Laborator POO

Principiile Programarii Orientate Obiect

28.02.2012

- Bibliografie laborator :
  - C++, The Complete Reference, *Herbert Schildt*
  - Thinking in C++, *Bruce Eckel*
  - http://stackoverflow.com
- Paradigme de programare folosite :
  - Paradigma de programare functionala
  - ***Paradigma de programare orientata obiect***
  - Paradigma de programare declarativa
- Limbajul de programare C++ pune la dispozitie implementarea conceptului de programare orientate obiect.
- Paradigma de programare se poate descrie prin 3 concepte de baza :
  - *INCAPSULAREA* reprezinta mecanismul prin care date si modalitati de procesare a acestora sunt puse la un loc in **clase**. Prin aceasta modalitate se permita o mai buna evidenta si structurare a datelor si de asemeni o buna modularizare si gestiune a eventualelor erori ce pot aparea in cadrul programului.
  - *MOSTENIREA* reprezinta mecanismul prin care se pot defini noi clase pe baza proprietatilor unor clase deja existente.
  - *POLIMORFISMUL* reprezinta mecanismul prin intermediul caruia printr-o singura interfata poti manipula o intreaga categorie de actiuni din cadrul unei clase sau a unei ierarhii de clase.

< EXEMPLU 0 >

```
#include <string>
#include <iostream>

using namespace std;

class MyClass
{
public:

	string MyString;

};

int main()
```

```
{
	MyClass MyObject;
	MyObject.MyString = "Hello World!";
	cout<<MyObject.MyString<<endl;
	system("PAUSE");
	return EXIT_SUCCESS;
   }
```

< EXEMPLU 1 >

```
#include <iostream>

using namespace std;

class complex
{
public:
	double re;
	double im;

	void read()
	{
		cout<<"Introduceti partea reala:"<<endl;
		cin>>re;
		cout<<"Introduceti partea imaginara:"<<endl;
		cin>>im;
	}
	void write();
};
void complex::write()
{
	cout<<re<<"+i*"<<im<<endl ;
};


int main()
{
	complex a;
	a.read();
	a.write();


	system("PAUSE");
	return EXIT_SUCCESS;
   }
```

< EXEMPLU 2 >

```
#include <iostream>

using namespace std;

class complex
{
	double re;
```

```
      double im;
public:
      void  setre(double x){re=x;};
      void  setim(double y) {im=y;};
      double getre(){return re;};
      double getim(){return im;};
};
void read(complex& a)
{double x,y;
cout<<"Introduceti partea reala: ";
cin>>x;
a.setre(x);
cout<<"Introduceti partea imaginara: ";
cin>>y;
a.setim(y);}
void write(complex& a)
{ cout<<a.getre()<<"+i*"<<a.getim()<<endl;}

int main()
{
      complex a;
      read(a);
      //  cout<<a.re<<endl; //--- atributul re nu mai este public
      write(a);
      cout<<a.getre()<<endl;
      system("PAUSE");
      return EXIT_SUCCESS;
   }
```

< EXEMPLU 3 >

```
#include <iostream>

using namespace std;

class complex
{
      double re;
      double im;
public:
      void  setre(double x){re=x;};
      void  setim(double y) {im=y;};
      double getre(){return re;};
      double getim(){return im;};
};
void read(complex& a)
{double x,y;
cout<<"Introduceti partea reala: ";
cin>>x;
a.setre(x);
cout<<"Introduceti partea imaginara: ";
cin>>y;
a.setim(y);}
void write(complex& a)
```

```
{ cout<<a.getre()<<"+i*"<<a.getim()<<endl;}
complex aduna(complex a, double b)
{ complex c;
c.setim(a.getim());
c.setre(a.getre()+b);
return c;
}
complex aduna(double a, complex b)
{ complex c;
c.setim(b.getim());
c.setre(b.getre()+a);
return c;
}
complex aduna(complex a, complex b)
{ complex c;
///to be filled
return c;
}

int main()
{
      complex a,c;
      double x=3.5;
      read(a);
      //  cout<<a.re<<endl; //--- atributul re nu mai este public
      write(a);
      cout<<a.getre()<<endl;
      cout<<"Rezultatele adunarii:"<<endl;
      c=aduna(a,x);
      write(c);
      c=aduna(x,a);
      write(c);
      system("PAUSE");
      return EXIT_SUCCESS;
   }
```