

Special Topics No. 7: Buffered I/O for Higher Efficiency

In Chapter 11, we presented several different routines for file input and output. In this document, we will discuss efficient file I/O using buffered input and output. In addition, we will describe the differences between the character and byte streams. Many of the classes introduced in Chapter 11 have dealt with byte streams, and in this document, we will present the `FileReader` and `FileWriter` classes that deal with character streams.

Introduction

In Chapter 11, we mentioned the concept of *stream*, which represents a sequence, or a flow, of data. A reader is associated to an input stream, and a writer is associated to an output stream. To read from or write to a file, we attach some kind of a stream object to a file. The most basic stream is called a *byte stream* that is used for reading and writing unstructured sequences of bytes. We described in Chapter 11 different subclasses of `InputStream` and `OutputStream` that provide both low-level and high-level access. For the low-level access presented in Section 11.2, bytes are read from or written to byte streams. For the high-level access presented in 11.3, primitive data values are read from or written to byte streams after appropriate conversion between primitive data and bytes. In this document, we will describe a way to improve the performance of access byte streams by using a data buffer.

In Section 11.3, we also described different classes for accessing textfiles. We defined the *textfile* as a file whose contents are stored in the ASCII format. Some of the `InputStream` and `OutputStream` subclasses allow reading and writing textual data to textfiles, but they operate correctly only when dealing with Latin-1 characters (16-bit version of ASCII where the higher-order byte is zero). Newer classes `FileReader` and `FileWriter` allow reading and writing a sequence of Unicode character data to streams. The stream that deals correctly with Unicode characters is called a *character stream*. We will present in this document how to perform a buffered I/O on character streams. Using the `FileReader` and `FileWriter` classes, we can redefine the *textfile* as a file whose contents are stored in the some character-encoding format. (Note: There are many different types of character encoding. Latin-1, UTF-8, and UTF-16 are some of the more common encoding schemes. Try visiting non-English websites and select the encoding scheme menu choice to see the different types of encoding scheme available. With IE5.0, for example, you will see menu choice in the popup menu.)

1 BufferedInputStream and BufferedOutputStream

The following three statements are from the `TestDataOutputStream` class on page 532:

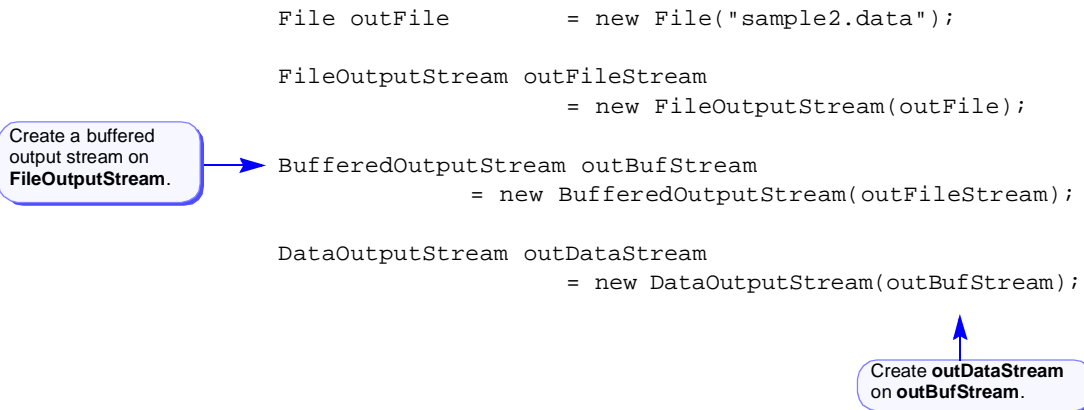
```
File outFile      = new File("sample2.data");

FileOutputStream outFileStream
    = new FileOutputStream(outFile);

DataOutputStream outDataStream
    = new DataOutputStream(outFileStream);
```

Using the `DataOutputStream`, it is possible that whenever we output primitive data values using the methods such as `writeInt`, `writeDouble`, and so forth, the underlying system is called to perform output operation to a device (for example, a hard disk where the file is stored). Since the physical device access is very costly, it is much better to save the output data in the internal buffer and let the actual physical device access occur only when the buffer is full. So, for example, instead of performing 100 physical device access to write 100 integers individually, it is much more efficient to save 100 integers in a buffer and perform one physical device access to store 100 integers at once. We can achieve such buffered output by using the `BufferedOutputStream` class.

