Mathématiques 1
Les Suites

Institut Paul Lambin

26 novembre 2021

Suites Mathématiques : Introduction

Le concept abstrait de ”suite” est omniprésent en informatique.
On le retrouve dans les notions de

- fichier
- séquence
- vecteur
- chaîne (de caractère)
- tableau
- liste
- file
- pile
- ...

Dans tous ces cas $\rightarrow$ l'**ordre** est important et les **répétitions** possibles.

Attention : l'**accès aux données** varie d'une structure à l'autre

Suites Mathématiques : Définition

Une **suite** est une séquence d'objets, non nécessairement distincts, numérotés par une partie initiale de $\mathbb{N}_0$.

Attention ! :

- l'**ordre** a de l'importance
- un **même** objet peut être présent **plusieurs fois**
- les objets sont a priori **quelconques** (de toutes sortes)

Dans la suite on se limitera à des suites de naturels

Suites Mathématiques : Notation

On utilise généralement la notation **parenthésée**.

Exemple :

$s = (5, 2, a, \texttt{true}, 2) \rightarrow$ suite dont les éléments sont $\begin{cases} e_{s1} = 5 \\ e_{s2} = 2 \\ e_{s3} = a \\ e_{s4} = \texttt{true} \\ e_{s5} = 2 \end{cases}$

Remarques :

- Les éléments de la suite sont numérotés par la partie $\{1, 2, 3, 4, 5\}$ de $\mathbb{N}_0$
- Elle a des éléments de différents types (entier, caractère, booléen)
- Elle contient deux fois le même élément (l'entier 2)

Propriétés de suites : longueur d'une suite

La **longueur** d'une suite est le **nombre d'éléments** de la suite.

Exemple :

La suite $s = (5, 2, a, \texttt{true}, 2)$ est de longueur 5 car elle possède 5 éléments.

Remarques :

1) Une suite peut être de longueur infinie $\rightarrow$ Suite de Fibonacci
2) La **suite vide**, notée $()$, est la seule suite de longueur **0**.

Proprités des suites : Égalité de deux suites

Deux suites sont **égales** si elles possèdent les **mêmes éléments dans le même ordre**.

Exemples :

Soit les suites

- $s = (5, 2, a, \texttt{true}, 2)$
- $u = (5, 2, a, 2, \texttt{true})$
- $v = (5, 2, a, \texttt{true}, 2)$

Alors

1) Les suites s et v sont égales : définies sur le même $I = \{1, 2, 3, 4, 5\}$.
   Et pour tout entier i de la partie I on a $e_{si} = e_{vi}$.

$$\text{En effet} \begin{cases} e_{s1} = e_{v1} = 5 \\ e_{s2} = e_{v2} = 2 \\ e_{s3} = e_{v3} = a \\ e_{s4} = e_{v4} = \texttt{true} \\ e_{s5} = e_{v5} = 2 \end{cases}$$

2) La suite s n'est pas égale à la suite u car $e_{s4} = 3 \neq \texttt{true} = e_{u4}$

Proprietés des suites : Sous-suite

Une suite s_2 est une sous-suite d'une suite s_1 si

- les éléments de s_2 sont **tous présents dans** s_1,

  ET

- les éléments apparaissent dans s_1 **dans le même ordre que dans** s_2.

Exemples :

1) $s_2 = (5, 2, 4)$ est une sous-suite de $s_1 = (7, 5, 3, 2, 1, , 8, 4)$
2) $s_2 = (5, 2, 4, \mathbf{4})$ n'est une pas une sous-suite de $s_1 = (7, 5, 3, 2, 1, 8, 4)$
   $\rightarrow$ la suite s_1 ne contient qu'une fois l'entier 4.
3) $s_2 = (5, 2, 4)$ n'est pas une sous-suite de $s_1 = (7, 5, 3, \mathbf{4}, 1, 8, \mathbf{2})$
   $\rightarrow$ les éléments de s_2 n'apparaissent pas dans le même ordre dans s_1.

Suites mathématiques : Nature Récursive

Le concept de suite est naturellement récursif.

En effet, une suite est

- soit vide

- soit constituée de deux choses :

  1) le premier élément de la suite appelé **tête** de la suite
  2) une suite appelée **corps** de la suite
     (la suite de départ dont on a retiré le premier élément)

Suites mathématiques : Nature Récursive

Exemples :

Soit $s_1 = (\text{rouge}, 5, a, \texttt{true})$.

