class String

In the preceding sections, we discussed primitive types to deal with data consisting of numbers and characters. What about data values such as a person's name? A person's name contains more than one character. Such values are called strings. More formally, a **string** is a sequence of zero or more characters. Strings in Java are enclosed in double quotation marks (not in single quotation marks, as are the `char` data types).

Most often, we process strings as a single unit. To process strings effectively, Java provides the `class String`. The `class String` contains various operations to manipulate a string. You will see this class used throughout the book. Chapter 3 discusses various operations provided by the `class String`. Moreover, technically speaking, the `class String` is not a primitive type.

A string that contains no characters is called a **null** or **empty** string. The following are examples of strings. Note that `""` is the empty string.

```
"William Jacob"
"Mickey"
""
```

A string, such as `"hello"`, is sometimes called a **character string** or **string literal** or **string constant**. However, if no confusion arises, we refer to characters between double quotation marks simply as strings.

Every character in a string has a specific position in the string. The position of the first character is 0, the position of the second character is 1, and so on. The **length** of a string is the number of characters in it.

EXAMPLE 2-9

String	"Sunny Day"								
Character in the string	'S'	'u'	'n'	'n'	'y'	' '	'D'	'a'	'y'
Position of the character in the string	0	1	2	3	4	5	6	7	8

The length of the string `"Sunny Day"` is 9.

When determining the length of a string, you must also count any spaces contained in the string. For example, the length of the string `"It is a beautiful day."` is 22.

Strings and the Operator +

One of the most common operations performed on strings is the concatenation operation, which allows a string to be appended at the end of another string. The operator + can be used to concatenate (or join) two strings as well as a string and a numeric value or a character.

Next we illustrate how the operator + works with strings. Consider the following expression:

```
"Sunny" + " Day"
```

This expression evaluates to

```
"Sunny Day"
```

Now consider the following expression:

```
"Amount Due = $" + 576.35
```

When the operator + evaluates, the numeric value `576.35` is converted to the string `"576.35"`, which is then concatenated with the string:

```
"Amount Due = $"
```

Therefore, the expression `"Amount Due = $" + 576.35` evaluates to

```
"Amount Due = $576.35"
```

Example 2-10 further explains how the operator + works with the `String` data type.

EXAMPLE 2-10

Consider the following expression:

```
"The sum = " + 12 + 26
```

This expression evaluates to

```
"The sum = 1226"
```

This is not what you might have expected. Rather than adding `12` and `26`, the values `12` and `26` are concatenated. This is because the associativity of the operator + is from left to right, so the operator + is evaluated from left to right. The expression

```
"The sum = " + 12 + 26
```

is evaluated as follows:

```
"The sum = " + 12 + 26  = ("The sum = " + 12) + 26
                        = "The sum = 12" + 26
                        = "The sum = 12" + 26
                        = "The sum = 1226"
```

Now consider the following expression:

```
"The sum = " + (12 + 26)
```

This expression evaluates as follows:

```
    "The sum = " + (12 + 16)
=   "The sum = " + 38
=   "The sum = 38"
```

Next, consider the expression:

```
12 + 26 + " is the sum"
```

This expression evaluates as follows:

```
    12 + 26 + " is the sum"
=   (12 + 26) + " is the sum"
=   38 + " is the sum"
=   "38 is the sum"
```

Now consider the expression:

```
"The sum of " + 12 + " and " + 26 + " = " + (12 + 26)
```

Notice the parentheses around `12 + 26`. This expression evaluates as follows:

```
    "The sum of " + 12 + " and " + 26 + " = " + (12 + 26)
=   "The sum of 12" + " and " + 26 + " = " + (12 + 26)
=   "The sum of 12 and " + 26 + " = " + (12 + 26)
=   "The sum of 12 and 26" + " = " + (12 + 26)
=   "The sum of 12 and 26 = " + (12 + 26)
=   "The sum of 12 and 26 = " + 38
=   "The sum of 12 and 26 = 38"
```

The following Java program shows the effect of the preceding statements:

```
// This program illustrates how the String concatenation works.

public class Example2_10
{
    public static void main(String[] args)
    {
        System.out.println("The sum = " + 12 + 26);

        System.out.println("The sum = " + (12 + 26));

        System.out.println(12 + 26 + " is the sum");

        System.out.println("The sum of " + 12 + " and " + 26
                           + " = " + (12 + 26));
    }
}
```

Sample Run:

```
The sum = 1226
The sum = 38
38 is the sum
The sum of 12 and 26 = 38
```

The `class String` contains many useful methods to manipulate strings. We will take a closer look at this class in Chapter 3 and illustrate how to manipulate strings. The complete description of this class can be found at the Web site *http://java.sun.com/javase/7/docs/api/*.

class String

This section explains how to use `String` methods to manipulate strings. First, we review some terminology that we typically use while working with strings and the `class String`.

