PC – TD/TP 3

Vous devez terminer l'implémentation des programmes en dehors des TP

Exercice 1 : Tableaux (en TD et en TP)

Cet exercice consiste à écrire une classe `RangeArray` simulant des tableaux d'entiers dont les indices des cases sont compris entre deux valeurs entières `indexMin` et `indexMax` (avec `indexMin <= indexMax`). Le code suivant est un exemple d'utilisation de la classe `RangeArray` :

```
int indexMin = -3;
int indexMax = 1;
RangeArray squares = new RangeArray(indexMin, indexMax);
for (int index =  squares.getIndexMin();
         index <= squares.getIndexMax();
         index++)
   squares.set(index, index*index);
for (int index =  squares.getIndexMin();
         index <= squares.getIndexMax();
         index++)
   System.out.println(index + " -> " + squares.get(index));
System.out.println(squares.rangeSize());
```

Il doit produire la sortie suivante :

```
-3 -> 9
-2 -> 4
-1 -> 1
0 -> 0
1 -> 1
5
```

Dans la suite de l'exercice, nous supposerons que les indices fournis aux méthodes `get` et `set` sont toujours compris entre `indexMin` et `indexMax`. Nous supposerons également que le premier argument passé au constructeur de `RangeArray` est toujours inférieur ou égal au second. Notez que `indexMin` et `indexMax` peuvent être négatifs.

1. Proposez une implémentation de la classe `RangeArray`. Pour stocker les entiers à l'intérieur de l'instance, vous devez allouer un tableau de la taille strictement nécessaire.

2. Modifiez la classe que vous avez écrite pour la rendre générique de sorte que le code suivant compile et fonctionne :

```
RangeArray<String> habitations = new RangeArray<String>(3, 5);
habitations.set(3, "Maison");
habitations.set(4, "Immeuble");
habitations.set(5, "Hutte");
for (int index =  habitations.getIndexMin();
         index <= habitations.getIndexMax();
         index++)
   System.out.print(habitations.get(index).length() + " ");
```

   Il doit produire la sortie suivante : `6 8 5`

3. Modifiez la classe `RangeArray<T>` de sorte qu'elle implémente l'interface `Iterable<T>`. Le code suivant doit compiler et fonctionner :

```
RangeArray<String> habitations = new RangeArray<String>(3, 5);
habitations.set(3, "Maison");
habitations.set(4, "Immeuble");
habitations.set(5, "Hutte");
for (String habitation : habitations)
  System.out.print(habitation + " ");
```

   Il doit produire la sortie suivante : `Maison Immeuble Hutte`

Exercice 2 : Itérateur sur des grilles (en TD et en TP)

On considère le code suivant manipulant une classe générique `Grid<T>` :

```
String[][] elements = { { "A", "C", "E", "G" },
                        { "B", "D", "F", "H" } };
Grid<String> grid = new Grid<String>(elements);
for (int line = 0; line < grid.nbLines(); line++)
  for (int column = 0; column < grid.nbColumns(); column++) {
    String element = grid.get(line, column);
    System.out.print(element+" ");
  }
System.out.println();
```

Le code de la classe `Grid<T>` est donné ci-dessous. Il suppose que la grille est non vide et qu'elle est rectangulaire, c'est-à-dire que toutes les lignes ont le même nombre de colonnes.

```
public class Grid<T> implements Iterable<T> {
  private T[][] elements;

  public Grid(T[][] elements) { this.elements = elements; }

  public T get(int line, int column) {
    return elements[line][column];
  }

  public int nbLines()   { return elements.length; }
  public int nbColumns() { return elements[0].length; }

  public Iterator<T> iterator() {
    return new GridIterator<T>(this);
  }
}
```

1. Qu'affiche le code donné en début d'exercice ?

2. Donnez l'équivalent du code suivant en utilisant explicitement les méthodes `T next()` et `boolean hasNext()` de l'interface `Iterator<T>` (c'est-à-dire sans la syntaxe spéciale du `for`) :

   ```
   for (String element : grid) System.out.print(element+" ");
   System.out.println();
   ```

3. Implémentez la classe `GridIterator<T>` de sorte que le code donné à la question 2 produise la même sortie que le code donné en début d'exercice. Vous devez :
   a. préciser la déclaration de la classe ;
   b. préciser quels sont les attributs de la classe ;
   c. Implémenter les méthodes `T next()` et `boolean hasNext()`.

   On ne demande pas d'implémenter la méthode `void remove()` de l'interface `Iterator<T>`.

Exercice 3 : Vecteur paramétré (uniquement en TP)

1. Modifiez le vecteur d'objets que vous avez écrit dans les TD précédents en un vecteur générique `Vector<T>` où `T` représente le type des objets que l'utilisateur peut placer dans le vecteur. Le constructeur prend en paramètre la capacité initiale du vecteur. Vous devez implémentez les méthodes suivantes :
   — `void ensureCapacity(int capacity)`
   — `void resize(int size)`
   — `int size()`
   — `boolean isEmpty()`
   — `void add(T element)`
   — `void set(int index, T element)`
   — `T get(int index)`
2. Faites en sorte que la classe `Vector<T>` implémente l'interface `Iterable<T>`. L'itérateur retourné par la méthode `iterator()` est une instance de la classe `VectorIterator` externe à la classe `Vector`.
3. Modifiez la classe `VectorIterator` enfin d'en faire une classe interne et privée à la classe `Vector`.
4. Ajoutez une méthode `void addAll(Vector<T> elements)` à la classe `Vector` ?
5. Ajoutez une méthode `boolean isSorted()` qui retourne `true` si les éléments présents dans le vecteur sont triés. Cette méthode suppose que les objets présents dans le vecteur implémentent l'interface `Comparable`. La méthode `compareTo` de cette interface est utilisée pour comparer les éléments deux à deux. Que se passe-t-il si des éléments n'implémentent pas l'interface `Comparable` ?
6. Ajoutez une méthode `boolean isSorted(Comparator<T> comparator)` qui fournit la même fonctionnalité que la méthode précédente en prenant, ici, un comparateur en paramètre.
7. Ajoutez une méthode statique `boolean isSorted(Vector<E> vector)` qui retourne `true` si les éléments du vecteur `vector` sont triés.