Alors

1) s_1 est une suite car constituée $\begin{cases} \text{d'une tête} & : \quad \text{rouge} \\ \text{d'un corps} & : \quad s_2 = (5, a, \texttt{true}) \end{cases}$

2) s_2 est une suite car constituée $\begin{cases} \text{d'une tête} & : \quad 5 \\ \text{d'un corps} & : \quad s_3 = (a, \texttt{true}) \end{cases}$

3) s_3 est une suite car constituée $\begin{cases} \text{d'une tête} & : \quad a \\ \text{d'un corps} & : \quad s_4 = (\texttt{true}) \end{cases}$

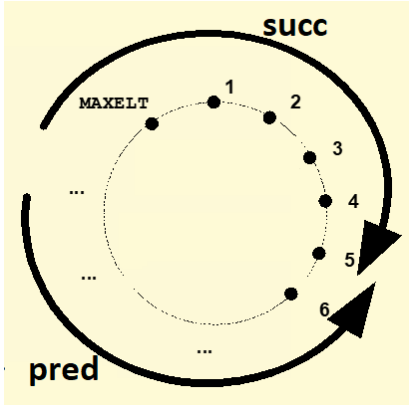
4) s_4 est une suite car constituée $\begin{cases} \text{d'une tête} & : \quad \texttt{true} \\ \text{d'un corps} & : \quad s_5 = () \end{cases}$

Suites mathématiques : Nature Récursive

L'univers de travail sera les naturels entre 1 et `MAXELT`

$\rightarrow$ Univers : $\{1, 2, 3, \cdots, \texttt{MAXELT}\}$

Par facilité, on va considérer que cette univers est circulaire :



De plus on dispose de deux outils :

- `succ` : fournit le successeur d'un entier dans l'univers (l'entier suivant)
- `pred` : fournit le prédécesseur d'un entier de l'univers (l'entier précédent)

Attention !

- le successeur de `MAXELT` dans notre univers est l'entier 1.
- le prédécesseur de 1 dans notre univers est l'entier `MAXELT`.

Les Suites : Implémentation : La classe `Elt`

Les entiers de notre univers seront des instances de la classe `Elt`
Cette classe fournit

- le `Elt` "constant" `MAXELT` :

```
public static final Elt MAXELT
```

- les constructeurs :

```
public Elt(int i)
 // construit un Elt de valeur i
 // produit une "llegalArgumentException" si
 // i n'appartient pas à l'Univers.

public Elt(Elt e)
 // constructeur par recopie;
 // produit une "IllegalArgumentException" si e est null.
```

Les Suites : Implémentation : La classe `Elt`

Cette classe fournit

- les méthodes :

```
public int val()
 // renvoie la valeur du Elt courant.

public boolean equals(Object o)
 // renvoie true si le Elt courant est égal à o.
 // L'égalité se fait sur la valeur de l'Elt.

public int hashCode()
 // renvoie un hashCode uniquement basé sur la
 // valeur de l'Elt

public Elt succ()
 // renvoie le successeur du Elt courant.

public Elt pred()
 // renvoie le prédécesseur du Elt courant.
```

Les Suites : Implémentation : La classe `Elt`

- ainsi que les méthodes de lecture :

```
public static Elt lireElt()
 // renvoie le Elt lu au clavier

public static Suite lireSuite()
 // renvoie la Suite lue au clavier
```

Attention ! En cas d'entrée incorrecte, ces méthodes exigent de recommencer la saisie.

Les Suites : Implémentation : La classe `SuiteDeBase`

Classe abstraite implémentant les fonctionnalités de base des suites d'`Elt`
Elle fournit les constructeurs :

```
public SuiteDeBase()
 // construit la Suite vide.

public SuiteDeBase(SuiteDeBase s)
 // constructeur par recopie.

public SuiteDeBase(String st)
 // constructeur à partir d'une chaîne de caractères;
 // par exemples : "(6,7,33,6)" ou "6..Maxelt,15"
 // ou "(10,7,33 6 " ou "()" etc.

public SuiteDeBase(Elt x,SuiteDeBase s)
 // construit la Suite dont x est la tête,
 // et dont s est le corps.
```

