

Céline de Roland
Michaël KOZA
L3-STIC-INFO

Compte rendu TP Types Abstraits

Exercice 1 : assertions sur le tri de Shell

L'algorithme du tri de shell possède 4 niveaux de boucles itératives. Nous avons donc effectué $4 \times 3 = 12$ assertions : pour chaque boucle l'invariant au début (noté D), l'invariant à la fin (noté F), et l'invariant en sortie de boucle (noté S), plus un numéro pour indiquer le niveau de profondeur.

Sur l'exemple en page suivante nous pouvons constater qu'il y a de la cohérence entre nos assertions et ce qu'il se passe en réalité.

Pour chaque assertion, nous avons utilisé une variable booléenne pour voir si le test a réussi, et une autre variable pour voir si tous les tests ont réussi. En exécutant le programme, on peut donc voir que nos assertions semblent correctes.

Page suivante : l'algorithme de shell avec les assertions formulées en langue naturelle

Puis : l'exemple. Nous n'avons pas mis toutes les assertions dans l'exemple, qui est déjà très long. Nous avons choisi des endroits qui nous ont semblé appropriés pour montrer la cohérence de nos assertions.

```

do
{
    //ASSERTION 1D : toutes les séries d'incrément incr sont triées
    ( au début, incr = n → ce sont des séries d'un seul élément)

    incr = incr/2;

    for (k = 1; k <= incr; k++)
    {
        //ASSERTION 2D: les k - 1 premières séries d'incr sont triées

        for (i = incr + k; i <= n; i = i + incr)
        {
            //ASSERTION 3D : kieme série d'incr triée jusqu'à i-incr

            x = t[i];
            j = i - incr;

            while (j >= 1 && t[j] > x)
            {
                //ASSERTION 4D
                //Jusqu'à j triés
                //De j + 2incr jusqu'à i : supérieurs à x et triés

                t[j + incr] = t[j];

                j = j - incr;

                //ASSERTION 4F
                //Jusqu'à j triés
                //De j + 2incr jusqu'à i : supérieurs à x et triés

            }

            //ASSERTION 4S
            //Jusqu'à j triés ET inférieurs à x
            //A partir de j + 2incr jusqu'à i triés ET sups à x

            t[j + incr] = x;

            //ASSERTION 3F : kieme série d'incr triée de k à i
        }

        //ASSERTION 2F/3S: la kieme série d'incr est triée jusqu'à n
    }

    //ASSERTION 1F/2S : toutes les séries d'incrément incr sont triées

}
while (incr != 1);

//ASSERTION 1S : la série d'incrément 1 est triée ie le tableau est trié

```

Exemple

Incr = 18

9 3 4 6 2 5 10 7 9 1 4 8 12 3 6 4 7 5

1D : toutes les séries d'incrément 18 sont triées

Incr = 9

2D : les 0 premières séries d'incrément 9 sont triées

9	3	4	6	2	5	10	7	9	1	4	8	12	3	6	4	7	5
9	3	4	6	2	5	10	7	9	9	4	8	12	3	6	4	7	5
1	3	4	6	2	5	10	7	9	9	4	8	12	3	6	4	7	5

2F/2S/2D : la première série d'incrément 9 est triée

1	3	4	6	2	5	10	7	9	9	4	8	12	3	6	4	7	5
---	---	---	---	---	---	----	---	---	---	---	---	----	---	---	---	---	---

2F/2S/2D : les 2 premières séries d'incrément 9 sont triées

1	3	4	6	2	5	10	7	9	9	4	8	12	3	6	4	7	5
---	---	---	---	---	---	----	---	---	---	---	---	----	---	---	---	---	---

2F/2S/2D : les 3 premières séries d'incrément 9 sont triées

1	3	4	6	2	5	10	7	9	9	4	8	12	3	6	4	7	5
---	---	---	---	---	---	----	---	---	---	---	---	----	---	---	---	---	---

2F/2S/2D : les 4 premières séries d'incrément 9 sont triées

1	3	4	6	2	5	10	7	9	9	4	8	12	3	6	4	7	5
---	---	---	---	---	---	----	---	---	---	---	---	----	---	---	---	---	---

2F/2S/2D : les 5 premières séries d'incrément 9 sont triées

1	3	4	6	2	5	10	7	9	9	4	8	12	3	6	4	7	5
---	---	---	---	---	---	----	---	---	---	---	---	----	---	---	---	---	---

