CSCI 2120:
Software Design & Development II

UNIT3: I/O management

io api

BufferedOutputStream

Overview

Introduction

- A **BufferedOutputStream in Java** is a concrete subclass of `FilterOutputStream` class.
- It wraps (buffers) an output stream into a buffered stream and makes write operations on the stream more efficient and fast.
- Java `BufferedOutputStream` adds a buffer to an output stream that stores individual data (in bytes) temporarily into a memory buffer.
- It is used to speed up the output by reducing the number of disk or file writes by adding an additional layer of functionality around the underlying stream.

Introduction

- For example, suppose we write data into a file using `FileOutputStream` as: `fos.write(ch)`;
- Here, we invoke `write()` method to write a character (`ch`) into a file. Then, the operating system writes character `ch` into the file.
- Now suppose we need to write 50 characters into a file. The underlying operating system would be called 50 times to write the 50 characters. It will waste a lot of time.
- But if we buffer output to a file by wrapping a `FileOutputStream` within a `BufferedOutputStream`, it will first write characters into a temporary block of memory called a buffer.
- And then all the characters from the buffer will be at once written into a file by the operating system. Thus, the buffered output stream will take less time to make write operations fast.

Inner Workings of BufferedOutputStream

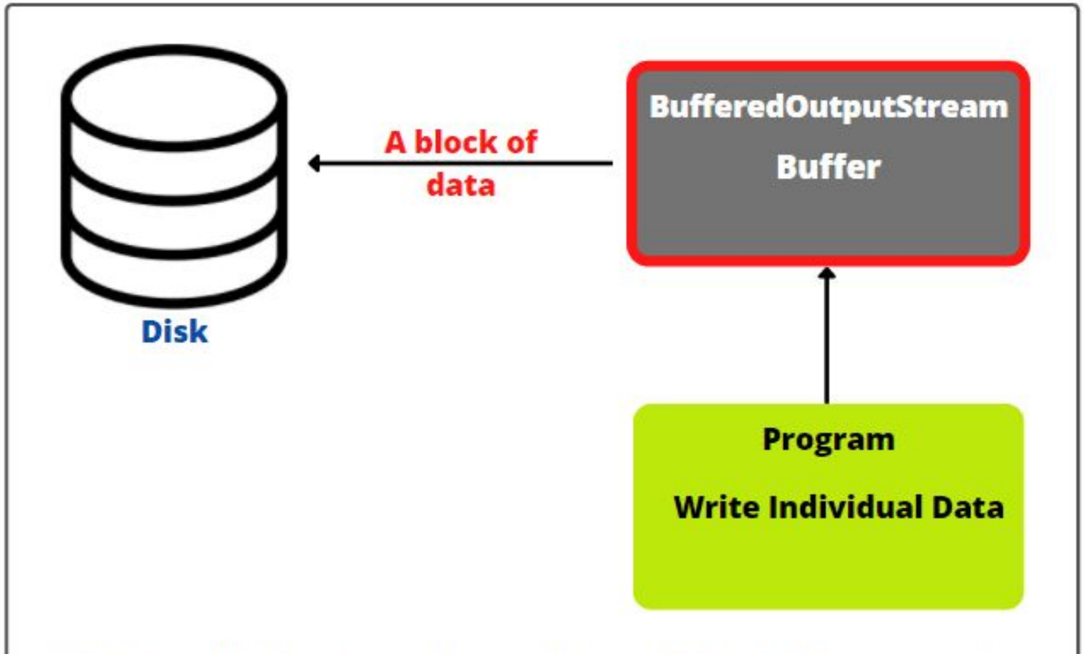

Fig: BufferedOutputStream places data in a buffer for fast processing.

`BufferedOutputStream` is a buffer internally between the program and disk (destination). During the write operation, the individual data are first stored temporarily in the memory buffer.

When the memory buffer is full, all data from the buffer are written to the disk once.

A buffered output stream adds more efficiency than writing data directly into a stream and makes the performance fast. This is because the number of times accessing the disk is reduced.

BufferedOutputStream class declaration

`BufferedOutputStream` class is derived from class `FilterOutputStream`, which has `OutputStream` as a base class. It implements `Closeable`, `Flushable`, and `AutoCloseable` interfaces.

The general declaration of `BufferedOutputStream` class is as follows:

```
public class BufferedOutputStream
       extends FilterOutputStream
    implements Closeable, Flushable, AutoCloseable
```

`BufferedOutputStream` was added in Java 1.0 version. Its in `java.io.BufferedOutputStream` package.

