Set 13: Advanced String Operations

Skill 13.01: Apply advanced String methods
Skill 13.02: Parse Strings with a Scanner

Skill 13.01: Apply advanced String methods

Skill 13.01 Concepts

The String methods and techniques we have encountered in previous lessons include *concatenation*, *length()*, *substring()*, *toLowerCase()*, *toUpperCase()*, *escape sequences ("\n", "\t", etc)*, *equals()*, and *equalsIgnoreCase()*

We will now look at some of the signatures (and examples) of some of the more advanced String methods.
Recall that a signature is as follows:

```
returnType methodName (type parameter1, type parameter2, etc);
```

- returnType - The type of variable returned when the method is called (int, double, Strings, etc.)
- methodName - The name of the method
- type - The parameter required of the method (int, double, String, etc)
- parameter1 - The value of the parameter

Below, we will look at several new methods associated with the String class

For all the examples below we will assume that "s" is a String as follows,

Recall that the indices of the individual characters of this String are as follows,

character	T	a	k	e		a		H	i	k	e	!
index	0	1	2	3	4	5	6	7	8	9	10	11

int compareTo(Object myObj)

The *compareTo* method accepts any Object, here we will specify a String object. The general syntax for usage of compareTo is shown below,

Code	**Output**	**Explanation**
`String s = "Take a Hike";` `int j = s.compareTo("Idaho");` `System.out.println(j);`	11	The ascii value of *T* is 11 greater than the ascii value of *I*

In the example above, *11* is printed because the first letter in "Take a Hike" is 11 places *after* the first letter in "Idaho" in the alphabet.

Now, let's consider the reverse,

Code	Output	Explanation
`String s = “Take a Hike”;` `int j = ("Idaho").compareTo(s);` `System.out.println(j);`	-11	The ascii value of *I* is 11 less than the ascii value of *T*

In the above example, because the first letter in "Idaho" is 11 digits *before* the first letter in "Take a Hike", -11 is printed.

Now, let's consider what happens when we mix lower and upper case letters in our comparison. For example,

Code	Output	Explanation
`String s = “Take a Hike”;` `int j = ("idaho").compareTo(s);` `System.out.println(j);`	21	The ascii value of *i* is 21 greater than the ascii value of *T*.

The result above can be explained in terms of the *char* values assocated with first letters in "idaho" and "Take a Hike". The first letter, *i*, has a value of 105 and, *T*, has a value of 84. 105 - 84 = 21.

int indexOf()

This method comes in 6 flavors (each is described below). For each case, all return negative one (-1) if the search is unsuccessful.

Signature	Description	Code Example	Output
`public int indexOf(String str)`	Search from left to right for the first occurrence of the *String str*.	`s = "The Walking Dead";` `int j = s.indexOf("Walking");` `System.out.println(j);`	4
`public int indexOf(String s, int from)`	Starting at index, from, search from left to right for the first occurrence of the *String s*	`s = "The Walking Dead";` `int j = s.indexOf("Walking", 7);` `System.out.println(j);` `int j = s.indexOf("a", 7);` `System.out.println(j);`	-1 14
`public int indexOf(char ch)`	Search from left to right for the first occurrence of the *char ch*	`s = "The Walking Dead";` `int j = s.indexOf('W');` `System.out.println(j);`	4
`public int indexOf(int ascii)`	This method is very similar to the one above, except instead of a char we give the ASCII code of the char desired	`s = "The Walking Dead";` `int j = s.indexOf(68); //ASCII code for 'D'` `System.out.println(j);`	12
`public int indexOf(char ch, int from)`	Starting at index from, search from left to right for the first occurrence of the *char ch*	`s = "The Walking Dead";` `int j = s.indexOf('e', 4)` `System.out.println(j);`	13
`public int indexOf(int ascii, int from)`	This method is very similar to the one above, except instead of a character we give the ASCII code of the character desired.	`s = "The Walking Dead";` `int j = s.indexOf(101, 4); //ASCII code for 'e'` `System.out.println(j);`	13

char charAt(int index)

This method returns the character at the specified index.