2F/2S/2D : les 6 premières séries d'incrément 9 sont triées

1	3	4	6	2	5	10	7	9	9	4	8	12	3	6	4	7	5
1	3	4	6	2	5	10	7	9	9	4	8	12	3	6	10	7	5
1	3	4	6	2	5	4	7	9	9	4	8	12	3	6	10	7	5

2F/2S/2D : les 7 premières séries d'incrément 9 sont triées

1	3	4	6	2	5	4	7	9	9	4	8	12	3	6	10	7	5
---	---	---	---	---	---	---	---	---	---	---	---	----	---	---	----	---	---

2F/2S/2D : les 8 premières séries d'incrément 9 sont triées

1	3	4	6	2	5	4	7	9	9	4	8	12	3	6	10	7	5
1	3	4	6	2	5	4	7	9	9	4	8	12	3	6	10	7	9
1	3	4	6	2	5	4	7	5	9	4	8	12	3	6	10	7	9

2F/2S/2D : les 9 premières séries d'incrément 9 sont triées

1	3	4	6	2	5	4	7	5	9	4	8	12	3	6	10	7	9
---	---	---	---	---	---	---	---	---	---	---	---	----	---	---	----	---	---

1F : toutes les séries d'incrément 9 sont triées

Incr = 4

2D : les 0 premières séries d'incrément 4 sont triées

1 3 4 6 2 5 4 7 5 9 4 8 12 3 6 10 7 9

3D/3F : 1ere série triée jusqu'au rang 1

j i
1 3 4 6 2 5 4 7 5 9 4 8 12 3 6 10 7 9

3D/3F : 1ere série triée jusqu'au rang 5

j i
1 3 4 6 2 5 4 7 5 9 4 8 12 3 6 10 7 9

3D/3F : 1ere série triée jusqu'au rang 9

j i
1 3 4 6 2 5 4 7 5 9 4 8 12 3 6 10 7 9

3D/3F : 1ere série triée jusqu'au rang 13

j i
1 3 4 6 2 5 4 7 5 9 4 8 12 3 6 10 7 9
1 3 4 6 2 5 4 7 5 9 4 8 12 3 6 10 12 9
1 3 4 6 2 5 4 7 5 9 4 8 7 3 6 10 12 9

3F : 1ere série triée jusqu'au rang 17

3S/2S : 1ere série triée

3D/3F : 2eme série triée jusqu'au rang 2

j i
1 3 4 6 2 5 4 7 5 9 4 8 7 3 6 10 12 9

3D/3F : 2eme série triée jusqu'au rang 6

j i
1 3 4 6 2 5 4 7 5 9 4 8 7 3 6 10 12 9

3D/3F : 2eme série triée jusqu'au rang 10

j i
1 3 4 6 2 5 4 7 5 9 4 8 7 3 6 10 12 9
1 3 4 6 2 5 4 7 5 9 4 8 7 9 6 10 12 9
1 3 4 6 2 5 4 7 5 5 4 8 7 9 6 10 12 9
1 3 4 6 2 3 4 7 5 5 4 8 7 9 6 10 12 9

