6.5. THE STRING CLASS

A string is a sequence of characters. In many languages, strings are treated as an array of characters, but in Java a string is an object. The String class has 11 constructors and more than 40 methods for manipulating strings.

6.5.1 Constructing a String

You can create a string object from a string literal or from an array of characters. To create a string from a string literal, use a syntax like this one:

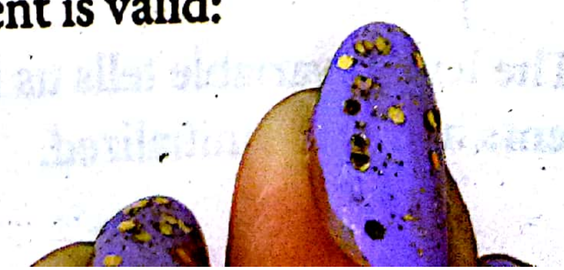
```
String newString = new String(stringLiteral);
```

The argument **stringLiteral** is a sequence of characters enclosed inside double quotes. For Example

```
String message = new String("Welcome to Java");
```

Java treats a string literal as a String object. So, the following statement is valid:

```
String message = "Welcome to Java";
```



You can also create a string from an array of characters. For example, the following statements create the string "Good Day":

```
char[] charArray = {'G','o','o','d', ' ', 'D','a','y'};
String message = new String(charArray);
```

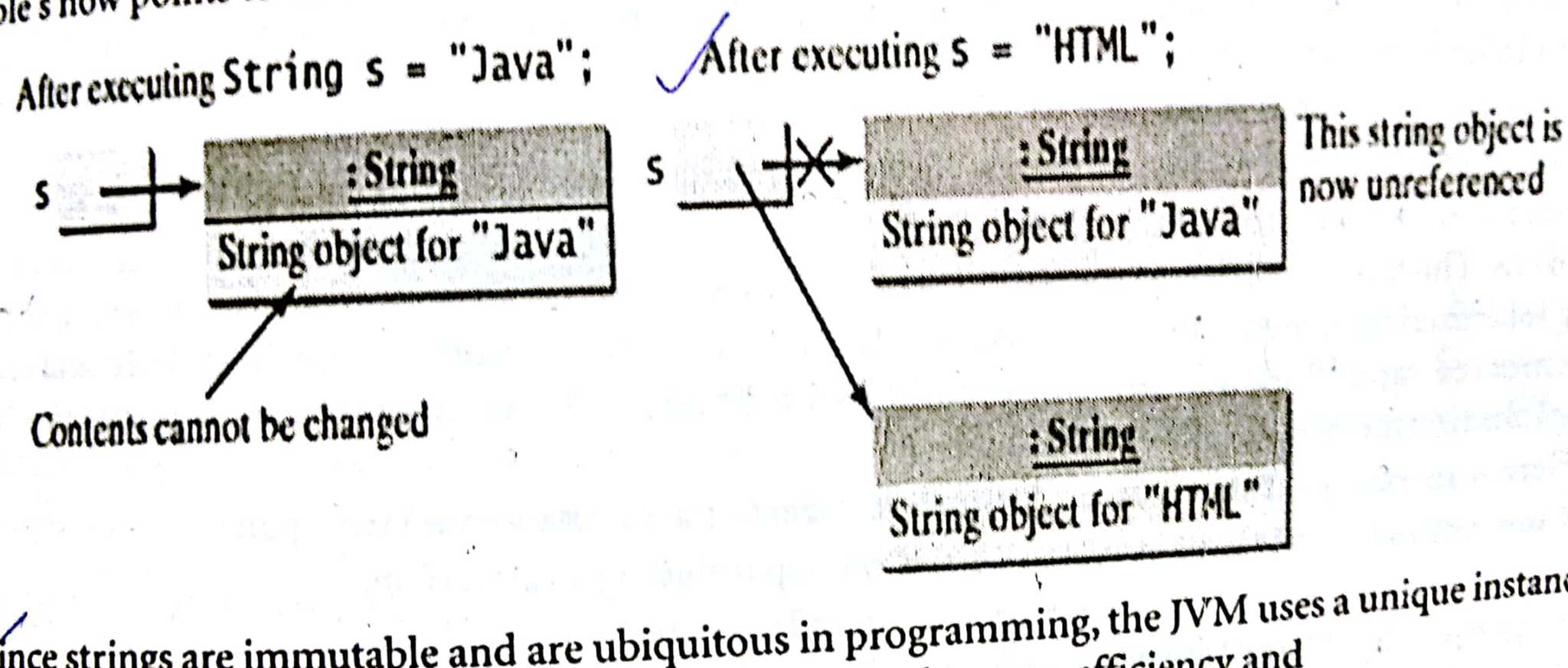
6.5.2. Immutable Strings and Interned Strings

