Academia Tehnică Militară "Ferdinand I“
Facultatea de Sisteme Informatice şi Securitate Cibernetică

Paradigme de programare (în Java)

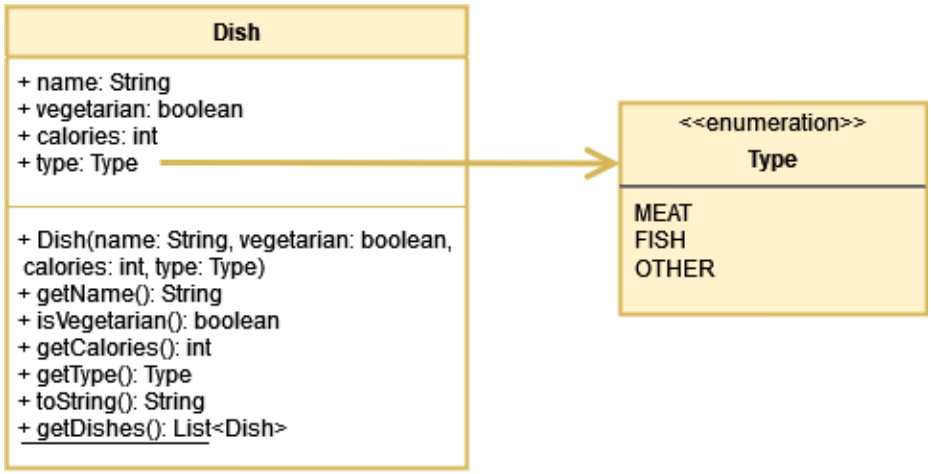
Lab 6/Curs 6- Programare funcțională

Col(r) Traian Nicula

Se creează un proiect Maven cu numele ***lab6***, folosind ca artefact **archetype-quickstart-jdk8** și se salvează în folder-ul cu numele studentului. Pentru fiecare exercițiu se va crea câte o clasă de test (Exercise1 etc.) cu metoda statică *main*, precum și alte clase cerute de exerciții.

Se rezolvă următoarele exerciții:

1. Scrieți un program Java care: (**1.5p**)
   - creează clasele din diagrama UML de mai jos



   - creează o metoda statică *List<Dish> getDishes()* în clasa ***Dish*** care returnează o listă conținând instanțe ale clasei ***Dish*** construite cu valorile:

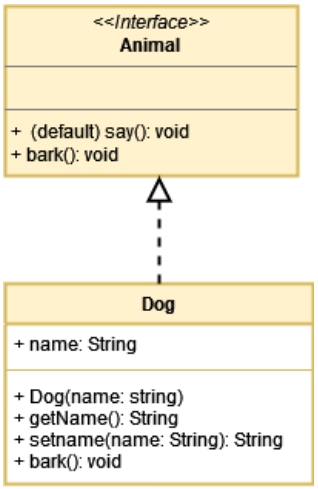
Name	Vegetarian	Calories	Type
pork	false	800	Type.MEAT
beef	false	700	Type.MEAT
chicken	false	400	Type.MEAT
french fries	true	530	Type.OTHER
rice	true	350	Type.OTHER
season fruit	true	120	Type.OTHER
pizza	true	550	Type.OTHER
prawns	false	400	Type.FISH
salmon	false	450	Type.FISH

   - folosește Stream API cu metoda statică *Collectors.counting()* pentru a număra felurile de mâncare returnate de *getDishes().*
   - afișează rezultatul

2. Scrieți un program Java care: (**1p**)
   - creează un Comparator ce compară felurile de mâncare în funcție de calorii
   - folosește Stream API cu metodele *Collectors.maxBy* și *Collectors minBy* pentru a determina valoarea maximă și minimă a caloriilor conținute de felurile de mâncare din lista returnată de metoda statică *Dish.getDishes()*
   - afișează rezultatele.
3. Scrieți un program Java care: (**1p**)
   - folosește Stream API cu metodele *Collectors.summingInt* și *Collectors.averagingInt* pentru a determina suma și valoarea medie a caloriilor conținute de felurile de mâncare din lista returnată de metoda statică *Dish.getDishes()*
   - afișează rezultatele.
4. Scrieți un program Java care: (**1p**)
   - folosește Stream API cu metoda *Collectors. summarizingInt* pentru a colecta statistici privind caloriile conținute de felurile de mâncare din lista returnată de metoda statică *Dish.getDishes()*
   - afișează rezultatele.
5. Scrieți un program Java care: (**1p**)
   - folosește Stream API cu metoda *Collectors.joining* pentru a genera o listă, separată cu **“, ”**, a numelor felurilor de mâncare din lista returnată de metoda statică *Dish.getDishes()*
   - afișează rezultatele.
6. Scrieți un program Java care: (**1p**)
   - folosește Stream API cu metoda *Collectors.reducing* pentru a determina suma caloriilor conținute de felurile de mâncare din lista returnată de metoda statică *Dish.getDishes()*
   - afișează rezultatele.
7. Scrieți un program Java care: (**1p**)
   - folosește Stream API cu metoda *Collectors.groupingBy* pentru a clasifica în funcție de tip felurile de mâncare din lista returnată de metoda statică *Dish.getDishes()*
   - afișează rezultatele.
8. Scrieți un program Java care: (**1.5p**)
   - creează enumerația *CaloricLevel* cu valorile DIET, NORMAL, FAT
   - folosește Stream API cu metoda *Collectors.groupingBy* pentru a clasifica felurile de mâncare returnare de metoda statică *Dish.getDishes(),* în funcție de calorii, astfel:
     a. DIET – calorii <= 400
     b. NORMAL – calorii > 400 și <= 700
     c. FAT – calorii > 700
   - afișează rezultatele.
9. Scrieți un program Java care: (**1p**)
   - folosește Stream API cu metoda *Collectors.groupingBy* pentru a clasifica în funcție de tip și a determina numărul felurile de mâncare din lista returnată de metoda statică *Dish.getDishes()*
   - afișează rezultatele.

Temă pentru acasă:

10. Scrieți un program Java care: (**0.5p**)
    - folosește Stream API cu metoda *Collectors.partitioningBy* și *Collectors.groupingBy* pentru a clasifica felurile de mâncare din lista returnată de metoda statică *Dish.getDishes()* în vegetariene sau nu, iar în cadrul fiecărei categorii să fie clasificate după tip
    - afișează rezultatele.
11. Scrieți un program Java care: (**1p**)
    - folosind Stream API creează o metodă statică, cu semnătura *long squareSumSequential(long interval),* care calculează în mod secvențial suma pătratelor unui interval închis de numere
    - folosind Stream API creează o metodă statică, cu semnătura *long squareSumParallel(long interval),* care calculează în mod paralel suma pătratelor unui interval închis de numere
    - creează o metoda pentru măsurarea performanței cu semnătura *long measurePerformance(Function<Long, Long> test, long n)* (vezi cursul)
    - testează și afișează performanța în milisecunde pentru o valoare de 1_000_000, atât pentru modul secvențial cât și pentru modul paralel.
12. Scrieți un program Java care: (**0.5p**)
    - creează obiectele din diagrama UML de mai jos astfel:
        a. implementează metoda implicită *say()* în interfața *Animal* ca să tipărească un mesaj
        b. suprascrie metoda bark() în clasa ***Dog*** ca să tipărească un alt mesaj



    - creează o instanță a clasei Dog și cheamă metodele *say()* și *bark()* pe acesta.