Java IOs

RES, Lecture 1

Olivier Liechti

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

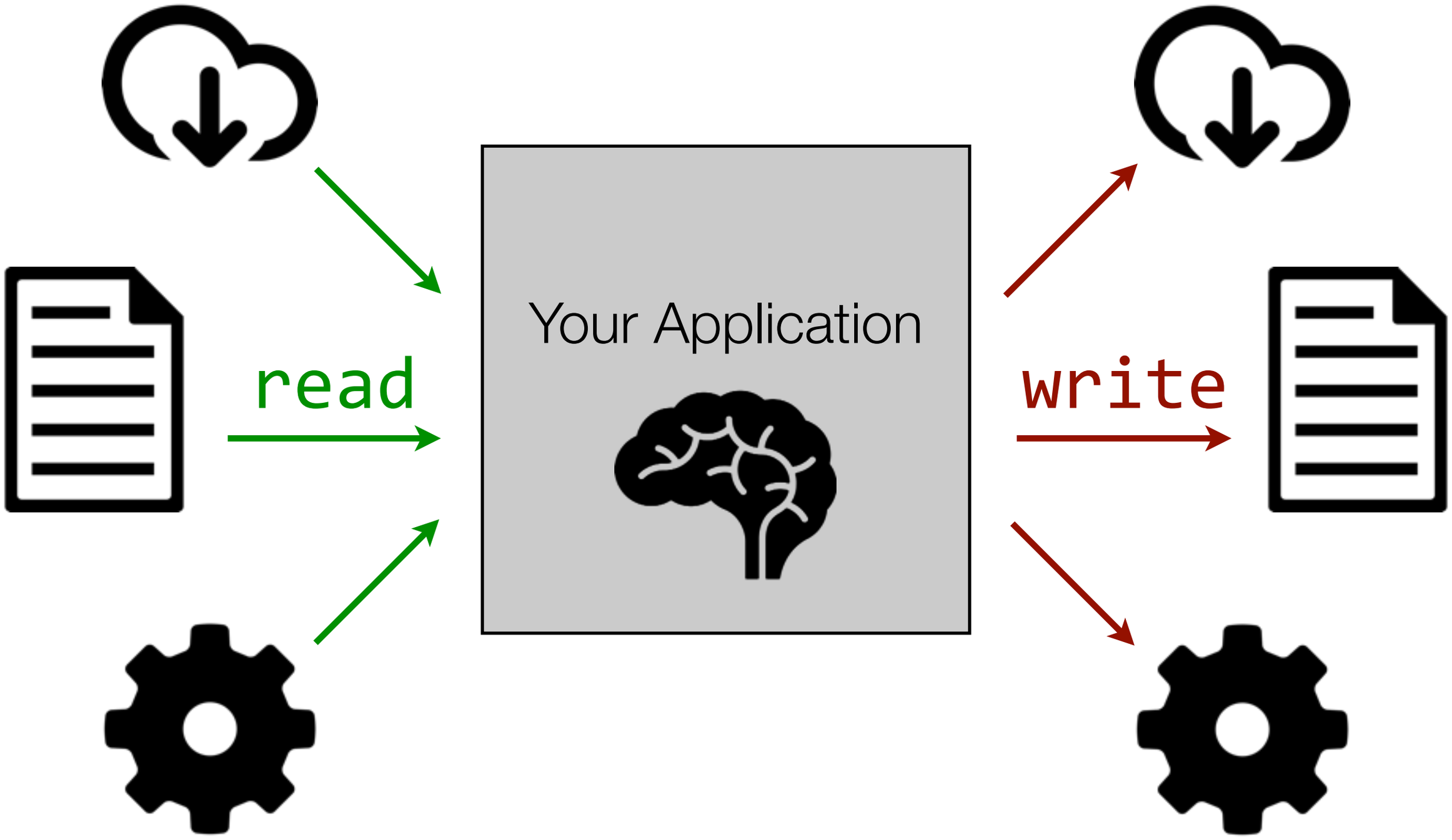
Agenda

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- **Week 1**
  - Universal API
  - Sources, Sinks and Streams
  - The Decorator Pattern and The Mighty Filter Classes
- **Week 2**
  - Performance and Buffering
  - Binary vs. Character-Oriented IOs
  - Shit Happens... Dealing with IO Exceptions

Universal API

What do we mean by IO?



A Universal API

*At the end of the day, whether you are "talking" to a **file**, to a **network endpoint** or to a **process** does not matter.*

*You are always doing the same thing: **reading** and/or **writing** bytes or characters.*

*The Java IO API is the **toolbox** that you need for that purpose.*

Sources, Sinks & Streams

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

System.out.println("I like IO");

out

```
public static final PrintStream out
```

The "standard" output stream. This stream is already open and ready to accept output data. Typically this stream corresponds to display output or another output destination specified by the host environment or user.

For simple stand-alone Java applications, a typical way to write a line of output data is:

```
System.out.println(data)
```

See the `println` methods in class `PrintStream`.

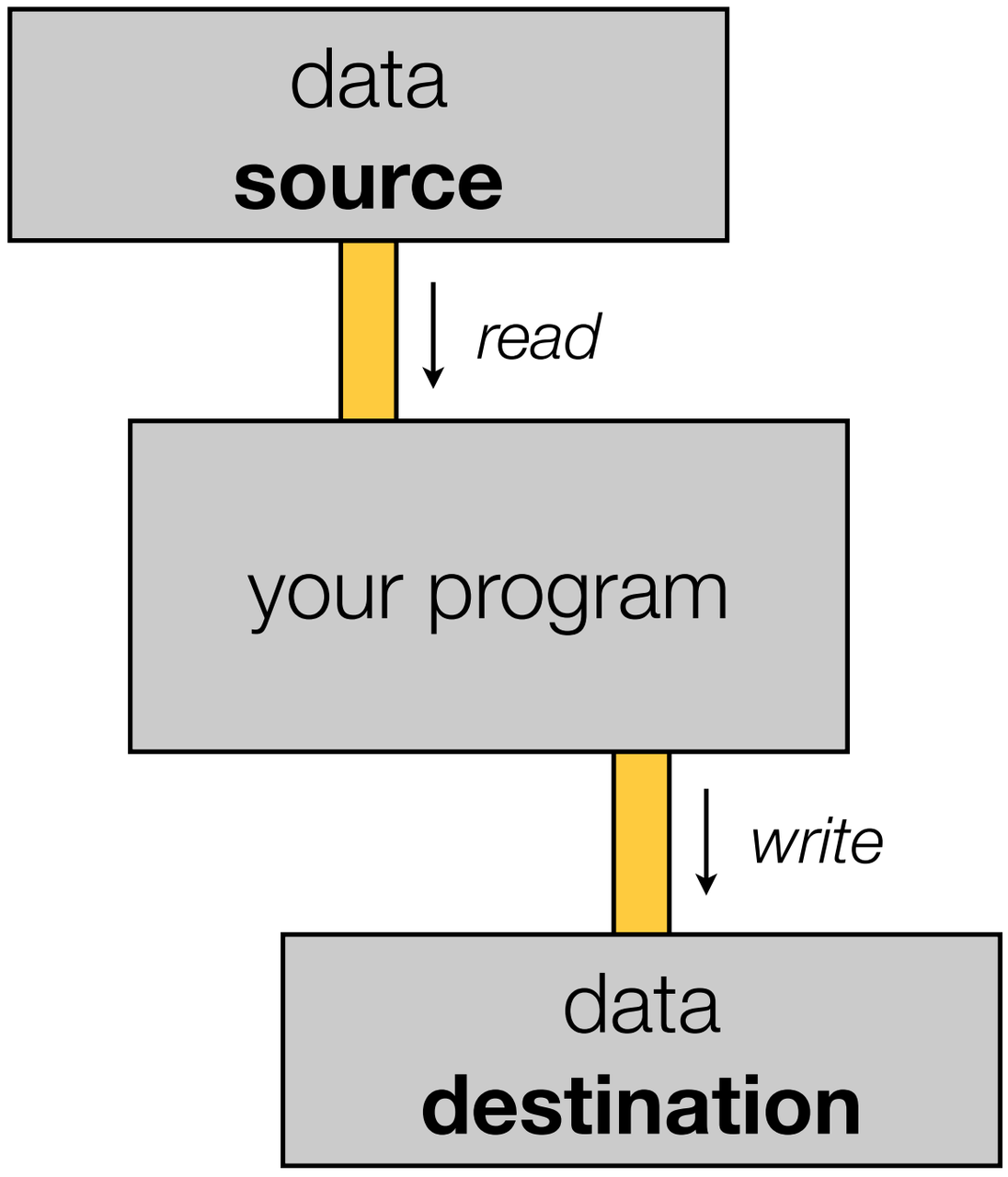
See Also:

`PrintStream.println()`, `PrintStream.println(boolean)`, `PrintStream.println(char)`, `PrintStream.println(char[])`, `PrintStream.println(double)`, `PrintStream.println(float)`, `PrintStream.println(int)`, `PrintStream.println(long)`, `PrintStream.println(java.lang.Object)`, `PrintStream.println(java.lang.String)`

Streams

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- When we talk about “input/output”, or IOs, we think about **producing** and **consuming streams of data**.
- There are different **sources** that contain or produce data.
- There are also different **destinations** that receive or consume data.
- Think about **files**, **network endpoints**, **memory**, **processes**, etc.
- Your **program** can **read data** from a stream. Your program can **write data** to a stream.