Les Suites : Implémentation : La classe `SuiteDeBase`

Elle fournit les méthodes :

```
public boolean estVide()
 // renvoie true si la Suite courante est vide.

public Elt tete()
 // renvoie la tête de la Suite courante
 // si celle-ci est NON VIDE ;
 // lance une MathException dans le cas contraire.

public SuiteDeBase corps()
 // renvoie une copie du corps de la Suite courante
 // si celle-ci est NON VIDE ;
 // lance une MathException dans le cas contraire.
```

Les Suites : Implémentation : La classe `SuiteDeBase`

```
public void couper()
 // supprime le premier Elt de la Suite courante
 // si celle-ci est NON VIDE ;
 // lance une MathException dans le cas contraire.

public void ajouter(Elt e)
 // rajoute le Elt e au DÉBUT de la Suite courante.

public String toString()
 // renvoie une notation parenthésée
 // de la Suite courante.

public Iterator <Elt> iterator()
 // renvoie un itérateur sur la Suite courante.
```

Les Suites : Implémentation : L'itérateur

Un itérateur est un dispositif qui permet de parcourir tous les `Elt` d'une `Suite`.

Un itérateur est lui-même un objet qui dispose de deux méthodes :

- `public boolean hasNext()`

```
/** renvoie true tant que l'itérateur
    n'a pas terminé son parcours. */
```

- `public Elt next()`

```
/** renvoie l'un après l'autre les Elt de la Suite;
    le premier au 1er appel, le deuxième au 2e appel, etc */
```

Attention : Tout appel à `next()` doit être "protégé" par un appel à `hasNext()`

→ S'il n'y a plus d'élément à parcourir, un appel à la méthode `next` va générer une `NoSuchElementException`

→ `hasNext()` permet de savoir s'il reste encore des éléments à parcourir.

renvoie `true` s'il reste au moins un élément à parcourir

renvoie `false` s'il ne reste plus d'élément à parcourir.

Les Suites : Implémentation : L'itérateur

Exemple d'utilisation :

```
Suite s = Io.lireSuite();
Iterator<Elt> iter = s.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next());
}
```

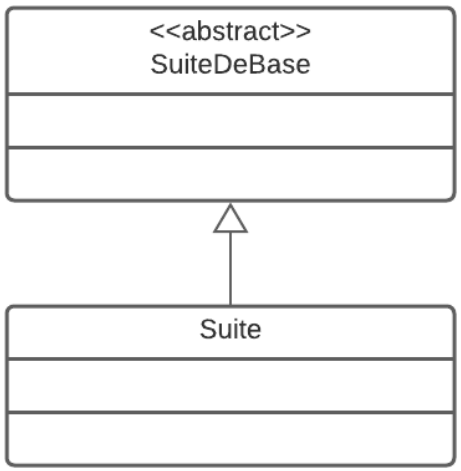
Remarques :

1) Si `s` est vide, alors le premier appel de `hasNext()` renvoie `false`
2) Un appel à la méthode `next()` alors qu'il n'y a plus d'élément à parcourir (quand `hasNext()` renvoie `false`) génère une `NoSuchElementException`.
3) La méthode ci-dessus va parcourir les éléments de la suite et les afficher dans l'ordre défini par la suite.

Classe à implémenter : Consignes

A vous maintenant de compléter une classe qui

- `s'appelle` `Suite`
- `hérite` de la classe `SuiteDeBase`.



Une instance de la classe `Suite` aura donc

- toutes les fonctionnalités de la classe `SuiteDeBase`
- plus celles que l'on vous demande d'implémenter dans la fiche 8.

Classe à implémenter : Exemples de méthode

Exemple 1 : Méthode `contient(Elt e)`

Cette méthode renvoie

- `true` si l'`Elt e` se trouve dans la suite courante
- `false` sinon.

Version itérative

```
public boolean contient(Elt e) {
    Iterator<Elt> it = this.iterator();
    while (it.hasNext()) {
        if (e.equals(it.next()))
            return true ;
    }
    return false ;
}
```

- Parcours de la suite grâce à un itérateur
- l'`Elt` renvoyé par l'itérateur est égal à l'`Elt e` $\rightarrow$ on renvoie `true`.
- Parcours complet de la suite sans trouver l'`Elt e` $\rightarrow$ on renvoie `false`.

Classe à implémenter : Exemples de méthode

Exemple 1 : Méthode `contient(Elt e)`

Version récursive

```
public boolean contient(Elt e) {
    if (this.estVide())
        return false;
    if (this.tete().equals(e))
        return true;
    return this.corps().contient(e);
}
```

1) cas où l'on peut répondre directement à la question "est-ce que la suite contient l'`Elt e` ?" :

Cas 1 : Si la suite est vide, alors elle ne contient pas l'`Elt e`

Cas 2 : Si la tête de la suite est l'`Elt e` alors la suite le contient

$\rightarrow$ cas "bêtes" ou cas de base.

2) Si la suite n'est pas vide et que sa tête n'est pas l'`Elt` recherché

$\rightarrow$ appel récursif pour le chercher dans le corps de la suite.

Classe à implémenter : Exemples de méthode

Exemple 2 : Méthode `moinsPremOcc(Elt x)`

Renvoie la `Suite` obtenue en supprimant dans la `Suite` courante la première occurrence (éventuelle) de `x`.

Version récursive

```
public Suite moinsPremOcc(Elt x){
    if (this.estVide())
        return new Suite() ;
    if (this.tete().equals(x))
        return this.corps() ;
    return new Suite(this.tete(),this.corps().moinsPremOcc(x));
}
```

1) Cas où la suppression peut se faire directement → Cas "bêtes" :

Cas 1 : la suite est vide → on renvoie la suite vide.

Cas 2 : la tête de la suite est égale à `x` → on renvoie le corps de la suite.

2) La tête n'est pas l'`Elt` à supprimer alors la suite recherchée a la même tête que la suite courante et son corps est le corps de la suite courante moins l'éventuelle première occurrence de l'`Elt x`
→ appel récursif sur le corps.

Classe à implémenter : Exemples de méthode

Exemple 2 : Méthode `moinsPremOcc(Elt x)`

Version itérative

```
public Suite moinsPremOcc(Elt x){
    Suite save = new Suite();
    Iterator<Elt> it = this.iterator();
    boolean trouve = false ;
    while(!trouve && it.hasNext()){
        Elt y = it.next()
        if (!y.equals(x))
            save.ajouter(y);
        else
            trouve = true ;
    }
    while (it.hasNext()) {
        save.ajouter(it.next());
    }
    return save.inverse();
}
```

Classe à implémenter : Exemples de méthode

Exemple 2 : Méthode `moinsPremOcc(Elt x)`

Dans la version itérative

1) grâce à l'itérateur on parcourt la suite.
2) Tant qu'on ne trouve pas l'`Elt` à supprimer, on ajoute les éléments parcourus dans une nouvelle suite.
3) Dès que on trouve l'`Elt` à supprimer, on sort de la première boucle.
4) On recommence à parcourir la suite et on ajoute tous les éléments de la suite à partir de l'élément suivant dans la nouvelle suite.
5) Etant donné que l'ajout se fait à l'avant d'une suite, la nouvelle suite créée contient les `Elt` dans l'ordre inverse.
   - $\rightarrow$ appel à la méthode `inverse()` pour obtenir la suite demandée dans le bon ordre.

Classe à implémenter : Dernières consignes

1. Programmez les méthodes de la classe `Suite` **de manière récursive le plus possible**.

2. **Interdiction de modifier** la suite courante sauf consigne explicie.
   Dans les deux méthodes présentées ci-avant, l'état de la suite `Suite` courante doit être le même avant et après l'exécution de la méthode.

3. Pour tester vos méthodes, la classe `TestSuite` est à votre disposition.
   Elle permet de tester chaque méthode séparément.

Classe à implémenter : Dernières consignes

4. Si vous obtenez l'erreur suivante :

```
Exception in thread "main" java.lang.StackOverflowError
   at Suite.longueur(Suite.java:57)
   at Suite.longueur(Suite.java:57)
   at Suite.longueur(Suite.java:57)
   at Suite.longueur(Suite.java:57)
   at Suite.longueur(Suite.java:57)
   at Suite.longueur(Suite.java:57)
   at Suite.longueur(Suite.java:57)
   at Suite.longueur(Suite.java:57)
   at Suite.longueur(Suite.java:57)
   at Suite.longueur(Suite.java:57)
   at Suite.longueur(Suite.java:57)
   at Suite.longueur(Suite.java:57)
```

→ trop d'appels récursifs lors de l'exécution de votre méthode.
→ provient souvent d'appels récursifs "infinis"
→ le programme n'arrive jamais dans un cas de base.