
SOMMAIRE :

Table des matières

PARTIE 1

Test 1 :

Pour ce test j'ai tout d'abord créer un programme calculant le temps en millisecondes pris pour remplir aléatoirement de lettre des ensembles de 10 valeurs

Le programme initialise tout d'abord les constantes et variables dont nous allons avoir besoin

```
/*----Constante----*/
final int TAILLE = 10;

/*----Variable-----*/
char[]    tabLet;
String    chaineLet;
long      tpsDebut, tpsFin, tpsDiff;
```

TAILLE est une constante contenant le nombre de valeurs dans nos différents ensemble.
tabLet est le tableau de caractère que nous allons utiliser pour nos tests.
chaineLet est la chaine de caractère que nous allons utiliser pour nos tests.
tpsDebut contient nombre de millisecondes écoulées depuis le 1er Janvier 1970 au début d'une insertion.
tpsFin contient nombre de millisecondes écoulées depuis le 1er Janvier 1970 a la fin d'une insertion.
tpsDiff contient la différence en milliseconde entre tpsFin et tpsDebut.

Ensuite mon programme est séparé en 3 autres parties :

La première calcule le temps en millisecondes que notre tableau de caractère va prendre à être remplis de lettres aléatoire.

```
tabLet = new char[TAILLE];
tpsDebut = System.currentTimeMillis();

for (int i = 0 ; i < TAILLE ; i++)
{
     tabLet[i] = (char)('A' + (int)(Math.random() * 26));
}

tpsFin = System.currentTimeMillis();
tpsDiff = tpsFin - tpsDebut;
System.out.println("Temps pour remplir le tableau : " + tpsDiff);
```

Tout d'abord notre programme instaure notre tableau et stock dans tpsDebut le nombre de millisecondes écoulées depuis le 1er Janvier 1970. Il utilise ensuite une boucle pour remplir notre tableau de lettre aléatoire. Une fois fini il stock dans tpsFin à nouveau le nombre de milliseconde

écoulées depuis le 1er Janvier 1970. Pour finir il soustrait tpsFin par tpsDebut pour obtenir le nombre de milliseconde écoulés depuis notre premier enregistrement, puis il l'affiche.

La deuxième partie calcule le temps en millisecondes que notre tableau de caractère va prendre à être remplis de lettres aléatoires.

```
chaineLet = "";
tpsDebut = System.currentTimeMillis();

for (int i = 0 ; i < TAILLE ; i++)
{
     chaineLet = chaineLet + (char)('A' + (int)(Math.random() *
26));
}

tpsFin = System.currentTimeMillis();
tpsDiff = tpsFin - tpsDebut;
System.out.println("Temps pour remplir la chaine avec
concatenation : " + tpsDiff);
```

Dans cette partie le programme fait exactement comme à la précédente mais au lien de remplir un tableau il concatène une chaîne de caractère.

Enfin la troisième et dernière partie calcule le temps en millisecondes qu'un string prend à être remplie a partir d'un tableau rempli.

```
tpsDebut = System.currentTimeMillis();

chaineLet = new String(tabLet);

tpsFin = System.currentTimeMillis();
tpsDiff = tpsFin - tpsDebut;
System.out.println("Temps pour remplir la chaine avec un tableau :
" + tpsDiff);
```

Dans cette partie le programme fait globalement pareil qu'a la première mais au lieu de faire une boucle pour remplir notre chaîne il l'instaure directement avec un tableau déjà rempli.

Après avoir fini le premier programme, je l'ai dupliqué plusieurs fois en changeant la valeur de TAILLE pour tester avec les différentes taille demandé dans le sujet.