Code	Output	Explanation
`String s = "Take a Hike";` `char myChar = s.charAt(5);` `System.out.println(myCar);`	a	Prints the char at index 5

String trim()

This method removes whitespace from both ends of the String while leaving interior whitespace intact. (Whitespace consists of new line (\n), tab (\t), and spaces)

Code	Output	Explanation
`String s = "Take a Hike";` `s = "\t\t" + s + "\n";` `System.out.println("X" + s.trim() + "X");`	XTake a HikeX	Removes the space before and after the string which was added in the second line.

boolean contains(String ss)

Code	Output	Explanation
`String s = "Take a Hike";` `System.out.println(s.contains("ike"));`	true	Prints true because the String *s* contains "ike"

Skill 13.02: Parse Strings with a Scanner

Skill 13.02 Concepts

Pass a *String* to a *Scanner*

In a previous lesson we learned how to use the *Scanner* class to input text from the keyboard. Here, we illustrate further uses of *Scanner* in parsing (processing) a *String*. Instead of passing *System.in* to the *Scanner* as we did for the keyboard input, we pass a *String* as follows,

```
Scanner sc = new Scanner("Please, no more Exams!");
```

Once a *String* is passed to the *Scanner*, it is possible to analyze it. The process of analyzing a *String* or text is called *parsing*.

Delimiters

A delimiter is a series of characters that separate the text presented to a *Scanner* object into separate chunks called *tokens*. The default delimiter is whitespace. The tokens that make up the *String* below could be accessed using the *next()* method.

Code	Output	Explanation
`Scanner sc = new Scanner("Please, no more Exams!");` `String firstWord = sc.next();` `System.out.println(firstWord);` `String secondWord = sc.next();` `System.out.println(secondWord);` `String thirdWord = sc.next();` `System.out.println(thirdWord);` `String lastWord = sc.next();` `System.out.println(lastWord);`	`Please,` `no` `more` `exams!`	The *next()* method returns the letters up to the next space. The space is referred to as the "delimiter" that separates the String.

The *useDelimiter* method allows us to create a custom delimiter.

Code	Output	Explanation
`Scanner sc = new Scanner("Please, no more Exams!");` `sc.useDelimiter(",\s");` `String string1 = sc.next();` `System.out.println(string1);` `String string2 = sc.next();` `System.out.println(string2);`	`Please` `No more Exams!`	The useDelimiter method allows you to separate the String using any delimiter you want. Here we used the comma followed by a space. Everything up to the comma + space is printed, then everything after the comma+space is printed.

Parse a String using a loop

The portion of the string that is returned by the *hasNext()* method is referred to as a token.

In the above examples we were able to get each token in a String using different delimiters. But, to get each part required that we copy and paste the *next()* method over and over again.

What if we wanted to read all the words in any String, regardless of its length? Below is an example of how we can use the *hasNext()* method to evaluate whether or not another token exists. The hasNext() is useful for retrieving all the tokens from a string regardless of its length.

Below we repeat the same examples from above using a loop in combination with *hasNext()* to retrieve all the tokens in a String.

Code	Output	Explanation
`Scanner sc = new Scanner("Please, no more Exams!");` `while(sc.hasNext()){` `  String temp = sc.next();` `  System.out.println(temp);` `}`	Please, no more Exams!	*hasNext()* checks if there is another token in the string. If there is, the token is assigned to temp and printed.

Code	Output	Explanation
`Scanner sc = new Scanner("Please, no more Exams!");` `sc.useDelimiter(",\s");` `while(sc.hasNext()){` `    String temp = sc.next();` `    System.out.println(temp);` `}`	Please, No more exams	Here we set the delimiter to the comma followed by the space. The tokens printed are everything up to the comma+space and everything after the comma+space

Skill 13.02 Exercise 1