**Metode Avansate de Programare, Lab1 – duck_taskrunner**

**Termene de predare (Deadline 😉): – fiecare săptămâna de întârziere va diminua nota finală cu 1 punct**

- **Săptămâna 3:** Diagrama UML realizata in StarUML (https://staruml.io/) cu proiectarea problemei Natatie
- **Săptămâna 4:** Implementarea Java, bazată pe diagrama creată anterior

## Contents



## Precondiții

Instalați un mediu de dezvoltare intregrat pt Java (Java IDE ) pe laptopurile proprii și încercați să vă familiarizați cu acesta.

> *La curs si seminar vom folosi*: InteliJ IDEA, Ultimate Edition (Pentru a putea activa licenta, creati un JetBrains Account cu adresa de mail de la facultate).

Instalați tool-ul STARUML pt crearea diagramei UML.

Sursele se vor uploada pe git, vezi linkul de mai jos.

- **STARUML:** https://staruml.io/
- **GIT studenti: https://classroom.github.com/a/oqRSTCM9**

- **JDK: minim versiunea 8 (recomandat versiunea 17)**
- **IDE: IntelliJ IDEA https://www.jetbrains.com/idea/**
- **TEAMS: enunturi lab si cursuri**

# Enunț Problema Natatie

**Modelați, proiectați și implementați (în limbajul Java) problema Natatie, enunțată mai jos, folosind paradigma de programare orientată obiect (POO).**

## Fișier de intrare: natatie.in

### Enunt:

Prințul Mugurel trebuie să organizeze un nou spectacol pentru locuitorii din **Imperiul Rațelor de Cauciuc**. De data aceasta s-a gândit la ceva inedit: o cursă de natație pe Râul Macilor. Mugurel a adunat cele mai bune **N** raţe din imperiu, numerotate de la 1 la N, fiecare **rață** fiind caracterizată prin **viteză** şi nivel de **rezistenţă**.

Pe Râul Macilor s-au amenajat **M** culoare de înot, numerotate de la 1 la M; pe fiecare culoar este câte o baliză, situată la o anumită distanță (în metri) față de linia de start, iar această distanță este strict mai mare decât distanța balizei de pe culoarul anterior. Mugurel alege M rațe dintre cele N care sunt așezate adecvat la linia de start, fiecare pe câte un culoar de înot. Apoi, toate aceste rațe alese pornesc simultan, fiecare rață înoată pe culoarul ei, până la baliza corespunzătoare, și se întoarce înapoi la linia de start, pe același culoar. Durata cursei se măsoară de la pornirea simultană a rațelor, până la momentul când toate rațele ajung înapoi la linia de start. Deoarece Mugurel ține la sănătatea locuitorilor săi, nu dezavantajează raţele mai puţin rezistente. Așadar, rața care înoată până la baliza de pe culoarul ei trebuie să aibă nivelul de rezistență mai mare sau egal cu al celei de pe culoarul alăturat, numerotat cu o valoare mai mică. Mugurel dorește să ofere un show de neuitat tuturor spectatorilor, așa că vrea să obțină un nou record imperial, alegând rațele corespunzător, astfel încât cursa să se încheie cât mai repede.

### Cerință

Determinați durata minimă pe care o poate avea cursa.

Observatie: Pentru determinarea timpului minim se poate face apel la orice metode de rezolvare a problemei (Cautare Binara, Programare Dinamica sau Backtracking etc)

### Date de intrare

Fișierul de intrare natatie.in conține:

- pe prima linie două numere naturale N și M, cu semnificația din enunț;
- pe a doua linie N numere naturale, reprezentând vitezele rațelor (măsurate în metri pe secundă);
- pe a treia linie N numere naturale, reprezentând nivelurile de rezistență ale rațelor;
- pe a patra linie M numere naturale, în ordine strict crescătoare, reprezentând distanțele balizelor.

