Laborator 5: Functii Virtuale. Controlul Tipului la Rulare. Clase Abstracte. Interfete

Functii Virtuale. Controlul Tipului la Rulare

In cadrul laboratorului de astazi vom discuta despre conceptul de polimorfism de rulare si cum este acesta realizabil in contextul limbajului C++.

Pana acum am discutat despre polimofismul de compilare ce este posibil prin intermediul conceptului de *supraincarcare.*Acest lucru permite folosirea aceleasi functionalitati(interfete) oferite in cadrul codului vostru dar cu seturi de parametri diferiti.

In cadrul paradigmei de programare orientate obiect, este posibil si un alt tip de polimorfism, polimorfismul de rulare. Si anume, schimbarea dinamica(in timpul rularii) a functionalitatilor definite in urma procesului de derivare.

Exemplu:

```
#include <iostream>
#include <math.h>
using namespace std;

class MyBaseClass
{
	int a, b;
public :
	MyBaseClass()
	{
		a = 0;
		b = 0;
	}
	void set(int value1, int value2)
	{
		a = value1;
		b = value2;
	}
	void Print()
	{
		cout<<"Esti in base class "<<a<<" "<<b<<endl;
	}
};

class MyDerivedClass : public MyBaseClass
{
	int c;
public :
	MyDerivedClass()
	{
		c = 0;
```

```
	}
	void setC(int value)
	{
		c = value;
	}
	void Print()
	{
		cout<<"Esti in derived class "<<c<<endl;
	}
};

int main()
{
	MyBaseClass *p, a;
	a.set(2, 3);
	MyDerivedClass b;
	b.setC(4);
	p = &a;
	p->Print();//se acceseaza metoda Print a clasei de baza
	p = &b;
	p->Print();//se acceseaza metoda Print a clasei de baza
	system("PAUSE");
	return 0;
}
```

Dupa cum se poate observa din exemplul de mai sus, avem definite in ambele clase, functia Print in interiorul careia se afiseaza valorile variabilelor membru ale clasei si faptul ca esti ori in clasa de baza ori in clasa derivata. In programul principal avem definit un pointer catre clasa de baza prin intermediul caruia pointam pe rand la fiecare instanta a claselor noastre, clasa derivata respectiv clasa de baza. Deci, implicit schimbam functionalitatea pointerului p in functie de clasa catre care dorim sa pointam. Dar, in situatia de fata, cand dorim sa pointam catre obiectul derivat si sa ne folosim de functia Print definita in cadrul clasei derivate, avem surpriza de a obtine tot functionalitatea definita in cadrul clasei de baza.

Pentru a ne putea folosi de metoda Print definita in cadrul clasei derivata, trebuie sa adaugam cuvantul ***virtual*** in fata declaratiei functiei Print din clasa de baza. Astfel functia Print a devenit ***functie virtuala***. Prin intermediul acestui mecanism, este posibil procesul de *suprascriere(override)* al metodei Print in cadrul eventualelor clase derivate. Atributul **virtual** este valabil numai in situatia in care obiectele clasei de baza respectiv derivate, sunt accesate via o structura de tip pointer. Astfel, se respecta conceptual de polimorfism, *“o singura interfata, functionalitati multiple”.* In aceasta situatie, interfata este reprezentata de metoda cu acelasi nume implementata in cadrul ierarhiei de clase. Functionalitatea multipla este realizata prin faptul ca in fiecare clasa din ierarhia de derivare poate avea alta implementare. In orice alta situatie, ele se comporta normal ca orice alta functie. Acest gen de polimorfism este denumit polimorfism de rulare, deoarece functionalitatea se stabileste in timpul rularii, in functie de variabila catre care indica pointerul.

!! Pentru a putea schimba functionalitatea metodei dorite in timpul executiei, trebuie ca sa aveti neaparat un pointer catre clasa de baza din cadrul ierarhiei de derivare folosite.

