ensiie
école nationale supérieure d'informatique pour l'industrie et l'entreprise

TD ILO n° 1

Exercice 1 : Points 2D

Ecrivez la classe Point2D décrite en cours contenant les coordonnées x, y, un compteur d'instances ainsi qu'une constante de classe epsilon (10^{-6}).

a) Décrivez d'abord les constructeurs de la classe :

- Un constructeur valué avec deux valeurs flottantes pour les coordonnées
- Un constructeur par défaut
- Un constructeur de copie

Réutilisez autant que faire ce peut le constructeur valué dans le constructeur par défaut et dans le constructeur de copie.

b) Où peut être incrémenté et décrémenté le compteur d'instances ?

c) Décrivez ensuite les méthodes d'accès aux attributs.

d) Redéfinissez la méthode d'affichage « toString » pour afficher une chaine de caractères représentant un point 2D : « x = … y = … »

```
public String toString() {…}
```

e) Ecrivez une méthode « déplace » permettant de déplacer un point existant de dx et dy.

f) Déterminez deux types de méthodes pour calculer la distance entre deux points.

g) Déterminez de la même manière, une ou deux méthodes pour tester l'égalité entre deux points à une distance epsilon près.

Exercice 2 : Composition

Le but de cet exercice est de vous faire utiliser des classes Java existantes pour réaliser vos propres classes.
Nous nous proposons ici de construire plusieurs implémentations d'un même interface Pile<E>.

a) Spécifiez une interface décrivant les comportements d'une pile : Pile<E>

- Quels sont ses membres ?

b) Implémentation d'une classe Pile en utilisant la classe standard « Vector »

Ecrivez l'implémentation de la pile définie précédemment en utilisant comme conteneur des données de la pile, la classe Vector<E> : VPile<E>. Commencez par un constructeur par défaut et un constructeur de copie (à partir d'une Pile<E>)

La classe **Vector** implémente un tableau de taille variable d'objets, elle fait partie de la distribution standard de Java (voir un extrait de la documentation en annexe).

c) Implémentation d'une classe Pile en utilisant une classe Table<E>

Nous avons à notre disposition la classe « Table<E> » qui est en fait un tableau d'éléments de type « E » et qui possède l'interface suivante :

```
public class Table<E>
{
  public Table(int nbCases){…}
  public E getValAt (int i){…}
  public void setValAt(int i, E e){…}
  public int size(){…}
 …
}
```

Initialise la « Table » courante avec « nbCases » emplacements

Renvoie le contenu de la $i^{ème}$ case de la Table

Affecte le contenu de la $i^{ème}$ case de la Table avec l'objet o.

Renvoie le nombre de cases dans la table.

Ecrivez à nouveau l'implémentation d'une pile utilisant la classe Table<E> ci-dessus : TPile<E>. Commencez par un constructeur par valué et un constructeur de copie (à partir d'une Pile<E>).

Exercice 3 : Point 3D

A partir du point 2D, implémentez un point 3D par héritage. Utilisez autant que faire se peut, les méthodes de la classe mère pour ne pas tout réécrire dans la classe fille.

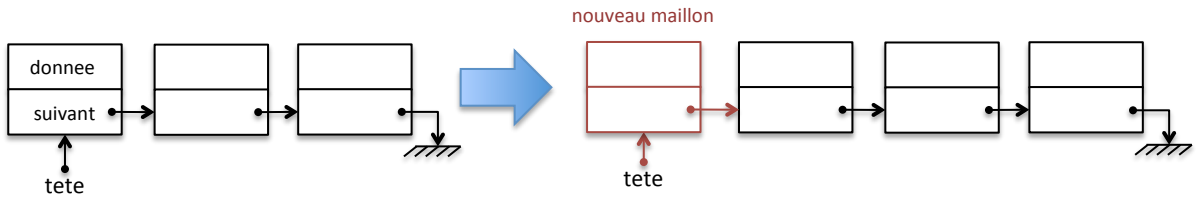
Exercice 4 : Listes chaînées

On cherche à réaliser une liste simplement chaînée composée de maillons contenant une donnée de type « E » ainsi qu'une référence vers un maillon suivant (s'il y en a un). La liste chaînée doit implémenter l'interface `Iterable<E>` qui définit une factory méthode (`Iterator<E> iterator()`) permettant d'obtenir un itérateur sur la liste. Les itérateurs (`Iterator<E>` en java) définissent 3 méthodes :

- **`boolean hasNext()`** : qui permet de savoir s'il reste des éléments à itérer.
- **`E next()`** : qui permet de renvoyer le prochain élément et de passer au suivant.
- **`void remove()`** : qui permet de retirer des éléments itérés l'élément qui vient d'être renvoyé par « `next()` ».

