Tutoriat 2
Moștenire și compunere

1. Compunerea

```
class Student {
   ....
};
class Facultate {
   Student s[100]; // compunere
};
```

   a. Ce este?
   Compunerea reprezintă **declararea unui obiect** de tipul unei clase, ca dată membră a altei clase.

2. Moștenirea

   a. Ce este?
      - Al doilea principiu fundamental în POO.
      - Extinderea unei clase, prin crearea altor clase cu proprietăți comune.
      - Se realizează folosind **simbolul :** pus între numele clasei derivate și modificatorul de acces(opțional) alături de numele clasei de bază.

   b. Clasă de bază vs clasă derivată
      - Clasă de bază – cea **din care** se moștenește
      - Clasă derivată – clasa **care** moștenește
      - Obs: Tot ceea ce este **private** în clasa de **bază** devine ***inacesibil*** în clasa derivată.

```
class Animal {
   // clasa de bază
};
class Dog : Animal {
   // clasa derivată
};
```

   c. Tipuri de moșteniri
      i. Moștenire **private**
         - Totul din clasa de bază devine private în clasa derivată.
         - Datele și metodele care sunt deja private în clasa de bază devin inaccesibile în clasa derivată.
         - Tip de moștenire by default când nu este specificat modificatorul de acces la moștenire.
      ii. Moștenire **protected**
         - Tot ce nu e private în clasa de bază devine protected în clasa derivată.
      iii. Moștenire **public**
         - Aceasta este cea mai des folosită moștenire.
         - Totul rămâne la fel ca în clasa de bază.
         - Datele și metodele private tot inaccesibile rămân.

When the component is declared as:	When the class is inherited as:	The resulting access inside the subclass is:
public	public	Public
protected		protected
private		none
public	protected	protected
protected		protected
private		none
public	private	private
protected		private
private		none

d. Constructori la moștenire
   - Apelați în ordine inversă de la clasa **derivată** spre clasa de **bază**.
   - Se pot apela anumiți constructori din clasa de bază folosind **lista de inițializare.**
   - Obs: Este obligatoriu ca în clasa de bază să existe constructorul fără parametrii(dacă nu apelăm explicit alt constructor).

```
class Animal {
    public:
        Animal() {
            cout << "Animal() ";
        }
        Animal(int x) {
            cout << "Animal(x) " << x << " ";
        }
};
class Dog : public Animal {
    public:
        Dog() : Animal(4) {
            cout << "Dog ";
        }
};
int main() {
    Dog d; // Animal(x) 4 Dog
    return 0;
}
```

e. Destructori la moștenire
   - Apelați în ordine de la clasa de **bază** spre clasa de **derivată**.

```
class Animal {
    public:
        Animal() {
            cout << "C: animal ";
        }
        ~Animal() {
        cout << "D: animal ";
        }
};
class Dog : public Animal {
    public:
        Dog() {
            cout << "C: dog ";
        }
        ~Dog() {
        cout << "D: dog ";
        }
};
int main() {
    Dog d; //C: animal C: dog D: dog D: animal
    return 0;
}
```

f. Moștenirea multiplă

- O clasă poate moșteni mai multe clase în același timp.
- Obs: la moștenirea multiplă, constructorii sunt apelați **în ordinea în care sunt specificați la moștenire**(indifferent de ordinea din lista de inițialziare).

```
class Animal {
    public:
        Animal() {
            cout << "Animal ";
        }
};
class Reptile {
    public:
        Reptile() {
            cout << "Reptile ";
        }
};
class Snake : public Animal, public Reptile {
    public:
        Snake() {
            cout << "Snake ";
        }
};
```

```
int main() {
    Snake s; // Animal Reptile Snake
}
```

3. Mostenire vs compunere

    a. Compunerea: o clasă **are** un obiect de tipul altei clase.
    Ex: Facultatea are mai mulți studenți/profesori/cursuri…
        Firma are mai mulți ingineri/manageri/directori…
    b. Moștenirea: o clasă **este** de tipul altei clase(are câmpuri comune cu aceasta).
    Ex: Animal este câinele/pisica/pasărea....
        Forma geometrică este pătratul/rombul/dreptunghiul...

4. Probleme frecvente

    a. Lipsa de acces în clasa derivată (avem date membre **private** în clasa de **bază**).
    b. Tipul de moștenire(moștenire private transformă cosntructorii în private).

```
class Animal {
    public:
        Animal() {
            cout << "Animal ";
        }
};
class Dog : Animal {// Mostenire private
    // aici constructorul din Animal e private
    public:
        Dog() {
            cout << "Dog ";
        }
};
class Puppy : public Dog {
    // aici constructorul din Animal nu e accesibil pentru ca e private in Dog
    public:
        Puppy() {
            cout << "Puppy ";
        }
};
int main() {
    Puppy p; // eroare la compilare
    return 0;
}
```