## Date de ieșire

Pe prima linie a fișierului de ieșire natatie.out se va afișa un singur număr, reprezentând durata minimă a cursei (măsurată în secunde). Răspunsul este considerat corect dacă valoarea absolută a diferenței dintre durata reală și cea afișată este ≤ 10^(-3).

Pe următoarele linii ale fișierului de ieșire se vor afișa toate informatiile despre un culoar, sub forma: Duck X on lane Y: t=z secunde

## Restricții

- $1 \le M \le N \le 3 \cdot 10^3$
- $1 \le v_i, r_i \le 10^9$, $1 \le i \le N$
- $1 \le d_j \le 10^9$, $1 \le j \le M$
- $d_j < d_{j+1}$, $1 \le j < M$
- Timpul în care o rață cu viteză v parcurge distanța d este d/v.

# Cerințe de proiectare: duck_taskrunner

Se va extinde proiectul de la seminar 1, **taskrunner**, derivand noi task-uri astfel incat sa rezolve problema Natatie (Noul proiect se va numi **duck_taskrunner)**

- Principii de proiectare: se vor respecta principiile de proiectare SOLID (vezi cursul 1)
- Șabloane de proiectare: Singleton, Factory, Decorator (sem 1+2)

SUGESTII DE PROIECTARE: NU AVEM sugestii pt lab1 - Sunteti incurajati să veniți cu propria proiectare pentru rezolvarea problemei, care să țină cont, desigur, de principiile de proiectare orientată obiect, învatate pana acum (Vezi SOLID design principles).

Nu vă fie teamă să vă jucati cu clasele, veniți cu propria abstractizare și fiți creativi și originali.

## Livrabile: **incarcate pe git https://classroom.github.com/a/oqRSTCM9**

- Saptamana 3: Diagrama UML realizata in StarUML (https://staruml.io/) cu proiectarea problemei Natatie
- Saptamana 4: Implementarea Java, bazata pe diagrama creata anterior

## Restricții de implementare:

- Nu se vor folosi colectii generice predefinite, se va lucra cu array simplu
- Nu se va folosi Java8 features, nici operații pe stream-uri (Stream)

Exemplu: 5 rate și 3 culoare (L1, L2, L3)

Natatie.in

5 3
4 2
5 2
5 5
3 5
2 7
3 6 10

Obs. Exemplu de mai jos foloseste **metoda cautarii binare** pt det timpului minim, T, căutat in intervalul [0 10] secunde (timpul maxim – rata cu viteza cea mai mica pe culoarul cel mai lung).

| Test | T | L1 eligibile | L2 eligibile | L3 eligibile | Fezabil? | Cum alegem (în ordine L1 → L2 → L3) |
|---|---|---|---|---|---|---|
| 1 | 5 | R1,R2,R3,R4,R5 | R1,R2,R3,R4 | R1,R2,R3 | ✅ | **L1:** prima rață din listă care încape = **R1** (1.5 ≤ 5) → marcăm R1 folosită. **L2:** următoarea disponibilă care încape = **R2** (2.4 ≤ 5). **L3:** următoarea = **R3** (4.0 ≤ 5). |
| 2 | 2.5 | R1,R2,R3,R4 | R2,R3 | — | ❌ | **L1:** R1 (1.5 ≤ 2.5). **L2:** R2 (2.4 ≤ 2.5). **L3:** următoarea R3 are 4.0 > 2.5 → nimeni nu încape ⇒ **eșec**. |
| 3 | 3.75 | R1,R2,R3,R4,R5 | R1,R2,R3 | — | ❌ | **L1:** R1 (1.5 ≤ 3.75). **L2:** R2 (2.4 ≤ 3.75). **L3:** R3 are 4.0 > 3.75 → nimeni nu încape ⇒ **eșec**. |
| 4 | 4.375 | R1,R2,R3,R4,R5 | R1,R2,R3,R4 | R2,R3 | ✅ | **L1:** R1 (ok). **L2:** R2 (ok). **L3:** următoarea disponibilă e **R3** (4.0 ≤ 4.375) ⇒ **reușită**. |
| 5 | 4.0625 | R1,R2,R3,R4,R5 | R1,R2,R3 | R2,R3 | ✅ | **L1:** R1 (ok). **L2:** R2 (ok). **L3:** R3 (4.0 ≤ 4.0625) ⇒ **reușită**. |
| 6 | 3.90625 | R1,R2,R3,R4,R5 | R1,R2,R3 | — | ❌ | **L1:** R1 (ok). **L2:** R2 (ok). **L3:** R3 are 4.0 > 3.90625 → nimeni nu încape ⇒ **eșec**. |

