# Programare avansata pe obiecte - Laborator 2

***Alina Puscasu*** *alina.puscasu@endava.com*

***https://github.com/alina-puscasu/pao_lab_2022***

## O clasa poate contine:

- **Field**-uri: retin starea
- **Metode**: manipuleaza starea si efectueaza operatii
- **Constructori**: creeaza obiectele, initializeaza starea
- Ca sa mostenim o clasa folosim cuvantul cheie **extends**
    1. Toate clasele din java sunt copii ai clasei **Object (java.lang package)**
        - ✦ **toString**
        - ✦ **hashCode**
        - ✦ **equals**
    2. Java nu suporta mostenirea multipla

## Obiecte

- Cand declaram un obiect, alocam spatiu pentru a stoca referinta catre acesta
- Crearea unui obiect se face prin **constructor**
    1. prin cuvantul cheie **new**, cream obiectul si ii asignam o referinta
    2. by default, compilatorul creaza un constructor fara parametri
    3. cand invocam in clasa copil un constructor al clasei parinte folosind cuvantul cheie super, acest apel trebuie sa fie prima linie din constructorul clasei copil
- **this** se refera la starea curenta a obiectului
- **null** nu pointeaza catre nimic (niciun bloc de adresa) dar poate fi asignat referintelor
- Cuvantul cheie **super** este folosit pentru a accesa variabile si metode din clasa parinte

## Metode

```
access_modifier return_type name (typed_parameter_list) {
        body
}
```

- Cand nu returnam nicio valoare, folosim pentru return type: void
- Lista de parametri este optionala, asadar putem avea **methodName()**
- Putem defini o metoda cu acelasi nume in aceeasi clasa, dar ea trebuie sa difere prin numarul si/sau tipul parametrilor -> acest proces se numeste **supraincarcare** (**overloading**)
- Putem defini o metoda cu aceeasi semnatura ca cea din clasa parinte in clasa copil -> acest proces se numeste **suprascriere** (**overriding**)

*Equals and Hashcode*

- Folosim **equals** pentru a verifica daca doua obiecte sunt egale. Aceasta egalitate se poate face:
    1. Shallow: doar verificam daca e vorba de aceeasi referinta
    2. Deep: comparam si starea obiectelor pe baza membrilor sai
- Folosim **hashcode** mai ales in colectii (HashMap, HashSet, HashTable). Aceasta metoda trebuie implementata in fiecare clasa unde implementam si equals
- *Daca 2 obiecte sunt egale, inseamna ca ele au si acelasi hashcode; viceversa nu este valabila!*

## Modificatori de acces pentru datele membre/metode – in ordine descrescatoare:

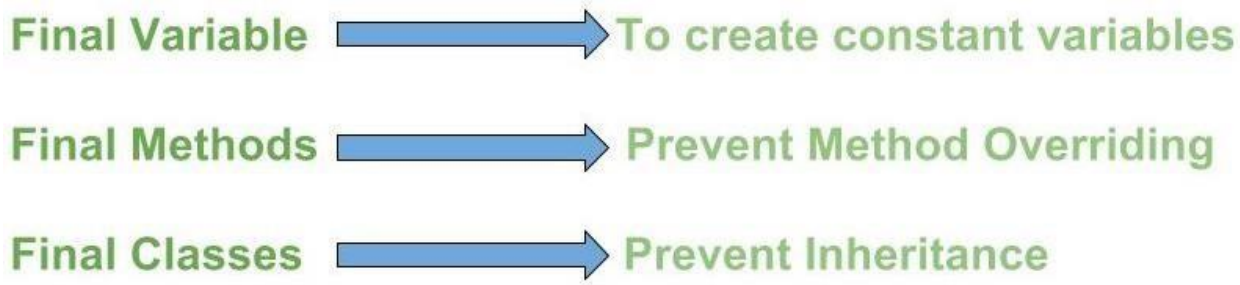
| Specificator | Vizibilitate in clasa | Vizibilitate in subclasa | Vizibilitate in pachet | Vizibilitate oriunde |
|---|---|---|---|---|
| **Public** | x | x | x | x |
| **Protected** | x | x | x | |
| **Default** | x | | x | |
| **Private** | x | | | |

- Este o buna practica sa accesam field-urile unei clase prin metode **publice** de tip **getter** si **setter**, iar field-urile sa fie declarate **private**

## Modificatorii ***final*** si ***static*** pentru datele membre/metode

### Final

- Putem folosi pe clase, variabile si metode




### Static

- Putem folosi pe clase, variabile, metode dar si blocuri
- Nu e nevoie de o instanta o clasei ca sa le putem accesa
- Static aplicat pe o variabila inseamna ca valoarea acesteia este impartita intre toate instantele clasei; orice modificare se va reflecta peste tot

- Metodele statice pot chema doar alte metode statice si pot contine doar date statice
- Spre deosebire de C++, in Java intalnim si blocuri statice dar nu putem avea variabile locale statice!
- Blocurile statice sunt de obicei folosite pentru initializarea variabilelor si se executa o singura data, cand se initializeaza clasa

Este o buna practica ca variabilele de tip constante sa fie si final si static! Ca si naming convention, folosim litere mari si daca sunt mai multe cuvinte le separam cu _: **MY_CONSTANT**.

## Ordinea initializarii

1. Blocurile statice, in ordinea aparitiei lor
2. Fieldurile si blocurile de initializare; ele se executa inaintea constructorilor dar nu inainte ca acesta sa fie chemat
3. Constructori

## Exercitii

1. Write a program to create a Person object, with the following attributes:

   - name as string
   - surname as string
   - age as int
   - identity number as long
   - type as string.

   Define a constructor for this class as well as accessors (getters) and mutators (setters) for all the attributes.

   Create two objects of type Person and display the information for them on separate lines.

2. Write a program to create a Room object, the attributes of this object are:

   - room number
   - room type
   - room floor

   Define a constructor for this class as well as accessors and mutators for all the attributes.

   Create two objects of type Room and display the information for them on separate lines.

3. Write a program to create an object Subject with the following attributes:

   - room as Room
   - noOfStudents as integer
   - teacher as Person

   Define a constructor for this class as well as accessors and mutators for all the attributes.

   Create two objects of type Subject and display the information for them on separate lines.

4. Implement a Singleton class as you learned in the course.