Consider the following statements:

```
String name;                //Line 1
name = "Lisa Johnson";      //Line 2
```

The statement in Line 1 declares `name` to be a `String` variable. The statement in Line 2 creates the string `"Lisa Johnson"` and assigns it to `name`.

In the statement in Line 2, we usually say that the `String` object, or the string `"Lisa Johnson"`, is assigned to the `String` variable or the variable `name`. In reality, as explained before, a `String` *object* with the value `"Lisa Johnson"` is instantiated (if it has not already been created) and the address of the object is stored in `name`. Whenever we use the term "the string `name`", we are referring to the object containing the string `"Lisa Johnson"`. Similarly, when we use the terms (reference) variable `name` or `String` variable `name`, we simply mean `name`, whose value is an address.

The remainder of this section describes various features of the `class String`. In Chapter 2, you learned that two strings can be joined using the operator +. The `class String` provides various methods that allow us to process strings in various ways. For example, we can find the length of a string, extract part of a string, find the position of a particular string in another string, convert a number into a string, and convert a numeric string into a number.

Each method associated with the `class String` implements a specific operation and has a specific name. For example, the method for determining the length of a string is named `length`, and the method for extracting a string from within another string is named `substring`.

As explained in the earlier section, Using Predefined Classes and Methods in a Program, in general, to use a method you must know the name of the `class` containing the

method and the name of the `package` containing the `class`; you must import the `class`; and you must know the method name, its parameters, and what the method does. However, because the Java system automatically makes the `class` `String` available, you do not need to import this `class`. Therefore, in order to use a `String` method, you need to know its name, parameters, and what the method does.

Recall that a string (literal) is a sequence of 0 or more characters, and string literals are enclosed in double quotation marks. The **index** (position) of the first character is `0`, the index of the second character is `1`, and so on. The length of a string is the number of characters in it, not the largest index.

If `length` denotes the length of a string and `length` is not zero (that is, string is not null), then `length - 1` gives the index of the last character in the string.

The general expression to use a `String` method on a `String` variable is:

```
StringVariable.StringMethodName(parameters)
```

In this statement, the variable name and the method name are separated with the dot (`.`). For example, if `name` is a `String` variable, and `name = "Lisa Johnson"`, then the value of the expression

```
name.length()
```

is `12`.

Table 3-1 lists commonly used methods of the `class` `String`. Suppose that `sentence` is a `String`. Suppose that `sentence = "Programming with Java";`. Then each character in `sentence` and its position is as follows:

sentence = "Programming with Java";																			
P	r	o	g	r	a	m	m	i	n	g	' '	w	i	t	h	' '	J	a	v	a
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

TABLE 3-1 Some Commonly Used `String` Methods

```
char charAt(int index)
  //Returns the character at the position specified by index
  //Example: sentence.charAt(3) returns 'g'
```

```
int indexOf(char ch)
  //Returns the index of the first occurrence of the character
  //specified by ch; If the character specified by ch does not
  //appear in the string, it returns -1
  //Example: sentence.indexOf('J') returns 17
  //         sentence.indexOf('a') returns 5
```

TABLE 3-1 Some Commonly Used `String` Methods (continued)

```
int indexOf(char ch, int pos)
  //Returns the index of the first occurrence of the character
  //specified by ch; The parameter pos specifies where to
  //begin the search; If the character specified by ch does not
  //appear in the string, it returns -1
  //Example: sentence.indexOf('a', 10) returns 18
```

```
int indexOf(String str)
  //Returns the index of the first occurrence of the string
  //specified by str; If the string specified by str does not
  //appear in the string, it returns -1
  //Example: sentence.indexOf("with") returns 12
  //         sentence.indexOf("ing") returns 8
```

```
int indexOf(String str, int pos)
  //Returns the index of the first occurrence of the String
  //specified by str; The parameter pos specifies where to begin
  //the search; If the string specified by str does not appear
  //in the string, it returns -1
  //Example: sentence.indexOf("a", 10) returns 18
  //         sentence.indexOf("Pr", 10) returns -1
```

```
String concat(String str)
  //Returns the string that is this string concatenated with str
  //Example: The expression
  //         sentence.concat(" is fun.")
  //         returns the string "Programming with Java is fun."
```

```
int compareTo(String str)
  //Compares two strings character by character
  //Returns a negative value if this string is less than str
  //Returns 0 if this string is same as str
  //Returns a positive value if this string is greater than str
```

```
boolean equals(String str)
  //Returns true if this string is same as str
```

```
int length()
  //Returns the length of the string
  //Example: sentence.length() returns 21, the number of characters in
  //         "Programming with Java"
```

```
String replace(char charToBeReplaced, char charReplacedWith)
  //Returns the string in which every occurrence of
  //charToBeReplaced is replaced with charReplacedWith
  //Example: sentence.replace('a', '*') returns the string
  //         "Progr*mming with J*v*"
  //         Each occurrence of a is replaced with *
```