What can you do with IO streams?

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

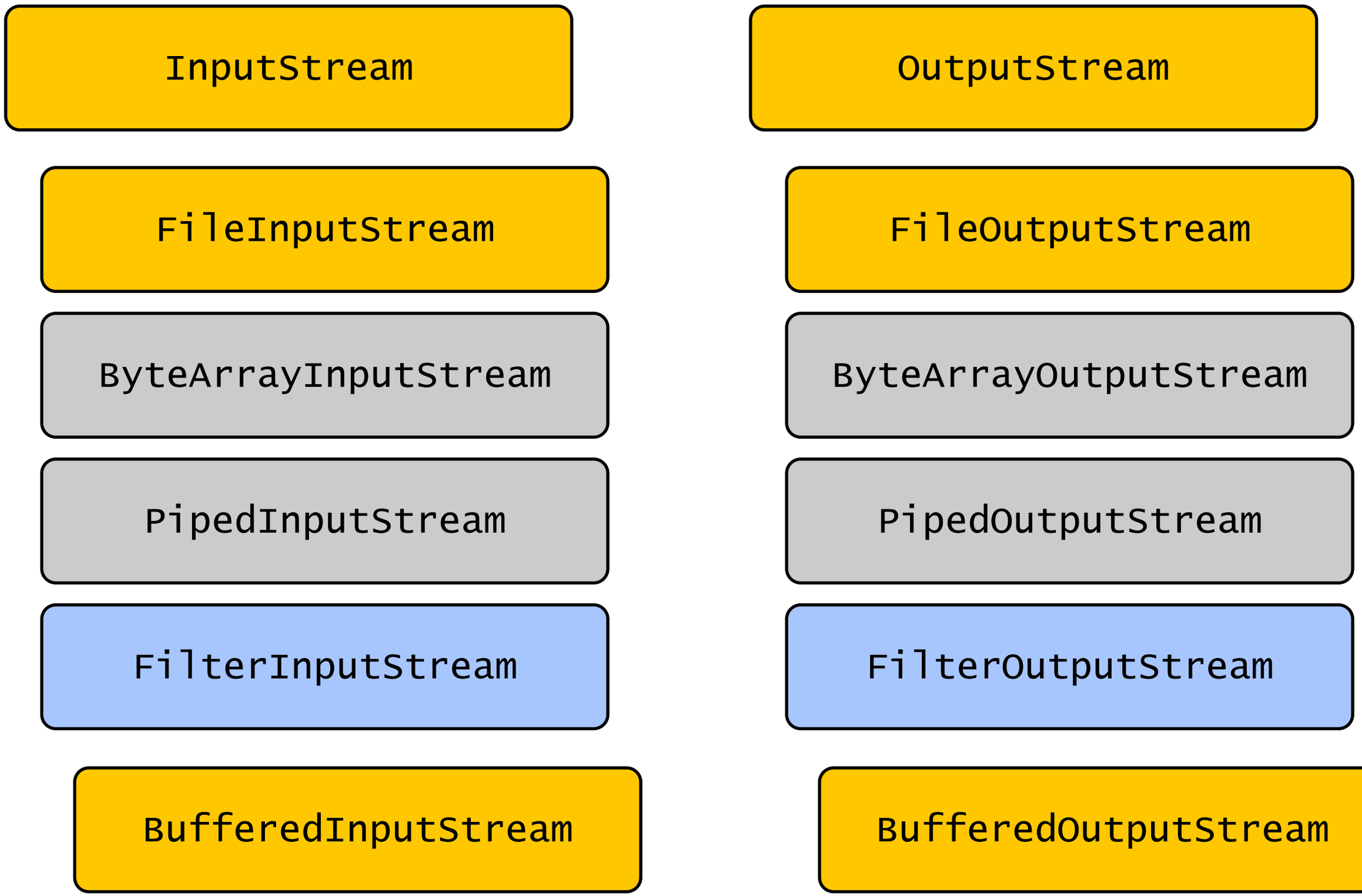
- **Read data from a file**. Think about a program that reads an XML configuration file at start-up. Think about your favorite text editor.
- **Write data to a file**. Think about a java compiler that produces .class files.
- **Receive data over the network**. Think about a web server receiving requests from clients.
- **Send data over the network**. Think about the same server sending responses to the clients.
- **Exchange data with other programs** (processes) running on the same machine.
- **Etc.**

In the end, it's **always the same thing**... it's about **reading** and **writing** data!

If it's the same thing, then we want to have a **single API** for dealing with all sorts of IOs!

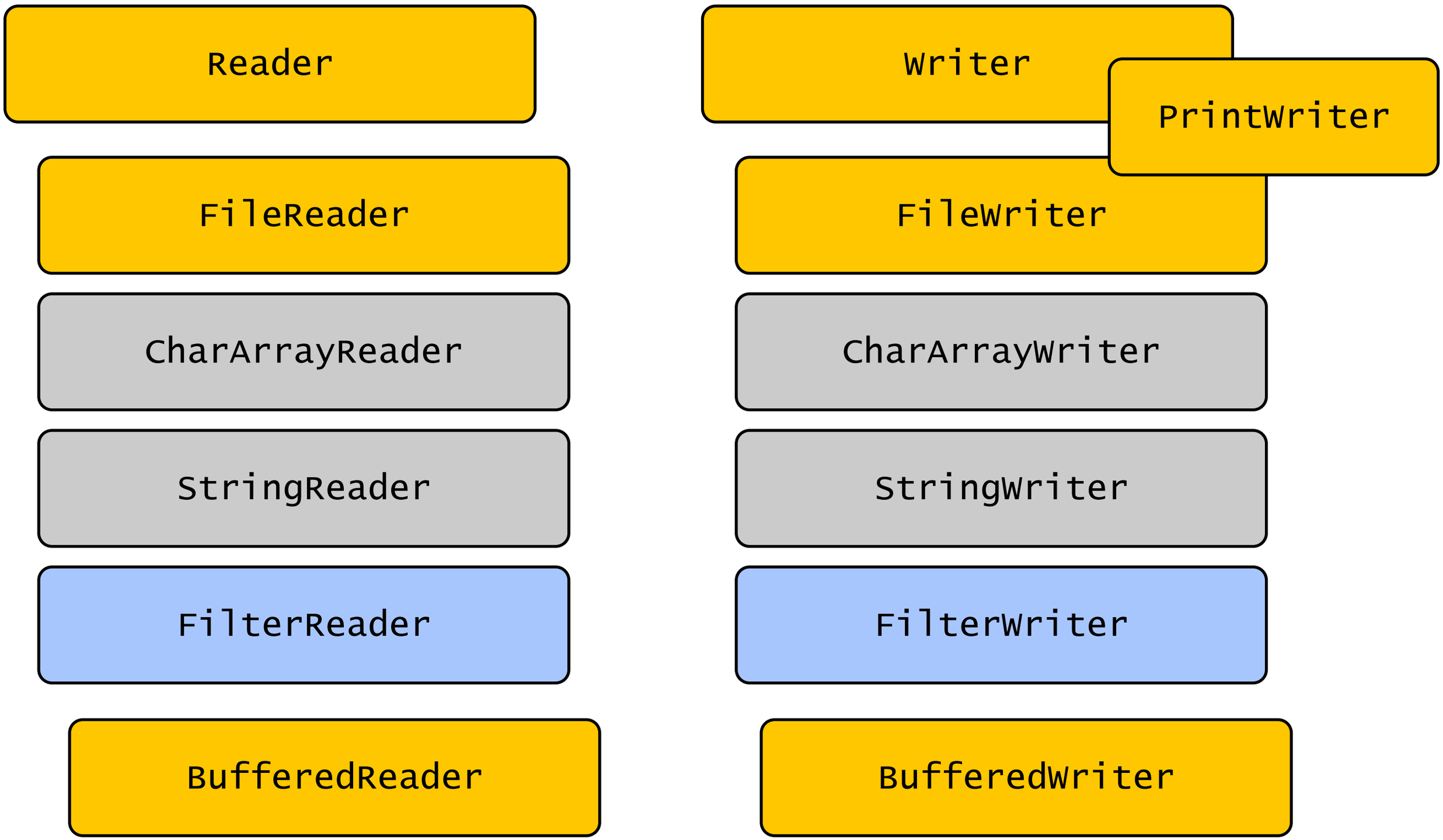
Classes in the `java.io` package (1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud



Classes in the `java.io` package (2)

