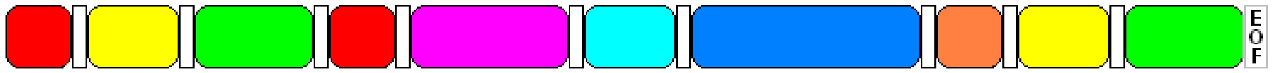
05.03 Virtual Lecture Notes: Reading Text Files



Using Scanner to Read Text Files

Reading information from text files is very simple, but Java is very picky, so follow the pattern and you won't have any trouble. If you doubt how picky Java can be, try to run this program without downloading the data1.txt file first.

Information from an input stream can be read sequentially from a text file, using familiar methods of the `Scanner` class. The important concept to envision is that a stream of tokens is being read in from a file with a **while** loop, which continues to iterate as long as there are more tokens to be read. (The `File` and `IOException` classes are also used, but they will not be discussed in detail at this time.)

A simple program to read a series of `String` tokens is shown below.

```
< 1>  import java.util.Scanner;
< 2>  import java.io.File;
< 3>  import java.io.IOException;
< 4>  public class ReadTextFile
< 5>  {
< 6>     public static void main(String[] args) throws IOException
< 7>     {
< 8>        String token = "";
< 9>        Scanner inFile = new Scanner(new File("data1.txt"));
<10>        while (inFile.hasNext())
<11>        {
<12>           token = inFile.next( );
<13>           System.out.println (token);
<14>        }
<15>        inFile.close();
<16>     }//end of main method
<17>  }//end of class
```

Run the program and observe the output. Then, study the following line-by-line explanation of the program.

Lines	
< 1>	imports the `Scanner` class from the `java.util` library, giving access to the input methods.
< 2>	imports the `File` class from the `java.io` library, which allows a `File` object to be created in Line <9>.
< 3>	imports the `IOException` class from the `java.io` library to handle errors reading files.

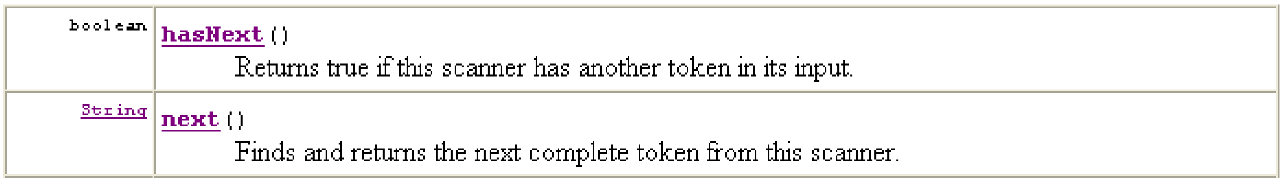
< 4> declares a class named **ReadDataScanner**.
< 5> opening curly brace marking the beginning of the class (matches up with line <16>).
< 6> the **main()** method where program execution begins. Notice that an **exception** has been added to the method header.
< 7> opening curly brace to start the **main()** method (matches up with line <14>).
< 8> declares and initializes a **String** object to receive the tokens from the file.
< 9> creates a **Scanner** object called **inFile,** which represents the text file (data1.txt) to be read in by the **while** loop.
<10> **while** loop condition that checks to see if there is another token to be read in. If the condition is **true**, the loop continues; if it is **false**, execution jumps to line <15>.
<11> curly brace marking the beginning of the **while** loop (matches up with line <14>).
<12> the next token in the file is read and assigned to the **token** variable.
<13> the individual token read in is printed.
<14> closing curly brace marking the end of the **while** loop (matches up with line <11>).
<15> ensures that the text file is closed after being used. This is very important!
<16> closing curly brace marking the end of the **main()** method (matches up with line <7>).
<17> closing curly brace marking the end of the class (matches up with line <5>).

Sometimes an error causes a program to crash. In a sense, Java takes "exception" with the code you have written and lets you know it! Exceptions will be thoroughly covered in future lessons.

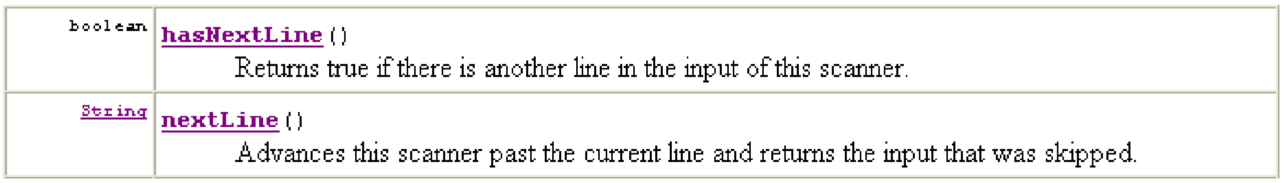
Modifications

Be sure that you downloaded the necessary text files before proceeding with the modifications to the **ReadTextFile** class.

1. Each of the following pairs of methods work in concert, to read in a stream of **String** data from a text file. Study the excerpts from the **Scanner** class API to gain an overview of how each of these methods performs.



boolean	hasNext() Returns true if this scanner has another token in its input.
String	next() Finds and returns the next complete token from this scanner.