Voici les résultats :

```
ll210679@di-720-05:~/Bureau/SAE_1_2/Test1$ java Test1_Ensemble10
Temps pour remplir le tableau : 1
Temps pour remplir la chaine avec concatenation : 7
Temps pour remplir la chaine avec un tableau : 0
ll210679@di-720-05:~/Bureau/SAE_1_2/Test1$ java Test1_Ensemble10000
Temps pour remplir le tableau : 2
Temps pour remplir la chaine avec concatenation : 26
Temps pour remplir la chaine avec un tableau : 1
ll210679@di-720-05:~/Bureau/SAE_1_2/Test1$ java Test1_Ensemble100000
Temps pour remplir le tableau : 4
Temps pour remplir la chaine avec concatenation : 568
Temps pour remplir la chaine avec un tableau : 2
ll210679@di-720-05:~/Bureau/SAE_1_2/Test1$ java Test1_Ensemble1000000
Temps pour remplir le tableau : 22
Temps pour remplir la chaine avec concatenation : 54988
Temps pour remplir la chaine avec un tableau : 5
```

Remplir \ Taille Ensemble	10	10 000	100 000	1 000 000
Tableau	1ms	2ms	4ms	22ms
Chaine (avec concaténation)	7ms	26ms	568ms	54 988ms
Chaine (avec tableau)	0ms	1ms	2ms	5ms

On peut observer qu'il est bien plus rapide de remplir un tableau en rajoutant les valeurs une à une qu'une chaîne. De plus une chaîne et bien plus rapide à remplir à l'aide d'un tableau déjà remplis que en concaténant. Nous pouvons donc conclure que pour remplir une chaîne le plus rapidement possible avec des valeurs aléatoire il faut remplir un tableau, puis instaurer notre chaîne à partir du tableau.

Test 2 :

Pour ce test j'ai procéder comme au test 1, j'ai tout d'abord créé un programme pour un ensembles de 10 valeurs que je dupliquerai plus tard. Celui-ci calcule l'occurrence de chaque lettre de l'alphabet, et surtout le temps qu'il met a faire tout cela avec un tableau et une chaine.

Ce programme initialise d'abord les mêmes variable qu'au test 1 mais rajouter un tableau d'entier tabOccu qui va contenir l'occurence pour chaque lettre.

```
/*----Constante----*/
final int TAILLE = 10;

/*----Variable-----*/
char[]    tabLet;
String    chaineLet;
long      tpsDebut, tpsFin, tpsDiff;
int[]     tabOccu;
```

J'ai ensuite instauré un tableau remplit de lettre aléatoire puis j'ai rempli une chaine de ses mêmes valeurs.

```
tabLet = new char[TAILLE];
for (int i = 0 ; i < TAILLE ; i++)
{
     tabLet[i] = (char)('A' + (int)(Math.random() * 26));
}
chaineLet = new String(tabLet);
```

Après j'ai calculé le temps pris pour calculer les occurrences de mon tableau :

```
tabOccu = new int[26];
tpsDebut = System.currentTimeMillis();

for (int i = 0 ; i < TAILLE ; i++)
{
     tabOccu[(int)(tabLet[i] - 'A')]++;
}

tpsFin = System.currentTimeMillis();
tpsDiff = tpsFin - tpsDebut;
System.out.println("Temps pour parcourir le tableau : " +
tpsDiff);
```

Dans cette partie du programme nous créons tout d'abord un tableau rempli d'entier égale à 0 pour calculer nos occurrences. Ensuite nous commençons a compter le temps d'exécution comme pour tout les programmes du Test 1. Nous parcourons les lettres de nos ensembles à l'aide d'une boucle puis nous situons l'emplacement de la lettre trouvé dans l'alphabet pour lui rajouté une valeur dans notre tableau d'occurence.

