Strings

Strings, which are widely used in Java programming, are a sequence of characters. In the Java programming language, strings are objects.

The Java platform provides the `String` class to create and manipulate strings.

Creating Strings

The most direct way to create a string is to write:

```java
String greeting = "Hello world!";
```

In this case, "Hello world!" is a string literal—a series of characters in your code that is enclosed in double quotes. Whenever it encounters a string literal in your code, the compiler creates a `String` object with its value—in this case, `Hello world!`.

As with any other object, you can create `String` objects by using the `new` keyword and a constructor. The `String` class has thirteen constructors that allow you to provide the initial value of the string using different sources, such as an array of characters:

```java
char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.'};
String helloString = new String(helloArray);
System.out.println(helloString);
```

The last line of this code snippet displays `hello`.

**Note:** The `String` class is immutable, so that once it is created a `String` object cannot be changed. The `String` class has a number of methods, some of which will be discussed below, that appear to modify strings. Since strings are immutable, what these methods really do is create and return a new string that contains the result of the operation.

String Length

Methods used to obtain information about an object are known as accessor methods. One accessor method that you can use with strings is the `length()` method, which returns the number of characters contained in the string object. After the following two lines of code have been executed, `len` equals 17:

```java
String palindrome = "Dot saw I was Tod";
int len = palindrome.length();
```

A palindrome is a word or sentence that is symmetric—it is spelled the same forward and backward, ignoring case and punctuation. Here is a short and inefficient program to
reverse a palindrome string. It invokes the `String method `charAt(i)`, which returns the i\textsuperscript{th} character in the string, counting from 0.

```java
public class StringDemo {
    public static void main(String[] args) {
        String palindrome = "Dot saw I was Tod";
        int len = palindrome.length();
        char[] tempCharArray = new char[len];
        char[] charArray = new char[len];

        // put original string in an array of chars
        for (int i = 0; i < len; i++) {
            tempCharArray[i] = palindrome.charAt(i);
        }

        // reverse array of chars
        for (int j = 0; j < len; j++) {
            charArray[j] = tempCharArray[len - 1 - j];
        }

        String reversePalindrome = new String(charArray);
        System.out.println(reversePalindrome);
    }
}
```

Running the program produces this output:

doT saw I was toD

To accomplish the string reversal, the program had to convert the string to an array of characters (first for loop), reverse the array into a second array (second for loop), and then convert back to a string. The `String` class includes a method, `getChars()`, to convert a string, or a portion of a string, into an array of characters so we could replace the first for loop in the program above with

```java
palindrome.getChars(0, len, tempCharArray, 0);
```

**Concatenating Strings**

The `String` class includes a method for concatenating two strings:

```java
string1.concat(string2);
```

This returns a new string that is string1 with string2 added to it at the end.

You can also use the `concat()` method with string literals, as in:

```
"My name is ".concat("Rumplestiltskin");
```

Strings are more commonly concatenated with the + operator, as in
"Hello," + " world" + "!"
which results in
"Hello, world!"
The + operator is widely used in print statements. For example:
String string1 = "saw I was ";
System.out.println("Dot " + string1 + "Tod");
which prints
Dot saw I was Tod
Such a concatenation can be a mixture of any objects. For each object that is not a
String, its toString() method is called to convert it to a String.

---

**Note:** The Java programming language does not permit literal strings to span lines in
source files, so you must use the + concatenation operator at the end of each line in a
multi-line string. For example,
String quote = "Now is the time for all good " +
        "men to come to the aid of their country.";
Breaking strings between lines using the + concatenation operator is, once again, very
common in print statements.

---

**Creating Format Strings**

You have seen the use of the printf() and format() methods to print output with
formatted numbers. The String class has an equivalent class method, format(), that
returns a String object rather than a PrintStream object.

Using String's static format() method allows you to create a formatted string that you
can reuse, as opposed to a one-time print statement. For example, instead of

System.out.printf("The value of the float variable is %f, while the
value of the " + "integer variable is %d, and the string is %s",
floatVar, intVar, stringVar);
you can write
String fs;
fs = String.format("The value of the float variable is %f, while the
value of the " + "integer variable is %d, and the string is %s",
floatVar, intVar, stringVar);
System.out.println(fs);

**Converting Strings to Numbers**
Frequently, a program ends up with numeric data in a string object—a value entered by the user, for example.