A String object is immutable; its contents cannot be changed. Does the following code change the contents of the string?

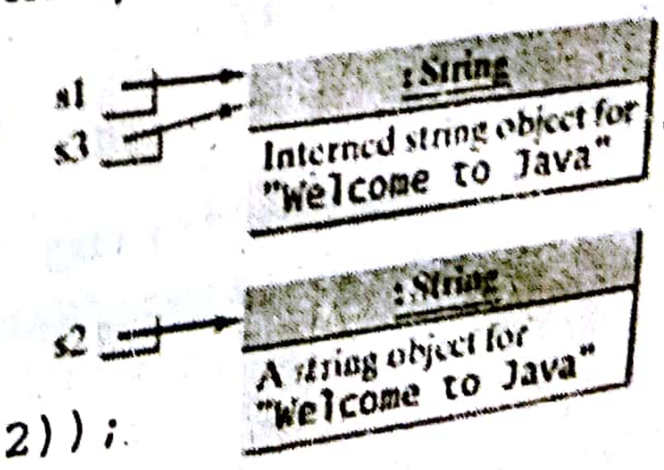
```
String s = "Java";
s = "HTML";
```

The answer is no. The first statement creates a **String** object with the content "Java" and assigns its reference to **s**.

The second statement creates a new **String** object with the content "HTML" and assigns its reference to s. The first **String** object still exists after the assignment, but it can no longer be accessed, because variable s now points to the new object, as shown in following figure.



Since strings are immutable and are ubiquitous in programming, the JVM uses a unique instance for string literals with the same character sequence in order to improve efficiency and save memory. Such an instance is called interned. For example,




```
String s1 = "Welcome to Java";
String s2 = new String("Welcome to Java");
String s3 = "Welcome to Java";

System.out.println("s1 == s2 is " + (s1 == s2));
System.out.println("s1 == s3 is " + (s1 == s3));
```

display

s1 == s2 is false

s1 == s3 is true

In the preceding statements, **s1** and **s3** refer to the same interned string "Welcome to Java", therefore **s1 == s3 is true**. However, **s1 == s2 is false**, because **s1** and **s2** are two different string objects, even though they have the same contents.

6.6 STRINGTOKENIZER

The processing of text often consists of parsing a formatted input string. Parsing is the division of text into a set of discrete parts, or tokens, which in a certain sequence can convey a semantic meaning. The StringTokenizerclass provides the first step in this parsing process, often called the lexer(lexical analyzer) or scanner. StringTokenizerimplements the Enumeration interface. Therefore, given an input string, you can enumerate the individual tokens contained in it using StringTokenizer. To use StringTokenizer, you specify an input string and a string that contains delimiters. Delimiters are characters that separate tokens. Each character in the delimiters string is considered a valid delimiter-for example, ",;:"sets the delimiters to a comma, semicolon, and colon. The default set of delimiters consists of the whitespace characters: space, tab, newline, and carriage return.

The StringTokenizerconstructors are shown here:

```
StringTokenizer(String str)
StringTokenizer(String str, String delimiters)
StringTokenizer(String str, String delimiters, booleandelimAsToken)
```

In all versions, stris the string that will be tokenized. In the first version, the default delimiters are used. In the second and third versions, delimiters is a string that specifies the delimiters. In the third version, if delimAsTokenis true, then the delimiters are also returned as tokens when the string is parsed. Otherwise, the delimiters are not returned.