Pour finir nous je calcule le temps pris pour calculer mes occurrences de notre chaine :

```
tabOccu = new int[26];
tpsDebut = System.currentTimeMillis();
```

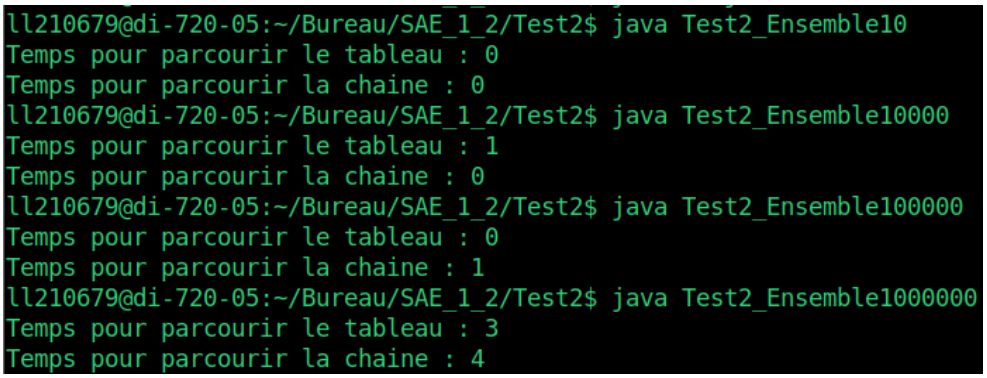
```
for (int i = 0 ; i < TAILLE ; i++)
{
     tabOccu[(int)(chaineLet.charAt(i) - 'A')]++;
}

tpsFin = System.currentTimeMillis();
tpsDiff = tpsFin - tpsDebut;
System.out.println("Temps pour parcourir la chaine : " + tpsDiff);
```

Ce bout de programme est globalement le même que celui d'au dessus sauf que la méthode pour récupérer les différentes lettre de notre chaîne est différente que pour le tableau.

Pour finir j'ai dupliqué les programmes puis changé la valeur de TAILLE comme dans le Test 1.

voici les résultats :



Parcourir \Taille Ensemble	10	10 000	100 000	1 000 000
Tableau	0ms	1ms	0ms	3ms
Chaine	0ms	0ms	1ms	4ms

Nous pouvons observer que les résultats sont globalement les mêmes, explorer un tableau est plus rapide mais à seulement 1 milliseconde prêt.

Test 3 :

Pour ce test j'ai procéder comme au 2 test d'avant, j'ai tout d'abord créé un programme pour un ensembles de 10 valeurs que je dupliquerai plus tard. Celui-ci remplace les voyelles majuscule par des minuscules, et surtout le temps qu'il met a faire tout cela avec un tableau et une chaîne.

Ce programme initialise d'abord les mêmes variable qu'au test 1 mais rajouter un tableau d'entier sansMajTabLet qui va nous servir à enlever les majuscules de notre tableau, et une chaîne sansMajChaineLet qui va nous servir à enlever les majuscules de notre chaîne.

```
/*----Constante----*/
final int TAILLE = 10;

/*----Variable-----*/
char[]    tabLet, sansMajTabLet;
String    chaineLet, sansMajChaineLet;
long      tpsDebut, tpsFin, tpsDiff;
```

J'ai ensuite instauré un tableau remplit de lettre aléatoire puis j'ai rempli une chaine de ses mêmes valeurs.

```
tabLet = new char[TAILLE];
for (int i = 0 ; i < TAILLE ; i++)
{
     tabLet[i] = (char)('A' + (int)(Math.random() * 26));
}
chaineLet = new String(tabLet);
```

Après ceci j'ai créé un sous algorithme qui prend en paramètre un caractère et détecte si c'est une majuscule voyelle. C'est à dire que si c'est une majuscule voyelle le sous programme renvoie `true` et si ce n'est pas le cas `false`.

```
private static boolean estVoyelle (char let)
{
     if (let == 'A' || let == 'E' || let == 'I' || let == 'O' ||
let == 'U' || let == 'Y')
     {
          return true;
     }
     return false;
}
```

