

mar 10, 17 11:55

## Makefile

Page 1/2

```

1  # Executables
2  OSTYPE = $(shell uname -s)
3  JAVAC = javac
4  JAVA = java
5  # A2PS = a2ps-utf8
6  A2PS = a2ps
7  GHOSTVIEW = gv
8  DOCP = javadoc
9  ARCH = zip
10 #ARCH = tar zcvf
11 PS2PDF = ps2pdf -dPDFX=true -sPAPERSIZE=a4
12 DATE = $(shell date +%Y-%m-%d)
13 # Options de compilation
14 #CFLAGS = -verbose
15 CFLAGS =
16 ifeq ($(findstring Darwin,$(OSTYPE)),Darwin)
17     # MacOS systems
18     CLASSPATH=./opt/local/share/java/junit.jar:opt/local/share/java/hamcrest-core.jar
19 else
20     # Other systems
21     CLASSPATH=.
22 endif
23
24 JAVAOPTIONS = --verbose
25
26 PROJECT=Ensembles
27 # nom du fichier d'impression
28 OUTPUT = $(PROJECT)
29 # nom du r  pertoire ou se situera la documentation
30 DOC = doc
31 # lien vers la doc en ligne du JDK
32 WEBLINK = "http://docs.oracle.com/javase/6/docs/api/"
33 # lien vers la doc locale du JDK
34 LOCALLINK = "file:///Users/davidroussel/Documents/docs/java/api/"
35 # nom de l'archive
36 ARCHIVE = $(PROJECT)
37 # format de l'archive pour la sauvegarde
38 ARCHFMT = zip
39 #ARCHFMT = tar
40 # R  pertoire source
41 SRC = src
42 # R  pertoire bin
43 BIN = bin
44 # R  pertoire Listings
45 LISTDIR = listings
46 # R  pertoire Archives
47 ARCHDIR = archives
48 # R  pertoire Figures
49 FIGDIR = graphics
50 # noms des fichiers sources
51 MAIN = RunAllTests
52 SOURCES = $(foreach name, $(MAIN), $(SRC)/$(name).java) \
53 $(SRC)/listes/package-info.java \
54 $(SRC)/listes/Iliste.java \
55 $(SRC)/listes/Liste.java \
56 $(SRC)/tableaux/package-info.java \
57 $(SRC)/tableaux/Tableau.java \
58 $(SRC)/ensembles/package-info.java \
59 $(SRC)/ensembles/Ensemble.java \
60 $(SRC)/ensembles/EnsembleGenerique.java \
61 $(SRC)/ensembles/EnsembleVector.java \
62 $(SRC)/ensembles/EnsembleListe.java \
63 $(SRC)/ensembles/EnsembleTableau.java \
64 $(SRC)/ensembles/EnsembleFactory.java \
65 $(SRC)/ensembles/EnsembleTri.java \
66 $(SRC)/ensembles/EnsembleTriVector.java \
67 $(SRC)/ensembles/EnsembleTriListe.java \
68 $(SRC)/ensembles/EnsembleTriTableau.java \
69 $(SRC)/ensembles/EnsembleTriGenerique.java \
70 $(SRC)/ensembles/EnsembleTriVector2.java \
71 $(SRC)/ensembles/EnsembleTriListe2.java \
72 $(SRC)/ensembles/EnsembleTriTableau2.java \
73 $(SRC)/ensembles/EnsembleTriFactory.java \
74 $(SRC)/tests/package-info.java \
75 $(SRC)/tests/AllTests.java \
76 $(SRC)/tests/AllEnsembleTest.java \
77 $(SRC)/tests/ListeTest.java \
78 $(SRC)/tests/TableauTest.java \
79 $(SRC)/tests/EnsembleTriTest.java
80
81 OTHER = Sujet.pdf
82
83 .PHONY : doc ps
84
85 # Les cibles de compilation
86 # pour g  n  rer l'application
87 all : $(foreach name, $(MAIN), $(BIN)/$(name).class)
88
89 #r  gle de compilation g  n  rique
90 $(BIN)/%.class : $(SRC)/%.java

```

Vendredi mars 10, 2017

Makefile

mar 10, 17 11:55

## Makefile

Page 2/2

```

91 $(JAVAC) -sourcepath $(SRC) -classpath $(BIN):$(CLASSPATH) -d $(BIN) $(CFLAGS) $<
92
93 # Edition des sources $(EDITOR) doit   tre une variable d'environnement
94 edit :
95     $(EDITOR) $(SOURCES) Makefile &
96
97 # nettoyer le r  pertoire
98 clean :
99     find bin/ -type f -name "*.class" -exec rm -f {} \;
100     rm -rf *~ $(DOC)/* $(LISTDIR)/*
101
102 realclean : clean
103     rm -f $(ARCHDIR)/*.$(ARCHFMT)
104
105 # g  n  rer le listing
106 $(LISTDIR) :
107     mkdir $(LISTDIR)
108
109 ps : $(LISTDIR)
110     $(A2PS) -2 --file-align=fill --line-numbers=1 --font-size=10 \
111     --chars-per-line=100 --tabsize=4 --pretty-print \
112     --highlight-level=heavy --prologue="gray" \
113     -o$(LISTDIR)/$(OUTPUT).ps Makefile $(SOURCES)
114
115 pdf : ps
116     $(PS2PDF) $(LISTDIR)/$(OUTPUT).ps $(LISTDIR)/$(OUTPUT).pdf
117
118 # g  n  rer le listing lisible pour G  rard
119 bigps :
120     $(A2PS) -1 --file-align=fill --line-numbers=1 --font-size=10 \
121     --chars-per-line=100 --tabsize=4 --pretty-print \
122     --highlight-level=heavy --prologue="gray" \
123     -o$(LISTDIR)/$(OUTPUT).ps Makefile $(SOURCES)
124
125 bigpdf : bigps
126     $(PS2PDF) $(LISTDIR)/$(OUTPUT).ps $(LISTDIR)/$(OUTPUT).pdf
127
128 # voir le listing
129 preview : ps
130     $(GHOSTVIEW) $(LISTDIR)/$(OUTPUT); rm -f $(LISTDIR)/$(OUTPUT) $(LISTDIR)/$(OUTPUT)~
131
132 # g  n  rer la doc avec javadoc
133 doc : $(SOURCES)
134     $(DOCP) -private -d $(DOC) -author -link $(LOCALLINK) $(SOURCES)
135     # $(DOCP) -private -d $(DOC) -author -linkoffline $(WEBLINK) $(LOCALLINK) $(SOURCES)
136
137 # g  n  rer une archive de sauvegarde
138 $(ARCHDIR) :
139     mkdir $(ARCHDIR)
140
141 archive : pdf $(ARCHDIR)
142     $(ARCH) $(ARCHDIR)/$(ARCHIVE)-$(DATE).$(ARCHFMT) $(SOURCES) $(LISTDIR)/*.pdf $(OTHER) $(BIN) Mak
143     efile
144
145 # ex  cution des programmes de test
146 run : all
147     $(foreach name, $(MAIN), $(JAVA) -classpath $(BIN):$(CLASSPATH) $(name) $(JAVAOPTIONS) )

```

1/33

sep 30, 15 16:46

**RunAllTests.java**

Page 1/1

```

1 import org.junit.runner.JUnitCore;
2 import org.junit.runner.Result;
3 import org.junit.runner.notification.Failure;
4
5 import tests.AllTests;
6
7 /**
8  * Exécution de tous les tests du package "tests"
9  * @author davidroussel
10 */
11 public class RunAllTests
12 {
13     /**
14      * Programme principal de lancement des tests
15      * @param args non utilisés
16      */
17     public static void main(String[] args)
18     {
19         System.out.println("Test des ensembles");
20
21         Result result = JUnitCore.runClasses(AllTests.class);
22
23         int failureCount = result.getFailureCount();
24
25         if (failureCount == 0)
26         {
27             System.out.println("Every thing went fine");
28         }
29         else
30         {
31             for (Failure failure : result.getFailures())
32             {
33                 System.err.println(failure);
34             }
35         }
36     }
37 }

```

oct 20, 14 17:22

**package-info.java**

Page 1/1

```

1 /**
2  * Package contenant l'implémentation des listes simplement chaînées dâfinies
3  * dans l'interface {@link listes.IListe} et implémentâes dans la classe
4  * {@link listes.Liste}
5  */
6 package listes;

```

nov 04, 15 18:02

lListe.java

Page 1/2

```

1 package listes;
2 import java.util.Iterator;
3
4 /**
5  * Interface d'une liste g n rique d' l ments.
6  *
7  * @note On consid rera que la liste ne peut pas contenir d'elt null
8  * @author David Roussel
9  * @param <E> le type des  l ments de la liste.
10 */
11 public interface lListe<E> extends Iterable<E>
12 {
13     /**
14      * Ajout d'un  l ment en fin de liste
15      *
16      * @param elt l' l ment   ajouter en fin de liste
17      * @throws NullPointerException si l'on tente d'ajouter un  l ment null
18      */
19     public abstract void add(E elt) throws NullPointerException;
20
21     /**
22      * Insertion d'un  l ment en t te de liste
23      *
24      * @param elt l' l ment   ajouter en t te de liste
25      * @throws NullPointerException si l'on tente d'ins rer un  l ment null
26      */
27     public abstract void insert(E elt) throws NullPointerException;
28
29     /**
30      * Insertion d'un  l ment   la (index+1) me place
31      *
32      * @param elt l' l ment   ins rer
33      * @param index l'index de l' l ment   ins rer
34      * @return true si l' l ment a pu  tre ins r    l'index voulu, false sinon
35      *         ou si l' l ment   ins rer  tait null
36      */
37     public abstract boolean insert(E elt, int index);
38
39     /**
40      * Suppression de la premi re occurrence de l' l ment e
41      * (en utilisant l'it rateur)
42      *
43      * @param elt l' l ment   rechercher et   supprimer.
44      * @return true si l' l ment a  t  trouv  et supprim  de la liste
45      * @note doit fonctionner m me si e est null
46      */
47     public default boolean remove(E elt)
48     {
49         /*
50          * TODO Compl ter ...
51          */
52         return false;
53     }
54
55     /**
56      * Suppression de toutes les instances de e dans la liste
57      * (en utilisant l'it rateur)
58      *
59      * @param elt l' l ment   supprimer
60      * @return true si au moins un  l ment a  t  supprim 
61      * @note doit fonctionner m me si e est null
62      */
63     public default boolean removeAll(E elt)
64     {
65         boolean result = false;
66         /*
67          * TODO Compl ter ...
68          */
69         return result;
70     }
71
72     /**
73      * Nombre d' l ments dans la liste
74      * (en utilisant l'it rateur)
75      *
76      * @return le nombre d' l ments actuellement dans la liste
77      */
78     public default int size()
79     {
80         int count = 0;
81         /*
82          * TODO Compl ter ...
83          */
84         return count;
85     }
86
87     /**
88      * Effacement de la liste.
89      * (en utilisant l'it rateur)
90

```

Vendredi mars 10, 2017

src/listes/lListe.java

nov 04, 15 18:02

lListe.java

Page 2/2

```

91     */
92     public default void clear()
93     {
94         /*
95          * TODO Compl ter ...
96          */
97     }
98
99     /**
100      * Test de liste vide
101      *
102      * @return true si la liste est vide, false sinon
103      */
104     public default boolean empty()
105     {
106         /*
107          * TODO Remplacer par l'impl mentation ...
108          */
109         return false;
110     }
111
112     /**
113      * Test d' galit  au sens du contenu de la liste
114      *
115      * @param o la liste dont on doit tester le contenu
116      * @return true si o est une liste. que tous les maillons des deux listes
117      *         sont identiques (au sens du  gals de chacun des maillons). dans
118      *         le m me ordre, et que les deux listes ont la m me longueur. false
119      *         sinon
120      * @note On serait tent  d'en faire une "default method" dans la mesure o 
121      *         l'on ne peut n'utiliser que l'it rateur pour parcourir les  l ments de
122      *         la liste MAIS les m thodes par d faut n'ont pas le droit de
123      *         surcharger les m thodes de la superclasse Object.
124      */
125     @Override
126     public abstract boolean equals(Object o);
127
128     /**
129      * hashCode d'une liste
130      *
131      * @return le hashCode de la liste
132      * @note On serait tent  d'en faire une "default method" dans la mesure o 
133      *         l'on ne peut n'utiliser que l'it rateur pour parcourir les  l ments de
134      *         la liste MAIS les m thodes par d faut n'ont pas le droit de
135      *         surcharger les m thodes de la superclasse Object.
136      */
137     @Override
138     public abstract int hashCode();
139
140     /**
141      * Repr sentation de la chaine sous forme de chaine de caract re.
142      *
143      * @return une chaine de caract re repr sentant la liste chain e
144      * @note On serait tent  d'en faire une "default method" dans la mesure o 
145      *         l'on ne peut n'utiliser que l'it rateur pour parcourir les  l ments de
146      *         la liste MAIS les m thodes par d faut n'ont pas le droit de
147      *         surcharger les m thodes de la superclasse Object.
148      */
149     @Override
150     public abstract String toString();
151
152     /**
153      * Obtention d'un it rateur pour parcourir la liste : <code>
154      * Liste<Type> l = new Liste<Type>();
155      * ...
156      * for (Iterator<Type> it = l.iterator(); it.hasNext(); )
157      * {
158      *     ... it.next() ...
159      * }
160      * ou bien
161      * for (Type elt : l)
162      * {
163      *     ... elt ...
164      * }
165      * </code>
166      *
167      * @return un nouvel it rateur sur la liste
168      * @see {@link Iterable#iterator()}
169      */
170     @Override
171     public abstract Iterator<E> iterator();
172 }

```

3/33

oct 20, 14 17:22

## package-info.java

Page 1/1

```

1 /**
2  * Package contenant la classe {@link tableaux.Tableau} : tableau de donn  es de
3  * taille variable
4  */
5 package tableaux;

```

nov 20, 14 14:52

## Tableau.java

Page 1/5

```

1 package tableaux;
2
3 import java.util.Collection;
4 import java.util.Iterator;
5 import java.util.NoSuchElementException;
6
7 /**
8  * Tableau de donn  es
9  *
10  * @author davidroussel
11  * @param <E> le type des donn  es stock  es dans le tableau
12  */
13 public class Tableau<E> implements Iterable<E>
14 {
15     /**
16      * Le tableau de donn  es
17      */
18     protected E[] table;
19
20     /**
21      * Nombre d'  l  ments actuellement dans le tableau. Et index du prochain
22      *   l  ment    ins  rer
23      */
24     protected int size;
25
26     /**
27      * Nombre de cases max du tableau
28      */
29     protected int capacity;
30
31     /**
32      * Nombres de cases initiales par d  faut du tableau de donn  es. Et nombre de
33      * cases    rajouter en cas de manque de cases
34      */
35     public static final int INCREMENT = 5;
36
37     /**
38      * constructeur par d  faut d'un tableau de donn  es
39      */
40     @SuppressWarnings("unchecked")
41     public Tableau()
42     {
43         table = (E[]) new Object[INCREMENT];
44         size = 0;
45     }
46
47     /**
48      * constructeur de copie    partir d'un autre {@link Iterable}
49      *
50      * @param elements l'it  rable dont on doit copier les   l  ments
51      */
52     public Tableau(Iterable<E> elements)
53     {
54         this();
55         for (E elt : elements)
56         {
57             ajouter(elt);
58         }
59     }
60
61     /**
62      * Nombre d'  l  ments actuellement dans le tableau
63      *
64      * @return Le nombre d'  l  ments actuellement dans le tableau
65      */
66     public int taille()
67     {
68         return size;
69     }
70
71     /**
72      * Nombre d'  l  ments maximum (actuellement) dans le tableau
73      *
74      * @return le nombre de l'  l  ments amx dans le tableau actuellement
75      */
76     public int capacite()
77     {
78         return capacity;
79     }
80
81     /**
82      * Ajout d'un element    la fin du tableau
83      *
84      * @param element l'  l  ment    ins  rer
85      */
86     public void ajouter(E element)
87     {
88         if (size    capacity)
89         {
90             // ajouterCapacite(Math.max(INCREMENT, (size - capacity) + 1));

```

nov 20, 14 14:52

## Tableau.java

Page 2/5

```

91     int scl = (size - capacity) + 1;
92     ajouterCapacite((INCREMENT ≥ scl ? INCREMENT : scl));
93 }
94 table[size] = element;
95 size++;
96 }
97
98 /**
99  * Ajout de nbCases au tableau
100  *
101  * @param nbCases nombre de cases à ajouter.
102  */
103 protected void ajouterCapacite(int nbCases)
104 {
105     if (nbCases > 0)
106     {
107         capacity += nbCases;
108         @SuppressWarnings("unchecked")
109         E[] newTable = (E[]) new Object[capacity];
110         for (int i = 0; i < size; i++)
111         {
112             newTable[i] = table[i];
113             table[i] = null; // avoid weak references
114         }
115         table = newTable;
116     }
117 }
118
119 /**
120  * Retrait de la première occurrence d'un élément
121  *
122  * @param element l'élément à retirer du tableau
123  * @return true si l'élément a été trouvé et retiré
124  */
125 public boolean retrait(E element)
126 {
127     for (Iterator<E> it = iterator(); it.hasNext(); )
128     {
129         if (it.next().equals(element))
130         {
131             it.remove();
132             return true;
133         }
134     }
135     return false;
136 }
137
138 /**
139  * Effacement de tous les éléments du tableau
140  */
141 public void efface()
142 {
143     for (Iterator<E> it = iterator(); it.hasNext(); )
144     {
145         it.next();
146         it.remove();
147     }
148 }
149
150 /**
151  * Insertion d'un élément en début de tableau
152  *
153  * @param element l'élément à insérer
154  */
155 public void insertElement(E element)
156 {
157     try
158     {
159         insertElement(element, 0);
160     }
161     catch (IndexOutOfBoundsException ioobe)
162     {
163         System.err.println("Tableau:insertElement: " + ioobe);
164     }
165 }
166
167 /**
168  * Insertion d'un élément à la place index
169  *
170  * @param element l'élément à insérer dans le tableau
171  * @param index l'index où insérer l'élément
172  * @throws IndexOutOfBoundsException si l'index où insérer l'élément est
173  *         invalide
174  */
175 public void insertElement(E element, int index)
176     throws IndexOutOfBoundsException
177 {
178     if ((index ≤ size) ∧ (index ≥ 0))
179     {

