**Metode Avansate de Programare – 2021-2022 SEMINAR 1+2 + LABORATOR 2**

**Acest document contine cerintele pt:**

1. **Seminarul 1+2. (Eventual completari cu alte exercitii)**

* Obiective: Familiarizarea cu limbajul Java, recapitulare concepte OOP, mecanisme OOP, principii OOP, sabloane de proiectare (Factory Method, Decorator, Singleton, Template Method), relatii intre clase(UML)

1. **Laborator 2. (cele care la final au** Cerință laborator) - (deadline Saptamana 4)

*Rezolvați următoarele cerințe:*

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 ?)

1. 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

1. Derivați clasa **SortingTask** din Task care sortează un vector de numere întregi si afiseaza vectorul sortat, via metoda execute(). Cerință laborator 2p

Observatie: Se vor acorda doua puncte doar daca SortingTask permite sortarea unui vector conform unei strategii, altfel se acorda 1p. Se cer doua strategii de sortare – BubbleSort si (QuickSort sau MergeSort). Sugestie: SortingTask incapsuleaza un AbstractSorter ce are metoda sort.

1. 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.

1. Consideram că interfata ***Container*** specificǎ interfaţa comunǎ pentru colecţii de obiecte Task, în care se pot adǎuga şi din care se pot elimina elemente.

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

*Creaţi douǎ tipuri de containere concrete:*

1. **StackContainer**  - care implementeazǎ, folosind o reprezentare pe un array, o strategie de tip [LIFO](http://en.wikipedia.org/wiki/LIFO_(computing));

2. **QueueContainer** - care implementeazǎ, folosind o reprezentare pe un array, o strategie de tip [FIFO](http://en.wikipedia.org/wiki/FIFO); Cerință laborator 2p

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***). Cerință laborator 1p

1. Considerăm interfaţa *Factory* care conţine o metodǎ *createContainer*, ce primeşte ca parametru o strategie (FIFO sau LIFO) şi care întoarce un container asociat acelei strategii [[Factory Method Pattern](http://en.wikipedia.org/wiki/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);  
}

1. **Implementati clasa** *TaskContainerFactory* care implementeaza interfata Factory, astfel incat sa nu poata exista decat o singura instanta de acest tip. *[*[Singleton Pattern](https://en.wikipedia.org/wiki/Singleton_pattern)*]* (Cerință laborator 1p) + discutie in timpul seminarului.
2. Considerăm interfața

**public interface** TaskRunner {  
 **void** executeOneTask(); //executa un task din colecţia de task-uri de executat   
 **void** executeAll(); //executǎ toate task-urile din colecţia de task-uri.  
 **void** addTask(Task t); //adaugǎ 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.

1. 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 specificǎ în ce ordine se vor executa task-urile (*LIFO* sau *FIFO*);

1. 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).
2. Definiti clasa abstractă **AbstractTaskRunner** [[Decorator Pattern]](https://en.wikipedia.org/wiki/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.
3. Extindeţi clasa AbstractTaskRunner astfel:
4. PrinterTaskRunner - care afişeazǎ un mesaj dupǎ execuţia unui task în care se specificǎ ora la care s-a executat task-ul.
5. DelayTaskRunner – care execută taskurile cu întârziere; (Cerință laborator 1p)

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

1. 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.
2. 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. (Cerință de laborator 1p)
3. Creati diagrama de clase. Ce relații intre clase există in diagrama creată? (Cerință de laborator 1p)

***Alte referinte:***

A se vedea si cursul 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>