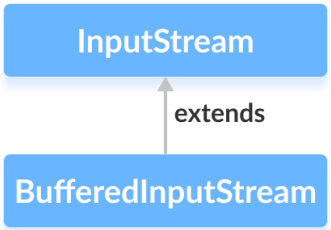
Java BufferedInputStream Class

In this tutorial, we will learn about Java BufferedInputStream and its methods with the help of examples.

The `BufferedInputStream` class of the `java.io` package is used with other input streams to read the data (in bytes) more efficiently.

It extends the `InputStream` abstract class.



Working of BufferedInputStream

The `BufferedInputStream` maintains an internal **buffer of 8192 bytes**.

During the read operation in `BufferedInputStream`, a chunk of bytes is read from the disk and stored in the internal buffer. And from the internal buffer bytes are read individually.

Hence, the number of communication to the disk is reduced. This is why reading bytes is faster using the `BufferedInputStream`.

Create a BufferedInputStream

In order to create a `BufferedInputStream`, we must import the `java.io.BufferedInputStream` package first. Once we import the package here is how we can create the input stream.

```
// Creates a FileInputStream
FileInputStream file = new FileInputStream(String path);

// Creates a BufferedInputStream
BufferedInputStream buffer = new BufferInputStream(file);
```

In the above example, we have created a `BufferdInputStream` named `buffer` with the `FileInputStream` named `file`.

Here, the internal buffer has the default size of 8192 bytes. However, we can specify the size of the internal buffer as well.

```
// Creates a BufferedInputStream with specified size internal buffer
BufferedInputStream buffer = new BufferInputStream(file, int size);
```

The `buffer` will help to read bytes from the files more quickly.

Methods of BufferedInputStream

The `BufferedInputStream` class provides implementations for different methods present in the `InputStream` class.

read() Method

- `read()` - reads a single byte from the input stream
- `read(byte[] arr)` - reads bytes from the stream and stores in the specified array

- `read(byte[] arr, int start, int length)` - reads the number of bytes equal to the `length` from the stream and stores in the specified array starting from the position `start`

Suppose we have a file named **input.txt** with the following content.

```
This is a line of text inside the file.
```

Let's try to read the file using `BufferedInputStream`.

```java
import java.io.BufferedInputStream;
import java.io.FileInputStream;

class Main {
    public static void main(String[] args) {
        try {

            // Creates a FileInputStream
            FileInputStream file = new FileInputStream("input.txt");

            // Creates a BufferedInputStream
            BufferedInputStream input = new BufferedInputStream(file);

            // Reads first byte from file
            int i = input .read();

            while (i != -1) {
                System.out.print((char) i);

                // Reads next byte from the file
                i = input.read();
            }
            input.close();
        }

        catch (Exception e) {
            e.getStackTrace();
        }
    }
```

Output

```
This is a line of text inside the file.
```

In the above example, we have created a buffered input stream named `buffer` along with `FileInputStream`. The input stream is linked with the file **input.txt**.

```
FileInputStream file = new FileInputStream("input.txt");
BufferedInputStream buffer = new BufferedInputStream(file);
```

Here, we have used the `read()` method to read an array of bytes from the internal buffer of the buffered reader.

---

available() Method

To get the number of available bytes in the input stream, we can use the `available()` method. For example,

```
import java.io.FileInputStream;
import java.io.BufferedInputStream;

public class Main {

   public static void main(String args[]) {

      try {

         // Suppose, the input.txt file contains the following text
         // This is a line of text inside the file.
         FileInputStream file = new FileInputStream("input.txt");
```

```
        // Creates a BufferedInputStream
        BufferedInputStream buffer = new BufferedInputStream(file);

        // Returns the available number of bytes
        System.out.println("Available bytes at the beginning: " + buffer.available());

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

        // Returns the available number of bytes
        System.out.println("Available bytes at the end: " + buffer.available());

        buffer.close();
    }
```

Output

```
Available bytes at the beginning: 39
Available bytes at the end: 36
```

In the above example,

1. We first use the `available()` method to check the number of available bytes in the input stream.
2. Then, we have used the `read()` method 3 times to read 3 bytes from the input stream.
3. Now, after reading the bytes we again have checked the available bytes. This time the available bytes decreased by 3.

skip() Method

To discard and skip the specified number of bytes, we can use the `skip()` method. For example,

```
import java.io.FileInputStream;
import java.io.BufferedInputStream;

public class Main {

   public static void main(String args[]) {

      try {
         // Suppose, the input.txt file contains the following text
         // This is a line of text inside the file.
         FileInputStream file = new FileInputStream("input.txt");

         // Creates a BufferedInputStream
         BufferedInputStream buffer = new BufferedInputStream(file);

         // Skips the 5 bytes
         buffer.skip(5);
         System.out.println("Input stream after skipping 5 bytes:");

         // Reads the first byte from input stream
         int i = buffer.read();
         while (i != -1) {
            System.out.print((char) i);

            // Reads next byte from the input stream
            i = buffer.read();
         }

         // Closes the input stream
```

Output

```
Input stream after skipping 5 bytes: is a line of text inside the file.
```

In the above example, we have used the `skip()` method to skip 5 bytes from the file input stream. Hence, the bytes `'T'`, `'h'`, `'i'`, `'s'` and `' '` are skipped from the input stream.

---

close() Method

To close the buffered input stream, we can use the `close()` method. Once the `close()` method is called, we cannot use the input stream to read the data.

Other Methods Of BufferedInputStream

Methods	Descriptions
`mark()`	mark the position in input stream up to which data has been read
`reset()`	returns the control to the point in the input stream where the mark was set