BPOO - Support TD 6

Dut/Info-S2/M2103

Table des matières

Version corrigée

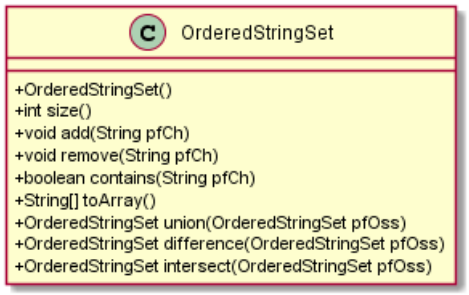
Cette version comporte des indications pour les réponses aux exercices.

PreReq	Tableaux dynamiques, structures chaînées.
ObjTD	**Mettre en oeuvre en java une classe Ensemble**.
Durée	**2 séance** de 1,5h

1. Domaine d'étude : implémenter un ensemble ordonné de String

On s'intéresse dans ce TD à la mise en oeuvre d'une classe **Ensemble ordonné** de chaînes de caractères.

Le diagramme UML suivant montre l'interface générale de la classe attendue.



Un ensemble ordonné est une **collection ordonnée qui n'accepte pas les doublons** :

- Par exemple, un ensemble de String n'accepte qu'une seule fois la valeur "bonjour". Par contre "bonjour" et "BONjour" sont acceptées.
- L'ensemble constitué des valeurs "a", "d", "b", "c" sera ordonné en { "a", "b", "c", "d" }.
- Après chaque ajout, l'ensemble est ordonné lorsqu'on appelle `toArray()`.
- Une approche par programmation par contrats est utilisée : lors des opérations add/remove :
    - L'ajout d'une chaîne déjà présente ne réalise aucun ajout sans levée d'exception ; on aurait pu renvoyer un booléen.
    - La suppression d'une chaîne non présente ne réalise aucune suppression sans levée d'exception ; on aurait pu renvoyer un booléen.
    - *On supposera dans tout l'exercice que la valeur null est refusée (non ajoutée, non enlevé, non recherchée, sans levée d'exception).*

Exemple d'utilisation :

```
String[] res;

OrderedStringSet oss, oss2, oss3;

oss = new OrderedStringSet();
  res = oss.toArray();   for (String s : res) {System.out.print("  " + s);}
```

```
  //  (Rien)

oss.add("S3");
  res = oss.toArray();   for (String s : res) {System.out.print("  " + s);}
  //  "S3"

oss.add("S1");
oss.add("S2");
  res = oss.toArray();   for (String s : res) {System.out.print("  " + s);}
  //  "S1"  "S2"  "S3"

oss.add("S1");
oss.add("S2");
oss.add("S4");
oss.add(null);
  res = oss.toArray();   for (String s : res) {System.out.print("  " + s);}
  //  "S1"  "S2"  "S3"  "S4"

System.out.println( oss.contains("S2") ); // true
System.out.println( oss.contains("S4") ); // true
System.out.println( oss.contains("toto") ); // false
System.out.println( oss.contains("s1") ); // false
System.out.println( oss.contains(null) ); // false

oss.remove(null);
oss.remove("s2");
oss.remove("toto");
  res = oss.toArray();   for (String s : res) {System.out.print("  " + s);}
  //  "S1"  "S2"  "S3"  "S4"

oss.remove("S2");
  res = oss.toArray();   for (String s : res) {System.out.print("  " + s);}
  //  "S1"  "S3"  "S4"

oss.remove("S1");
oss.remove("S4");
oss.remove("S3")
  res = oss.toArray();   for (String s : res) {System.out.print("  " + s);}
  //  (Rien)

oss2 = new OrderedStringSet();
oss2.add("S1"); oss2.add("S2"); oss2.add("S3");

oss3 = new OrderedStringSet();
oss3.add("S1"); oss3.add("S3"); oss3.add("S4");

oss = oss2.union(oss3);
  res = oss.toArray();   for (String s : res) {System.out.print("  " + s);}
  //  "S1"  "S2"  "S3"  "S4"

oss = oss2.intersect(oss3);
  res = oss.toArray();   for (String s : res) {System.out.print("  " + s);}
  //  "S1"  "S3"

oss = oss2.difference(oss3);
  res = oss.toArray();   for (String s : res) {System.out.print("  " + s);}
  //  "S2"
```

2. Réflexion sur la mise en oeuvre

Dans un premier temps, on imagine deux implémentations :

- par structure chaînée (simplement chaînée) comme déjà vu, avec les éléments ordonnés,
- par tableau dynamique comme déjà vu : le tableau a, à tout moment, une taille de celle du nombre d'éléments contenus dans l'ensemble, et avec les éléments ordonnés.

