Java

Strings

String related classes

- Java provides four String related classes
- java.lang package
  - ***String*** class: Storing and processing Strings but Strings created using the String class cannot be modified (**immutable**)
  - ***StringBuffer/StringBuilder*** class: Create flexible Strings that can be modified
- java.util package
  - ***StringTokenizer*** class: Can be used to extract tokens from a String

String

String

- String class provide many constructors and more than 40 methods for examining in individual characters in a sequence
- You can create a String from a String value or from an array of characters.
  - String newString = new String(stringValue);
- The argument stringValue is a sequence of characters enclosed inside double quotes
  - String message = new String (“Welcome”);
  - String message = “Welcome”;

String Constructors

```
3   public class StringConstructorTest {
4       public static void main(String[] args) {
5           char charArray[ ] = { 'b',  'i',  'r',  't',  'h',  ' ',  'd',  'a',  'y'} ;
6           byte byteArray[ ] = { ( byte)  'n',  ( byte)  'e',  ( byte)  'w', ( byte)  ' ',
7                   ( byte)  'y',  ( byte)  'e',  ( byte)  'a',  ( byte)  'r'} ;
8
9           String s = new String("hello"); // hello
10          String s1 = new String() ; //
11          String s2 = new String(s) ; // hello
12          String s3 = new String(charArray); // birth day
13          String s4 = new String(charArray, 6, 3); // day
14          String s5 = new String(byteArray, 4, 4); // year
15          String s6 = new String(byteArray); // new year
16          String s7 = "Wel"  +  "come"; // Welcome
17
18          System.out.println(s);
19          System.out.println(s1);
20          System.out.println(s2);
21          System.out.println(s3);
22          System.out.println(s4);
23          System.out.println(s5);
24          System.out.println(s6);
25          System.out.println(s7);
26      }
27  }
```

String Length

- Returns the length of a String
  - ***length()***
- Example:

  String s1="Hello";

  System.out.println(s1.length());

Extraction

- Get the character at a specific location in a string
  - ***s1.charAt(1)***
- Get the entire set of characters in a string
  - ***s1.getChars(0, 5, charArray, 0)***

Extracting Substrings

- substring method enable a new String object to be created by copying part of an existing String object
  - ***substring (int startIndex)*** - copies the characters form the starting index to the end of the String
  - ***substring(int beginIndex, int endIndex)*** - copies the characters from the starting index to one beyond the endIndex

String Comparisons

- *equals*
  - Compare any two string objects for equality using lexicographical comparison. ***s1.equals("hello")***
- *equalsIgnoreCase*
  - ***s1.equalsIgnoreCase(s2)***
- *compareTo*
  - ***s1.compareTo(s2)***
  - s1 > s2 (positive), s1 < s2 (negative), s1 = s2 (zero)

String Comparisons

```java
public class StringEqualsTest {
    public static void main(String[] args) {
        String s1 = "Hello";
        String s2 = new String( original: "Hello");
        String s3 = "Hello";
        System.out.println("s1 == Hello " + s1.equals("Hello")); // true
        System.out.println("s1 == s2 " + s1.equals(s2)); // true
        System.out.println("s1 == s3 " + s1.equals(s3)); // true
        System.out.println("s2 == s3 " + s2.equals(s3)); // true
        System.out.println(s1 == s2); // false
        System.out.println(s1 == s3); // true
        System.out.println(s2 == s3); // false
    }
}
```

For details have a look at section 1-4 from https://www.baeldung.com/java-string-pool

String Comparisons

- *regionMatches* compares portions of two String objects for equality
  - ***s1.regionMatches (0, s2, 0, 5)***
  - ***s1.regionMatches (true, 0, s2, 0, 5)***
- If the first argument is true, the method ignores the case of the characters being compared
- *startsWith* and endsWith check whether a String starts or ends with a specified String
  - ***s1.startsWith (s2)***
  - ***s1.endsWith (s2)***

String Concatenation

- Java provide the *concat* method to concatenate two strings.