```

nov 20, 14 14:52

## Tableau.java

Page 3/5

```

181     if (index == size)
182     {
183         ajouter(element);
184     }
185     else // index >= 0 & < size
186     {
187         if ((size + 1) ≥ capacity)
188         {
189             ajouterCapacite(INCREMENT);
190         }
191         // décalage des éléments
192         for (int i = size; i > index; i--)
193         {
194             table[i] = table[i - 1];
195         }
196         table[index] = element;
197         size++;
198     }
199 }
200
201 else
202 {
203     throw new IndexOutOfBoundsException("Invalid Index : "
204         + Integer.toString(index));
205 }
206 }
207
208 /**
209  * Factory method fournissant un itérateur sur le tableau
210  *
211  * @return un nouvel itérateur sur le tableau
212  */
213 @Override
214 public Iterator<E> iterator()
215 {
216     return new TabIterator<E>();
217 }
218
219 /**
220  * Test d'égalité avec un autre objet.
221  * @return true ssi l'objet est un {@link Tableau} et qu'il contient
222  * les mêmes éléments dans le même ordre.
223  * @see java.lang.Object#equals(java.lang.Object)
224  */
225 @Override
226 public boolean equals(Object obj)
227 {
228     if (obj == null)
229     {
230         return false;
231     }
232
233     if (obj == this)
234     {
235         return true;
236     }
237
238     if (getClass().isInstance(obj))
239     {
240         Tableau<?> tab = (Tableau<?>) obj;
241         Iterator<E> it1 = iterator();
242         Iterator<?> it2 = tab.iterator();
243
244         for (; it1.hasNext() ∧ it2.hasNext(); )
245         {
246             if (!it1.next().equals(it2.next()))
247             {
248                 return false;
249             }
250         }
251
252         return !it1.hasNext() ∧ !it2.hasNext();
253     }
254     else
255     {
256         return false;
257     }
258 }
259
260 /**
261  * Code de hachage d'un tableau.
262  * Le code de hachage est compatible avec celui fourni par toute {@link Collection}
263  * contenant les mêmes éléments dans le même ordre.
264  * @return le code de hachage résultant des éléments du Tableau
265  * @see java.lang.Object#hashCode()
266  */
267 @Override
268 public int hashCode()
269 {

```

nov 20, 14 14:52

## Tableau.java

Page 4/5

```

271 {
272     final int prime = 31;
273     int result = 1;
274     for (E elt : this)
275     {
276         result = (prime * result) + (elt == null ? 0 : elt.hashCode());
277     }
278     return result;
279 }
280
281 /**
282  * Chaîne de caractères représentant les éléments du tableau ainsi que sa
283  * taille et sa capacité courante
284  * @return une nouvelle chaîne de caractères représentant le Tableau
285  * @see java.lang.Object.toString()
286  */
287 @Override
288 public String toString()
289 {
290     StringBuilder sb = new StringBuilder();
291
292     sb.append("[");
293     for (Iterator<E> it = iterator(); it.hasNext(); )
294     {
295         sb.append(it.next().toString());
296         if (it.hasNext())
297         {
298             sb.append(", ");
299         }
300     }
301     sb.append("]");
302     sb.append(Integer.toString(size));
303     sb.append(", ");
304     sb.append(Integer.toString(capacity));
305     sb.append(")");
306
307     return new String(sb);
308 }
309
310 /**
311  * Itérateur sur un {link Tableau}
312  *
313  * @author davidroussel
314  * @param <F> le type des éléments à itérer
315  */
316 private class TabIterator<F> implements Iterator<F>
317 {
318     /**
319      * L'index courant de l'itérateur. index de l'élément courant dans le
320      * tableau
321      */
322     private int index;
323
324     /**
325      * Indique si next vient d'être appelé ce qui permet (éventuellement)
326      * d'appeler remove.
327      */
328     private boolean nextCalled;
329
330     /**
331      * Constructeur par défaut d'un itérateur sur un tableau
332      */
333     public TabIterator()
334     {
335         index = 0;
336         nextCalled = false;
337     }
338
339     /**
340      * Clause de continuation
341      *
342      * @return true si l'itérateur peut encore itérer (utiliser la méthode
343      * {link #next()})
344      */
345     @Override
346     public boolean hasNext()
347     {
348         return index < size;
349     }
350
351     /**
352      * Incrémentation de l'itérateur
353      * @return la donnée correspondant à la position courante de l'itérateur
354      * @throws NoSuchElementException si l'itérateur ne peut plus itérer,
355      * lorsque celui-ci a déjà atteint le dernier élément à itérer
356      */
357     @Override
358     public F next() throws NoSuchElementException
359     {
360         if (hasNext())

```

nov 20, 14 14:52

## Tableau.java

Page 5/5

```

361 {
362     @SuppressWarnings("unchecked")
363     F element = (F) table[index];
364     index++;
365     nextCalled = true;
366     return element;
367 }
368 else
369 {
370     throw new NoSuchElementException();
371 }
372 }
373
374 /**
375  * Suppression du dernier élément renvoyé par {link #next()}.
376  * Attention. remove ne peut être appelé qu'après avoir appelé
377  * {link #next()}.
378  *
379  * @post l'élément précédent l'élément courant de l'itérateur a été
380  * supprimé.
381  */
382 @Override
383 public void remove() throws IllegalStateException
384 {
385     if (nextCalled) // index >= 1
386     {
387         for (int i = index - 1; i < (size - 1); i++)
388         {
389             table[i] = table[i + 1];
390         }
391         size--;
392         index--;
393         nextCalled = false;
394     }
395     else
396     {
397         throw new IllegalStateException("Next not called yet");
398     }
399 }
400 }
401 }

```

oct 20, 14 17:21

## package-info.java

Page 1/1

```

1 /**
2  * Package contenant la d  finition d'un {@link ensembles.Ensemble} comme   tant
3  * une collection (a priori non ordonn  e, m  me si le conteneur sous-jacent peut
4  *   tre ordonn  ). {@link ensembles.EnsembleGenerique} fournit une impl  mentation
5  * partielle des ensembles sans conna  tre encore le conteneur sous-jacent (qui
6  * peut   tre un {@link java.util.Vector}, ou bien une {@link listes.Liste}, ou
7  * encore un {@link tableaux.Tableau}. {@link ensembles.EnsembleGenerique}
8  * n'impl  mente pas les op  rations :
9  * <ul>
10 * <li>d'ajout {@link ensembles.EnsembleGenerique#ajout(Object)} puisqu'elle est
11 * sp  cifique au conteneur sous-jacent</li>
12 * <li>de construction d'un it  rateur
13 * {@link ensembles.EnsembleGenerique#iterator()} puisqu'elle est aussi
14 * sp  cifique au conteneur sous-jacent</li>
15 * <li>les op  rations ensembliste comme
16 * {@link ensembles.Ensemble#union(Ensemble)},
17 * {@link ensembles.Ensemble#intersection(Ensemble)},
18 * {@link ensembles.Ensemble#complement(Ensemble)} et
19 * {@link ensembles.Ensemble#difference(Ensemble)} de part le fait qu'elle est
20 * une classe abstraite et ne peut donc pas "cr  er" l'ensemble r  sultat de
21 * l'op  ration ensembliste. En revanche elle propose une impl  mentation bas  e
22 * sur les m  thodes de classes dans lesquelles l'ensemble r  sultat est d  j   cr    
23 * (par une des classes filles)</li>
24 * </ul>
25 * {@link ensembles.EnsembleGenerique} impl  mente donc
26 * <ul>
27 * <li>{@link ensembles.Ensemble#union(Ensemble, Ensemble, Ensemble)}</li>
28 * <li>{@link ensembles.Ensemble#intersection(Ensemble, Ensemble, Ensemble)}</li>
29 * <li>{@link ensembles.Ensemble#complement(Ensemble, Ensemble, Ensemble)}</li>
30 * <li>{@link ensembles.Ensemble#difference(Ensemble, Ensemble, Ensemble)}</li>
31 * </ul>
32 */
33 package ensembles;

```

nov 04, 15 17:54

## Ensemble.java

Page 1/4

```

1 package ensembles;
2
3 import java.util.Iterator;
4
5 /**
6  * Interface d  finissant un ensemble comme une collection non tri  e d'  l  ments
7  * sans doublons. Le fait que les   l  ments sont consid  r  s comme non tri  s
8  * impliquera que la comparaison de deux ensembles ne devra pas prendre en
9  * compte l'ordre (apparent) des   l  ments.
10 *
11 * @author davidroussel
12 */
13 public interface Ensemble<E> extends Iterable<E>
14 {
15     /**
16      * Ajout d'un   l  ment    un ensemble ssi celui ci n'est pas null et qu'il
17      * n'est pas d  j   pr  sent
18      *
19      * @param element l'  l  ment    ajouter    l'ensemble (on consid  rera que l'on
20      * ne peut pas ajouter d'  l  ment null)
21      * @return true si l'  l  ment a pu   tre ajout      l'ensemble, false sinon ou
22      * si l'on a tent   d'ins  rer un   l  ment null (auquel cas il n'est
23      * pas ins  r  )
24      */
25     public abstract boolean ajout(E element);
26
27     /**
28      * Retrait d'un   l  ment de l'ensemble en utilisant le remove de l'it  rateur
29      * fournit par {@link #iterator()}
30      *
31      * @param element l'  l  ment    supprimer de l'ensemble
32      * @return true si l'  l  ment   tait pr  sent dans l'ensemble (au sens de la
33      * comparaison profonde) et qu'il a   t   retir  , false sinon
34      */
35     public default boolean retrait(E element)
36     {
37         /**
38          * TODO Compl  ter ...
39          */
40         return false;
41     }
42
43     /**
44      * Teste si l'ensemble est vide en utilisant l'it  rateur ou bien le
45      * {@link #cardinal}
46      *
47      * @return renvoie true si l'ensemble ne contient aucun   l  ment, false sinon
48      * @see ensembles.Ensemble#estVide()
49      * @note Attention. si l'on utilise cardinal dans estVide, il ne faut pas
50      * utiliser estVide dans cardinal et vice versa.
51      */
52     public default boolean estVide()
53     {
54         /**
55          * TODO Remplacer par l'impl  mentation ...
56          */
57         return false;
58     }
59
60     /**
61      * Test d'appartenance d'un   l  ment    l'ensemble en utilisant l'it  rateur
62      * pour parcourir les   l  ments
63      *
64      * @param element l'  l  ment dont on doit tester l'appartenance
65      * @return true si l'  l  ment est pr  sent dans l'ensemble (au sens de la
66      * comparaison profonde), false sinon
67      */
68     public default boolean contient(E element)
69     {
70         /**
71          * TODO Compl  ter ...
72          */
73         return false;
74     }
75
76     /**
77      * Test si ensemble est un sous-ensemble de l'ensemble courant. C'est    dire
78      * si l'ensemble courant contient tous les   l  ments de l'ensemble pass   en
79      * argument
80      *
81      * @note Si l'ensemble pass   en argument est null il ne sera pas consid  r  
82      * comme contenu.
83      * @param ensemble l'ensemble dont on veut tester s'il est un sous-ensemble
84      * de l'ensemble courant
85      * @return true si ensemble est un sous-ensemble de l'ensemble courant,
86      * false sinon. false si ensemble est null.
87      */
88     public default boolean contient(Ensemble<E> ensemble)
89     {
90

```

nov 04, 15 17:54

## Ensemble.java

Page 2/4

```

91  /*
92   * TODO Compl ter ...
93   */
94
95   return false;
96 }
97
98 /**
99  * Efface tous les  l ments de l'ensemble en utilisant le remove de
100  * l'it rateur fourni par {@link #iterator()}
101  */
102 public default void efface()
103 {
104     /*
105     * TODO Compl ter ...
106     */
107 }
108
109 /**
110  * Taille de l'ensemble en utilisant l'it rateur
111  *
112  * @return le nombre d' l ments dans l'ensemble
113  * @see ensembles.Ensemble#cardinal() Attention : si l'on utilise estVide
114  *      dans cardinal, il ne faut pas utiliser cardinal dans estVide
115  * @note Cette m thode aura int r t    tre r impl ment e dans les classes
116  *      filles qui utilisent des conteneurs pouvant donner leur taille
117  *      directement
118  */
119 public default int cardinal()
120 {
121     int count = 0;
122
123     /*
124     * TODO Compl ter ...
125     */
126
127     return count;
128 }
129
130 /**
131  * Union avec un autre ensemble : (this union ensemble).
132  *
133  * @param ensemble l'autre ensemble avec lequel on veut cr er une union
134  * @return un nouvel ensemble contenant l'union de l'ensemble courant et de
135  *      l'ensemble pass  en argument
136  */
137 public abstract Ensemble<E> union(Ensemble<E> ensemble);
138
139 /**
140  * Impl mentation de classe de l'union de deux ensemble dans un autre
141  * ensemble
142  *
143  * @param ens1 le premier ensemble
144  * @param ens2 le second ensemble
145  * @param res l'ensemble contenant l'union de ens1 et ens2
146  */
147 public static <E> void union(Ensemble<E> ens1, Ensemble<E> ens2, Ensemble<E> res)
148 {
149     /*
150     * TODO Compl ter ...
151     */
152 }
153
154 /**
155  * Intersection avec un autre ensemble : (this inter ensemble).
156  *
157  * @param ensemble l'autre ensemble avec lequel on veut cr er une
158  *      intersection
159  * @return un nouvel ensemble contenant l'intersection de l'ensemble courant
160  *      et de l'ensemble pass  en argument
161  */
162 public abstract Ensemble<E> intersection(Ensemble<E> ensemble);
163
164 /**
165  * Impl mentation de classe de l'intersection de deux ensemble dans un autre
166  * ensemble
167  *
168  * @param ens1 le premier ensemble
169  * @param ens2 le second ensemble
170  * @param res l'ensemble contenant l'intersection de ens1 et ens2
171  */
172 public static <E> void intersection(Ensemble<E> ens1, Ensemble<E> ens2, Ensemble<E> res)
173 {
174     /*
175     * TODO Compl ter ...
176     */
177 }
178
179 /**
180  * Compl ment avec un autre ensemble : (this - ensemble).

```

Vendredi mars 10, 2017

src/ensembles/Ensemble.java

nov 04, 15 17:54

## Ensemble.java

Page 3/4

```

181  *
182  * @param ensemble l'autre ensemble avec lequel on veut cr er le compl ment
183  * @return un nouvel ensemble contenant uniquement les  l ments pr sents
184  *      dans l'ensemble courant mais PAS dans l'ensemble pass  en
185  *      argument
186  */
187 public abstract Ensemble<E> complement(Ensemble<E> ensemble);
188
189 /**
190  * Impl mentation de classe du compl ment de deux ensembles dans un autre
191  * ensemble.
192  *
193  * @param ens1 le premier ensemble
194  * @param ens2 le second ensemble
195  * @param res l'ensemble contenant le compl ment de ens1 - ens2
196  */
197 public static <E> void complement(Ensemble<E> ens1, Ensemble<E> ens2, Ensemble<E> res)
198 {
199     /*
200     * TODO Compl ter ...
201     */
202 }
203
204 /**
205  * Diff rence sym trique avec un autre ensemble : (this delta ensemble).
206  * L'ensemble correspondant   la diff rence sym trique contient les  l ments
207  * qui sont soit dans l'ensemble courant, soit dans l'autre ensemble mais
208  * pas dans les deux ensembles = (this - ensemble) union (ensemble - this)
209  *
210  * @param ensemble l'autre ensemble avec lequel on veut cr er une diff rence
211  *      sym trique
212  * @return un nouvel ensemble contenant la diff rence sym trique de
213  *      l'ensemble courant et de l'ensemble pass  en argument
214  * @see ensembles.Ensemble#difference(ensembles.Ensemble)
215  */
216 public default Ensemble<E> difference(Ensemble<E> ensemble)
217 {
218     /*
219     * TODO Remplacer par l'impl mentation en utilisant
220     * - Soit (A - B)  M-^H  (B - A)
221     * - Soit (A  M-^H  B) - (B  M-^H  A)
222     */
223     return null;
224 }
225
226 /**
227  * Type des  l ments de l'ensemble
228  *
229  * @return une instance de la classe Class repr sentant le type des  l ments
230  *      de l'ensemble si celui ci n'est pas vide, ou bien null si
231  *      l'ensemble est vide.
232  * @note cette m thode sera utile dans l'impl mentation de la m thode
233  *      {@link #equals(Object)} pour d terminer si deux ensembles ont le
234  *      m me type d' l ments
235  * @see ensembles.Ensemble#typeElements()
236  */
237 @SuppressWarnings("unchecked")
238 public default Class<E> typeElements()
239 {
240     Iterator<E> it = iterator();
241     if (it != null)
242     {
243         if (it.hasNext())
244         {
245             return (Class<E>) it.next().getClass();
246         }
247     }
248
249     return null;
250 }
251
252 // -----
253 // M thodes   impl menter d finies dans la classe Object
254 // -----
255
256 /**
257  * Test d' galit  entre deux ensembles
258  *
259  * @param o l'objet   comparer
260  * @return true si l'objet   comparer est un ensemble et qu'il contient les
261  *      m mes  l ments (pas forc ment dans le m me ordre). Si les deux
262  *      ensembles sont vides on consid re qu'ils seront  gaux quel que
263  *      soit leur type de contenu (dans la mesure o  l'on ne veut pas le
264  *      d terminer avec {@link ensembles.Ensemble#typeElements()})
265  * @note une interface ne peut pas impl menter par d faut des m thodes
266  *      surcharg es de la classe Object (celles ci d cendant de l' tat
267  *      interne des objets, ce qui n'est pas le cas d'une interface)
268  */
269 @Override
270 public abstract boolean equals(Object o);

```

8/33



nov 04, 15 17:54

## Ensemble.java

Page 4/4

```

271 /**
272  * Hashcode d'un ensemble. Le HashCode d'un ensemble doit être calculé comme
273  * étant la somme des hascodes de ses éléments afin de ne pas tenir compte
274  * de l'ordre des éléments dans la collection sous-jacente.
275  */
276
277 * @return le hashage d'un ensemble
278 * @note une interface ne peut pas implémenter par défaut des méthodes
279 * surchargées de la classe Object (celles-ci dépendant de l'état
280 * interne des objets, ce qui n'est pas le cas d'une interface)
281 */
282 @Override
283 public abstract int hashCode();
284
285 /**
286  * Affichage des éléments de l'ensemble sous la forme : par exemple pour un
287  * ensemble de 3 elts : "[elt1, elt2, elt3]" où elt n représente le toString
288  * du nième elt.
289  */
290 * @return une chaîne de caractères représentant les éléments de l'ensemble
291 * séparés par des virgules et encadrés par des crochets
292 * @note une interface ne peut pas implémenter par défaut des méthodes
293 * surchargées de la classe Object (celles-ci dépendant de l'état
294 * interne des objets, ce qui n'est pas le cas d'une interface)
295 */
296 @Override
297 public abstract String toString();
298
299 // -----
300 // Méthodes à implémenter définies dans l'interface Iterable<E>
301 // -----
302 /**
303  * Factory method fournissant un itérateur sur l'ensemble
304  */
305 * @return un nouvel itérateur sur cet ensemble
306 */
307 @Override
308 public abstract Iterator<E> iterator();
309 }

