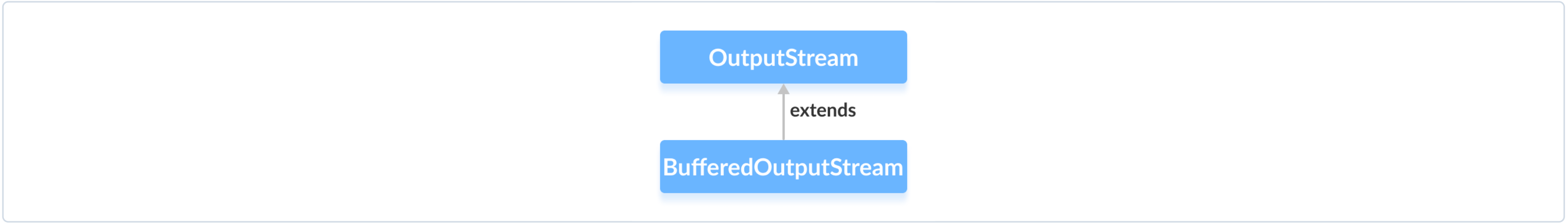
Java BufferedOutputStream Class

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

The `BufferedOutputStream` class of the `java.io` package is used with other output streams to write the data (in bytes) more efficiently.

It extends the `OutputStream` abstract class.



Working of BufferedOutputStream

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

During the write operation, the bytes are written to the internal buffer instead of the disk. Once the buffer is filled or the stream is closed, the whole buffer is written to the disk.

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

Create a BufferedOutputStream

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

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

// Creates a BufferedOutputStream
BufferedOutputStream buffer = new BufferOutputStream(file);
```

In the above example, we have created a `BufferdOutputStream` named `buffer` with the `FileOutputStream` 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 BufferedOutputStream with specified size internal buffer
BufferedOutputStream buffer = new BufferOutputStream(file, int size);
```

The `buffer` will help to write bytes to files more quickly.

Methods of BufferedOutputStream

The `BufferedOutputStream` class provides implementations for different methods in the `OutputStream` class.

write() Method

- `write()` - writes a single byte to the internal buffer of the output stream
- `write(byte[] array)` - writes the bytes from the specified array to the output stream
- `write(byte[] arr, int start, int length)` - writes the number of bytes equal to `length` to the output stream from an array starting from the position `start`

Example: BufferedOutputStream to write data to a File

```java
import java.io.FileOutputStream;
import java.io.BufferedOutputStream;

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

        String data = "This is a line of text inside the file";

        try {
            // Creates a FileOutputStream
            FileOutputStream file = new FileOutputStream("output.txt");

            // Creates a BufferedOutputStream
            BufferedOutputStream output = new BufferedOutputStream(file);

            byte[] array = data.getBytes();

            // Writes data to the output stream
            output.write(array);
            output.close();
        }

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

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

```
FileOutputStream file = new FileOutputStream("output.txt");
BufferedOutputStream output = new BufferedOutputStream(file);
```

To write data to the file, we have used the `write()` method.

Here when we run the program, the **output.txt** file is filled with the following content.

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

> **Note**: The `getBytes()` method used in the program converts a string into an array of bytes.

---

flush() Method

To clear the internal buffer, we can use the `flush()` method. This method forces the output stream to write all data present in the buffer to the destination file. For example,

```java
import java.io.FileOutputStream;
import java.io.BufferedOutputStream;

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

        String data = "This is a demo of the flush method";

        try {
            // Creates a FileOutputStream
            FileOutputStream file = new FileOutputStream(" flush.txt");

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

            // Writes data to the output stream
            buffer.write(data.getBytes());

            // Flushes data to the destination
            buffer.flush();
            System.out.println("Data is flushed to the file.");
            buffer.close();
        }

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

Output

```
Data is flushed to the file.
```

When we run the program, the file **flush.txt** is filled with the text represented by the string `data`.

---

close() Method

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