Mathématiques 1
Les Suites : Bonnes pratiques

Institut Paul Lambin

5 décembre 2021

Gestion des tests inutiles

Exemple : Méthode `contient(Elt e)`

Dans la description de cette méthode il est mit
`@throws IllegalArgumentException en cas de paramètre invalide`
$\rightarrow$ non gérée dans l'implémentation précédente.

Première solution

```
public boolean contient(Elt e) {
    if(e==null)
        throw new IllegalArgumentException("e est null") ;
    if (this.estVide())
        return false;
    if (this.tete().equals(e))
        return true;
    return this.corps().contient(e);
}
```

Gestion des tests inutiles

Exemple : Méthode `contient(Elt e)`

Remarques :

1) Ce code est correct.

2) A chaque appel récursif on va tester si `e` est `null` !

   → Si la suite à *n* éléments, on va faire jusqu'à *n* fois le test !

   → Si `e` n'est pas `null` lors du premier appel, il ne peut pas le devenir !

   → Beaucoup de tests inutiles !

Solution :

1. Introduction d'une méthode privée pour la partie récursive.
2. Test dans la méthode publique.

Gestion des tests inutiles

Exemple : Méthode `contient(Elt e)`

Solution propre

```
public boolean contient(Elt e) {
    if(e==null)
        throw new IllegalArgumentException("e est null") ;
    return contientBis(e) ;
}

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

Gestion des tests inutiles

Exemple : Méthode `contient(Elt e)`

Remarques :

1) La méthode publique teste le paramètre et puis appelle la méthode bis
   $\rightarrow$ le paramètre n'est testé qu'une fois !

2) La méthode bis doit être **privée**
   $\rightarrow$ c'est un choix d'implémentation que l'utilisateur n'a pas à connaître

3) La méthode bis est **récursive**
   $\rightarrow$ c'est elle qui fait le travail de recherche de l'élément.

Conclusion : Cette solution est plus propre et plus efficace !

Gestion des appels à d'autres méthodes

Exemple : Méthode `position(Elt e)`

Description de la méthode :

```
/** renvoie la position de la première occurrence de e
          dans la Suite courante ;
 *  renvoie 0 si e n'a pas d'occurrence dans la Suite courante
 *  @throws IllegalArgumentException en cas de paramètre invalide */
```

Analyse :

1) Paramètre à tester $\rightarrow$ introduction d'une méthode bis privée

2) Cas "**triviaux**" :

Cas 1 : La suite est vide $\rightarrow$ elle ne contient pas `e`
$\rightarrow$ on renvoie 0.

Cas 2 : La tête de la suite est l'`Elt e`
$\rightarrow$ on renvoie 1 (`e` est le 1^{er} élément de la suite)

Gestion des appels à d'autres méthodes

Exemple : Méthode `position(Elt e)`

3. Cas "**récursif**" : `e` n'est pas la tête de la suite
   $\rightarrow$ on regarde dans le corps :

   Exemple : position de 5 dans la suite $\begin{matrix}(4, & 2, & 5, & 3, & 2, & 10)\\ 1 & 2 & 3 & 4 & 5 & 6\end{matrix}$:

   $\rightarrow$ 5 n'est pas la tête de la suite

   $\rightarrow$ on regarde dans le corps

   $\rightarrow$ $\begin{matrix}(2, & 5, & 3, & 2, & 10)\\ 1 & 2 & 3 & 4 & 5\end{matrix}$ $\rightarrow$ 5 est en $2^{\text{ème}}$ position dans le corps

   $\rightarrow$ 5 est en $2+1=3^{\text{ème}}$ position dans la suite.

$\rightarrow$ `position(e) = corps().position(e) + 1`

Gestion des appels à d'autres méthodes

Exemple : Méthode `position(Elt e)`

Ébauche de solution

