Chapter 9 Strings

Liang, Introduction to Java Programming, Eighth Edition, (c) 2011 Pearson Education, Inc. All rights reserved. 0132130807

Constructing Strings

String newString = new String(stringLiteral);

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

Since strings are used frequently, Java provides a shorthand initializer for creating a string:

String message = "Welcome to Java";

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

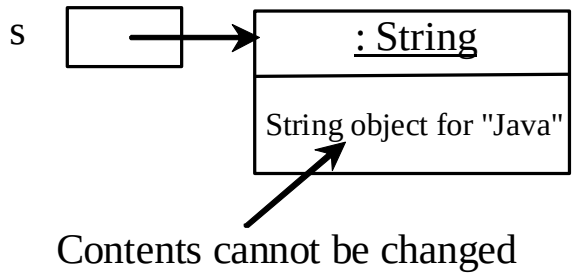
```
String s = "Java";
s = "HTML";
```

animation

Trace Code

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

After executing `String s = "Java";`

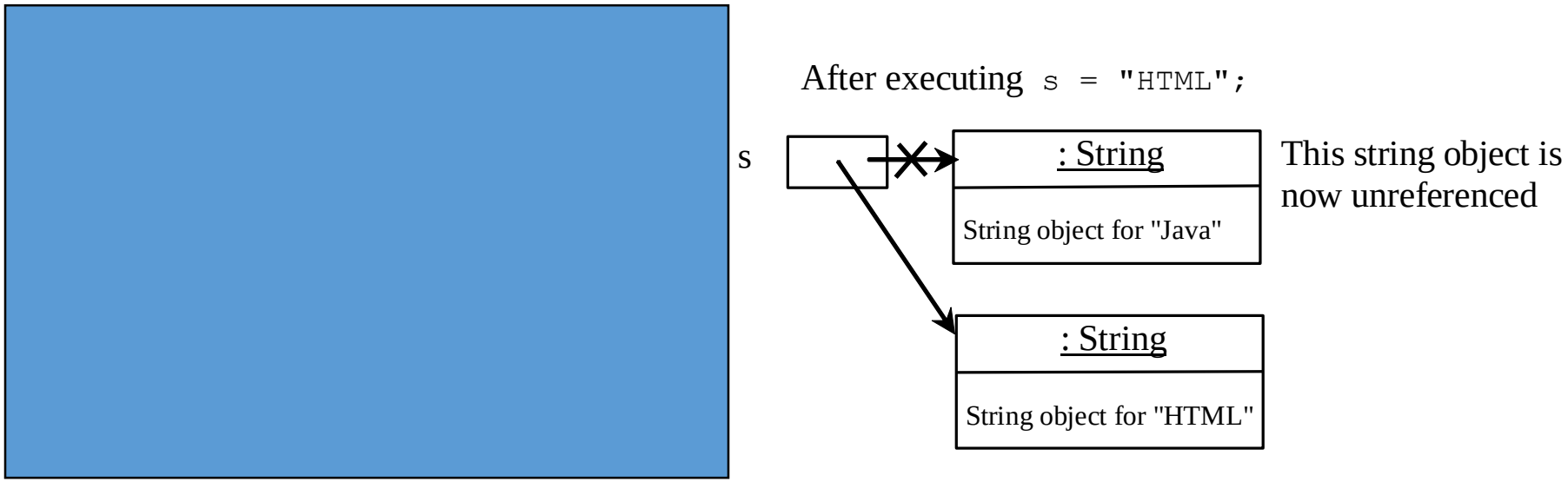


animation

Trace Code

String s = "Java";

s = "HTML";



String Comparisons

java.lang.String	
+equals(s1: String): boolean	Returns true if this string is equal to string s1.
+equalsIgnoreCase(s1: String): boolean	Returns true if this string is equal to string s1 case-insensitive.
+compareTo(s1: String): int	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than s1.
+compareToIgnoreCase(s1: String): int	Same as compareTo except that the comparison is case-insensitive.
+regionMatches(toffset: int, s1: String, offset: int, len: int): boolean	Returns true if the specified subregion of this string exactly matches the specified subregion in string s1.
+regionMatches(ignoreCase: boolean, toffset: int, s1: String, offset: int, len: int): boolean	Same as the preceding method except that you can specify whether the match is case-sensitive.
+startsWith(prefix: String): boolean	Returns true if this string starts with the specified prefix.
+endsWith(suffix: String): boolean	Returns true if this string ends with the specified suffix.

String Comparisons

- equals

```
String s1 = new String("Welcome“);
String s2 = "welcome";

 if (s1.equals(s2)){
   // s1 and s2 have the same contents
 }

 if (s1 == s2) {
  // s1 and s2 have the same reference
 }
```

String Comparisons, cont.

- `compareTo(Object object)`

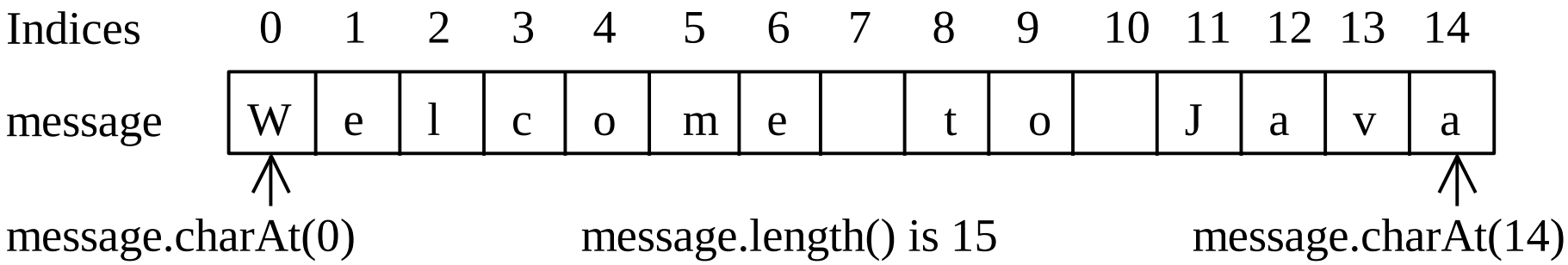
```
String s1 = new String("Welcome‟);
String s2 = "welcome";

 if (s1.compareTo(s2) > 0) {
   // s1 is greater than s2
 }
 else if (s1.compareTo(s2) == 0) {
   // s1 and s2 have the same contents
 }
 else
    // s1 is less than s2
```

String Length, Characters, and Combining Strings

java.lang.String	
+length(): int	Returns the number of characters in this string.
+charAt(index: int): char	Returns the character at the specified index from this string.
+concat(s1: String): String	Returns a new string that concatenate this string with string s1.

Retrieving Individual Characters in a String

- Do not use `message[0]`
- Use `message.charAt(index)`
- Index starts from `0`



String Concatenation

```
String s3 = s1.concat(s2);
```

String s3 = s1 + s2;

s1 + s2 + s3 + s4 + s5 same as

(((s1.concat(s2)).concat(s3)).concat(s4)).concat(s5);

Liang, Introduction to Java Programming, Eighth Edition, (c) 2011 Pearson Education, Inc. All rights reserved. 0132130807

Extracting Substrings

java.lang.String
+subString(beginIndex: int): String +subString(beginIndex: int, endIndex: int): String

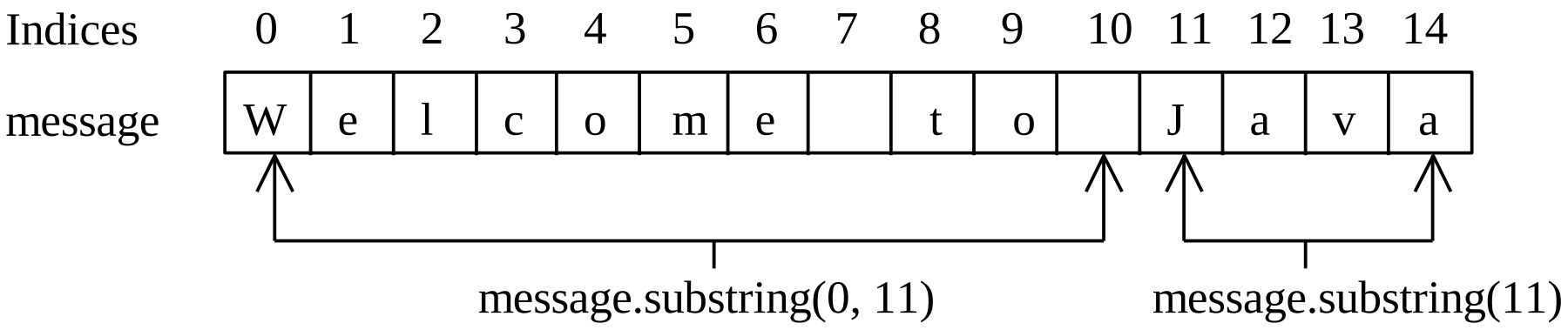
Returns this string's substring that begins with the character at the specified beginIndex and extends to the end of the string, as shown in Figure 8.6.

Returns this string's substring that begins at the specified beginIndex and extends to the character at index endIndex – 1, as shown in Figure 8.6. Note that the character at endIndex is not part of the substring.

Extracting Substrings

You can extract a single character from a string using the charAt method. You can also extract a substring from a string using the substring method in the String class.

```
String s1 = "Welcome to Java";
String s2 = s1.substring(0, 11) + "HTML";
```



Convert Character and Numbers to Strings

The String class provides several static valueOf methods for converting a character, an array of characters, and numeric values to strings. These methods have the same name valueOf with different argument types char, char[], double, long, int, and float. For example, to convert a double value to a string, use String.valueOf(5.44). The return value is string consists of characters ‘5’, ‘.’, ‘4’, and ‘4’.

`StringBuilder` and `StringBuffer`

The `StringBuilder`/`StringBuffer` class is an alternative to the `String` class. In general, a StringBuilder/StringBuffer can be used wherever a string is used. StringBuilder/StringBuffer is more flexible than String. You can add, insert, or append new contents into a string buffer, whereas the value of a String object is fixed once the string is created.

Main Method Is Just a Regular Method

You can call a regular method by passing actual parameters. Can you pass arguments to main? Of course, yes. For example, the main method in class B is invoked by a method in A, as shown below:

```
public class A {
  public static void main(String[] args) {
    String[] strings = {"New York",
      "Boston", "Atlanta"};
    B.main(strings);
  }
}
```

```
class B {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i]);
  }
}
```

Command-Line Parameters

```
class TestMain {
  public static void main(String[] args) {
  ...
  }
}
```

```
java TestMain arg0 arg1 arg2 ... argn
```

In the main method, get the arguments from `args[0], args[1], ..., args[n]`, which corresponds to `arg0, arg1, ..., argn` in the command line.