**Ouput posibil**, folosind strategia de afisare LIFO (de la rata cea mai rezistenta - > la cea mai putin rezistenta)

Best time: 4.000 s

Lane 1 (d=3) <- Duck 1 (v=4.00, st=2)

Lane 2 (d=6) <- Duck 2 (v=5.00, st=2)

Lane 3 (d=10) <- Duck 3 (v=5.00, st=5)

Duck 3 on lane 3: t=4.000 s

Task executed at: 13:54

Duck 2 on lane 2: t=2.400 s

Task executed at: 13:54

Duck 1 on lane 1: t=1.500 s

Task executed at: 13:54

## Barem de notare:

3p puncte diagrama UML

5p implementare

- 1p punct citirea datelor
- 2p puncte proiectarea OOP – SE CORDA 2 PT FARA depunctare, daca proiectarea urmeaza paradigma orientata obiect (clase si o arhitectura modulara minimalista)
- 1p Clean code, java standards
- 1p punct det timpului minim

1p conformitatea cu diagrama a codului scris in Java

# Enunț seminar: task_runner

1. Definiți clasa **abstractă Task** avand atributele: taskID(String), descriere(String) si metodele: un constructor cu parametri, set/get, execute() (metoda abstracta), toString() si metodele equals - hashCode; De ce trebuie sa fie clasa Task abstracta?

   *Contractul equals - hashcode:* dacă obj1.equals(obj2) atunci obj1. hashCode() == obj2.hashCode().
   *Ce se intampla cand avem o relatie de mostenire intre doua clase si suprascriem equals?* (a=b => b=a ?)

2. Derivați clasa ***MessageTask*** din clasa *Task*, avand atributele *mesaj* (String), *from*(String), *to*(String) si *date* (*LocalDateTime*) și afișează pe ecran, via metoda *execute*, textul mesajului (valoarea atributului mesaj) si data la care a fost creat; (Vezi si *DateTimeFormatter*)

   Clasa MessageTask ar putea fi refactorizata, astfel inca sa incapsuleze un obiect de tipul Message avand atributele: id, subject, body, from, to, date

3. Suplimentar (pt cine nu are frica de clase): Derivați clasa **SortingTask** din Task care sortează un vector de numere întregi si afiseaza vectorul sortat, via metoda execute().SortingTask permite sortarea unui vector conform unei strategii. Se cer doua strategii de sortare – BubbleSort si (QuickSort sau MergeSort). Sugestie: SortingTask incapsuleaza un AbstractSorter ce are metoda sort.

4. Scrieti un program de test care creeaza un vector (array) de 5 task-uri de tipul ***MessageTask*** si le afiseaza pe ecran in urmatorul format:

**Exemplu: id=1|description=Feedback lab1|message=Ai obtinut 9.60|from=Gigi|to=Ana|date=2018-09-27 09:29**

   Observatie: Se va respecta formatul de afisare al datei.