Une implémentation par ArrayList<E> serait plus simple à mettre en oeuvre mais relève de la même approche : un tableau "dynamique".

Questions : Indiquer rapidement les solutions pour mettre en oeuvre (quoi faire) les opérations dans chaque approche :

- Pour l'insertion (add) :
  - Tableau dynamique :

.

.

.

- Liste chaînée :

.

.

.

- Pour la suppression (remove) :
  - Tableau dynamique :

.

.

.

  - Liste chaînée :

.

.

.

- Pour la recherche (contains) :
  - Tableau dynamique :

.

.

.

  - Liste chaînée :

.

.

.

Réponse

- Pour l'insertion :
  - Tableau dynamique :
    - Nécessite de trouver si l'élément n'existe pas avant de l'insérer afin de créer un tableau plus grand.
    - Insertion en début, milieu, fin (ensemble ordonné).
  - Liste chaînée :
    - On peut rechercher la place de l'élément en même temps que son existence et l'ajouter dés que sa place est trouvée (ensemble ordonné).
- Pour la suppression
  - Tableau dynamique :
    - Nécessite de trouver si l'élément existe avant de le supprimer afin de créer un tableau plus petit.
    - Suppression en début, milieu, fin (ensemble ordonné).
  - Liste chaînée :
    - On peut rechercher l'élément et le supprimer dés que sa place est trouvée (ensemble ordonné).
- Pour la recherche (contains) :
  - Tableau dynamique :
    - Recherche dichotomique : trouve en log2 (n) maxi → on dit l'algorithme de complexité O (log2 (n))
  - Liste chaînée :

- Linéaire : trouve en n maxi → on dit l'algorithme de complexité O (n)

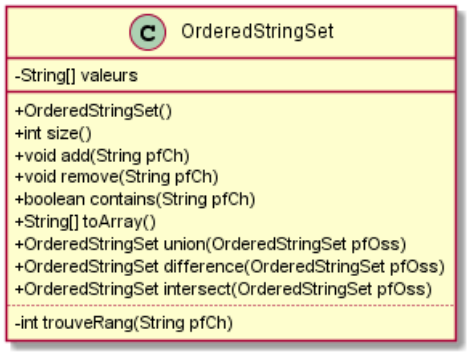
```
           log2 (10) == 3.32
          log2 (100) == 6.64
        log2 (1 000) == 9.96
       log2 (10 000) == 13.28
    log2 (1 000 000) == 19.93
log2 (1 000 000 000) == 29.89
```

3. Mise en oeuvre par tableau dynamique

3.1. Principes

On retient une mise en oeuvre par tableau dynamique en stockant les données de façon ordonnée.

Le diagramme UML suivant montre la conception réalisée pour cette implémentation.



Quelques exemples d'opérations et de résultat :

1. `OrderedStringSet oss = new OrderedStringSet()` :

```
tableau  ||  // Tableau de longueur 0
valeurs  ++
         ||
```

2. `oss.add("S3");` :

```
tableau  |  0 |
valeurs  +----+
         |"S3"|  // Attention => en vrai des références
```

3. `oss.add("S1");` :

```
tableau  | 0  | 1  |
valeurs  +----+----+
         |"S1"|"S3"|  // Attention => en vrai des références
```

4. `oss.add("S2");` :

```
tableau  | 0  | 1  | 2  |
valeurs  +----+----+----+
         |"S1"|"S2"|"S3"|  // Attention => en vrai des références
```

5. `oss.add("S1");` :

```
tableau  | 0  | 1  | 2  |
valeurs  +----+----+----+
         |"S1"|"S2"|"S3"|  // Attention => en vrai des références
```

6. `oss.add(null);` :

```
tableau  | 0  | 1  | 2  |
valeurs  +----+----+----+
         |"S1"|"S2"|"S3"|  // Attention => en vrai des références
```

7. `oss.contains("S2")` ⇒ true
8. `oss.contains("S1")` ⇒ true
9. `oss.contains("toto")` ⇒ false
10. `oss.contains("s1")` ⇒ false
11. `oss.contains(null)` ⇒ false
12. `oss.remove(null);` :

```
tableau  | 0  | 1  | 2  |
valeurs  +----+----+----+
         |"S1"|"S2"|"S3"|  // Attention => en vrai des références
```

13. `oss.remove("s2");` :

```
tableau  | 0  | 1  | 2  |
valeurs  +----+----+----+
         |"S1"|"S2"|"S3"|  // Attention => en vrai des références
```