```

mar 10, 16 20:05

## EnsembleGenerique.java

Page 1/2

```

1 package ensembles;
2
3 import java.util.Iterator;
4
5 /**
6  * Ensemble générique implémentant partiellement les opérations communes à tous
7  * les ensembles quels que soit les conteneurs sous-jacents utilisés pour
8  * stocker les éléments de l'ensemble. L'ensemble générique est implémenté en
9  * majeure partie grâce à l'itérateur fourni par la méthode {@link #iterator()}
10  */
11 * @author davidroussel
12 */
13 public abstract class EnsembleGenerique<E> implements Ensemble<E>
14 {
15     /**
16      * (non-Javadoc)
17      * @see ensembles.Ensemble#ajout(java.lang.Object)
18      */
19     @Override
20     public abstract boolean ajout(E element);
21
22     /**
23      * (non-Javadoc)
24      * @see ensembles.Ensemble#union(ensembles.Ensemble)
25      */
26     @Override
27     public abstract Ensemble<E> union(Ensemble<E> ensemble);
28
29     /**
30      * (non-Javadoc)
31      * @see ensembles.Ensemble#intersection(ensembles.Ensemble)
32      */
33     @Override
34     public abstract Ensemble<E> intersection(Ensemble<E> ensemble);
35
36     /**
37      * (non-Javadoc)
38      * @see ensembles.Ensemble#complement(ensembles.Ensemble)
39      */
40     @Override
41     public abstract Ensemble<E> complement(Ensemble<E> ensemble);
42
43     /**
44      * (non-Javadoc)
45      * @see ensembles.Ensemble#iterator()
46      */
47     @Override
48     public abstract Iterator<E> iterator();
49
50     /**
51      * Test d'égalité entre deux ensembles
52      */
53     * @param o l'objet à comparer
54     * @return true si l'objet à comparer est un ensemble et qu'il contient les
55     * mêmes éléments (pas forcément dans le même ordre). Si les deux
56     * ensembles sont vides on considère qu'ils seront égaux quel que
57     * soit leur type de contenu (dans la mesure où l'on ne peut pas le
58     * déterminer avec {@link ensembles.Ensemble#typeElements()})
59     * @see java.lang.Object#equals(java.lang.Object)
60     */
61     @Override
62     public boolean equals(Object obj)
63     {
64         /**
65          * TODO Remplacer par :
66          * 1 - obj == null ? ==> false
67          * 2 - obj == this ? ==> true
68          * 3 - obj est une instance de Ensemble<?> ?
69          * - caster obj en Ensemble<?>
70          * - les typeElements() sont identiques ?
71          * - si typeElements des 2 est null :
72          *   ensembles vides ==> true
73          * - sinon - caster obj en (Ensemble<E>)
74          * - si tous les elts de l'un sont contenus dans l'autre ==> true
75          * - sinon ==> false
76          * - sinon (types éléments différents) ==> false
77          * - sinon obj n'est pas une instance de Ensemble<?> ==> false
78          */
79         return false;
80     }
81
82     /**
83      * Hashcode d'un ensemble en utilisant l'itérateur pour parcourir les
84      * éléments. Le HashCode d'un ensemble doit être calculé comme étant la
85      * somme des hascodes de ses éléments afin de ne pas tenir compte de
86      * l'ordre des éléments dans la collection sous-jacente.
87      */
88     * @return le hashage d'un ensemble
89     * @see java.lang.Object#hashCode()
90     */

```

mar 10, 16 20:05

## EnsembleGenerique.java

Page 2/2

```

91  @Override
92  public int hashCode()
93  {
94      int result = 0;
95      /*
96       * TODO Compl ter ...
97       */
98      return result;
99  }
100
101  /**
102   * Affichage des  l ments de l'ensemble sous la forme : par exemple pour un
103   * ensemble de 3 elts : "[elt1, elt2, elt3]" o   elt  repr sente le toString
104   * du ni me elt.
105   *
106   * @return une chaine de caract re repr sant les  l ments de l'ensemble
107   *         s par   par des virgules et encadr   par des crochets
108   * @see java.lang.Object#toString()
109   */
110  @Override
111  public String toString()
112  {
113      StringBuilder sb = new StringBuilder();
114      sb.append("[");
115      /*
116       * TODO Compl ter ...
117       */
118      sb.append("]");
119
120      return new String(sb);
121  }
122  }

```

nov 04, 15 18:00

## EnsembleTableau.java

Page 1/2

```

1  package ensembles;
2
3  import java.util.Iterator;
4
5  import tableaux.Tableau;
6
7  /**
8   * Ensemble   base de tableaux
9   *
10  * @author davidroussel
11  */
12  public class EnsembleTableau<E> extends EnsembleGenerique<E>
13  {
14      /**
15       * Conteneur sous-jacent : un Tableau<E>
16       */
17      protected Tableau<E> tableau;
18
19      /**
20       * Constructeur par d faut d'un ensemble   base de {@link tableaux.Tableau}
21       */
22      public EnsembleTableau()
23      {
24          /*
25           * TODO Remplacer par l'initialisation du tableau
26           */
27          tableau = null;
28      }
29
30      /**
31       * Constructeur de copie   partir d'un {@link Iterable}
32       *
33       * @param elements l'it rable dont on doit copier les  l ments
34       */
35      public EnsembleTableau(Iterable<E> elements)
36      {
37          /*
38           * TODO Remplacer par l'initialisation du tableau, puis l'ajout (au
39           * sens des ensembles) des  l ments de "elements"
40           */
41          tableau = null;
42      }
43
44      /**
45       * Aient d'un  l ment   un ensemble ssi celui ci n'est pas null et qu'il
46       * n'est pas d     pr sent
47       * Ce qui revient dans le cas pr sent   ajouter un  l ment au tableau si
48       * celui ci n'y est pas d     pr sent
49       *
50       * @param element l' l ment   ajouter   l'ensemble (on consid rera que l'on
51       *                 ne peut pas ajouter d' l ment null)
52       * @return true si l' l ment a pu  tre ajout     l'ensemble. false sinon ou
53       *         si l'on a tent   d'ins rer un  l ment null (auquel cas il n'est
54       *         pas ins r  )
55       * @see ensembles.EnsembleGenerique#ajout(java.lang.Object)
56       */
57      @Override
58      public boolean ajout(E element)
59      {
60          /*
61           * TODO Compl ter ...
62           */
63          return false;
64      }
65
66      /**
67       * Taille de l'ensemble : r impl mentation en utilisant les propri t s du
68       * tableau sous-jacent plut t que l'it rateur (am lioration de performances)
69       *
70       * @return le nombre d' l ments dans l'ensemble
71       * @see ensembles.EnsembleGenerique#cardinal()
72       */
73      @Override
74      public int cardinal()
75      {
76          /*
77           * TODO Remplacer par une impl mentation plus performante que celle
78           * fournie par d faut par l'interface Ensemble<E>
79           */
80          return 0;
81      }
82
83      /**
84       * Union avec un autre ensemble en r utilisant la m thode de classe union
85       *  crite dans l'ensemble G n rique (
86       * {@link ensembles.EnsembleGenerique#union(ensembles.Ensemble, ensembles.En
87       * semble)}
88       * ) et un nouvel {@link ensemble.EnsembleTableau} pour stocker le r sultat.
89       *
90       * @param ensemble l'autre ensemble avec lequel on veut cr  er une union

```

nov 04, 15 18:00

## EnsembleTableau.java

Page 2/2

```

90  * @return un nouvel ensemble contenant l'union de l'ensemble courant et de
91  * l'ensemble passé en argument
92  * @see ensembles.EnsembleGenerique#union(ensembles.Ensemble,
93  * ensembles.Ensemble, ensembles.Ensemble)
94  */
95  @Override
96  public Ensemble<E> union(Ensemble<E> ensemble)
97  {
98      /*
99       * TODO Remplacer par :
100      * - la création d'un nouvel ensemble résultat
101      * - l'union de this et ensemble dans résultat en utilisant ce que
102      * l'on a déjà écrit
103      * - le renvoi de résultat
104      */
105      return null;
106  }
107
108  /**
109   * Intersection avec un autre ensemble en utilisant la méthode de classe
110   * intersection écrite dans l'ensemble Générique (
111   * @link ensembles.EnsembleGenerique#intersection(ensembles.Ensemble, ensembles.Ensemble, ensembles.Ensemble)
112   * ) et un nouvel (@link ensemble.EnsembleTableau) pour stocker le résultat.
113   *
114   * @param ensemble l'autre ensemble avec lequel on veut créer une
115   * intersection
116   * @return un nouvel ensemble contenant l'intersection de l'ensemble courant
117   * et de l'ensemble passé en argument
118   * @see ensembles.EnsembleGenerique#intersection(ensembles.Ensemble,
119   * ensembles.Ensemble, ensembles.Ensemble)
120   */
121  @Override
122  public Ensemble<E> intersection(Ensemble<E> ensemble)
123  {
124      /*
125       * TODO Remplacer par :
126       * - la création d'un nouvel ensemble résultat
127       * - l'intersection de this et ensemble dans résultat en utilisant
128       * ce que l'on a déjà écrit
129       * - le renvoi de résultat
130       */
131      return null;
132  }
133
134  /**
135   * Complément avec un autre ensemble en utilisant la méthode de classe
136   * complément écrite dans l'ensemble Générique (
137   * @link ensembles.EnsembleGenerique#complement(ensembles.Ensemble, ensembles.Ensemble, ensembles.Ensemble)
138   * ) et un nouvel (@link ensemble.EnsembleTableau) pour stocker le résultat.
139   *
140   * @param ensemble l'autre ensemble avec lequel on veut créer le complément
141   * @return un nouvel ensemble contenant uniquement les éléments présents
142   * dans l'ensemble courant mais PAS dans l'ensemble passé en
143   * argument
144   * @see ensembles.EnsembleGenerique#complement(ensembles.Ensemble,
145   * ensembles.Ensemble, ensembles.Ensemble)
146   */
147  @Override
148  public Ensemble<E> complement(Ensemble<E> ensemble)
149  {
150      /*
151       * TODO Remplacer par :
152       * - la création d'un nouvel ensemble résultat
153       * - le complément de this et ensemble dans résultat en utilisant
154       * ce que l'on a déjà écrit
155       * - le renvoi de résultat
156       */
157      return null;
158  }
159
160  /**
161   * Factory method fournissant un itérateur sur l'ensemble en utilisant
162   * l'itérateur du tableau sous-jacent
163   *
164   * @return un nouvel itérateur sur cet ensemble
165   * @see ensembles.EnsembleGenerique#iterator()
166   */
167  @Override
168  public Iterator<E> iterator()
169  {
170      /*
171       * TODO Remplacer par la création d'un itérateur du tableau
172       */
173      return null;
174  }
175  }

```

oct 24, 15 17:37

## EnsembleFactory.java

Page 1/1

```

1  package ensembles;
2
3  import java.lang.reflect.Constructor;
4  import java.lang.reflect.InvocationTargetException;
5
6  /**
7   * Factory permettant de créer différents types d'ensembles utilisés dans les
8   * tests
9   *
10  * @author davidroussel
11  */
12  public class EnsembleFactory<E>
13  {
14      /**
15       * Obtention d'un nouvel ensemble d'après le type d'ensemble souhaité et un
16       * contenu (éventuel) à copier dans le nouvel ensemble
17       *
18       * @param typeEnsemble le type d'ensemble demandé: soit
19       *      (@link ensembles.EnsembleVector), soit
20       *      (@link ensembles.EnsembleListe), soit
21       *      (@link ensembles.EnsembleTableau)
22       * @param contenu le contenu éventuel à copier dans le nouvel ensemble ( si
23       * celui ci est nul le constructeur par défaut sera appelé, s'il
24       * est non null, le constructeur de copie sera appelé
25       *
26       * @return une nouvelle instance de l'ensemble correspondant au type demandé
27       * @throws SecurityException Si le SecurityManager ne permet pas l'accès au
28       *      constructeur demandé
29       * @throws NoSuchMethodException Si le constructeur demandé n'existe pas
30       * @throws IllegalArgumentException Si le nombre d'arguments fournis au
31       *      constructeur n'est pas le bon
32       * @throws InstantiationException si la classe demandée est abstraite
33       * @throws IllegalAccessException Si le constructeur demandé est
34       *      inaccessible
35       * @throws InvocationTargetException si le constructeur invoqué déclenche
36       *      une exception
37       */
38      @SuppressWarnings("unchecked")
39      public static <E> Ensemble<E> getEnsemble(Class<? extends Ensemble<E>> typeEnsemble, Iterable<E>
40      content)
41      throws SecurityException, NoSuchMethodException, IllegalArgumentException, Instantiation
42      Exception,
43      IllegalAccessException, InvocationTargetException
44      {
45          Constructor<? extends Ensemble<E>> constructor = null;
46          Class<?>[] argumentsTypes = null;
47          Object[] arguments = null;
48          Object instance = null;
49
50          if (content == null)
51          {
52              argumentsTypes = new Class<?>[0];
53              arguments = new Object[0];
54          }
55          else
56          {
57              argumentsTypes = new Class<?>[1];
58              argumentsTypes[0] = Iterable.class;
59              arguments = new Object[1];
60              arguments[0] = content;
61          }
62
63          constructor = typeEnsemble.getConstructor(argumentsTypes);
64
65          if (constructor != null)
66          {
67              instance = constructor.newInstance(arguments);
68          }
69
70          return (Ensemble<E>) instance;
71      }
72  }

```

nov 05, 15 15:29

## EnsembleTri.java

Page 1/1

```

1 package ensembles;
2
3 import java.util.Collection;
4
5 /**
6  * Ensemble d'éléments triés. Les éléments doivent donc être des
7  * (link Comparable) afin de pouvoir réaliser l'insertion triée de nouveaux
8  * éléments dans (link #ajout(Comparable)). A titre d'information les
9  * (link Integer) et les (link String) sont des (link Comparable).
10  */
11 * @author davidroussel
12 */
13 public interface EnsembleTri<E> extends Comparable<E> extends Ensemble<E>
14 {
15     /**
16      * Note : les redéfinitions ci-dessous ne sont pas techniquement nécessaires
17      * (sauf rang()) mais permettent de documenter les changements nécessaires
18      * dans la implémentation de ces méthodes spécialement pour les
19      * ensembles triés.
20      */
21
22     /**
23      * Ajout d'un nouvel élément de manière à maintenir l'ensemble trié
24      *
25      * @param element l'élément à ajouter de manière triée
26      * @return true si l'élément n'était pas déjà présent dans l'ensemble, false
27      *         sinon.
28      */
29     @Override
30     public abstract boolean ajout(E element);
31
32     /**
33      * Code de hachage d'un ensemble trié. Il est nécessaire de réimplémenter le
34      * code de hachage pour les ensembles triés car on considérera que deux
35      * ensembles contenant les mêmes éléments mais dans des ordres différents
36      * seront eux-mêmes différents. Il faut donc que la méthode hashCode prenne
37      * en compte l'ordre des éléments (Comme dans les autres (link Collection)
38      * d'ailleurs).
39      *
40      * @return le code de hachage de cet ensemble trié.
41      * @see listes.Liste#hashCode() tableaux.Tableau#hashCode() pour un exemple
42      * de hachage utilisant l'ordre des éléments
43      */
44     @Override
45     public abstract int hashCode();
46
47     /**
48      * Test d'égalité d'un ensemble trié. Il est nécessaire de réimplémenter la
49      * comparaison avec un autre ensemble car l'ordre des éléments aura son
50      * importance dans la comparaison ce qui n'était pas le cas avec les
51      * ensembles non triés.
52      *
53      * @return true si l'objet obj est aussi un ensemble (pas forcément trié) et
54      *         qu'il contient exactement les mêmes éléments dans le même ordre.
55      */
56     @Override
57     public abstract boolean equals(Object obj);
58
59     /**
60      * Calcule le rang où doit être inséré un élément de manière triée dans
61      * l'ensemble trié
62      *
63      * @param Element l'élément dont on veut calculer le rang dans l'ensemble
64      *                trié
65      * @return le rang d'insertion de l'élément dans l'ensemble trié
66      */
67     public default int rang(E element)
68     {
69         /**
70          * calcul du rang d'un nouvel élément : On parcourt les éléments de this
71          * et si un elt de this est plus grand que l'élément à insérer (elt de
72          * this).compareTo(element) >= 0) on a trouvé le rang où insérer, on
73          * quitte alors la boucle sans passer au suivant et on renvoie le nombre
74          * d'itérations effectuées. Cas limites : - element < 1er elt de this on
75          * quitte la boucle immédiatement - element > dernier elt de this la
76          * boucle va jusqu'au bout
77          */
78         int res = 0;
79         /**
80          * TODO Compléter ...
81          */
82         return res;
83     }
84 }

```

nov 05, 15 15:32

## EnsembleTriTableau.java

Page 1/2

```

1 package ensembles;
2
3 import java.util.Collection;
4
5 import tableaux.Tableau;
6
7 /**
8  * Ensemble trié utilisant un (link Tableau)
9  */
10 * @author davidroussel
11 */
12 public class EnsembleTriTableau<E> extends Comparable<E> extends
13     EnsembleTableau<E> implements EnsembleTri<E>
14 {
15     /**
16      * Constructeur par défaut d'un ensemble trié utilisant un (link Tableau)
17      */
18     public EnsembleTriTableau()
19     {
20         /**
21          * TODO Compléter si besoin ...
22          */
23     }
24
25     /**
26      * Constructeur de copie à partir d'un autre itérable
27      *
28      * @param elements l'itérable dont on veut copier les éléments
29      */
30     public EnsembleTriTableau(Iterable<E> elements)
31     {
32         /**
33          * TODO Compléter ...
34          */
35     }
36
37     /**
38      * Ajout d'un élément de manière triée dans l'ensemble utilisant un
39      * (link Tableau)
40      * @param element l'élément à ajouter de manière triée (on considérera que
41      *                l'on ne peut pas ajouter d'élément null)
42      * @return true si l'élément n'était pas déjà présent dans l'ensemble. false
43      *         sinon ou si l'on a tenté d'insérer un élément null (auquel cas il
44      *         n'est pas inséré)
45      * @see ensembles.EnsembleTableau#ajout(java.lang.Object)
46      * @see tableaux.Tableau#insertElement(E, int)
47      */
48     @Override
49     public boolean ajout(E element)
50     {
51         /**
52          * TODO Compléter ...
53          */
54         return false;
55     }
56
57     /**
58      * Test d'égalité d'un ensemble trié. Il est nécessaire de réimplémenter la
59      * comparaison avec un autre ensemble car l'ordre des éléments aura son
60      * importance dans la comparaison ce qui n'était pas le cas avec les
61      * ensembles non triés.
62      *
63      * @return true si l'objet obj est aussi un ensemble (pas forcément trié) et
64      *         qu'il contient exactement les mêmes éléments dans le même ordre.
65      * @see ensembles.EnsembleGenerique#equals(java.lang.Object)
66      */
67     @Override
68     public boolean equals(Object obj)
69     {
70         /**
71          * TODO Remplacer par ...
72          * 1 - obj == null ? ==> false
73          * 2 - obj == this ? ==> true
74          * 3 - obj est une instance de Ensemble<?>
75          *     - caster obj en Ensemble<?>
76          *     - si obj et this ont exactement les mêmes éléments dans le
77          *       même ordre ==> true
78          *     - sinon ==> false;
79          *     - sinon (obj n'est pas un Ensemble<?>) ==> false
80          */
81         return false;
82     }
83
84     /**
85      * Code de hachage d'un ensemble trié. Il est nécessaire de réimplémenter le
86      * code de hachage pour les ensembles triés car on considérera que deux
87      * ensembles contenant les mêmes éléments mais dans des ordres différents
88      * seront eux-mêmes différents. Il faut donc que la méthode hashCode prenne
89      * en compte l'ordre des éléments (Comme dans les autres (link Collection)
90      */

```

nov 05, 15 15:32

## EnsembleTriTableau.java

Page 2/2

```

91 * d'ailleurs).
92 *
93 * @return le code de hachage de cet ensemble trié.
94 * @see tableaux.Tableau#hashCode() pour un exemple de hachage utilisant
95 * l'ordre des éléments.
96 * @see ensembles.EnsembleGenerique#hashCode()
97 */
98 @Override
99 public int hashCode()
100 {
101     final int prime = 31;
102     int result = 1;
103     /*
104      * TODO Compléter ...
105      */
106     return result;
107 }
108 }