3D/3F : 2eme série triée jusqu'au rang 14

j i
1 3 4 6 2 3 4 7 5 5 4 8 7 9 6 10 12 9

3D/3F : 2eme série triée jusqu'au rang 18

2D/2S : 2eme série triée

j i
1 3 4 6 2 3 4 7 5 5 4 8 7 9 6 10 12 9

j i
1 3 4 6 2 3 4 7 5 5 4 8 7 9 6 10 12 9

j i
1 3 4 6 2 3 4 7 5 5 4 8 7 9 6 10 12 9

2D/2S : 3eme série triée

j i
1 3 4 6 2 3 4 7 5 5 4 8 7 9 6 10 12 9

j i
1 3 4 6 2 3 4 7 5 5 4 8 7 9 6 10 12 9

j i
1 3 4 6 2 3 4 7 5 5 4 8 7 9 6 10 12 9

2S : 4eme série triée

1F/1D : toutes les séries d'incrément 4 sont triées

Incr = 2																	
1	3	4	6	2	3	4	7	5	5	4	8	7	9	6	10	12	9
j i																	
1	3	4	6	2	3	4	7	5	5	4	8	7	9	6	10	12	9
j i																	
1	3	4	6	2	3	4	7	5	5	4	8	7	9	6	10	12	9
1	3	4	6	4	3	4	7	5	5	4	8	7	9	6	10	12	9
1	3	2	6	4	3	4	7	5	5	4	8	7	9	6	10	12	9
j i																	
1	3	2	6	4	3	4	7	5	5	4	8	7	9	6	10	12	9
j i																	
1	3	2	6	4	3	4	7	5	5	4	8	7	9	6	10	12	9
j i																	
1	3	2	6	4	3	4	7	5	5	4	8	7	9	6	10	12	9
1	3	2	6	4	3	4	7	5	5	5	8	7	9	6	10	12	9
1	3	2	6	4	3	4	7	4	5	5	8	7	9	6	10	12	9
j i																	
1	3	2	6	4	3	4	7	4	5	5	8	7	9	6	10	12	9
j i																	
1	3	2	6	4	3	4	7	4	5	5	8	7	9	6	10	12	9
1	3	2	6	4	3	4	7	4	5	5	8	7	9	7	10	12	9
1	3	2	6	4	3	4	7	4	5	5	8	6	9	7	10	12	9
j i																	
1	3	2	6	4	3	4	7	4	5	5	8	6	9	7	10	12	9
j i																	
1	3	2	6	4	3	4	7	4	5	5	8	6	9	7	10	12	9
1	3	2	6	4	6	4	7	4	5	5	8	6	9	7	10	12	9
1	3	2	3	4	6	4	7	4	5	5	8	6	9	7	10	12	9
j i																	
1	3	2	3	4	6	4	7	4	5	5	8	6	9	7	10	12	9

4D : la 2eme série est triée jusqu'au rang 8 et c'est tout

j i

1	3	2	3	4	6	4	7	4	5	5	8	6	9	7	10	12	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	---

4F/4D : la 2eme série est triée jusqu'au rang 6 et

à partir du rang 8 jusqu'au rang 8 triée et > 5

j i

1	3	2	3	4	6	4	7	4	7	5	8	6	9	7	10	12	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	---

4S/4F : la 2eme série est triée jusqu'au rang 4 et

à partir du rang 6 jusqu'au rang 8 triée et > 5

j i

1	3	2	3	4	6	4	6	4	7	5	8	6	9	7	10	12	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	---

3F : la 2eme série est triée jusqu'au rang 10

j i

1	3	2	3	4	5	4	6	4	7	5	8	6	9	7	10	12	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	---

j i

1	3	2	3	4	5	4	6	4	7	5	8	6	9	7	10	12	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	---

j i

1	3	2	3	4	5	4	6	4	7	5	8	6	9	7	10	12	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	---

j i

1	3	2	3	4	5	4	6	4	7	5	8	6	9	7	10	12	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	---

j i

1	3	2	3	4	5	4	6	4	7	5	8	6	9	7	10	12	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	---

j i

1	3	2	3	4	5	4	6	4	7	5	8	6	9	7	10	12	10
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	----

j i

1	3	2	3	4	5	4	6	4	7	5	8	6	9	7	9	12	10
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----

Exercice 2 :

- 1) Implémenter un type abstrait « chaîne de caractères » à l'aide de deux structures de données différentes.

Pour répondre à cette question, nous n'avons pas été très originaux car nous avons décidé d'utiliser deux structures bien connues, à savoir les **tableaux** et les **listes-chaînées**.

Pour implémenter un type abstrait, nos deux classes tableau et Liste ont donc les même méthodes, avec les même signatures.

Indispensable pour implémenter un type abstrait. L'utilisateur n'a pas à connaître comment sont construites chacune de nos classes mais simplement les méthodes qui les composent pour pouvoir les utiliser.

- Les tableaux

Pour créer notre classe tableau, nous avons décidé d'utiliser :

- Un tableau **value**.
 - Une variable **offset** qui nous indique à quel rang du tableau nous nous situons.
- Utilisation d'un getoffset.

```
private int getOffset() { return this.offset; } //testée indirectement JUnit
```

- Une variable **taille** qui représente la taille de notre tableau (initialement 0).

- Les Listes

Pour créer nos listes, nous avons décidé d'utiliser :

- Le type Élément qui constitue chaque nœud de notre liste. Un Élément est constitué d'un **elem** qui correspond à la valeur du nœud et d'un **suiv** qui est un pointeur vers l'élément suivant de notre liste. Évidemment si notre liste est vide ou que nous nous trouvons sur l'élément final, alors suiv est égal à null.
- La classe élément est constituée de plusieurs méthodes qui nous donnent :
L'élément actuel (getteurs et setteurs), l'éléments suivant (getteurs et setteurs).

```
public char getElement()  
{  
    return elem;  
}  
  
public void setElem(char elem)  
{  
    this.elem = elem;  
}  
  
public Element getSuivant()  
{  
    return suiv;  
}  
  
public void setSuivant (Element suiv)  
{  
    this.suiv = suiv;  
}
```

- Les méthodes de ces deux types :

Comme expliqué ci dessus nous avons donc choisi de créer les même méthodes (avec évidemment les même noms) pour les deux types.

