Programare avansata pe obiecte – laborator 2

***Andreea Ciocan***

[*Andreea.ciocan@endava.com*](mailto:Andreea.ciocan@endava.com)

https://github.com/deeas/PAO-labs-2024

## O clasa poate contine:

* Field-uri: tin starea
* Metode: manipuleaza starea si efectueaza operatii
* Constructori: crearea obiectelor, initializare
* 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**
  1. 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 return type void
* Lista de parametri e optionala
* Putem define o metoda cu acelasi nume in aceeasi clasa, dar ea trebuie sa difere prin numarul si/sau tipul parametrilor -> acest proces se numeste **supraincarcare**
* Putem define o metoda cu aceeasi semnatura ca cea din clasa parinte in clasa copil -> acest proces se numeste **suprascriere**

*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

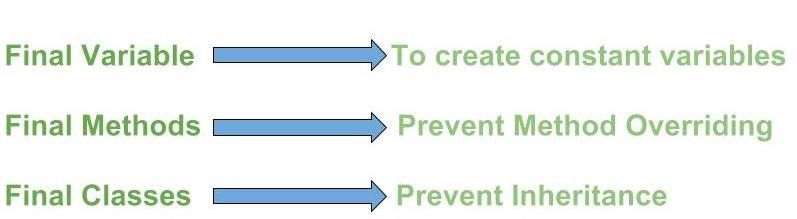
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Specificator** | **Clasa** | **Subclasa** | **Pachet** | **Oriunde** |
| **Private** | x |  |  |  |
| **Protected** | x | x | x |  |
| **Public** | x | x | x | x |
| **Default** | x |  | x |  |

- Este o buna practica sa accesam field-urile unei clase prin metode de tip getter si setter

# Modificatorii static si final pentru datele membre/metode

## Final

- Putem folosi pe clase, variabile si metode



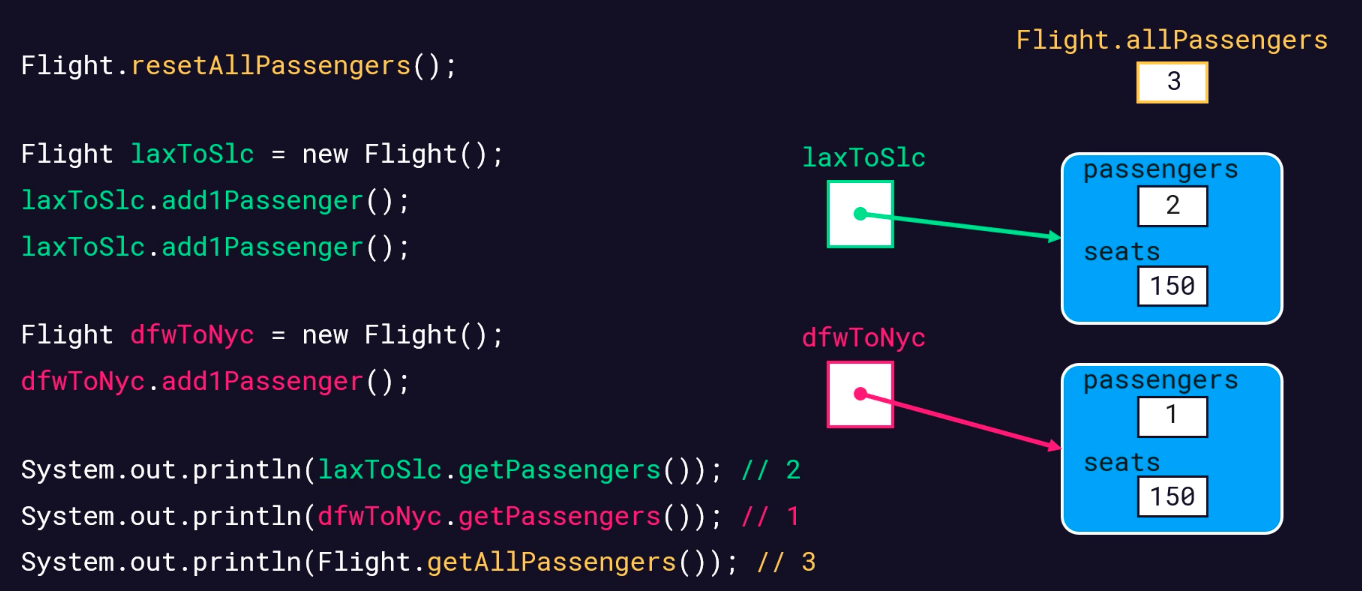
## Static

* 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

E o buna practica ca variabilele de tip constante sa fie si final si static! Ca si naming convention, folosim litera mare si daca sunt mai multe cuvinte le separam cu \_.

A screen shot of a computer code

Description automatically generated



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. Să se creeze o clasă numită Complex, care are doi membri de tip int (real și imaginary),

care vor fi private. Realizați următoarele subpuncte:

* să se creeze trei constructori: primul primește doi parametri de tip int

(primul desemnează partea reală a unui număr complex, al doilea partea imaginară),

al doilea nu primește niciun parametru și apelează primul constructor cu valorile 0 și 0,

* să se scrie metode de tip getter și setter, prin care putem accesa membrii privați ai clasei
* să se scrie o metodă de tip void numită addWithComplex, care primește ca parametru un

obiect de tip Complex, prin care se adună numărul complex dat ca parametru la numărul

care apelează funcția (adică la this)

* să se scrie o metodă de tip void numita showNumber, prin care se afișează numărul complex

2) Implementeaza o clasa numita ChocolateBox, care va conține:

* câmpurile private flavor (String) și origin (String)
* un constructor fără parametri
* un constructor ce va inițializa toate campurile
* o metoda de tip float getVolume(), care va intoarce valoarea 0;
* suprascrie metoda toString()

Din ea derivați clasele Milka, Oreo, Merci, astfel:

* Milka va contine length, width, height
* Oreo va conține un camp radius și unul height (cilindru)
* Merci, fiind un cub, va conține un camp lenght

Clasele vor avea:

* constructori fără parametrii
* constructori care permit inițializarea membrilor.
* Suprascrieti metoda getVolume() pentru a intoarce volumul specific fiecarei cutii de ciocolata, in functie de tipul sau.
* Suprascrieti metoda toString() în clasele derivate, astfel încat aceasta să utilizeze implementarea metodei toString() din clasa de bază.
* Suprascrieti metoda equals()

1. In model package
   1. Create a class Animal, with the following attributes: name as string, sound as string, age as int, noOfLegs as int initialized with 4, food as string. Define:

* A default constructor for this class
* A constructor with noOfLegs as parameter and print number of legs in it
* A constructor with name and sound as params
* accessors and mutators for all the attributes.
* two public methods to print animal name and sound
  1. Create a class Bird which extends Animal, has a constant: NO\_OF\_LEGS = 2, overrides all the methods from parent and has a constructor with one parameter: sound which calls the parent constructor
  2. Create an enum CatBread
  3. Create a class Cat which extend Animal with catBread as attribute. Add a constructor with params: name, sound and catBread
  4. Create a class AnimalHotel with the following attributes: a constant for number of rooms available, an array of animals with dimension = number of rooms and a preinitialized hotelName

In service package create a service called AnimalService to expose two methods:

addAnimal with parameters: AnimalHotel, Animal, and an int for position, to add an animal in the hotel on the specified position

printAnimals with one parameter AnimalHotel, to print all the animals in the hotel with their name, and what is it (Bird or Cat)

In main package, create a class called PlayWithAnimal. In main method add in the hotel a cat called Mimi, age 9 and a bird called Cip, age 2