```

mar 10, 16 20:04

## EnsembleTriGenerique.java

Page 1/2

```

1 package ensembles;
2
3 import java.util.Collection;
4 import java.util.Iterator;
5
6 /**
7  * Implémentation générique partielle d'un ensemble trié sous forme de
8  * décorateur d'un ensemble ordinaire.
9  *
10  * @author davidroussel
11  */
12 public abstract class EnsembleTriGenerique<E> extends Comparable<E>
13     extends EnsembleGenerique<E> implements EnsembleTri<E>
14 {
15     /**
16      * Ensemble de base sous-jacent décoré par les ensembles triés.
17      */
18     protected Ensemble<E> ensemble;
19
20     /**
21      * Ajout d'un nouvel élément de manière à maintenir l'ensemble trié en
22      * utilisant la méthode {@link #insérerAuRang(E element, int rang)} ssi
23      * l'élément peut être inséré dans cet ensemble trié.
24      *
25      * @param element l'élément à ajouter de manière triée (on considérera que
26      * l'on ne peut pas ajouter d'élément null)
27      * @return true si l'élément n'était pas déjà présent dans l'ensemble. false
28      * sinon ou si l'on a tenté d'insérer un élément null (auquel cas il
29      * n'est pas inséré).
30      * @see ensembles.EnsembleListe#ajout(java.lang.Object)
31      */
32     @Override
33     public boolean ajout(E element)
34     {
35         /*
36          * TODO Compléter ...
37          */
38         return false;
39     }
40
41     /**
42      * Insertion d'un nouvel élément au rang choisi en utilisant
43      * {@link #rang(E element)} pour calculer le rang d'insertion de l'élément
44      *
45      * @param element l'élément à insérer
46      * @param rang le rang où insérer cet élément
47      * @return true si l'élément a été inséré au rang choisi. false si l'élément
48      * n'a pas pu être inséré à cause d'un rang invalide
49      * @note On remarquera que la méthode ne teste pas au préalable l'existence
50      * de l'élément à insérer dans l'ensemble car c'est la méthode
51      * {@link #ajout(E)} qui s'en chargera
52      */
53     protected abstract boolean insérerAuRang(E element, int rang);
54
55     /**
56      * Test d'égalité d'un ensemble trié. Il est nécessaire de réimplémenter la
57      * comparaison avec un autre ensemble car l'ordre des éléments aura son
58      * importance dans la comparaison ce qui n'était pas le cas avec les
59      * ensembles non triés.
60      *
61      * @return true si l'objet obj est aussi un ensemble (pas forcément trié) et
62      * qu'il contient exactement les mêmes éléments dans le même ordre.
63      * @see ensembles.EnsembleGenerique#equals(java.lang.Object)
64      */
65     @Override
66     public boolean equals(Object obj)
67     {
68         /*
69          * TODO Remplacer par ...
70          * 1 - obj == null ? ==> false
71          * 2 - obj == this ? ==> true
72          * 3 - obj est une instance de Ensemble<?>
73          *     - caster obj en Ensemble<?>
74          *     - si obj et this ont exactement les mêmes éléments dans le même ordre ==> true
75          *     - sinon ==> false;
76          * - sinon (obj n'est pas un Ensemble<?>) ==> false
77          */
78         return false;
79     }
80
81     /**
82      * Code de hachage d'un ensemble trié. Il est nécessaire de réimplémenter le
83      * code de hachage pour les ensembles triés car on considérera que deux
84      * ensembles contenant les mêmes éléments mais dans des ordres différents
85      * seront eux-mêmes différents. Il faut donc que la méthode hashCode prenne
86      * en compte l'ordre des éléments (Comme dans les autres {@link Collection}
87      * d'ailleurs).
88      *
89      * @return le code de hachage de cet ensemble trié.

```

mar 10, 16 20:04

## EnsembleTriGenerique.java

Page 2/2

```

90  * @see Listes.Liste#hashCode() ou tableaux.Tableau#hashCode() pour un
91  * exemple de hashage utilisant l'ordre des éléments
92  * @see ensembles.EnsembleGenerique#hashCode()
93  */
94  @Override
95  public int hashCode()
96  {
97      final int prime = 31;
98      int result = 1;
99      /*
100       * TODO Compléter ...
101       */
102      return result;
103  }
104
105  /**
106   * Union avec un autre ensemble : reste semblable à l'union avec un
107   * ensemble non trié mais s'applique sur l'ensemble donné (@link #ensemble)
108   *
109   * @param ensemble l'autre ensemble avec lequel on veut créer une union
110   * @return un nouvel ensemble contenant l'union de l'ensemble courant et de
111   *         l'ensemble passé en argument
112   * @see ensembles.EnsembleGenerique#union(ensembles.Ensemble)
113   */
114  @Override
115  public Ensemble<E> union(Ensemble<E> autre)
116  {
117      /*
118       * TODO Remplacer par l'implémentation ...
119       */
120      return null;
121  }
122
123  /**
124   * Intersection avec un autre ensemble : reste semblable à l'intersection
125   * avec un ensemble non trié mais s'applique sur l'ensemble donné (@link #ensemble)
126   *
127   * @param ensemble l'autre ensemble avec lequel on veut créer une
128   *         intersection
129   * @return un nouvel ensemble contenant l'intersection de l'ensemble courant
130   *         et de l'ensemble passé en argument
131   * @see ensembles.EnsembleGenerique#intersection(ensembles.Ensemble)
132   */
133  @Override
134  public Ensemble<E> intersection(Ensemble<E> autre)
135  {
136      /*
137       * TODO Remplacer par l'implémentation ...
138       */
139      return null;
140  }
141
142  /**
143   * Complément avec un autre ensemble : reste semblable au complément avec
144   * avec un ensemble non trié mais s'applique sur l'ensemble donné (@link #ensemble)
145   *
146   * @param ensemble l'autre ensemble avec lequel on veut créer un complément
147   * @return un nouvel ensemble contenant le complément de l'ensemble courant
148   *         et de l'ensemble passé en argument
149   * @see ensembles.EnsembleGenerique#complement(ensembles.Ensemble)
150   */
151  @Override
152  public Ensemble<E> complement(Ensemble<E> autre)
153  {
154      /*
155       * TODO Remplacer par l'implémentation ...
156       */
157      return null;
158  }
159
160  /**
161   * Factory method fournissant un itérateur sur l'ensemble en utilisant
162   * l'itérateur de l'ensemble ordinaire sous-jacent.
163   *
164   * @return un nouvel itérateur sur cet ensemble
165   * @see ensembles.EnsembleGenerique#iterator()
166   */
167  @Override
168  public Iterator<E> iterator()
169  {
170      /*
171       * TODO Remplacer par l'implémentation ...
172       */
173      return null;
174  }
175
176  }
177

```

oct 24, 15 17:38

## EnsembleTriFactory.java

Page 1/1

```

1  package ensembles;
2
3  import java.lang.reflect.InvocationTargetException;
4
5  /**
6   * Factory permettant de créer différents types d'ensembles triés utilisés dans
7   * les tests
8   *
9   * @author davidroussel
10  */
11  public class EnsembleTriFactory<E> extends Comparable<E>
12  {
13      /**
14       * Obtention d'un nouvel ensemble trié d'après le type d'ensemble souhaité
15       * et un contenu (éventuel) à copier dans le nouvel ensemble
16       *
17       * @param typeEnsemble le type d'ensemble demandé : soit
18       *         {@link ensembles.EnsembleTriVector1} soit
19       *         {@link ensembles.EnsembleTriVector2} soit
20       *         {@link ensembles.EnsembleTriListe1} soit
21       *         {@link ensembles.EnsembleTriListe2} soit
22       *         {@link ensembles.EnsembleTriTableau1} soit
23       *         {@link ensembles.EnsembleTriTableau2}
24       * @param contenu le contenu éventuel à copier dans le nouvel ensemble ( si
25       *         celui ci est nul le constructeur par défaut sera appelé, s'il
26       *         est non null le constructeur de copie sera appelé
27       * @return une nouvelle instance de l'ensemble correspondant au type demandé
28       * @throws SecurityException Si le SecurityManager ne permet pas l'accès au
29       *         constructeur demandé
30       * @throws NoSuchMethodException Si le constructeur demandé n'existe pas
31       * @throws IllegalArgumentException Si le nombre d'arguments fournis au
32       *         constructeur n'est pas le bon
33       * @throws InstantiationException si la classe demandée est abstraite
34       * @throws IllegalAccessException Si le constructeur demandé est
35       *         inaccessible
36       * @throws InvocationTargetException si le constructeur invoqué déclenche
37       *         une exception
38       */
39      public static <E> extends Comparable<E>> EnsembleTri<E> getEnsemble(Class<?> extends EnsembleTri<E>
40      >> typeEnsemble,
41      Iterable<E> contenu) throws SecurityException, NoSuchMethodException, IllegalArgumentException,
42      IllegalAccessException, InstantiationException, InvocationTargetException
43      {
44          return (EnsembleTri<E>) EnsembleFactory.<E> getEnsemble(typeEnsemble, contenu);
45      }
46  }

```

nov 03, 13 19:24

**package-info.java**

Page 1/1

```
1  /**
2   * Package contenant les classes de test
3   */
4  package tests;
```

oct 01, 14 17:15

**AllTests.java**

Page 1/1

```
1  package tests;
2
3  import org.junit.runner.RunWith;
4  import org.junit.runners.Suite;
5  import org.junit.runners.Suite.SuiteClasses;
6
7  /**
8   * Suite de tests
9   * @author davidroussel
10  */
11  @RunWith(Suite.class)
12  @SuiteClasses({
13      {
14          AllEnsembleTest.class,
15          EnsembleTriTest.class
16      }
17  })
18  public class AllTests
19  {
20      // Nothing
21  }
```

mar 11, 16 16:05

AllEnsembleTest.java

Page 1/14

```

1 package tests;
2
3 import static org.junit.Assert.assertEquals;
4 import static org.junit.Assert.assertFalse;
5 import static org.junit.Assert.assertNotNull;
6 import static org.junit.Assert.assertTrue;
7 import static org.junit.Assert.fail;
8
9 import java.lang.reflect.InvocationTargetException;
10 import java.util.ArrayList;
11 import java.util.Arrays;
12 import java.util.Collection;
13 import java.util.Collections;
14 import java.util.HashMap;
15 import java.util.Iterator;
16 import java.util.List;
17 import java.util.Map;
18
19 import org.junit.After;
20 import org.junit.AfterClass;
21 import org.junit.Before;
22 import org.junit.BeforeClass;
23 import org.junit.Test;
24 import org.junit.runner.RunWith;
25 import org.junit.runners.Parameterized;
26 import org.junit.runners.Parameterized.Parameters;
27
28 import ensembles.Ensemble;
29 import ensembles.EnsembleFactory;
30 import ensembles.EnsembleTableau;
31 import ensembles.EnsembleTri;
32
33 /**
34  * Classe de test pour tous les types d'ensembles.
35  * {@link ensembles.EnsembleVector}. {@link ensembles.EnsembleListe},
36  * {@link ensembles.EnsembleTableau}
37  * Mais aussi pour les méthodes communes avec les ensembles triés tels que
38  * {@link ensembles.EnsembleTriVector}. {@link ensembles.EnsembleTriVector2},
39  * {@link ensembles.EnsembleTriListe}. {@link ensembles.EnsembleTriListe2}.
40  * {@link ensembles.EnsembleTriTableau}, {@link ensembles.EnsembleTriTableau2}
41  * @author davidroussel
42  */
43 @RunWith(value = Parameterized.class)
44 public class AllEnsembleTest
45 {
46     /**
47      * 1'ensemble à tester
48      */
49     private Ensemble<String> ensemble;
50
51     /**
52      * Le type d'ensemble à tester.
53      */
54     private Class<? extends Ensemble<String>> typeEnsemble;
55
56     /**
57      * Nom du type d'ensemble à tester
58      */
59     private String typeName;
60
61     /**
62      * Les différentes natures d'ensembles à tester
63      */
64     @SuppressWarnings("unchecked")
65     private static final Class<? extends Ensemble<String>>[] typesEnsemble =
66         (Class<? extends Ensemble<String>>[]) new Class<?>[]
67         {
68             /*
69              * TODO Commenter / décommenter les lignes ci-dessous en fonction
70              * de votre avancement (Attention la dernière ligne non commentée
71              * ne doit pas avoir de virgule)
72              */
73             EnsembleTableau.class,
74             // EnsembleVector.class,
75             // EnsembleListe.class,
76             // EnsembleTriVector.class,
77             // EnsembleTriVector2.class,
78             // EnsembleTriTableau.class,
79             // EnsembleTriTableau2.class,
80             // EnsembleTriListe.class,
81             // EnsembleTriListe2.class
82         };
83
84     /**
85      * Elements pour remplir l'ensemble : "Lorem ipsum dolor sit amet"
86      */
87     private static final String[] elements1 = new String[] {
88         "Lorem",
89         "ipsum",
90         "sit",

```

Vendredi mars 10, 2017

src/tests/AllEnsembleTest.java

mar 11, 16 16:05

AllEnsembleTest.java

Page 2/14

```

91     "dolor",
92     "amet"
93 };
94
95 /**
96  * Autres Elements pour remplir un ensemble :
97  * "dolor amet consectetur adipiscing elit"
98  */
99     private static final String[] elements2 = new String[] {
100         "dolor",
101         "amet",
102         "consectetur",
103         "adipiscing",
104         "elit"
105     };
106
107     /**
108      * Elements union de {@value #elements1} et {@link #elements2}
109      */
110     private static final String[] allSingleElements = new String[] {
111         "Lorem",
112         "ipsum",
113         "sit",
114         "dolor",
115         "amet",
116         "consectetur",
117         "adipiscing",
118         "elit"
119     };
120
121     /**
122      * Elements union triée de {@value #elements1} et
123      * {@link #elements2}
124      */
125     private static final String[] allSingleElementsSorted = new String[] {
126         "Lorem",
127         "adipiscing",
128         "amet",
129         "consectetur",
130         "dolor",
131         "elit",
132         "ipsum",
133         "sit"
134     };
135
136     /**
137      * Elements communs à {@value #elements1} et {@link #elements2}
138      */
139     private static final String[] commonSingleElements = new String[] {
140         "dolor",
141         "amet"
142     };
143
144     /**
145      * Elements du complément de {@value #elements1} et
146      * {@link #elements2}
147      */
148     private static final String[] complementElements1 = new String[] {
149         "Lorem",
150         "ipsum",
151         "sit"
152     };
153
154     /**
155      * Elements du complément de {@value #elements2} et
156      * {@link #elements1}
157      */
158     private static final String[] complementElements2 = new String[] {
159         "consectetur",
160         "adipiscing",
161         "elit"
162     };
163
164     /**
165      * Elements non communs à {@value #elements1} et
166      * {@link #elements2}
167      */
168     private static final String[] diffSingleElements = new String[] {
169         "Lorem",
170         "ipsum",
171         "sit",
172         "consectetur",
173         "adipiscing",
174         "elit"
175     };
176
177     /**
178      * Elements pour remplir l'ensemble avec des doublons pour vérifier que ceux
179      * ci ne seront pas ajoutés dans les ensembles
180      */

```

16/33



mar 11, 16 16:05

## AllEnsembleTest.java

Page 3/14

```

181 private static final String[] elements = new String[elements1.length
182     + elements2.length];
183
184 /**
185  * Collection pour contenir les éléments de remplissage
186  */
187 private ArrayList<String> listElements;
188
189 /**
190  * Construit une instance de Ensemble<String> en fonction d'un type
191  * d'ensemble à créer et éventuellement d'un contenu l'ensemble à mettre en
192  * place
193  *
194  * @param testName le message à retourner dans les assertions en fonction du
195  *   test dans lequel est employée cette méthode
196  * @param type le type d'ensemble à créer
197  * @param content le contenu à mettre en place dans le nouvel ensemble, ou
198  *   bien null si aucun contenu n'est requis.
199  * @return un nouvel ensemble du type demandé evt rempli avec le contenu
200  *   fournit s'il est non null.
201  */
202 private static Ensemble<String>
203     constructEnsemble(String testName,
204         Class<? extends Ensemble<String>> type,
205         Iterable<String> content)
206 {
207     Ensemble<String> ensemble = null;
208
209     try
210     {
211         ensemble = EnsembleFactory.<String>getEnsemble(type, content);
212     }
213     catch (SecurityException e)
214     {
215         fail(testName + " constructor security exception");
216     }
217     catch (NoSuchMethodException e)
218     {
219         fail(testName + " constructor not found");
220     }
221     catch (IllegalArgumentException e)
222     {
223         fail(testName + " wrong constructor arguments");
224     }
225     catch (InstantiationException e)
226     {
227         fail(testName + " instantiation exception");
228     }
229     catch (IllegalAccessException e)
230     {
231         fail(testName + " illegal access");
232     }
233     catch (InvocationTargetException e)
234     {
235         fail(testName + " invocation exception");
236     }
237
238     return ensemble;
239 }
240
241 /**
242  * Compare les éléments d'un ensemble pour vérifier qu'ils sont tous dans
243  * un tableau donné
244  * @param testName le nom du test dans lequel est utilisée cette méthode
245  * @param ensemble l'ensemble dont on doit comparer les éléments
246  * @param array le tableau utilisé pour vérifier la présence des éléments
247  *   de l'ensemble
248  * @return true si tous les éléments du tableau sont présents dans l'ensemble
249  */
250 private static boolean compareElts2Array(String testName,
251     Ensemble<String> ensemble, String[] array)
252 {
253     for (String elt : array)
254     {
255         boolean contenu = ensemble.contient(elt);
256         assertTrue(testName + " contient(" + elt + ") failed", contenu);
257         if (!contenu)
258         {
259             return false;
260         }
261     }
262     return true;
263 }
264
265 /**
266  * Vérifie qu'un ensemble ne contient qu'un seul exemplaire de chacun
267  * de ses éléments
268  * @param testName le nom du test dans lequel est employée cette méthode
269  * @param ensemble l'ensemble à tester
270  * @return true si chaque élément de l'ensemble n'existe qu'à un seul

```

mar 11, 16 16:05

## AllEnsembleTest.java

Page 4/14

```

271 * exempleire.
272 */
273 private static <E> boolean checkCount(String testName, Ensemble<E> ensemble)
274 {
275     Map<E, Integer> wordCount = new HashMap<E, Integer>();
276     for (E elt : ensemble)
277     {
278         if (!wordCount.containsKey(elt))
279         {
280             wordCount.put(elt, Integer.valueOf(1));
281         }
282         else
283         {
284             Integer count = wordCount.get(elt);
285             count = Integer.valueOf(count.intValue() + 1);
286             wordCount.put(elt, count);
287         }
288     }
289
290     for (Integer i : wordCount.values())
291     {
292         int countValue = i.intValue();
293         assertEquals(testName + " count check #" + countValue + " failed",
294             1, countValue);
295         if (countValue != 1)
296         {
297             return false;
298         }
299     }
300
301     return true;
302 }
303
304 /**
305  * Mélange les éléments d'un tableau
306  * @param elements les éléments à mélanger
307  * @return un tableau de même dimension avec les éléments dans un autre
308  *   ordre
309  */
310 private static String[] shuffleElements(String[] elements)
311 {
312     List<String> listElements = Arrays.asList(elements);
313
314     Collections.shuffle(listElements);
315
316     String[] result = new String[elements.length];
317     int i = 0;
318     for (String elt : listElements)
319     {
320         result[i++] = elt;
321     }
322
323     return result;
324 }
325
326 /**
327  * Paramètres à transmettre au constructeur de la classe de test.
328  *
329  * @return une collection de tableaux d'objet contenant les paramètres à
330  *   transmettre au constructeur de la classe de test
331  */
332 @Parameters(name = "{index}:{}")
333 public static Collection<Object[]> data()
334 {
335     Object[][] data = new Object[typesEnsemble.length][2];
336     for (int i = 0; i < typesEnsemble.length; i++)
337     {
338         data[i][0] = typesEnsemble[i];
339         data[i][1] = typesEnsemble[i].getSimpleName();
340     }
341
342     return Arrays.asList(data);
343 }
344
345 /**
346  * Constructeur paramétré par le type d'ensemble à tester.
347  * Lance pour chaque test
348  * @param typeEnsemble le type d'ensemble à créer
349  * @param le nom du type d'ensemble à tester (pour le faire apparaître
350  *   dans le déroulement des tests).
351  */
352 public AllEnsembleTest(Class<? extends Ensemble<String>> typeEnsemble,
353     String typeEnsembleName)
354 {
355     this.typeEnsemble = typeEnsemble;
356     typeName = typeEnsembleName;
357 }
358
359 /**
360  * Mise en place avant l'ensemble des tests