TABLE 3-1 Some Commonly Used `String` Methods (continued)

```
String substring(int beginIndex)
  //Returns the string which is a substring of this string
  //beginning at beginIndex until the end of the string.
  //Example: sentence.substring(12) returns the string
  //         "with Java"

String substring(int beginIndex, int endIndex)
  //Returns the string which is a substring of this string
  //beginning at beginIndex until endIndex - 1

String toLowerCase()
  //Returns the string that is the same as this string, except
  //that all uppercase letters of this string are replaced with
  //their equivalent lowercase letters
  //Example: sentence.toLowerCase() returns "programming with java"

String toUpperCase()
  //Returns the string that is the same as this string, except
  //that all lowercase letters of this string are replaced with
  //their equivalent uppercase letters
  //Example: sentence.toUpperCase() returns "PROGRAMMING WITH JAVA"

boolean startsWith(String str)
  //Returns true if the string begins with the string specified by str;
  //otherwise, this methods returns false.

boolean endsWith(String str)
  //Returns true if the string ends with the string specified by str
  //otherwise, this methods returns false.

boolean regionMatches(int ind, String str, int strIndex, int len)
  //Returns true if the substring of str starting at strIndex and length
  //specified by len is same as the substring of this String
  //object starting at ind and having the same length

boolean regionMatches(boolean ignoreCase, int ind,
                      String str, int strIndex, int len)
  //Returns true if the substring of str starting at strIndex and length
  //specified by len is same as the substring of this String
  //object starting at ind and having the same length. If ignoreCase
  //is true, then during character comparison, case is ignored.
```

NOTE

Table 3-1 lists only some of the methods for string manipulation. Moreover, the table gives only the name of the method, the number of parameters, and the type of the method. The reader can find a list of `String` methods at the Web site *http://java.sun.com/javase/7/docs/api/*. The methods `equals` and `compareTo` are explained in Chapter 4 and the methods, `startsWith`, `endsWith`, and `regionMatches` are explained in Example 3-3.

EXAMPLE 3-2

Consider the following statements:

```
String sentence;
String str1;
String str2;
int index;

sentence = "Now is the time for the birthday party.";
```

The following statements further show how `String` methods work.

Statement	Effect / Explanation
`sentence.charAt(16)`	Returns: `'f'` In `sentence`, the character at position 16 is `'f'`.
`sentence.length()`	Returns: 38 The number of characters in sentence is 38.
`sentence.indexOf('t')`	Returns: 7 This is the index of the first `'t'` in `sentence`.
`sentence.indexOf("for")`	Returns: 16 In `sentence`, the starting index of the string `"for"`.
`sentence.substring(0, 6)`	Returns: `"Now is"` In `sentence`, the substring starting at index 0 until the index 5 (= 6 – 1) is `"Now is"`.
`sentence.substring(7, 12)`	Returns: `"the t"` In `sentence`, the substring starting at index 7 until the index 11 (= 12 – 1) is `"the t"`.
`sentence.substring(7, 22)`	Returns: `"the time for th"` In `sentence`, the substring starting at index 7 until the index 21 (= 22 – 1) is `"the time for th"`.
`sentence.substring(4, 10)`	Returns: `"is the"` In `sentence`, the substring starting at index 4 until the index 9 (= 10 – 1) is `"is the"`.
`str1 = sentence.substring(0, 8);`	`str1 = "Now is t"` In `sentence`, the substring starting at index 0 until the index 7 (= 8 – 1) is `"Now is t"`. So the value assigned to `str1` is `"Now is t"`.

Statement	Effect / Explanation
`str2 = sentence.substring(2, 12);`	`str2 = "w is the t"` In `sentence`, the substring starting at index 2 until the index 11 (= 12 – 1) is `"w is the t"`. So the value assigned to `str2` is `"w is the t"`.

`index = sentence.indexOf("birthday");`
`str1 = sentence.substring(index, index + 14);`
	`index = 24`
	`str1 = "birthday party"`
	The starting index of `"birthday"` in `sentence` is 24. So the value of index is 24. Now index is 24, so index + 14 is 38. The substring starting at the position 24 until the position 37 (= 38 – 1) is `"birthday party"`.

