Some Methods in the Class `String` (Part 1 of 8)

Display 1.4 Some Methods in the Class `String`

`int length()`

Returns the length of the calling object (which is a string) as a value of type `int`.

EXAMPLE

After program executes `String greeting = "Hello!";`
`greeting.length()` returns 6.

`boolean equals(Other_String)`

Returns `true` if the calling object string and the *Other_String* are equal. Otherwise, returns `false`.

EXAMPLE

After program executes `String greeting = "Hello";`
`greeting.equals("Hello")` returns `true`
`greeting.equals("Good-Bye")` returns `false`
`greeting.equals("hello")` returns `false`

Note that case matters. `"Hello"` and `"hello"` are not equal because one starts with an uppercase letter and the other starts with a lowercase letter.

(continued)

Some Methods in the Class `String` (Part 2 of 8)

Display 1.4 Some Methods in the Class `String`

`boolean equalsIgnoreCase(`*Other_String*`)`

Returns `true` if the calling object string and the *Other_String* are equal, considering uppercase and lowercase versions of a letter to be the same. Otherwise, returns `false`.

EXAMPLE

After program executes `String name = "mary!";`
`greeting.equalsIgnoreCase("Mary!")` returns `true`

`String toLowerCase()`

Returns a string with the same characters as the calling object string, but with all letter characters converted to lowercase.

EXAMPLE

After program executes `String greeting = "Hi Mary!";`
`greeting.toLowerCase()` returns `"hi mary!"`.

(continued)

Some Methods in the Class `String` (Part 3 of 8)

Display 1.4 Some Methods in the Class `String`

`String toUpperCase()`

Returns a string with the same characters as the calling object string, but with all letter characters converted to uppercase.

EXAMPLE

After program executes `String greeting = "Hi Mary!";`
`greeting.toUpperCase()` returns `"HI MARY!"`.

`String trim()`

Returns a string with the same characters as the calling object string, but with leading and trailing white space removed. Whitespace characters are the characters that print as white space on paper, such as the blank (space) character, the tab character, and the new-line character `'\n'`.

EXAMPLE

After program executes `String pause = "   Hmm   ";`
`pause.trim()` returns `"Hmm"`.

(continued)

Some Methods in the Class `String` (Part 4 of 8)

Display 1.4 Some Methods in the Class `String`

`char charAt(`*Position*`)`

Returns the character in the calling object string at the *Position*. Positions are counted 0, 1, 2, etc.

EXAMPLE

After program executes `String greeting = "Hello!";`
`greeting.charAt(0)` returns `'H'`, and
`greeting.charAt(1)` returns `'e'`.

`String substring(`*Start*`)`

Returns the substring of the calling object string starting from *Start* through to the end of the calling object. Positions are counted 0, 1, 2, etc. Be sure to notice that the character at position *Start* is included in the value returned.

EXAMPLE

After program executes `String sample = "AbcdefG";`
`sample.substring(2)` returns `"cdefG"`.

(continued)

Some Methods in the Class `String` (Part 5 of 8)

Display 1.4 Some Methods in the Class `String`

`String substring(`*Start*`,` *End*`)`

Returns the substring of the calling object string starting from position *Start* through, but not including, position *End* of the calling object. Positions are counted 0, 1, 2, etc. Be sure to notice that the character at position *Start* is included in the value returned, but the character at position *End* is not included.

EXAMPLE

After program executes `String sample = "AbcdefG";`
`sample.substring(2, 5)` returns `"cde"`.

`int indexOf(`*A_String*`)`

Returns the index (position) of the first occurrence of the string *A_String* in the calling object string. Positions are counted 0, 1, 2, etc. Returns −1 if *A_String* is not found.

EXAMPLE

After program executes `String greeting = "Hi Mary!";`
`greeting.indexOf("Mary")` returns 3, and
`greeting.indexOf("Sally")` returns −1.

(continued)

Some Methods in the Class `String` (Part 6 of 8)

Display 1.4 Some Methods in the Class `String`

`int indexOf(A_String, Start)`

Returns the index (position) of the first occurrence of the string *A_String* in the calling object string that occurs at or after position *Start*. Positions are counted 0, 1, 2, etc. Returns −1 if *A_String* is not found.