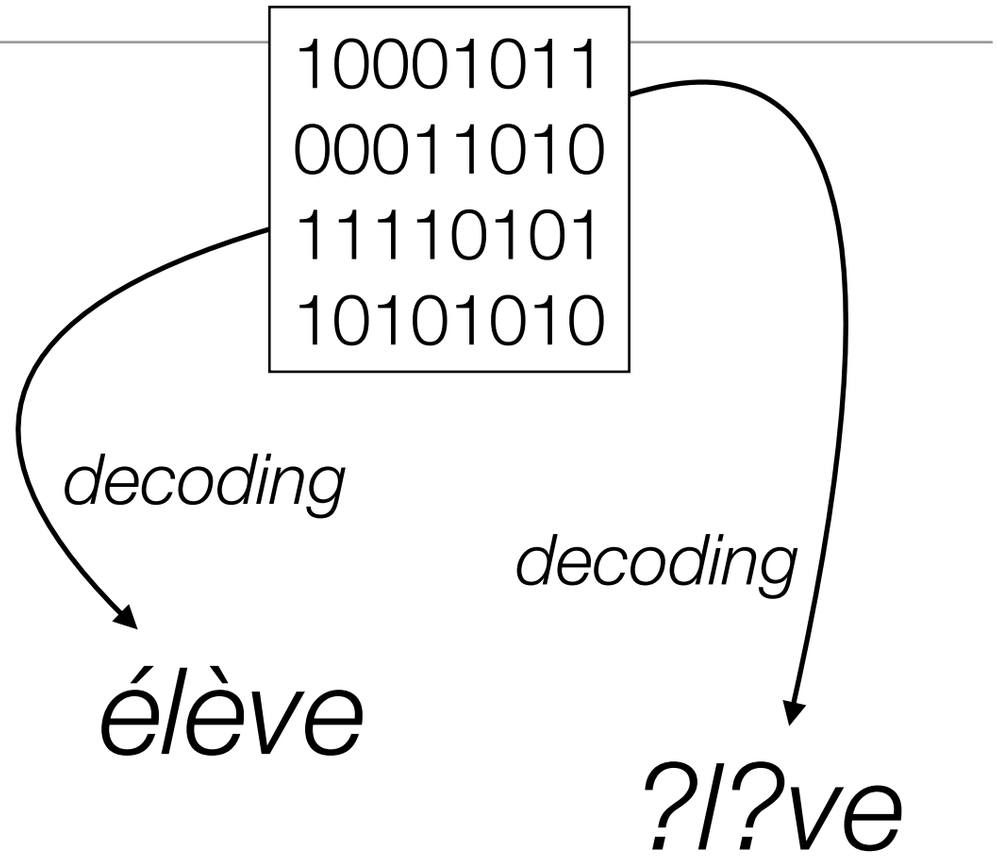
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Bytes vs. Characters

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- There are different types of data sources. Some sources produce **binary data**. Other sources produce **textual data**.
- It's always a **series of 0's and 1's**. The real question is : "how do you interpret these bits"?
- When you deal with **textual data**, the interpretation is not always the same. It depends on the "**character encoding system**"?
- When you deal with IOs, you have to **use different classes** for processing binary, respectively textual data. **Otherwise, you will corrupt data**!



For binary data, use **inputStreams** and **outputStreams**.
For text data, use **readers** and **writers**.

Reading Bytes, One at The Time

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
int b = fis.read();
while ( b != -1 ) {

  // b has an int value between -1 and 255
  // -1 indicates that we are at the end of the stream
  // b can be casted to a byte, remember that in Java,
  // bytes are signed and have values in the range [-128..127].

  b = fis.read();
}
```

Or if you absolutely want to save 1 line...

```
int b;
while ( (b = fis.read()) != -1 ) {

}
```

Reading Bytes, in Blocks

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
final int BUFFERSIZE = 255;
byte[] buffer = new byte[BUFFERSIZE];

int numberOfNewBytes = fis.read(buffer);

  // we know that numberOfNewBytes bytes have been read
  // we know that these bytes are available in the buffer
  // WARNING: there might be left-over junk after these bytes!

while ( numberOfNewBytes != -1 ) {

  // do something with the bytes in buffer[0..numberOfNewBytes-1]
  // ignore what is left in buffer[numberOfNewBytes.. BUFFERSIZE]

  // read the next chunk of bytes
  numberOfNewBytes = fis.read(buffer);

}
```

Writing Bytes

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Write 1 byte

```
OutputStream os = ...;
int b;
// The byte to be written is the eight low-order b.
// The 24 high-order bits of b are ignored. So, b can have an
// int value in the range [0..255]
os.write(b);
```

Write a block of bytes

```
OutputStream os = ...;
byte[] data = new byte[BLOCK_SIZE];

data[3] = 22; data[4] = 5; data[5] = 9; data[6] = 7;
// data[0..2] contains junk, data[7..BLOCK_SIZE-1] too

int offset = 3; // because we have started to fill at slot 3
int length = 4; // because we have filled 4 slots
os.write(data, offset, length);
```

Design **Your Code** to Be Universal

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
/**
 * This interface will work only for data sources on the file
 * system. In the method implementation, I would need to create
 * a FileInputStream from f and read bytes from it.
 */
 public interface IPoorlyDesignedService {
  public void readAndProcessBinaryDataFromFile(File f);
}
```

data source

VS