14. `oss.remove("toto");` :

```
tableau  | 0  | 1  | 2  |
valeurs  +----+----+----+
         |"S1"|"S2"|"S3"|  // Attention => en vrai des références
```

15. `oss.remove("S2");` :

```
tableau  | 0  | 1  |
valeurs  +----+----+
         |"S1"|"S3"|  // Attention => en vrai des références
```

16. `oss.remove("S1");`
17. `oss.remove("S3");` :

```
tableau  ||
valeurs  ++
         ||
```

3.2. Réalisation

Certaines méthodes de la classe sont déjà données :

- On vous donne la méthode `trouveRang(String pfCh)` en version non optimisée.
- On vous donne aussi : le constructeur, `size()`, `toArray()`.

Ecrire le corps des méthodes suivantes :

- `contains()`, `remove()`, `add()`.
- Ecrire les méthodes qui renvoient **un nouvel objet OrderedStringSet** : `union()` (receveur union paramètre), `difference()` (receveur "moins" paramètre), +intersect() + (receveur intersection paramètre)

- Ces méthodes sont elles des transformateurs ? des accesseurs (observateurs) ?

- Récrire `trouveRang(String pfCh)` avec une recherche par dichotomie.
- Récrire `add(String pfCh)` en réutilisant une recherche par dichotomie pour : a) soit trouver la chaîne et donc sortir de la fonction, b) soit trouver le rang où insérer la chaîne puis réaliser l'insertion.
- Pourquoi `toArray()` renvoie une copie du tableau interne et non le tableau lui-même ?

Comparaison de chaînes de caractères : `public int compareTo(String anotherString)`

- Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this String object is compared lexicographically to the character sequence represented by the argument string. The result is a negative integer if this String object lexicographically precedes the argument string. The result is a positive integer if this String object lexicographically follows the argument string. The result is zero if the strings are equal; compareTo returns 0 exactly when the equals(Object) method would return true.
- **Parameters**: anotherString - the String to be compared.
- **Returns**: the value 0 if the argument string is equal to this string; a value less than 0 if this string is lexicographically less than the string argument; and a value greater than 0 if this string is lexicographically greater than the string argument.

Réponses

Pourquoi `toArray()` renvoie une copie du tableau interne et non le tableau lui-même ?

- Car renverrai la référence du tableau
- ⇒ le tableau interne serait accédé depuis l'extérieur et donc modifiable ...

```java
public class OrderedStringSet {

	private String[] valeurs;

	public OrderedStringSet() {
		this.valeurs = new String[0];
	}

	public int size() {
		return this.valeurs.length;
	}

	private int trouveRang(String pfCh) {
		int deb, fin, milieu;
		deb = 0;
		fin = this.valeurs.length - 1;

		while (deb <= fin) {
			milieu = (deb + fin) / 2;
			if (this.valeurs[milieu].compareTo(pfCh) == 0) {
				return milieu;
			} else if (this.valeurs[milieu].compareTo(pfCh) < 0) {
				deb = milieu + 1;
			} else {
				fin = milieu - 1;
			}
		}

		return -1;
	}

	// 3 versions de add !!
	public void add(String pfCh) {
		int rang, i, j;
		boolean insere;
		String nt[];

		if (pfCh == null) {
			return;
		}

		rang = this.trouveRang(pfCh);
```

```
                if (rang != -1) { // Si existe : stop
                        return;
                }

                // Créer le tableau
                nt = new String[this.valeurs.length + 1];

                insere = false;
                j = 0;
                for (i = 0; i < this.valeurs.length; i++) {
                        if (insere) {
                                // Si déja placé : simple recopie
                                nt[j] = this.valeurs[i];
                                j++;
                        } else {
                                if (this.valeurs[i].compareTo(pfCh) < 0) {
                                        // Si après : recopie
                                        nt[j] = this.valeurs[i];
                                        j++;
                                } else {
                                        // Si avant : insérer et recopier
                                        nt[j] = pfCh;
                                        j++;
                                        nt[j] = this.valeurs[i];
                                        j++;
                                        insere = true;
                                }
                        }
                }
                if (!insere) { // si pas encore insérer : le faire(dernier)
                        nt[nt.length - 1] = pfCh;
                }

                this.valeurs = nt;

                this.initParcours();
        }

        // VERSION _1_2
        public void addV_1_2(String pfCh) {
                int rangVal, i, j;
                boolean insere;
                String nt[];

                if (pfCh == null) {
                        return;
                }

                rangVal = -1;
                i = 0;
                while (rangVal==-1 && i < this.valeurs.length) {
                        if (this.valeurs[i].compareTo(pfCh) < 0) {
                                i++;
                        } else if (this.valeurs[i].compareTo(pfCh) > 0) {
                                rangVal = i;
                        } else if (this.valeurs[i].compareTo(pfCh) == 0) {
                                return ;
                        }
                }

                if (rangVal == -1) {
                        rangVal = this.valeurs.length;
                }

                // Créer le tableau
                nt = new String[this.valeurs.length + 1];

                for (i = 0; i < rangVal; i++) {
                        nt[i] = this.valeurs[i];
                }

                nt[rangVal] = pfCh;

                j = rangVal+1;
                for (i = rangVal; i < this.valeurs.length; i++) {
                        nt[j] = this.valeurs[i];
                        j++;
                }

                this.valeurs = nt;
        }

        // VERSION 2
```

```
	public void addV2 (String s) {
		int i, j, rangVal;
		String nt[];

		if (s == null) {
			return;
		}

		int deb, fin, milieu;
		deb = 0;
		fin = this.valeurs.length - 1;

		while (deb <= fin) {
			milieu = (deb + fin) / 2;
			if (this.valeurs[milieu].compareTo(s) == 0) {
				// Présent => stop
				return ;
			} else if (this.valeurs[milieu].compareTo(s) < 0) {
				deb = milieu + 1;
			} else {
				fin = milieu - 1;
			}
		}

		//		System.out.println("\n\n+++ Insertion de : "+s+" rang : "+deb);
		//		Test.twoModesPrint("Dans : ", this);

		// Créer le tableau
		nt = new String[this.valeurs.length + 1];

		rangVal = deb;

		for (i = 0; i < rangVal; i++) {
			nt[i] = this.valeurs[i];
		}

		nt[rangVal] = s;

		j = rangVal+1;
		for (i = rangVal; i < this.valeurs.length; i++) {
			nt[j] = this.valeurs[i];
			j++;
		}

		this.valeurs = nt;
	}


	public void remove(String pfCh) {
		int rang, i, j;

		if (pfCh == null) {
			return;
		}

		rang = this.trouveRang(pfCh);
		if (rang == -1) {
			// Si n'existe pas : stop
			return;
		}

		String[] nt;
		// Créer le tableau
		nt = new String[this.valeurs.length - 1];

		j = 0;
		// Recopier jusqu'à rang exclu
		for (i = 0; i < rang; i++) {
			nt[j] = this.valeurs[i];
			j++;
		}
		// Recopier à partir de (rang+1)
		for (i = rang + 1; i < this.valeurs.length; i++) {
			nt[j] = this.valeurs[i];
			j++;
		}
		this.valeurs = nt;

		// Reinit iterateur
		this.initParcours();
	}

	public boolean contains(String pfCh) {
```

```
                if (pfCh == null) {
                        return false;
                }

                return this.trouveRang(pfCh) != -1;
        }

        public String[] toArray() {
                String[] resultat;
                resultat = new String[this.valeurs.length];
                int i;
                for (i = 0; i < this.valeurs.length; i++) {
                        resultat[i] = this.valeurs[i];
                }
                return resultat;
        }

        public OrderedStringSet union(OrderedStringSet pfOss) {
                OrderedStringSet resultat = new OrderedStringSet();
                String[] temp;

                temp = pfOss.toArray();
                for (int i = 0; i < temp.length; i++) {
                        resultat.add(temp[i]);
                }
                for (int i = 0; i < this.valeurs.length; i++) {
                        resultat.add(this.valeurs[i]);
                }

                return resultat;
        }

        public OrderedStringSet difference(OrderedStringSet pfOss) {
                OrderedStringSet resultat = new OrderedStringSet();

                for (int i = 0; i < this.valeurs.length; i++) {
                        if (!pfOss.contains(this.valeurs[i])) {
                                resultat.add(this.valeurs[i]);
                        }
                }

                return resultat;
        }

        public OrderedStringSet intersect(OrderedStringSet pfOss) {
                OrderedStringSet resultat = new OrderedStringSet();

                for (int i = 0; i < this.valeurs.length; i++) {
                        if (pfOss.contains(this.valeurs[i])) {
                                resultat.add(this.valeurs[i]);
                        }
                }

                return resultat;
        }

        // Itérateur

        private int rangCourant;

        public void initParcours() {
                this.rangCourant = 0;
        }

        public boolean hasNext() {
                return this.rangCourant < this.valeurs.length;
        }

        public String next() {
                return this.valeurs[this.rangCourant++];
        }
}
```

Dernière mise à jour 2017-03-06 00:21:43 CET