OOP CURS 5

Silviu Ojog

LINK Academy

Recapitulare

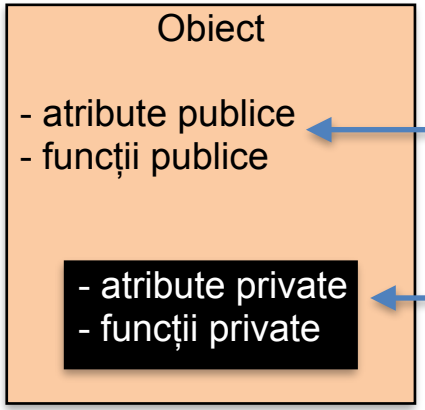
- Ce știm până acum despre OOP?
  - Obiect vs clasa?
  - Ce este un constructor?
  - Ce înseamnă self?
  - Ce înseamnă __init__?
  - Ce înseamnă __str__?

Recapitulare

- Ce știm până acum despre OOP?
  - Variabila publica, interna, privata?
    - self.variabilaPublica
    - self._variabilaInterna (self._variabilaProtected)
    - self.__variabilaPrivata

Încapsulare

Procesul prin care ținem datele și funcțiile separate de exterior. (atribute și funcții private)



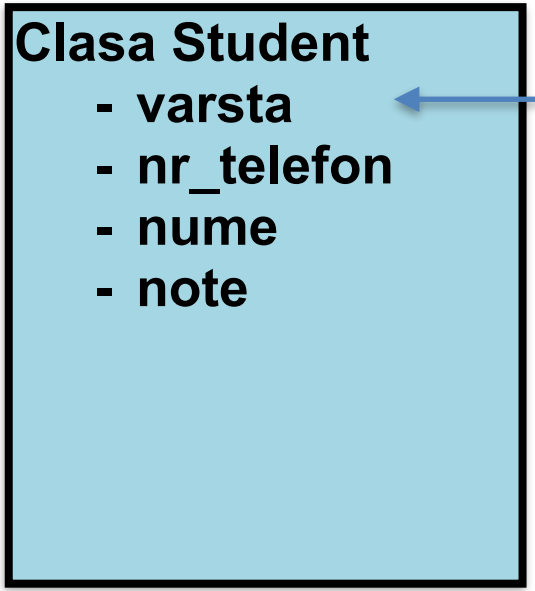
Accessible oriunde, în cadrul definirii clasei, și in exteriorul ei

Accessible doar în cadrul definirii clasei

Componentele private

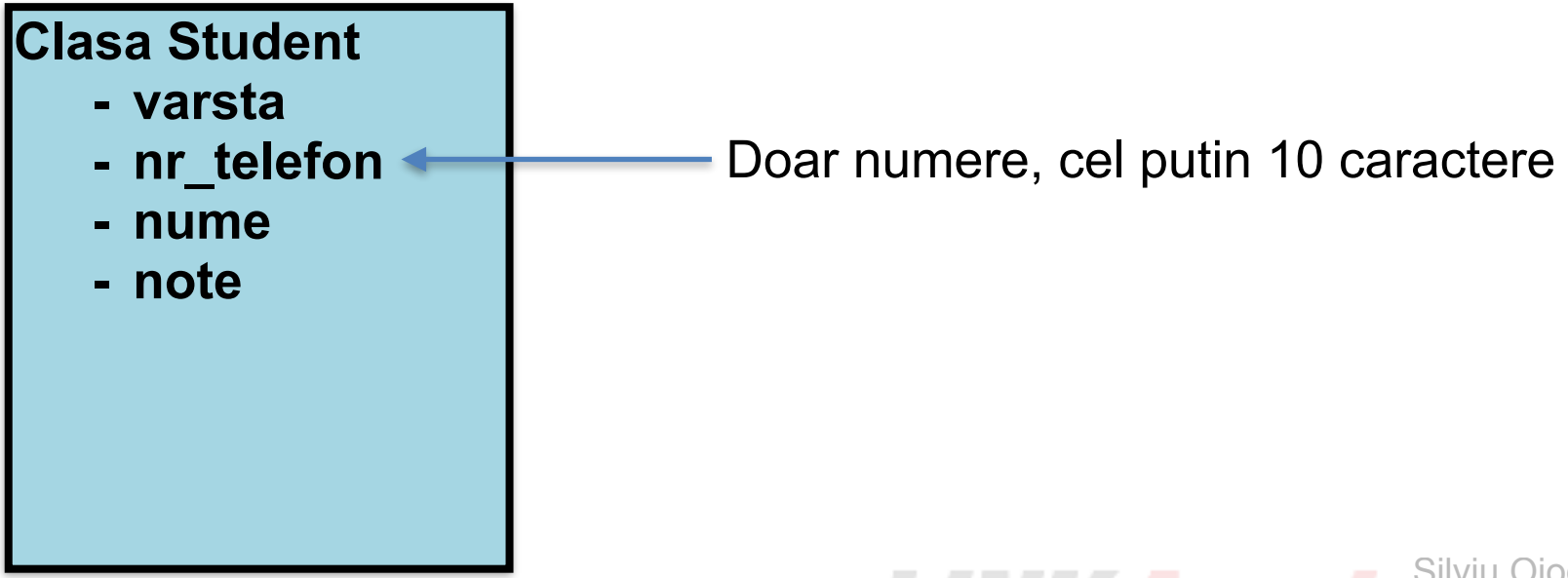
- By default în multe limbaje de programare metodele și atribuible sunt private.
- Componentele private pot fi “accesate” din exteriorul obiectului cu ajutorul unor metode publice. (getter și setter)
- Scopul este de verifica valorile atribuite.