```
/**
 * This interface is much better. The client using the service
 * has a bit more responsibility (and work). It is up to the
 * client to select a data source (which can still be a file,
 * but can be something else). The method implementation
 * will ignore where it is reading bytes from. Nice for reuse,
 * nice for testing.
 */
public interface INicelyDesignedService {
  public void readAndProcessBinaryData(InputStream is);
}
```

stream

Example: **02-FileIOExample** (1)

The Mighty Filter Classes

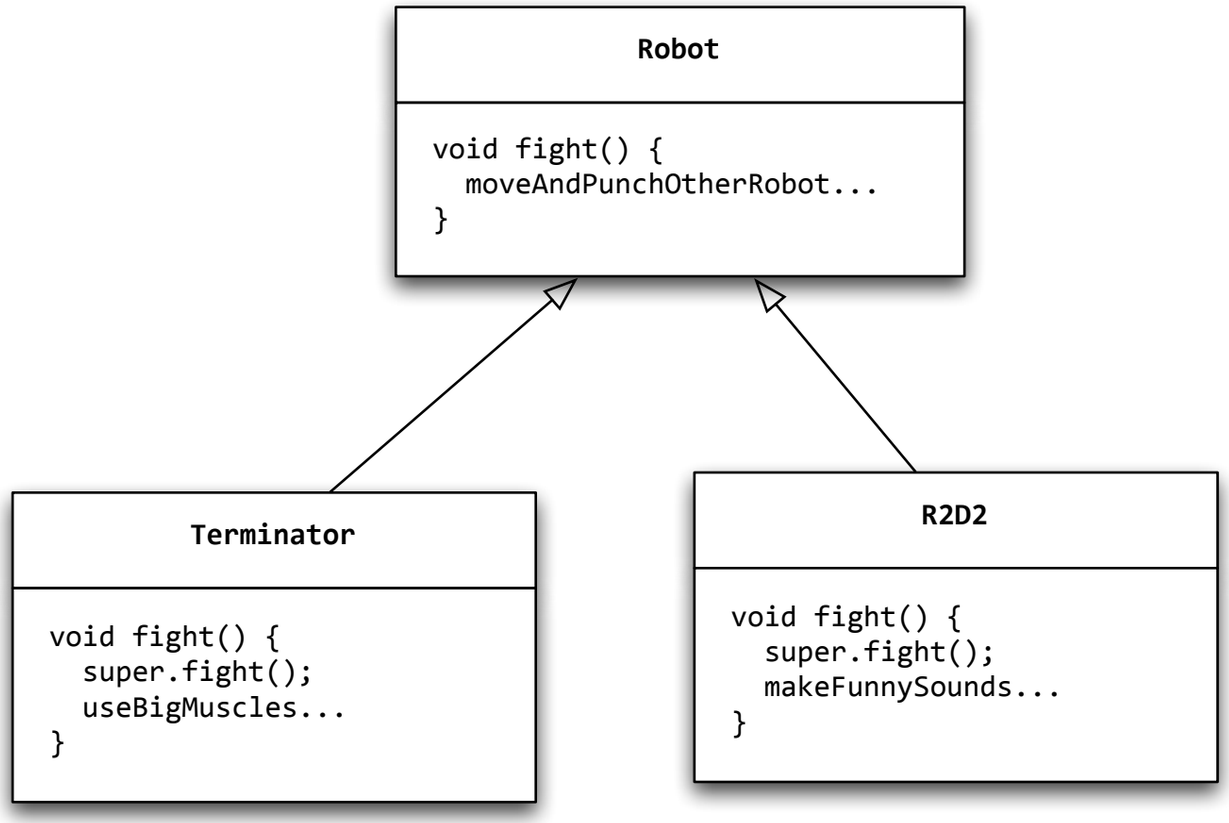
The Decorator Design Pattern

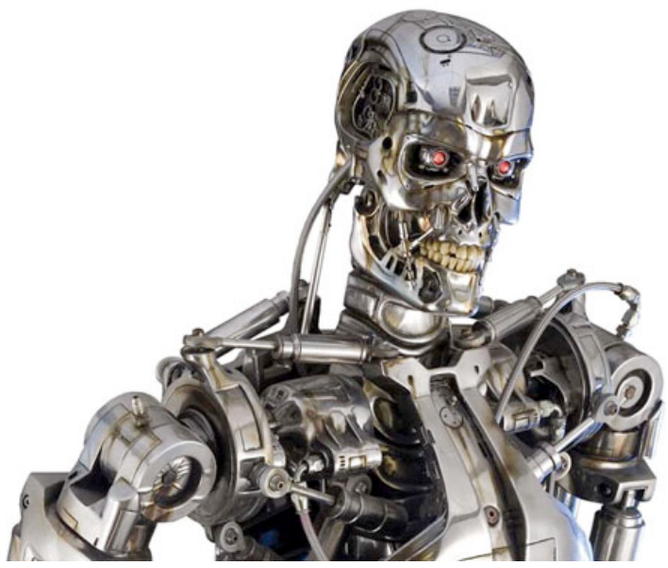
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

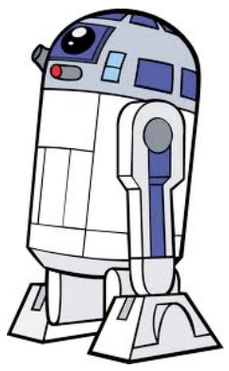
- The **decorator design pattern** is a solution that is often used when creating object-oriented models.
- It makes it possible to **add behavior to an existing class**, without modifying the code of this class. In other words, it makes it possible to **decorate** an existing class with additional behaviors.
- The design pattern also makes it possible to **define a collection of decorators**, and to **combine** them in arbitrary ways at runtime. In other words, it is possible to decorate a class with several behaviors.

Example : Fighting Robots

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



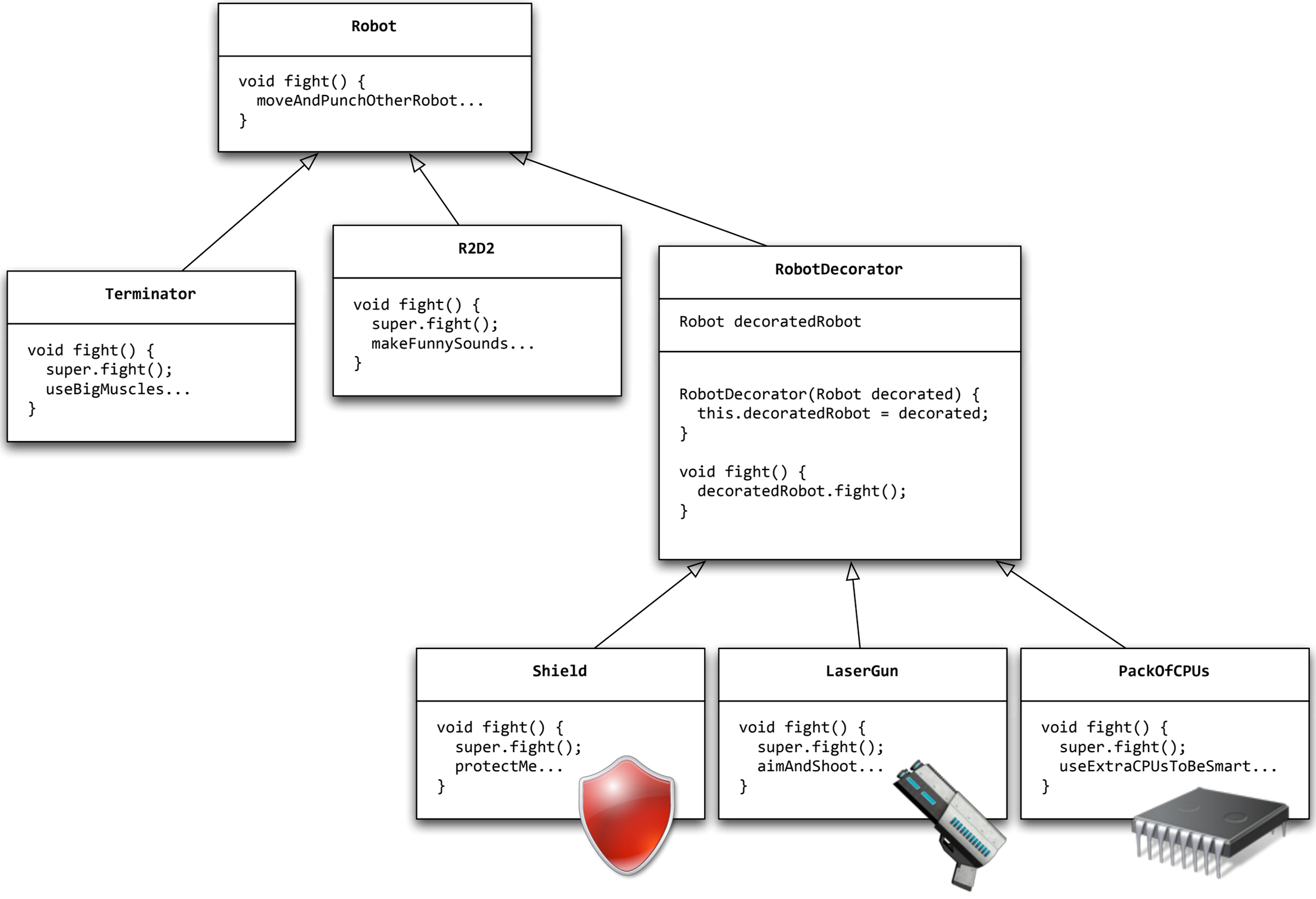




```
Robot bigOne = new Terminator();
Robot smallOne = new R2D2();

bigOne.fight();
smallOne.fight();