`sentence.replace('t', 'T')`	Returns: `"Now is The Time for The birThday parTy"`
`sentence.toUpperCase()`	Returns: `"NOW IS THE TIME FOR THE BIRTHDAY PARTY"`

The following program tests the preceding statements:

```
// This program illustrate how various String methods work.

public class VariousStringMethods
{
    public static void main(String[] args)
    {
        String sentence;
        String str1;
        String str2;
        String str3;
        int index;

        sentence = "Now is the time for the birthday party";

        System.out.println("sentence = \"" + sentence + "\"");
        System.out.println("The length of sentence = "
                           + sentence.length());

        System.out.println("The character at index 16 in "
                           + "sentence = " + sentence.charAt(16));

        System.out.println("The index of first t in sentence = "
                           + sentence.indexOf('t'));

        System.out.println("The index of for in sentence = "
                           + sentence.indexOf("for"));
```

```
        System.out.println("sentence.substring(0, 6) = \""
                           + sentence.substring(0, 6) + "\"");

        System.out.println("sentence.substring(7, 12) = \""
                           + sentence.substring(7, 12) + "\"");

        System.out.println("sentence.substring(7, 22) = \""
                           + sentence.substring(7, 22) + "\"");

        System.out.println("sentence.substring(4, 10) = \""
                           + sentence.substring(4, 10) + "\"");

        str1 = sentence.substring(0, 8);

        System.out.println("str1 = \"" + str1 + "\"");

        str2 = sentence.substring(2, 12);
        System.out.println("str2 = \"" + str2 + "\"");

        System.out.println("sentence in uppercase = \""
                          + sentence.toUpperCase() + "\"");

        index = sentence.indexOf("birthday");

        str1 = sentence.substring(index, index + 14);

        System.out.println("str1 = \"" + str1 + "\"");

        System.out.println("sentence.replace('t', 'T') = \""
                            + sentence.replace('t', 'T') + "\"");
    }
}
```

Sample Run:

```
sentence = "Now is the time for the birthday party"
The length of sentence = 38
The character at index 16 in sentence = f
The index of first t in sentence = 7
The index of for in sentence = 16
sentence.substring(0, 6) = "Now is"
sentence.substring(7, 12) = "the t"
sentence.substring(7, 22) = "the time for th"
sentence.substring(4, 10) = "is the"
str1 = "Now is t"
str2 = "w is the t"
sentence in uppercase = "NOW IS THE TIME FOR THE BIRTHDAY PARTY"
str1 = "birthday party"
sentence.replace('t', 'T') = "Now is The Time for The birThday parTy"
```

EXAMPLE 3-3

Consider the following statements:

```
String sentence;
String str1;
String str2;
String str3;
String str4;

sentence = "It is sunny and warm.";
str1 = "warm.";
str2 = "Programming with Java";
str3 = "sunny";
str4 = "Learning Java Programming is exciting";
```

The following statements show how `String` methods `startsWith`, `endsWith`, and `regionMatches` work.

Expression	Effect
`sentence.startsWith("It")`	Returns `true`
`sentence.startsWith(str1)`	Returns `false`
`sentence.endsWith("hot")`	Returns `false`
`sentence.endsWith(str1)`	Returns `true`
`sentence.regionMatches(6, str3, 0, 5)`	Returns `true`
`sentence.regionMatches(true, 6, "Sunny", 0, 5)`	Returns `true`
`str4.regionMatches(9, str2, 17, 4)`	Returns `true`

For the most part, the statements are straightforward. Let's look at the last three statements, which use the method `regionMatches`:

```
sentence.regionMatches(6, str3, 0, 5)
```

In this statement, we want to determine whether `str3` appears as a substring in the string `sentence` starting at position `6`. Notice that the last three arguments, `str3`, `0`, and `5`, specify that in `str3` the starting index is `0` and the length of the substring is `5`. The substring in `sentence` starting at position `6` and of length `5` matches `str3`. So this expression returns `true`.

The expression:

```
sentence.regionMatches(true, 6, "Sunny", 0, 5)
```

is similar to the previous expression, except that when the substrings are compared, the case is ignored, that is, uppercase and lowercase letters are considered the same. Next, let's look at the expression:

```
str4.regionMatches(9, str2, 17, 4)
```

In this expression, we want to determine whether the substring in `str2` starting at position `17` and of length `4` is the same as the substring in `str4` starting at position `9` and of length `4`. This expression returns `true` because these substrings are the same.

The program `Ch3_SomeStringMethods.java`, which shows the effect of the preceding statements, can be found in the Additional Student Files folder at *www.cengagebrain.com*.

To summarize the preceding discussion of the `class String`:

1. `String` variables are reference variables.
2. A string object is an instance of the `class String`.
3. The `class String` contains various methods to process strings.
4. A `String` variable invokes a `String` method using the dot operator, the method name, and the set of arguments (if any) required by the method.