pot fi supraîncărcați

::	.*	.	?:

Overriding

Overriding - Metode

Suprascrierea reprezintă redefinirea metodelor din clasa părinte în clasa copil. Metodele din clasa părinte nu se modifică, iar acestea pot fi modificate doar dacă sunt public sau protected.

Clasa copil poate să suprascrie o metodă doar dacă are același return type și aceeași semnătură.

Suprascrierea se determină la runtime, în mod dinamic. Decizia în ceea ce privește metoda care se apelează se face pe baza obiectului. Overriding se mai numește și polimorfism dinamic (runtime polymorphism).

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

using namespace std;
class Animal {
  public:
    void sound()
    {
    cout << "Every animal has a specific sound\n" ;
    }
};

class Cat : public Animal {
  public:
    void sound()
    {
    cout << "Meow\n" ;
    }
};

class Dog : public Animal {
  public:
    void sound()
    {
    cout << "Woof\n" ;
    }
};
```

```cpp
int main() {
    Animal necunoscut;
    Cat pisica;
    Dog caine;

    necunoscut.sound();
    pisica.sound();
    caine.sound();

    pisica.Animal::sound();
    caine.Animal::sound();

    return 0;
}
```

Exemplu function overriding

Funcții virtuale

O funcție virtuală reprezintă o metodă care este declarată în clasa de bază folosind **virtual** și este suprascrisă în clasa copil.

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

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

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

int main() {
     Child copil;
     Parent* parinte = &copil;
     parinte->print();
     Parent parinte2;
     parinte2.print();
     return 0;
}
```

Summary

De Reținut

Unul dintre cei 4 piloni ai POO. Polimorfism înseamnă mai multe forme.

Polimorfismul poate fi de două tipuri: la compile time - Supraîncărcare (Overloading) și la run time - Suprascriere (Overriding)

Overloading: reprezintă posibilitatea de a avea în cadrul unei clase mai multe metode denumite la fel, dar care diferă prin implementarea lor. Supraîncărcarea are loc la compilare, ea mai purtând numele și de polimorfism static.

În c++ putem face overloading și la operatori.

Overriding: reprezintă redefinirea metodelor din clasa părinte în clasa copil. Overriding se mai numește și polimorfism dinamic (runtime polymorphism).

O funcție virtuală reprezintă o metodă care este declarată în clasa de bază folosind **virtual** și este suprascrisă în clasa copil.

Exerciții

Exerciții

1. **3p** Creați o clasă “Volume” care să aibă o metodă supraîncărcată pentru a calcula și afișa volumul pentru cub, sferă și piramidă, plus o metodă care să aibă același nume și să afișeze un mesaj.

2. **3p** Modelați o companie de angajați la care să aveți o moștenire multinivel pentru angajați (minim 3 clase, A- > B -> C). Fiecare clasă va avea 2 metode care pleacă din clasa de bază și sunt suprascrise în clasele copii.

3. **3p** Creați o clasă care să aibă operatorul “+” overloaded și să fie folosit pentru concatenarea a două string-uri private ce aparțin de clasa respectivă.