Ensuite j'ai codé la partie qui va transformer les voyelles majuscule en minuscule du tableau tout en calculant le temps en milliseconde que cela prend.

```
sansMajTabLet = new char[TAILLE];
tpsDebut = System.currentTimeMillis();

for (int i = 0 ; i < TAILLE ; i++)
{
     if (estVoyelle(tabLet[i]))
     {
          sansMajTabLet[i] = (char)(tabLet[i] + 32);
     }
     else
```

```
	{
		sansMajTabLet[i] = tabLet[i];
	}
}
tabLet = sansMajTabLet;

tpsFin = System.currentTimeMillis();
tpsDiff = tpsFin - tpsDebut;
System.out.println("Temps pour traiter le tableau : " + tpsDiff);
```

Pour l'expliquer simplement, nous créons un tableau sansMajTabLet de la même taille que tabLet. Ensuite nous explorons tabMaj et si la lettre exploré est une voyelle alors nous l'ajoutons à sansMajTabLet en minuscule, sinon nous l'ajoutons à sansMajTabLet sans modification. A la fin nous remplacons les valeurs de tabLet par celle de sansMajLet, ainsi toutes les voyelles sous devenu des minuscules. Bien évidemment le temps d'exécution de tout ça est calculé et de la même manière qu'au 2 Tests précédent.

Ensuite j'ai codé la partie qui va faire pareil que ce que nous venons de voir mais pour les chaînes.

(voir feuille suivante)

```
sansMajChaineLet = "";
tpsDebut = System.currentTimeMillis();

for (int i = 0 ; i < TAILLE ; i++)
{
	if (estVoyelle(chaineLet.charAt(i)))
	{
		sansMajChaineLet = sansMajChaineLet + ((char)
(chaineLet.charAt(i) + 32));
	}
	else
	{
		sansMajChaineLet = sansMajChaineLet +
chaineLet.charAt(i);
	}
```

```
}
chaineLet = sansMajChaineLet;

tpsFin = System.currentTimeMillis();
tpsDiff = tpsFin - tpsDebut;
System.out.println("Temps pour traiter la chaine : " + tpsDiff);
```

Le programme fait exactement pareil que pour les tableau mais cette fois-ci avec des chaînes.

Pour finir j'ai dupliqué les programmes puis changé la valeur de TAILLE comme dans les Tests précédent.

voici les résultats :

```
ll210679@di-720-05:~/Bureau/SAE_1_2/Partie_1/Test_3$ java Test3_Ensemble10
Temps pour traiter le tableau : 0
Temps pour traiter la chaine : 8
ll210679@di-720-05:~/Bureau/SAE_1_2/Partie_1/Test_3$ java Test3_Ensemble10000
Temps pour traiter le tableau : 1
Temps pour traiter la chaine : 27
ll210679@di-720-05:~/Bureau/SAE_1_2/Partie_1/Test_3$ java Test3_Ensemble100000
Temps pour traiter le tableau : 1
Temps pour traiter la chaine : 563
ll210679@di-720-05:~/Bureau/SAE_1_2/Partie_1/Test_3$ java Test3_Ensemble1000000
Temps pour traiter le tableau : 6
Temps pour traiter la chaine : 53873
```

On peut observer que traiter un tableau est beaucoup plus rapide que de traiter une chaîne.

Traiter \ Taille Ensemble	10	10 000	100 000	1 000 000
Tableau	0ms	1ms	1ms	6ms
Chaine	8ms	27ms	563ms	53 873ms

PARTIE 2

Test 1 :