To use a buffer output, we create a `BufferedOutputStream` object between the `FileOutputStream` and `DataOutputStream` objects as



```
File outFile          = new File("sample2.data");

FileOutputStream outFileStream
    = new FileOutputStream(outFile);

Create a buffered
output stream on
FileOutputStream. → BufferedOutputStream outBufStream
                    = new BufferedOutputStream(outFileStream);

DataOutputStream outDataStream
    = new DataOutputStream(outBufStream);

Create outDataStream
on outBufStream. ↑
```

Figure 1 shows the difference between the original version and the new buffered output version.

The default buffer size for a `BufferedOutputStream` object is 512 bytes. You can change the default to the size more suitable for your program but pass the size as the second argument to the constructor. For example, the statement

```
BufferedOutputStream outBufStream
    = new BufferedOutputStream(outFileStream, 2048);
```

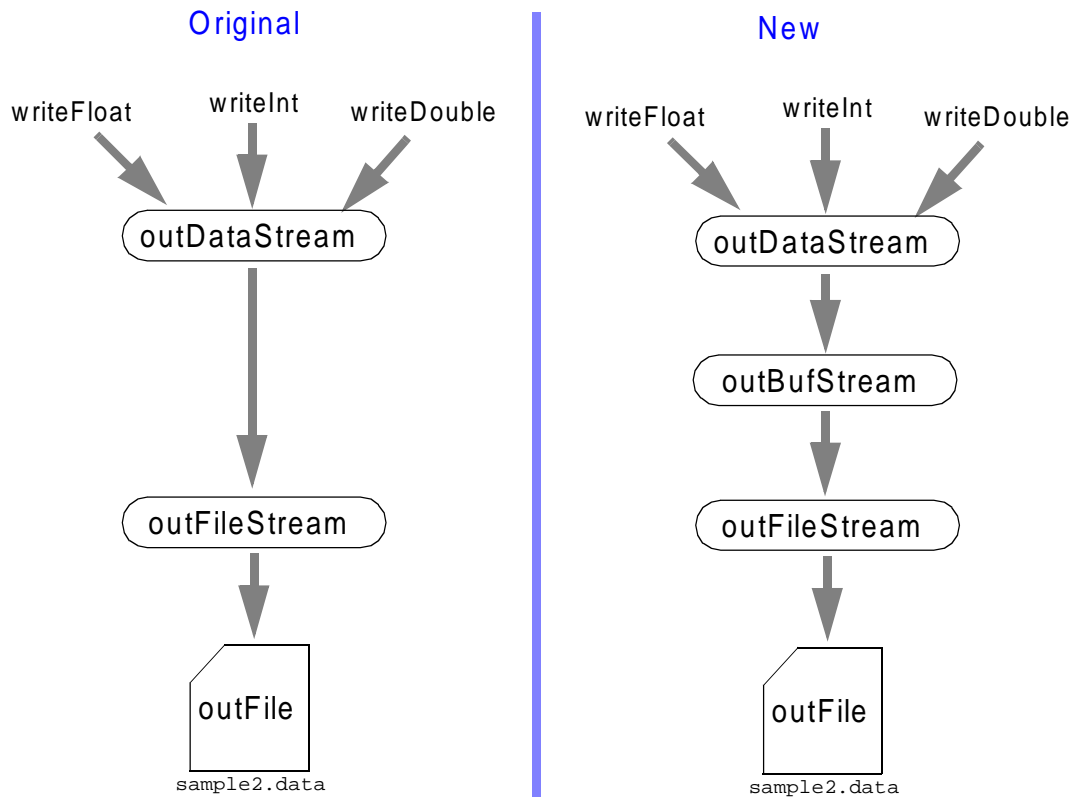
will create a 2K buffer (2048 bytes).

A buffered input works in the analogous manner. Here's what we did in the `TestDataInputStream` class on page 533:

```
File inFile          = new File("sample2.data");
```

FIGURE 1

The difference between the original version and the new buffered output version.



```
FileInputStream inFileStream
    = new FileInputStream(inFile);

DataInputStream inDataStream
    = new DataInputStream(inFileStream);
```

We change it to a buffered input by placing a `BufferedInputStream` object between the `FileInputStream` and `DataInputStream` objects as:

```
File inFile          = new File("sample2.data");

FileInputStream inFileStream
    = new FileInputStream(inFile);
```

Create a buffered input stream on **FileInputStream**.

