String class methods

Strings of characters, such as "Enter the amount:", are treated slightly differently than values of the primitive types. Java has no primitive type for strings. However, Java supplies a class called String that can be used to create and process strings of characters. The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

The following declares greeting to be the name for a String variable:

String greeting;

String greeting = "Hello!";

Once a String variable, such as greeting, has been given a value, you can display it on the screen as follows:

System.out.println(greeting);

A string can have any number of characters. For example, "Hello" has five characters. A string can even have zero characters. Such a string is called the **empty string** and is written as a pair of adjacent double quotes, like so:"".

Concatenation of Strings

You can use + to join, or concatenate, strings together.

Notice that no spaces are added when you concatenate two strings by means of the + operator. If you wanted sentence set to "Hello my friend", you could change the assignment statement to

sentence = greeting + " my friend";

Notice that no spaces are added when you concatenate two strings by means of the + operator.

String Methods

Length() method
The method length returns the number of characters in a String object.

So "Hello".length( ) returns the integer 5.
String greeting = "Hello";
int n = greeting.length( );

Positions in a string begin with 0, not with 1. In the string "Hi Mom", 'H' is in position 0, 'i' is in position 1, the blank character is in position 2, and so forth. A position is usually referred to as an **index** in computer parlance. So it would be more normal to say that 'H' is at index 0, 'i' is at index 1, and so on.

Indexof()
A **substring** is simply a portion of a string. For example, the string defined by

String phrase = "Java is fun.";

has a substring "fun" that begins at index 8. The method indexOf returns the index of a substring given as its argument. The invocation phrase.indexOf("fun") will return 8 because the 'f' in "fun" is at index 8. If the substring occurs more than once in a string, indexOf returns the index of the first occurrence of its substring argument.

UpperCase() and toLowerCase() method
The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.
Example:
```
String s="Sachin";
System.out.println(s.toUpperCase());//SACHIN
System.out.println(s.toLowerCase());//sachin
System.out.println(s);//Sachin(no change in original)
```
Output:
```
SACHIN
sachin
Sachin
```

trim() method
The string trim() method eliminates white spaces before and after string.
```
String s="  Sachin  ";
```
Example:
```
System.out.println(s);//  Sachin
System.out.println(s.trim());//Sachin
```
Output:

```
Sachin
Sachin
```

startsWith() and endsWith() method
It returns true if the string starts or ends with specified string.
Example:

```
String s="Sachin";
System.out.println(s.startsWith("Sa"));//true
System.out.println(s.endsWith("n"));//true
```

Output:

```
true
true
```

charAt() method
The string charAt() method returns a character at specified index.
Example:

```
String s="Sachin";
System.out.println(s.charAt(0));//S
System.out.println(s.charAt(3));//h
```

length() method
The string length() method returns length of the string.
Example:

```
String s="Sachin";
System.out.println(s.length());//6
```

valueOf() method
The string valueOf() method coverts given type such as int, long, float, double, boolean, char and char array into string.
Example:

```
int a=10;
String s=String.valueOf(a);
System.out.println(s+10);
```