BufferedOutputStream Constructors

To wrap `OutputStream`, `BufferedInputStream` class provides two constructors that are as follows:

1. BufferedOutputStream(OutputStream outputStream)

This constructor creates the new buffered output stream object that buffers an output stream specified by `outputStream`. It uses the default buffer size for writing the data to the specified output stream.

The general syntax to wrap `FileOutputStream` into `BufferedOutputStream` is as follows:

```
// Creates an instance of FileOutputStream with specified file path.
FileOutputStream fos = new FileOutputStream(String path);

// Creates a BufferedOutputStream object and passes fos to its constructor.
// Wrapping the file stream in a BufferedInputStream.
BufferedOutputStream buffer = new BufferedOutputStream(fos);
```

BufferedOutputStream Constructors

To wrap `OutputStream`, `BufferedInputStream` class provides two constructors that are as follows:

2. BufferedOutputStream(OutputStream outputStream, int size)

This constructor creates a new buffered output stream object that buffers an output stream with specified buffer size.

The general syntax to wrap `FileOutputStream` into the buffered stream is as follows:

```
// Creates a BufferedOutputStream object with specified size of internal buffer.
BufferedOutputStream buffer = new BufferInputStream(fos, int size);

//For example:
// Here, buffer size is defined as 1024 bytes
BufferedOutputStream buffer = new BufferedOutputStream(fos, 1024);
```

BufferedOutputStream Methods

`BufferedOutputStream` class in Java does not define any new methods. All the methods in `BufferedOutputStream` are inherited from the `OutputStream` class.

Some important methods are as follows:

BufferedOutputStream Methods

Method	Description
void write(int b)	This method writes the specified byte to the buffered output stream.
void write(byte[ ] b, int n, int m)	This method writes the bytes from the specified byte-input stream m into a specified byte array, starting from the given nth byte.
void flush()	This method flushes the buffered output stream. It can be used to clear the internal buffer by forcing the output stream to write all data present in the buffer to the destination file.
void close()	This method closes the buffered output stream. Once the method is invoked, we cannot write the data again.

BufferedOutputStream Methods - Checked Exceptions

Almost all the methods in the I/O stream classes throw an exception named `IOException`. This exception is thrown when an Input/Output operation fails because of an interrupted call.

Therefore, we need to declare to throw `java.io.IOException` in the method or put the code in a `try-catch` block, as shown below:

```
//Declaring IOException exception in the method
public static void main(String[] args) throws IOException {
   // Perform I/O operations.
}
//or, Using try-catch block
public static void main(String[] args) {
   try {
       // Perform I/O operations
   }
   catch (IOException ex) {
       ex.printStackTrace();
   }
}
```

Example 1: Write data from file

1. Let’s take an example program where we will improve the efficiency of writing data into a file using buffered output stream. In this program, we will write the textual information in the file using `BufferedOutputStream` object.

Example 1: Write data from file

```java
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
public class BufferedOutputStreamTester1 {
   public static void main(String[] args) {
       try {
           // Create a FileOutputStream object to connect out_buff1.
           FileOutputStream fos = new FileOutputStream("./src/out_buff1.txt");

           // Create a BufferedOutputStream object to wrap FileOutputStream.
           BufferedOutputStream bos = new BufferedOutputStream(fos);

           String s="Welcome to UNO Computer Science";
           byte b[]=s.getBytes(); // Converting String into array bytes.
           bos.write(b); // Write data to the output stream.
           bos.close(); // Closing output stream.

           System.out.println("Successfully written...");
       }
       catch(Exception e) {
           System.out.println(e);
       }
   }
}
```

Example 1: Write data from file

Output:

```
Successfully written...
```

In this example program,

the `out_buff1.txt` file contains the text: `“Welcome to UNO Computer Science”.`

Example 2: Flush buffer to file

2. Let's create a program to `flush` all data present in the internal buffer to the file. Look at the source code below.

Example 2: Flush buffer to file

```java
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class BufferedOutputStreamTester2 {
   public static void main(String[] args) throws IOException {
       String data = "Hello Java";
       // Create a FileOutputStream object to connect out_buff2.txt.
       FileOutputStream fos = new FileOutputStream("./src/out_buff2.txt");

       // Create a BufferedOutputStream object to wrap FileOutputStream in BufferedOutputStream.
       BufferedOutputStream bos = new BufferedOutputStream(fos, 1024);

       bos.write(data.getBytes()); // Writing data to output stream.
       bos.flush(); // Flushing all data from internal buffer to destination file.
       bos.close();
       System.out.println("Successfully written...");
   }
}
```

Example 2: Flush buffer to file

Output:

```
Successfully written...
```

In this example program,

the `out_buff2.txt` file contains the text: `“Hello Java”.`

END