

UNIVERSITE DE BOUIRA

FACULTE DES SCIENCES ET DES SCIENCES APPLIQUEES

DEPARTEMENT D'INFORMATIQUE

LICENCE INFORMATIQUE, 4^{ème} SEMESTRE (L2)

MODULE : PROGRAMMATION ORIENTEE OBJET

Semaine du 02/05/2021 au 06/05/2020

MISE A NIVEAU JAVA

TP 03 :

TRAITEMENT DES CHAINES DE CARACTERES

MISE EN OEUVRE DE LA CLASSE String

1. Introduction Brève à la classe String

(Ce tutoriel a été repris du site <https://devstory.net/10217/java-string-stringbuffer-stringbuilder>)

Pour les méthode de la classe String se concentrer sur celle reportée dans un fond jaune (tableau des méthode)

String est l'une des classes très importantes en **Java** et toute personne qui commence par la programmation **Java** doit utiliser des instructions célèbres **System.out.println ()** pour imprimer quelque chose sur la **Console**. Beaucoup de débutants de Java ne savent pas que **String** est immuable et est **final** (c'est-à-dire qu'il n'autorise pas de la classe à hériter), toutes des modifications apportées sur **String** produisent un autre objet **String**.

3.1- String est vraiment une classe très spéciale

En Java, String est une classe spéciale, la raison est qu'elle est régulièrement utilisée dans un programme donc elle doit contenir la performance et la flexibilité nécessaires. C'est pourquoi **String** a un objet et deux propriétés primitifs.

String est primitif :

Vous pouvez créer un **string literal** (chaîne littérale), **string literal** est stocké dans la pile (stack), n'a pas besoin de grand espace de stockage, et est moins coûteux à manipuler.

- String literal = "Hello World";

Vous pouvez utiliser l'opérateur + pour joindre deux chaînes, cet opérateur est familier et appliqué au type de données primitives **int**, **float**, **double**.

Les **string literal** sont contenus dans un réservoir (common pool). Donc les deux strings literal ont le même contenu en utilisant la même zone de mémoire sur la pile, cela permet d'économiser de la mémoire.

String est un objet

Puisque **String** est une classe donc elle peut être créée par le **nouvel** opérateur.

- String object = new String("Hello World");

L'objet **String** est stocké sur Heap nécessite de gérer la complexité et le coût de l'espace de stockage de la mémoire.

Deux **Strings** d'objets ont le même contenu stocké sur deux zones différentes de la mémoire de pile.

Par exemple :

```
?
1
2 // Implicit construction via string literal
3 String str1 = "Java is Hot";
4
5 // Explicit construction via new
6 String str2 = new String("I'm cool");
```

