



Programming in Java – Part 11 - Basics of Input and Output



Paolo Vercesi
ESTECO SpA

Agenda



Input and output streams

Data streams

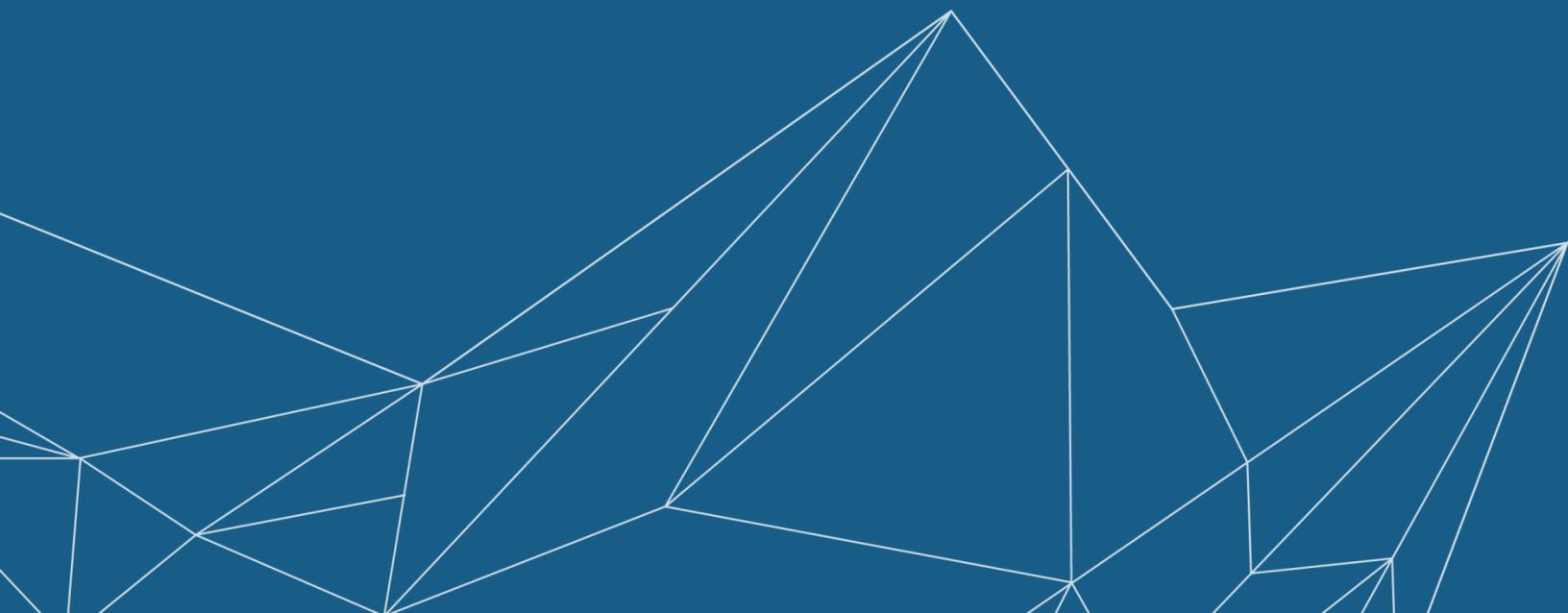
Readers and writers

Console I/O



Input and Output streams

Reading and writing binary data



I/O Streams

I/O in Java is based on *streams*, not to be confused with the streams in `java.util.stream`
The abstraction is the same, but the implementation is different

I/O streams represent a flow of *binary data*

Input streams are used to read from (binary data) sources

Output streams are used to write to (binary data) targets



Introducing java.io.InputStream

```
public class InputStream implements Closeable {  
...  
    public abstract int read() throws IOException;  
...  
}
```

try-with-resources



```
try (InputStream is = ...) {  
    int read;  
    while ((read = is.read()) != -1) {  
        IO.println("Read: " + read);  
    }  
}
```

Reads the **next byte** of data from the input stream

The value byte is returned as an **int** in the range 0 to 255

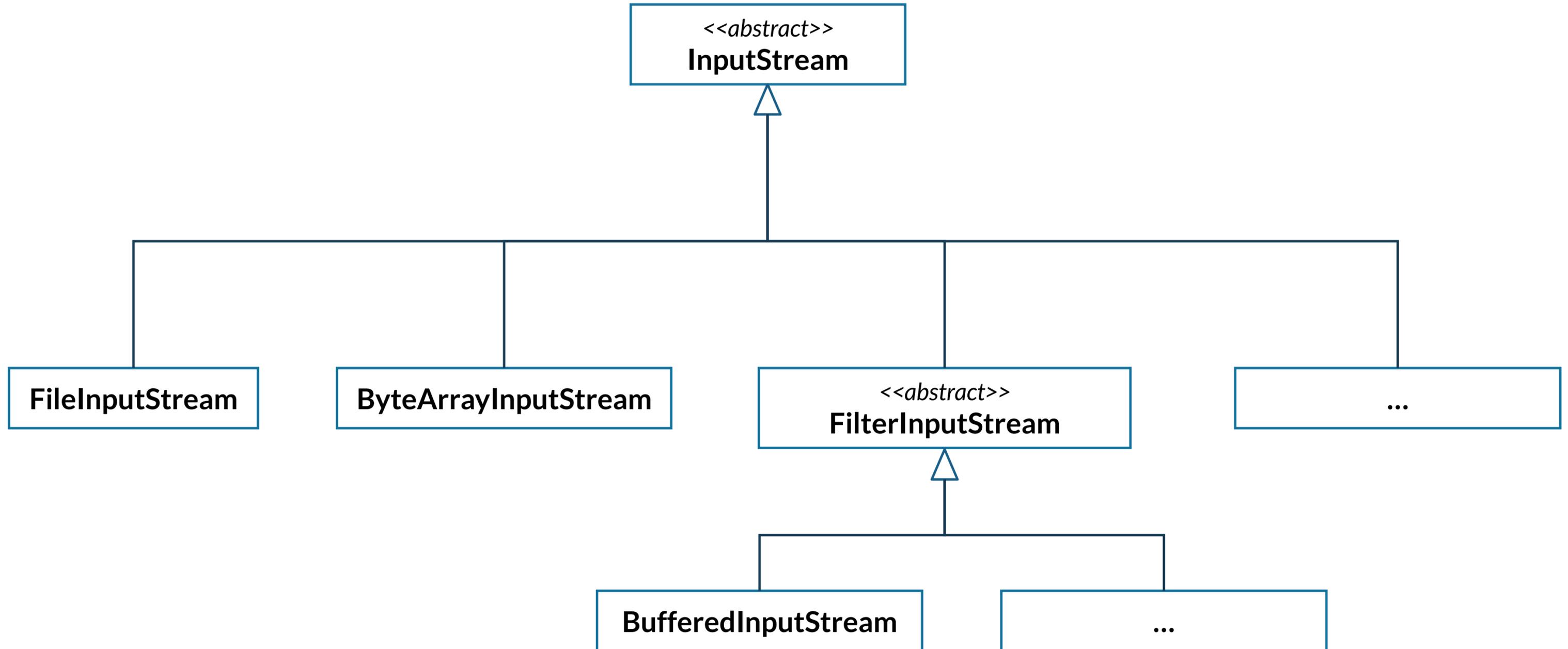
If no byte is available because the **end of the stream** has been reached, the value **-1** is returned

The method **blocks** until

- input data is available
- the end of the stream is detected
- an exception is thrown



The InputStream hierarchy



Examples of InputStream 1/3

```
String fileName = "G:\\My Drive\\ ... \\Input and Output.pptx";
try (InputStream fis = new FileInputStream(fileName)) {
    int count = 0;
    while (fis.read() != -1) {
        count++;
    }
    IO.println("Read: " + count);
}
```



Examples of InputStream 2/3

```
URL url = new URL("https://www.google.it");
try (InputStream urlStream = url.openStream()) {
    int read;
    while ((read = urlStream.read()) != -1) {
        IO.print((char) read);
    }
}
```

WARNING we are converting
a stream of bytes into chars



Examples of InputStream 3/3

```
byte[] byteArray = ...
try (InputStream is = new ByteArrayInputStream(byteArray)) {
    int read;
    while ((read = is.read()) != -1) {
        IO.print(read);
    }
}
```



Other methods in InputStream

```
public int read(byte b[]) throws IOException
```

```
public int read(byte b[], int off, int len) throws IOException
```

```
public byte[] readNBytes(int len) throws IOException
```

```
public int readNBytes(byte[] b, int off, int len) throws IOException
```

```
public byte[] readAllBytes() throws IOException
```

```
public long skip(long n) throws IOException
```

```
public void skipNBytes(long n) throws IOException
```

```
public long transferTo(OutputStream out) throws IOException
```

```
public int available() throws IOException
```

```
public synchronized void mark(int readlimit)
```

```
public synchronized void reset() throws IOException
```

```
public boolean markSupported()
```

```
public void close() throws IOException
```



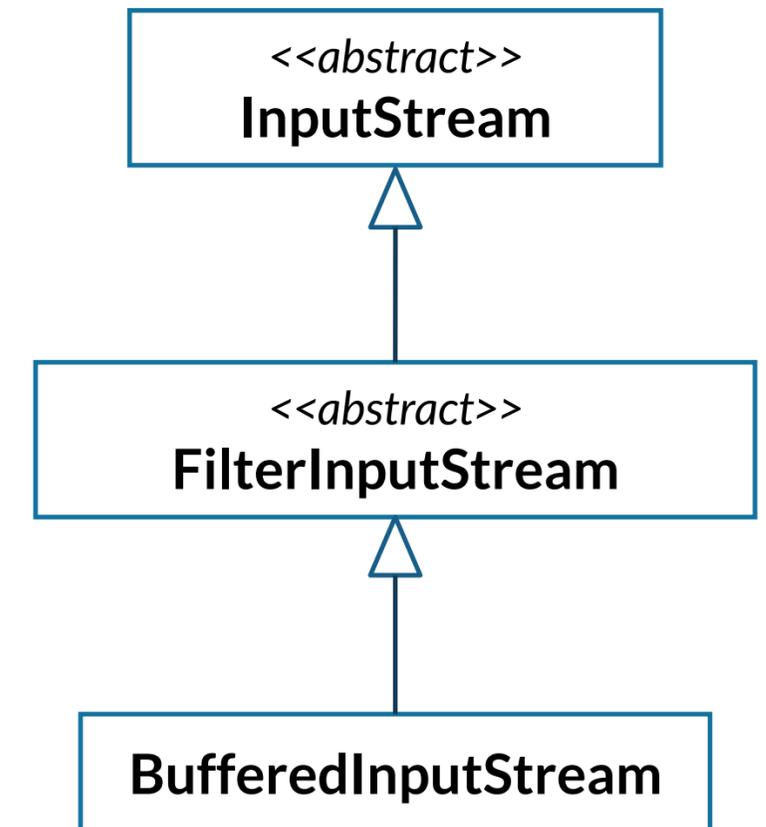
BufferedInputStream

When reading from the filesystem or from the network, the reading of *small chunks of data can be very inefficient*

