Writing Characters

José Paumard
PHD, JAVA CHAMPION, JAVA ROCK STAR

@JosePaumard https://github.com/JosePaumard

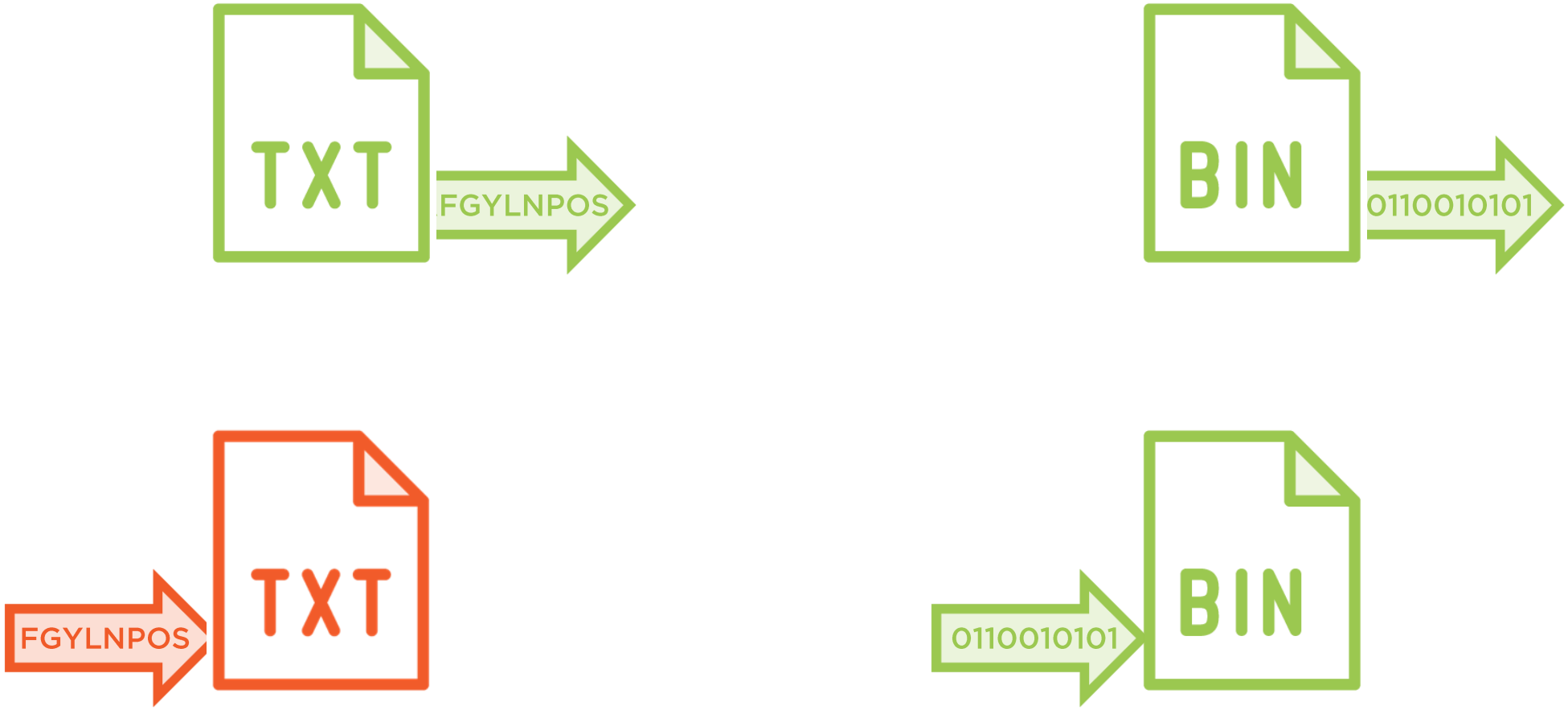
Agenda

Concept of Writer

How to write characters to files

Then to in-memory arrays

Introducing Writers


TXT
FGYLNPOS
BIN
0110010101
FGYLNPOS
TXT
0110010101
BIN

The Writer Abstract Class

The Writer is an abstract class

It defines the basic operations:

- Write of a single character
- Write of an array of characters
- Write a String
- Append a single char or a string

