POO: Încapsularea

~al doilea pilon al programării orientate pe obiecte~

Încapsularea este un mecanism care leagă împreună cod şi date şi le păstrează pe ambele în siguranţă faţă de intervenţii din afară şi de utilizări greşite. Mai mult, încapsularea este cea care permite crearea unui obiect. Spus simplu, un obiect este o entitate logică ce încapsulează atât date cât şi cod care manevrează aceste date. Într-un obiect o parte din cod şi/sau date pot fi particulare acelui obiect şi inaccesibile în afara sa. În acest fel, un obiect dispune de un nivel semnificativ de protecţie care împiedică modificarea accidentală sau utilizarea incorectă a părţilor proprii obiectului de către secţiuni ale programului cu care nu are legătură.

În cele din urmă, un obiect este o variabilă de un tip definit de utilizator. Când se defineşte un obiect, implicit se crează un nou tip de date.

Avantaje:

- poti face clasa read-only sau write-only (getteri si setteri)
- control asupra datelor (starea obiectului)

În C++ încapsularea este îndeplinită prin două aspecte:

1. folosirea claselor pentru unirea structurile de date şi a funcţiilor destinate manipulării lor;
2. folosirea secţiunilor private şi publice, care fac posibilă separarea mecanismului intern de interfaţa clasei;

O clasă reprezintă un tip de date definit de utilizator, care se comportă întocmai ca un tip predefinit de date. Pe lângă variabilele folosite pentru descrierea datelor, se descriu şi metodele (funcţiile) folosite pentru manipularea lor.

Instanţa unei clase reprezintă un obiect - este o variabilă declarată ca fiind de tipul clasei definite. Variabilele declarate în cadrul unei clase se numesc variabile membru, iar funcţiile declarate în cadrul unei clase se numesc metode sau funcţii membru. Metodele pot accesa toate variabilele declarate în cadrul clasei, private sau publice.

Membrii unei clase reprezintă totalitatea metodelor şi a variabilelor membre ale clasei.

Se recomandă ca în general toate câmpurile să fie declarate private. În acest fel, este se *încapsulează* în obiect, iar accesul la ele se face prin metode publice care permit validarea modificărilor sau a citirilor de date din obiect:

```
class Animal {
private:
    std::string mName;
    std::string mColor;
    int mAge;
protected:
    bool mHasFeathers; // accesibil din eventualele clase derivate
public:
    virtual void makeSound() {
        printf("Animal %s makes a sound!\n", mName.c_str());}// this is a getter
method
    std::string getName() {
        return mName;}// this is a setter method
    void setAge(int age) {
        if(age > 0) { mAge = age;    }}};
```

Incapsulare în Python

Mai întâi va trebui să vorbim mai întâi despre atribute şi metode *private*. În Python nu există modificatori de acces ca în limbaje precum Java sau C++ , ci toate atributele şi metodele sunt *publice*. Public înseamnă că sunt accesibile din exteriorul clasei (printr-un obiect), din orice modul.

Un atribut sau o metodă privată va putea fi folosită doar în interiorul unei clase, şi nu va putea fi accesată din obiectul instanţiat. Putem simula acest comportament în Python, prefixând numele componentei respective cu dublu underscore __.

Exemplu:

```
class Impartire:
    def __init__(self, deimpartit, impartitor):
        self.deimpartit = deimpartit
        self.impartitor = impartitor
    def executa(self):
        return self.deimpartit / self.impartitor
obj = Impartire(9, 3)
print(obj.executa())
# Se va afisa 3.0
```

Încapsulare în Java

Pe scurt, încapsularea este procesul prin care "înfăşurăm" câmpurile unei clase prin nişte metode cu scopul de a avea mai mult control asupra valorilor acestora. În mod normal, acest procedeu este folosit pentru câmpurile declarate private.

De exemplu:

```
public class Caine{
    private String nume;
    private int lungimeCoada;
    public String getNume(){
        return nume;}
    public void setNume(String nume){
        this.nume = nume;}
    public void getLungimeCoada(){
        return lungimeCoada}
    public int setLungimeCoada(int lungimeCoada){
        this.lungimeCoada = lungimeCoada;}}
```

Observăm că ambele câmpuri din clasa de mai sus sunt declarate folosind modificatorul de acces private, prin urmare, acestea nu pot fi accesate din afara clasei curente.