Le programme est le même qu'au Test 1 de la Partie 1 mais adapté au StringBuilder

```
ll210679@di-720-05:~/Bureau/SAE_1_2/Partie_2/Test_1$ java Test1_Ensemble10
Temps pour remplir le tableau : 0
Temps pour remplir la chaine avec concatenation : 0
Temps pour remplir la chaine avec un tableau : 0
ll210679@di-720-05:~/Bureau/SAE_1_2/Partie_2/Test_1$ java Test1_Ensemble10000
Temps pour remplir le tableau : 1
Temps pour remplir la chaine avec concatenation : 1
Temps pour remplir la chaine avec un tableau : 0
ll210679@di-720-05:~/Bureau/SAE_1_2/Partie_2/Test_1$ java Test1_Ensemble100000
Temps pour remplir le tableau : 4
Temps pour remplir la chaine avec concatenation : 4
Temps pour remplir la chaine avec un tableau : 3
ll210679@di-720-05:~/Bureau/SAE_1_2/Partie_2/Test_1$ java Test1_Ensemble1000000
Temps pour remplir le tableau : 22
Temps pour remplir la chaine avec concatenation : 23
Temps pour remplir la chaine avec un tableau : 6
```

On peut remarquer que remplir de lettre aléatoire un StringBuilder est de la même rapidité que le faire dans un tableau de caractère. Il est donc plus optimiser de créer une chaîne de caractère grâce à un StringBuilder que grâce à un tableau car celui-ci possède une étape en plus lui ajoutant du temps d'exécution et donc le rendant plus lent que le StringBuilder.

Test 2 :

Le programme est le même qu'au Test 1 de la Partie 1 mais adapté au StringBuilder.

```
ll210679@di-720-05:~/Bureau/SAE_1_2/Partie_2/Test_2$ java Test2_Ensemble10
Temps pour parcourir le tableau : 0
Temps pour parcourir la chaine : 0
ll210679@di-720-05:~/Bureau/SAE_1_2/Partie_2/Test_2$ java Test2_Ensemble10000
Temps pour parcourir le tableau : 1
Temps pour parcourir la chaine : 1
ll210679@di-720-05:~/Bureau/SAE_1_2/Partie_2/Test_2$ java Test2_Ensemble100000
Temps pour parcourir le tableau : 0
Temps pour parcourir la chaine : 2
ll210679@di-720-05:~/Bureau/SAE_1_2/Partie_2/Test_2$ java Test2_Ensemble1000000
Temps pour parcourir le tableau : 4
Temps pour parcourir la chaine : 6
```

Nous pouvons observer que les résultats sont globalement les mêmes, explorer un tableau est plus rapide mais à seulement 2 milliseconde prêt.

PARTIE 3

Les String en java représente un type de donnée dans plein de langage, comme toutes type celles-ci possèdes des avantages et des inconvénients. C'est ce que nous allons traiter de suite.

AVANTAGES

- La saisie par les utilisateurs :

Les Strings sont très pratiques pour accueillir des valeurs entrée par un utilisateur. Celles-ci permet de stocker une énorme quantité de caractères, ce que peut de type font.

- Facilement manipulable :

Les Strings sont simple à manipuler car il existe des méthodes très efficaces pour leurs apporter des modification, comme par exemple la méthode .replace qui change le/les caractères mis en premier paramètre pour les changer par le/les caractères mis en second paramètre.

- Affichage console :

Les Strings sont tout le temps utilisé lorsqu'il est nécessaire de faire un affichage console. Il sont simple à renseigner et facilement personnalisable.

INCONVÉNIENTS

- Lent à concaténer :

Les Strings sont bien pratiques dans l'ensemble, cependant comme vu précédemment beaucoup utiliser la concaténation est très lent sachant qu'il exister des méthodes bien plus optimiser comme par exemple, remplir un tableau de caractère.

- Lent à traiter :

Comme nous avons pu le constater lors du Test 3 de la Partie 1, les Strings prennent beaucoup de temps à être traiter, ils ne sont pas le type de donnée le plus adapté pour traiter des caractères en masse.

CONCLUSION

Pour conclure nous pouvons dire que les Strings sont un type de données très pratiques et facile à utiliser, seulement ils ne sont pas très optimiser en terme de rapidité et peuvent énormément ralentir un programme, il existe beaucoup mieux comme les StringBuilders qui en sont beaucoup plus rapide.