CSCI 2120:
Software Design & Development II

UNIT3: I/O management

io api

BufferedInputStream

Overview

1. Introduction
2. Inner Workings of BufferedInputStream
3. BufferedInputStream class declaration
4. BufferedInputStream constructors
5. BufferedInputStream methods
6. BufferedInputStream Examples

Introduction

- A **BufferedInputStream in Java** is a concrete subclass of `FilterInputStream` that wraps (buffers) an input stream into a buffered stream and makes read operations on the stream more efficient and fast.

- In simple words, it adds buffering capabilities to an input stream that stores data (in bytes) temporarily into a memory buffer by reading from the stream.

- `BufferedInputStream` is used to speed up the input by reducing the number of disk or file reads by adding an additional layer of functionality around the underlying stream.

Introduction

- For example, `FileInputStream` and `FileOutputStream` in Java are unbuffered, meaning that each read and write request is handled directly by the operating system.

- It makes the program less efficient because for every read or write request, they will access a disk or file. It will waste a lot of time.

- On the other hand, if we buffer input data from a file by wrapping a `FileInputStream` (for instance) into a `BufferedInputStream`, the buffered stream will store data into a temporary block of buffered memory.

- And then data is sent individually to the program from the buffer. Thus, buffered input stream in Java makes file operations more efficient and fast.

Inner Workings of BufferedInputStream

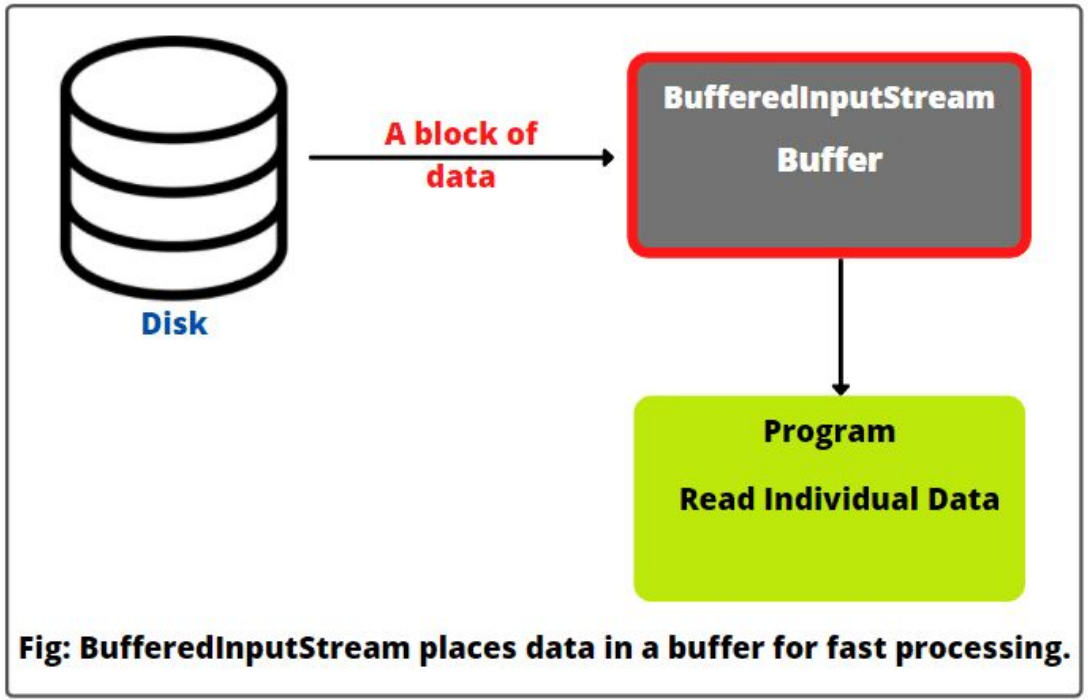

Fig: BufferedInputStream places data in a buffer for fast processing.

`BufferedInputStream` is a buffer internally between the program and the source. During the read operation, the whole block of data (in bytes) is read from the disk and temporarily stored into the internal buffer in the memory once.

The data are then transferred (read) individually to the program from the buffer, as shown in Figure.

Inner Workings of BufferedInputStream

The key points about `BufferedInputStream` in Java are:

- When the input data (in bytes) from the stream are skipped or read, the buffered stream automatically refilled many data from the input stream at a time.
- When a `BufferedInputStream` object is created, a buffer array is created internally.
- Buffers can be constructed by using `BufferedInputStream` and `BufferedOutputStream` classes.

BufferedInputStream class declaration

`BufferedInputStream` class is derived from class `FilterInputStream,` which has `InputStream` as a base class. It implements `Closeable` and `AutoCloseable` interfaces.

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

```
public class BufferedInputStream
       extends FilterInputStream
       implements Closeable, AutoCloseable
```

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

BufferedInputStream Constructors

To wrap InputStream, BufferedInputStream class defines two constructors that are as follows:

1. BufferedInputStream(InputStream inputStream)

This constructor creates a BufferedInputStream object that buffers an input stream specified by inputStream. It uses default buffer size. The default internal buffer size is 8192 bytes.

