Obiective: Familiarizarea cu limbajul Java, concepte OOP, mecanisme OOP , principii OOP, sabloane de proiectare (Factory, Decorator, Singleton)

Rezolvați următoarele cerințe:

I. Definiți clasa **abstractă** `Task` avand atributele: taskID(int), descriere(String) si metodele: un constructor cu parametrii, set/get, execute() (metoda abstracta), toString();
Obs: Din clasa Task vom deriva task-ri concrete, de exemplu FindReplaceTask, SortingTask, MessageTask etc.

II. Derivați clasa *MessageTask* din clasa *Task*, care încapsulează un mesaj (String) și afișează mesajul pe ecran, via metoda execute.

III. Cerință laborator: Derivați clasa SortingTask din Task, care sortează (folosind o strategie de sortare la alegere) un vector de numere întregi, via metoda execute() (afiseaza vectorul sortat).

IV. Cerință laborator: clasa `TaskArray` care incapsuleaza un vector de taskuri si a carei interfata ofera metodele get(int pos), add(Task elem), add(int index, Task elem) ….delete(pos), size();

V. Consideram că interfata Container

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

specifică interfaţa comună pentru colecţii de obiecte Task, în care se pot adăuga şi din care se pot elimina elemente. Creaţi două tipuri de containere:

1. `StackContainer` - care implementează, folosind o reprezentare pe un array, o strategie de tip LIFO;
Cerință laborator: Refactorizati clasa `StackContainer`, definită la seminar, astfel încât să reutilizați clasa `TaskArray` din cerința IV. (Clasa `StackContainer` va încapsula un obiect de tipul `TaskArray`).

2. Cerință laborator: `QueueContainer` - care implementează o strategie de tip FIFO ; Clasa `QueueContainer` va încapsula un obiect de tip TaskArray (variabilă de instanță).
Cerință laborator: `Evitati scrierea codului asemanator in diferite metode ale aplicatiei.`

VI. 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 doar prin apeluri ale metodei *createContainer*.

VII. Cerință laborator: **Evitaţi posibilitatea** instanţierii clasei *TaskContainerFactory* de mai multe ori. *[Singleton Pattern]*

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

VIII. 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.

IX. 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);

X. Definiti clasa abstractă **TaskRunnerDecorator** [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.

XI. Extindeţi clasa `TaskRunnerDecorator` astfel:

- `PrinterTaskRunner` - care afişează un mesaj după execuţia unui task în care se specifică ora la care s-a executat task-ul.
- Cerință laborator`:` DelayTaskRunner – care execută taskurile cu întârziere;

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

XII. 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.

XIII. Cerință de laborator 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.

XIV. Cerință de laborator: Creati diagrama de clase. Ce relații intre clase există in diagrama creată?

Referinte:

A se vedea si cursul 1.

Alte exemple:

[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