3. Feuille de TP

semaine 16 :

Agenda Iterable

On a la classe agenda suivante :

On voudrait pouvoir parcourir l'agenda de la manière suivante :

1. Pourquoi le code précédent ne compile-t-il pas ? Quel message d'erreur obtenez vous ?
2. Comment modifier la classe Agenda pour que le code précédent fonctionne ?
3. Testez votre code.

Ensemble

Dans cet exercice, on va écrire notre propre classe **Set**, pas de façon très efficace, mais au moins correcte.

La classe **Set** contient de nombreuses méthodes, mais la bibliothèque java nous vient en aide avec une **classe abstraite** `AbstractSet` qui va nous permettre de ne coder que le strict nécessaire

- la méthode `public Iterator<T> iterator()`,
- la méthode `public boolean add(T element)`,
- la méthode `public int size()`.

EnsembleInefficace.java

Ici on commence par la version la plus inefficace mais la plus simple.

On va implémenter un set en utilisant un attribut de type `ArrayList`. Les éléments présents dans l'ArrayList sont les éléments du Set.

1. Complétez le code suivant en implémentant **AbstractSet**, en ajoutant toutes les méthodes nécessaires

```
public class EnsembleInefficace<T>          ...                                 {

  private                           listeInterne;

  public EnsembleInefficace() {




  }

  public int size() {



  }


  public Iterator<T> iterator() {
```

(suite sur la page suivante)

(suite de la page précédente)

```
    }

    @Override
    public boolean add(T elem) {




    }

    // .....
}
```

2. Créez un exécutable pour tester votre classe. (Ajoutez plusieurs fois le même élément).
3. Quelle est la complexité de votre méthode `add` ?

La méthode contains de votre classe (que vous pouvez trouver dans la classe AbstractCollection qui est étendue par AbstractSet) est la suivante :

```
public boolean contains(Object o) {
  Iterator<E> e = iterator();
  if (o == null) {
      while (e.hasNext())
          if (e.next() == null)
              return true;
  }
  else {
      while (e.hasNext())
          if (o.equals(e.next()))
              return true;
          }
      return false;
  }
```

4. Quelle est la complexité de cette méthode ?
5. Selon vous, quelle est la complexité des méthodes `add` et `contains` de `HashSet` ?

Vers mieux

Pour obtenir de meilleures complexités pour les méthodes du set, on va utiliser l'idée suivante.

Notre set contient un attribut de type `ArrayList<T>`, *gigantesque* et quasiment vide (i.e. contenant principalement des cases à `null`). Quelques cases ne sont pas `null`, ce sont les éléments de notre ensemble.

Pour déterminer la position d'un élément dans notre tableau, on utilise son hashCode : l'élément de hashCode `h` sera en position `h` (modulo la taille du tableau).

1. Avec une telle implémentation, comment savoir si un élément donnée appartient à notre ensemble ? (on ne demande pas de code, simplement une explication en français)
2. Avec une telle implémentation, comment ajouter un élément à notre ensemble ?
3. Complétez le code suivant en ajoutant d'éventuelles nécessaires méthodes

```
public class EnsembleMieux<T>          ...                                    {

  private                           listeInterne;

  private int nbElements ;

  public EnsembleMieux() {
  // Remplissez listeInterne de 10000 cases à null



  }

  public int size() {




  }


  public Iterator<T> iterator() {
  // On le fera dans la suite.
     return null;
  }

  @Override
  public boolean add(T elem) {




  }

  @Override
  public boolean contains(Object o) {



  }

  // ...
 }
```

Il nous reste maintenant à écrire la méthode `public Iterator<T> iterator()`

4. La première étape va consister à définir une classe **Iterateur<T>** permettant d'itérer sur une `List<T>` dont certains éléments sont `null`, en 'sautant' les éléments null. Cette classe aura 3 attributs privés : une **position** (la position actuelle de l'itérateur), une **lastPosition** (la position précédente de l'itérateur) et une **List<T>**.

- Complétez :

```
public class Iterateur<T> implements Iterator<T>{
     private List<T> valeurs;
     private int position;
     private int lastPosition;

     public Iterateur(List<T> array) {
```

(suite sur la page suivante)

(suite de la page précédente)

```
      // TODO
    }

    @Override
    public T next() throws NoSuchElementException {
      // TODO
    }

    @Override
    public boolean hasNext(){
      // TODO
    }

    @Override
    public void remove(){
      // TODO
    }
}
```

5. Ajoutez la méthode iterator à la classe `EnsembleMieux`
6. Testez le tout sur l'exécutable suivant

```
public class Executable{
      public static void main(String[] args){
              Set<Integer> e = new EnsembleMieux<>();
              e.add(5);
              e.add(5);
              e.add(7);
              e.add(null);
              e.add(5);
              e.add(7);
              e.add(6);
              e.remove(6);
              for (Integer val : e) {
                 System.out.print(val + " ");
              }
    }
}
```

doit afficher

```
[5 7]
```

7. Qu'est-ce qui ne va pas dans notre classe `EnsembleMieux` ? On verra par la suite comment y remédier. Mais qu'est-ce qui est bien dans votre classe `EnsembleMieux` par rapport à l'implémentation de l'exercice précédent ?