Java String Methods

Cody Henrichsen

2019

String Methods

java.lang.String (AP subset)
+ String() : Constructor + String(String) : Constructor + length() : int + substring(int) : String + substring(int, int) : String + indexOf(String) : int + equals(String) : boolean + compareTo(String) : int

java.lang.String (Cool and helpful subset)
+ contains() : boolean + lastIndexOf(String) : int + indexOf(String, int) : int + toLowerCase() : String + toUpperCase() : String + trim() : String + split(String) : String [ ] + concat(String) : String + getBytes() : byte [ ]

AP Subset

java.lang.String (AP subset)
+ String() : Constructor + String(String) : Constructor + length() : int + substring(int) : String + substring(int, int) : String + indexOf(String) : int + equals(String) : boolean + compareTo(String) : int

Constructors

- How to make a String
  - Parameter-less Constructor
    - Creates an empty String AKA ""
    - `String myString = new String();`
  - String parameter Constructor
    - Creates a String instance with the contents of parameter
    - `String myOtherString = new String("Words");`
  - Literal (not really a constructor)
    - Creates a String from the supplied text;
    - NO new
    - `String literal = "This is a literal";`

Bigness

- Since a String is immutable the "bigness" never changes and continues with the Java naming convention of length for a fixed size item
  - `String myString = "wowzers";`
  - `int length = myString.length(); // 7`

Parts of a String

- When you only want part of a `String` then substring is the way to go. Remember that in Java, the index always starts at 0. Of course, all values need to be in the range of 0…`length()`
  - `substring(int)`
    - This method takes from the supplied index to the end of the String.
    - This is great when you only want the last part of a String, say maybe a file extension or the conjugation of a regular verb
    - `myString.substring(2)` is the same as `myString.substring(2, myString.length())`
  - `substring(int, int)`
    - This is the more refined method of getting part of a `String`. The first parameter is the inclusive starting index and the second parameter is the exclusive ending index.
    - This is THE way to iterate over each letter in a String
      - `String currentLetter = myString.substring(index, index + 1);`

Basic Location : indexOf(String)

- When you need to know the location of the supplied `String` in the calling `String`
  - Returns a 0-based index for the location if found within the calling `String` for the FIRST occurrence
  - If the parameter `String` is NOT present within the calling `String` it will return -1
    - If the parameter is too big to fit in the calling `String`

Comparing Strings: compareTo(String)

- One of the objectives in almost every introductory computer science course involves the sorting of `String` data. Java provides for this with the `compareTo(String)` method. Depending on what the lexicographic (Dictionary order) relationship between the calling `String` and the parameter is starting at the **FIRST** substring, you will receive one of three values.
  - An integer less than 0 if the calling `String` would be found before the parameter
    - `"a".compareTo("b");`
    - `"car".compareTo("cat");`
    - `"Some text".compareTo("other text"); //Based on the first letter`
    - `"A".compareTo("a"); // Case matters!!!`
  - The integer 0 if the calling `String` is the same as the parameter `String`
    - `"a".compareTo("a");`
  - An integer greater than 0 if the calling `String` would be found after the parameter
    - `"b".compareTo("a");`
    - `"cat".compareTo("car");`

Comparing Strings: equals(String)

- This is probably one of the most helpful methods of the AP subset on String. It allows you to compare String values as an object. This is often used for if blocks both in the test as well as example programs.
- One of the ways to remember to use .equals(String) instead of == for a String is that .equalsIgnoreCase(String) also exists since == is **NOT** the best way to check for String equality in Java
  - `"Me".equals("me"); //false`
  - `"Me".equals("Me"); //true`
  - `"Me".equalsIgnoreCase("me"); //true`
  - `"me" == "me"; //false*`
    - Gee, thanks StringPool

Cool Subset

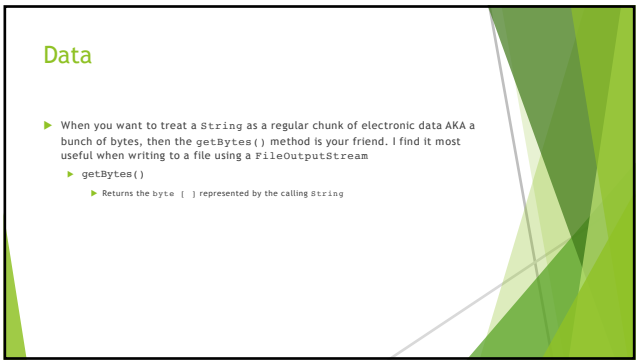
java.lang.String (Cool and helpful subset)
+ contains() : boolean + lastIndexOf(String) : int + indexOf(String, int) : int + toLowerCase() : String + toUpperCase() : String + trim() : String + split(String) : String [ ] + concat(String) : String + getBytes() : byte [ ]

Take a look inside

- When working with data it is often necessary to look inside the String or Strings that are in question. These methods provide more functionality than just the first occurrence of a substring than the `indexOf(String)` method supplies
  - `contains(String)`
    - Is the parameter located within the calling `String`. This one chains well with `.toLowerCase()`
  - `lastIndexOf(String)`
    - What is the last index of the supplied String. This one is especially useful for locating the extension of a filename because of the period may be in a username
  - `indexOf(String, int)`
    - The overloaded `indexOf` is Fantastic! This version is great for looking ahead in a String for repeat occurrences

Change the String

- String values are **immutable**, so the variable only changes if you use an assignment operator, but you can use these methods to change a String provided you assign the result to a variable or simply use the temporary result.
  - `toLowerCase()` AND `toUpperCase()`
    - These two are pretty self explanatory. Just remember that `String` values including emoji: {😺, 🖤, ☠️}, and non-Roman characters {わたし} do not have a "case" and will not be affected by these
  - `trim()`
    - Did you ever want to get rid of all the extra spaces caused by resting your head/cat on the space bar? Then trim() is what you need to call
  - `split(String)`
    - OMG this is possibly my favorite `String` method! When you need to grab all the words in a `String` and hold onto each one at the same time. Great for: database, text analysis, inflating from files
  - `concat(String)`
    - You want to build a `String` from other `Strings` and don't want to use an overloaded + operator? Then `concat(String)` is your friend.


Data
When you want to treat a String as a regular chunk of electronic data AKA a bunch of bytes, then the getBytes() method is your friend. I find it most useful when writing to a file using a FileOutputStream
getBytes()
Returns the byte [ ] represented by the calling String