```

mar 11, 16 16:05

## AllEnsembleTest.java

Page 5/14

```

361  * @throws java.lang.Exception
362  */
363  @BeforeClass
364  public static void setUpBeforeClass() throws Exception
365  {
366      int j = 0;
367      for (int i = 0; i < elements1.length; i++)
368      {
369          elements[j++] = elements1[i];
370      }
371      for (int i = 0; i < elements2.length; i++)
372      {
373          elements[j++] = elements2[i];
374      }
375      System.out.println("-----");
376      System.out.println("Test des ensembles");
377      System.out.println("-----");
378  }
379
380  /**
381   * Nettoyage apr s l'ensemble des tests
382   * @throws java.lang.Exception
383   */
384  @AfterClass
385  public static void tearDownAfterClass() throws Exception
386  {
387      System.out.println("-----");
388      System.out.println("Fin Test des ensembles");
389      System.out.println("-----");
390  }
391
392  /**
393   * Mise en place avant chaque test
394   * @throws java.lang.Exception
395   */
396  @Before
397  public void setUp() throws Exception
398  {
399      ensemble = constructEnsemble("setUp", typeEnsemble, null);
400      assertNotNull("setUp non null ensemble failed", ensemble);
401
402      listElements = new ArrayList<String>();
403      for (String elt : elements)
404      {
405          listElements.add(elt);
406      }
407  }
408
409  /**
410   * Nettoyage apr s chaque test
411   * @throws java.lang.Exception
412   */
413  @After
414  public void tearDown() throws Exception
415  {
416      ensemble.affiche();
417      ensemble = null;
418      listElements.clear();
419      listElements = null;
420  }
421
422  /**
423   * Test method for {@link ensembles.EnsembleVector#EnsembleVector()} or
424   * {@link ensembles.EnsembleListe#EnsembleListe()} or
425   * {@link ensembles.EnsembleTableau#EnsembleTableau()}
426   */
427  @Test
428  public final void testDefaultConstructor()
429  {
430      String testName = new String(typeName + "()");
431      System.out.println(testName);
432
433      ensemble = constructEnsemble(testName, typeEnsemble, null);
434      assertNotNull(testName + " non null instance failed", ensemble);
435
436      assertEquals(testName + " instance type failed", typeEnsemble,
437          ensemble.getClass());
438      assertTrue(testName + " empty instance failed", ensemble.estVide());
439      assertEquals(testName + " instance size failed", 0, ensemble.cardinal());
440  }
441
442  /**
443   * Test method for {@link ensembles.EnsembleVector#EnsembleVector(Iterable)}
444   * or {@link ensembles.EnsembleListe#EnsembleListe(Iterable)} or
445   * {@link ensembles.EnsembleTableau#EnsembleTableau(Iterable)}
446   */
447  @Test
448  public final void testCopyConstructor()
449  {
450      String testName = new String(typeName + "(Iterable)");

```

Vendredi mars 10, 2017

src/tests/AllEnsembleTest.java

mar 11, 16 16:05

## AllEnsembleTest.java

Page 6/14

```

451      System.out.println(testName);
452
453      ensemble = constructEnsemble(testName, typeEnsemble, listElements);
454      assertNotNull(testName + " non null instance failed", ensemble);
455
456      assertEquals(testName + " instance type failed", typeEnsemble,
457          ensemble.getClass());
458      assertFalse(testName + " not empty instance failed", ensemble.estVide());
459      boolean compare = compareElts2Array(testName, ensemble, allSingleElements);
460      assertTrue(testName + " elts compare failed", compare);
461
462      // Tous les  l ments de ensemble doivent se retrouver dans list
463      for (String elt : ensemble)
464      {
465          assertTrue(testName + "check content [" + elt + "] failed",
466              listElements.contains(elt));
467      }
468
469      // Tous les  l ments de l'ensemble n'existent qu'  un seul exemplaire
470      boolean countCheck = AllEnsembleTest.<String>checkCount(testName, ensemble);
471
472      assertTrue(testName + "after count check failed", countCheck);
473  }
474
475  /**
476   * Test method for {@link ensembles.Ensemble#ajout(java.lang.Object)}.
477   */
478  @Test
479  public final void testAjout()
480  {
481      String testName = new String(typeName + ".ajout(E)");
482      System.out.println(testName);
483
484      // Ensemble vide avant remplissage
485      assertEquals(testName + " ensemble vide failed", 0, ensemble.cardinal());
486      int count = 0;
487      for (String elt : elements)
488      {
489          if (!ensemble.contient(elt))
490          {
491              count++;
492          }
493          ensemble.ajout(elt);
494      }
495      // Ensemble non vide apr s remplissage
496      assertEquals(testName + " ensemble rempli failed", count,
497          ensemble.cardinal());
498
499      // Verif taille ensemble
500      boolean countCheck = AllEnsembleTest.<String>checkCount(testName, ensemble);
501      assertTrue(testName + "after count check failed", countCheck);
502
503      // Comparaison des elts avec allSingleElements
504      boolean compare = compareElts2Array(testName, ensemble, allSingleElements);
505      assertTrue(testName + "elts compare failed", compare);
506
507      // Ajout d'un elt null
508      boolean ajoutNull = ensemble.ajout(null);
509      assertFalse(testName + " ajout null is true", ajoutNull);
510  }
511
512  /**
513   * Test method for {@link ensembles.Ensemble#retrait(java.lang.Object)}.
514   */
515  @Test
516  public final void testRetrait()
517  {
518      String testName = new String(typeName + ".retrait(E)");
519      System.out.println(testName);
520
521      ensemble = constructEnsemble(testName, typeEnsemble, listElements);
522      assertNotNull(testName + " non null instance failed", ensemble);
523
524      String[] elementsToRemove = shuffleElements(allSingleElements);
525
526      for (String elt : elementsToRemove)
527      {
528          ensemble.retrait(elt);
529
530          assertFalse(testName + "no more contains " + elt + " failed",
531              ensemble.contient(elt));
532      }
533
534      assertTrue(testName + " ensemble vide apr s retraits failed",
535          ensemble.estVide());
536  }
537
538  /**
539   * Test method for {@link ensembles.Ensemble#estVide()}.
540   */

```

18/33

mar 11, 16 16:05

## AllEnsembleTest.java

Page 7/14

```

541  @Test
542  public final void testEstVide()
543  {
544      String testName = new String(typeName + ".estVide()");
545      System.out.println(testName);
546
547      assertTrue(testName + " ensemble vide failed", ensemble.estVide());
548      assertFalse(testName + " ens vide rien Ã itÃrer failed",
549                  ensemble.iterator().hasNext());
550
551      ensemble = constructEnsemble(testName, typeEnsemble, listElements);
552      assertNotNull(testName + " non null instance failed", ensemble);
553
554      assertFalse(testName + " ensemble vide failed", ensemble.estVide());
555      assertTrue(testName + " ens non vide iterable failed",
556                  ensemble.iterator().hasNext());
557  }
558
559  /**
560   * Test method for {@link ensembles.Ensemble#contient(java.lang.Object)}.
561   */
562  @Test
563  public final void testContientENull()
564  {
565      String testName = new String(typeName + ".contient((E)null)");
566      System.out.println(testName);
567      String mot = null;
568
569      // Contient null sur ensemble vide
570      assertFalse(testName + " ens vide !contient(null) failed",
571                  ensemble.contient(mot));
572
573      // remplissage ensemble
574      ensemble = constructEnsemble(testName, typeEnsemble, listElements);
575      assertNotNull(testName + " non null instance failed", ensemble);
576      assertEquals(testName + " instance remplie failed",
577                  allSingleElements.length, ensemble.cardinal());
578
579      // Contient null sur ensemble non vide
580      assertFalse(testName + " ens plein !contient(null) failed",
581                  ensemble.contient((String) null));
582  }
583
584  /**
585   * Test method for {@link ensembles.Ensemble#contient(java.lang.Object)}.
586   */
587  @Test
588  public final void testContientE()
589  {
590      String testName = new String(typeName + ".contient(E)");
591      System.out.println(testName);
592      String mot = new String("Bonjour");
593
594      // Contient mot quelconque sur ensemble vide
595      assertFalse(testName + " ens vide !contient(" + mot + ") failed",
596                  ensemble.contient(mot));
597
598      ensemble = constructEnsemble(testName, typeEnsemble, listElements);
599      assertNotNull(testName + " non null instance failed", ensemble);
600
601      // Contient mot quelconque sur ensemble non vide
602      assertFalse(testName + " ens vide contient(" + mot + ") failed",
603                  ensemble.contient(mot));
604
605      // Contient mots contenus
606      boolean compare = compareElts2Array(testName, ensemble, allSingleElements);
607      assertTrue(testName + " elts compare failed", compare);
608  }
609
610  /**
611   * Test method for {@link ensembles.Ensemble#contient(ensembles.Ensemble)}.
612   */
613  @Test
614  public final void testContientEnsembleNull()
615  {
616      String testName = new String(typeName + ".contient((Ensemble<E>)null)");
617      System.out.println(testName);
618
619      // !Contient ensemble null dans ensemble vide
620      assertFalse(testName + " ens vide !contient(null) failed",
621                  ensemble.contient((Ensemble<String>) null));
622
623      // !Contient ensemble null dans ensemble plein
624      ensemble = constructEnsemble(testName, typeEnsemble, listElements);
625      assertNotNull(testName + " non null instance failed", ensemble);
626      assertEquals(testName + " instance remplie taille failed",
627                  allSingleElements.length, ensemble.cardinal());
628
629      assertFalse(testName + " ens plein non !contient(null) failed",
630                  ensemble.contient((Ensemble<String>) null));

```

Vendredi mars 10, 2017

src/tests/AllEnsembleTest.java

mar 11, 16 16:05

## AllEnsembleTest.java

Page 8/14

```

631  }
632
633  /**
634   * Test method for {@link ensembles.Ensemble#contient(ensembles.Ensemble)}.
635   */
636  @Test
637  public final void testContientEnsembleOfE()
638  {
639      for (int i = 0; i < typesEnsemble.length; i++)
640      {
641          Class<? extends Ensemble<String>> otherType = typesEnsemble[i];
642          String otherTypeName = otherType.getSimpleName();
643
644          String testName = new String(typeName + ".contient("
645              + otherTypeName + "<E>");
646          System.out.println(testName);
647
648          // sous ensemble vide
649          Ensemble<String> sousEnsemble = constructEnsemble(testName,
650              typesEnsemble[i], null);
651          assertNotNull(testName + " sousEnsemble non null instance failed",
652              sousEnsemble);
653
654          // Contient sous ensemble vide dans ensemble vide
655          assertTrue(testName + " ens vide contient sous ens["
656              + typesEnsemble[i].getSimpleName() + "] vide failed",
657              ensemble.contient(sousEnsemble));
658
659          // remplissage ensemble
660          for (String elt : elements1)
661          {
662              ensemble.ajout(elt);
663          }
664
665          // Contient sous ensemble vide dans ensemble non vide
666          assertTrue(testName + " ens plein contient sous ens["
667              + typesEnsemble[i].getSimpleName() + "] vide failed",
668              ensemble.contient(sousEnsemble));
669
670          // remplissage sous ensemble
671          for (int j = 0; j < (elements1.length / 2); j++)
672          {
673              sousEnsemble.ajout(elements1[j]);
674          }
675
676          // Contient sous ensemble non vide ds ens non vide
677          assertTrue(testName + " ens plein contient sous ens["
678              + typesEnsemble[i].getSimpleName() + "] failed",
679              ensemble.contient(sousEnsemble));
680
681          // !Contient sous ensemble non vide non contenu ds ens non vide
682          sousEnsemble.ajout("consectetur");
683          assertFalse(testName + " ens plein !contient sous ens["
684              + typesEnsemble[i].getSimpleName() + "] failed",
685              ensemble.contient(sousEnsemble));
686
687          ensemble.efface();
688      }
689
690  /**
691   * Test method for {@link ensembles.Ensemble#efface()}.
692   */
693  @Test
694  public final void testEfface()
695  {
696      String testName = new String(typeName + ".efface()");
697      System.out.println(testName);
698
699      assertTrue(testName + " ens vide avant effacement failed",
700                  ensemble.estVide());
701
702      // Effacement ensemble vide
703      ensemble.efface();
704      assertTrue(testName + " ens vide aprÃ effacement failed", ensemble.estVide());
705
706      // Effacement ensemble non vide
707      ensemble = constructEnsemble(testName, typeEnsemble, listElements);
708      assertNotNull(testName + " non null instance failed", ensemble);
709      assertFalse(testName + " ens non vide aprÃ remplissage failed",
710                  ensemble.estVide());
711      ensemble.efface();
712      assertTrue(testName + " ens vide aprÃ remplissage & effacement failed",
713                  ensemble.estVide());
714  }
715
716  /**
717   * Test method for {@link ensembles.Ensemble#cardinal()}.
718   */
719  @Test
720

```

19/33

mar 11, 16 16:05

## AllEnsembleTest.java

Page 9/14

```

721 public final void testCardinal()
722 {
723     String testName = new String(typeName + ".cardinal()");
724     System.out.println(testName);
725
726     assertTrue(testName + " ensemble vide failed", ensemble.estVide());
727     assertEquals(testName + " cardinal 0 sur ensemble vide failed", 0,
728         ensemble.cardinal());
729
730     ensemble = constructEnsemble(testName, typeEnsemble, listElements);
731     assertNotNull(testName + " non null instance failed", ensemble);
732
733     assertFalse(testName + " ensemble non vide failed", ensemble.estVide());
734     assertEquals(testName + " cardinal " + allSingleElements.length
735         + " sur ensemble rempli failed", allSingleElements.length,
736         ensemble.cardinal());
737 }
738
739 /**
740  * Test method for {@link ensembles.Ensemble#union(ensembles.Ensemble)}.
741  */
742 @Test
743 public final void testUnion()
744 {
745     for (int i = 0; i < typesEnsemble.length; i++)
746     {
747         Class<? extends Ensemble<String>> otherType = typesEnsemble[i];
748         String otherTypeName = otherType.getSimpleName();
749
750         String testName = new String(typeName + ".union(" + otherTypeName
751             + "<E>");
752         System.out.println(testName);
753
754         // remplissage ensemble avec singleElements
755         for (String elt : elements1)
756         {
757             ensemble.ajout(elt);
758         }
759
760         // remplissage other avec singleElements2
761         Ensemble<String> other = constructEnsemble(testName,
762             typesEnsemble[i], null);
763         assertNotNull(testName + " other instance non null failed", other);
764         for (String elt : elements2)
765         {
766             other.ajout(elt);
767         }
768
769         Ensemble<String> union = ensemble.union(other);
770
771         assertNotNull(testName + " non null union instance failed", union);
772         assertFalse(testName + " self union", ensemble == union);
773         assertFalse(testName + " self union", other == union);
774         assertEquals(testName + " taille failed",
775             allSingleElements.length, union.cardinal());
776         boolean compare = compareElts2Array(testName, union,
777             allSingleElements);
778         assertTrue(testName + " elts compare failed", compare);
779     }
780 }
781
782 /**
783  * Test method for {@link ensembles.Ensemble#intersection(ensembles.Ensemble)}.
784  */
785 @Test
786 public final void testIntersection()
787 {
788     for (int i = 0; i < typesEnsemble.length; i++)
789     {
790         Class<? extends Ensemble<String>> otherType = typesEnsemble[i];
791         String otherTypeName = otherType.getSimpleName();
792
793         String testName = new String(typeName + ".intersection("
794             + otherTypeName + "<E>");
795         System.out.println(testName);
796
797         // remplissage ensemble avec singleElements
798         for (String elt : elements1)
799         {
800             ensemble.ajout(elt);
801         }
802
803         // remplissage other avec singleElements2
804         Ensemble<String> other = constructEnsemble(testName,
805             typesEnsemble[i], null);
806         assertNotNull(testName + " other non null instance failed", other);
807         for (String elt : elements2)
808         {
809             other.ajout(elt);
810         }

```

mar 11, 16 16:05

## AllEnsembleTest.java

Page 10/14

```

811     Ensemble<String> intersection = ensemble.intersection(other);
812
813     assertNotNull(testName + " non null intersection instance failed",
814         intersection);
815     assertFalse(testName + " self intersection", ensemble == intersection);
816     assertFalse(testName + " self intersection", other == intersection);
817     assertEquals(testName + " taille failed",
818         commonSingleElements.length, intersection.cardinal());
819     boolean compare = compareElts2Array(testName, intersection,
820         commonSingleElements);
821     assertTrue(testName + " elts compare failed", compare);
822 }
823
824 /**
825  * Test method for {@link ensembles.Ensemble#complement(ensembles.Ensemble)}.
826  */
827 @Test
828 public final void testComplement()
829 {
830     for (int i = 0; i < typesEnsemble.length; i++)
831     {
832         Class<? extends Ensemble<String>> otherType = typesEnsemble[i];
833         String otherTypeName = otherType.getSimpleName();
834
835         String testName = new String(typeName + ".complement("
836             + otherTypeName + "<E>");
837         System.out.println(testName);
838
839         // remplissage ensemble avec singleElements
840         for (String elt : elements1)
841         {
842             ensemble.ajout(elt);
843         }
844
845         // remplissage other avec singleElements2
846         Ensemble<String> other = constructEnsemble(testName,
847             typesEnsemble[i], null);
848         assertNotNull(testName + " other non null instance failed", other);
849         for (String elt : elements2)
850         {
851             other.ajout(elt);
852         }
853
854         Ensemble<String> complement1 = ensemble.complement(other);
855
856         assertNotNull(testName + " non null complement instance 1 failed",
857             complement1);
858         assertFalse(testName + " self complement1", ensemble == complement1);
859         assertFalse(testName + " self complement1", other == complement1);
860         assertEquals(testName + " taille 1 failed",
861             complementElements1.length, complement1.cardinal());
862         boolean compare = compareElts2Array(testName, complement1,
863             complementElements1);
864         assertTrue(testName + " elts compare 1 failed", compare);
865
866         Ensemble<String> complement2 = other.complement(ensemble);
867
868         assertNotNull(testName + " non null complement instance 2 failed",
869             complement2);
870         assertFalse(testName + " self complement2", ensemble == complement2);
871         assertFalse(testName + " self complement2", other == complement2);
872         assertEquals(testName + " taille 2 failed",
873             complementElements2.length, complement2.cardinal());
874         compare = compareElts2Array(testName, complement2,
875             complementElements2);
876         assertTrue(testName + " elts compare 2 failed", compare);
877     }
878 }
879
880 /**
881  * Test method for {@link ensembles.Ensemble#difference(ensembles.Ensemble)}.
882  */
883 @Test
884 public final void testDifference()
885 {
886     for (int i = 0; i < typesEnsemble.length; i++)
887     {
888         Class<? extends Ensemble<String>> otherType = typesEnsemble[i];
889         String otherTypeName = otherType.getSimpleName();
890
891         String testName = new String(typeName + ".difference("
892             + otherTypeName + "<E>");
893         System.out.println(testName);
894
895         // remplissage ensemble avec singleElements
896         for (String elt : elements1)
897         {
898             ensemble.ajout(elt);
899         }