Java offers buffered input to speedup the reading of small chunks of data

The `BufferedInputStream` reads data in advance in a buffer of a specified size

```
public class BufferedInputStream extends FilterInputStream {  
    public BufferedInputStream(InputStream in)  
    public BufferedInputStream(InputStream in, int size)  
    ...  
}
```



A `BufferedInputStream` is an `InputStream` wrapping another input stream



Working with BufferedInputStream

```
String fileName = "G:\\My Drive\\ ... \\Input and Output.pptx";
try (InputStream fis = new BufferedInputStream(new FileInputStream(fileName))) {
    int count = 0;
    while (fis.read() != -1) {
        count++;
    }
    IO.println("Read: " + count);
}
```

```
URL url = new URL("https://www.google.it");
try (InputStream urlStream = new BufferedInputStream(url.openStream())) {
    int read;
    while ((read = urlStream.read()) != -1) {
        IO.print((char) read);
    }
}
```



Introducing OutputStream

```
public class OutputStream implements Closeable {  
...  
    public abstract void write(int b)  
        throws IOException;  
...  
}
```

try-with-resources



```
try (OutputStream os = ...) {  
    int[] data = ...;  
    for (int datum : data) {  
        os.write(datum);  
    }  
}
```

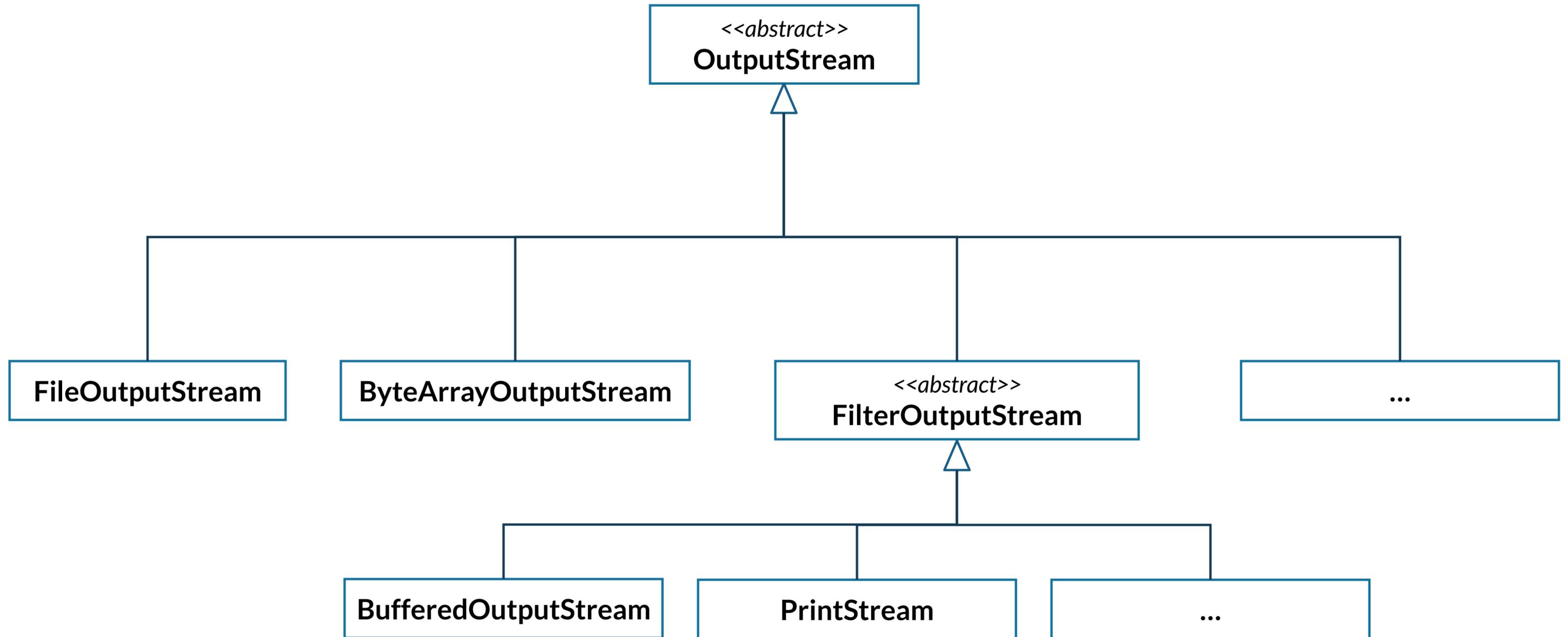
*Writes the specified **byte** to this output stream*

*The byte to be written is the **8 low-order bits** of the argument *b**

*The **24 high-order bits** of *b* are ignored*



The OutputStream hierarchy



Examples of OutputStream

```
try (OutputStream fos = new FileOutputStream("A:\\git\\sdm\\pippo.dat")) {  
    for (int i = 0; i < 10; i++) {  
        fos.write(i);  
    }  
}
```

```
try (ByteArrayOutputStream baos = new ByteArrayOutputStream()) {  
    for (int i = 0; i < 10; i++) {  
        baos.write(i);  
    }  
    IO.println(Arrays.toString(baos.toByteArray()));  
}
```



Other methods of OutputStream

```
public void write(byte b[]) throws IOException
```

```
public void write(byte b[], int off, int len) throws IOException
```

```
public void flush() throws IOException
```

```
public void close() throws IOException
```



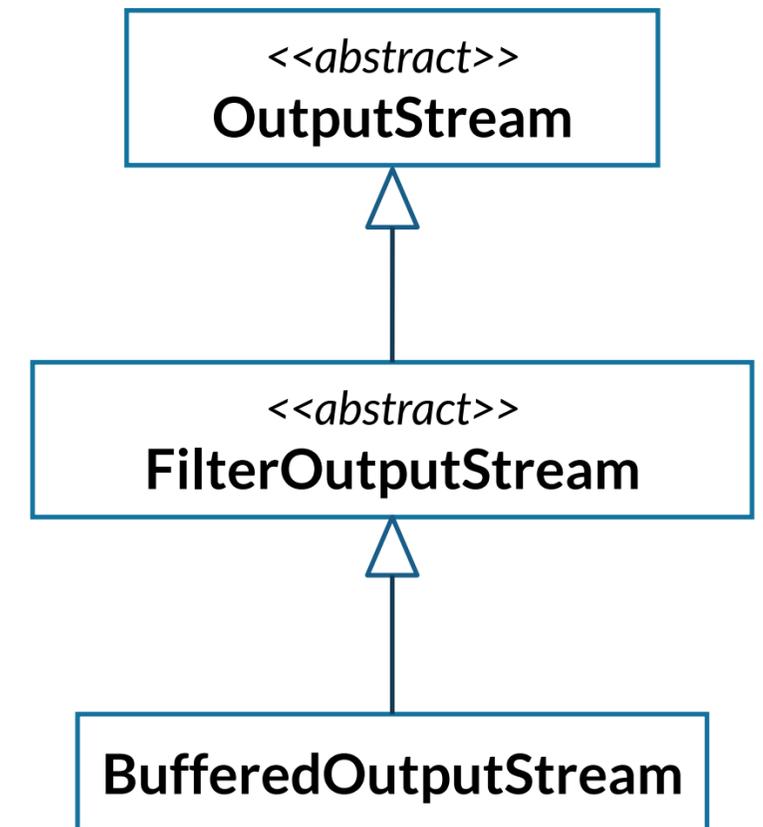
BufferedOutputStream

When writing to the filesystem or to the network, the writing of *small chunks of data can be very inefficient*

Java offers buffered output to speedup the writing of small chunks of data

The `BufferedOutputStream` writes data to the wrapped stream only when the buffer is full or when `flush()` is invoked

```
public class BufferedOutputStream extends FilterOutputStream {  
    public BufferedOutputStream(OutputStream out)  
    public BufferedOutputStream(OutputStream out, int size)  
    ...  
}
```



A `BufferedOutputStream` is an `OutputStream` wrapping another output stream



Working with BufferedOutputStream

```
String fileName = "A:\\git\\sdm\\pippo.dat";
try (OutputStream fos = new BufferedOutputStream(new FileOutputStream("..")) {
    for (int i = 0; i < 10; i++) {
        fos.write(i);
    }
}
```



Streams must be closed

Use *try-with-resources* if you open (create) and use the stream from the *same* method

Explicitly invoke `close()` if you open (create) and use the stream in *different* methods