  String s1 = new String (“Happy ”);

  String s2 = new String (“Birthday”);

  String s3 = s1.concat(s2);

  s3 will be “Happy Birthday”

String Search

- Find the position of character/String within a String
  - ***int indexOf(char ch)***
  - ***int lastIndexOf(char ch)***

String Split

- split() method splits a String against given regular expression and returns a character array
- ***String test = "abc,def,123";***

  String[] out = test.split(",");

  out[0] - abc, out[1] - def, out[2] - 123

String Conversions

- Generally, the contents of a String cannot be changed once the string is created,
- Java provides conversion methods
- ***toUpperCase()*** and ***toLowerCase()***
  - Converts all the characters in the string to lowercase or uppercase
- ***trim()***
  - Eliminates blank characters from both ends of the string
- ***replace(oldChar, newChar)***
  - Replaces a character in the string with a new character

String to Other Conversions

- The String class provides ***valueOf*** methods for converting a character, an array of characters and numeric values to strings
  - ***valueOf*** method take different argument types

String to Other Conversions

Type	To String	From String
boolean	String.valueOf(boolean)	Boolean.parseBoolean(String)
byte	String.valueOf(int)	Byte.parseByte(String, int base)
short	String.valueOf(int)	Short.parseShort (String, int base)
Int	String.valueOf(int)	Integer.parseInt (String, int base)
long	String.valueOf(long)	Long.parseLong (String, int base)
float	String.valueOf(float)	Float.parseFloat(String)
double	String.valueOf(double)	Double.parseDouble(String)

String Conversion Example

- To convert an int to a String (3 different ways):

  int n = 123;

  String s1 = Integer.toString(n);

  String s2 = String.valueOf(n);

  String s3 = n + "";

- To convert a string to an int:

  String s = "1234";

  int n = Integer.parseInt(s);

StringBuffer/StringBuilder

StringBuffer

- Can be used wherever a string is used
  - More flexible than String
  - Can add, insert, or append new contents into a string buffer
- The StringBuffer class has three constructors and more than 30 methods for managing the buffer and for modifying strings in the buffer
- Every StringBuffer is capable of storing a number of characters specified by its capacity

StringBuffer Constructors

- ***public StringBuffer()***
  - No characters in it and an initial capacity of 16 characters
- ***public StringBuffer(int length)***
  - No characters in it and an initial capacity specified by the length argument
- ***public StringBuffer(String string)***
  - Contains String argument and an initial capacity of the buffer is 16 plus the length of the argument

StringBuilder

- The functionalities of StringBuilder is the same as StringBuffer, but StringBuilder is faster
- StringBuffer is synchronized, StringBuilder is not
- StringBuffer came first
  - Sun was concerned with correctness under all conditions, they made it synchronized to make thread-safe just in case
- StringBuilder came later
  - Drop-in replacement for StringBuffer without the synchronization, not thread-safe

StringBuffer/StringBuilder

```
1  public class StringBufferTest {
2      public static void main(String[] args) {
3          StringBuffer sb = new StringBuffer("hello");
4          //StringBuilder sb = new StringBuilder("hello");
5          System.out.println(sb.capacity());
6          System.out.println(sb.charAt(0));
7          sb.append('w');
8          sb.append("orld");
9          System.out.println(sb);
10         sb.insert( offset: 5,  c: ' ');
11         System.out.println(sb);
12         sb.delete(5, sb.length());
13         System.out.println(sb);
14         sb.reverse();
15         System.out.println(sb);
16     }
17 }
```

StringTokenizer

StringTokenizer

- Break a string into pieces (tokens) so that contained information can be retrieved and processed
- Specify a set of characters as delimiters when constructing a StringTokenizer object
- **StringTokenizer** class is available since **JDK 1.0** and the **String.split()** is available since **JDK 1.4**
- **String.split()** does produce empty tokens, but **StringTokenizer** doesn't
- *StringTokenizer is a legacy class, retained for compatibility reasons, the use is discouraged!*

StringTokenizer

- Constructors
  - ***StringTokenizer(String str, String delim)***
  - ***StringTokenizer(String str)***
- Methods
  - ***hasMoreToken()*** - Returns true if there is a token left in the string
  - ***nextToken()*** - Returns the next token in the string
  - ***nextToken(String delim)*** - Returns the next token in the string after reseting the delimiter to delim
  - ***countToken( )*** - Returns the number of tokens remaining in the string tokenizer

StringTokenizer

```
3   import java.util.StringTokenizer;
4
5   public class StringTokenizerTest {
6       public static void main(String[] args) {
7           String s = new String("what's your news");
8           StringTokenizer tokens = new StringTokenizer(s);
9           System.out.println(tokens.countTokens());
10          while (tokens.hasMoreTokens()) {
11              System.out.println(tokens.nextToken());
12          }
13
14          String t = new String("what's,your,news");
15          tokens = new StringTokenizer(t, ",");
16          System.out.println(tokens.countTokens());
17          while (tokens.hasMoreTokens()) {
18              System.out.println(tokens.nextToken());
19          }
20      }
21  }
```