Agenda Iterable

On a la classe **Agenda** suivante :

```
import java.util.Collections;
import java.util.ArrayList;
import java.util.List;

class PasDeDisponibiliteException extends Exception{
    PasDeDisponibiliteException(String message) {super(message);}
}
class AgendaVideException extends Exception {
    AgendaVideException(String message) {super(message);}
}

class Agenda
{
    private List<RendezVous> contenu;
    public Agenda(){
        this.contenu = new ArrayList<>();
    }

    public RendezVous getPremier() throws AgendaVideException {
        if(this.contenu.size() == 0)
            throw new AgendaVideException("Agenda sans rendez vous");
        return Collections.min(this.contenu);
    }

    public void ajoute (RendezVous v) throws PasDeDisponibiliteException{
        for(RendezVous existant : this.contenu)
            if(v.intersecte(existant))
                throw new PasDeDisponibiliteException("Déjà un rendez vous à cette
heure");
        this.contenu.add(v);
    }
    @Override
    public String toString()
    {
        return this.contenu.toString();
    }
}
```

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

```java
import java.text.ParseException ;
class Executable{

    public static void main(String[] args)
    {
        Agenda a = new Agenda();
        try
        {
            a.ajoute(new RendezVous("10:05/03/2017", "11:05/03/2017"));
            a.ajoute(new RendezVous("11:05/03/2017", "12:05/03/2017"));
            a.ajoute(new RendezVous("9:05/03/2017", "10:05/03/2017"));
            a.ajoute(new RendezVous("8:05/03/2017", "9:05/03/2017"));
            a.ajoute(new RendezVous("12:05/03/2017", "13:05/03/2017"));



            for(RendezVous rdv: a){
                System.out.println(rdv);
            }
        }
        catch(ParseException e)
        {
                System.out.println(e);
        }
        catch(PasDeDisponibiliteException e)
        {
            System.out.println(e);
        }
    }
}
```

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 ?

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 définit 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 possible mais la plus simple.

On va implémenter un Set en utilisant un attribut de type `List`. Les éléments présents dans la List sont les éléments du Set.

1. Complétez le code suivant en implémentant **AbstractSet**.

```
public class EnsembleInefficace<T>
{

  private                          listeInterne;

  public EnsembleInefficace(){



  }

  public int size(){


  }

  public Iterator<T> iterator(){



  }

  public boolean add(T elem){



  }
}
```

2. Créez un exécutable pour tester votre classe (Note : pensez à déclarer une instance de Set).
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) dont le code est le suivant :

```
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 de l'ensemble, on va utiliser l'idée suivante.

Notre ensemble contient un attribut de type `List<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 donné 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 :

```
public class VersMieux<T>                                                    {

  private                            listeInterne;

  private int nbElements ;

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



  }

  public int size(){



  }

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

  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 de valeur null. Cette classe aura trois 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) {
       // TODO
     }
     @Override
     public T next(){
       // TODO
     }
     @Override
     public boolean hasNext(){
       // TODO
     }
     @Override
     public void remove(){
       // TODO
     }
 }
```

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

```
public class Executable{
      public static void main(String[] args){
             Set<Integer> e = new VersMieux<>();
             e.add(5);
             e.add(5);
             e.add(7);
             e.add(null);
             e.add(5);
             e.add(7);
             e.add(6);
             e.remove(6);
             System.out.println(e);
     }
}
```

qui doit afficher

```
[5, 7]
```

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