Data streams

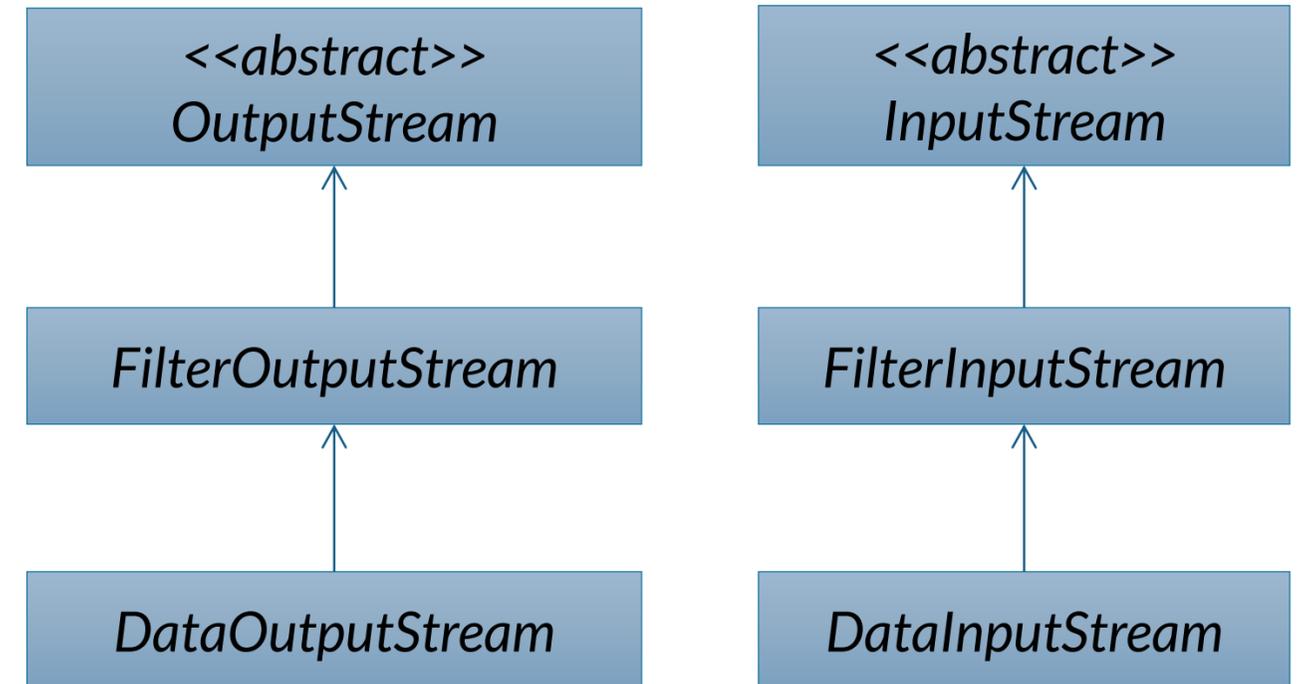
Reading and writing Java types

Primitive types I/O

`DataOutputStream` and `DataInputStream` enable you to write or read primitive data to or from a stream

They implement the `DataOutput` and `DataInput` interfaces, respectively. These interfaces define methods that convert primitive values to or from a sequence of bytes

These streams make it easy to store **binary data**, such as integers or floating-point values, in a file



DataInputStream

```
public class DataInputStream extends FilterInputStream implements DataInput {
    public DataInputStream(InputStream in)
        public final boolean readBoolean() throws IOException
        public final byte readByte() throws IOException
        public final int readUnsignedByte() throws IOException
        public final short readShort() throws IOException
        public final int readUnsignedShort() throws IOException
        public final char readChar() throws IOException
        public final int readInt() throws IOException
        public final long readLong() throws IOException
        public final float readFloat() throws IOException
        public final double readDouble() throws IOException
        public final String readUTF() throws IOException
    ...
}
```



DataOutputStream

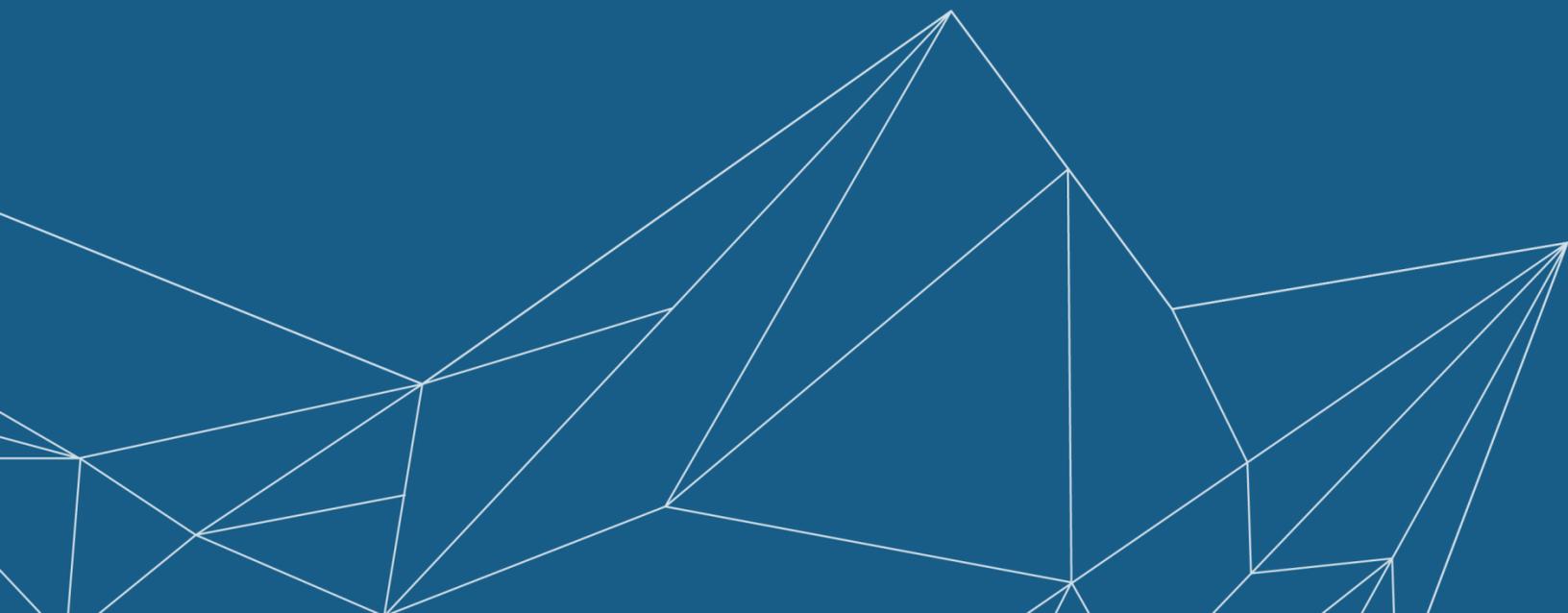
```
public class DataOutputStream extends FilterOutputStream implements DataOutput {  
    public DataOutputStream(OutputStream out)  
    public void flush() throws IOException  
    public final void writeBoolean(boolean v) throws IOException  
    public final void writeByte(int v) throws IOException  
    public final void writeShort(int v) throws IOException  
    public final void writeChar(int v) throws IOException  
    public final void writeInt(int v) throws IOException  
    public final void writeLong(long v) throws IOException  
    public final void writeFloat(float v) throws IOException  
    public final void writeDouble(double v) throws IOException  
    public final void writeBytes(String s) throws IOException  
    public final void writeChars(String s) throws IOException  
    public final void writeUTF(String str) throws IOException  
    ...  
}
```





Readers and Writers

Reading and writing text

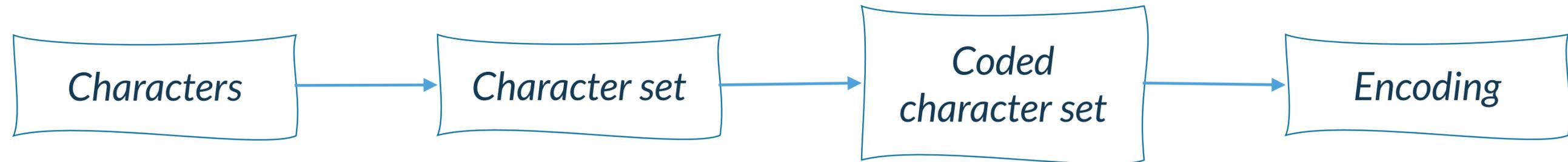


Text streams

What about *reading and writing text*?



Character sets and encoding



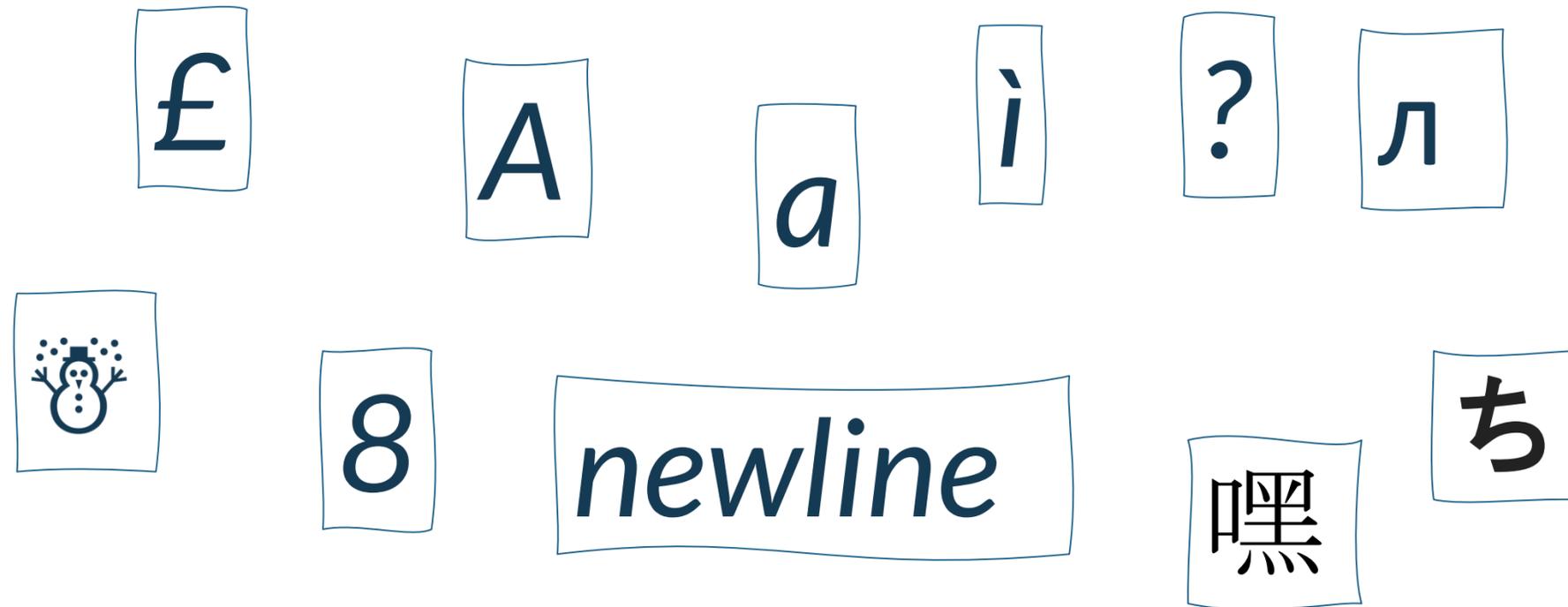
To know everything about character sets and encodings:

<https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/>



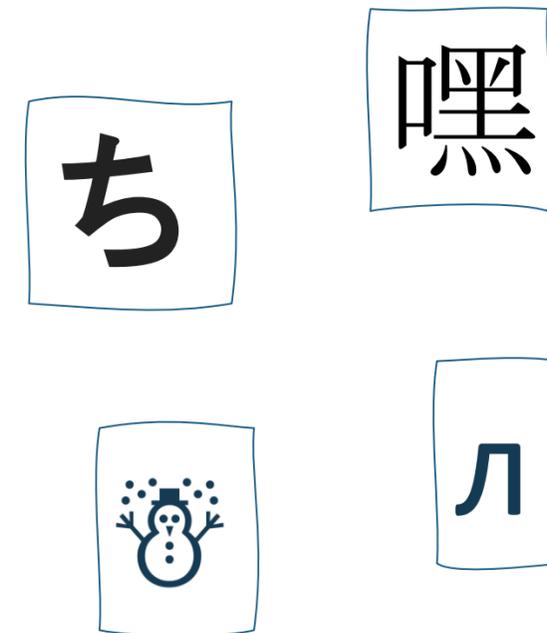
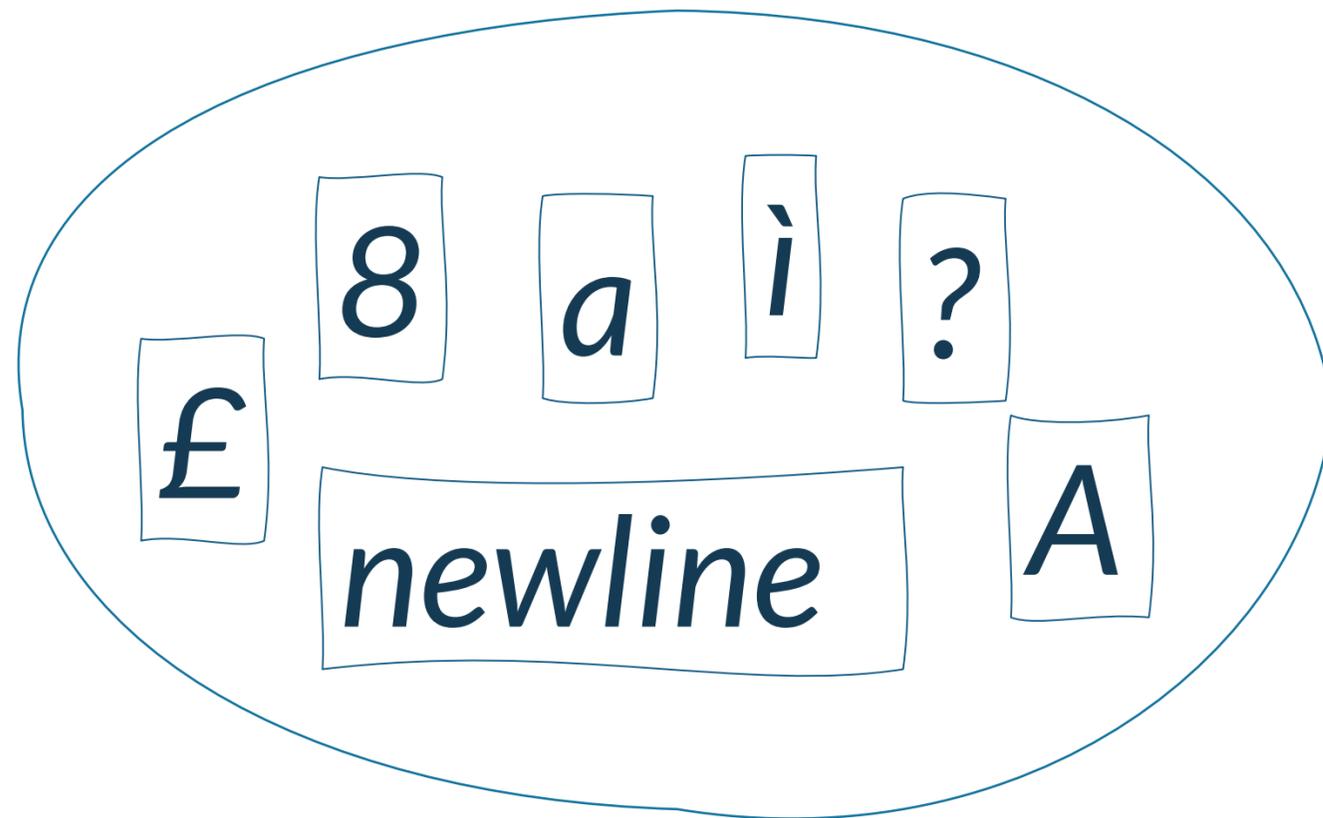
Character

A *character* is a minimal unit of text that has semantic value.



Character set

A **character set** is a collection of characters that might be used by multiple languages. For example, the **Latin character** set is used by English and most European languages, though the **Greek character** set is used only by the Greek language.



Unicode terminology – Coded character set

A *coded character set* is a character set where each character is assigned a unique number (code point)

US-ASCII

Code point	Character
0	NUL
1	SOH
...	...
65	A
66	B
67	C
...	...
126	~
127	DEL

Windows-1252/ISO-8859-1

Code point	Character
0	NUL
1	SOH
...	...
65	A
66	B
67	C
...	...
254	þ
255	ÿ

Windows-1250

Code point	Character
0	NUL
1	SOH
...	...
65	A
66	B
67	C
...	...
254	ţ
255	·

Windows-1252 and ISO-8859-1 are not the same character set, but they differ for some code points assigned to control codes
For HTML5 they can be considered the same <https://www.w3.org/TR/encoding/>



Unicode

The Unicode standard defines 144,697 characters and their respective code points
This character set is called **Universal Coded Character Set** (UCS, Unicode)

Positions 0 through 127 of UCS are the same as in **US-ASCII**

Positions 0 through 255 of UCS and Unicode are the same as in **ISO-8859-1**

Positions 0 through 65535 of UCS (**Basic Multilingual Plan**) cover all the commonly used languages



How do we **encode** this information in computers?



Encodings

1 byte is enough to encode the whole US-ASCII and ISO-8859-1 character sets

For characters sets with more than 256 characters with need to use **multibyte encodings**

Generally, a character sets define its own encoding and so the term charset is used to refer to both the character set and the encoding, e.g., HTTP and HTML define a charset parameter and attribute, respectively, to define the combination character set/encoding

UCS is currently the most important character sets and it has multiple encodings, so this character set is represented by the name of the encoding, UTF-8, UTF-16, or UTF-32

A 00000041	Ω 000003A9	語 00008A9E	卍 00010384	UTF-32
A 0041	Ω 03A9	語 8A9E	卍 D800 DF84	UTF-16
A 41	Ω CE A9	語 E8 AA 9E	卍 F0 90 8E 84	UTF-8



Java characters

The Java primitive type `char` uses 16 bit to represent characters

So, the `char` type is not able to represents all Unicode characters, indeed, Java internally represents text in 16-bits **code units** using UTF-16

The `char` type does not represent characters but code units, this is relevant only when we are using a language outside the **Basic Multilanguage Plane**, e.g., the cuneiform language, Phoenician, etc.



Encodings supported by Java

Every implementation of the Java platform is required to support the following standard charsets, usually, every implementation supports *many more charsets*

Charset	Description
US-ASCII	Seven-bit ASCII, a.k.a. ISO646-US, a.k.a. the Basic Latin block of the Unicode character set
ISO-8859-1	ISO Latin Alphabet No. 1, a.k.a. ISO-LATIN-1
UTF-8	Eight-bit UCS Transformation Format
UTF-16BE	Sixteen-bit UCS Transformation Format, big-endian byte order
UTF-16LE	Sixteen-bit UCS Transformation Format, little-endian byte order
UTF-16	Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark

If in doubt, create documents in **UTF-8**

What about reading documents in unknown encodings?



There is **no** such thing
as **Plain Text**



Text streams

To write (read) text to (from) an output(input) stream we need to *encode (decode)* the text into (from) a *binary stream*

Fortunately, Java is doing this for us, given we provide a very tiny piece of information, the *encoding/charset* of the stream

Since Java 18, Java uses *UTF-8* as the *default charset*

Unfortunately, that *default charset* could be different from the *native charset* of your platform

E.g., the native charset on Linux can be UTF-8 while on Windows can be Windows-1252 (in Italy)

Always ask yourself these two questions

1. are there characters outside the US-ASCII charset?
2. if yes, what's the encoding?



Introducing java.io.Reader

```
public abstract class Reader implements Closeable {  
    ...  
    public int read() throws IOException;  
    ...  
}
```

```
try (Reader reader = ... ) {  
    int ch = -1;  
    while ((ch = reader.read()) != -1) {  
        System.out.print((char) ch);  
    }  
}
```

