Curs 3 – POO

1. Initializare obiecte -> **metode constructor**
2. Distrugerea obiectelor -> **metoda destructor**
3. Metode **settler/getter** -> setare/extragere a datelor membre dintr-un obiect
4. Separarea implementarii unei clase de definitia sa -> **header** -> operatorul de resolutie

1) Initializare Obiectelor

Exp:

```
Class Complex{
        Double modul;
        Double re = 0;
        Double im = 0;
        Public:
                Void init(double re, double im){
                        This->re = re;
                        This->im = im;
                }
};
Int main(){
        Complex ob; //re,im cu VALORI REZIDUALE – OBIECT IN STARE IMPLICITA, VALORILE SUNT NULE
        Ob.init(3,4);
```

Commentato [AM1]: Complex calculat

Commentato [AM2]: Se aloca spatiu pnetru un obiect SI se apeleaza un constructor

OBSERVATII

1) Sunt 2 operatii distincte:
   i) Declararea a obiectului
   ii) Initializarea obiectului
2) Exista mai multe modalitati de a initializa un obiect

(Dae membre calculate, date membre ce vor fi initializate dupa declarare, copie a unui obiect)

Metode Constructorului

- Au rolul de a defini initializarea unui obiect
- O clasa poate sa contina mai multe **metode constructor**

Sintaxa

- Au aceasi denumire cu cea a clasei
- Nu returneaza **niciodata** nicio valoare
- Nu au tip returnat (nici void)
- Nu sunt **apelate explicit**

OBS) La declararea unui obiect, se apeleaza **IMPLICIT** o metoda constructor;

Tipuri de metode constructor

1) **Constructor fara argumente:** Are rolul de a initializa datele membre cu valori **implicite ≠ reziduala**
   Exp: Salariu = salariul minim;
   Varsta pentru o persoana de vot = 18;
   Anul de studii = 1;
2) **Constructor cu argumente:** Are rolul de a initializa datele membre cu **variabilele argumentelor**.
   Exemplu:
```
Complex(double re, double im){
        This->re = re;
        This->im = im;
}
```
3) **Constructorul de copiere:** Are rolul de a initializa un obiect cu **datele membre** ale altui obiect, **creat anterior**.
4) **Constructorul de conversie**

OBSERVATII:

1) Daca o clasa nu are constructor, atunci **compilatorul ataseaza unul implicit**, care initializeaza **date membre cu valori reziduale.**
2) O clasa poate avea mai **multi constructori**
   Ex.
```
Complex(){...}
Complex(double re, double im){...} -> TEHNICA DE SUPRAINCARCARE (OVERLOADING)
-----
Complex ob1; -> Apeleaza un constructor fara argumente
Complex ob2(2,3); -> Apeleaza un constructor cu argumente
Complex ob3(ob1) -> Apeleaza constructorul de copiere
int suma(int a, int b);
double suma(double a, double b);
suma(1,2);
suma(1.2, 1.5);
```
   Ex2: Modelare conceptului **PRODUS**
```
Class Produs {
        Char denumire[50];
        Double pret;
        Int stoc;
    Public:
        Produs(){
            Strcpy(denumire, "###");
            Pret = 0.0;
            Stoc = 0;
        }
        Produs(char *denumire, double pret, int stoc){
            Strcpy(this->denumire, denumire);
```

Commentato [AM3]: Pentru o clasa:
Class C {
C(){dm1 = val1 …}
}

Commentato [AM4]: Sintaxa:
C(tip1 arg1, tip2 arg2){
dm1 = arg1;
dm2 = arg2;
}

Commentato [AM5]: Prin **OVERLOADING**

Commentato [AM6]: Sintaxa:
C (C &obSursa)

Commentato [AM7]: Modalitate prin care o sursa poate sa contina mai multe functii cu **aceasi denumire**, dar care difera prin **numarul si tipul argumentelor.**

Commentato [AM8]: Char denumire[]

```
            This->pret = pret;
            This->stoc = stoc;
        }
        Produs(char *denumire){
            Strcpy(this->denumire, denumire);
            Pret = 0.0;
            Stoc = 0;
        }
        ~Produs(){};
        Void afisare(){
            Cout << denumire << " " << pret << " " << stoc << endl;
        }
};

Int main(){
    Produs p1; //s-a apelat constructor fara argumente
    P1.afisare; //### 0.0 0
    Produs p2('cafea',8.5,100); //s-a apelat constructorul cu 3 argumente

    Ifstream f("cale fisier");
```

2) Distrugerea Obiectelor -> Destructor

Metoda destructor: are rolul de a elibera zona de memorie alocata prin constructor;

Sintaxa:

- Are aceasi denumire a clasei, **precedatade simbolul ~**
- De regula, **nu are argumente**
- **nu are tip returnat**
- de regula, **nu are COD**

OBSERVATII:

1) Daca o clasa nu are metoda destructor, atunci compilatorul o ataseaza (pentru segmentul de memorie STACK)
2) Destructorul se poate apela in **2 meoduri**:
   a) Daca obiectul a fost alocat **STATIC** -> Atunci destructorul se apeleaza automat la inchiderea blocului)
   b) Obiectul a fost alocat **DINAMIC**
3) O clasa poate sa contina **DOAR UN SINGUR CONSTRUCTOR (NO OVERLOADING);**

4) Metode setter & getter

```
Produs ob;
Ob.pret = 8.5; ERROR -> pret is private
Cout << ob.pret; ERROR -> pret is private
```

   a) Metodele setter au rolul de a schimba (**seta**) valoarea unei date membre
   Void setData(tip val){ data = val; }
   *Void setDenumire(char *sir){ strcpy(denumire,sir); }*

Commentato [AM9]: Pentru o clasa C {... public: ~C()}

Commentato [AM10]: Produs ob1;

Commentato [AM11R10]: STACK

Commentato [AM12]: Produs *ob = new Produs();

Delete ob; (se apeleaza destructorul).

Commentato [AM13R12]: HEAP

Commentato [AM14]: ORICE FUNCTIE IN CLASA TREBUIE SA FACA ATOMIC **DOAR UN SINGUR LUCRU**

Void setPret(double val){ pret=val; }

b) Metodele getter au rolul de a **returna valoarea** unei date membre.
Sintaxa: tip_returnat getData() {
Return data;
}
Double getPret() { return pret; };
Cout << ob.getPret();
Int getStoc() { return stoc };
Char getDenumire() {return denumire};*

STRUCTURA UNEI CLASE

1. **Datele membre** (private)
2. **Metode constructor** pentru a initializa datele membre
3. **Metoda destrctor**
4. **Metode getter/setter**
5. **Metoda de afisare** (deocamdata)
6. **Metode specifice** tipului de data modelat (calcul ValStoc, calculNrCredite, modulComplex)