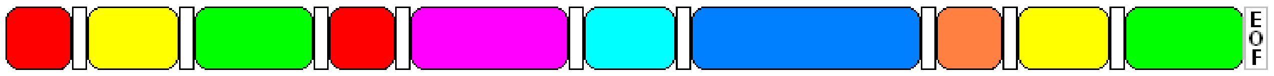
Virtual Lecture Notes: Writing Text Files



The output stream of all your previous programs has been directed to the monitor for display, but output can also be directed to the printer and to secondary storage devices like files. The important concept to envision is that a stream of tokens is being printed with a loop that continues to iterate as long as there are more tokens to print to the file. Whether you use a **`while`** loop or a **`for`** loop will depend on the context of the program. (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.io.IOException;
< 2> import java.io.PrintWriter;
< 3> import java.io.File;
< 4> class PrintWriterDemo_v1
< 5> {
< 6>    public static void main (String [ ] args) throws IOException
< 7>    {
< 8>
< 9>       PrintWriter outFile = new PrintWriter (new File("hello.txt"));
<10>
<11>       for (int loop = 1; loop <= 5; loop++)
<12>       {
<13>          outFile.println(loop + " Hello World");
<14>       }
<15>       outFile.close ( );
<16>    }
<17> }
```

This is an important demonstration program with many subtle features, so it is worth playing with for a while.

- Create a project folder called File Output in the Unit05 Lessons folder.
- Create a class called PrintWriterDemo_v1 in the newly created folder.
- Enter this program into BlueJ and compile the source code.
- Run the program and observe the output.

Actually, you cannot observe the output because the print statement in line <13> streams output to a file on the hard disk, not the screen. To retrieve the output of this program, you have to open the file with a simple text editor like Windows Notepad.

In ,ine <9>, hello.txt is a relative path name, so the file is created in the same folder as the PrintWriterDemo_v1 file. If you want to store a file somewhere else, use an absolute path name (e.g., c:\Java Projects\AnyOtherFolder\hello.txt).

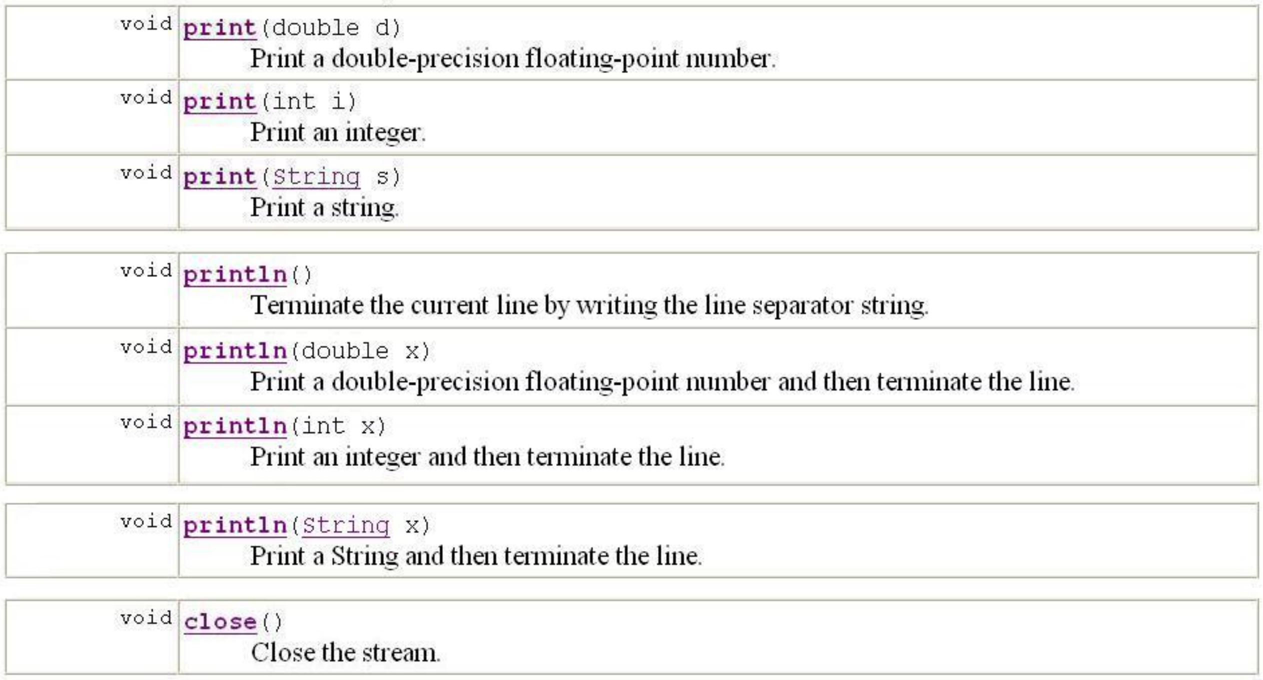
After running the program and observing the contents of the file saved to the disk, analyze the code line-by-line. This is another pattern that you can use over and over again with only minor variations, anytime you need to store information in a file.