```
public int position(Elt e) {
    if (e == null)
        throw new IllegalArgumentException();
    return positionBis(e) ;
}

private int positionBis(Elt e) {
    if (this.estVide())
        return 0;
    if (e.equals(this.tete()))
        return 1;
    return this.corps().positionBis(e)+1;
}
```

Gestion des appels à d'autres méthodes

Exemple : Méthode `position(Elt e)`

Problème :

```
public static void main(String[] args) {
   Suite s = new Suite("(4,2,5,3,2,10") ;
   System.out.println("s.position(5) = "+s.position(new Elt(5))) ;
   System.out.println("s.position(7) = "+s.position(new Elt(7))) ;
}
```

```
----jGRASP exec: java Suite

s.position(5) = 3 ⇨ OK
s.position(7) = 6 ⇨ KO
----jGRASP: operation complete.
```

→ si l'élément n'est pas dans la suite on parcours toute la suite en ajoutant 1 à la position à chaque appel récursif

→ on renvoie la longueur de la suite au lieu de 0 !

Première solution :

On doit renvoyer 0 si l'élément n'est pas dans la suite

→ appel à la méthode `contient`

Gestion des appels à d'autres méthodes

Exemple : Méthode `position(Elt e)`

Ébauche de solution

```
public int position(Elt e) {
    if (e == null)
        throw new IllegalArgumentException();
    return positionBis(e) ;
}

private int positionBis(Elt e) {
    if (!this.contient(e))
        return 0;
    if (e.equals(this.tete()))
        return 1;
    return this.corps().positionBis(e)+1;
}
```

Gestion des appels à d'autres méthodes

Exemple : Méthode `position(Elt e)`

Tests :

```
public static void main(String[] args) {
   Suite s = new Suite("(4,2,5,3,2,10") ;
   System.out.println("s.position(5) = "+s.position(new Elt(5))) ;
   System.out.println("s.position(7) = "+s.position(new Elt(7))) ;
}
```

```
----jGRASP exec: java Suite

s.position(5) = 3 ⇒ OK
s.position(7) = 0 ⇒ OK
----jGRASP: operation complete.
```

→ Tests bons !

Problème :

On fait l'appel à la méthode `contient` dans la méthode bis récursive !

→ A chaque appel récursif on parcours la suite pour voir si elle contient `e`

→ Inefficace : en $O(n^2)$!

Solution :

→ faire une seule fois l'appel dans la méthode publique !

Gestion des appels à d'autres méthodes

Exemple : Méthode `position(Elt e)`

Solution en $O(n)$

```
public int position(Elt e) {
     if (e == null)
          throw new IllegalArgumentException();
     if (!this.contient(e))
          return 0;
     return positionBis(e) ;
}

private int positionBis(Elt e) {
     if (e.equals(this.tete()))
          return 1;
     return this.corps().positionBis(e)+1;
}
```

Gestion des appels à d'autres méthodes

Exemple : Méthode `position(Elt e)`

Remarques :

1) Si `e` est le dernier élément de la suite alors on la parcourt 2 fois
   - → 1 fois dans `contient` et une fois dans `positionBis`
   - → Méthode en $O(n)$

2) Il existe une solution en seul parcours !
   - → Pas d'appel à la méthode `contient`
   - → A vous de jouer !

Remarques Finales

1) Pensez à utiliser une méthode privée pour éviter les tests inutiles

2) Si vous faites un appel à un autre méthode :
   - Réfléchissez bien à l'endroit de l'appel
   - Réfléchissez à l'utilité de cet appel
   - Un appel à une méthode récursive dans une méthode récursive $\rightarrow O(n^2)$

3) Certaines méthodes sont en $O(n^2) \rightarrow$ `reduite()` par exemple.

4) Ne pas mélanger programmation itérative et programmation récursive.
   - $\rightarrow$ Soit l'une soit l'autre
   - $\rightarrow$ ~~les deux en même temps~~
   - $\rightarrow$ vous devez tout programmer en récursif
     (sauf avis explicite contraire dans l'énoncé ou de la part des professeurs)