Daca o functie este declarata ca fiind virtuala, atunci aceast atribut este mostenit indiferent de cate derivari au loc.

Exemplu:

```
#include <iostream>
#include <math.h>
using namespace std;

class MyBaseClass
{
	int a, b;
public :
	MyBaseClass()
	{
		a = 0;
		b = 0;
	}
	void set(int value1, int value2)
	{
		a = value1;
		b = value2;
	}
	virtual void Print()
	{
		cout<<"Esti in base class "<<a<<" "<<b<<endl;
	}
};

class MyDerivedClass : public MyBaseClass
{
	int c;
public :
	MyDerivedClass()
	{
		c = 0;
	}
	void setC(int value)
	{
		c = value;
	}
	void Print()
	{
		cout<<"Esti in derived class "<<c<<endl;
	}
};

int main()
{
	MyBaseClass *p, a;
	a.set(2, 3);
	MyDerivedClass b;
	b.setC(4);
	p = &a;
	p->Print();//se acceseaza metoda Print a clasei de baza
	p = &b;
```

```
	p->Print();//se acceseaza metoda Print a clasei derivate
	system("PAUSE");
	return 0;
}
```

!! A nu se confunda conceptul de *supraincarcare(overload)* cu conceptual de *suprascriere(override)*. Pentru a putea beneficia de atributul **virtual** al unei functii, trebuie sa aveti aceeasi signatura pentru functia ce o doriti suprascrisa(acelasi nume, acelasi tip returnat , acelasi numar si tip de parametri).

Pentru a putea schimba controlul tipul la rulare, se pot folosi de asemeni si referinte.

Exemplu:

```
#include <iostream>
#include <math.h>
using namespace std;

class MyBaseClass
{
	int a, b;
public :
	MyBaseClass()
	{
		a = 0;
		b = 0;
	}
	void set(int value1, int value2)
	{
		a = value1;
		b = value2;
	}
	virtual void Print()
	{
		cout<<"Esti in base class "<<a<<" "<<b<<endl;
	}
};

class MyFirstDerivedClass : public MyBaseClass
{
	int c;
public :
	MyFirstDerivedClass()
	{
		c = 0;
	}
	void setC(int value)
	{
		c = value;
	}
	void Print()
	{
		cout<<"Esti in first derived class "<<c<<endl;
	}
};
```

```
class MySecondDerivedClass : public MyFirstDerivedClass
{
	int d;
public :
	MySecondDerivedClass()
	{
		d = 0;
	}
	void setD(int value)
	{
		d = value;
	}
	void Print()
	{
		cout<<"Esti in second derived class "<<d<<endl;
	}
};

void functionalitate(MyBaseClass &obj)
{
	obj.Print();
}

int main()
{
	MyBaseClass *p, a;
	a.set(2, 3);
	MyFirstDerivedClass b;
	MySecondDerivedClass c;
	b.setC(4);
	c.setD(1);
	functionalitate(a);
	functionalitate(b);
	functionalitate(c);
	system("PAUSE");
	return 0;
}
```

Clase Abstracte

Exista posibilitatea de a declara o functie in interiorul clasei de baza, urmand ca aceasta sa fie definite in interiorul claselor derivate. Acest lucru este posibil prin intermediul functiilor virtuale pure.

In cadrul clasei de baza, se scrie doar antetul functiei virtuale pure, urmand ca aceasta sa fie implementata in clasele derivate, ce mostenesc clasa de baza respectiva.

Clasele care implementeaza functii virtuale pure se numesc clase abstracte si nu pot fi instantiate obiecte din cadrul acestora. Totusi se pot declara pointeri catre obiecte din cadrul acestei clase.