```

mar 11, 16 16:05

## AllEnsembleTest.java

Page 11/14

```

901     }
902     // remplissage other avec singleElements2
903     Ensemble<String> other = constructEnsemble(testName,
904         typesEnsemble[i], null);
905     assertNotNull(testName + " other non null instance failed", other);
906
907     for (String elt : elements2)
908     {
909         other.ajout(elt);
910     }
911
912     Ensemble<String> difference = ensemble.difference(other);
913
914     assertNotNull(testName + " difference non null instance failed",
915         difference);
916     assertFalse(testName + " self difference", ensemble == difference);
917     assertFalse(testName + " self difference", other == difference);
918     assertEquals(testName + " taille failed", diffSingleElements.length,
919         difference.cardinal());
920     boolean compare = compareElts2Array(testName, difference,
921         diffSingleElements);
922     assertTrue(testName + " elts compare failed", compare);
923 }
924
925 /**
926  * Test method for {@link ensembles.Ensemble#typeElements()}.
927  */
928 @Test
929 public final void testTypeElements()
930 {
931     String testName = new String(typeName + ".typeElements()");
932     System.out.println(testName);
933
934     assertNotNull(testName + " non null instance failed", ensemble);
935
936     // type elt sur ensemble vide == null
937     assertEquals(testName + " sur ens vide failed", null,
938         ensemble.typeElements());
939
940     // type elt sur ensemble non vide == String
941     ensemble = constructEnsemble(testName, typeEnsemble, listElements);
942     assertNotNull(testName + " non null instance failed", ensemble);
943     assertEquals(testName + " sur ens non vide failed", String.class,
944         ensemble.typeElements());
945 }
946
947 /**
948  * Test method for {@link ensembles.Ensemble#equals(java.lang.Object)}.
949  */
950 @Test
951 public final void testEquals()
952 {
953     String testName = new String(typeName + ".equals(Object)");
954     System.out.println(testName);
955
956     // Equals sur null
957     assertFalse(testName + " sur null failed", ensemble.equals(null));
958
959     // Equals sur this
960     assertTrue(testName + " sur this failed", ensemble.equals(ensemble));
961
962     // Equals sur autre objet
963     assertFalse(testName + " sur Object failed",
964         ensemble.equals(new Object()));
965
966     // remplissage ensemble
967     for (String elt : allSingleElementsSorted)
968     {
969         ensemble.ajout(elt);
970     }
971
972     String[] allsingleElementsShuffle = shuffleElements(allSingleElements);
973
974     for (int i = 0; i < typesEnsemble.length; i++)
975     {
976         Class<? extends Ensemble<String>> otherType = typesEnsemble[i];
977         String otherTypeName = otherType.getSimpleName();
978
979         Ensemble<String> other = constructEnsemble(testName,
980             typesEnsemble[i], null);
981
982         // Equals sur Ensemble mÃame contenu mÃame ordre
983         assertNotNull(testName + " other non null instance failed", other);
984         for (String elt : allSingleElementsSorted)
985         {
986             other.ajout(elt);
987         }
988         assertEquals(testName + " ens identique, ordre identique["

```

mar 11, 16 16:05

## AllEnsembleTest.java

Page 12/14

```

989         + otherTypeName + "] failed", ensemble, other);
990
991         // Equals sur Ensemble mÃame contenu ordre diffÃ@rent
992         other efface();
993         for (String elt : allsingleElementsShuffle)
994         {
995             other.ajout(elt);
996         }
997
998         // ensemble est toujours sorted car construit avec
999         // allSingleElementsSorted
1000         if ((ensemble instanceof EnsembleTri<?>) ^
1001             ¬(other instanceof EnsembleTri<?>))
1002         {
1003             assertFalse(testName + " ens identique, ordre diffÃ@rent["
1004                 + otherTypeName + "] failed", ensemble.equals(other));
1005         }
1006         else
1007         {
1008             assertEquals(testName + " ens identique, ordre diffÃ@rent["
1009                 + otherTypeName + "] failed", ensemble, other);
1010         }
1011
1012         // Equals sur Ensemble contenu diffÃ@rent
1013         other.ajout("bonjour");
1014         assertFalse(testName + " ens diffÃ@rent failed",
1015             ensemble.equals(other));
1016     }
1017 }
1018
1019 /**
1020  * Test method for {@link ensembles.Ensemble#hashCode()}.
1021  */
1022 @Test
1023 public final void testHashCode()
1024 {
1025     String testName = new String(typeName + ".hashCode()");
1026     System.out.println(testName);
1027     int hash;
1028     boolean trie = ensemble instanceof EnsembleTri<?>;
1029     if (trie)
1030     {
1031         hash = 1;
1032     }
1033     else
1034     {
1035         hash = 0;
1036     }
1037
1038     // hash code ensemble vide ==
1039     // 0 pour les Ensemble
1040     // 1 pour les EnsembleTri
1041     assertEquals(testName + " hashcode ens vide failed", hash,
1042         ensemble.hashCode());
1043
1044     // hash code ensemble non vide ==
1045     // somme des hashcode des elts pour les Ensemble
1046     // comme les collections pour les EnsembleTri
1047     for (String elt : allSingleElements)
1048     {
1049         ensemble.ajout(elt);
1050     }
1051     if (trie)
1052     {
1053         final int prime = 31;
1054         for (String elt : allSingleElementsSorted)
1055         {
1056             hash = (prime * hash) + (elt == null ? 0 : elt.hashCode());
1057         }
1058     }
1059     else
1060     {
1061         for (String elt : allSingleElements)
1062         {
1063             hash += elt.hashCode();
1064         }
1065     }
1066
1067     assertEquals(testName + " hashcode ens non vide failed", hash,
1068         ensemble.hashCode());
1069 }
1070
1071 /**
1072  * Test method for {@link ensembles.Ensemble#toString()}.
1073  */
1074 @Test
1075 public final void testToString()
1076 {
1077     String testName = new String(typeName + ".toString()");
1078     System.out.println(testName);
1079 }

```

mar 11, 16 16:05

## AllEnsembleTest.java

Page 13/14

```

1081     ensemble = constructEnsemble(testName, typeEnsemble, listElements);
1082     assertNotNull(testName + " non null instance failed", ensemble);
1083
1084     StringBuilder sb = new StringBuilder();
1085     sb.append("[");
1086     Iterator<String> it = ensemble.iterator();
1087     if (it != null)
1088     {
1089         for (; it.hasNext(); )
1090         {
1091             sb.append(it.next().toString());
1092             if (it.hasNext())
1093             {
1094                 sb.append(",");
1095             }
1096         }
1097         sb.append("]");
1098
1099         String expected = sb.toString();
1100
1101         assertEquals(testName, expected, ensemble.toString());
1102     }
1103     else
1104     {
1105         fail(testName + " null iterator");
1106     }
1107 }
1108
1109 /**
1110  * Test method for {@link ensembles.Ensemble#iterator()}.
1111  */
1112 @Test
1113 public final void testIterator()
1114 {
1115     String testName = new String(typeName + ".iterator()");
1116     System.out.println(testName);
1117
1118     Iterator<String> it = null;
1119
1120     // iterator existe
1121     it = ensemble.iterator();
1122     assertNotNull(testName + " non null instance failed", it);
1123
1124     // iterator sur ens vide n'a pas d'elts Ã itÃ©rer
1125     assertFalse(testName + " !hasNext() sur ens vide failed", it.hasNext());
1126
1127     // remplissage
1128     for (String elt : allSingleElements)
1129     {
1130         ensemble.ajout(elt);
1131     }
1132
1133     it = ensemble.iterator();
1134
1135     // iterator sur ens rempli
1136     assertTrue(testName + " hasNext() sur ens rempli failed", it.hasNext());
1137
1138     String[] array;
1139     if (ensemble instanceof EnsembleTri<?>)
1140     {
1141         array = allSingleElementsSorted;
1142     }
1143     else
1144     {
1145         array = allSingleElements;
1146     }
1147
1148     // comparaison des elts
1149     for (int i = 0; (i < array.length) ^ it.hasNext(); i++)
1150     {
1151         assertEquals(testName + "check elt: " + array[i] + " failed",
1152             array[i], it.next());
1153     }
1154
1155     // plus l'elts Ã itÃ©rer
1156     assertFalse(testName + " !hasNext() fin comparaison failed",
1157         it.hasNext());
1158
1159     // retrait des elts avec l'itÃ©rateur
1160     it = ensemble.iterator();
1161     for (int i = 0; (i < array.length) ^ it.hasNext(); i++)
1162     {
1163         it.next();
1164         it.remove();
1165         assertFalse(testName + " retrait elt: " + array[i] + " failed",
1166             ensemble.contient(array[i]));
1167     }
1168
1169     // plus l'elts Ã itÃ©rer
1170

```

mar 11, 16 16:05

## AllEnsembleTest.java

Page 14/14

```

1171     assertFalse(testName + " !hasNext() fin retrait failed", it.hasNext());
1172     assertTrue(testName + " ens vide aprÃ©s retraits failed",
1173         ensemble.estVide());
1174 }
1175 }

```

oct 08, 15 12:23

## ListeTest.java

Page 1/8

```

1 package tests;
2
3 import static org.junit.Assert.assertEquals;
4 import static org.junit.Assert.assertFalse;
5 import static org.junit.Assert.assertNotNull;
6 import static org.junit.Assert.assertNotSame;
7 import static org.junit.Assert.assertSame;
8 import static org.junit.Assert.assertTrue;
9 import static org.junit.Assert.fail;
10
11 import java.util.ArrayList;
12 import java.util.Iterator;
13 import java.util.NoSuchElementException;
14
15 import org.junit.After;
16 import org.junit.AfterClass;
17 import org.junit.Before;
18 import org.junit.BeforeClass;
19 import org.junit.Test;
20
21 import listes.Liste;
22
23 /**
24  * Classe de test de la liste Chainée
25  * @author davidroussel
26  */
27 public class ListeTest
28 {
29     /**
30      * La liste à tester.
31      * La nature du contenu de la liste importe peu du moment qu'il est
32      * homogène : donc n'importe quel type ferait l'affaire.
33      */
34     private Liste<String> liste = null;
35
36     /**
37      * Liste des éléments à insérer dans la liste
38      */
39     private static String[] elements;
40
41     /**
42      * Mise en place avant l'ensemble des tests
43      * @throws java.lang.Exception
44      */
45     @BeforeClass
46     public static void setUpBeforeClass() throws Exception
47     {
48         System.out.println("-----");
49         System.out.println("Test de la Liste");
50         System.out.println("-----");
51     }
52
53     /**
54      * Nettoyage après l'ensemble des tests
55      * @throws java.lang.Exception
56      */
57     @AfterClass
58     public static void tearDownAfterClass() throws Exception
59     {
60         System.out.println("-----");
61         System.out.println("Fin Test de la Liste");
62         System.out.println("-----");
63     }
64
65     /**
66      * Mise en place avant chaque test
67      * @throws java.lang.Exception
68      */
69     @Before
70     public void setUp() throws Exception
71     {
72         elements = new String[] {
73             "Hello",
74             "Brave",
75             "New",
76             "World"
77         };
78         liste = new Liste<String>();
79     }
80
81     /**
82      * Nettoyage après chaque test
83      * @throws java.lang.Exception
84      */
85     @After
86     public void tearDown() throws Exception
87     {
88         liste.clear();
89         liste = null;
90     }

```

Vendredi mars 10, 2017

oct 08, 15 12:23

## ListeTest.java

Page 2/8

```

91     }
92
93     /**
94      * Méthode utilitaire de remplissage de la liste avec les éléments
95      * du tableau #elements
96      */
97     private final void remplissage()
98     {
99         if (liste != null)
100         {
101             for (String elt : elements)
102             {
103                 liste.add(elt);
104             }
105         }
106     }
107
108     /**
109      * Test method for {@link listes.Liste#Liste()}.
110      */
111     @Test
112     public final void testListe()
113     {
114         String testName = new String("Liste<String>()");
115         System.out.println(testName);
116
117         assertNotNull(testName + " instance non null failed", liste);
118         assertTrue(testName + " liste vide failed", liste.empty());
119     }
120
121     /**
122      * Test method for {@link listes.Liste#Liste(listes.Liste)}.
123      */
124     @Test
125     public final void testListeListeOfT()
126     {
127         String testName = new String("Liste<String>(Liste<String>)");
128         System.out.println(testName);
129
130         Liste<String> liste2 = new Liste<String>();
131         liste = new Liste<String>(liste2);
132
133         assertNotNull(testName + " instance non null failed", liste);
134         assertTrue(testName + " liste vide failed", liste.empty());
135
136         remplissage();
137         assertFalse(testName + " liste remplie failed", liste.empty());
138         liste2 = new Liste<String>(liste);
139         assertNotNull(testName + " copie liste remplie failed", liste2);
140         assertEquals(testName + " contenus Ã©gaux failed", liste, liste2);
141     }
142
143     /**
144      * Test method for {@link listes.Liste#add(java.lang.Object)}.
145      */
146     @Test
147     public final void testAdd()
148     {
149         String testName = new String("Liste<String>.add(E)");
150         System.out.println(testName);
151
152         // Ajout dans une liste vide
153         liste.add(elements[0]);
154         assertFalse(testName + " liste non vide failed", liste.empty());
155         Iterator<String> it = liste.iterator();
156         String insertedElt = it.next();
157         assertEquals(testName + " contrôle ref element[0] failed", insertedElt, elements[0]);
158         // Si assertSame réussit assertEquals n'est plus nécessaire
159
160         // Ajout dans une liste non vide
161         for (int i=1; i < elements.length; i++)
162         {
163             liste.add(elements[i]);
164         }
165
166         /*
167          * Attention le prochain "it" a été invalidé par l'ajout
168          * Lors du dernier next le current de l'itérateur est passé à null
169          * puisqu'il n'avait pas (encore) de suivant. donc retenter un
170          * next sur le même itérateur génèrera un NoSuchElementException.
171          * Il faut donc obtenir un itérateur pour parcourir la liste
172          * après un ajout
173          */
174         it = liste.iterator();
175         for (int j = 0; j ≤ i; j++)
176         {
177             insertedElt = it.next();
178             assertEquals(testName + " contrôle ref element[" + i + "] failed",
179                 insertedElt, elements[i]);
180         }

```

src/tests/ListeTest.java

23/33





oct 08, 15 12:23

## ListeTest.java

Page 5/8

```

361     result = liste.remove("Hello");
362     // liste = Brave -> New -> World
363     assertTrue(testName + " Hello fin failed", result);
364     Iterator<String> it = liste.iterator();
365     it.next(); // Brave
366     it.next(); // New
367     String lastElt = it.next(); // World
368     assertEquals(testName + " Hello fin failed", lastElt, elements[3]);
369
370     // suppression d'un Ã©lément existant en milieu de liste
371     result = liste.remove(elements[2]);
372     // liste = Brave -> World
373     assertTrue(testName + " New milieu failed", result);
374     it = liste.iterator();
375     String firstElt = it.next(); // Brave
376     lastElt = it.next(); // World
377     assertEquals(testName + " first elt left failed", firstElt, elements[1]);
378     assertEquals(testName + " last elt left failed", lastElt, elements[3]);
379 }
380
381 /**
382  * Test method for {@link listes.Liste#removeAll(java.lang.Object)}.
383  */
384 @Test
385 public final void testRemoveAll()
386 {
387     String testName = new String("Liste<String>.removeAll(E)");
388     System.out.println(testName);
389
390     // suppression d'un Ã©lément non null d'une liste vide
391     boolean result = liste.removeAll(elements[0]);
392     assertTrue(testName + " supprTous elt liste vide failed", liste.empty());
393     assertFalse(testName + " supprTous elt liste vide failed", result);
394
395     // suppression d'un Ã©lément null d'une liste vide
396     result = liste.removeAll(null);
397     assertTrue(testName + " supprTous elt null liste vide failed", liste.empty());
398     assertFalse(testName + " supprTous elt null liste vide failed", result);
399
400     elements[2] = new String("Hello");
401     remplissage();
402     liste.add("Hello"); // "Hello" not same as elements[0]
403     // liste = Hello -> Brave -> Hello -> World -> Hello
404
405     // suppression d'un Ã©lément null d'une liste non vide
406     result = liste.removeAll(null);
407     assertFalse(testName + " supprTous elt null liste failed", result);
408
409     // suppression d'un élément existant au début, au milieu et à la fin
410     result = liste.removeAll("Hello");
411     // liste = Brave -> World
412     assertTrue(testName + " supprimeTous Hello", result);
413     Iterator<String> it = liste.iterator();
414     String firstElt = it.next();
415     String lastElt = it.next();
416     assertFalse(testName + " 2 elts left failed", it.hasNext());
417     assertEquals(testName + " first elt left failed", firstElt, elements[1]);
418     assertEquals(testName + " last elt left failed", lastElt, elements[3]);
419 }
420
421 /**
422  * Test method for {@link listes.Liste#size()}.
423  */
424 @Test
425 public final void testSize()
426 {
427     String testName = new String("Liste<String>.size()");
428     System.out.println(testName);
429
430     // taille d'une liste vide
431     assertTrue(testName + " taille liste vide failed", liste.size() == 0);
432
433     remplissage();
434     assertFalse(testName + " remplissage failed", liste.empty());
435
436     // taille d'une liste non vide
437     assertTrue(testName + " taille liste pleine failed",
438         liste.size() == elements.length);
439 }
440
441 /**
442  * Test method for {@link listes.Liste#get(int)}.
443  */
444 @Test
445 public final void testGet()
446 {
447     String testName = new String("Liste<String>.get(int)");
448     System.out.println(testName);
449 }
450 // get sur une liste vide