And it can be closed

```
Writer writer = ...; // we will see how to create one later

writer.write('H');

writer.write("Hello world!");
```

The write method does not return anything

```
Writer writer = ...; // we will see how to create one later

writer.write('H');

writer.write("Hello world!", 0, 5);
```

The write method does not return anything

This second call only writes `Hello`

```
Writer writer = ...; // we will see how to create one later

String hello = "Hello world!";

writer.write(hello.toCharArray(), 0, 5);
```

The write method does not return anything

This second call only writes `Hello`

You can also write from an array of chars

Dealing with Exceptions

I/O Operations Will Throw Exceptions

All these methods declare checked exceptions, just as the readers do

The patterns to deal with them are the same

Including the try-with-resources pattern to open a reader or a writer

```
try (Writer writer = ...;) {

    writer.write("Hello world!");

} catch (IOException e) {
    // deal with the exception
}
```

Creating Writers

As the Reader class, the Writer is an abstract class

Extended by 2 categories of concrete classes

1) classes for a certain type of output

- Disk: FileWriter

- In-memory: CharArrayWriter

- a fake StringWriter! That writes to a StringBuffer

2) classes that add behavior to Writer

- BufferedWriter

- write with a format: PrintWriter

```
File file = new File("files/data.txt");

Writer writer = new FileWriter(file);
```

The `FileWriter` class creates a writer on a file

By default, a file writer writes from the beginning of a file

```
File file = new File("files/data.txt");

Writer writer = new FileWriter(file, true);
```

The `FileWriter` class creates a writer on a file

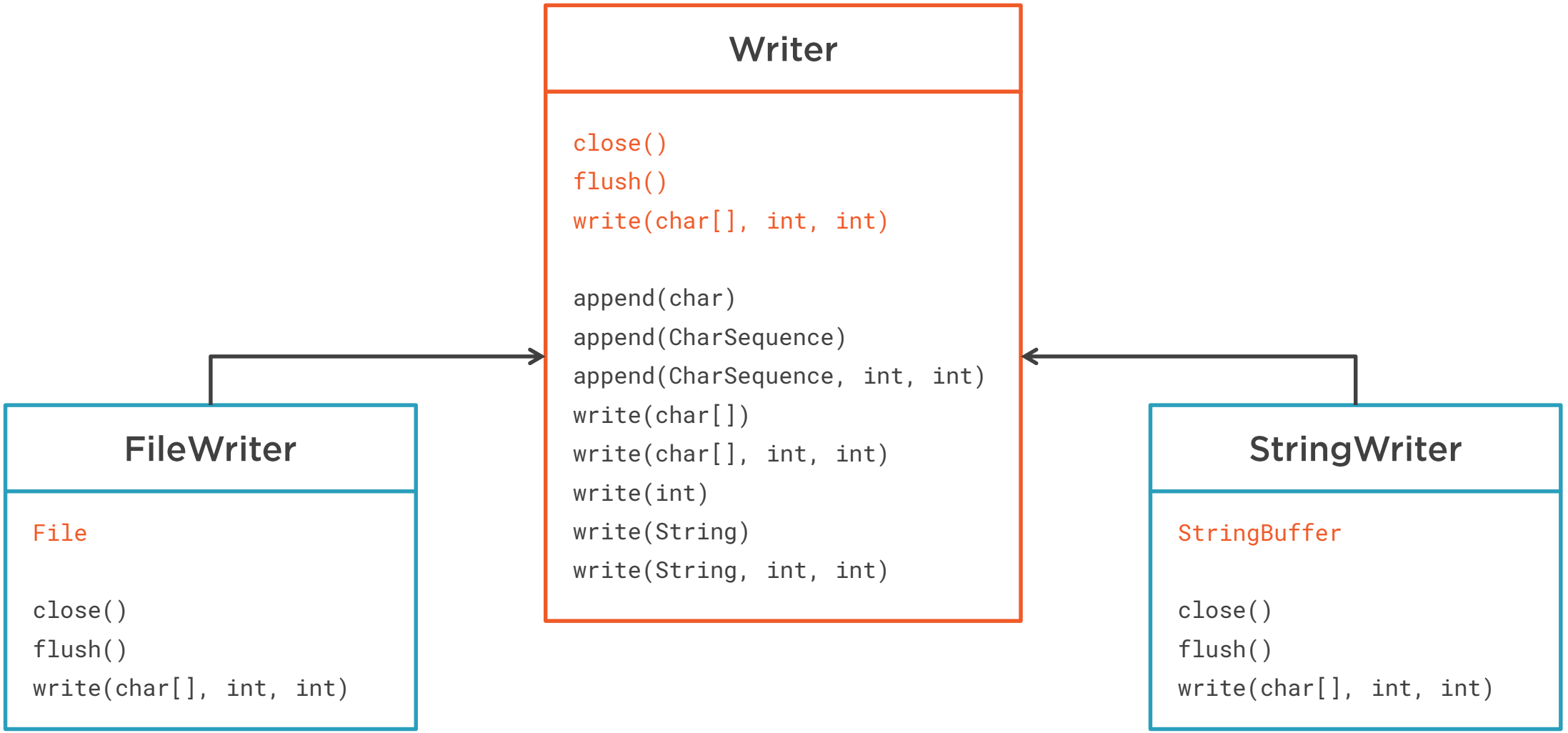
By default, a file writer writes from the beginning of a file