Exemplu:

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

class MyBaseClass
{
	int a;
public :
	MyBaseClass()
	{
		a = 0;
	}
	void set(int value)
	{
		a = value;
	}
	virtual void ShowString()=0;

};

class MyDerivedClass : public MyBaseClass
{
	string s;
public :
	MyDerivedClass()
	{
		s = "";
	}
	void setS(string value)
	{
		s = value;
	}
	void ShowString()
	{
		cout<<s<<endl;
	}
};



int main()
{
	MyDerivedClass b;
	b.setS("laborator");
	b.ShowString();
	system("PAUSE");
	return 0;
}
```

Interfete

Clasele abstracte ce contin doar metode virtuale pure se numesc interfete. Sunt folosite pentru a putea genera o masca de implementare pentru un anumit set de functionalitati.

Exemplu:

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

class MyInterface
{
public :

	virtual void computeSum()=0;
	virtual void ShowString()=0;

};

class MyClass : public MyInterface
{
	int a, b;
	string s;
public :
	MyClass()
	{
		a = 0;
		b = 0;
		s = "";
	}
	void setS(string value)
	{
		s = value;
	}
	void setInt(int v1, int v2)
	{
		a = v1;
		b = v2;
	}
	void ShowString()
	{
		cout<<s<<endl;
	}
	void computeSum()
	{
		cout<<a+b<<endl;
	}
};


int main()
{
	MyClass b;
```

```
        b.setS("laborator");
        b.setInt(19, 29);
        b.ShowString();
        b.computeSum();
        system("PAUSE");
        return 0;
}
```

Exercitii

1. Definiti clasa Persoana, ce are un camp Nume de tipul string. Adaugati metoda Show, de tip void ce nu primeste parametri in cadrul careia se afiseaza numele persoanei.
   Definiti clasa Adult, ce mosteneste clasa Persoana si are un camp Salariu de tipul int. Adaugati metoda Show, de tip void ce nu primeste parametri in cadrul careia se afiseaza numele si salariul adultului respectiv.
   Definiti clasa Copil, ce mosteneste clasa Persoana si are un camp Alocatie de tipul int. Adaugati metoda Show, de tip void ce nu primeste parametri si in cadrul careia se afiseaza numele si alocatia copilului respectiv.
   In cadrul functiei main, definiti 3 obiecte din fiecare clasa, atribuiti valori campurilor fiecarei clase. Definiti un pointer catre clasa Persoana si pointati pe rand catre fiecare din cele 3 obiecte si apelati metoda Show a fiecarei clase via pointerul respectiv, afisand caracteristicile fiecarui obiect(numele persoanei, numele si salariul adultului, numele si alocatia copilului).

2. Scrieti definitia clasei PunctGeometric, ce contine 2 variabile membru private de tip int, ce reprezinta coordonatele punctului geometric in plan si contine constructor de initializare si metode de acces asupra variabilelor respective. De asemeni, declarati o metoda virtuala pura de calculare a ariei.
   Scrieti definitia a 2 clase, Cerc si Patrat, ce mostenesc clasa PunctGeometric. Clasa Cerc, contine o variabila membru private de tip int denumita raza. Folosindu-va de metoda virtuala pura a clasei de baza, calculati aria cercului.
   Clasa patrat, contine o variabila membru private de tip int denumita distanta, ce reprezinta distanta de la centrul patratului la oricare dintre laturi. Calculati aria patratului prin intermediul metodei virtuale pure.
   In functia main, instantiati 2 obiecte de tip Patrat, respectiv Cerc, atribuitile caracteristicile necesare(centru, distanta respectiv raza) si afisati aria acestora.
3. Definiti o interfata pentru a putea fi utilizata in lucrul cu cozi de int. Permiteti folosirea urmatoarelor tipuri de operatii asupra structurii de tip coada :
   - Front() – returneaza elementul din capul cozii
   - Back() – returneaza elementul din spatele cozii
   - Push_Back() – introduce un element prin spatele cozii
   - Pop_Front() – scoate un element din capul cozii

   Scrieti o implementare pentru coada respective in cadrul unei clase, si realizati o instantiere a clasei respective in functia main.