Output:

```
1010
```

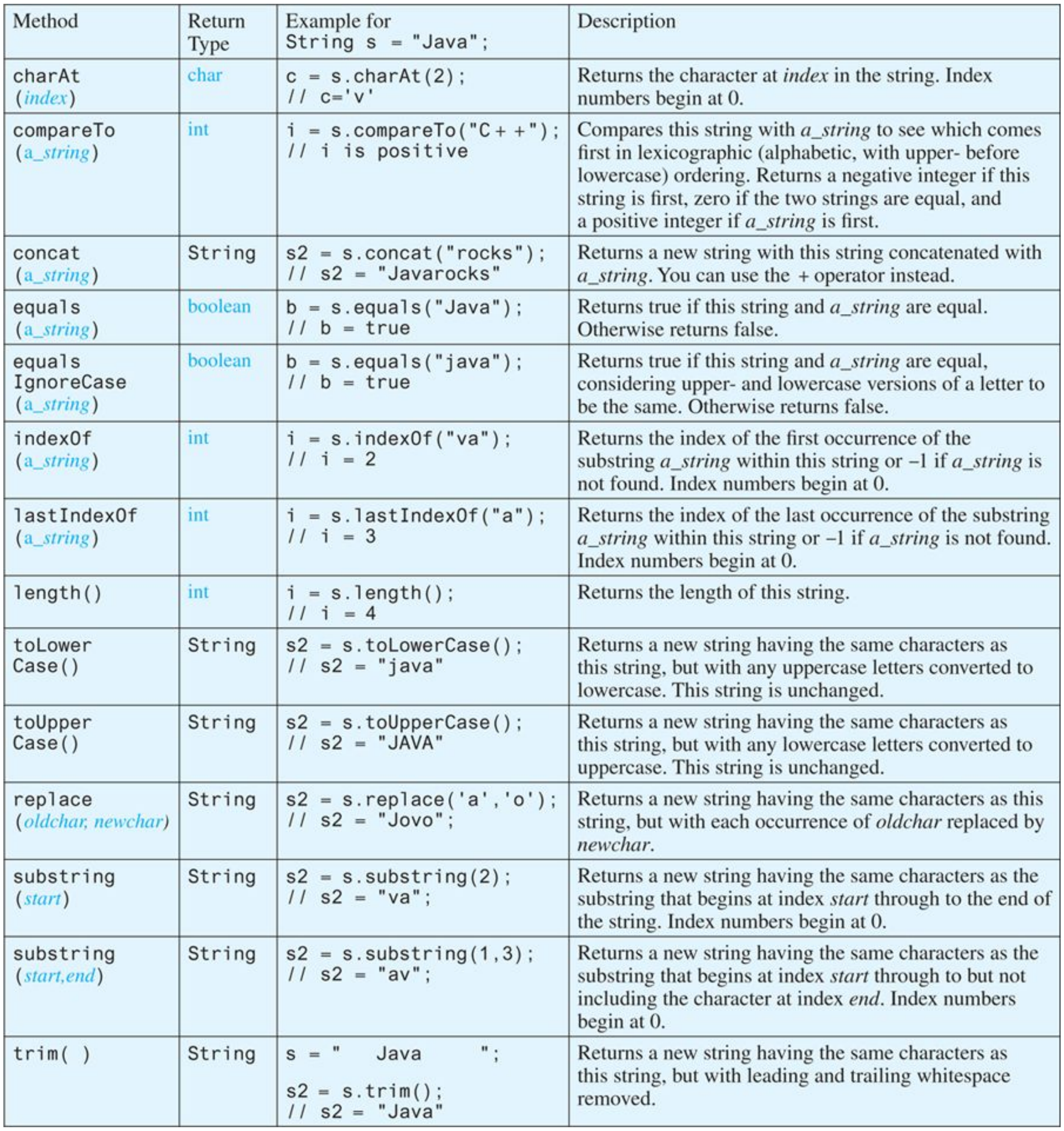
replace() method
The string replace() method replaces all occurrence of first sequence of character with second sequence of character.
Example:

```
String s1="Java is a programming language. Java is a platform. Java is an Island.";
String replaceString=s1.replace("Java","Kava");//replaces all occurrences of "Java" to
//"Kava"
System.out.println(replaceString);
```

Output

```
Kava is a programming language. Kava is a platform. Kava is an Island.
```

List of String Methods:



Method	Return Type	Example for `String s = "Java";`	Description
`charAt` (*index*)	char	`c = s.charAt(2);` `// c='v'`	Returns the character at *index* in the string. Index numbers begin at 0.
`compareTo` (*a_string*)	int	`i = s.compareTo("C++");` `// i is positive`	Compares this string with *a_string* to see which comes first in lexicographic (alphabetic, with upper- before lowercase) ordering. Returns a negative integer if this string is first, zero if the two strings are equal, and a positive integer if *a_string* is first.
`concat` (*a_string*)	`String`	`s2 = s.concat("rocks");` `// s2 = "Javarocks"`	Returns a new string with this string concatenated with *a_string*. You can use the + operator instead.
`equals` (*a_string*)	boolean	`b = s.equals("Java");` `// b = true`	Returns true if this string and *a_string* are equal. Otherwise returns false.
`equals IgnoreCase` (*a_string*)	boolean	`b = s.equals("java");` `// b = true`	Returns true if this string and *a_string* are equal, considering upper- and lowercase versions of a letter to be the same. Otherwise returns false.
`indexOf` (*a_string*)	int	`i = s.indexOf("va");` `// i = 2`	Returns the index of the first occurrence of the substring *a_string* within this string or –1 if *a_string* is not found. Index numbers begin at 0.
`lastIndexOf` (*a_string*)	int	`i = s.lastIndexOf("a");` `// i = 3`	Returns the index of the last occurrence of the substring *a_string* within this string or –1 if *a_string* is not found. Index numbers begin at 0.
`length()`	int	`i = s.length();` `// i = 4`	Returns the length of this string.
`toLower Case()`	`String`	`s2 = s.toLowerCase();` `// s2 = "java"`	Returns a new string having the same characters as this string, but with any uppercase letters converted to lowercase. This string is unchanged.
`toUpper Case()`	`String`	`s2 = s.toUpperCase();` `// s2 = "JAVA"`	Returns a new string having the same characters as this string, but with any lowercase letters converted to uppercase. This string is unchanged.
`replace` (*oldchar, newchar*)	`String`	`s2 = s.replace('a','o');` `// s2 = "Jovo";`	Returns a new string having the same characters as this string, but with each occurrence of *oldchar* replaced by *newchar*.
`substring` (*start*)	`String`	`s2 = s.substring(2);` `// s2 = "va";`	Returns a new string having the same characters as the substring that begins at index *start* through to the end of the string. Index numbers begin at 0.
`substring` (*start,end*)	`String`	`s2 = s.substring(1,3);` `// s2 = "av";`	Returns a new string having the same characters as the substring that begins at index *start* through to but not including the character at index *end*. Index numbers begin at 0.
`trim( )`	`String`	`s = "   Java    ";` `s2 = s.trim();` `// s2 = "Java"`	Returns a new string having the same characters as this string, but with leading and trailing whitespace removed.