Cu toate acestea, ne-am dori să avem totuşi o cale de a accesa şi modifica aceste câmpuri şi din exteriorul clasei, fără a modifica, totuşi, modificatorul de acces ales.

Pentru a rezolva această problemă este suficient să creăm o metodă declarată cu modificatorul de acces public (sau alt modificator de acces care ar permite accesarea metodei din afara clasei curente) care să returneze valoarea câmpului nostru şi o altă metodă declarată tot public care să schimbe valoarea câmpului cu o valoare primită ca parametru.

Metoda care returnează valoarea câmpului poartă numele de **getter**, iar cea care care modifică valoarea câmpului se numeşte **setter**.

Ex: 2, lab 3-4

Problema rezolvată C++

```
#include <iostream>
#include <string>
using namespace std;
class tablou
{
    public:
        string denumire;
        string curent;
        string autor;
        tablou(string x, string y, string z);
};
tablou::tablou(string x, string y, string z)
{
    denumire = x;
    curent = y;
    autor = z;
}
int main()
{
    tablou tablouObj1("Al 9-lea Val", "Romantism", "Ivan Aivazovsky");
        cout << tablouObj1.denumire << " " << tablouObj1.curent << " " <<
tablouObj1.autor << " "<<endl;
    tablou tablouObj2("Edecarii de pe Volga", "Realism", "Ilya Repin");
        cout << tablouObj2.denumire << " " << tablouObj2.curent << " " <<
tablouObj2.autor << " "<<endl;
    tablou tablouObj3("Fata cu piersici", "Ipresionism", "Valentin Serov");
        cout << tablouObj3.denumire << " " << tablouObj3.curent << " " <<
tablouObj3.autor << " "<<endl;
    return 0;
}
```

Varianta Python:

```
class Tablou:
    def __init__(self, denumire, curent, autor):
        self.denumire = denumire
        self.curent = curent
        self.autor = autor
    def seteazaDenumire(self, newDenumire):
          self.denumire = newDenumire
    def getDenumire(self):
          return self.denumire
    def afiseazaToateInformatiile(self):
        print('Denumire : ', self.denumire, '; Curent: ', self.curent,'; Autor: ',
self.autor)

tablou1=Tablou('Al 9-lea Val','Romantism','Ivan Aivazovsky')
tablou2=Tablou('Edecarii de pe Volga','Realism','Ilya Repin')
```

```
tablou3=Tablou('Fata cu piersici','Ipresionism','Valentin Serov')

tablou1.afiseazaToateInformatiile()
tablou2.afiseazaToateInformatiile()
tablou3.afiseazaToateInformatiile()
tablou1.seteazaDenumire('Al 9 Val')
tablou1.afiseazaToateInformatiile()
```

Cod Java, aceeași problemă:

```
public class tablou
{
		public String denumire;
		public String curent;
		public String autor;
		public tablou(String x, String y, String z)
		{
			denumire = x;
			curent = y;
			autor = z;
		}
}

package <missing>;

public class GlobalMembers
{
	public static void main(String[] args)
	{
		tablou tablouObj1 = new tablou("Al 9-lea Val", "Romantism", "Ivan
Aivazovsky");
			System.out.print(tablouObj1.denumire);
			System.out.print(" ");
			System.out.print(tablouObj1.curent);
			System.out.print(" ");
			System.out.print(tablouObj1.autor);
			System.out.print(" ");
			System.out.print("\n");
		tablou tablouObj2 = new tablou("Edecarii de pe Volga", "Realism", "Ilya
Repin");
			System.out.print(tablouObj2.denumire);
			System.out.print(" ");
			System.out.print(tablouObj2.curent);
			System.out.print(" ");
			System.out.print(tablouObj2.autor);
			System.out.print(" ");
			System.out.print("\n");
		tablou tablouObj3 = new tablou("Fata cu piersici", "Ipresionism",
"Valentin Serov");
			System.out.print(tablouObj3.denumire);
			System.out.print(" ");
			System.out.print(tablouObj3.curent);
			System.out.print(" ");
```

```
                    System.out.print(tablouObj3.autor);
                    System.out.print(" ");
                    System.out.print("\n");
        }
}
```