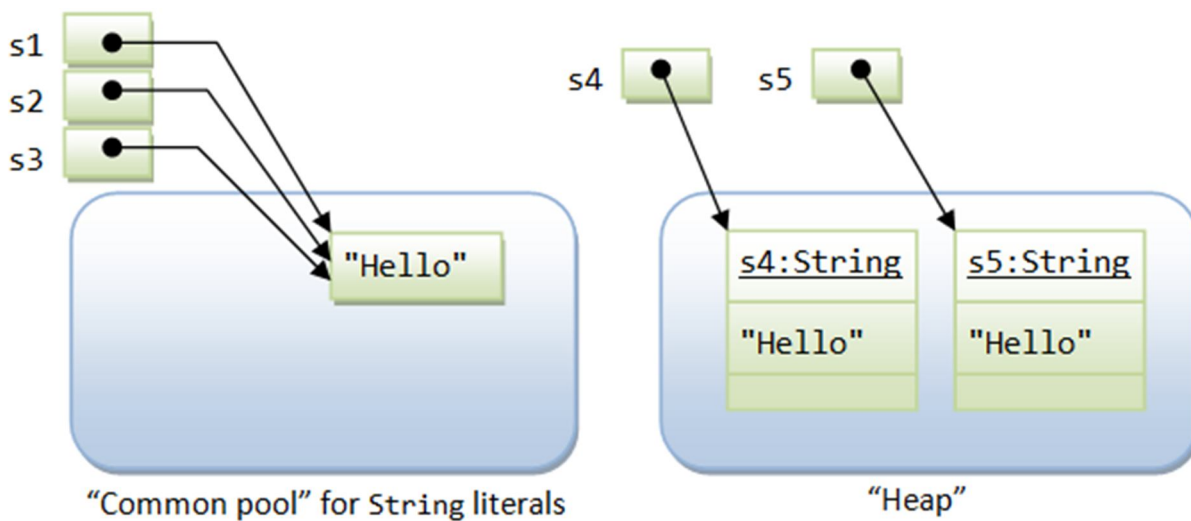
3.2- String Literal vs. String Object

Tel que mentionné, il y a deux façons de construction d'une chaîne (**String**) : construction implicite en assignant une chaîne littérale (String literal) ou en créant explicitement un **String** objet via le nouvel opérateur et le constructeur. Par exemple :

```
?
1 String s1 = "Hello"; // String literal
2 String s2 = "Hello"; // String literal
3 String s3 = s1; // same reference
4 String s4 = new String("Hello"); // String object
5 String s5 = new String("Hello"); // String object
```

Nous allons expliquer par l'illustration ci-dessous :

```
String s1 = "Hello"; // String literal
String s2 = "Hello"; // String literal
String s3 = s1; // same reference
String s4 = new String("Hello"); // String object
String s5 = new String("Hello"); // String object
```



Les **strings literal** ont le même contenu, elles partageront le même emplacement de stockage dans le réservoir commun (common pool). Pendant ce temps, les objets String sont stockés dans le Heap et ne partagent pas les emplacements de stockage, y compris deux objets string qui ont le même contenu.

equals() vs ==

La méthode **equals()** sert à comparer deux objets, pour String il signifie la comparaison du contenu de deux chaînes. Pour les types de référence (reference), l'opérateur **==** signifie la comparaison de l'adresse de la zone de stockage des objets. Observez l'exemple :

```
?
1 String s1 = "Hello"; // String literal
2 String s2 = "Hello"; // String literal
3 String s3 = s1; // same reference
4 String s4 = new String("Hello"); // String object
5 String s5 = new String("Hello"); // String object
6
7 s1 == s1; // true, same pointer
8 s1 == s2; // true, s1 and s2 share storage in common pool
9 s1 == s3; // true, s3 is assigned same pointer as s1
10 s1 == s4; // false, different pointers
11 s4 == s5; // false, different pointers in heap
12
13 s1.equals(s3); // true, same contents
14 s1.equals(s4); // true, same contents
15 s4.equals(s5); // true, same contents
```

En fait, vous devez utiliser **string literal**, au lieu d'utiliser l'opérateur **"new"** qui vous permet d'accélérer votre programme.

3.3- Les méthodes de String

Voici les méthodes de **String** :

SN	Methods	Description
1	char charAt(int index)	Returns the character at the specified index.
2	int compareTo(Object o)	Compares this String to another Object.
3	int compareTo(String anotherString)	Compares two strings lexicographically.
4	int compareToIgnoreCase(String str)	Compares two strings lexicographically, ignoring case differences.
5	String concat(String str)	Concatenates the specified string to the end of this string.