--
bigOne   > useBigMuscles
smallOne > makeFunnySounds

bigOne wins.
```

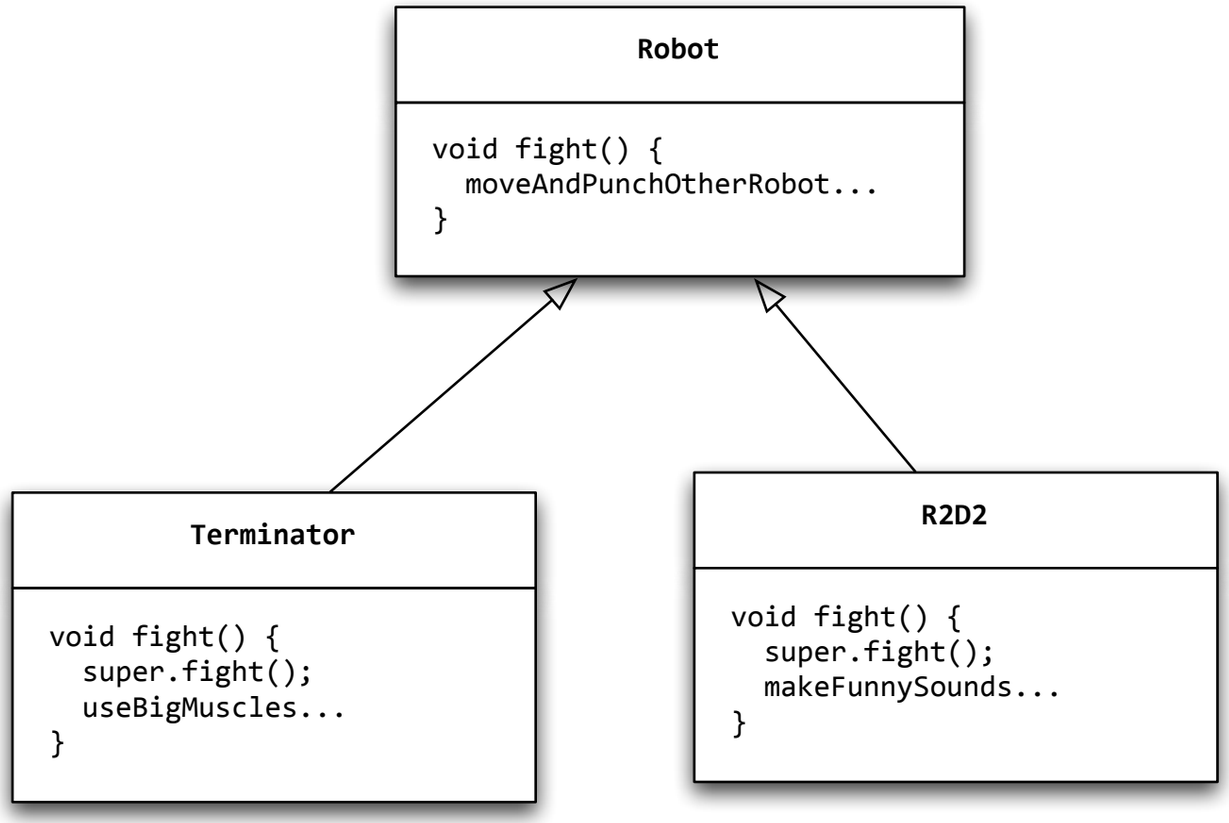
Example : Decorators for Robots

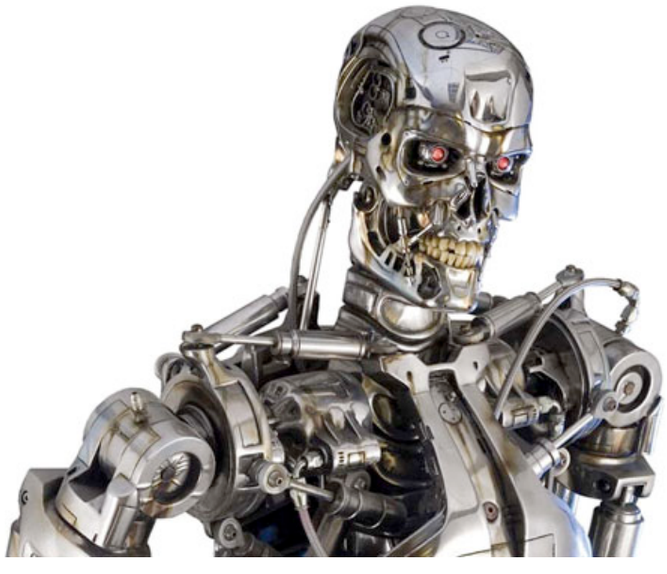
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Example : Decorated Robots

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

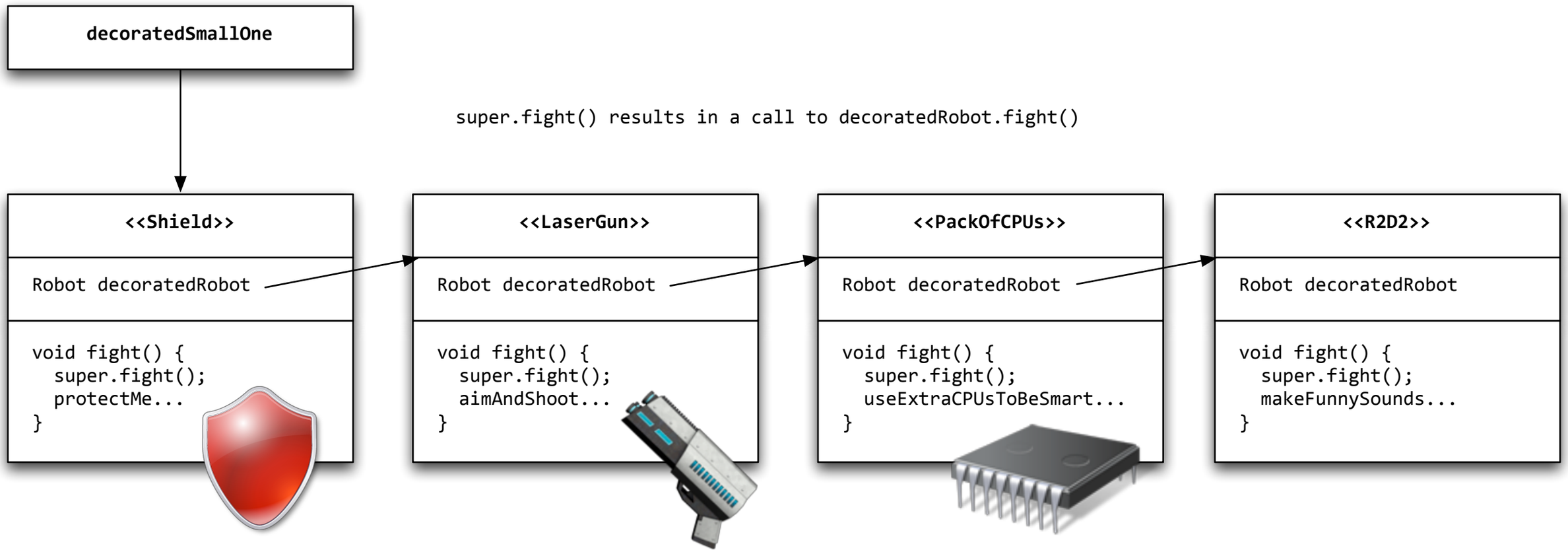




```
Robot bigOne = new Terminator();
Robot decoratedSmallOne =
  new Shield(
    new LaserGun(
      new PackOfCPU(
        new R2D2()
      )
    )
  );

bigOne.fight();
decoratedSmallOne.fight();

--
bigOne    > useBigMuscles
smallOne > makeFunnySounds,
useExtraCPUsToBeSmart, aimAndShoot,
protectMe

decoratedSmallOne wins.
```

Invocation Chain

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud




```
decoratedSmallOne.fight()
-> shield.fight();
-> laserGun.fight() + protectMe;
-> packOfCPUs.fight() + aimAndShoot + protectMe
-> R2D2.fight() + useExtraCPUsToBeSmart + aimAndShoot + protectMe
-> makeFunnySounds + useExtraCPUsToBeSmart + aimAndShoot +
protectMe.
```

Example: **02-FileIOExample** (2)

Performance and Buffering

Buffered IOs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Do you know what happens when you do a

```
int c = read();
```

?

Buffered IOs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

It’s a bit like when you are thirsty and feel like **drinking** something...

!

Buffered IOs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

It takes you 56' to sip a beer

20 min

5 min

10 min

20 min

1 min

Buffered IOs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Thirsty again...

?

Buffered IOs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

It takes you 56' again to sip the next beer

20 min

5 min

10 min

20 min

1 min

Buffered IOs

Can we do better...

?

Buffered IOs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

It still takes you 56' to bring back a pack of beers...

20 min

5 min

10 min

20 min

1 min

Buffered IOs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Thirsty again...

?

Buffered IOs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

It now only takes 2 minutes to sip the next one!

1 min

1 min

Buffered IOs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Coming back to

```
int c = read();
```

- If you don’t use buffered IOs, calling read() will issue **one system call** to retrieve **one single byte**... which is not efficient.

- With buffered IOs, calling read() will **pre-fetch “several” bytes** and store it in a **temporary memory space** (i.e. in a **buffer**). “several” defines the **buffer size**.

- Subsequent calls to read() will be able to **fetch bytes directly from the buffer**, which is very fast.

What about

```
write(c);
```

?

Buffered IOs

It’s the same thing! There is on gotcha:

Sometimes, you want to immediately send the content of the buffer to the output stream.