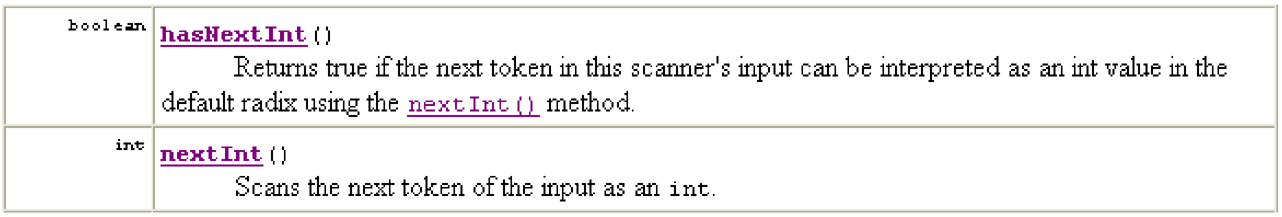


boolean	hasNextLine() Returns true if there is another line in the input of this scanner.
String	nextLine() Advances this scanner past the current line and returns the input that was skipped.

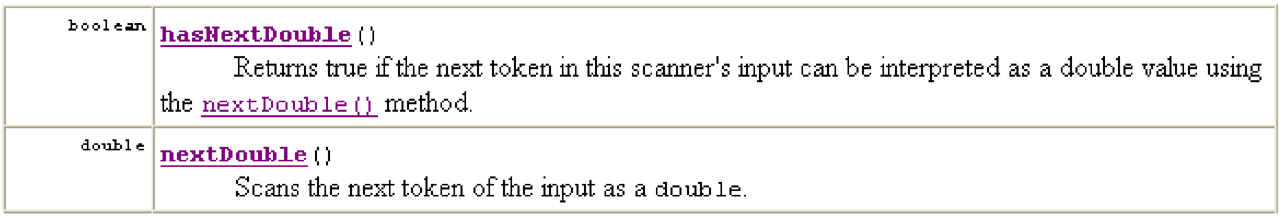
The **boolean** methods **hasNext()** and **hasNextLine()** are used in the **while** loop condition to determine when the loop should terminate. The **hasNext()** method deals with individual tokens delimited by white space, whereas the **hasNextLine()** method deals with entire lines of tokens separated by a carriage return. As long as there are more tokens to read from the file, these methods will be evaluated as **true** and more information will be read by the **next()** and **nextLine()** methods, respectively. When the end-of-file (EOF) marker is encountered, the file contains no more tokens and the condition will evaluate to **false,** terminating the loop.

- Modify the program to use the **hasNextLine()** and **nextLine()** methods.
- Examine the contents of the **data1.txt** file and the **data2.txt** file.
- Compare the performance of the **next()** and **nextLine()** when reading information from the **data1.txt** file and the **data2.txt** file.
- Compare the performance of these two methods when reading the **data6.txt** file. Look up the API Detail Table for these two sets of methods to understand why they produce different output for the same text file.
- Modify the program to read the **data3.txt** file. What happens if you add code to the program to add the numbers read in from the file?
- Modify the program to read the **data4.txt** and **data5.txt** files. What can you conclude about the versatility of the methods that read tokens as Strings?

2. Each of the following pairs of methods works together to read in a stream of numeric data from a text file. Carefully study the excerpts from the **Scanner** class API.



boolean	**hasNextInt**() Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the nextInt() method.
int	**nextInt**() Scans the next token of the input as an int.



boolean	**hasNextDouble**() Returns true if the next token in this scanner's input can be interpreted as a double value using the nextDouble() method.
double	**nextDouble**() Scans the next token of the input as a double.

The **hasNextInt()** and **hasNextDouble()** determine whether there are more numeric values of a specific type in the input stream from a file. If there are, the condition evaluates as **true** and the loop continues; however, if there are no more values to read from the file, the condition is **false** and the loop terminates. The **nextInt()** method and the **nextDouble()** method read integers and decimals, respectively.

- Modify the program to use the **hasNextInt()** and **nextInt()** methods.

- What happens when the **`data1`**, **`data2`**, and **`GettysburgAddress`** text files are read with integer methods?
- Change the text file to **`data3.txt`**. Are the data read correctly?
- Change the text file to **`data4.txt`**. Are the data read correctly?
- Change the text file to **`data5.txt`**. Are the data read correctly?

- Modify the program to use the **`hasNextDouble()`** and **`nextDouble()`** methods.
- What happens when the **`data1`**, **`data2`**, and **`data6`** text files are read with double methods?
- Change the text file to **`data3.txt`**. Are the data read correctly?
- Change the text file to **`data4.txt`**. Are the data read correctly?
- Change the text file to **`data5.txt`**. Are the data read correctly?

With these four sets of methods and **`while`** loops, you should be able to read any sequential file. However, if you forget to import the **`Scanner`**, **`File`** and **`IOException`** classes, or if you don't add **`throws IOException`** to the class header, Java will complain (doink). Follow the pattern for reading text files and you will be successful.

Creating Your Own Text Files

There are two basic ways to create text files: write a program to create a text file or use a simple text editor to type in the data you want. Soon you will learn how to write data to a file from a program.

If you are going to make your own text file by typing information from the keyboard, it is better to use something simple like Windows Notepad that is already configured to produce text files, rather than a full-featured word processor (e.g., Word). Use a simple text editor (e.g., Notepad) and create each of the following files.

- Type in the first seven prime numbers with each number separated by a single space. Press the Enter key at the end of the line. Save the file as **`data7.txt`**.
- Type in the first seven prime numbers with one number on each line. Press the Enter key after each line. Save the file as **`data8.txt`**.
- Type in 10 decimal numbers of your choice, separate each number by a single space. Press the Enter key at the end of the line. Save this file as **`data9.txt`**.
- Type in the same 10 decimal numbers, but this time each value should be on a separate line. Press the Enter key after typing each line. Save this file as **`data10.txt`**.
- Type in several sentences. Where you press the Enter key for the text in this file is up to you. Save the file as **`data11.txt`**.

After creating your own text files, try reading them with the appropriate pairs of **`Scanner`** methods.