1) La methode **estVide()**

Cette méthode nous renvoie un booléen qui nous dit si la chaîne est vide ou non.

```
/* Savoir si Liste Vide */  
public boolean isVide()  
{  
    return this.premier == null;  
}
```

Liste

```
public boolean isVide() //testée JUnit  
{  
    return (this.longueur() == 0);  
}
```

Tableau

2) La methode **IsEgal(chaîne c2)**

Cette méthode nous renvoie un booléen qui nous dit si la chaîne est égale ou non à la chaîne c2.

```
public boolean isEgal(Chaîne c2) //testée indirectement JUnit  
{  
    boolean ok_value = true;  
    if (this.longueur() == c2.longueur())  
    {  
        for (int i = 0; i < this.longueur(); i++)  
        {  
            if (this.charAt(i) != c2.charAt(i)) { ok_value = false; }  
        }  
    }  
    else  
    {  
        ok_value = false;  
    }  
    return (ok_value);  
}
```

Tableau

```
public boolean isEgal(Chaîne c2)  
{  
    boolean ok_value = true;  
    if (this.longueur() == c2.longueur())  
    {  
        for (int i = 0; i < this.longueur(); i++)  
        {  
            if (this.charAt(i) != c2.charAt(i)) { ok_value = false; }  
        }  
    }  
    else  
    {  
        ok_value = false;  
    }  
    return (ok_value);  
}
```

Chaîne

3) La méthode **Concaténer (char c)**

Méthode qui concatène à la fin d'une chaîne un caractère passé en paramètre.

```
/* Insertion d'un élément à la fin */
public void concatener(char c)
{
    Element e = new Element(c);
    Element ptr = getPremier();
    if (this.isVide())
    {
        this.premier = e;
    }
    else
    {
        while(ptr.getSuivant() != null)
        {
            ptr = ptr.getSuivant();
        }
        ptr.setSuivant(e);
    }
}
```

Liste

```
public void concatener(char c2) //testée JUnit
{
    char pvalue[] = new char[1];
    pvalue[0] = c2;
    Chaîne chaîne2 = new Chaîne(pvalue) ;
    this.concatener(chaîne2);
}
```

Tableau

4) La méthode **Concaténer (Chaîne c2)**

Méthode qui concatène deux chaînes.

```
public void concatener(Chaîne c2) //testée JUnit
{
    char tmp[];
    int taille = this.taille + c2.longueur();
    tmp = Arrays.copyOf(this.value, taille);
    for (int i = c2.getOffset(); i < c2.longueur(); i++)
    {
        tmp[this.taille + i] = c2.charAt(i);
    }
    this.taille = taille;
    this.value = tmp;
}
```

Tableau

```

public void concatener(Chaine c2)
{
    if (this.isVide())
    {
        this.premier = c2.getPremier();
    }
    else
    {
        Element ptr = this.getPremier();
        while (ptr.getSuivant() != null)
        {
            ptr = ptr.getSuivant();
        }
        ptr.setSuivant(c2.getPremier());
    }
}

```

Liste

5) Méthode **CharAt(int num)**

Méthode qui nous renvoie le caractère situé au rang num d'une chaîne.

```

public char charAt(int num)
{
    Element ptr = this.getPremier();
    while (num > 0)
    {
        if (this.isVide())
        {
            System.out.println("l'element n'existe pas !!!");
            return '0';
        }
        else
        {
            ptr = ptr.getSuivant();
            num--;
        }
    }

    return ptr.getElement();
}

```

Liste

```
public char charAt(int num) //testée JUnit
{
    return this.value[this.offset + num];
}
```

Tableau

6) Méthode **SousChaine (int start, int longueurf)**

Méthode qui nous renvoie une sous-chaîne avec start qui représente le rang où la sous-chaîne doit commencer et longueurf la longueur totale de la sous-chaîne.

Dans cette méthode, il faut bien penser à tous les cas.

En effet, si la longueurf + start > longueur de la chaîne alors il y a une erreur de saisie.