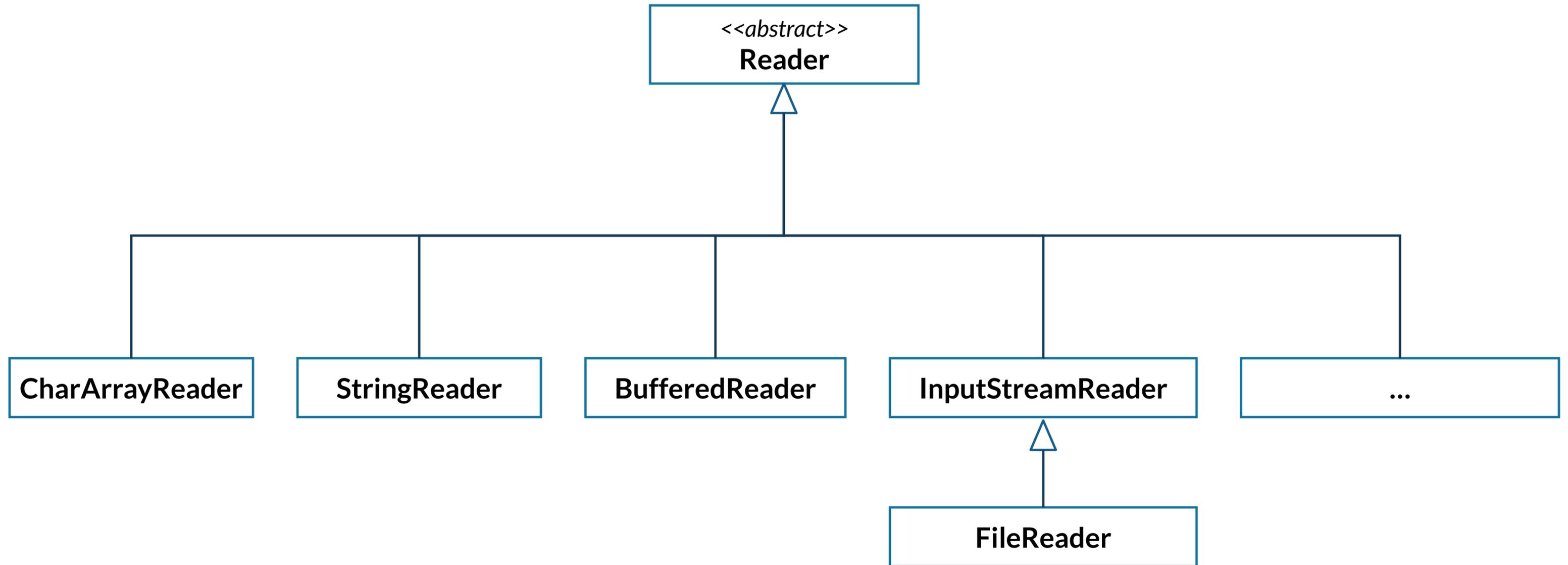
Reads a **single character** as an integer in the range 0 to 65535 or -1 if the end of the stream has been reached

This method will **block** until

- a character is available
- an I/O error occurs
- or the end of the stream is reached.



The Reader hierarchy



Examples of Reader 1/2

```
String fileName = "A:\\git\\sdm\\src\\it\\units\\sdm\\iostreams\\Examples.java";  
try (Reader reader = new InputStreamReader(new FileInputStream(fileName), UTF_8)) {  
    int ch = -1;  
    while ((ch = reader.read()) != -1) {  
        IO.print((char) ch);  
    }  
}
```

```
try (Reader reader = new FileReader(fileName, StandardCharsets.UTF_8)) {  
    int ch = -1;  
    while ((ch = reader.read()) != -1) {  
        IO.print((char) ch);  
    }  
}
```



Examples of Reader 2/2

```
URL url = new URL("https://www.google.it");
try (InputStream urlStream = url.openStream()) {
    int read;
    while ((read = urlStream.read()) != -1) {
        IO.print((char) read);
    }
}
```

We guess the encoding to be UTF-8

```
URL url = new URL("https://www.google.it");
try (Reader reader = new InputStreamReader(url.openStream(), StandardCharsets.UTF_8)) {
    int ch;
    while ((ch = reader.read()) != -1) {
        IO.print((char) ch);
    }
}
```



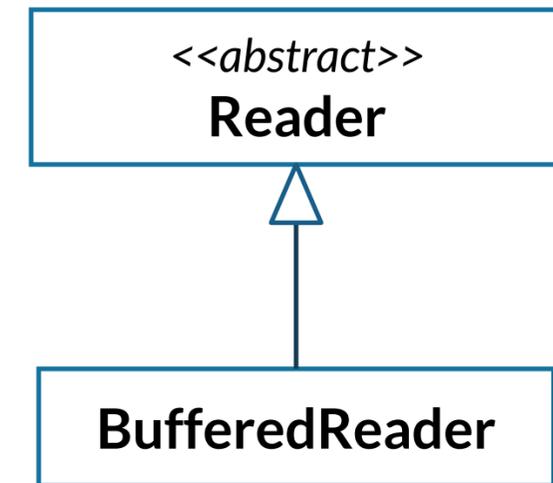
BufferedReader

When reading from the filesystem or from the network, the reading of *small chunks of data can be very inefficient*

Java offers buffered input to speedup the reading of small chunks of data

The `BufferedReader` reads data in advance in a buffer of a specified size

```
public class BufferedReader extends Reader {  
    public BufferedReader(Reader in)  
    public BufferedReader(Reader in, int size)  
    ...  
}
```



A `BufferedReader` *is a* `Reader` wrapping another reader



Working with BufferedReader

```
try (Reader reader = new BufferedReader(new FileReader(fileName, UTF_8))) {  
    int ch = -1;  
    while ((ch = reader.read()) != -1) {  
        IO.print((char) ch);  
    }  
}
```

```
try (BufferedReader reader = new BufferedReader(new FileReader(fileName, UTF_8))) {  
    String line;  
    while ((line = reader.readLine()) != null) {  
        IO.println(line);  
    }  
}
```

```
try (BufferedReader reader = new BufferedReader(new FileReader(fileName, UTF_8))) {  
    reader.lines().forEach(IO::println);  
}
```



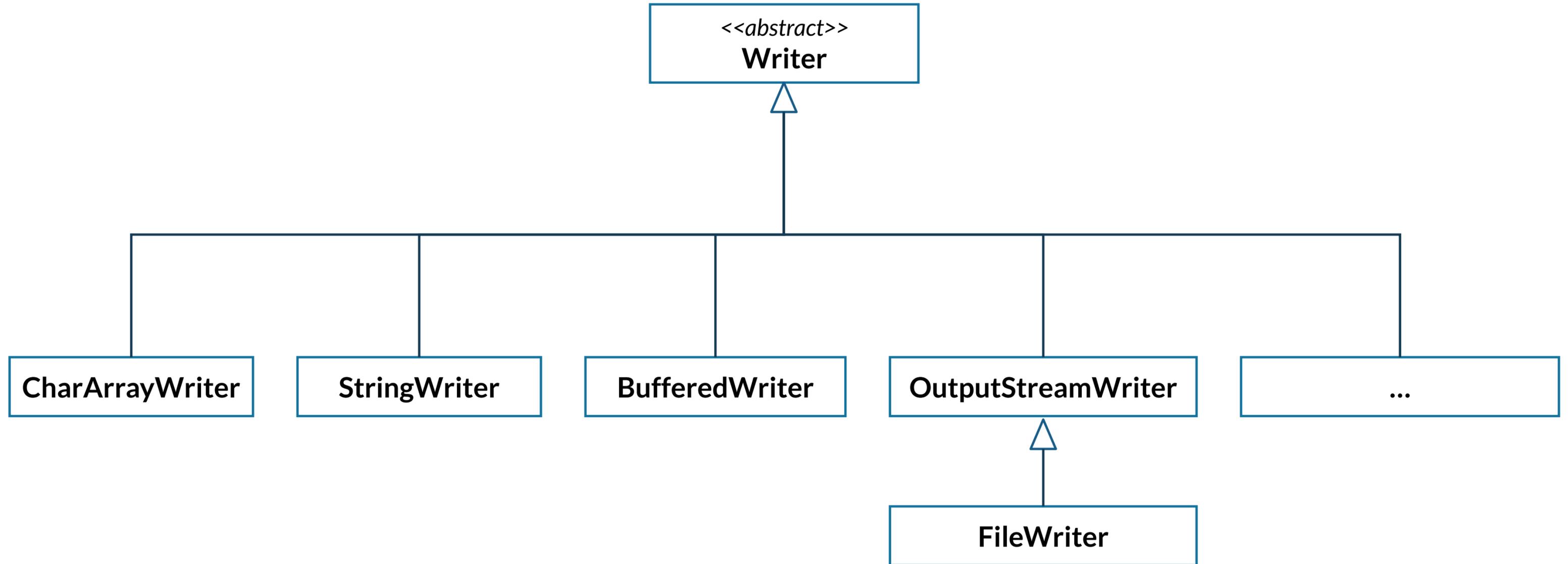
Introducing java.io.Writer

```
public abstract class Writer implements Closeable {  
    ...  
    public void write(int c) throws IOException  
    public void write(String str) throws IOException  
    ...  
}
```

Writes a single character. The character to be written is contained in the 16 low-order bits of the given integer value; the 16 high-order bits are ignored

```
String data = "some data";  
  
try (Writer writer = ...) {  
    writer.write(data);  
}  
  
try (Writer writer = ...) {  
    for (int i = 0; i < data.length(); i++) {  
        writer.write(data.charAt(i));  
    }  
}
```

The Writer hierarchy



Examples of Writer

```
String data = "some data";  
try (Writer writer = new FileWriter("A:\\git\\sdm\\pippo.txt", StandardCharsets.UTF_8)) {  
    writer.write(data);  
}
```

```
try (Writer writer = new OutputStreamWriter(new FileOutputStream(fileName1), UTF_8)) {  
    writer.write(data);  
}
```



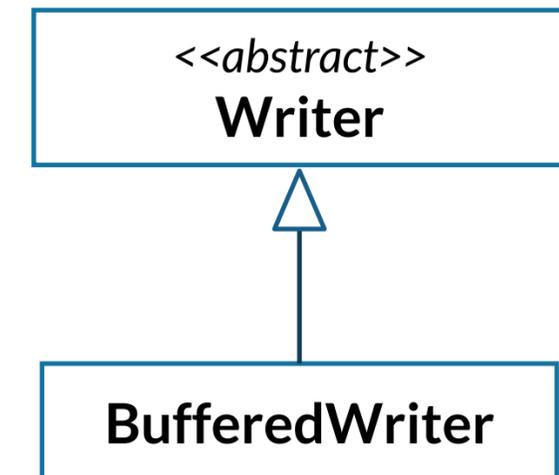
BufferedWriter

When writing to the filesystem or to the network, the writing of *small chunks of data can be very inefficient*

Java offers buffered output to speedup the writing of small chunks of data

The `BufferedWriter` writes data to the wrapped writer only when the buffer is full or when `flush()` is invoked

```
public class BufferedWriter extends Writer {  
    public BufferedWriter(Writer writer)  
    public BufferedWriter(Writer writer, int size)  
    ...  
}
```



A `BufferedWriter` *is a* `Writer` wrapping another writer



Readers and Writers must be closed

Use *try-with-resources* if you open (create) and use the stream from the *same* method