Componentele private

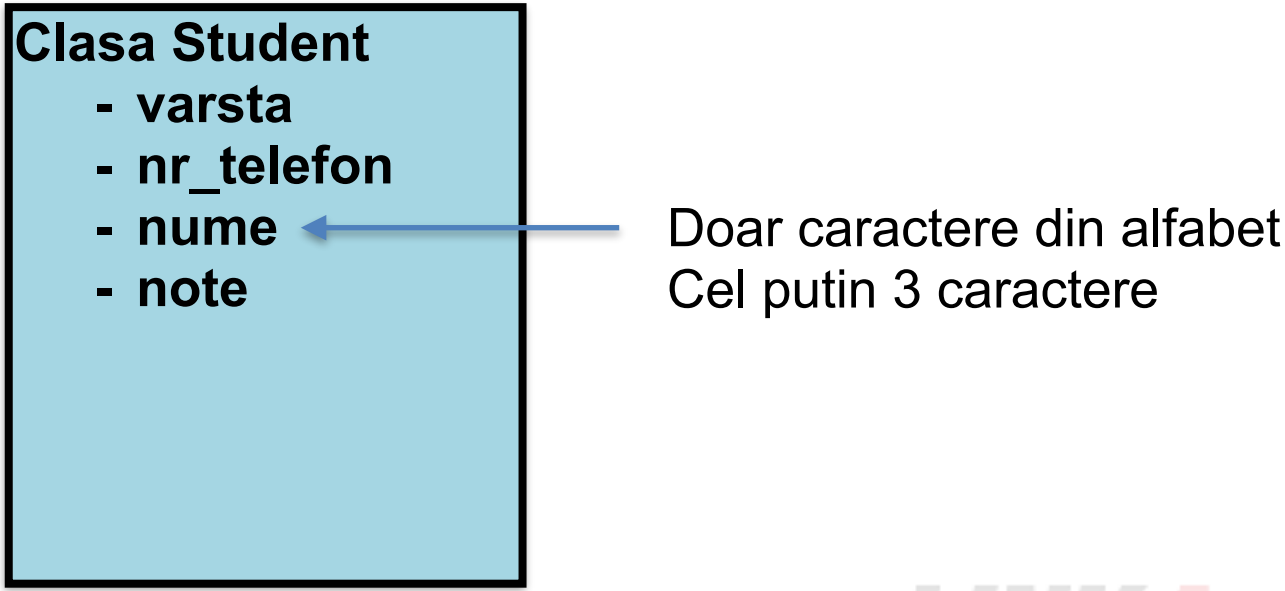


Cel puțin 16/18 ani?
Maxim 120 ani?
Nu pot fi insera valori de tip “@#@”
“kjds”, “-”