EXAMPLE

After program executes `String name = "Mary, Mary quite contrary";`
`name.indexOf("Mary", 1)` returns 6.
The same value is returned if 1 is replaced by any number up to and including 6.
`name.indexOf("Mary", 0)` returns 0.
`name.indexOf("Mary", 8)` returns −1.

`int lastIndexOf(A_String)`

Returns the index (position) of the last occurrence of the string *A_String* in the calling object string. Positions are counted 0, 1, 2, etc. Returns −1, if *A_String* is not found.

EXAMPLE

After program executes `String name = "Mary, Mary, Mary quite so";`
`greeting.indexOf("Mary")` returns 0, and
`name.lastIndexOf("Mary")` returns 12.

(continued)

Some Methods in the Class `String` (Part 7 of 8)

Display 1.4 Some Methods in the Class `String`

`int compareTo(A_String)`

Compares the calling object string and the string argument to see which comes first in the lexicographic ordering. Lexicographic order is the same as alphabetical order but with the characters ordered as in Appendix 3. Note that in Appendix 3 all the uppercase letters are in regular alphabetical order and all the lowercase letters are in alphabetical order, but all the uppercase letters precede all the lowercase letters. So, lexicographic ordering is the same as alphabetical ordering provided both strings are either all uppercase letters or both strings are all lowercase letters. If the calling string is first, it returns a negative value. If the two strings are equal, it returns zero. If the argument is first, it returns a positive number.

EXAMPLE

After program executes `String entry = "adventure";`
`entry.compareTo("zoo")` returns a negative number,
`entry.compareTo("adventure")` returns `0`, and
`entry.compareTo("above")` returns a positive number.

(continued)

Some Methods in the Class `String` (Part 8 of 8)

Display 1.4 Some Methods in the Class `String`

`int compareToIgnoreCase(`*A_String*`)`

Compares the calling object string and the string argument to see which comes first in the lexicographic ordering, treating uppercase and lowercase letters as being the same. (To be precise, all uppercase letters are treated as if they were their lowercase versions in doing the comparison.) Thus, if both strings consist entirely of letters, the comparison is for ordinary alphabetical order. If the calling string is first, it returns a negative value. If the two strings are equal ignoring case, it returns zero. If the argument is first, it returns a positive number.

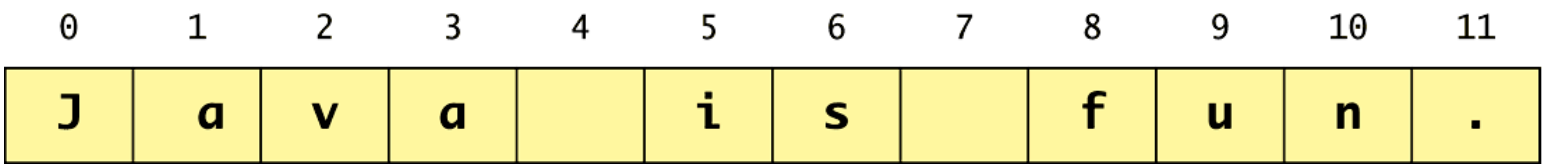
EXAMPLE

After program executes `String entry = "adventure";`
`entry.compareToIgnoreCase("Zoo")` returns a negative number,
`entry.compareToIgnoreCase("Adventure")` returns `0`, and
`"Zoo".compareToIgnoreCase(entry)` returns a positive number.

String Indexes

Display 1.5 String Indexes

The 12 characters in the string `"Java is fun."` have indexes `0` through `11`.



Notice that the blanks and the period count as characters in the string.

Escape Sequences

- A backslash (\) immediately preceding a character (i.e., without any space) denotes an *escape sequence* or an *escape character*
  - The character following the backslash does not have its usual meaning
  - Although it is formed using two symbols, it is regarded as a single character

Escape Sequences

Display 1.6 Escape Sequences

```
\" 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. White space up to the next tab stop.
```