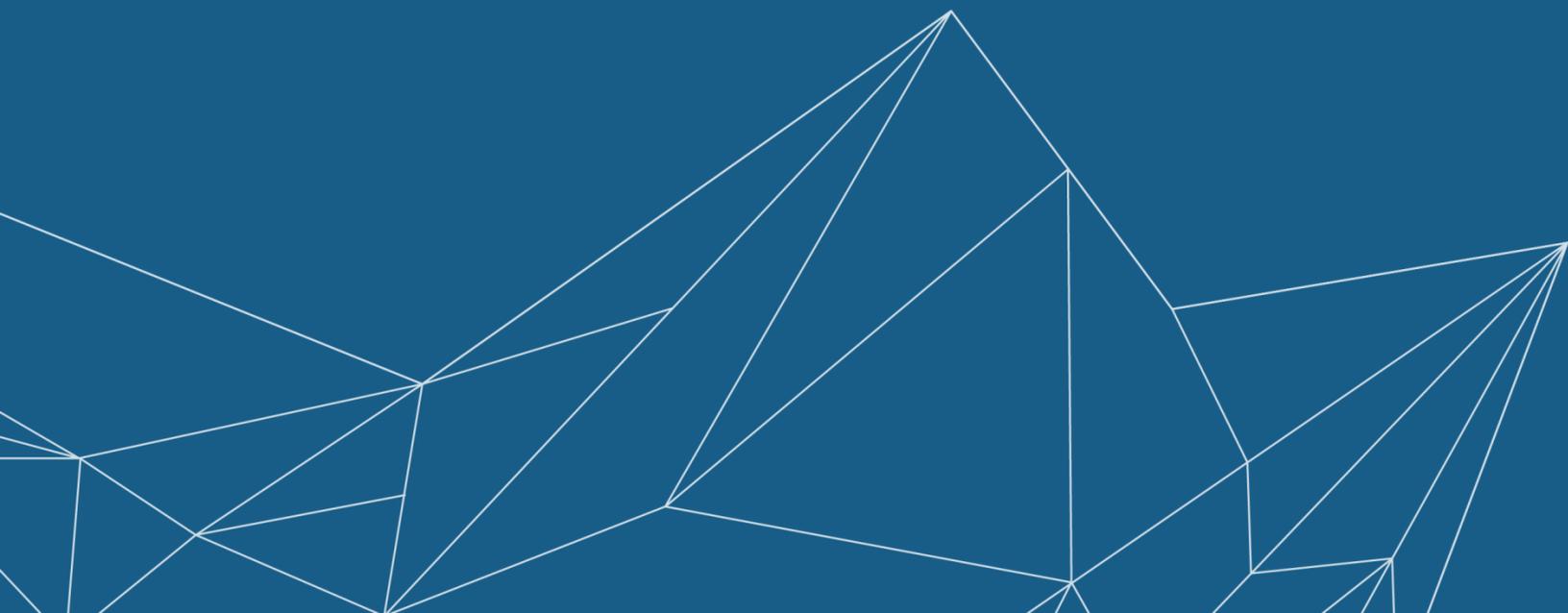
Explicitly invoke `close()` if you open (create) and use the stream from *different* methods





Console I/O

Reading and writing from the console



Console I/O

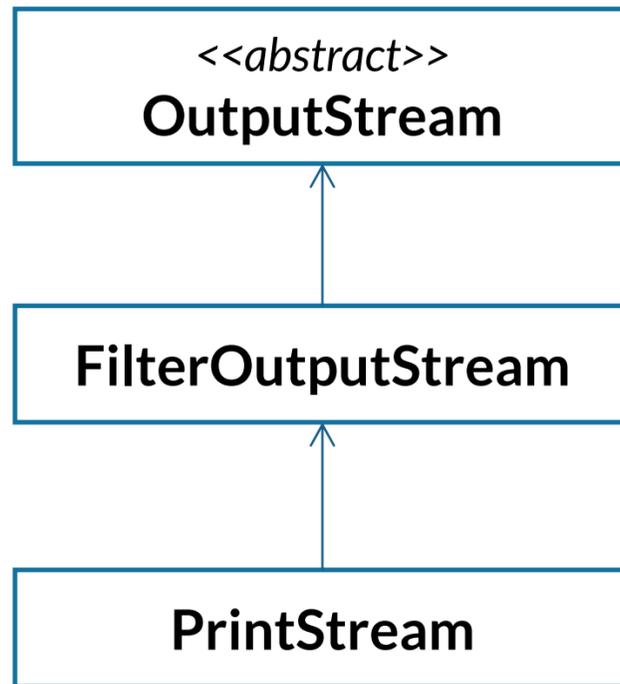
System.in is an object of type `InputStream`

System.out and *System.err* are objects of type `PrintStream`.

These are byte streams, even though they are typically used to read and write characters from and to the console



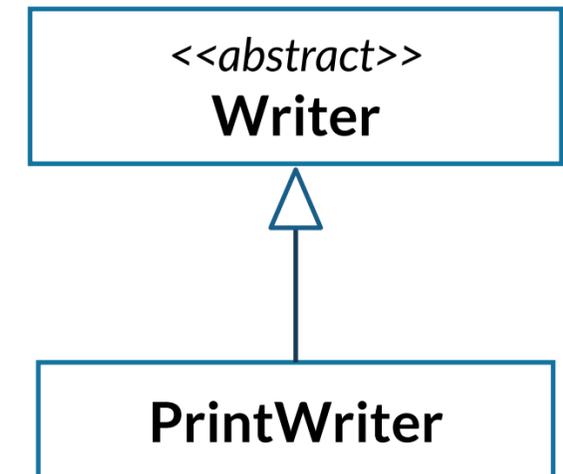
Digression - PrintStream & PrintWriter



`PrintStream` & `PrintWriter` are a stream and a writer providing functionalities to conveniently **print** Java data types in **text format** into streams

I.e., printing a byte to a `PrintStream` or a `PrintWriter` results in writing its **textual representation** rather than its **binary representation**

`PrintStream` and `PrintWriter` are directly used to write text



PrintXXX API

```
void println()  
void print/println(boolean x)  
void print/println(char x)  
void print/println(char[] x)  
void print/println(double x)  
void print/println(float x)  
void print/println(int x)  
void print/println(long x)  
void print/println(Object x)  
void print/println(String x)
```

```
PrintWriter format(String format, Object... args)
```

```
PrintWriter format(Locale l, String format, Object... args)
```

None of these methods throws any `IOException`, use `checkError()` to test the error status.



Reading from the console

```
public class Echo {  
  
    public static void main(String[] args) throws IOException {  
        BufferedReader reader = new BufferedReader(  
            new InputStreamReader(System.in, System.console().charset()));  
  
        String line;  
        while (!(line = reader.readLine()).isEmpty()) {  
            System.out.println("READ: " + line);  
        }  
    }  
}
```

DO NOT CLOSE *System.in*



The `java.lang.io.Console` class

```
Charset charset()  
void flush()  
Console format(String format, Object... args)  
Console printf(String format, Object... args)  
Reader reader()  
String readLine()  
String readLine(String format, Object... args)  
char[] readPassword()  
char[] readPassword(String format, Object... args)  
PrintWriter writer()
```

The methods `format()` and `printf()` are *exactly the same*
Remember to *flush* when you use `writer()`



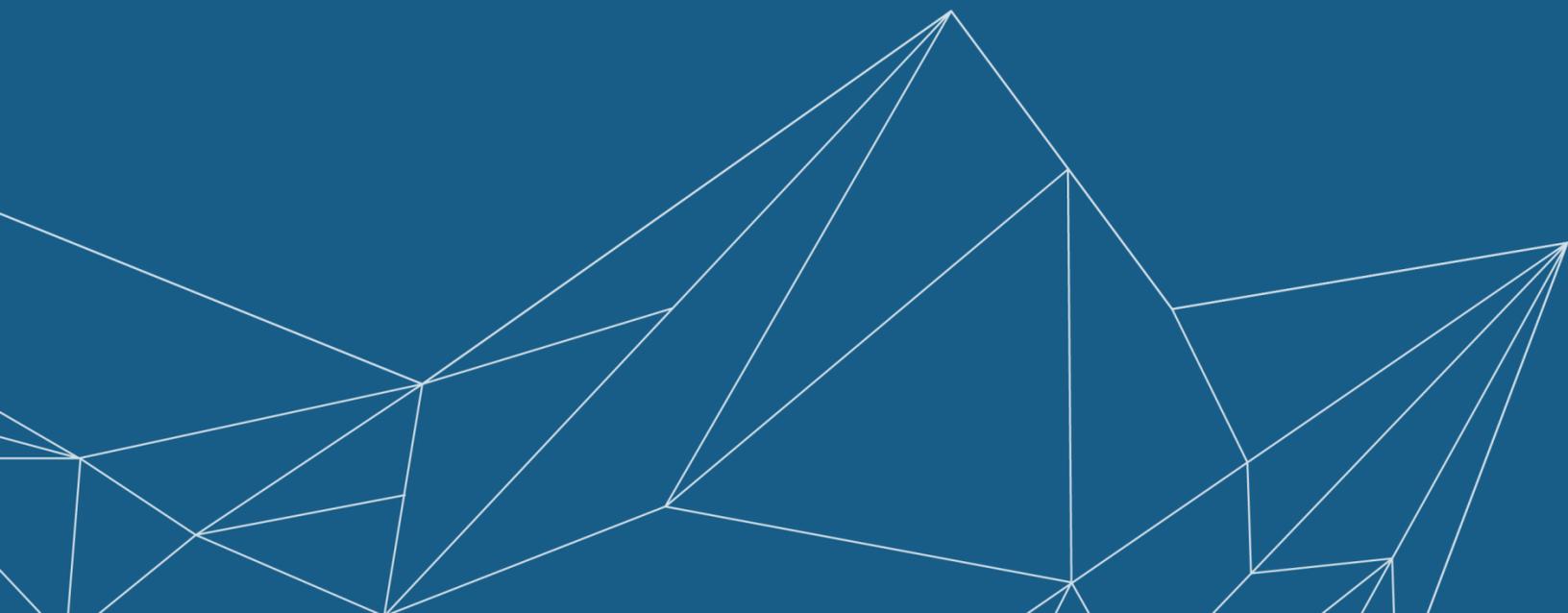
Reading from the Console

```
public class Echo {  
    public static void main(String[] args) throws IOException {  
        String line;  
        while (!(line = System.console().readLine()).isEmpty()) {  
            System.console().printf("READ: %s\n", line);  
        }  
    }  
}
```





Exercises



Exercise 1

Write a program that given a file writes the hexdump of the file in another file.