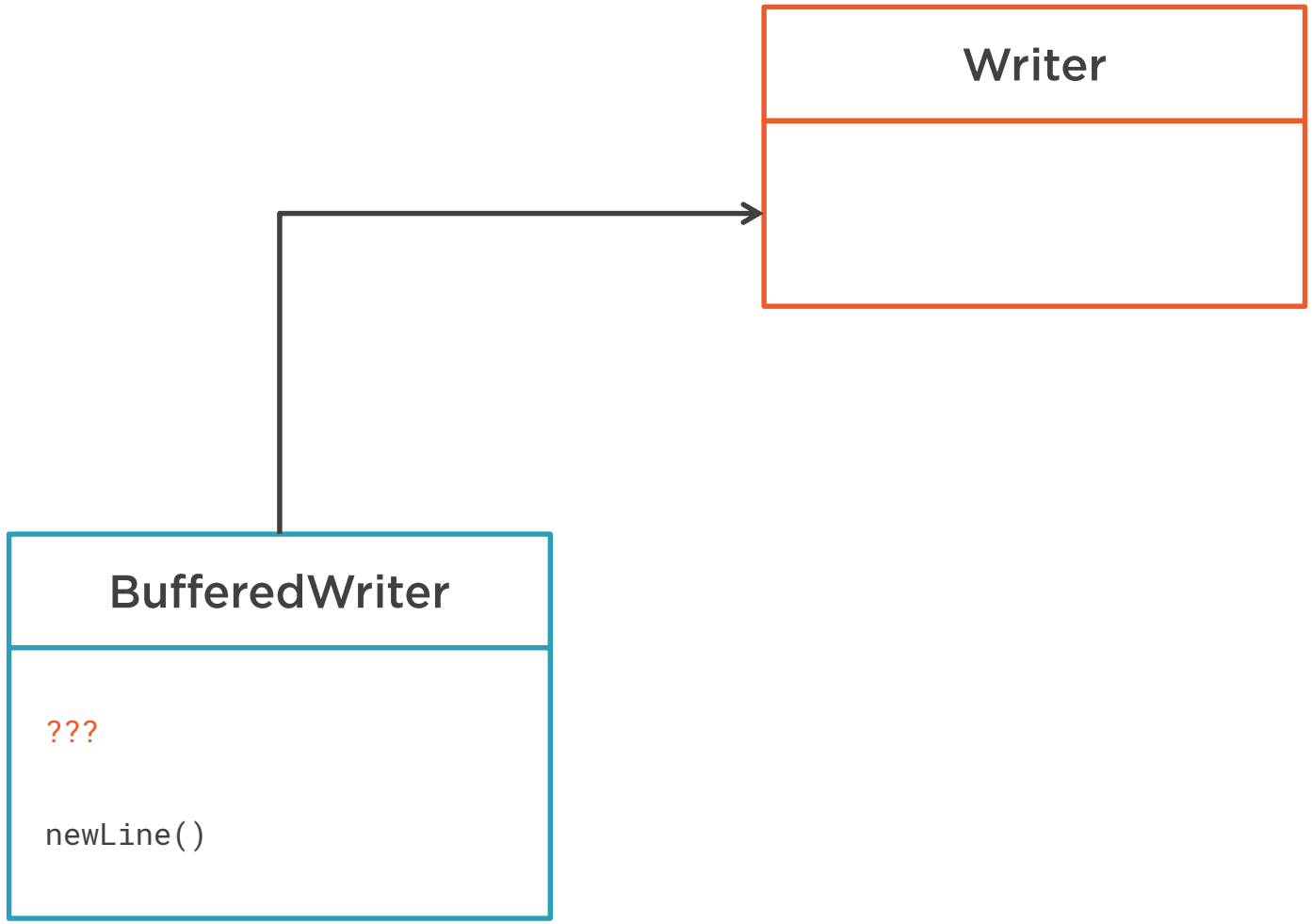
It can also append the new content to the existing file