The `Number` subclasses that wrap primitive numeric types (Byte, Integer, Double, Float, Long, and Short) each provide a class method named `valueOf` that converts a string to an object of that type. Here is an example, `ValueOfDemo`, that gets two strings from the command line, converts them to numbers, and performs arithmetic operations on the values:

```java
public class ValueOfDemo {
    public static void main(String[] args) {

        //this program requires two arguments on the command line
        if (args.length == 2) {
            //convert strings to numbers
            float a = (Float.valueOf(args[0])).floatValue();
            float b = (Float.valueOf(args[1])).floatValue();

            //do some arithmetic
            System.out.println("a + b = " + (a + b));
            System.out.println("a - b = " + (a - b));
            System.out.println("a * b = " + (a * b));
            System.out.println("a / b = " + (a / b));
            System.out.println("a % b = " + (a % b));
        } else {
            System.out.println("This program requires two command-line arguments.");
        }
    }
}
```

The following is the output from the program when you use 4.5 and 87.2 for the command-line arguments:

```
a + b = 91.7
a - b = -82.7
a * b = 392.4
a / b = 0.0516055
a % b = 4.5
```

**Note:** Each of the `Number` subclasses that wrap primitive numeric types also provides a `parseXXXX()` method (for example, `parseFloat()`) that can be used to convert strings to primitive numbers. Since a primitive type is returned instead of an object, the
parseFloat() method is more direct than the valueOf() method. For example, in the ValueOfDemo program, we could use:
float a = Float.parseFloat(args[0]);
float b = Float.parseFloat(args[1]);

**Converting Numbers to Strings**

Sometimes you need to convert a number to a string because you need to operate on the value in its string form. There are several easy ways to convert a number to a string:

```java
int i;
String s1 = "" + i; //Concatenate "i" with an empty string;
//conversion is handled for you.

or
String s2 = String.valueOf(i); //The valueOf class method.
```

Each of the Number subclasses includes a class method, toString(), that will convert its primitive type to a string. For example:

```java
int i;
double d;
String s3 = Integer.toString(i);
String s4 = Double.toString(d);
```

The **ToStringDemo** example uses the toString method to convert a number to a string. The program then uses some string methods to compute the number of digits before and after the decimal point:

```java
public class ToStringDemo {

    public static void main(String[] args) {
        double d = 858.48;
        String s = Double.toString(d);

        int dot = s.indexOf('.');

        System.out.println(dot + " digits before decimal point.");
        System.out.println((s.length() - dot - 1) + " digits after decimal point.");
    }
}
```

The output of this program is:
3 digits before decimal point.
2 digits after decimal point.
Manipulating Characters in a String

The String class has a number of methods for examining the contents of strings, finding characters or substrings within a string, changing case, and other tasks.

Getting Characters and Substrings by Index

You can get the character at a particular index within a string by invoking the `charAt()` accessor method. The index of the first character is 0, while the index of the last character is `length() - 1`. For example, the following code gets the character at index 9 in a string:

```java
String anotherPalindrome = "Niagara. O roar again!";
char aChar = anotherPalindrome.charAt(9);
```

Indices begin at 0, so the character at index 9 is 'O', as illustrated in the following figure:

![Figure showing characters at indices]

If you want to get more than one consecutive character from a string, you can use the `substring` method. The `substring` method has two versions, as shown in the following table:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String substring(int beginIndex, int endIndex)</code></td>
<td>Returns a new string that is a substring of this string. The first integer argument specifies the index of the first character. The second integer argument is the index of the last character - 1.</td>
</tr>
<tr>
<td><code>String substring(int beginIndex)</code></td>
<td>Returns a new string that is a substring of this string. The integer argument specifies the index of the first character. Here, the returned substring extends to the end of the original string.</td>
</tr>
</tbody>
</table>

The following code gets from the Niagara palindrome the substring that extends from index 11 up to, but not including, index 15, which is the word "roar":

```java
String anotherPalindrome = "Niagara. O roar again!";
```
Other Methods for Manipulating Strings

Here are several other String methods for manipulating strings:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String[] split(String regex) String[] split(String regex, int limit)</td>
<td>Searches for a match as specified by the string argument (which contains a regular expression) and splits this string into an array of strings accordingly. The optional integer argument specifies the maximum size of the returned array. Regular expressions are covered in the lesson titled &quot;Regular Expressions.&quot;</td>
</tr>
<tr>
<td>CharSequence subSequence(int beginIndex, int endIndex)</td>
<td>Returns a new character sequence constructed from beginIndex index up until endIndex - 1.</td>
</tr>
<tr>
<td>String trim()</td>
<td>Returns a copy of this string with leading and trailing white space removed.</td>
</tr>
<tr>
<td>String toLowerCase() String toUpperCase()</td>
<td>Returns a copy of this string converted to lowercase or uppercase. If no conversions are necessary, these methods return the original string.</td>
</tr>
</tbody>
</table>

Searching for Characters and Substrings in a String

Here are some other String methods for finding characters or substrings within a string. The String class provides accessor methods that return the position within the string of a specific character or substring: indexOf() and lastIndexOf(). The indexOf() methods search forward from the beginning of the string, and the lastIndexOf()
methods search backward from the end of the string. If a character or substring is not found, `indexOf()` and `lastIndexOf()` return -1.

The `String` class also provides a search method, `contains`, that returns true if the string contains a particular character sequence. Use this method when you only need to know that the string contains a character sequence, but the precise location isn't important.