E.g., the command

```
java it.units.sdm.HexDump Streams.pdf
```

Shall produce a file `Streams.pdf.hexdump` with the following content (don't print characters outside the US-ASCII charset).

```
2550 4446 2d31 2e37 0d0a 25b5 b5b5 b50d %PDF-1.7..%.....
0a31 2030 206f 626a 0d0a 3c3c 2f54 7970 .1 0 obj..<</Typ
652f 4361 7461 6c6f 672f 5061 6765 7320 e/Catalog/Pages
3220 3020 522f 4c61 6e67 2865 6e29 202f 2 0 R/Lang(en) /
5374 7275 6374 5472 6565 526f 6f74 2033 StructTreeRoot 3
...
```

Write a program that do the reverse, i.e., creates a files from the hexdump, and verify that the recreated file is still usable!



Exercise 2

Write a program to change the charset of a text file

```
java it.units.sdm.Recode pippo.txt UTF-8 Windows-1252
```

`pippo.txt` should contain some non US-ASCII character, e.g., 'è'.

The output file could be named `pippo.txt.Windows-1252`

Extra step, throw an exception if the program encounter an unmappable byte sequence, e.g., when reading a file containing the '嘿' character by using the Windows-1252 charset. Hint: explore the `java.io.Files` class



Exercise 3

Write a class (or a set of classes) that given a text file ~~string~~ it produces a Term Frequency table. Consider the option to provide a list of stop words, normalization, etc. Provide an option to print the table in alphabetical order and by frequency.

“Term frequency (TF) means how often a term occurs in a document. In the context of natural language, terms correspond to words or phrases ...”



Term	Frequency
<i>english</i>	8
<i>language</i>	7
<i>words</i>	12

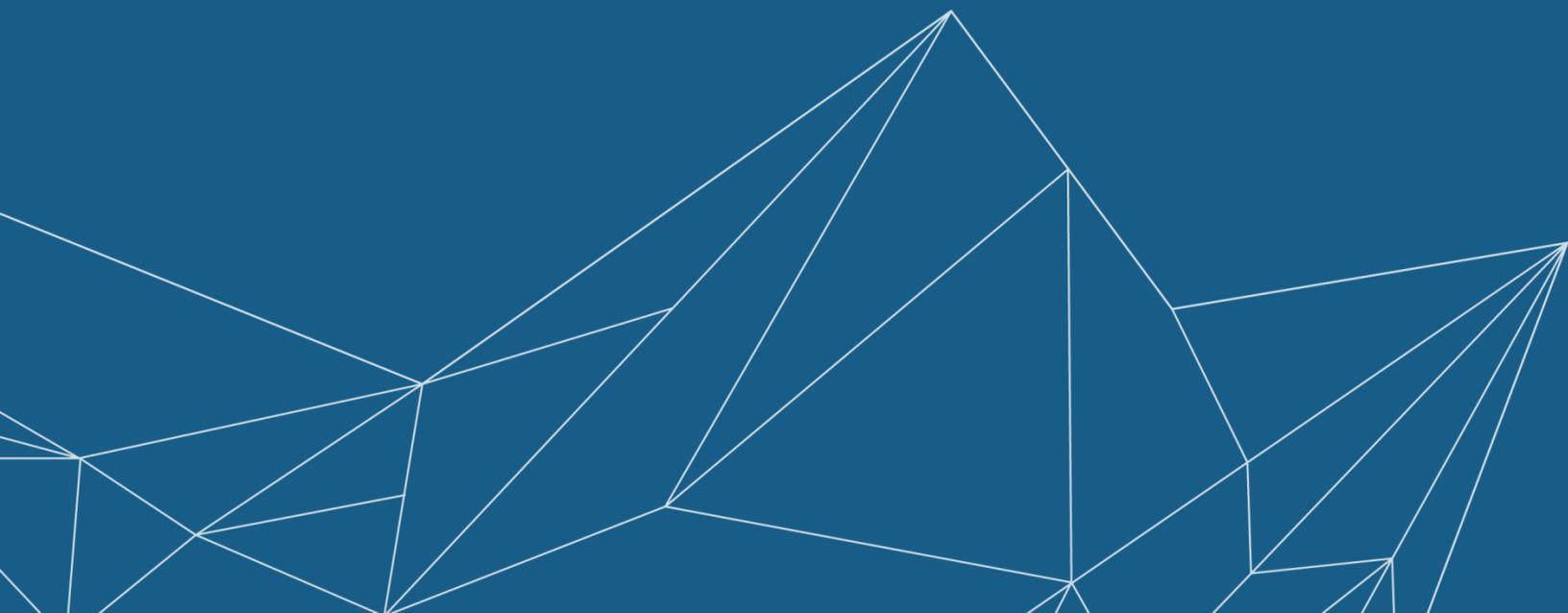
...

<i>input</i>	7
<i>cactus</i>	1
<i>fireworks</i>	3





Solution of exercises



Exercise 1

Write a program that given a file writes the hexdump of the file in another file.

E.g., the command

```
java it.units.sdm.HexDump Streams.pdf
```

Shall produce a file `Streams.pdf.hexdump` with the following content (don't print characters outside the US-ASCII charset).

```
2550 4446 2d31 2e37 0d0a 25b5 b5b5 b50d %PDF-1.7..%.....
0a31 2030 206f 626a 0d0a 3c3c 2f54 7970 .1 0 obj..<</Typ
652f 4361 7461 6c6f 672f 5061 6765 7320 e/Catalog/Pages
3220 3020 522f 4c61 6e67 2865 6e29 202f 2 0 R/Lang(en) /
5374 7275 6374 5472 6565 526f 6f74 2033 StructTreeRoot 3
...
```

Write a program that do the reverse, i.e., creates a files from the hexdump, and verify that the recreated file is still usable!



```
public class HexDump {

    public static void main(String[] args) throws IOException {
        HexDump hexDump = new HexDump();
        try (FileInputStream inputStream = new FileInputStream(args[0])) {
            hexDump.dump(inputStream, new PrintWriter(System.out, true));
        }
    }

    public void dump(InputStream inputStream, Writer writer) throws IOException {
        StringBuilder binary = new StringBuilder();
        StringBuilder text = new StringBuilder();
        String lineSeparator = "";
        int groups = 0;
        int read;
        while ((read = inputStream.read()) != -1) {
            binary.append(DateFormat.of().toHexDigits((byte) read));
            text.append(read < 32 || read > 126 ? '.' : (char) read);
            if (++groups % 2 == 0) {
                binary.append(' ');
            }
            if (groups % 16 == 0) {
                writer.append(lineSeparator).append(binary).append(" ").append(text);
                binary.delete(0, binary.length());
                text.delete(0, text.length());
                lineSeparator = "\n";
            }
        }
        if (!binary.isEmpty()) {
            writer.append(binary).append(" ".repeat(41 - binary.length())).append(text);
        }
    }
}
```



```
@Test
void testOneGroup() throws IOException {
    HexDump hexDump = new HexDump();
    InputStream inputStream = new ByteArrayInputStream(new byte[] {0x25, 0x50});
    StringWriter writer = new StringWriter();

    hexDump.dump(inputStream, writer);

    assertEquals("2550", writer.toString());
}
```

```
@Test
void testTwoGroups() throws IOException {
    HexDump hexDump = new HexDump();
    InputStream inputStream = new ByteArrayInputStream(new byte[] {0x25, 0x50, 0x44, 0x46});
    StringWriter writer = new StringWriter();

    hexDump.dump(inputStream, writer);

    assertEquals("2550 4446", writer.toString());
}
```



```
@Test
```

```
void testNonPrintables() throws IOException {  
    HexDump hexDump = new HexDump();  
    InputStream inputStream = new ByteArrayInputStream(new byte[] {0x25, 0x50, 0x44, 0x46, 0x2d, 0x31, 0x2e, 0x37, 0x0d,  
0x0a, 0x25, (byte) 0xb5, (byte) 0xb5, (byte) 0xb5, (byte) 0xb5, 0x0d});  
    StringWriter writer = new StringWriter();  
  
    hexDump.dump(inputStream, writer);  
  
    assertEquals("2550 4446 2d31 2e37 0d0a 25b5 b5b5 b50d  %PDF-1.7..%.....", writer.toString());  
}
```

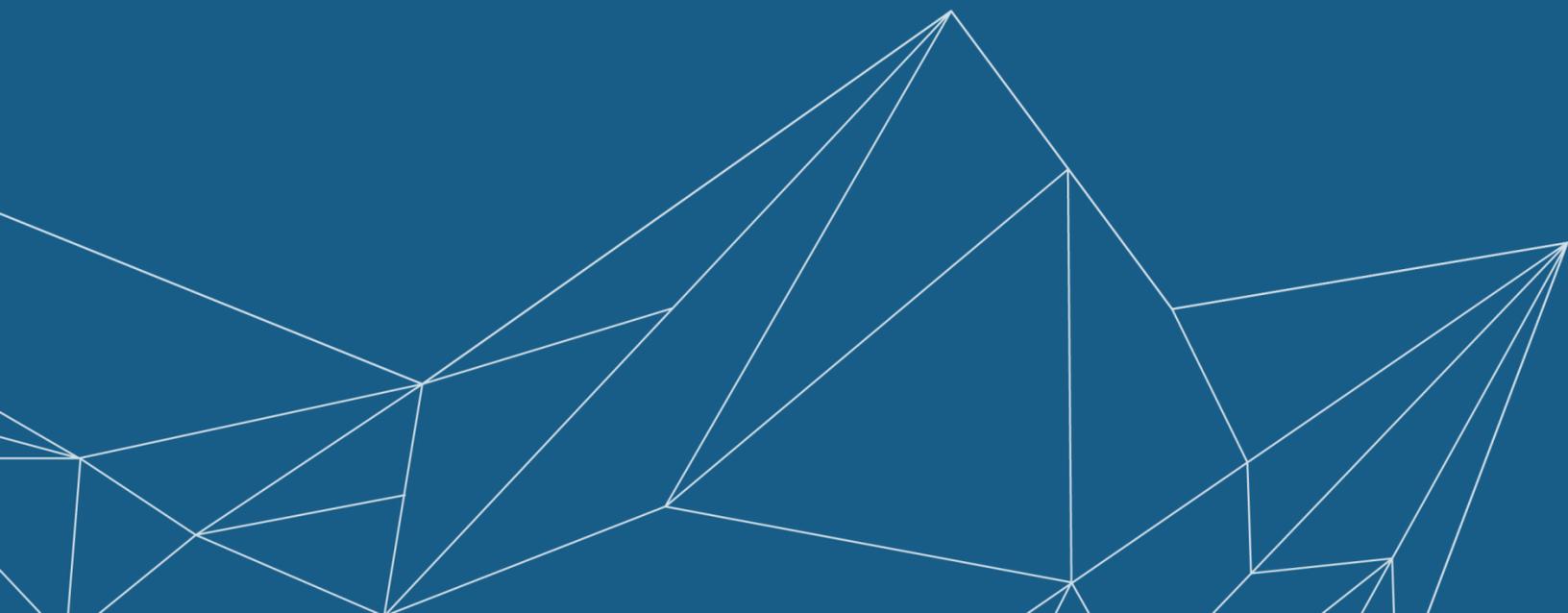
```
@Test
```

```
void testMultipleLines() throws IOException {  
    HexDump hexDump = new HexDump();  
    InputStream inputStream = new ByteArrayInputStream(new byte[] {  
        0x25, 0x50, 0x44, 0x46, 0x2d, 0x31, 0x2e, 0x37, 0x0d, 0x0a, 0x25, (byte) 0xb5, (byte) 0xb5, (byte) 0xb5,  
(byte) 0xb5, 0x0d,  
        0x0a, 0x31, 0x20, 0x30, 0x20, 0x6f, 0x62, 0x6a, 0x0d, 0x0a, 0x3c, 0x3c, 0x2f, 0x54, 0x79, 0x70  
    });  
    StringWriter writer = new StringWriter();  
  
    hexDump.dump(inputStream, writer);  
  
    assertEquals("""  
        2550 4446 2d31 2e37 0d0a 25b5 b5b5 b50d  %PDF-1.7..%.....  
        0a31 2030 206f 626a 0d0a 3c3c 2f54 7970  .1 0 obj..<</Typ""", writer.toString());  
}
```