```

Vendredi mars 10, 2017

src/tests/ListeTest.java

oct 08, 15 12:23

## ListeTest.java

Page 6/8

```

451 // assertTrue(testName + " get liste vide failed", liste.get(0) == null);
452 // assertTrue(testName + " get liste vide failed", liste.get(-1) == null);
453
454 // remplissage()
455 // assertFalse(testName + " remplissage failed", liste.empty());
456
457 // get dans une liste non vide
458 // for (int i = -1; i <= liste.size(); i++)
459 // {
460 //     if ((i >= 0) && (i < liste.size()))
461 //     {
462 //         assertNotNull(testName + " get(" + i + ") liste pleine failed",
463 //             liste.get(i));
464 //         assertTrue(testName + " get(" + i + ") liste pleine failed",
465 //             liste.get(i).equals(elements[i]));
466 //     }
467 //     else
468 //     {
469 //         assertTrue(testName + " get(" + i + ") liste pleine failed",
470 //             liste.get(i) == null);
471 //     }
472 // }
473 // }
474
475 /**
476  * Test method for {@link listes.Liste#clear()}.
477  */
478 @Test
479 public final void testClear()
480 {
481     String testName = new String("Liste<String>.clear()");
482     System.out.println(testName);
483
484     // effacement d'une liste vide
485     liste.clear();
486     assertTrue(testName + " effacement liste vide failed", liste.empty());
487
488     remplissage();
489     assertFalse(testName + " remplissage failed", liste.empty());
490
491     // effacement d'une liste non vide
492     liste.clear();
493     assertTrue(testName + " effacement failed", liste.empty());
494 }
495
496 /**
497  * Test method for {@link listes.Liste#empty()}.
498  */
499 @Test
500 public final void testEmpty()
501 {
502     String testName = new String("Liste<String>.empty()");
503     System.out.println(testName);
504
505     assertTrue(testName + " vide failed", liste.empty());
506
507     remplissage();
508
509     assertFalse(testName + " non vide failed", liste.empty());
510 }
511
512 /**
513  * Test method for {@link listes.Liste#equals(java.lang.Object)}.
514  */
515 @Test
516 public final void testEqualsObject()
517 {
518     String testName = new String("Liste<String>.equals(Object)");
519     System.out.println(testName);
520
521     remplissage();
522
523     // Inegalite sur objet null
524     boolean result = liste.equals(null);
525     assertFalse(testName + " null object failed", result);
526
527     // Egalite sur soi-même
528     result = liste.equals(liste);
529     assertTrue(testName + " self failed", result);
530
531     // Egalite sur liste copiée
532     Liste<String> liste2 = new Liste<String>(liste);
533     result = liste.equals(liste2);
534     assertTrue(testName + " copy failed", result);
535
536     // Inegalite sur listes de tailles différentes
537     liste2.add("of Pain");
538     result = liste.equals(liste2);
539     assertFalse(testName + " copy + of Pain failed", result);
540 }

```

25/33

oct 08, 15 12:23

## ListeTest.java

Page 7/8

```

541 // Inegalite sur liste Ã contenu dans une autre ordre
542 liste2.clear();
543 for (String elt : elements)
544 {
545     liste2.insert(elt);
546 }
547 result = liste.equals(liste2);
548 assertFalse(testName + " reversed copy failed", result);
549
550 // Egalite avec une collection standard de mÃame contenu
551 // SSI equals compare un Iterable plut qu'une Liste
552 ArrayList<String> alist = new ArrayList<String>();
553 for (String elt : elements)
554 {
555     alist.add(elt);
556 }
557 assertTrue(testName + " equality with std Iterable failed",
558             liste.equals(alist));
559
560 /**
561  * Test method for {@link listes.Liste#toString()}.
562  */
563 @Test
564 public final void testToString()
565 {
566     String testName = new String("Liste<String>.toString()");
567     System.out.println(testName);
568
569     remplissage();
570
571     assertEquals(testName, "[Hello->Brave->New->World]", liste.toString());
572 }
573
574 /**
575  * Test method for {@link listes.Liste#iterator()}.
576  */
577 @Test(expected = NoSuchElementException.class)
578 public final void testIterator()
579 {
580     String testName = new String("Liste<String>.iterator()");
581     System.out.println(testName);
582
583     Iterator<String> it = liste.iterator();
584     assertFalse(testName + " liste vide", it.hasNext());
585
586     remplissage();
587
588     it = liste.iterator();
589     assertTrue(testName + " liste non vide", it.hasNext());
590
591     int i = 0;
592     while (it.hasNext())
593     {
594         String nextElt = it.next();
595         assertNotNull(testName + "next elt not null", nextElt);
596         assertEquals(testName + "next elt", elements[i++], nextElt);
597         it.remove(); // ne doit pas invalider l'itÃrateur
598     }
599
600     assertFalse(testName + " finished", it.hasNext());
601
602     // Un appel supplÃmentaire Ã next sur un itÃrateur terminÃ
603     // doit soulever une NoSuchElementException
604     it.next();
605
606     fail(testName + " next sur itÃrateur terminÃ");
607 }
608
609 /**
610  * Test method for {@link listes.Liste#hashCode()}.
611  */
612 @Test
613 public final void testHashCode()
614 {
615     String testName = new String("Liste<String>.hashCode()");
616     System.out.println(testName);
617
618     // hashCode d'une liste vide = 1
619     int listeHash = liste.hashCode();
620     assertEquals(testName + " liste vide failed", 1, listeHash, 0);
621
622     remplissage();
623
624     // hashCode de la liste standard
625     listeHash = liste.hashCode();
626     assertEquals(testName + " liste standard failed", 1161611233, listeHash);
627
628     /*
629     * Contrat hashCode : Si a.equals(b) alors a.hashCode() == b.hashCode()
630     */

```

oct 08, 15 12:23

## ListeTest.java

Page 8/8

```

631 */
632 Liste<String> liste2 = new Liste<String>(liste);
633 assertNotSame(testName + " egalite liste distinctes failed", liste, liste2);
634 assertEquals(testName + " egalite liste equals failed", liste, liste2);
635 assertEquals(testName + " egalite liste hashCode failed", liste.hashCode(),
636             liste2.hashCode(), 0);
637
638 liste2.add("Hourra");
639 assertFalse(testName + " inegalite liste equals failed", liste.equals(liste2));
640 assertFalse(testName + " inegalite liste hashCode failed",
641             liste.hashCode() == liste2.hashCode());
642
643 // HashCode similaire Ã celui d'une collection standard
644 ArrayList<String> collection = new ArrayList<String>();
645 for (String elt : elements)
646 {
647     collection.add(elt);
648 }
649 int collectionHash = collection.hashCode();
650 assertEquals(testName + " hashCode standard failed", listeHash, collectionHash);
651 }
652 }

```

oct 20, 14 17:22

## TableauTest.java

Page 1/7

```

1 package tests;
2
3 import static org.junit.Assert.assertEquals;
4 import static org.junit.Assert.assertFalse;
5 import static org.junit.Assert.assertNotNull;
6 import static org.junit.Assert.assertTrue;
7 import static org.junit.Assert.fail;
8
9 import java.util.ArrayList;
10 import java.util.Collections;
11 import java.util.Iterator;
12
13 import org.junit.After;
14 import org.junit.AfterClass;
15 import org.junit.Before;
16 import org.junit.BeforeClass;
17 import org.junit.Test;
18
19 import tableaux.Tableau;
20
21 /**
22  * Classe de teste de la classe {@link tableaux.Iterable}
23  * @author davidroussel
24  */
25 public class TableauTest
26 {
27     /**
28      * Le tableau à tester
29      */
30     private Tableau<String> tableau;
31
32     /**
33      * Des éléments pour remplir le tableau
34      * Le nombre d'éléments doit être supérieur à {@link Iterable#INCREMENT}
35      */
36     private final static String[] elementsArray = new String[] {
37         "Hello",
38         "Brave",
39         "New",
40         "World",
41         "of",
42         "Pain"
43     };
44
45     /**
46      * Une collection standard pour comparer avec le tableau
47      */
48     private ArrayList<String> elementsCollection;
49
50     /**
51      * Mise en place avant l'ensemble des tests
52      * @throws java.lang.Exception
53      */
54     @BeforeClass
55     public static void setUpBeforeClass() throws Exception
56     {
57         System.out.println("-----");
58         System.out.println("Test du Tableau");
59         System.out.println("-----");
60     }
61
62     /**
63      * Nettoyage après l'ensemble des tests
64      * @throws java.lang.Exception
65      */
66     @AfterClass
67     public static void tearDownAfterClass() throws Exception
68     {
69         System.out.println("-----");
70         System.out.println("Fin Test du Tableau");
71         System.out.println("-----");
72     }
73
74     /**
75      * Mise en place avant chaque test
76      * @throws java.lang.Exception
77      */
78     @Before
79     public void setUp() throws Exception
80     {
81         tableau = new Tableau<String>();
82         elementsCollection = new ArrayList<String>();
83         for (String elt : elementsArray)
84         {
85             elementsCollection.add(elt);
86         }
87     }
88
89     /**
90

```

oct 20, 14 17:22

## TableauTest.java

Page 2/7

```

91     * Nettoyage après chaque test
92     * @throws java.lang.Exception
93     */
94     @After
95     public void tearDown() throws Exception
96     {
97         tableau efface();
98         tableau = null;
99         elementsCollection.clear();
100     }
101
102     /**
103      * Comparaison des éléments de deux Iterables
104      * @param testName le nom du test dans lequel est appelée cette méthode
105      * @param i1 le premier iterable à tester
106      * @param i2 le second iterable avec lequel comparer
107      * @return true si les deux iterables possèdent le même nombre
108      * d'éléments et que tous les éléments sont identiques et dans le même ordre
109      */
110     private boolean compareElements(String testName,
111                                     Iterable<String> i1,
112                                     Iterable<String> i2)
113     {
114         Iterator<String> it1 = i1.iterator();
115         Iterator<String> it2 = i2.iterator();
116
117         for (; it1.hasNext() & it2.hasNext(); )
118         {
119             String s1 = it1.next();
120             String s2 = it2.next();
121
122             assertEquals(testName + "compare " + s1 + " with " + s2, s1, s2);
123
124             if (!s1.equals(s2))
125             {
126                 return false;
127             }
128         }
129
130         return !it1.hasNext() & !it2.hasNext();
131     }
132
133     /**
134      * Liste d'indices d'index compris entre 0 et nbElements - 1;
135      *
136      * @param nbElements le nombre d'indices
137      * @return un tableau contenant nbElements éléments compris entre
138      * [0..nbElements-1] et mis dans un ordre aléatoire
139      */
140     private int[] shuffledIndexes(int nbElements)
141     {
142         int[] shuffled = new int[nbElements];
143
144         ArrayList<Integer> list = new ArrayList<Integer>();
145         for (int i = 0; i < nbElements; i++)
146         {
147             list.add(Integer.valueOf(i));
148         }
149
150         Collections.shuffle(list);
151
152         Iterator<Integer> il = list.iterator();
153         for (int i = 0; (i < nbElements) & il.hasNext(); i++)
154         {
155             shuffled[i] = il.next().intValue();
156         }
157
158         return shuffled;
159     }
160
161     /**
162      * Test method for {@link tableaux.Iterable#Iterable()}.
163      */
164     @Test
165     public final void testTableau()
166     {
167         String testName = new String("Tableau()");
168         System.out.println(testName);
169
170         assertNotNull(testName + " instance", tableau);
171         assertEquals(testName + " tableau vide", tableau.taille(), 0);
172     }
173
174     /**
175      * Test method for {@link tableaux.Iterable#Iterable(java.lang.Iterable)}.
176      */
177     @Test
178     public final void testTableauIterableOfE()
179

```

oct 20, 14 17:22

## TableauTest.java

Page 3/7

```

181 {
182     String testName = new String("Tableau(Iterable<E>)");
183     System.out.println(testName);
184
185     tableau = new Tableau<String>(elementsCollection);
186
187     assertNotNull(testName + " instance", tableau);
188     assertEquals(testName + " tableau non vide", tableau.taille(),
189         elementsCollection.size());
190
191     boolean compare = compareElements(testName, tableau, elementsCollection);
192
193     assertTrue(testName + " elements comparison result", compare);
194 }
195
196 /**
197  * Test method for {@link tableaux.Iterable#taille()}.
198  */
199 @Test
200 public final void testTaille()
201 {
202     String testName = new String("Tableau.taille()");
203     System.out.println(testName);
204
205     assertEquals(testName + " tableau vide", tableau.taille(), 0);
206     int taille = 0;
207     for (String elt : elementsArray)
208     {
209         tableau.ajouter(elt);
210         taille++;
211         assertEquals(testName + " tableau[" + taille + "]",
212             tableau.taille(), taille);
213     }
214
215     tableau.efface();
216     assertEquals(testName + " tableau nettoy  ", tableau.taille(), 0);
217 }
218
219 /**
220  * Test method for {@link tableaux.Iterable#capacite()}.
221  */
222 @Test
223 public final void testCapacite()
224 {
225     String testName = new String("Tableau.capacite()");
226     System.out.println(testName);
227     int predictedCapacity = 0;
228
229     assertEquals(testName + "capacite tableau vide", tableau.capacite(),
230         predictedCapacity);
231
232     int nb = 0;
233     for (String elt : elementsArray)
234     {
235         nb++;
236         if (nb > tableau.capacite())
237         {
238             predictedCapacity += Tableau.INCREMENT;
239         }
240         tableau.ajouter(elt);
241         assertEquals(testName + " tableau[" + nb + "]",
242             tableau.capacite(), predictedCapacity);
243     }
244 }
245
246 /**
247  * Test method for {@link tableaux.Iterable#ajouter(java.lang.Object)}.
248  */
249 @Test
250 public final void testAjouter()
251 {
252     String testName = new String("Tableau.ajouter(E)");
253     System.out.println(testName);
254     int predictedSize = 0;
255
256     for (String elt : elementsArray)
257     {
258         tableau.ajouter(elt);
259         predictedSize++;
260
261         String lastElement = null;
262         for (Iterator<String> itt = tableau.iterator(); itt.hasNext();)
263         {
264             lastElement = itt.next();
265         }
266
267         assertEquals(testName + " size", predictedSize, tableau.taille());
268         assertEquals(testName + "last elt comparison", elt, lastElement);
269     }
270 }

```

Vendredi mars 10, 2017

src/tests/TableauTest.java

oct 20, 14 17:22

## TableauTest.java

Page 4/7

```

271 }
272
273 /**
274  * Test method for {@link tableaux.Iterable#retrait(java.lang.Object)}.
275  */
276 @Test
277 public final void testRetrait()
278 {
279     String testName = new String("Tableau.retrait(E)");
280     System.out.println(testName);
281
282     tableau = new Tableau<String>(elementsCollection);
283     int nbElements = elementsArray.length;
284     int nbElementsLeft = nbElements;
285
286     boolean result = compareElements(testName, tableau, elementsCollection);
287     assertTrue(testName + " no more elts to compare", result);
288     // on va retirer des elts de tableau et elementsCollection dans un
289     // ordre al  atoire
290     int[] indexes = shuffledIndexes(nbElements);
291
292     for (int i = 0; i < nbElements; i++)
293     {
294         tableau.retrait(elementsArray[indexs[i]]);
295         elementsCollection.remove(elementsArray[indexs[i]]);
296         nbElementsLeft = elementsCollection.size();
297
298         result = compareElements(testName, tableau, elementsCollection);
299         assertTrue(testName + nbElementsLeft + "elts compared", result);
300     }
301 }
302
303 /**
304  * Test method for {@link tableaux.Iterable#efface()}.
305  */
306 @Test
307 public final void testEfface()
308 {
309     String testName = new String("Tableau.efface()");
310     System.out.println(testName);
311
312     tableau = new Tableau<String>(elementsCollection);
313
314     assertTrue(testName + "tableau initial non vide", tableau.taille() > 0);
315
316     tableau.efface();
317
318     assertEquals(testName + "tableau final vide", tableau.taille(), 0);
319     Iterator<String> it = tableau.iterator();
320     assertFalse(testName + " pas d'elts    it  rer", it.hasNext());
321 }
322
323 /**
324  * Test method for {@link tableaux.Iterable#insertElement(java.lang.Object)}.
325  */
326 @Test
327 public final void testInsertElementE()
328 {
329     String testName = new String("Tableau.insertElement(E)");
330     System.out.println(testName);
331
332     for (String elt : elementsArray)
333     {
334         tableau.insertElement(elt);
335
336         Iterator<String> it = tableau.iterator();
337         assertEquals(testName + " first elt compare", elt, it.next());
338     }
339 }
340
341 /**
342  * Test method for {@link tableaux.Iterable#insertElement(java.lang.Object, int)}.
343  * Ajout    un index invalide dans une collection vide
344  */
345 @Test(expected = IndexOutOfBoundsException.class)
346 public final void testInsertElementEIntInvalidEmpty()
347 {
348     String testName = new String("Tableau.insertElement(E, int)");
349     System.out.println(testName);
350
351     tableau.insertElement("Bonjour", 1);
352
353     fail(testName + " Ajout ds tableau vide    index invalide r  ussi !");
354 }
355
356 /**
357  * Test method for {@link tableaux.Iterable#insertElement(java.lang.Object, int)}.
358  * Ajout    un index invalide dans une collection pleine
359  */
360 @Test(expected = IndexOutOfBoundsException.class)

```

28/33

oct 20, 14 17:22

## TableauTest.java

Page 5/7

```

361 public final void testInsertElementEIntInvalidFull()
362 {
363     String testName = new String("Tableau.insertElement(E, int)");
364     System.out.println(testName);
365
366     tableau = new Tableau<String>(elementsCollection);
367
368     tableau.insertElement("Bonjour", tableau.taille() + 1);
369
370     fail(testName + " Ajout ds tableau plein Ã index invalide rÃoussi !");
371 }
372
373 /**
374  * Test method for {@link tableaux.Iterable#insertElement(java.lang.Object, int)}.
375  */
376 @Test
377 public final void testInsertElementEInt()
378 {
379     String testName = new String("Tableau.insertElement(E, int)");
380     System.out.println(testName);
381     int nbElements = elementsArray.length;
382     elementsCollection.clear();
383     int currentSize = 0;
384     boolean result = false;
385
386     // Ajouts en dÃebut et fin
387     for (int i = 0; i < (nbElements / 2); i++)
388     {
389         // Ajout au dÃebut
390         tableau.insertElement(elementsArray[i], 0);
391         elementsCollection.add(0, elementsArray[i]);
392
393         currentSize = elementsCollection.size();
394
395         result = compareElements(testName, tableau, elementsCollection);
396         assertTrue(testName + " after push front", result);
397
398         // Ajout Ã la fin
399         int sourceIdx = nbElements-1-i;
400         tableau.insertElement(elementsArray[sourceIdx], currentSize);
401         elementsCollection.add(currentSize, elementsArray[sourceIdx]);
402
403         result = compareElements(testName, tableau, elementsCollection);
404         assertTrue(testName + " after push back", result);
405     }
406
407     currentSize = elementsCollection.size();
408
409     // Ajout au milieu
410     String extraElement = "Bonjour";
411     tableau.insertElement(extraElement, currentSize/2);
412     elementsCollection.add(currentSize/2, extraElement);
413
414     result = compareElements(testName, tableau, elementsCollection);
415     assertTrue(testName + " after push middle", result);
416 }
417
418 /**
419  * Test method for {@link tableaux.Iterable#iterator()}.
420  */
421 @Test
422 public final void testIterator()
423 {
424     String testName = new String("Tableau.iterator()");
425     System.out.println(testName);
426
427     // itÃrateur sur tableau vide
428     Iterator<String> itt = tableau.iterator();
429     assertFalse(testName + " itÃrateur sur tableau vide", itt.hasNext());
430
431     // itÃrateur su tableau rempli
432     tableau = new Tableau<String>(elementsCollection);
433     boolean result = compareElements(testName, tableau, elementsCollection);
434     assertTrue(testName, result);
435
436     // utilisation du remove sans next
437     for (itt = tableau.iterator(); itt.hasNext(); )
438     {
439         try
440         {
441             itt.remove();
442             fail(testName + " remove utilisÃ avec succÃs sans next dans boucle");
443         }
444         catch (IllegalStateException ise)
445         {
446             // rien, c'est normal
447         }
448         itt.next();
449         itt.remove();
450     }