Componentele private



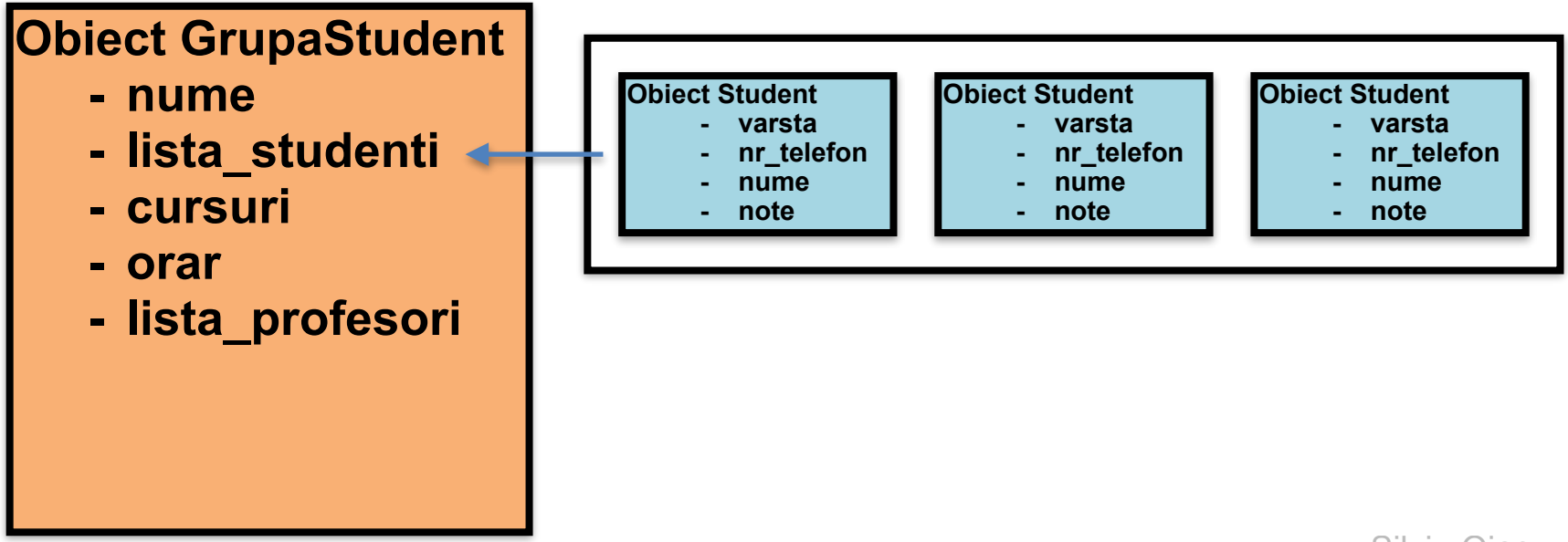
Componentele private



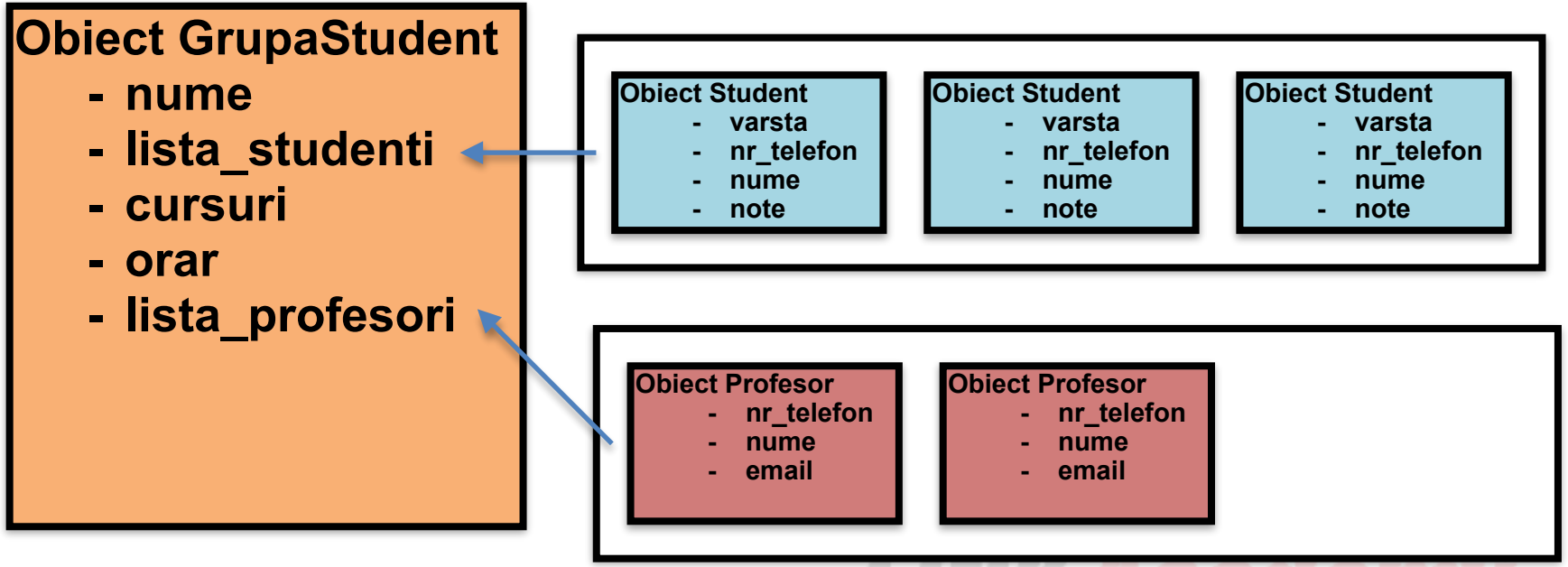
Componentele private

Clasa GrupaStudent
- nume
- lista_studenti
