
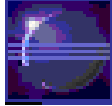


| | | |
|---|---|---|
|  | <p align="center">TP</p> <p align="center">Collections génériques en JAVA</p> |  |
|---|---|---|

| Objectifs | Temps alloué | Outils |
|--|--------------|---------|
| <ul style="list-style-type: none"> - Apprendre à manipuler les collections génériques | 3h00 | Eclipse |

Exercice 1 : Compréhension de l'architecture des tâches

L'architecture des tâches est donnée à la figure 1 où le détail des classes **TacheElementaire** et **TacheComplexe** n'est pas donné.

Une tâche est caractérisée par un nom et un coût. Une tâche est soit une tâche élémentaire, soit une tâche complexe qui est alors composée de sous-tâches.

Il est ainsi possible d'ajouter une sous-tâche à une tâche complexe, ajouter(Tache) ou de supprimer une sous-tâche, supprimer(Tache). Le coût d'une tâche complexe est la somme des coûts des tâches qui la composent.

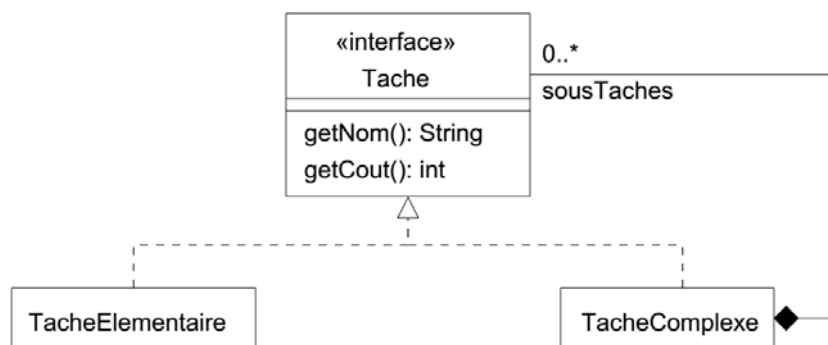


FIG. 1 – Architecture des tâches hiérarchiques

- 1) Écrire en Java la classe **TacheElementaire** qui est une réalisation de l'interface Tache.

```
public interface Tache {  
    /** Obtenir le nom de la tâche. */  
    String getNom();  
  
    /** Obtenir le coût de la tâche. */  
    int get Cout();  
}
```

Définition d'une tâche complexe

Nous nous intéressons maintenant à la classe TacheComplexe, en particulier à sa relation avec l'interface Tache. Une tâche complexe est composée d'un nombre quelconque de tâches. On décide d'utiliser l'interface java.util.Collection pour stocker les sous-tâches.

- 2) Écrire en Java la classe TacheComplexe.
- 3) Écrire un programme qui crée des tâches et des tâches complexes et qui affiche leur coût total.

Exercice 2 : Statistiques sur les Etudiants

Partie 1

On souhaite gérer un ensemble d'étudiants. Chaque étudiant dispose d':

- Une matricule;
- Un nom;
- Une liste de Notes (de taille indéfinie).

Les notes de chaque étudiant seront stockées dans une **ArrayList**.

Une méthode **addNote** permettant d'ajouter une note à l'étudiant sera définie.

Une classe Note permettra de contenir pour chaque cours, l'intitulé du cours ainsi que la note obtenue.

```
public class Note {  
    private String NomCours;  
    private double note;
```

Partie 2

On souhaite mettre en place une classe capable de réaliser des statistiques sur une collection d'objets, comme par exemple, des Etudiants, des Notes, ... Cette classe, qui sera nommée Stats, pourra ainsi calculer le maximum, le minimum et la moyenne d'une collection d'objets.

Toutes les classes qui peuvent faire l'objet de statistiques implémenteront une interface IStatisticable, qui est décrite comme suit:

```
public interface IStatisticable {  
    abstract double getAvg();  
    abstract double getMin();  
    abstract double getMax();  
  
}
```

Pour un Etudiant, on choisit de prendre :

- La moyenne de ses notes comme avg de l'Etudiant.
- Sa meilleure note comme Max
- Sa moins bonne note comme Min

```

public class Etudiant implements IStatisticable{
    private String matricule;
    private String Nom;
    private ArrayList<Note> Notes;

    //constructeurs, getters and setters
    /// .....
    void addNote(double note)
    {
    }

    @Override
    public double getAvg() {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public double getMin() {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public double getMax() {
        // TODO Auto-generated method stub
        return 0;
    }
}

```

La classe Stats sera ensuite utilisée et donnera pour :

- Chaque étudiant :
 - Sa moyenne ;
 - Sa meilleure note ;
 - Sa moins bonne note ;
- Chaque groupe d'étudiants (qui est présenté par un vecteur d'étudiants)
 - La moyenne du groupe ;
 - Le meilleur étudiant ;
 - Le moins bon étudiant.

```

public class Stats {
    private ArrayList<Etudiant> etudiantsList;

    //constructeurs

    public void getClassDetails()
    {

    }

    public double getClassAvg()
    {
        return 0;
    }

    public Etudiant getBestStudent()
    {
        return null;
    }

    public Etudiant getWorstStudent()
    {
        return null;
    }
}

```

Partie 3

- a- On souhaite pouvoir classer la liste d'étudiants suivant la matricule. Pour ce faire, on implémentera l'interface Comparable dans la classe Etudiant. La méthode compareTo devra donc être définie dans la classe Etudiant.

Une fois cela réalisé, on triera la liste d'étudiants à l'aide de la méthode **Collections.sort**.

- b- On souhaite également pouvoir trier la liste d'étudiants par moyenne et par nom. Dans ce but, deux nouvelles classes (CompareMoyenne et CompareNom) seront créées et implémenteront l'interface Comparator. Ces classes devront donc chacune définir une méthode compare prenant comme arguments les deux objets à comparer et réalisant un traitement similaire à celui de CompareTo dans l'exercice précédent.