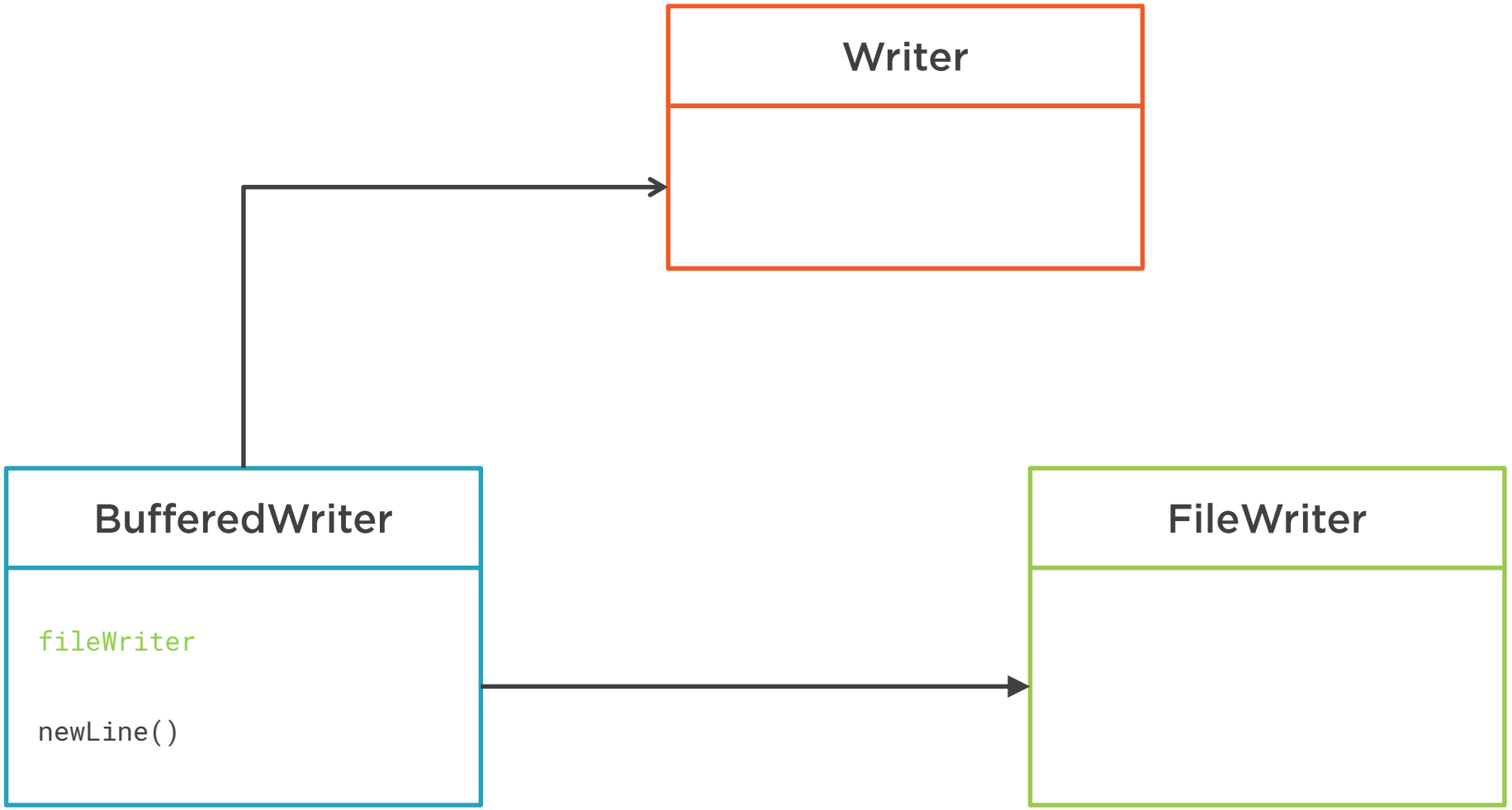
```
File file = new File("files/data.txt");

FileWriter writer = new FileWriter(file);

PrintWriter printer = new PrintWriter(writer);
```