6	<code>boolean contentEquals(StringBuffer sb)</code>	Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
7	<code>static String copyValueOf(char[] data)</code>	Returns a String that represents the character sequence in the array specified.
8	<code>static String copyValueOf(char[] data, int offset, int count)</code>	Returns a String that represents the character sequence in the array specified.
9	<code>boolean endsWith(String suffix)</code>	Tests if this string ends with the specified suffix.
10	<code>boolean equals(Object anObject)</code>	Compares this string to the specified object.
11	<code>boolean equalsIgnoreCase(String anotherString)</code>	Compares this String to another String, ignoring case considerations.
12	<code>byte getBytes()</code>	Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
13	<code>byte[] getBytes(String charsetName)</code>	Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.
14	<code>void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)</code>	Copies characters from this string into the destination character array.
15	<code>int hashCode()</code>	Returns a hash code for this string.
16	<code>int indexOf(int ch)</code>	Returns the index within this string of the first occurrence of the specified character.
17	<code>int indexOf(int ch, int fromIndex)</code>	Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
18	<code>int indexOf(String str)</code>	Returns the index within this string of the first occurrence of the specified substring.
19	<code>int indexOf(String str, int fromIndex)</code>	Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
20	<code>String intern()</code>	Returns a canonical representation for the string object.
21	<code>int lastIndexOf(int ch)</code>	Returns the index within this string of the last occurrence of the specified character.
22	<code>int lastIndexOf(int ch, int fromIndex)</code>	Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
23	<code>int lastIndexOf(String str)</code>	Returns the index within this string of the rightmost occurrence of the specified substring.
24	<code>int lastIndexOf(String str, int fromIndex)</code>	Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
25	<code>int length()</code>	Returns the length of this string.
26	<code>boolean matches(String regex)</code>	Tells whether or not this string matches the given regular expression.
27	<code>boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)</code>	Tests if two string regions are equal.
28	<code>boolean regionMatches(int toffset, String other, int ooffset, int len)</code>	Tests if two string regions are equal.
29	<code>String replace(char oldChar, char newChar)</code>	Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

30	String replaceAll(String regex, String replacement)	Replaces each substring of this string that matches the given regular expression with the given replacement.
31	String replaceFirst(String regex, String replacement)	Replaces the first substring of this string that matches the given regular expression with the given replacement.
32	String[] split(String regex)	Splits this string around matches of the given regular expression.
33	String[] split(String regex, int limit)	Splits this string around matches of the given regular expression.
34	boolean startsWith(String prefix)	Tests if this string starts with the specified prefix.
35	boolean startsWith(String prefix, int toffset)	Tests if this string starts with the specified prefix beginning a specified index.
36	CharSequence subSequence(int beginIndex, int endIndex)	Returns a new character sequence that is a subsequence of this sequence.
37	String substring(int beginIndex)	Returns a new string that is a substring of this string.
38	String substring(int beginIndex, int endIndex)	Returns a new string that is a substring of this string.
39	char[] toCharArray()	Converts this string to a new character array.
40	String toLowerCase()	Converts all of the characters in this String to lower case using the rules of the default locale.
41	String toLowerCase(Locale locale)	Converts all of the characters in this String to lower case using the rules of the given Locale.
42	String toString()	This object (which is already a string!) is itself returned.
43	String toUpperCase()	Converts all of the characters in this String to upper case using the rules of the default locale.
44	String toUpperCase(Locale locale)	Converts all of the characters in this String to upper case using the rules of the given Locale.
45	String trim()	Returns a copy of the string, with leading and trailing whitespace omitted.
46	static String valueOf(primitive data type x)	Returns the string representation of the passed data type argument.

3.3.1- length()

LengthDemo.java

```

?
1  package org.o7planning.tutorial.str;
2
3  public class LengthDemo {
4
5      public static void main(String[] args) {
6          String str = "This is text";
7          int len = str.length();
8          System.out.println("String Length is : " + len);
9      }
10 }

```

Les résultats de l'exécution d'exemple :