Si start > longueur de la chaîne alors il y a une erreur de saisie.

```
public Chaine souschaine(int start, int longueurf) //testée JUnit
{
    if (start < 0)
    {
        start = 0;
    }
    if (longueurf < 0)
    {
        longueurf = 0;
    }
    if (start > this.longueur() - 1)
    {
        this.supprimer();
        return this;
    }
    if (start + longueurf > this.longueur())
    {
        longueurf = this.longueur() - start;
    }
    this.offset += start;
    this.taille = this.offset + longueurf;
    return this;
}
```

Tableau

```

public Chaîne souschaîne(int start, int longueurf)
{
    Element ptr = this.getPremier();
    Chaîne ctr = new Chaîne();

    if (start<0)
    {
        start =0;
    }
    if (longueurf <0)
    {
        longueurf =0;
    }
    if (start > this.longueur()-1)
    {
        this.supprimer();
        return this;
    }
    if (start + longueurf > this.longueur())
    {
        longueurf = this.longueur()-start;
    }
    for (int i = 0; i < start; i++)
    {
        ptr = ptr.getSuivant();
    }
    for (int i = 0; i < longueurf; i++)
    {
        ctr.concatener(ptr.getElement()) ;
        ptr = ptr.getSuivant();
    }
    return ctr;
}

```

Liste

7) Méthode **Longueur ()**

Méthode qui nous renvoie un entier qui représente la taille de la longueur de la chaîne.

```

public int longueur()
{
    int taille =0;
    Element ptr = getPremier();
    while (ptr != null)
    {
        taille++;
        ptr = ptr.getSuivant();
    }
    return taille;
}

```

Liste

```

public int longueur() //testée indirectement JUnit
{
    return (this.taille - this.getOffset());
}

```

Tableau

8) Méthode **Afficher()**

Méthode qui nous affiche la chaîne.

```

public void afficher() //testée visuellement
{
    for (int i = this.offset; i < this.taille; i++)
    {
        System.out.print(this.value[i]);
    }
    System.out.println();
}

```

Tableau

```

public void afficher()
{
    Element ptr = this.getPremier();
    while (ptr != null)
    {
        System.out.print(ptr.getElement());
        ptr = ptr.getSuivant();
    }
    System.out.println ();
}

```

Liste

9) Méthode **Supprimer()**

Méthode qui supprime tous les éléments d'une chaîne.

```

public Chaine supprimer()
{
    this.premier=null;
    return this;
}

```

Liste

```

public void supprimer() // testée JUnit
{
    this.offset = 0;
    char pvalue[] = new char[0];
    this.value = pvalue;
    this.taille = pvalue.length;
}

```

Tableau

10) Méthode **dupliquer()**

La méthode dupliquer permet de dupliquer un liste en une autre liste.

```

public Chaine dupliquer()
{
    Element ptr = this.getPremier();
    Chaine c = new Chaine();
    for (int i = 0; i < this.longueur(); i++)
    {
        c.concatener(ptr.getElement());
        ptr = ptr.getSuivant();
    }
    return c;
}

```

Liste

```

public Chaine dupliquer()
{
    char t[] = new char[this.taille];
    for (int i = this.getOffset(); i < this.taille; i++)
    {
        t[i - this.getOffset()] = this.value[i];
    }
    return new Chaine(t);
}

```

Tableau

REMARQUE :

Pour toutes les méthodes de nos deux types, nous avons utilisé les tests Junit et ses différentes phases pour tester au fur et à mesure nos méthodes.

Afin de nous assurer que les deux implémentations étaient parfaitement interchangeables, nous avons utilisé exactement la même classe de test sur les deux packages.

Par exemple, pour tester la méthode CharAt(), nous avons fait le test Junit suivant :

```
@Test
public void testCharAt() {
    //Initialisation
    char tc1[] = { 'a', 'b', 'c' };
    Chaîne c1 = new Chaîne(tc1);
    char expected = 'b';

    //Invocation
    char actual = c1.charAt(1);

    //Vérification
    Assert.assertEquals(expected, actual);
}
```

2) procédure Annonce :

Pour créer la procédure Annonce, qui regarde si une chaîne passée en paramètre annonce une autre chaîne, nous avons créé une classe AnnonceMain que nous avons testé avec Junit.

Dans cette classe AnnonceMain, nous avons créé la méthode **Annonce (Chaîne c1, Chaîne c2)** qui regarde si la chaîne c1 annonce la chaîne c2 et qui renvoie une chaîne c3 qui représente l'annonce. Si c3 est null alors on renvoie une chaîne vide.

Nous avons donc regardé premièrement si $c1 = c2$. Si tel est le cas alors on renvoie c1 puisque l'annonce est égale à la chaîne c1.