References



References

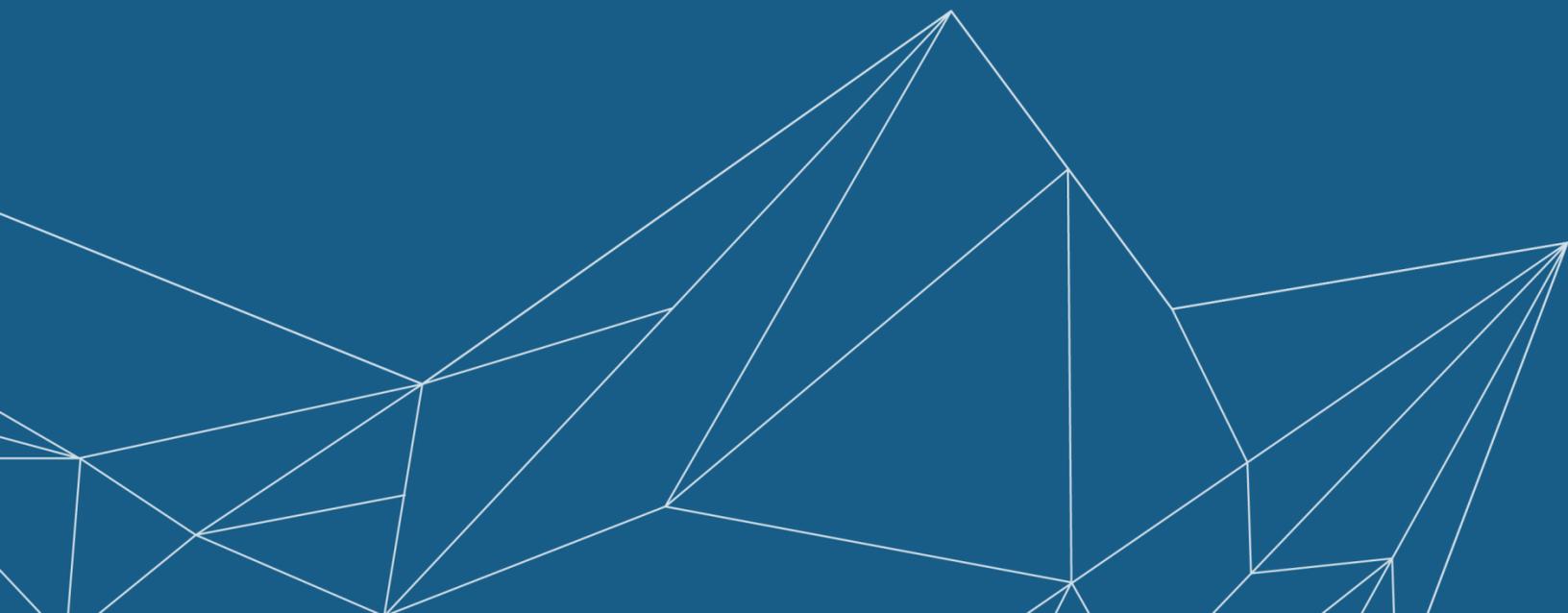
Historically the Java Input/Output library has been developed in two separate packages, while there is some overlap in terms of functionalities the implementation completely diverged, both implementations are still in use, the one we have just introduced is in the `java.io` package, the new one is in the `java.nio` package.

See <https://docs.oracle.com/en/java/javase/25/core/java-nio.html> for an introduction to the new API





Self assessment



Quiz 1 – Input and output streams

1. What is the return value of `InputStream.read()` when the end of stream is reached?
 - A. `0`
 - B. `-1`
 - C. `null`
 - D. `255`
2. Which of the following statements is TRUE about `InputStream.read()`?
 - A. *It reads characters using UTF-8 by default*
 - B. *It never blocks*
 - C. *It requires a buffer to function*
 - D. *It reads binary data one byte at a time*
3. In the `InputStream` hierarchy, which is an abstract class?
 - A. `InputStream`
 - B. `FileInputStream`
 - C. `ByteArrayInputStream`
 - D. `FilterInputStream`



Quiz 1 – Input and output streams

4. **What does try-with-resources ensure for streams?**
 - A. *They close automatically after use*
 - B. *They convert bytes to chars*
 - C. *They enable buffering*
 - D. *They suppress all exceptions*

5. **What is the main benefit of a `BufferedInputStream`?**
 - A. *It converts bytes to characters*
 - B. *It reads data in larger chunks to improve efficiency*
 - C. *It encrypts the data*
 - D. *It avoids blocking*

6. **A `BufferedInputStream` is:**
 - A. *An `InputStream` wrapping another input stream*
 - B. *A standalone input class*
 - C. *A text reader*
 - D. *A file handler*



Quiz 1 – Input and output streams

7. When does `BufferedOutputStream` send data to the underlying stream?

- A. Only when closed
- B. Randomly
- C. When its buffer is full OR when `flush()` is called
- D. Only at program termination

8. Why is buffering useful?

- A. It reduces the number of costly I/O operations on file system or network sources
- B. It increases memory usage dramatically
- C. It guarantees correct encoding
- D. It prevents all I/O exceptions

9. What part of `int b` does `OutputStream.write(b)` use?

- A. All 32 bits
- B. The 24 low-order bits
- C. The 8 low-order bits only
- D. None of the above



Quiz 1 – Input and output streams

10. What does `OutputStream` NOT do?

- A. Write bytes
- B. Convert integers to UTF-8 characters automatically
- C. Throw `IOException`
- D. Support `write(byte[])`

11. Closing an `OutputStream`:

- A. Is always optional and unnecessary
- B. Deletes the underlying file
- C. Flushes the stream and releases resources (especially with `try-with-resources`)
- D. Converts all data to text

12. Which method forces pending bytes to be written out?

- A. `push()`
- B. `clear()`
- C. `sync()`
- D. `flush()`



Quiz 2 – Data streams

1. `DataInputStream` and `DataOutputStream` are mainly used for:
 - A. *Writing text files*
 - B. *Handling primitive Java types in binary form*
 - C. *Encrypting files*
 - D. *Logging*
2. Which method exists in `DataInputStream`?
 - A. `readUTF()`
 - B. `readString()`
 - C. `readText()`
 - D. `readLine(String charset)`
3. `DataOutputStream.writeInt()` writes:
 - A. *A textual number*
 - B. *A single byte*
 - C. *Four bytes in binary format*
 - D. *A UTF-8 encoded string*



Quiz 3 – Readers and writers

1. **A Reader reads:**
 - A. *Bytes*
 - B. *Unicode code units (UTF-16)*
 - C. *ASCII only*
 - D. *Objects*
2. **What does InputStreamReader do?**
 - A. *Writes characters*
 - B. *Reads binary data*
 - C. *Encodes text to binary*
 - D. *Converts bytes to characters using a charset*
3. **BufferedReader improves performance by:**
 - A. *Removing white spaces*
 - B. *Reading large blocks at once*
 - C. *Encoding data*
 - D. *Caching characters permanently*



Quiz 3 – Readers and writers

4. Which Reader allows reading whole lines easily (e.g. `readLine()`)?

- A. `InputStream`
- B. `FileReader`
- C. `StringWriter`
- D. `BufferedReader`

5. `Writer.write(int c)` writes:

- A. *Four bytes*
- B. *The lower 16 bits as a character*
- C. *A UTF-8 sequence*
- D. *Nothing*

6. Writers must be closed because:

- A. *They otherwise can leak file descriptors and may not flush remaining data*
- B. *They auto-close automatically*
- C. *They store encryption keys*
- D. *They never flush automatically*



Quiz 4 – Console I/O

1. `System.out` is a(n):
 - A. `Reader`
 - B. `PrintStream`
 - C. `Writer`
 - D. `FileWriter`
2. Why must you NOT close `System.in`?
 - A. *It is buffered*
 - B. *It is automatically closed*
 - C. *It corrupts console encoding*
 - D. *It is a shared global resource used by the whole JVM*
3. Which method formats text in `Console`?
 - A. `format()` / `printf()`
 - B. `printFormatted()`
 - C. `writeFormat()`
 - D. `printTemplate()`



Correct answers

Quiz 1

1. B
2. D
3. A
4. A
5. B
6. A
7. C
8. A
9. C
10. B
11. C
12. D

Quiz 2

1. B

2. A
3. C

Quiz 3

1. B
2. D
3. B
4. D
5. B
6. A

Quiz 4

1. B
2. D
3. A





Thank you!

esteco.com