The screenshot shows an IDE console window with the following content:

```

<terminated> LengthDemo [Java Application] D:\DevPrograms\Java\jdk1.7.0_45\bin\javaw.exe (Oct 6, 2014, 11:38:37 PM)
String Length is : 12

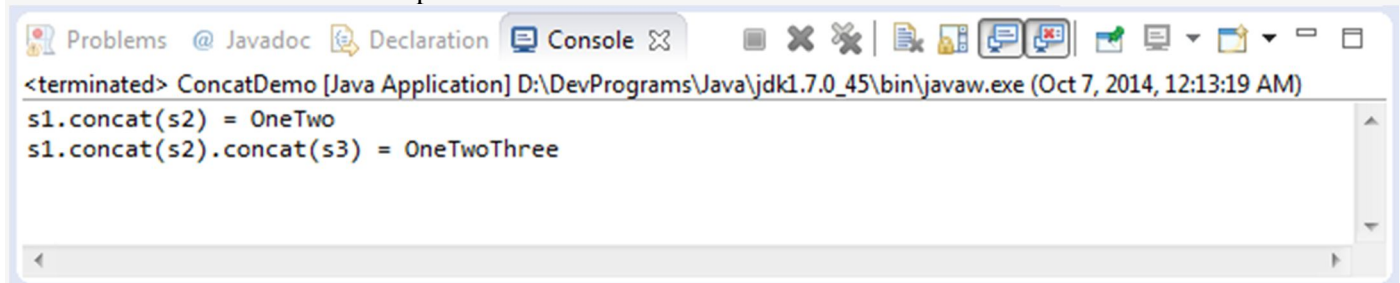
```

3.3.2- concat(String)

ConcatDemo.java

```
?  
1 package org.o7planning.tutorial.str;  
2  
3 public class ConcatDemo {  
4  
5     public static void main(String[] args) {  
6         String s1 = "One";  
7         String s2 = "Two";  
8         String s3 = "Three";  
9  
10        // s1.concat(s2) same as s1 + s2  
11        String s = s1.concat(s2);  
12        System.out.println("s1.concat(s2) = " + s);  
13  
14        // s1.concat(s2).concat(s3) same as s1 + s2 + s3;  
15        s = s1.concat(s2).concat(s3);  
16  
17        System.out.println("s1.concat(s2).concat(s3) = " + s);  
18    }  
19 }
```

Les résultats de l'exécution d'exemple :



```
Problems @ Javadoc Declaration Console  
<terminated> ConcatDemo [Java Application] D:\DevPrograms\Java\jdk1.7.0_45\bin\javaw.exe (Oct 7, 2014, 12:13:19 AM)  
s1.concat(s2) = OneTwo  
s1.concat(s2).concat(s3) = OneTwoThree
```

3.3.3- indexOf(..)

```
String str = "This is text";
```

```
int idx = str.indexOf("te");
```



```
int idx = str.indexOf('i');
```

```
idx = str.indexOf('i', 4)
```

IndexOfDemo.java

```
?  
1 package org.o7planning.tutorial.str;  
2  
3 public class IndexOfDemo {  
4
```

```

5     public static void main(String[] args) {
6         String str = "This is text";
7
8         // Find index within this string of the first occurrence 'i'.
9         // ==> 2
10        int idx = str.indexOf('i');
11        System.out.println("- indexOf('i') = " + idx);
12
13
14        // Find index within this string of the first occurrence 'i'
15        // starting the search at index 4.
16        // ==> 5
17        idx = str.indexOf('i', 4);
18        System.out.println("- indexOf('i',4) = " + idx);
19
20        // index within this string of the first occurrence of "te".
21        // ==> 8
22        idx = str.indexOf("te");
23        System.out.println("- indexOf('te') = " + idx);
24    }
25 }
26

```

Les résultats de l'exécution d'exemple :

```

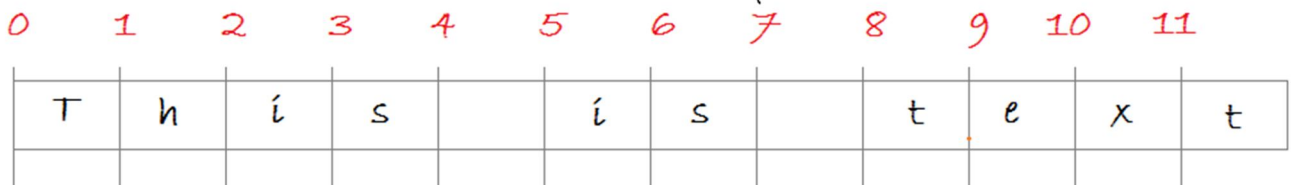
<terminated> IndexOfDemo [Java Application] D:\DevPrograms\Java\jdk1.7.0_45\bin\javaw.exe (Oct 9, 2014, 3:15:16 PM)
- indexOf('i') = 2
- indexOf('i',4) = 5
- indexOf('te') = 8

