Licence L3 M I 2 E

TD Pile
Classes et Interfaces
(d'après M-J Bellosta et B. Bérard)

Réalisation du type abstrait de données *Pile*

Le type Pile est une structure algorithmique permettant de gérer une collection d'objets avec la politique suivante d'accès aux éléments :

premier entré => dernier sorti

La classe *Stack* représente la réalisation du type Pile. Elle est définie comme une suite chaînée d'éléments appelés *maillon*. La classe *Maillon* est la réalisation Java d'un maillon. Ces deux classes sont dans l'espace de nom *tdr.td_pile*.

La classe *Maillon*

La classe *Maillon* est la réalisation Java d'un maillon. Elle comprend un attribut *element* qui est l'élément mis dans la pile et un attribut *next* qui référence le maillon suivant.
La classe *Maillon* n'est visible que par les classes de son espace de noms. Elle est définie dans le même fichier que la classe *Stack*. La classe *Maillon* a deux constructeurs :

1. un constructeur qui admet un élément passé en argument, dans ce cas le maillon suivant est établi à *null* lors de la déclaration des attributs.
2. un constructeur qui admet un élément passé en argument et le maillon suivant.

La classe *Maillon* a trois méthodes ayant une visibilité par défaut :

Object getElement()	retourne l'élément associé au maillon courant
Object next()	retourne le maillon suivant
boolean hasNext()	retourne *true* s'il existe un maillon suivant, false sinon

Les méthodes de réalisation sont :

- *void setNext(Maillon next)* affecte l'attribut *next* avec le maillon passée en argument
- *void setElement(Object o)* affecte l'attribut *element* avec l'objet *o* passé en argument

La classe *Stack*

La classe *Stack* est publique, elle a un attribut *top* qui référence le premier maillon. Elle a deux constructeurs :

1. un constructeur vide dans ce cas l'élément sommet est égal à *null* et la pile est vide et de taille zéro.
2. un constructeur qui admet un élément passé en argument de type *Object*, dans ce cas, il y a création d'un maillon (instance de la classe *Maillon*) avec l'élément suivant à *null*.

La classe *Stack* admet les quatre méthodes publiques suivantes :

void push(Object object)	place au sommet de la pile l'objet passé en argument
Object pop()	retourne l'élément qui se trouve au sommet de la pile et place l'élément suivant au sommet de la pile.
Object first()	accède au premier élément de la pile sans modifier la pile
boolean isEmpty()	retourne *true* si la pile est vide

Méthodes de réalisation de la classe *Stack* :

- *void setTop(Object o)* établit le sommet de la pile avec le maillon passé en argument
- *Maillon getTop()* retourne le sommet de la pile
- *Maillon next()* retourne le maillon suivant

Questions:

Les deux classes sont définies dans l'espace de noms : *tdr.td_pile.question123*,

1. Compléter le deuxième constructeur de *Stack* et le deuxième constructeur de *Maillon*
2. Dans *Stack*, compléter les méthodes *setTop(Object top)* et *next()* et préciser la visibilité des méthodes *setTop(Object)* , *getTop()* et *next()*; dans *Maillon*, compléter la méthode *toString()* et préciser la visibilité des méthodes *setNext(Maillon), setElement(Object)* et *toString()*.
3. Compléter la méthode *main* de la classe *Stack* pour obtenir les sorties du cadre ci-après. Les messages suivent le format suivant :
   *System.out.println("*** invocation de méthode *** résultat attendu :: "+ invocation de méthode);*
   Par exemple
   L'exécution de *System.out.println("*** p.isEmpty() *** must be true :: "+ p.isEmpty());*
   a donné :
   *** p.isEmpty() *** must be true :: true*
   L'exécution finale est :

```
*** p.isEmpty() *** must be true :: true
*** {titi, toto} *** :: {titi,toto }
*** p.isEmpty() *** must be false :: false
*** {3, titi, toto} *** :: {3,titi,toto }
*** p.pop() *** must be : 3 :: 3
*** p.pop() *** must be : titi :: titi
*** {toto} :: {toto }
*** p.first() *** must be : toto :: toto
*** p *** must be : {toto} :: {toto }
*** p *** must be :  {} :: {}
```

4. La classe *Stack* réalise l'interface *StackAbility* qui comprend les quatre opérations suivantes:
   void push(Object object), Object pop(), Object first() et boolean isEmpty()
   La classe *Maillon* réalise l'interface non publique *MaillonAbility* qui comprend les trois opérations
   Object getElement(), *Maillon next()* et *boolean hasNext()*.
   Définir les interfaces et effectuer les modifications requises dans le code.

5. On souhaite ajouter à la classe *Stack* la fonctionalité *int size()* qui retourne le nombre d'éléments contenus dans la pile. Ajoutez dans la classe *Stack* et/ou dans la classe *Maillon* les méthodes permettant de réaliser cette fonctionnalité de manière récursive.
   A titre d'exemple, voici l'exécution d'un programme de test :

```
*** p.isEmpty() *** must be true :: true
*** p.size() *** must be 0 :: 0
*** {titi, toto} *** :: {titi,toto }
*** p.size() *** must be 2 :: 2
```

6. Les classes *Stack* et *Maillon* réalisent l'interface *SizeAbility* comprenant l'opération *int size()* définie précédemment. Quelles sont les modifications à apporter dans le code ?
7. Ajouter la sémantique d'égalité de deux piles et modifier le programme principal pour tester l'égalité de 2 piles : 2 piles sont égales si leurs maillons de même rang sont égaux ; 2 maillons sont égaux si leur attribut *element* sont égaux.