```
os.flush();
```

Buffered IOs in Java

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- Remember the **Decorator** Design Pattern?
- Using buffered IOs is **as simple as decorating any of your byte or character streams** (don't forget about flushing buffered output streams when required!).

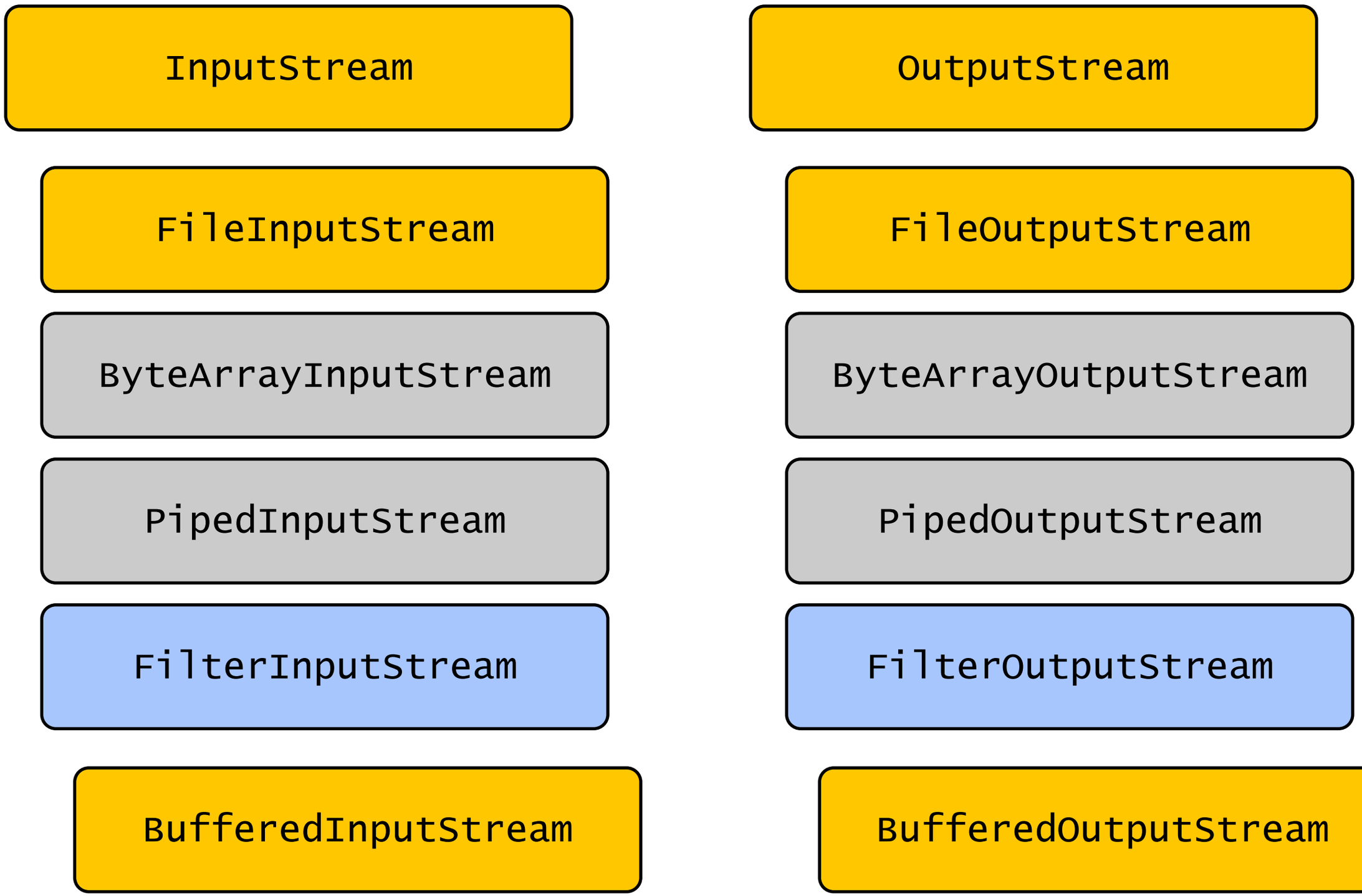
```
InputStream slow;
BufferedInputStream fast = new BufferedInputStream(slow);

OutputStream slow;
BufferedOutputStream fast = new BufferedOutputStream(slow);

Reader slow;
BufferedReader fast = new BufferedReader(slow);