```

Vendredi mars 10, 2017

src/tests/TableauTest.java

oct 20, 14 17:22

## TableauTest.java

Page 6/7

```

451     assertFalse(testName + " iterator terminÃ fin boucle", itt.hasNext());
452     assertEquals(testName + " tableau vide avec suite remove", 0,
453         tableau.taille());
454
455     }
456
457     /**
458     * Test method for {@link tableaux.Iterable#equals(java.lang.Object)}.
459     */
460     @Test
461     public final void testEqualsObject()
462     {
463         String testName = new String("Tableau.equals(Object)");
464         System.out.println(testName);
465
466         // Inegalite avec null
467         boolean result = tableau.equals(null);
468         assertFalse(testName + " inequality with null", result);
469
470         // Egalite avec this
471         assertTrue(testName + " self equality", tableau.equals(tableau));
472
473         // Egalite avec une copie de soi mÃme (vide)
474         Tableau<String> other = new Tableau<String>(tableau);
475         assertTrue(testName + " equality with copy", tableau.equals(other));
476
477         // Inegalite avec tableau de contenu diffÃrent
478         for (String elt : elementsArray)
479         {
480             tableau.ajouter(elt);
481
482             assertFalse(testName + " content inequality", tableau.equals(other));
483
484             // Egalite sur contenus identiques
485             for (String elt : elementsArray)
486             {
487                 other.ajouter(elt);
488
489                 assertTrue(testName + " content equality", tableau.equals(other));
490
491                 // Inegalite avec un objet quelconque
492                 assertFalse(testName + " type inequality", tableau.equals(new Object()));
493
494                 // Inegalite avec un autre Iterable
495                 assertFalse(testName + " inequality with Iterable",
496                     tableau.equals(elementsCollection));
497             }
498         }
499
500         /**
501         * Test method for {@link tableaux.Iterable#hashCode()}.
502         */
503         @Test
504         public final void testHashCode()
505         {
506             String testName = new String("Tableau.hashCode()");
507             System.out.println(testName);
508
509             // Hash code sur tableau vide
510             assertEquals(testName + " empty tableau", 1, tableau.hashCode());
511
512             tableau = new Tableau<String>(elementsCollection);
513
514             // Hash code sur tableau rempli Ãgal au hascode des collections standard
515             assertEquals(testName + " full tableau", tableau.hashCode(),
516                 elementsCollection.hashCode());
517         }
518
519         /**
520         * Test method for {@link tableaux.Iterable#toString()}.
521         */
522         @Test
523         public final void testToString()
524         {
525             String testName = new String("Tableau.toString()");
526             System.out.println(testName);
527
528             tableau = new Tableau<String>(elementsCollection);
529
530             StringBuilder sb = new StringBuilder();
531             sb.append("[");
532             for (Iterator<String> it = tableau.iterator(); it.hasNext(); )
533             {
534                 sb.append(it.next().toString());
535                 if (it.hasNext())
536                 {
537                     sb.append(", ");
538                 }
539             }
540             sb.append("]");
541             sb.append(Integer.toString(tableau.taille()));

```

29/33

oct 20, 14 17:22

## TableauTest.java

Page 7/7

```

541     sb.append(",");
542     sb.append(Integer.toString(tableau.capacite()));
543     sb.append("\n");
544     String expected = sb.toString();
545
546     assertEquals(testName, expected, tableau.toString());
547
548 }
549
550 }
```

nov 04, 15 18:18

## EnsembleTriTest.java

Page 1/6

```

1  package tests;
2
3  import static org.junit.Assert.assertEquals;
4  import static org.junit.Assert.assertFalse;
5  import static org.junit.Assert.assertNotNull;
6  import static org.junit.Assert.assertTrue;
7  import static org.junit.Assert.fail;
8
9  import java.lang.reflect.InvocationTargetException;
10 import java.util.ArrayList;
11 import java.util.Arrays;
12 import java.util.Collection;
13 import java.util.HashMap;
14 import java.util.Iterator;
15 import java.util.Map;
16
17 import org.junit.After;
18 import org.junit.AfterClass;
19 import org.junit.Before;
20 import org.junit.BeforeClass;
21 import org.junit.Test;
22 import org.junit.runner.RunWith;
23 import org.junit.runners.Parameterized;
24 import org.junit.runners.Parameterized.Parameters;
25
26 import ensembles.EnsembleTri;
27 import ensembles.EnsembleTriFactory;
28 import ensembles.EnsembleTriTableau;
29
30 /**
31  * Classe de test compl  mentaire pour tous les types d'ensembles tri  s.
32  * {@link ensembles.EnsembleTriVector}. {@link ensembles.EnsembleTriVector2},
33  * {@link ensembles.EnsembleTriListe}. {@link ensembles.EnsembleTriListe2}.
34  * {@link ensembles.EnsembleTriTableau}, {@link ensembles.EnsembleTriTableau2}
35  * @author davidroussel
36  */
37 @RunWith(value = Parameterized.class)
38 public class EnsembleTriTest
39 {
40     /**
41      * L'ensemble    tester
42      */
43     private EnsembleTri<String> ensemble;
44
45     /**
46      * Le type d'ensemble    tester.
47      */
48     private Class<? extends EnsembleTri<String>> typeEnsemble;
49
50     /**
51      * Nom du type d'ensemble    tester
52      */
53     private String typeName;
54
55     /**
56      * Les diff  rentes natures d'ensembles    tester
57      */
58     @SuppressWarnings("unchecked")
59     private static final Class<? extends EnsembleTri<String>>[] typesEnsemble =
60         (Class<? extends EnsembleTri<String>>[]) new Class<?>[]
61         {
62             /*
63              * TODO Commenter / d  commenter les lignes ci-dessous en fonction
64              * de votre avancement (Attention la derni  re ligne non comment  e
65              * ne doit pas avoir de virgule)
66              */
67             EnsembleTriVector.class,
68             EnsembleTriVector2.class,
69             EnsembleTriTableau.class,
70             EnsembleTriTableau2.class,
71             EnsembleTriListe.class,
72             EnsembleTriListe2.class
73         };
74
75     /**
76      * Elements pour remplir l'ensemble
77      */
78     private static final String[] elements = new String[] {
79         "Lorem", // 0
80         "ipsum", // 6
81         "sit", // 7
82         "dolor", // 4
83         "amet", // 2
84         "dolor", // 4
85         "amet", // 2
86         "consectetur", // 3
87         "adipiscing", // 1
88         "elit", // 5
89     };
90 }
```

nov 04, 15 18:18

## EnsembleTriTest.java

Page 2/6

```

91  /**
92   * Rang d'insertion des Ã©lÃ©ments successifs
93   */
94   private static final int[] insertionRank = new int[] {
95       0, // Lorem
96       1, // ipsum
97       2, // sit
98       1, // dolor
99       1, // amet
100      2, // dolor
101      1, // amet
102      2, // consectetur
103      1, // adipiscing
104      5, // elit
105  };
106  /**
107   * Elements triÃ©s pour contrÃªler le remplissage de l'ensemble
108   */
109   private static final String[] singleSortedElements = new String[] {
110       "Lorem", // 0
111       "adipiscing", // 1
112       "amet", // 2
113       "consectetur", // 3
114       "dolor", // 4
115       "elit", // 5
116       "ipsum", // 6
117       "sit" // 7
118  };
119  /**
120   * Elements triÃ©s pour contrÃªler le remplissage de l'ensemble
121   */
122   private static final String[][] insertSortedElements = new String[][] {
123       {"Lorem"},
124       {"Lorem", "ipsum"},
125       {"Lorem", "ipsum", "sit"},
126       {"Lorem", "dolor", "ipsum", "sit"},
127       {"Lorem", "amet", "dolor", "ipsum", "sit"},
128       {"Lorem", "amet", "dolor", "ipsum", "sit"},
129       {"Lorem", "amet", "dolor", "ipsum", "sit"},
130       {"Lorem", "amet", "dolor", "ipsum", "sit"},
131       {"Lorem", "amet", "consectetur", "dolor", "ipsum", "sit"},
132       {"Lorem", "adipiscing", "amet", "consectetur", "dolor", "ipsum", "sit"},
133       singleSortedElements
134  };
135  /**
136   * Collection pour contenir les Ã©lÃ©ments de remplissage
137   */
138   private ArrayList<String> listElements;
139
140  /**
141   * Construit une instance de EnsembleTri<String> en fonction d'un type
142   * d'ensemble Ã©crÃ© et Ã©ventuellement d'un contenu l'ensemble Ã© mettre en
143   * place
144   *
145   * @param testName le message Ã© Ã©mettre dans les assertions en fonction du
146   * test dans lequel est employÃ©e cette mÃ©thode
147   * @param type le type d'ensemble Ã© Ã©crÃ©
148   * @param content le contenu Ã© mettre en place dans le nouvel ensemble, ou
149   * bien null si aucun contenu n'est requis.
150   * @return un nouvel ensemble du type demandÃ© evt rempli avec le contenu
151   * fournit s'il est non null.
152   */
153   private static EnsembleTri<String>
154   constructEnsemble(String testName,
155       Class<? extends EnsembleTri<String>> type,
156       Iterable<String> content)
157   {
158       EnsembleTri<String> ensemble = null;
159
160       try
161       {
162           ensemble = EnsembleTriFactory.<String>getEnsemble(type, content);
163       }
164       catch (SecurityException e)
165       {
166           fail(testName + " constructor security exception");
167       }
168       catch (NoSuchMethodException e)
169       {
170           fail(testName + " constructor not found");
171       }
172       catch (IllegalArgumentException e)
173       {
174           fail(testName + " wrong constructor arguments");
175       }
176       catch (InstantiationException e)
177       {
178           fail(testName + " instantiation exception");
179       }
180  }

```

Vendredi mars 10, 2017

src/tests/EnsembleTriTest.java

nov 04, 15 18:18

## EnsembleTriTest.java

Page 3/6

```

181     catch (IllegalAccessException e)
182     {
183         fail(testName + " illegal access");
184     }
185     catch (InvocationTargetException e)
186     {
187         fail(testName + " invocation exception");
188     }
189
190     return ensemble;
191 }
192
193 /**
194  * Compare les Ã©lÃ©ments d'un ensemble pour vÃ©rifier qu'ils sont tous dans
195  * un tableau donnÃ© et dans le mÃªme ordre
196  * @param testName le nom du test dans lequel est utilisÃ© cette mÃ©thode
197  * @param ensemble l'ensemble dont on doit comparer les Ã©lÃ©ments
198  * @param array le tableau utilisÃ© pour vÃ©rifier la prÃ©sence des Ã©lÃ©ments
199  * de l'ensemble
200  * @return true si tous les Ã©lÃ©ments du tableau sont prÃ©sents dans l'ensemble
201  * et dans le mÃªme ordre
202  */
203   private static boolean compareElts2Array(String testName,
204       EnsembleTri<String> ensemble, String[] array)
205   {
206       Iterator<String> ite = ensemble.iterator();
207
208       if (ite != null)
209       {
210           for (int i = 0; (i < array.length) ^ ite.hasNext(); i++)
211           {
212               String ensembleElt = ite.next();
213               String arrayElt = array[i];
214               boolean check = ensembleElt.equals(arrayElt);
215               assertTrue(testName + "[" + i + "]=" + arrayElt + "==" +
216                   ensembleElt + " failed", check);
217               if (!check)
218               {
219                   return false;
220               }
221           }
222           return true;
223       }
224       else
225       {
226           return false;
227       }
228   }
229
230  /**
231   * VÃ©rifie qu'un ensemble ne contient qu'un seul exemplaire de chacun
232   * de ses Ã©lÃ©ments
233   * @param testName le nom du test dans lequel est employÃ©e cette mÃ©thode
234   * @param ensemble l'ensemble Ã© tester
235   * @return true si chaque Ã©lÃ©ment de l'ensemble n'existe qu'Ã© un seul
236   * exemplaire.
237   */
238   private static <E extends Comparable<E>>
239   boolean checkCount(String testName, EnsembleTri<E> ensemble)
240   {
241       Map<E, Integer> wordCount = new HashMap<E, Integer>();
242       for (E elt : ensemble)
243       {
244           if (!wordCount.containsKey(elt))
245           {
246               wordCount.put(elt, Integer.valueOf(1));
247           }
248           else
249           {
250               Integer count = wordCount.get(elt);
251               count = Integer.valueOf(count.intValue() + 1);
252               wordCount.put(elt, count);
253           }
254       }
255
256       for (Integer i : wordCount.values())
257       {
258           int countValue = i.intValue();
259           assertEquals(testName + " count check #" + countValue + " failed",
260               1, countValue);
261           if (countValue != 1)
262           {
263               return false;
264           }
265       }
266
267       return true;
268   }
269
270  /**

```

31/33

nov 04, 15 18:18

## EnsembleTriTest.java

Page 4/6

```

271 * ParamÃtres Ã transmettre au constructeur de la classe de test.
272
273 * @return une collection de tableaux d'objet contenant les paramÃtres Ã
274 * transmettre au constructeur de la classe de test
275 */
276 @Parameters(name = "{index}:{1}")
277 public static Collection<Object[]> data()
278 {
279     Object[][] data = new Object[typesEnsemble.length][2];
280     for (int i = 0; i < typesEnsemble.length; i++)
281     {
282         data[i][0] = typesEnsemble[i];
283         data[i][1] = typesEnsemble[i].getSimpleName();
284     }
285     return Arrays.asList(data);
286 }
287
288 /**
289 * Constructeur paramÃtrÃ par le type d'ensemble Ã tester.
290 * LancÃ pour chaque test
291 * @param typeEnsemble le type d'ensemble Ã tester
292 * @param nom le nom du type d'ensemble Ã tester (pour le faire apparaÃtre
293 * dans le dÃroulement des tests).
294 */
295 public EnsembleTriTest(Class<? extends EnsembleTri<String>> typeEnsemble,
296 String typeEnsembleName)
297 {
298     this.typeEnsemble = typeEnsemble;
299     typeName = typeEnsembleName;
300 }
301
302 /**
303 * Mise en place avant l'ensemble des tests
304 * @throws java.lang.Exception
305 */
306 @BeforeClass
307 public static void setUpBeforeClass() throws Exception
308 {
309     System.out.println("-----");
310     System.out.println("Test des ensembles triÃs");
311     System.out.println("-----");
312 }
313
314 /**
315 * Nettoyage aprÃs l'ensemble des tests
316 * @throws java.lang.Exception
317 */
318 @AfterClass
319 public static void tearDownAfterClass() throws Exception
320 {
321     System.out.println("-----");
322     System.out.println("Fin Test des ensembles triÃs");
323     System.out.println("-----");
324 }
325
326 /**
327 * Mise en place avant chaque test
328 * @throws java.lang.Exception
329 */
330 @Before
331 public void setUp() throws Exception
332 {
333     ensemble = constructEnsemble("setUp", typeEnsemble, null);
334     assertNotNull("setUp non null instance failed", ensemble);
335
336     listElements = new ArrayList<String>();
337
338     for (String elt : elements)
339     {
340         listElements.add(elt);
341     }
342 }
343
344 /**
345 * Nettoyage aprÃs chaque test
346 * @throws java.lang.Exception
347 */
348 @After
349 public void tearDown() throws Exception
350 {
351     ensemble.efface();
352     ensemble = null;
353     listElements.clear();
354     listElements = null;
355 }
356
357 /**
358 * Test method for
359 * {@link ensembles.EnsembleTriVector#EnsembleTriVector()} or
360 * {@link ensembles.EnsembleTriVector2#EnsembleTriVector2()} or

```

Vendredi mars 10, 2017

src/tests/EnsembleTriTest.java

nov 04, 15 18:18

## EnsembleTriTest.java

Page 5/6

```

361 * {@link ensembles.EnsembleTriListe#EnsembleTriListe()} or
362 * {@link ensembles.EnsembleTriListe2#EnsembleTriListe2()} or
363 * {@link ensembles.EnsembleTriTableau#EnsembleTriTableau()} or
364 * {@link ensembles.EnsembleTriTableau2#EnsembleTriTableau2()}
365 */
366 @Test
367 public final void testDefaultConstructor()
368 {
369     String testName = new String(typeName + "()");
370     System.out.println(testName);
371
372     ensemble = constructEnsemble(testName, typeEnsemble, null);
373     assertNotNull(testName + " non null instance failed", ensemble);
374
375     assertEquals(testName + " instance type failed", typeEnsemble,
376         ensemble.getClass());
377     assertTrue(testName + " empty instance failed", ensemble.estVide());
378     assertEquals(testName + " instance size failed", 0, ensemble.cardinal());
379 }
380
381 /**
382 * Test method for
383 * {@link ensembles.EnsembleTriVector#EnsembleTriVector(Iterable)} or
384 * {@link ensembles.EnsembleTriVector2#EnsembleTriVector2(Iterable)} or
385 * {@link ensembles.EnsembleTriListe#EnsembleTriListe(Iterable)} or
386 * {@link ensembles.EnsembleTriListe2#EnsembleTriListe2(Iterable)} or
387 * {@link ensembles.EnsembleTriTableau#EnsembleTriTableau(Iterable)} or
388 * {@link ensembles.EnsembleTriTableau2#EnsembleTriTableau2(Iterable)} or
389 */
390 @Test
391 public final void testCopyConstructor()
392 {
393     String testName = new String(typeName + "(Iterable)");
394     System.out.println(testName);
395
396     ensemble = constructEnsemble(testName, typeEnsemble, listElements);
397     assertNotNull(testName + " non null instance failed", ensemble);
398
399     assertEquals(testName + " instance type failed", typeEnsemble,
400         ensemble.getClass());
401     assertFalse(testName + " not empty instance failed", ensemble.estVide());
402     boolean compare = compareElts2Array(testName, ensemble,
403         singleSortedElements);
404     assertTrue(testName + " elts compare failed", compare);
405
406     // Tous les ÃlÃments de ensemble doivent se retrouver dans list
407     for (String elt : ensemble)
408     {
409         assertTrue(testName + "check content [" + elt + "] failed",
410             listElements.contains(elt));
411     }
412
413     // Tous les ÃlÃments de l'ensemble n'existent qu'Ã un seul exemplaire
414     boolean countCheck = EnsembleTriTest.<String>checkCount(testName,
415         ensemble);
416     assertTrue(testName + "after count check failed", countCheck);
417 }
418
419 /**
420 * Test method for {@link ensembles.EnsembleTri#ajout(java.lang.Comparable)}.
421 */
422 @Test
423 public final void testAjout()
424 {
425     String testName = new String(typeName + ".ajout(E)");
426     System.out.println(testName);
427
428     assertTrue(testName + " vide avant remplissage failed",
429         ensemble.estVide());
430
431     int size = 0;
432     for (int i = 0; i < elements.length; i++)
433     {
434         if (!ensemble.contient(elements[i]))
435         {
436             size++;
437         }
438         ensemble.ajout(elements[i]);
439         assertEquals(testName + " size failed", size, ensemble.cardinal());
440         boolean checkElts = compareElts2Array(testName, ensemble,
441             insertSortedElements[i]);
442         assertTrue(testName + " check elts failed", checkElts);
443     }
444 }
445
446 /**
447 * Test method for {@link ensembles.EnsembleTri#rang(java.lang.Comparable)}.
448 */
449 @Test
450

```

32/33



nov 04, 15 18:18

**EnsembleTriTest.java**

Page 6/6

```
451 public final void testRang()  
452 {  
453     String testName = new String(typeName + ".rang(E)");  
454     System.out.println(testName);  
455  
456     assertTrue(testName + " vide avant remplissage failed",  
457         ensemble.estVide());  
458  
459     for (int i = 0; i < elements.length; i++)  
460     {  
461         assertEquals(testName + " rang de " + elements[i] + "[" + i  
462             + "] failed", insertionRank[i], ensemble.rang(elements[i]));  
463         ensemble.ajout(elements[i]);  
464     }  
465 }  
466 }
```