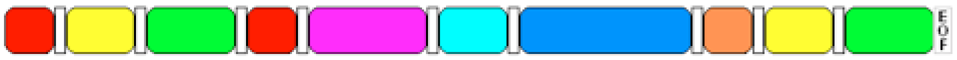
04.03 Virtual Lecture Notes: Reading Text Files



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:

```
<18>     import java.util.Scanner;
<10>    import java.io.File;
<11>    import java.io.IOException;
<12>    public class TextFileReaderV1
<13>    {
<14>      public static void main(String[] args)
throws IOException
<15>      {
<16>
<17>        String token = "";
<18>        File fileName = new File("data1.txt");
<19>        Scanner inFile = new Scanner(fileName);
<20>
<21>        //while loop will continue while inFile has a next token
to read
<22>        while( inFile.hasNext() )
<23>        {
<24>          token = inFile.next( );                //read next
token from file
<25>           System.out.println(token);             //print value
of token
<26>         }//end while
<27>
<28>         inFile.close();                            //close input
file
<29>
<30>       }//end of main method
<31>    }//end of class
```

Run the program and observe the output. Make sure the data1.txt file has been downloaded and saved to the project folder. Then, study the following line-by-line explanation of the program:

Lines

<9>	imports the **Scanner** class from the java.util library, giving access to the input methods.
<10>	imports the File class from the **java.io** library, which allows a **File** object to be created in Line <9>.
<11>	imports the **IOException** class from the **java.io** library to handle errors reading files.
<12>	declares a class named **TextFileReaderV1**.
<13>	opening curly brace marking the beginning of the class (matches up with line <31>).
<14>	the **main()** method where program execution begins. Notice that an exception has been added to the method header.
<15>	opening curly brace to start the **main()** method (matches up with line <30>).
<17>	declares and initializes a **String** object to receive the tokens from the file.
<18>	creates a **File** object called **fileName**, which represents the name of the text file.
<19>	creates a **Scanner** object called **inFile**, which represents the text file (data1.txt) to be read in by the while loop.

<22>	**while** loop condition that checks to see if there is another token to be read in from the file. If the condition is **true**, the loop continues; if it is **false**, the loop terminates and execution jumps to line <28>.
<23>	curly brace marking the beginning of the **while** loop (matches up with line <26>).
<24>	the next **token** in the file is read and assigned to the token variable.
<25>	the value of **token** is printed.
<26>	closing curly brace marking the end of the **while** loop (matches up with line <23>).
<28>	ensures that the text file is closed after being used. This is very important! If a file is opened, always be sure to close it.
<30>	closing curly brace marking the end of the **main()** method (matches up with line <15>).
<31>	closing curly brace marking the end of the class (matches up with line <13>).

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 **`TextFileReaderV1`** 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.

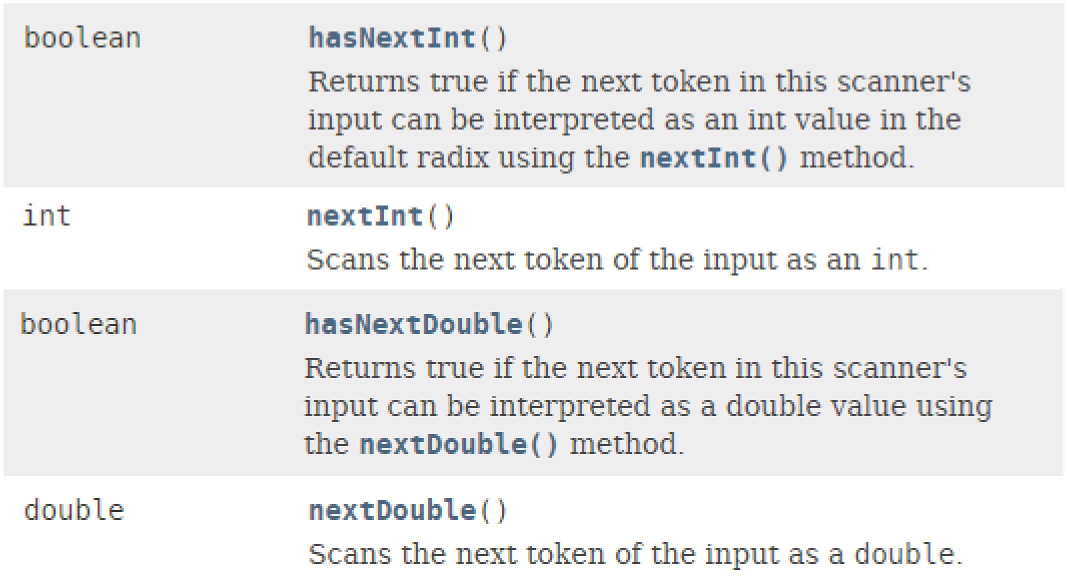
`boolean`	**`hasNext()`** Returns true if this scanner has another token in its input.
`String`	**`next()`** Finds and returns the next complete token from this scanner.
`boolean`	**`hasNextLine()`** Returns true if there is another line in the input of this scanner.
`String`	**`nextLine()`** Advances 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.

- Make a copy of the program and save it as **`TextFileReaderV2`**.
- 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.
- Modify the program to read the **`data5.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`** and **`data4.txt`** files. What happens if you add code to the program to add the numbers read in from the file? What can you conclude about the versatility of the methods that read tokens as Strings?

As you've seen demonstrated, the **`next()`** and **`nextLine()`** methods read data in as Strings either as one token at a time or an entire line at a time until the end of file is reached.

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()`** methods 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.

- Make a copy of the program and save it as **`TextFileReaderV3`**.
- Modify the program to use the **`hasNextInt()`** and **`nextInt()`** methods.
- Change the text file to **`data3.txt`**. Are the data read correctly?
- Add code to find the sum of the values read from the file.
- Try reading from the other data files. What happens when the other data files are read with integer methods?
- Make a copy of the program and save it as **`TextFileReaderV4`**.
- Modify the program to use the **`hasNextDouble()`** and **`nextDouble()`** methods.
- Change the text file to **`data4.txt`**. Are the data read correctly?
- Modify the program code to attempt to read from other data files. What happens when the other data files are attempted to be read when integer or double methods are used?

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 by giving you an error. 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. Either way, the file should not contain any extra white space after the last bit of data. If it does, errors will occur when reading in the file.

If you are going to make your own text file by typing information from the keyboard, it is better to use a simple text editor 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 (Windows) or TextEdit (Mac)) and create each of the following files:

- Type in the first seven prime numbers, with each number separated by a single space. 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 except the last one. Save the file as **`data8.txt`**.
- Type in 10 decimal numbers of your choice, separate each number by a single space. 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 except the last one. 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.

Print