- orar
- lista_profesori

Componentele private



Componentele private

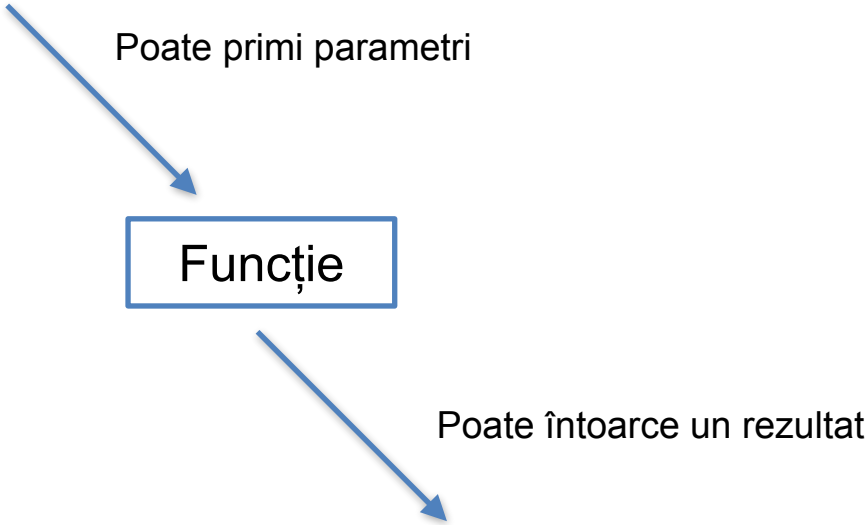


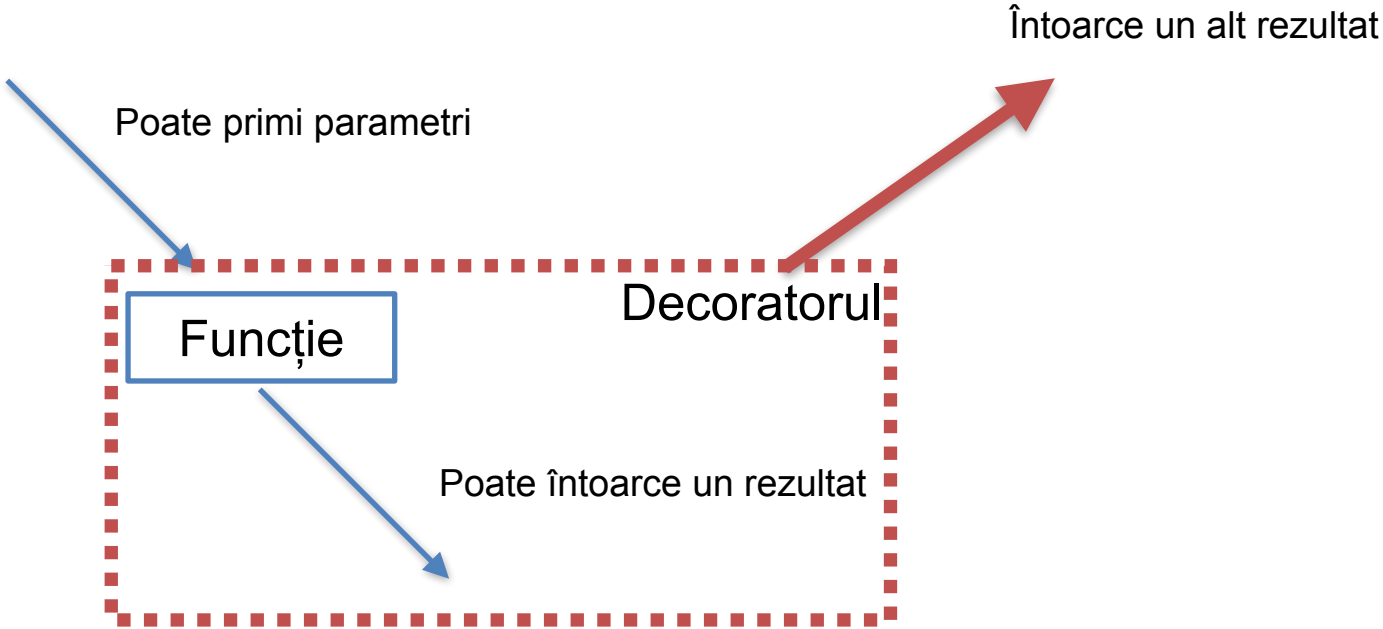
Recapitulare

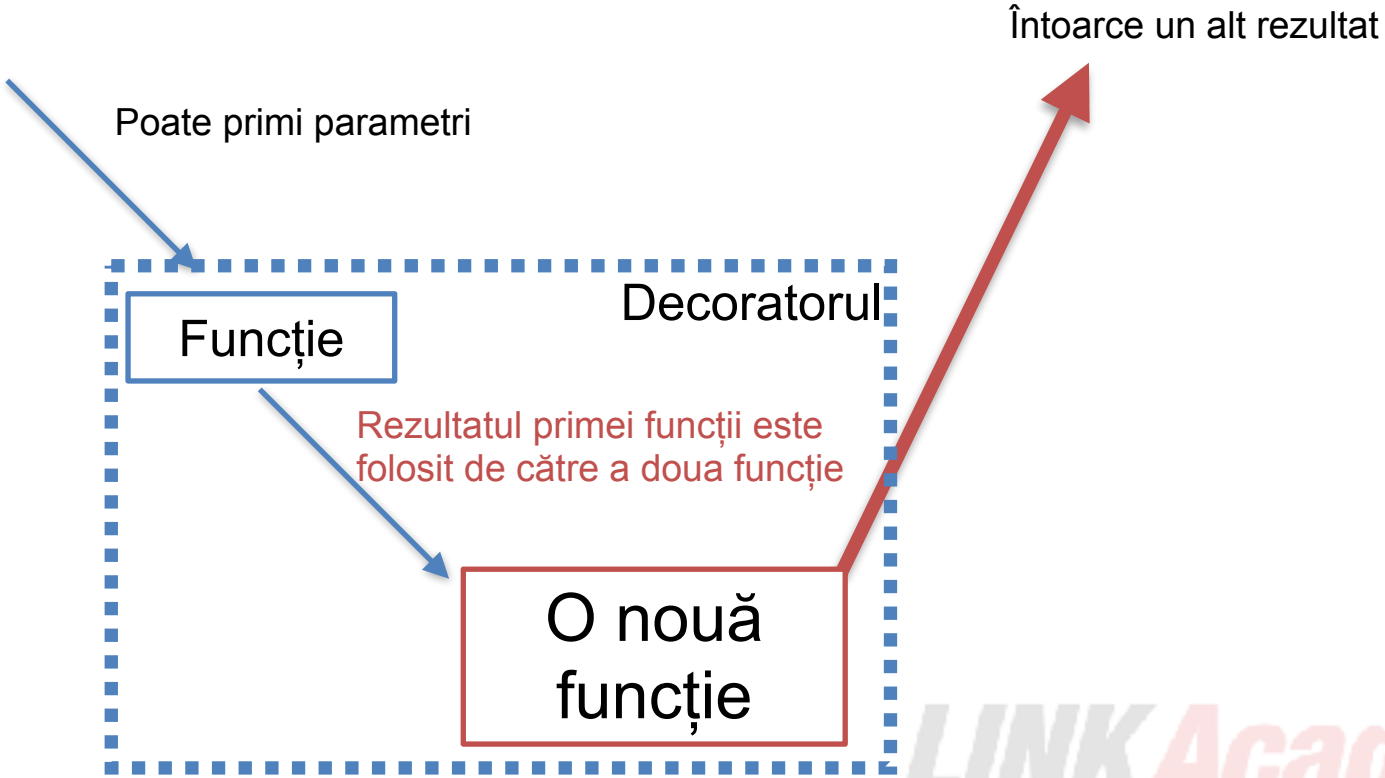
- Ce știm până acum despre OOP?
  - Variabila statica/clasa
    - Apartine la nivel de clasa
    - Poate fi accesat de catre oricare obiect
    - Modificarea lor se face peste tot