The general syntax to wrap FileInputStream into BufferedInputStream is as follows:

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

// Creates a BufferedInputStream object and passes fis into its constructor.
// Wrapping the file stream in a BufferedInputStream.
BufferedInputStream buffer = new BufferInputStream(fis);
```

BufferedInputStream Constructors

To wrap InputStream, BufferedInputStream class defines two constructors that are as follows:

2. BufferedInputStream(InputStream inputStream, int size)

This constructor creates a BufferedInputStream object that buffers an input stream with a specified buffer size.

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

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

BufferedInputStream Constructors

Now, perform all I/O operations through the buffered stream. At last, close the buffered stream by calling the `close()` method.

Closing the buffered stream automatically closes the underlying file stream. An exception named `IOException` will be thrown if an error occurs.

BufferedInputStream Methods

`BufferedInputStream` class does not define any new methods. All the methods in `BufferedInputStream` are inherited from the `InputStream` class. Some important methods are as follows:

BufferedInputStream Methods

Method	Description
int available()	This method returns the number of available bytes from the input stream without being blocked.
int read()	This method reads the next byte of data from the buffered input stream.
read(byte[ ] arr)	It reads bytes from the buffered input stream and stores them in the specified array.
int read(byte[ ] b, int n, int m)	It reads up to m bytes of data from this input stream and stores it into an array of bytes starting from the nth byte.

BufferedInputStream Methods

Method	Description
void close()	This method closes the buffered input stream and releases any of the system resources associated with the stream.
void reset()	It repositions the buffered input stream to the last marked position.
long skip(long x)	This method skips over and discards x bytes of data from the buffered input stream.
void mark(int readlimit)	This method marks the current position in the buffered input stream.
boolean markSupported()	It tests whether the buffered input stream supports the mark and reset methods.

BufferedInputStream 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: Read data from file

1. Let’s take a simple example program to read data or text from a file `myfile.txt` using `BufferedInputStream`. Look at the source code to understand better.

Example 1: Read data from file

```
import java.io.BufferedInputStream;
import java.io.FileInputStream;
public class BufferedInputStreamTester1 {
   public static void main(String[] args) {
       try {
           // Create a FileInputStream object to attach myfile to FileInputStream.
           FileInputStream fis = new FileInputStream("./src/myfile.txt");

           // Create a BufferedInputStream object to wrap FileInputStream.
           BufferedInputStream bis = new BufferedInputStream(fis);

           int i = 0;
           while ((i = bis.read()) != -1) {
               char ch = (char) i;
               System.out.print(ch);
           }
           bis.close();
           fis.close();
       }
       catch(Exception e) {
           System.out.println(e);
       }
   }
}
```

Example 1: Read data from file

Output:

```
Welcome to UNO Computer Science
```

In this example program,

1. We have created a buffered input stream named `bis` and connected it to `FileInputStream fis`.

2. Then, we have used a while loop and `read()` method to read all bytes from the internal buffer and display them on the console.

Here, we are assuming that you have the following data in "`myfile.txt`" file: Welcome to UNO Computer Science.

Example 2: Get number of available bytes

2. Let’s create a Java program to get the number of available bytes in the input stream. For this purpose, we will use `available()` method.

Example 2: Get number of available bytes

```
import java.io.BufferedInputStream;
import java.io.FileInputStream;
public class BufferedInputStreamTester2 {
   public static void main(String[] args) {
       try {
           // Create a FileInputStream object to attach myfile.txt
           FileInputStream fis = new FileInputStream("./src/myfile.txt");

           // Create a BufferedInputStream object to wrap FileInputStream.
           BufferedInputStream bis = new BufferedInputStream(fis);

           // Call available() method to get the available number of bytes in bufferedInputStream.
           System.out.println("Available bytes at the beginning: " + bis.available());

           // Reads bytes from the file
           bis.read();
           bis.read();
           bis.read();

       // Get the available number of bytes at the end.
           System.out.println("Available bytes at the end: " + bis.available());
           bis.close();
       }
       catch(Exception e) {
           System.out.println(e);
       }
   }
}
```

Example 2: Get number of available bytes

Output:

```
Available bytes at the beginning: 32
Available bytes at the end: 29
```

In this example program,

1. We first use the `available()` method to check the number of available bytes in the buffered input stream.

2. Then, we have used the `read()` method 3 times to read 3 bytes from the buffered input stream.

3. After reading three bytes, we again have checked the available number of bytes in the input stream. This time the available number of bytes is 29.

Example 3: Skip bytes

3. Let’s create a Java program to discard and skip the specified number of bytes from the input stream. For this purpose, we will use `skip()` method.

Example 3: Skip bytes

```
import java.io.BufferedInputStream;
import java.io.FileInputStream;
public class BufferedInputStreamTester3 {
   public static void main(String[] args) {
       try {
           // Create a FileInputStream object to attach myfile.txt
           FileInputStream fis = new FileInputStream("./src/myfile.txt");

           // Create a BufferedInputStream object to wrap FileInputStream.
           BufferedInputStream bis = new BufferedInputStream(fis);

           // Skips 5 bytes from the buffered input stream.
           bis.skip(5);
           System.out.println("Input stream after skipping first 5 bytes:");

           // Reads all available bytes from buffered input stream after skipping.
           int i = 0;
           while ((i = bis.read()) != -1) {
               System.out.print((char) i);
           }
           bis.close();
       }
       catch(Exception e) {
           System.out.println(e);
       }
   }
}
```

Example 3: Skip bytes

Output:

```
Input stream after skipping first 5 bytes:
me to UNO Computer Science
```

In this example program,

we have used `skip()` method for skipping the first 5 bytes from the buffered input stream and display the rest on the console. The skipping bytes are `‘W’, ‘e’, ‘l’, ‘c’, ‘o’`.

END