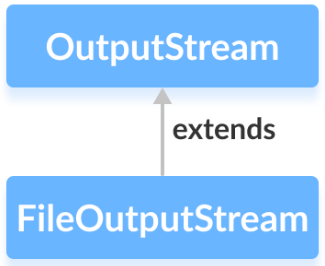
Java FileOutputStream Class

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

The `FileOutputStream` class of the `java.io` package can be used to write data (in bytes) to the files.

It extends the `OutputStream` abstract class.



Before you learn about `FileOutputStream`, make sure to know about Java Files.

Create a FileOutputStream

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

1. Using the path to file

```
// Including the boolean parameter
FileOutputStream output = new FileOutputStream(String path, boolean value);

// Not including the boolean parameter
FileOutputStream output = new FileOutputStream(String path);
```

Here, we have created an output stream that will be linked to the file specified by the `path`.

Also, `value` is an optional boolean parameter. If it is set to `true`, the new data will be appended to the end of the existing data in the file. Otherwise, the new data overwrites the existing data in the file.

2. Using an object of the file

```
FileOutputStream output = new FileOutputStream(File fileObject);
```

Here, we have created an output stream that will be linked to the file specified by `fileObject`.

Methods of FileOutputStream

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

write() Method

- `write()` - writes the single `byte` to the file output stream
- `write(byte[] array)` - writes the bytes from the specified array to the output stream
- `write(byte[] array, 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: FileOutputStream to write data to a File

```java
import java.io.FileOutputStream;

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

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

        try {
            FileOutputStream output = new FileOutputStream("output.txt");

            byte[] array = data.getBytes();

            // Writes byte to the file
            output.write(array);

            output.close();
        }

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

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

```java
FileOutputStream output = new FileOutputStream("output.txt");
```

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 output stream, we can use the `flush()` method. This method forces the output stream to write all data to the destination. For example,

```
import java.io.FileOutputStream;
import java.io.IOException;

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

        FileOutputStream out = null;
        String data = "This is demo of flush method";

        try {
            out = new FileOutputStream(" flush.txt");

            // Using write() method
            out.write(data.getBytes());

            // Using the flush() method
            out.flush();
            out.close();
        }
        catch(Exception e) {
            e.getStackTrace();
        }
    }
}
```

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

---

close() Method

To close the file output stream, we can use the `close()` method. Once the method is called, we cannot use the methods of `FileOutputStream`.

Other Methods Of FileOutputStream

Methods	Descriptions
`finalize()`	ensures that the `close()` method is called
`getChannel()`	returns the object of `FileChannel` associated with the output stream
`getFD()`	returns the file descriptor associated with the output stream