Lines	
< 1>	imports the `IOException` class from the `java.io` library to handle errors reading files.
< 2>	imports the `PrinWriter` class from the `java.io` library which allows the `outFile` object to be created in line <9>.
< 3>	imports the `File` class from the `java.io` library, which allows a `File` object to be created in line <9> (within the parentheses).
< 4>	declares a class named `HelloWorldPrintWriter`.
< 5>	opening curly brace marking the beginning of the class (matches up with line <17>).
< 6>	the `main()` method where program execution begins. Notice that additional code has been added to the method header. (This will be covered in future lessons.)
< 7>	opening curly brace to start the `main()` method (matches up with line <16>).
< 8>	white space to improve program readability.
< 9>	creates a `PrintWriter` object called `outFile,` which represents the text file (hello.txt) to be written by the `for` loop.
<10>	white space to improve program readability.
<11>	`for` loop condition which counts from 1 to 5. When the condition is no longer `true`, execution jumps to line <15>.
<12>	curly brace marking the beginning of the `for` loop (matches up with line <14>).
<13>	the `println()` method of the `PrintWriter` class prints the `String` literal to the file represented by the `outFile` object. In this case, the value of the `loop` variable and the text within quotation marks were concatenated to form the `String` literal.
<14>	closing curly brace marking the end of the `for` loop (matches up with line <12>).
<15>	the `close()` method of the `PrintWriter` class closes the output stream once writing to the file is complete.
<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>).

Exceptions will be covered in a later module. Just be sure to import the class and modify line <6> until the full explanation can be given.

Background

Be sure to peruse the Java API for the `PrintWriter` class. There are many more methods than the ones shown in the following abbreviated Method Summary Table. All the print methods should look familiar to you from printing output to the screen; however, in this context, the `print()` and `println()` methods are streaming output to a file on the hard drive. You will discover many more examples of different classes using the same method name to perform similar tasks.



void	print(double d) Print a double-precision floating-point number.
void	print(int i) Print an integer.
void	print(String s) Print a string.

void	println() Terminate the current line by writing the line separator string.
void	println(double x) Print a double-precision floating-point number and then terminate the line.
void	println(int x) Print an integer and then terminate the line.

void	println(String x) Print a String and then terminate the line.

void	close() Close the stream.

The **`close()`** method is very important and must always be included after you finish writing to a file. You close the door to your house when everyone streams out to get in the car for a family outing, don't you? Well, close your files when your finish writing to them, too.

Modifications

It's time to experiment with writing files. Use the following suggestions to explore some of the possibilities and try out things you think of as well. To verify that the changes you make work, open the file in a text editor like Windows Notepad.

1. Add a print statement to stream the output to the screen as well as the disk file.
2. What happens if you delete "**`throws IOException`**" on line <6>?
3. Change the file name in line <9>.
4. Increase the number of loop iterations in line <11>.
5. Print something other than Hello World to the file.
6. Use the **`print()`** method instead of the **`println()`** method in line <13>.
7. Add code to make the program pick random numbers between 0 and 99. Change the name of the file. Print the random numbers to the file with each number on a different line. Print the numbers to the file with all numbers on one line. Which methods print ints to a file?
8. Modify the code to pick decimal numbers greater than 0.0 and less than or equal to 1000.0. Change the name of the output file. Print the random numbers to the file with each number on a different line. Print the numbers to the file with all numbers on one line. Which methods print doubles to a file?
9. What happens if you comment out line <15>? If nothing appears to happen, don't assume that there is no consequence to excluding this statement.
10. Continue to experiment on your own with how simple it is to write information to files.