Exercițiul cu Punctul

- Trebuie creată clasa Point care sa abstractizeze un punct dintr-un plan (coordonate x si y)
- Clasa deţine următoarele caracteristici:
  - **Punctul x** si **punctul y**
- Clasa deţine următoarele implementari:
  - **show** - se printeaza coordinatele punctului;
  - **move** - se muta coordonatele;
  - **setToOrigin** - se seteaza coordinatele (0, 0);
  - **dist** - distanta intre doua puncte







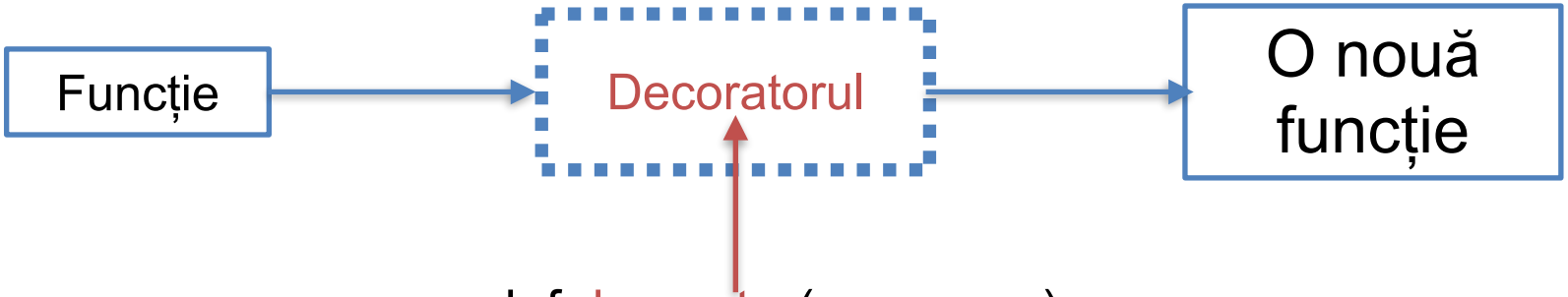
Un decorator transformă o funcție în altă funcție

Un decorator transformă o funcție în altă funcție


```
def functie(params…):
    #corpul funcției
```

Funcția inițială

Decoratorul este tot o funcție




```
def functie(params…):
    #corpul funcției
```

```
def decorator(params…):
    #corpul decoratorului
```

Decoratorul primește ca parametru prima funcție


```
def functie(params…):
    #corpul funcției
```

```
def decorator(funcție):
    #corpul decoratorului
```

Decoratorul returnează o nouă funcție


```
def functie(params…):
    #corpul funcției
```

```
def decorator(funcție):
    #corpul decoratorului

    return o_nouă_funcție
```

Decoratorul definește o nouă funcție


```
def functie(params…):
    #corpul funcției
```

```
def decorator(funcție):
    #corpul decoratorului

    return o_nouă_funcție
```

Trebuie definită o nouă funcție

Decoratorul definește o nouă funcție


```
def functie(params…):
    #corpul funcției
```

```
def decorator(funcție):
    def o_nouă_funcție(params):
        #Corpul noii funcții

    return o_nouă_funcție
```

Trebuie definită o nouă funcție

Noua funcție folosește prima funcție


```
def functie(params…):
    #corpul funcției
```

```
def decorator(funcție):
    def o_nouă_funcție(params):
        # Aici se cheamă noua funcție
        # Se poate returna ceva
        # în locul primei funcții
    return o_nouă_funcție
```

Logica noii funcții

Exercițiul cu TimeConverter

1. Creați clasa TimeConverter cu două proprietăți(atribute):
- **hours** - numărul de ore (int)
- **minutes** - numărul de minute (int)

2. Clasa trebuie să conțină următoarele metode publice:
- **toMinutes**() - convertește timpul total în minute
- **toHours**() - convertește timpul total în ore
- **addTime**() - adună încă un interval de timp și returnează rezultatul în minute
- **diffTime**() - calculează diferența față de alt interval de timp și returnează rezultatul în minute