```

3.3.4- substring(..)

```
String str = "This is text";
```

```
String substr = str.substring(2,7)
```

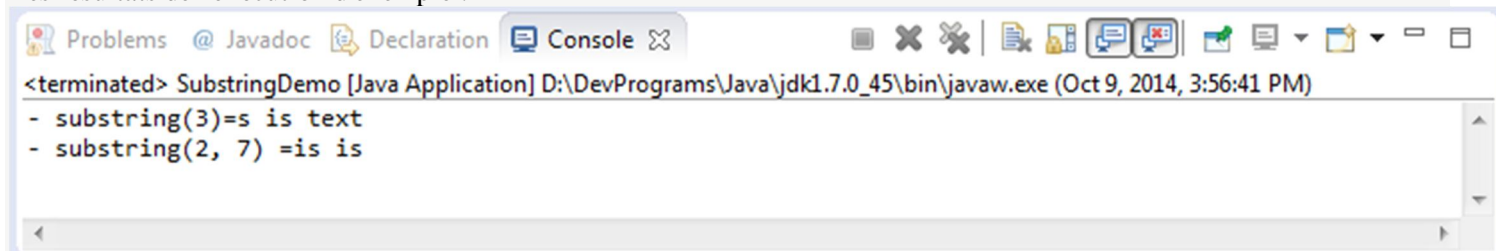


```
String substr = str.substring(3)
```

SubstringDemo.java

```
?
1 package org.o7planning.tutorial.str;
2
3 public class SubstringDemo {
4
5     public static void main(String[] args) {
6         String str = "This is text";
7
8         // Returns the substring from index 3 to the end of string.
9         String substr = str.substring(3);
10
11        System.out.println("- substring(3)=" + substr);
12
13        // Returns the substring from index 2 to index 7.
14        substr = str.substring(2, 7);
15
16        System.out.println("- substring(2, 7) =" + substr);
17    }
18 }
```

Les résultats de l'exécution d'exemple :



```
<terminated> SubstringDemo [Java Application] D:\DevPrograms\Java\jdk1.7.0_45\bin\javaw.exe (Oct 9, 2014, 3:56:41 PM)
- substring(3)=s is text
- substring(2, 7) =is is
```

3.3.5- replace

Quelques méthodes relatives au remplacement.

```
?
1 // Returns a new string resulting from replacing all occurrences of
2 // oldChar in this string with newChar.
3 public String replace(char oldChar, char newChar)
4
5 // Replaces each substring of this string that matches the given
6 // 'regular expression' with the given replacement.
7 public String replaceAll(String regex, String replacement)
8
9 // Replaces the first substring of this string that matches
10 // the given <'regular expression' with the given replacement.
11 public String replaceFirst(String regex, String replacement)
```

ReplaceDemo.java

```
?
1 package org.o7planning.tutorial.str;
2
3 public class ReplaceDemo {
4
5     public static void main(String[] args) {
6         String str = "This is text";
7
8         // Replace the character 'i' by 'x'.
9         String s2 = str.replace('i', 'x');
10
11        System.out.println("- s2=" + s2);
12
13        // Replace all the strings match "is" by "abc". (Regular Expression)
14        String s3 = str.replaceAll("is", "abc");
15
16        System.out.println("- s3=" + s3);
17
18        // Replaces the first substring of this string that matches "is" by "abc".
19        String s4 = str.replaceFirst("is", "abc");
```

```

20     System.out.println("- s4=" + s4);
21
22     // (See also document the regular expression)
23     // Replace all substring matching expression:
24     // "is|te": means "is" or "te" replaced by "+".
25     String s5 = str.replaceAll("is|te", "+");
26     System.out.println("- s5=" + s5);
27 }
28 }
29 }
30