5. Consideram că interfata ***Container*** specifică interfaţa comună pentru colecţii de obiecte Task, în care se pot adăuga şi din care se pot elimina elemente.

```
public interface Container {
   Task remove();
   void add(Task task);
   int size();
   boolean isEmpty();
}
```

   *Creaţi două tipuri de containere concrete:*

   1. **StackContainer** - care implementează, folosind o reprezentare pe un array, o strategie de tip LIFO;
   2. **QueueContainer** - care implementează, folosind o reprezentare pe un array, o strategie de tip FIFO; Suplimentar
   3. Refactorizati clasele **StackContainer** si **QueueContainer** astfel incat sa evitati codul duplicat (bad smell). Vezi refactorizarea „*Extract Superclass*" (Solutia: Create an abstract superclass; make the original classes subclasses of this superclass, vezi cartea: ***Refactoring: Improving the Design of Existing Code by Martin Fowler***). Suplimentar

6. Consideram că interfata ***Factory*** care conţine o metodă *createContainer*, ce primeşte ca parametru o strategie (FIFO sau LIFO) şi care întoarce un container asociat acelei strategii [Factory Method Pattern]. Creați clasa *TaskContainerFactory* care implementează interfața Factory. Creați containere de tipul Stack sau Queue doar prin apeluri ale metodei *createContainer*.

```
public interface Factory {

   Container createContainer(Strategy startegy);

}
```

7. **Implementati clasa** *TaskContainerFactory* care implementeaza interfata Factory, astfel incat sa nu poata exista decat o singura instanta de acest tip. *[Singleton Pattern]*

8. Considerăm interfața

```
public interface TaskRunner {
    void executeOneTask(); //executa un task din colecţia de task-uri de executat
    void executeAll();  //execută toate task-urile din colecţia de task-uri.
    void addTask(Task t); //adaugă un task în colecţia de task-uri de executat
    boolean hasTask(); //verifica daca mai sunt task-ri de executat

}
```

care specifică interfața comună pentru o colecție de taskuri de executat.

9. Creaţi clasa **StrategyTaskRunner** care implementează interfața **TaskRunner** și care conţine:
   - Un atribut privat de tipul Container;
   - Un constructor ce primeşte ca parametru o strategie prin care se specifică în ce ordine se vor executa task-urile (*LIFO* sau *FIFO*);

10. Scrieți un program de test care creeaza un vector de task-ri de tipul MessageTask si le executa, via un obiect de tipul StrategyTaskRunner, folosind strategia specificata ca parametru in linia de comanda. (main(String[] args).

11. Definiti clasa abstractă **AbstractTaskRunner** [Decorator Pattern] care implementează interfața **TaskRunner** si care conține ca și atribut privat o referință la un obiect de tipul Task Runner, referința primită ca parametrul prin intermediul constructorului.

12. Extindeţi clasa AbstractTaskRunner astfel:

    1. PrinterTaskRunner - care afişează un mesaj după execuţia unui task în care se specifică ora la care s-a executat task-ul.
    2. DelayTaskRunner – care execută taskurile cu întârziere; (Suplimentar)

```
try {
Thread.sleep(3000);
} catch (InterruptedException e) {
   e.printStackTrace();
}
```

13. Scrieți un program de test care creeaza un vector de task-ri de tipul MessageTask si le executa, initial via un obiect de tipul StrategyTaskRunner apoi via un obiect de tipul PrinterTaskRunner (decorator), folosind startegia specificata ca parametru in linia de comanda.

14. Scrieți un program de test care creeaza un vector de task-ri de tipul MessageTask si le executa, initial via un obiect de tipul StrategyTaskRunner apoi via un obiect de tipul DelayTaskRunner (decorator) apoi via un obiect de tipul PrinterTaskRunner (decorator), folosind startegia specificata ca parametru in linia de comanda.
15. Creati diagrama de clase. Ce relații intre clase există in diagrama creată?

## Referinte

A se vedea si cursul si sem 1.
Martin Fowler - Refactoring, improving the design of existing code.
[Factory Method Pattern:] https://www.tutorialspoint.com/design_pattern/factory_pattern.htm
[Decorator Pattern:] https://www.tutorialspoint.com/design_pattern/decorator_pattern.htm
[Singleton Pattern] https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm

## Code style

https://google.github.io/styleguide/javaguide.html
sau
https://www.oracle.com/java/technologies/javase/codeconventions-contents.html