Par exemple ($c1 = \text{valeur}$ et $c2 = \text{valeur}$).

Sinon, nous passons dans une boucle pour qui tant qu'un entier $i < c1.\text{longueur}()$ alors nous regardons $el1(i)$ de c1 avec $el2(j)$ de c2 (i et j initialiser à 0).

S'il sont égaux et que nous ne sommes pas au bout de c2 alors on ajoute $el1$ à $c3$ et on incrémente i et j. Si nous sommes au bout de c2 alors on vide c3 et on remet j à 0 et on recommence à regarder au début de c2. On incrémente toujours i.

Sinon on incrémente i et on remet j à 0.

Au final, on renvoie c3 qui correspond à l'annonce.

```
public static Chaîne Annonce(Chaîne c1, Chaîne c2) // testé avec Junit
{
    int i = 0, j = 0;
    Chaîne c3 = new Chaîne ();
    char el1;
    int long2 = c2.longueur();
    Chaîne t1, t2, souschaîne, debc2; boolean tout_reussi = true; //Pour
    les assertions
```



```

if (c1.isVide() || c2.isVide())
{
    System.out.println("Il ne peut pas y avoir d'annonce car au
moins une des deux listes est vide ");
    return c3;
}

char el2 = c2.charAt(0);

if (c1.isEgal(c2))
{
    return c1;
}
else
{
    for (i = 0; i < c1.longueur(); i++)
    {
        //ASSERTION
        // c3 contient les j caractères communs entre
        //la fin de (la souschaine de c1 du caractère 0 jusqu'au
        caractère i)
        //et le début de c2.
        t1 = c1.dupliquer(); t2 = c2.dupliquer();
        souschaine = t1.souschaine(i-j, j);
        debc2 = t2.souschaine(0, j);
        if (souschaine.isEgal(debc2) && c3.isEgal(debc2))
        { System.out.print(".");}
        else {System.out.print("F"); tout_reussi = false;}
        //FIN ASSERTION

        el1 = c1.charAt(i);

        if (el1 == el2)
        {
            long2--;
            if (long2 == 0 && i != c1.longueur())
            {
                c3.supprimer();
                j=0;
                el2 = c2.charAt(j);
                long2 = c2.longueur();
            }
            else
            {
                c3.concatener(el1);
                el2 = c2.charAt(++j);
            }
        }
        else
        {
            c3.supprimer();
            j=0;
            el2 = c2.charAt(j);
        }
        //ASSERTION
        // c3 contient les j caractères communs entre la fin de
        (la souschaine de c1 du caractère 0 jusqu'au caractère i
        + 1) et le début de c2.
    }
}

```

```

    t1 = c1.dupliquer(); t2 = c2.dupliquer();
    souschaine = t1.souschaine(i+1-j, j);
    debc2 = t2.souschaine(0, j);
    if (souschaine.isEgal(debc2) && c3.isEgal(debc2))
    { System.out.print(".");}
    else {System.out.print("F"); tout_reussi = false; }
    //FIN ASSERTION

}

//ASSERTION
// c3 contient les j caractères communs entre la fin de (la
// souschaine de c1 du caractère 0 jusqu'au caractère c1.longueur
// - 1) et le début de c2.
// c3 contient les j caractères communs entre la fin de (
c1
) et le début de c2.
t1 = c1.dupliquer(); t2 = c2.dupliquer();
souschaine = t1.souschaine(c1.longueur() - j, j);
debc2 = t2.souschaine(0, j);
if (souschaine.isEgal(debc2) && c3.isEgal(debc2))
{ System.out.print(".");}
else { System.out.print("F"); tout_reussi = false; }
//FIN ASSERTION

System.out.println();
if (tout_reussi) { System.out.println("tous les tests ont
réussi"); }
else {System.out.println("des tests ont échoué"); }

return c3;
}
}

```

Pour finir, nous avons créé dans le main de Annonce un petit algorithme qui permet à l'utilisateur de créer ses propres chaînes pour pouvoir regarder :

- Si une chaîne est égale à une autre.
- Si une chaîne annonce une autre chaîne.
- L'élément i d'une chaîne.
- La sous-chaîne d'une chaîne.
- La concaténation de deux chaîne.
- La suppression d'une chaîne.

Au final, l'utilisateur obtient le même résultat qu'on utilise le package ChaineTab ou le package ChaineListe.

En opérant aucune modification, le résultat reste donc le même. C'est le principe des types abstraits.