Writer slow;
BufferedWriter fast = new BufferedWriter(slow);
```

Example: **01-BufferedIOBenchmark**

Binary vs Character-Oriented IOs

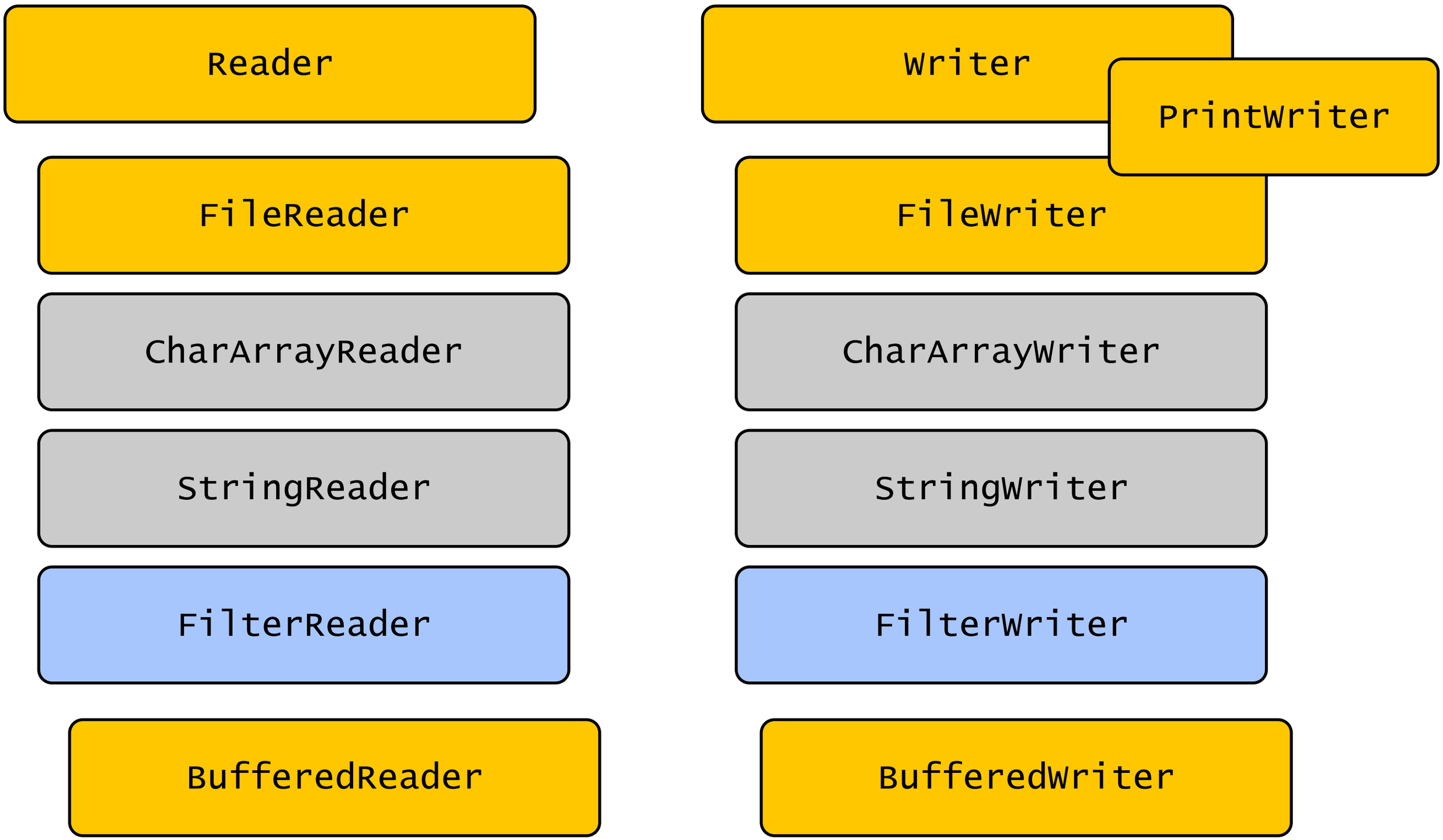
Classes in the `java.io` package (1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud



Classes in the `java.io` package (2)

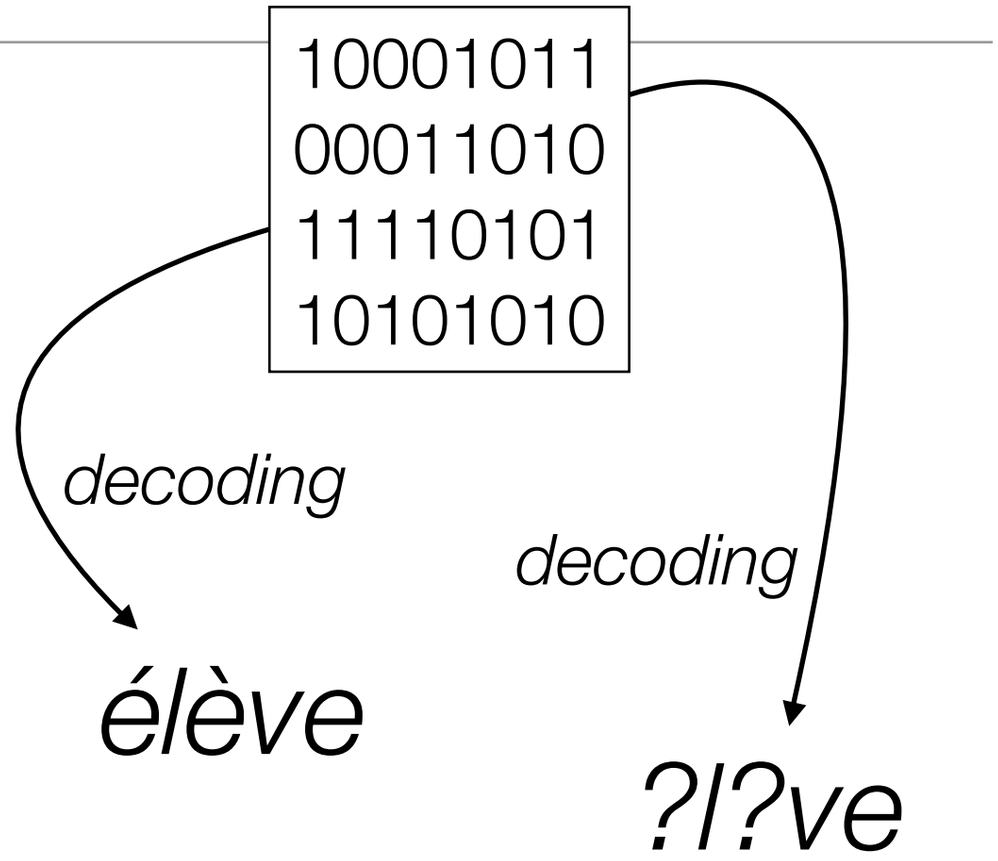
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Bytes vs. Characters

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- There are different types of data sources. Some sources produce **binary data**. Other sources produce **textual data**.
- It's always a **series of 0's and 1's**. The real question is : "how do you interpret these bits"?
- When you deal with **textual data**, the interpretation is not always the same. It depends on the "**character encoding system**"?
- When you deal with IOs, you have to **use different classes** for processing binary, respectively textual data. **Otherwise, you will corrupt data**!



For binary data, use **inputStreams** and **outputStreams**.
For text data, use **readers** and **writers**.

Bytes vs. Characters

“A byte is a sequence of 8 bits. Period.”

1000001 → 1000001 → 1000001

*“A character is the **interpretation** of a sequence of n bits. Producer and consumer need to **agree** on how to do this interpretation.”*

✓ ‘A’ ⇝ 1000001 ⇝ ‘A’

✗ ‘A’ ⇝ 1000001 ⇝ ‘▯’

Example : the ASCII encoding

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

‘A’ 1000001 65
64-32-16-08-04-02-01

USASCII code chart

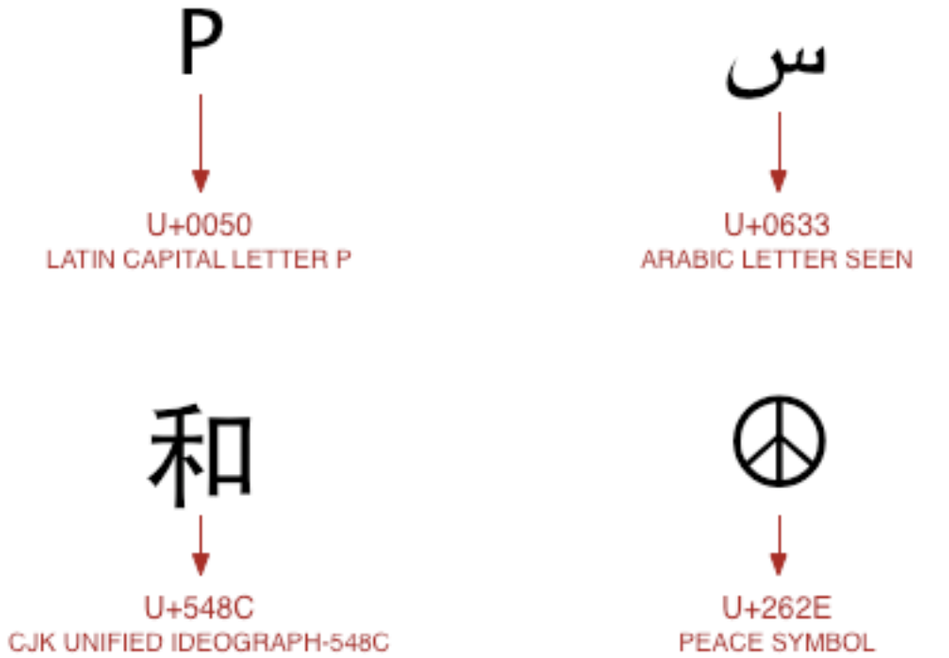
b_7 b_6 b_5 → / b_4 b_3 b_2 b_1 ↓	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0 0 0 0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	STX	DC2	"	2	B	R	b	r
0 0 1 1	ETX	DC3	#	3	C	S	c	s
0 1 0 0	EOT	DC4	$	4	D	T	d	t
0 1 0 1	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	ACK	SYN	&	6	F	V	f	v
0 1 1 1	BEL	ETB	'	7	G	W	g	w
1 0 0 0	BS	CAN	(	8	H	X	h	x
1 0 0 1	HT	EM	)	9	I	Y	i	y
1 0 1 0	LF	SUB	*	:	J	Z	j	z
1 0 1 1	VT	ESC	+	;	K	[	k	{
1 1 0 0	FF	FS	,	<	L	\	l	|
1 1 0 1	CR	GS	-	=	M	]	m	}
1 1 1 0	SO	RS	.	>	N	^	n	~
1 1 1 1	SI	US	/	?	O	_	o	DEL

Column → / Row ↓

Example : Unicode & UTF-8 encoding

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **Unicode** is an industry standard for representing text in almost all world languages (e.g. japanese, hebrew, arabic, etc.).
- With Unicode, **every character is associated with a "code point"**, which is nothing else than a number. It is expressed as 'U+' followed by an **hexadecimal** value.
- For instance, 'A' has the code point **U+41** (41 is 65 in hexadecimal).
- The code points for **latin characters** have the same value as in the ASCII encoding.
- **UTF-8** is **one of the encoding systems** used to represent characters of the Unicode character set.
- UTF-8 is a **variable-length encoding system**. Some characters are encoded with 1 byte, others with 2 bytes, etc.



http://wiki.secondlife.com/wiki/Unicode_In_5_Minutes

Java & Unicode

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- Java uses the **unicode character encoding system**. This means that your program can manipulate characters in different languages and alphabets.
- Every **char** variable is defined by **2 bytes**, i.e. 16 bits. The two bytes
- Think about **what happens when you read data from a source**. You will see a series of 1's and 0's. You know that these bits represent characters, but how do you know how to interpret them? The answer will depend on the source! Or more precisely, **it will depend on the encoding system used by the source**.
- **Same problem when you produce data**. You have text data in memory (char and String variables). You want to understand and control how this data is transformed in a series of bits. This is important if you want that other parties are able to read what you have produced!

Exploring the Unicode Charset