Escape Characters

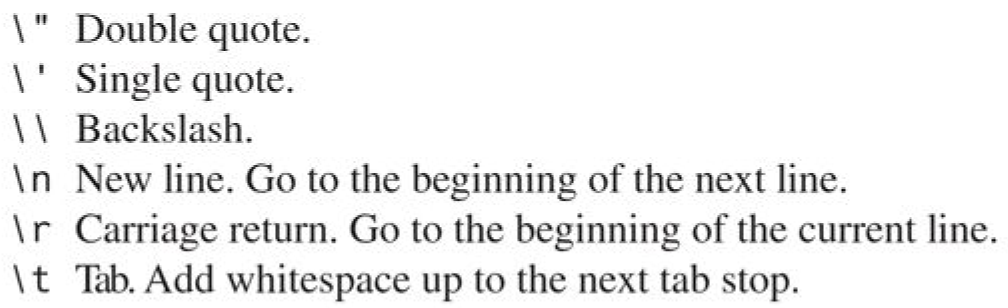
Suppose you want to display a string that contains quotation marks. For example, suppose you want to display the following on the screen:

The word "Java" names a language, not just a drink!

You tell the compiler that you mean to include the quote in the string by placing a backslash (\) before the troublesome character, like so:

```
System.out.println("The word \"Java\" names a language, " +  "not just a drink!");
```

Escape Characters:



`\"`	Double quote.
`\'`	Single quote.
`\\`	Backslash.
`\n`	New line. Go to the beginning of the next line.
`\r`	Carriage return. Go to the beginning of the current line.
`\t`	Tab. Add whitespace up to the next tab stop.

The escape sequence \n indicates that the string starts a new line at the \n. For example, the statement

```
System.out.println("The motto is\nGo for it!");
```

will write the following two lines to the screen:

```
The motto is
Go for it!
```

Comparing Strings

Equals() method: Use the equals method, not ==, to see whether two strings are equal. For example, the boolean expression s1.equals(s2) returns true if the strings s1 and s2 have equal values, and returns false otherwise. Notice that the two expressions
s1.equals(s2)
s2.equals(s1)

are equivalent.

equalsIgnoreCase() method: This method behaves like equals, except that equalsIgnoreCase considers the uppercase and lowercase versions of the same letter to be the same. For example, "Hello" and "hello" are not equal because their first characters, ‘H’ and ‘h’, are different characters. But the method equalsIgnoreCase would consider them equal. For example, the following will display Equal:

```
if ("Hello".equalsIgnoreCase("hello"))
    System.out.println("Equal");
```

compareto() method

The method compareTo tests two strings to determine their **lexicographic order.** Lexicographic order is similar to alphabetic order and is sometimes, but not always, the same as alphabetic order. In lexicographic ordering, the letters and other characters are ordered according to their Unicode sequence, which is shown in the appendix of this book.

If s1 and s2 are two variables of type String that have been given String values, the method call

```
s1.compareTo(s2)
```

compares the lexicographic ordering of the two strings and returns

- A negative number if s1 comes before s2
- Zero if the two strings are equal
- A positive number if s1 comes after s2

Thus, the boolean expression

```
s1.compareTo(s2) < 0
```

is true if s1 comes before s2 in lexicographic order and false otherwise.

Example:

```
class Teststringcomparison{
        public static void main(String args[]){
        String s1="Sachin";
        String s2="Sachin";
        String s3="Ratan";
        System.out.println(s1.compareTo(s2));//0
        System.out.println(s1.compareTo(s3));//1(because s1>s3)
        System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
         }
}
```