Lesson 28: Advanced *String* Methods

Following is a list of some of the *String* methods and techniques we encountered in Lesson 3:
Concatenation, length( ), substring( ), toLowerCase( ), toUpperCase( ), and escape sequences.

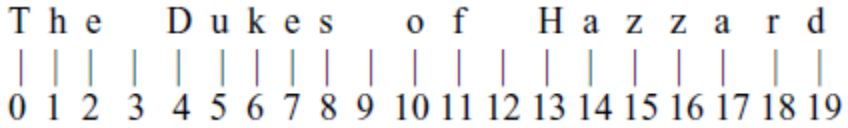
In Lesson 9 we studied how to compare *String*s:
equals( ) and equalsIgnoreCase( )

We will now look at some of the **signatures** (and examples) of some of the more advanced *String* methods. Recall from Lesson 15 that the layout of a signature is as follows:
public returnType methodName(type parameter1, type parameter2, …)

The variable *returnType* and *type* could be *double, int, boolean, String, char,* etc.

For the examples below assume that *s* is a *String* as follows:
s = "The Dukes of Hazzard"

For convenience, the indices of the individual characters of this *String* are given below:



public int compareTo(Object myObj)

Notice this method accepts any Object. Here, we will specifically use a ***String*** object.
The general syntax for usage of *compareTo* is: *int j = s.compareTo("coat room");*

This method has three rules:

a. If *s* alphabetically precedes "coat room" then it returns a negative *int*.
b. If *s* alphabetically comes after "coat room" it returns a positive *int*.
c. If *s* is alphabetically equal to "coat room" then it returns zero.

System.out.println(s.compareTo("coat room")); //**prints a negative number**

The reason we get a negative number in the example above is because 'T' alphabetically precedes 'c'. Refer back to Lesson 13 and you will see that the ASCII code for capital 'T' is 84 and the ASCII code for little 'c' is 99. The number 84 comes before 99 so we say that "The Dukes of Hazzard" comes before (or alphabetically precedes) "coat room".

There is another version of *compareTo, compareToIgnoreCase,* that is not case sensitive.

indexOf( )

This method comes in 6 flavors: (All return –1 if the search is unsuccessful.)

a. public int indexOf(String str)

Search from left to right for the first occurrence of the *String str*.

```
int j = s.indexOf("Hazzard");
System.out.println(j);          //13
```

b. public int indexOf(String s, int from)

Starting at index *from*, search from left to right for the first occurrence of the *String s.*

```
int j = s.indexOf("Hazzard", 15);
System.out.println(j); //-1…it couldn't find it when starting at 15
int j = s.indexOf("e", 4);
System.out.println(j); //7. First "e" is at 2, but we started searching at 4
```

c. public int indexOf(char ch)

Search from left to right for the first occurrence of the *char ch.*

```
int j = s.indexOf('D');
System.out.println(j);          //4
```

d. public int indexOf(int ascii)

This method is very similar to c. above, except, instead of a character we give the ASCII code of the character desired.

```
int j = s.indexOf(68);          // ASCII code for 'D' is 68
System.out.println(j);          //4
```

e. public int indexOf(char ch, int from)

Starting at index *from*, search from left to right for the first occurrence of the *char ch.*

```
int j = s.indexOf('e', 4);
System.out.println(j);          //7
```

f. public int indexOf(int ascii, int from)

This method is very similar to e. above, except, instead of a character we give the ASCII code of the character desired.

```
int j = s.indexOf(101, 4);      // ASCII code for 'e' is 101
System.out.println(j);          //7
```

lastIndexOf( )

This method also comes in 6 flavors: (All return –1 if the search is unsuccessful.)

These are exactly the same as the *indexOf( )* method, except here, we begin searching from the right side of the *String* instead of the left as with *indexOf( )*. Only two examples will be given since they are so similar to the previous examples.

a. public int lastIndexOf(String str)

```
int j = s.lastIndexOf("Haz");
System.out.println(j); //13
```

b. public int lastIndexOf(String s, int from)

```
int j = s.lastIndexOf("Haz", 11);
System.out.println(j); // -1…can't find since we start at 11.
```

c. public int lastIndexOf(char ch)
d. public int lastIndexOf(int ascii)
e. public int lastIndexOf(char ch, int from)
f. public int lastIndexOf(int ascii, int from)

public char charAt(int indx)

This method returns the character at the specified index.

```
char myChar = s.charAt(6);
System.out.println(myChar);          //k
```

public String replace(char old, char new)

This method replaces **all** occurrences of the character *old* with the character *new*.

```
String myString = s.replace('z', 'L');
System.out.println(myString); // The Dukes of HaLLard
```

public String replace(String old, String new)

This method replaces **all** occurrences of *String old* with *String new*.

```
String myString = s.replace("Dukes", "Nerds");
System.out.println(myString); // The Nerds of Hazzard
```

public String trim( )

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

```
String s = "\t Ding Dong \t \n " ;
System.out.println("X" + s.trim( ) + "X"); // XDing DongX
```

public boolean contains(String ss)

This method returns *true* when this *String* contains the *String ss*; otherwise, *false*.

```
boolean b = "Sticks and Stones".contains("tic"); //returns true
```

public boolean startsWith(String ss)

This methods returns *true* when this *String* contains *ss* as its leading substring.

```
boolean b = "Have a good day.".startsWith("Hav"); //returns true
```