

OTHERWISE, AND EXPRESSLY DISCLAIM ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. McGraw-Hill and its licensors do not warrant or guarantee that the functions contained in the work will meet your requirements or that its operation will be uninterrupted or error free. Neither McGraw-Hill nor its licensors shall be liable to you or anyone else for any inaccuracy, error or omission, regardless of cause, in the work or for any damages resulting therefrom. McGraw-Hill has no responsibility for the content of any information accessed through the work. Under no circumstances shall McGraw-Hill and/or its licensors be liable for any indirect, incidental, special, punitive, consequential or similar damages that result from the use of or inability to use the work, even if any of them has been advised of the possibility of such damages. This limitation of liability shall apply to any claim or cause whatsoever whether such claim or cause arises in contract, tort or otherwise.

CHAPTER 18

java.util Part 2: More Utility Classes

This chapter continues our discussion of **java.util** by examining those classes and interfaces that are not part of the Collections Framework. These include classes that tokenize strings, work with dates, compute random numbers, bundle resources, and observe events. Also covered are the **Formatter** and **Scanner** classes which make it easy to write and read formatted data. Finally, the subpackages of **java.util** are briefly mentioned at the end of this chapter.

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 **StringTokenizer** class provides the first step in this parsing process, often called the *lexer* (lexical analyzer) or *scanner*. **StringTokenizer** implements 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 **StringTokenizer** constructors are shown here:

```
StringTokenizer(String str)
```

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

```
StringTokenizer(String str, String delimiters, boolean delimAsToken)
```

In all versions, *str* is 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 *delimAsToken* is **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 **StringTokenizer** object, the **nextToken()** method is used to extract consecutive tokens. The **hasMoreTokens()** method returns **true** while there are more tokens to be extracted. Since **StringTokenizer** implements **Enumeration**, the **hasMoreElements()** and **nextElement()** methods are also implemented, and they act the same as **hasMoreTokens()** and **nextToken()**, respectively. The **StringTokenizer** methods are shown in [Table 18-1](#).

Here is an example that creates a **StringTokenizer** to parse “key=value” pairs. Consecutive sets of “key=value” pairs are separated by a semicolon.

```
// Demonstrate StringTokenizer.
import java.util.StringTokenizer;

class STDemo {
    static String in = "title=Java: The Complete Reference;" +
        "author=Schildt;" +
        "publisher=McGraw-Hill;" +
        "copyright=2011";

    public static void main(String args[]) {
        StringTokenizer st = new StringTokenizer(in, "=");

        while(st.hasMoreTokens()) {
            String key = st.nextToken();
            String val = st.nextToken();
            System.out.println(key + "\t" + val);
        }
    }
}
```

The output from this program is shown here:

```
title  Java: The Complete Reference
author  Schildt
publisher  McGraw-Hill
copyright  2011
```

Method	Description
int countTokens()	Using the current set of delimiters, the method determines the number of tokens left to be parsed and returns the result.
boolean hasMoreElements()	Returns true if one or more tokens remain in the string and returns false if there are none.
boolean hasMoreTokens()	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 <i>delimiters</i>)	Returns the next token as a String and sets the delimiters string to that specified by <i>delimiters</i> .

Table 18-1 The Methods Defined by **StringTokenizer**

BitSet