The `FileWriter` class creates a writer on a file

The methods of the print writer can be used to write to the file


Writer
close()
flush()
write(char[], int, int)
append(char)
append(CharSequence)
append(CharSequence, int, int)
write(char[])
write(char[], int, int)
write(int)
write(String)
write(String, int, int)
FileWriter
File
close()
flush()
write(char[], int, int)
StringWriter
StringBuffer
close()
flush()
write(char[], int, int)


Writer
BufferedWriter
???
newLine()


Writer
BufferedWriter
fileWriter
newLine()
FileWriter

The pattern is the same as for the readers, it follows the GoF Decorator pattern

BufferedWriter extends Writer

And is built on an instance of Writer

Just as for the readers...

Java 7 introduced factory methods for buffered writers

```
File file = new File("files/data.txt");
FileWriter fileWriter = new FileWriter(file);
BufferedWriter bufferedWriter1 = new BufferedWriter(fileWriter);

Path path = Paths.get("files/data.txt");
BufferedWriter bufferedWriter2 =
    Files.newBufferedWriter(path);
```

In this case, the file is written with the UTF-8 charset

```
File file = new File("files/data.txt");
FileWriter fileWriter = new FileWriter(file);
BufferedWriter bufferedWriter1 = new BufferedWriter(fileWriter);

Path path = Paths.get("files/data.txt");
BufferedWriter bufferedWriter2 =
    Files.newBufferedWriter(path, StandardCharsets.ISO_8859_1);
```

In this case, the file is written with the UTF-8 charset

But one can also pass other charsets

```
Path path = Paths.get("files/data.txt");
BufferedWriter bufferedWriter2 =
   Files.newBufferedWriter(path, StandardOpenOption.CREATE);
```

Java 7 also brought richer patterns to open files for writing

`StandardOpenOption` is an enumeration that implements `OpenOption`

```
Path path = Paths.get("files/data.txt");
BufferedWriter bufferedWriter2 =
   Files.newBufferedWriter(path, StandardOpenOption.APPEND);
```

Java 7 also brought richer patterns to open files for writing

`StandardOpenOption` is an enumeration that implements `OpenOption`

Some examples of OpenOption:

- WRITE, APPEND
- CREATE, CREATE_NEW
- DELETE_ON_CLOSE

Other OpenOption:

- READ
- TRUNCATE_EXISTING
- SPARSE_FILE
- SYNC, DSYNC

Closing and Flushing

A Writer Must Be Flushed

Writing on an I/O resource (disk or network) is usually made on a buffer

Then flushed to the output resource

There is a flush() method on Writer

Closing a writer triggers a flush call

A flush call propagates to all the streams

Until the output resource is reached

And will trigger a system call

The writing on the I/O is the responsibility of the operating system...

Printing with a Format

The PrintWriter class supports printing with a fomat

The syntax of the format follows the Unix sprintf rules

The format is documented in the Formatter class

Demo

Let us see some code!

Let us create simple writers and see them in action

See how the printer works

Module Wrap Up

What did you learn?

Writers!

Patterns to create writers, Java 1 and 7

How to open, close, and flush writers

How to deal with exceptions

Using formats to print information