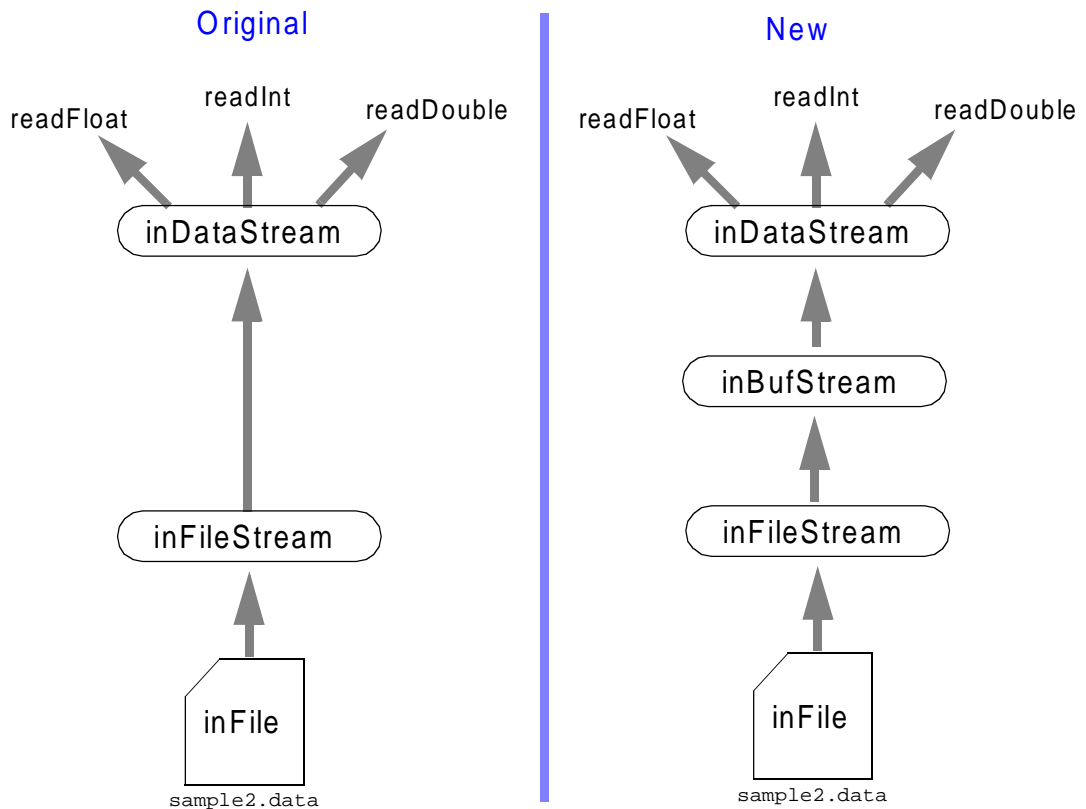
```
BufferedInputStream inBufStream
    = new BufferedInputStream(inFileStream);
DataInputStream inDataStream
    = new DataInputStream(inBufStream);
```

Create **inDataStream** on **inBufStream**.

Figure 2 shows the difference between the original version and the new buffered input version.

FIGURE 2

The difference between the original version and the new buffered input version.



Here's the key concept to remember:



For an improved performance, use `BufferedInputStream` and `BufferedOutputStream` to employ a data buffer with a programmer specifiable buffer size.

2 Buffered I/O for TextFiles

In Section 11.3, we gave sample programs to read from and write textual data to files. For the `TestBufferedReader` program, we used the `FileReader` and `BufferedReader` classes to perform a buffered input on a character stream. We will show you how to do the same for the output routine. In the sample program `TestPrintWriter` on page 536, we did not use the necessary classes to do a buffered output on a character stream. We were doing unbuffered output to a byte stream.

For character streams, we use some subclasses of `Reader` and `Writer` abstract classes. For example, we can use `FileReader` and `FileWriter` to read and write textfiles (files that contain a sequence of characters stored in some character-encoding format). Because at the bottom level, the physical devices use bytes, we need some classes that convert 16-bit characters to bytes correctly. Two such classes are `InputStreamReader` and `OutputStreamWriter`. `FileReader` and `FileWriter` are convenience subclasses of `InputStreamReader` and `OutputStreamWriter`, respectively, where the default character encoding and the default buffer size are used. To specify these values ourselves, we need to use an `InputStreamReader` or an `OutputStreamWriter`.

To use a buffered input or output on character streams, we use `BufferedReader` and `BufferedWriter`. Here's how we created necessary objects for a buffered input from a character stream in the `TestBufferedReader` program on page 537 as

```
File          inFile      = new File("sample3.data");
FileReader    fileReader  = new FileReader(inFile);
BufferedReader bufReader  = new BufferedReader
                               (fileReader);
```

Comparable objects for a buffered output to a character stream are created as

```
File          outFile     = new File("sample3.data");
FileWriter    fileWriter  = new FileWriter(outFile);
BufferedWriter bufWriter  = new BufferedWriter
                               (fileWriter);
PrintWriter   outputStream = new PrintWriter(bufWriter);
```

The actual methods for reading and writing data using `BufferedReader` and `PrintWriter` were already discussed in Chapter 11.