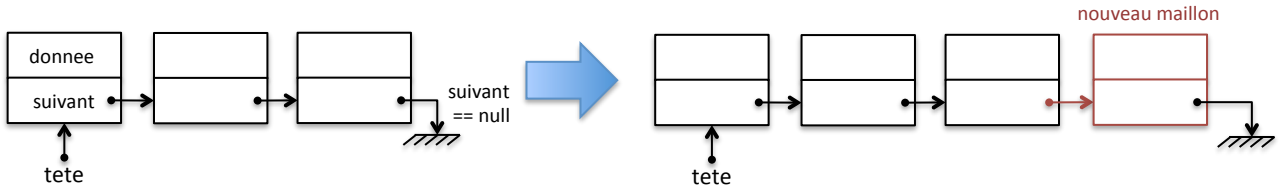
Implémentez la liste chaînée elle même avec les opérations suivantes, en considérant que vous disposez d'un itérateur sur la liste fournit par la méthode `Iterator<E> iterator()` :

- **`void insert(E elt)`** : ajoute l'élément elt en tête de liste en levant une `NullPointerException` si l'élément à insérer est null.



- **`boolean insert(E elt, int index)`** : Ajoute l'élément elt à la (index+1)^{ième} place. Renvoie vrai si l'élément a pu être inséré à la bonne place, ou false si l'élément n'a pas pu être inséré ou si celui ci était null.

- **`void add(E elt)`** : ajoute l'élément elt à la fin de la liste en levant une `NullPointerException` si l'élément à insérer est null.



- **`boolean remove(E elt)`** : supprime la première occurrence de l'élément elt dans la liste s'il est présent.

- **`boolean removeAll(E elt)`** : supprime toutes les occurrences de l'élément elt dans la liste.
- **`void clear()`** : supprime tous les éléments de la liste.
- **`boolean empty()`** : renvoie vrai si la liste ne contient aucun élément.
- **`int size()`** : renvoie le nombre d'éléments actuellement dans la liste.
- **`boolean equals(Object o)`** : renvoie vrai si o est un `Iterable<E>` de même longueur et contenant les mêmes éléments (dans le même ordre).
- **`int hashCode()`** : renvoie le hash code de la liste d'après les hashcodes de ses éléments (voir l'algorithme donné en cours).
- **`String toString()`** : renvoie une chaine de caractère représentant la liste sous la forme « [elt->elt->elt] ».
- **`Iterator<E> iterator()`** : On considèrera que cette méthode (avec l'itérateur sous-jacent) est fournie.

Annexe

Documentation de la classe Vector

```
public class Vector<E> extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

Constructors

`Vector()` Constructs an empty vector so that its internal data array has size `10` and its standard capacity increment is zero.
`Vector(Collection<? extends E> c)` Constructs a vector containing the elements of the specified collection, in the order they are returned by the collection's iterator.
`Vector(int initialCapacity)` Constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero.
`Vector(int initialCapacity, int capacityIncrement)` Constructs an empty vector with the specified initial capacity and capacity increment.

Methods

`boolean`	`add(E e)` Appends the specified element to the end of this Vector.
`void`	`add(int index, E element)` Inserts the specified element at the specified position in this Vector.
	⋮
`void`	`addElement(E obj)` Adds the specified component to the end of this vector, increasing its size by one.
	⋮
`void`	`clear()` Removes all of the elements from this Vector.

	⋮
E	elementAt(int index) Returns the component at the specified index.
	⋮
boolean	equals(Object o) Compares the specified Object with this Vector for equality.
E	firstElement() Returns the first component (the item at index 0) of this vector.
	⋮
E	get(int index) Returns the element at the specified position in this Vector.
int	hashCode() Returns the hash code value for this Vector.
	⋮
int	indexOf(Object o) Returns the index of the first occurrence of the specified element in this vector, or -1 if this vector does not contain the element.
int	indexOf(Object o, int index) Returns the index of the first occurrence of the specified element in this vector, searching forwards from index, or returns -1 if the element is not found.
void	insertElementAt(E obj, int index) Inserts the specified object as a component in this vector at the specified index.
boolean	isEmpty() Tests if this vector has no components.
	⋮
E	lastElement() Returns the last component of the vector.
int	lastIndexOf(Object o) Returns the index of the last occurrence of the specified element in this vector, or -1 if this vector does not contain the element.
int	lastIndexOf(Object o, int index) Returns the index of the last occurrence of the specified element in this vector, searching backwards from index, or returns -1 if the element is not found.
	⋮
E	remove(int index) Removes the element at the specified position in this Vector.
boolean	remove(Object o) Removes the first occurrence of the specified element in this Vector If the Vector does not contain the element, it is unchanged.
	⋮
boolean	removeElement(Object obj) Removes the first (lowest-indexed) occurrence of the argument from this vector.
void	removeElementAt(int index) Deletes the component at the specified index.
	⋮
E	set(int index, E element) Replaces the element at the specified position in this Vector with the specified element.
void	setElementAt(E obj, int index) Sets the component at the specified index of this vector to be the specified object.
int	size() Returns the number of components in this vector.
	⋮