Delimiters are not returned as tokens by the first two forms.

Once you have created a StringTokenizerobject, the nextToken( ) method is used to extract consecutive tokens. The hasMoreTokens( ) method returns true while there are more tokens to be extracted. Since StringTokenizerimplements Enumeration, the hasMoreElements( ) and nextElement( ) methods are also implemented, and they act the same as hasMoreTokens( ) and nextToken( ), respectively. The StringTokenizermethods are shown in Table 16-1.

Here is an example that creates a StringTokenizerto parse "key=value" pairs. Consecutive sets of "key=value" pairs are separated by a semicolon.

```
                    // Demonstrate StringTokenizer.
importjava.util.StringTokenizer;
E JAVA LIBRARY
classSTDemo
{
    static String in = "title=Java ;" + "author=rakheech          ;"
    "publisher=bhavya Books ;" + "copyright=2013";
public static void main(String args[])
{
    StringTokenizerst = new StringTokenizer(in, "=;");
    while(st.hasMoreTokens())
    {
        String key = st.nextToken();
        String val = st.nextToken();
        System.out.println(key + "\t" + val);
    }
}
```

Method	Description
`intcountTokens( )`	Using the current set of delimiters, the method determines the numberof tokens left to be parsed and returns the result.
`booleanhasMoreElements( )`	Returns true if one or more tokens remain in the string and returns false if there are none.
`booleanhasMoreTokens( )`	Returns true if one or more tokens remain in the string and returns false if there are none.
`Object nextElement( )`	Returns the next token as an Object.
`String nextToken( )`	Returns the next token as a String.
`String nextToken(String delimiters)`	Returns the next token as a String and sets the delimiters string to that specifiedby delimiters.

6.7. STRING BUFFERS

The fact that a String object is immutable is both a blessing and a curse. Immutability is a blessing because only one copy of a String object exists (conserving memory). Immutability is also a curse because attempts to modify String objects result in new String objects being created, which increases a program's memory requirements. And, if the activity takes place in a loop, the creation of those additional String objects can have a negative impact on performance. To deal with those problems, Java provides StringBuffer.

6.7.1 Constructing String Buffers

The StringBuffer class (located in the java.lang package) is used to create objects that represent mutable (read/write) strings. As with String, StringBuffer stores a string in an internal char [] value array field. Unlike String, StringBuffer dynamically expands that array (as necessary) to accommodate the string as it grows.

6.7.2 Constructors of String Buffer

Following constructors can be called to create a StringBuffer object.

1. `StringBuffer()`
2. `StringBuffer(int length)`
3. `StringBuffer(String s)`

1. StringBuffer()

It creates that object with an internal char [] value field that contains no characters and (initially) is capable of storing sixteen characters before being expanded.

2. StringBuffer(int length)

It creates a StringBuffer object with an internal char [] value field that contains no characters and can store length characters, prior to being resized. However, if length contains a negative value, that constructor throws a NegativeArraySizeException object.

3. StringBuffer(String s)

It creates a StringBuffer object with an internal char [] value field that contains all characters found in s

Example

The following code fragment uses the StringBuffer constructors to create string buffers:

```
// Create an empty StringBuffer whose internal array is capable of
// storing 16 characters before being expanded.
StringBuffer sb1 = new StringBuffer ();
// Create an empty StringBuffer whose internal array is capable of
// storing 30 characters before being expanded.
StringBuffer sb2 = new StringBuffer (30);
// Create a StringBuffer whose internal array contains characters
// A, B and C; and is capable of storing 19 characters (which
// includes A, B and C) before // being expanded.
StringBuffer sb3 = new StringBuffer ("ABC");
```

After a StringBuffer has been created and the string, represented by StringBuffer's internal array, has been modified, that string can be converted to a new String object by calling toString().

The following code fragment converts a StringBuffer's contents to a new String object by calling toString():

```
StringBuffersb = new StringBuffer ("ABC");
String s = sb.toString ();
```