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

http://www.fileformat.info/info/unicode/char/41/index.htm

Unicode Character 'LATIN CAPITAL LETTER A' (U+0041)

A

Browser Test Page
Outline (as SVG file)
Fonts that support U+0041

Unicode Data	
Name	LATIN CAPITAL LETTER A
Block	Basic Latin
Category	Letter, Uppercase [Lu]
Combine	0
BIDI	Left-to-Right [L]
Mirror	N
Index entries	Latin Uppercase Alphabet Uppercase Alphabet, Latin Capital Letters, Latin
Lower case	U+0061
Version	Unicode 1.1.0 (June, 1993)

Encodings	
HTML Entity (decimal)	A
HTML Entity (hex)	A
How to type in Microsoft Windows	Alt +41 Alt 065 Alt 65
UTF-8 (hex)	0x41 (41)
UTF-8 (binary)	01000001
UTF-16 (hex)	0x0041 (0041)
UTF-16 (decimal)	65
UTF-32 (hex)	0x00000041 (41)
UTF-32 (decimal)	65
C/C++/Java source code	"\u0041"
Python source code	u"\u0041"
More...	

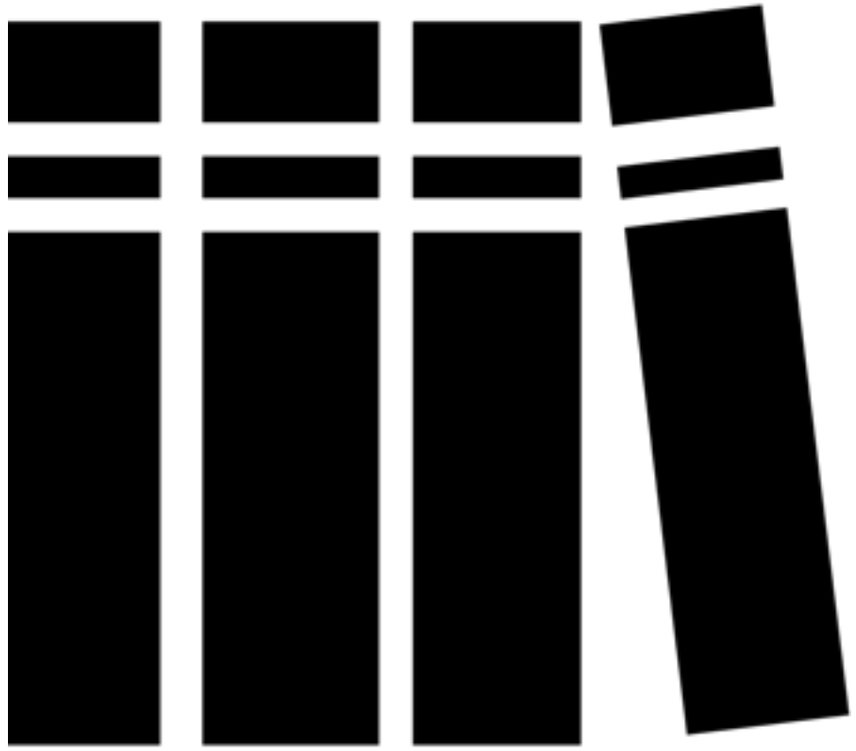
Example: **03-CharacterIODemo**

Shit Happens...
Dealing with IO Exceptions

Exception Summary

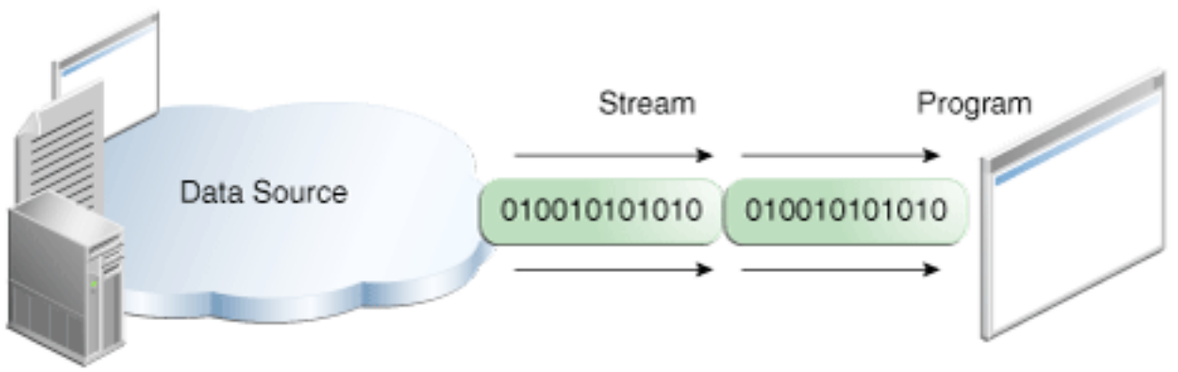
Exception	Description
CharConversionException	Base class for character conversion exceptions.
EOFException	Signals that an end of file or end of stream has been reached unexpectedly during input.
FileNotFoundException	Signals that an attempt to open the file denoted by a specified pathname has failed.
InterruptedIOException	Signals that an I/O operation has been interrupted.
InvalidClassException	Thrown when the Serialization runtime detects one of the following problems with a Class.
InvalidObjectException	Indicates that one or more deserialized objects failed validation tests.
IOException	Signals that an I/O exception of some sort has occurred.
NotActiveException	Thrown when serialization or deserialization is not active.
NotSerializableException	Thrown when an instance is required to have a Serializable interface.
ObjectStreamException	Superclass of all exceptions specific to Object Stream classes.
OptionalDataException	Exception indicating the failure of an object read operation due to unread primitive data, or the end of data belonging to a serialized object in the stream.
StreamCorruptedException	Thrown when control information that was read from an object stream violates internal consistency checks.
SyncFailedException	Signals that a sync operation has failed.
UnsupportedEncodingException	The Character Encoding is not supported.
UTFDataFormatException	Signals that a malformed string in modified UTF-8 format has been read in a data input stream or by any class that implements the data input interface.
WriteAbortedException	Signals that one of the ObjectStreamExceptions was thrown during a write operation.

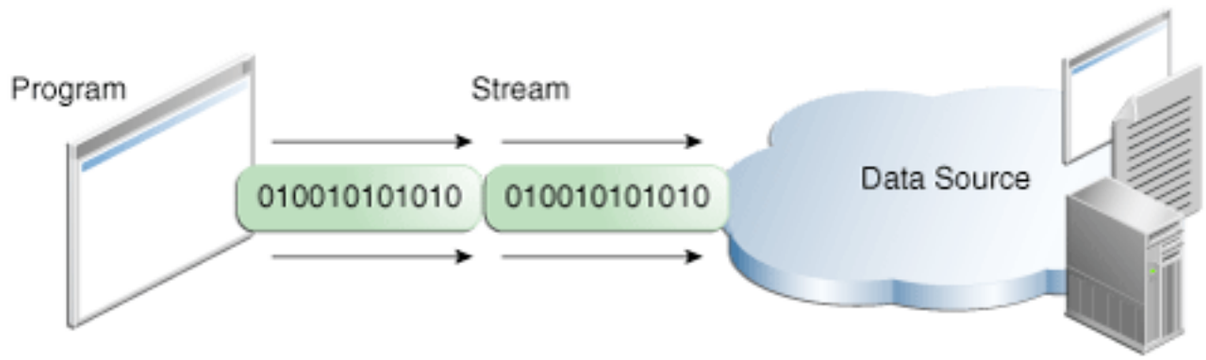
References



The IO Trail in the Java Tutorial

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

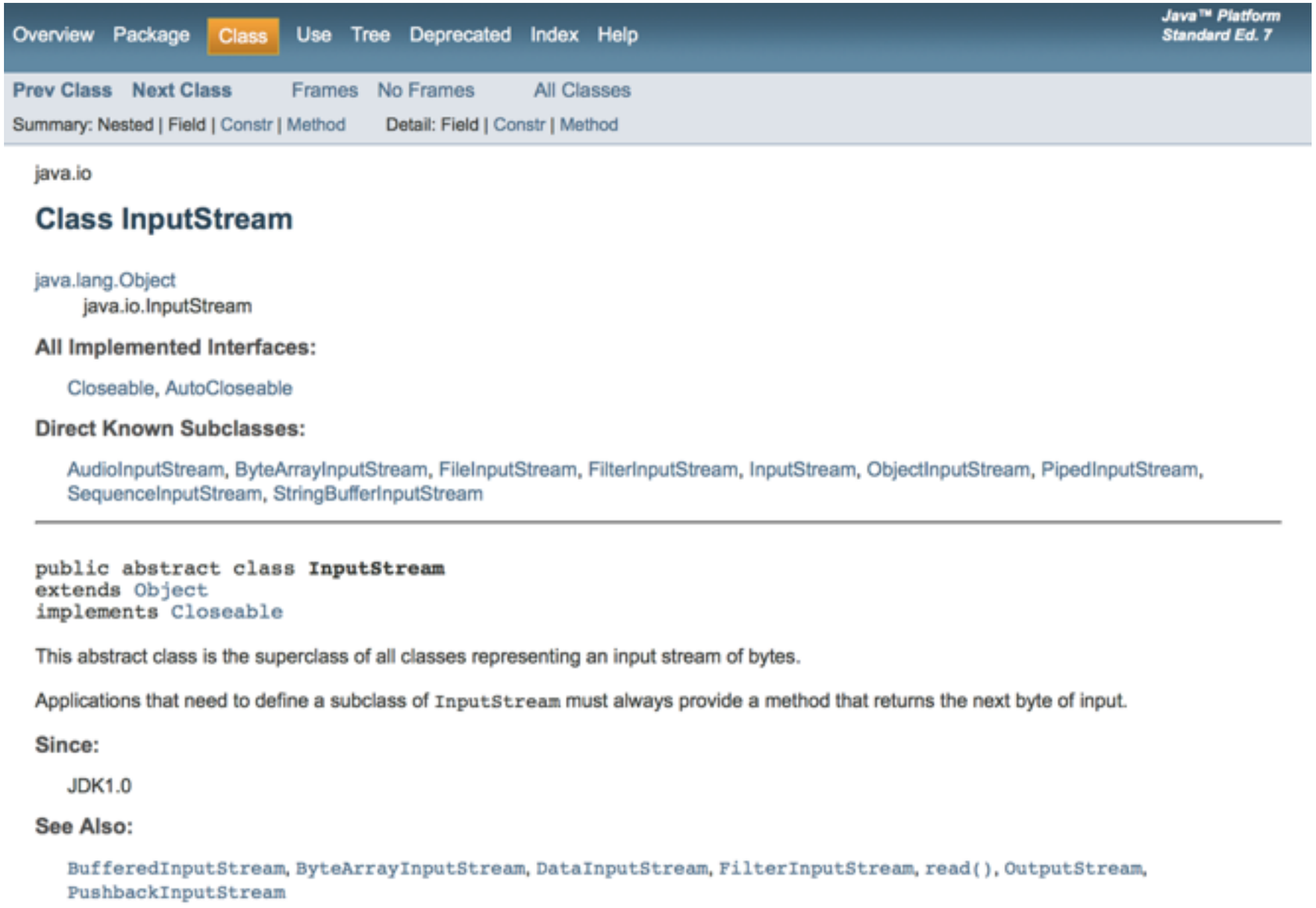




http://docs.oracle.com/javase/tutorial/essential/io/index.html

The `java.io` API documentation

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Overview Package Class Use Tree Deprecated Index Help — Java™ Platform Standard Ed. 7

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.io

Class InputStream

java.lang.Object
    java.io.InputStream

All Implemented Interfaces:

Closeable, AutoCloseable

Direct Known Subclasses:

AudioInputStream, ByteArrayInputStream, FileInputStream, FilterInputStream, InputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream

---

```
public abstract class InputStream
extends Object
implements Closeable
```

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of `InputStream` must always provide a method that returns the next byte of input.

Since:

JDK1.0

See Also:

`BufferedInputStream`, `ByteArrayInputStream`, `DataInputStream`, `FilterInputStream`, `read()`, `OutputStream`, `PushbackInputStream`

http://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html

Books

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud