8 THE STRING CLASS

Terry Marris 16 April 2001

8.1 OBJECTIVES

By the end of this lesson the student should be able to

- appreciate the difference between a char and a string
- understand and use the String class methods

8.2 PRE-REQUISITES

The student should know what a *char* is, should understand what the ASCII collating sequence is, and should be comfortable with using messages, receivers, parameters, arguments and return values.

8.3 PREVIEW

We see that a string is just an indexed sequence of characters. We can join two strings together, extract substrings from strings, create new strings and test strings for equality.

The standard Java package includes many useful classes. One such class is named *String*.

We summarise the most useful *String* class methods.

8.4 INTRODUCTION

In Chapter Nine we saw that a variable of type *char* holds just one character at a time. A *String* object holds any number of characters at a time. So where a *char* variable can hold the value *M*, a string value could be *Medium*.

```
char aChar = 'M';
String aString = "Medium";
```

8.5 STRINGS

We view a string as a sequence of characters. The sequence is always indexed from zero upwards.

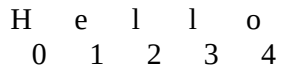


Figure 8.1 *A String is an indexed sequence of characters*

H is in index position zero. *e* is in index position one. And *o* is in index position four.

Notice the difference between

```
char aChar = 'M';
```
and
```
String aString = "M";
```

aChar holds just one character; it cannot hold more than one character at a time. *aString* can hold many characters at a time but just happens to hold one character in this case.

The quotes themselves are not part of the string; they serve to just mark its ends.

8.6 charAt(int)

The *char charAt(int index)* method returns the character at the given index location.

We define *aString* with value "*Catastrophe*"

```
String aString = "Catastrophe";
```

And picture

C	a	t	a	s	t	r	o	p	h	e
0	1	2	3	4	5	6	7	8	9	10

The statement

```
char aChar = aString.charAt(0);
```

would leave *aChar* containing *'C'*, the character at index position zero.

If we sent the message *charAt(10)* to *aString*, the character *'e'*, the character at index position ten, would be returned.

If we sent the message *charAt(-1)* or the message *charAt(11)* to *aString*, an *index out of bounds* error would occur because *aString* has no character at index position -1 or at index position 11.

8.7 indexOf(String)

The *int indexOf(String s)* method returns the index position of the given string, *s* in *this string*.

this string refers to the receiver, that is, the object receiving the message.

So, given

```
String aString = "Catastrophe";
int index = aString.indexOf("strop");
```

this string corresponds to *aString*. The parameter *String s* corresponds to the argument *"strop"*.

```
int index = aString.indexOf("strop");
```

corresponds to this string

corresponds to parameter String s

Figure 8.2 *Relation between this object and receiver, parameter and argument*

aString.indexOf("strop") returns four because "strop" begins at index location four in *aString*. And so four is stored in the *int* variable *index*.

aString.indexOf("C") returns zero. Notice the double quotes around the C. These are necessary because *indexOf(String)* demands a *String* argument value to satisfy its *String* parameter.

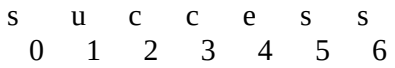
aString.indexOf("strap") returns -1 because *"strap"* is not an unbroken sequence of characters within *aString*. *aString.indexOf("X")* returns -1 for the same reason.

What would *aString.indexOf("c")* return? -1? Why?

8.8 length()

The length of a string is just the number of characters it contains. Look at

```
String aString = "success";
```



The length of *aString* is seven (just count the number of characters it contains - do not include the quotes).

The length of a string is always just one more than its last index value (six in our example). The last index value is always just one less than the length of the string.

```
String aString = "success";
int sizeOfAString = aString.length();
```

leaves *sizeOfAString* with value seven.

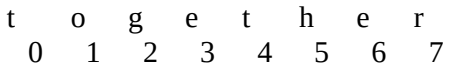
The *int length()* method returns an *int* that represents the number of characters in the string. It has no parameters.

8.9 subString(int, int)

A substring is just part of a string. It could be the entire string itself. It could be the empty string. Substrings of "*together*" include "*to*", "*get*", "*her*", "*together*" and "".

The method *String substring(int from, int to)* returns a new string that is a substring of this string, starting with index *from* up to (but not including) index *to*. *from* must be less than *to*. *from* cannot be negative nor can *to* be larger than the length of this string.

```
String aString = "together";
```




```
String aSubString = aString.substring(2, 5);
```

leaves *aSubString* with value "*get*", the characters in *aString* from index position two up to (but not including) index position five.

Notice

aString.substring(0, 2) returns "*to*".

aString.substring(5, 7) returns "*he*".

aString.substring(5, 8) returns "*her*"

But *aString.substring(5, 9)* returns an *index out of bounds* error because there is no character in index location eight.

aString.substring(2, 0) also returns an error because the first argument must be less than the second.

8.10 EQUALITY

Two strings are equal when they both contain exactly the same characters in exactly the same order and are both exactly the same length.

These two strings, "*cat*" and "*cat*", are equal.

These two strings, "*cat*" and "*cut*", are not equal. Neither are these two strings: "*cat*" and "*Cat*". Case matters. 'c' ≠ 'C' remember.

The *boolean equals(String s)* method returns *true* if this string has the same sequence of characters that *s* has.

```
String aString = "cat";
boolean isEqual = aString.equals("cat");
```

leaves *isEqual* with the value *true*.

But

```
String aString = "cat";
boolean isEqual = aString.equals("Cat");
```

leaves *isEqual* with the value *false*.

If we wish to regard words to be the same even if one contains capitals, and one does not, we uses the *boolean equalsIgnoreCase(String s)* method. The method returns *true* if this string has the same sequence of characters that *s* has when case is not taken into account.

```
String aString = "cat";
boolean isEqual = aString.equalsIgnoreCase("Cat");
```

leaves *isEqual* with the value *true*.

8.11 compareTo(String)

The *int compareTo(String s)* method returns an *int* less than zero (e.g. -5) if this string comes before s in the ASCII collating sequence, zero if this string is identical to s and an *int* greater than zero (e.g. 7) 0 if this string comes after s.

For example

```
int result = "bat".compareTo("cat");
```

leaves *result* with a negative value. We are not too concerned with the actual value, just with the fact that it is less than zero.

```
int result = "cat".compareTo("cat");
```

leaves *result* with the value zero.

```
int result = "cat".compareTo("bat");
```

leaves *result* with a positive value.

8.12 toUpperCase()

The method *String toUpperCase()* returns a new string that is a copy of this string except that all lower case characters are converted to upper case. Any upper case characters in this string are left unchanged. For example

```
String aString = "Cat";
String anUpperCaseString = aString.toUpperCase();
```

leaves *anUpperCaseString* with the character sequence "*CAT*".

8.13 replace(char, char)

The method *String replace(char oldCh, char newCh)* returns a new string that results from replacing all occurrences of *oldCh* with *newCh* in this string. If *oldCh* does not occur in this string, this string is returned unchanged.

For example

`"purple purpoise".replace('p', 't')` returns *turtle tortoise*

`"tom jones".replace('X', 'Y')` returns *tom jones*

8.14 CONCATENATION

The concatenation operator, +, does two things:

- it joins one string onto the end of another
- it converts whatever follows it to a string, if it can.

Look at the Java code fragment shown below.

```
int temperature = 20;
System.out.println("Temperature is " + temperature);
```

When executed, you expect

```
Temperature is 20
```

to be displayed on the monitor.

The concatenation operator conveniently converts the *temperature* (an *int*) into a string for us.

8.15 STRING CLASS TEST PROGRAM

We try out a few *String* class methods in a simple test program.

```
/* TestStrings.java
   Terry Marris  August 2000
*/


public class TestStrings {
  public static void main(String[] s)
  {
    String greeting = "Hello sailor!";
    System.out.println(greeting);
    String newGreeting = greeting.substring(0, 6) + "ladies!";
    System.out.println(newGreeting);
    System.out.println("The length of the new greeting is " +
                        newGreeting.length());

    String hello = "hello";
    String HELLO = "HELLO";
    System.out.println("hello equals HELLO: " +
                       hello.equals(HELLO));
    System.out.println("hello equals HELLO ignoring case: " +
                       hello.equalsIgnoreCase(HELLO));
  }
}
```

The program run is shown below.

```
Hello sailor!
Hello ladies!
The length of the new greeting is 13
hello equals HELLO: false
hello equals HELLO ignoring case: true
```

Exercise Identify which line of coding is responsible for each line of output.

8.16 STRING CLASS DOCUMENTATION

Some useful methods for dealing with string objects are shown in the following tables.

Constructors

String()	initialises a newly created *String* object so that it represents an empty string.
String(char[] s)	initialises a newly created *String* object with the given array of characters
String(String s)	initialises a newly created *String* that is a copy of the given *String*, s.

Methods

char	charAt(int index)	returns the character at the given index if the index is between 0 and length()-1.
int	compareTo(String s)	returns an int < 0 if this string comes before s in the ASCII collating sequence, 0 if this string is identical to s and an int > 0 if this string comes after s.
int	compareToIgnoreCase(String s)	same as compareTo(String s) but case is not taken into account.
boolean	endsWith(String suffix)	returns true if this string ends with the given suffix, or if s is the empty string, or if s equals this string.
boolean	equals(String s)	returns true if this string has the same sequence of characters that s has.
boolean	equalsIgnoreCase(String s)	returns true if this string has the same sequence of characters that s has, when case is not taken into account.
int	indexOf(String s)	returns the position in this string of the first occurrence of the given string s.
int	length()	returns the number of characters in this string.
String	replace(char oldCh, char newCh)	returns a new string that results from replacing all occurrences of oldCh with newCh in this string.
boolean	startsWith(String s)	returns true if this string starts with the given *String*, s.
String	substring(int from, int to)	returns a new string that is a substring of this string, starting with index *from* up to (but not including) index *to*. *from* must be less than *to*. *from* cannot be negative nor can *to* be larger than the length of this string.
String	toLowerCase()	returns a new string that is a copy of this string except that all upper case characters are converted to lower case.
String	toUpperCase()	returns a new string that is a copy of this string except that all lower case characters are converted to upper case.
String	trim()	returns a new string that is a copy of this string except that all leading and trailing spaces are removed.

8.17 FURTHER READING

HORSTMANN & CORNELL *Core Java 2 Volume 1 Fundamentals* pp 59

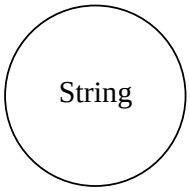
The document *String.html* that may be found at *c:\jdk1.2\docs\api\java\lang*. You could use your regular WEB browser to view this document.

8.18 EXERCISES

1 Print out and read the document *String.html* that may be found at *c:\jdk1.2\docs\api\java\lang*. You could use your regular WEB browser to view and print this document.

2 Try out all the methods listed in the table of §8.16 - String Class Documentation.

8.19 REVIEW



a sequence of zero to many characters

indexed from zero

enclosed within double quotes

charAt(int i) returns the character at location i

indexOf(String s) returns the location of s in this string

equals(String s) returns true if this string and s are the same

equalsIgnoreCase(String s) returns true regardless of case

replace(char ch1, char ch2) replaces every ch1 in this string
with ch2

substring(int start, int stop) returns the chars from position start
up to just before position stop

toUpperCase() returns this string entirely in capital letters

toLowerCase() returns this string entirely in small letters

length() returns the number of characters in this string

compareTo(String s) returns a negative number if this string < s
0 if this string == s
a positive number if this string > s

compareToIgnoreCase(String s) same as compareTo() but case
is ignored