The following table describes the various string search methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int indexOf(int ch)</code></td>
<td>Returns the index of the first (last) occurrence of the specified character.</td>
</tr>
<tr>
<td><code>int lastIndexOf(int ch)</code></td>
<td></td>
</tr>
<tr>
<td><code>int indexOf(int ch, int fromIndex)</code></td>
<td>Returns the index of the first (last) occurrence of the specified character,</td>
</tr>
<tr>
<td></td>
<td>searching forward (backward) from the specified index.</td>
</tr>
<tr>
<td><code>int indexOf(String str)</code></td>
<td>Returns the index of the first (last) occurrence of the specified substring.</td>
</tr>
<tr>
<td><code>int lastIndexOf(String str)</code></td>
<td></td>
</tr>
<tr>
<td><code>int indexOf(String str, int fromIndex)</code></td>
<td>Returns the index of the first (last) occurrence of the specified substring,</td>
</tr>
<tr>
<td></td>
<td>searching forward (backward) from the specified index.</td>
</tr>
<tr>
<td><code>boolean contains(CharSequence s)</code></td>
<td>Returns true if the string contains the specified character sequence.</td>
</tr>
</tbody>
</table>

**Note:** `CharSequence` is an interface that is implemented by the `String` class. Therefore, you can use a string as an argument for the `contains()` method.

**Replacing Characters and Substrings into a String**

The `String` class has very few methods for inserting characters or substrings into a string. In general, they are not needed: You can create a new string by concatenation of substrings you have removed from a string with the substring that you want to insert.
The `String` class does have four methods for replacing found characters or substrings, however. They are:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String replace(char oldChar, char newChar)</code></td>
<td>Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.</td>
</tr>
<tr>
<td><code>String replace(CharSequence target, CharSequence replacement)</code></td>
<td>Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.</td>
</tr>
<tr>
<td><code>String replaceAll(String regex, String replacement)</code></td>
<td>Replaces each substring of this string that matches the given regular expression with the given replacement.</td>
</tr>
<tr>
<td><code>String replaceFirst(String regex, String replacement)</code></td>
<td>Replaces the first substring of this string that matches the given regular expression with the given replacement.</td>
</tr>
</tbody>
</table>

An Example

The following class, `Filename`, illustrates the use of `lastIndexOf()` and `substring()` to isolate different parts of a file name.

Note: The methods in the following `Filename` class don't do any error checking and assume that their argument contains a full directory path and a filename with an extension. If these methods were production code, they would verify that their arguments were properly constructed.

```java
public class Filename {
    private String fullPath;
    private char pathSeparator, extensionSeparator;

    public Filename(String str, char sep, char ext) {
        fullPath = str;
        pathSeparator = sep;
        extensionSeparator = ext;
    }
}
```
public String extension() {
    int dot = fullPath.lastIndexOf(extensionSeparator);
    return fullPath.substring(dot + 1);
}

public String filename() { // gets filename without extension
    int dot = fullPath.lastIndexOf(extensionSeparator);
    int sep = fullPath.lastIndexOf(pathSeparator);
    return fullPath.substring(sep + 1, dot);
}

public String path() {
    int sep = fullPath.lastIndexOf(pathSeparator);
    return fullPath.substring(0, sep);
}

Here is a program, FilenameDemo, that constructs a Filename object and calls all of its methods:

public class FilenameDemo {
    public static void main(String[] args) {
        final String FPATH = "/home/mem/index.html";
        Filename myHomePage = new Filename(FPATH,
            '/', '.');
        System.out.println("Extension = " +
            myHomePage.extension());
        System.out.println("Filename = " +
            myHomePage.filename());
        System.out.println("Path = " +
            myHomePage.path());
    }
}

And here's the output from the program:
Extension = html
Filename = index
Path = /home/mem

As shown in the following figure, our extension method uses lastIndexOf to locate the last occurrence of the period (.) in the file name. Then substring uses the return value of lastIndexOf to extract the file name extension — that is, the substring from the period to the end of the string. This code assumes that the file name has a period in it; if the file name does not have a period, lastIndexOf returns -1, and the substring method throws a StringIndexOutOfBoundsException.
Also, notice that the extension method uses dot + 1 as the argument to substring. If the period character (.) is the last character of the string, dot + 1 is equal to the length of the string, which is one larger than the largest index into the string (because indices start at 0). This is a legal argument to substring because that method accepts an index equal to, but not greater than, the length of the string and interprets it to mean "the end of the string."

Questions and Exercises: Characters and Strings

Questions

1. Consider the following string:
2. String hannah = "Did Hannah see bees? Hannah did.";
   a. What is the value displayed by the expression hannah.length()?
   b. What is the value returned by the method call hannah.charAt(12)?
   c. Write an expression that refers to the letter b in the string referred to by hannah.

3. How long is the string returned by the following expression? What is the string?

   "Was it a car or a cat I saw?".substring(9, 12)

Exercises

1. Show two ways to concatenate the following two strings together to get the string
   "Hi, mom."
   String hi = "Hi, ";
   String mom = "mom.";

2. Write a program that computes your initials from your full name and displays them.

3. An anagram is a word or a phrase made by transposing the letters of another word or phrase; for example, "parliament" is an anagram of "partial men," and "software" is an anagram of "swear oft." Write a program that figures out whether one string is an anagram of another string. The program should ignore white space and punctuation.