```

Les résultats de l'exécution d'exemple :

```

<terminated> ReplaceDemo [Java Application] D:\DevPrograms\Java\jdk1.7.0_45\bin\javaw.exe (Oct 9, 2014, 7:07:03 PM)
- s2=Thxs xs text
- s3=Thabc abc text
- s4=Thabc is text
- s5=Th+ + +xt

```

3.3.6- Les autres exemples

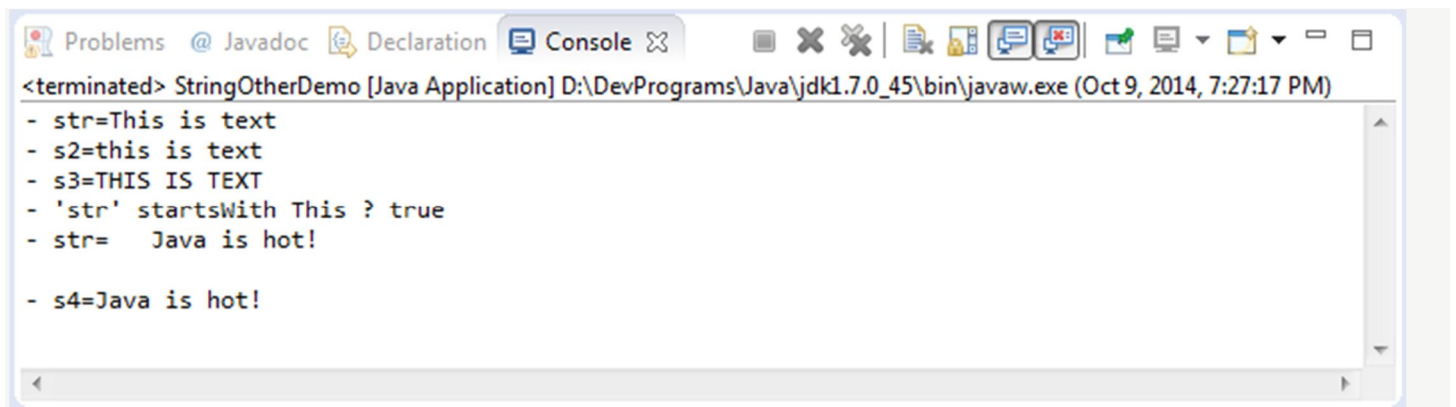
StringOtherDemo.java

```

?
1
2 package org.o7planning.tutorial.str;
3
4 public class StringOtherDemo {
5
6     public static void main(String[] args) {
7         String str = "This is text";
8
9         System.out.println("- str=" + str);
10
11        // Return lower case string.
12        String s2 = str.toLowerCase();
13
14        System.out.println("- s2=" + s2);
15
16        // Return upper case string
17        String s3 = str.toUpperCase();
18
19        System.out.println("- s3=" + s3);
20
21        // Check string started by "This" or not.
22        boolean swith = str.startsWith("This");
23
24        System.out.println("- 'str' startsWith This ? " + swith);
25
26        // A string with whitespace in beginning and end.
27        // Note: \t is tab character
28        // \n is new line character
29        str = " \t Java is hot! \t \n ";
30
31        System.out.println("- str=" + str);
32
33        // Returns a copy of the string, with leading and trailing whitespace omitted.
34        String s4 = str.trim();
35
36        System.out.println("- s4=" + s4);
37    }
38 }

```

Les résultats de l'exécution d'exemple :



```
<terminated> StringOtherDemo [Java Application] D:\DevPrograms\Java\jdk1.7.0_45\bin\javaw.exe (Oct 9, 2014, 7:27:17 PM)
- str=This is text
- s2=this is text
- s3=THIS IS TEXT
- 'str' startsWith This ? true
- str= Java is hot!

- s4=Java is hot!
```

Travail 1 : expérimenter tous les codes sources du ttoriel ci-dessus

Travail 2 : réaliser la méthode `int occurrence(String str, String substr)` : cette méthode compte le nombre de fois que la sous chaîne de caractère apparaît dans la chaîne `str`