Metode Avansate de Programare – 2018-2019 - LABORATOR 2

DEADLINE: Săptămâna 3

Rezolvați cerințele marcate cu rosu, Cerință laborator, completând Seminarul 1 si 2.

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

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*)
   Observatie: Constructorul clasei MessageTask are o lista mare de parametri. Ce solutii aveti?

3. 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.

4. Scrieti un program de test care creaza 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 ArrayList<Task>, o strategie de tip LIFO;
   2. **`QueueContainer`** - care implementează, folosind o reprezentare pe un ArrayList<Task>, o strategie de tip 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

6. Considerăm interfaţa *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 IFactory. Creați containere de tipul Stack sau Queue doar prin apeluri ale metodei *createContainer*.

```
public interface Factory {
    Container createContainer(Strategy startegy);
}
```

7. **Implementati clasa** *TaskContainerFactory* astfel incat sa nu poata exista decat o singura instanta de acest tip. *[*Singleton Pattern*]* (Cerință laborator 1p) + discutie in timpul seminarului.

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 creaza un vector de task-ri de tipul MessageTask si le executa, via un obiect de tipul `StrategyTaskRunner,` folosind startegia 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 Abstract`TaskRunner` 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; (Cerință laborator 1p)

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

13. Scrieți un program de test care creaza 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 creaza 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)
15. Creati diagrama de clase. Ce relații intre clase există in diagrama creată? (Cerință de laborator 1p)
16. Optional. Are legatura problema discutata la Seminarul 1 si 2 cu sablonul de proiectare Command?

Alte referinte:

A se vedea si cursul 1.

[1]Martin Fowler - Refactoring, improving the design of existing code.
[2][Factory Method Pattern:] https://www.tutorialspoint.com/design_pattern/factory_pattern.htm
[3][Decorator Pattern:] https://www.tutorialspoint.com/design_pattern/decorator_pattern.htm
[4][Singleton Pattern] https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm