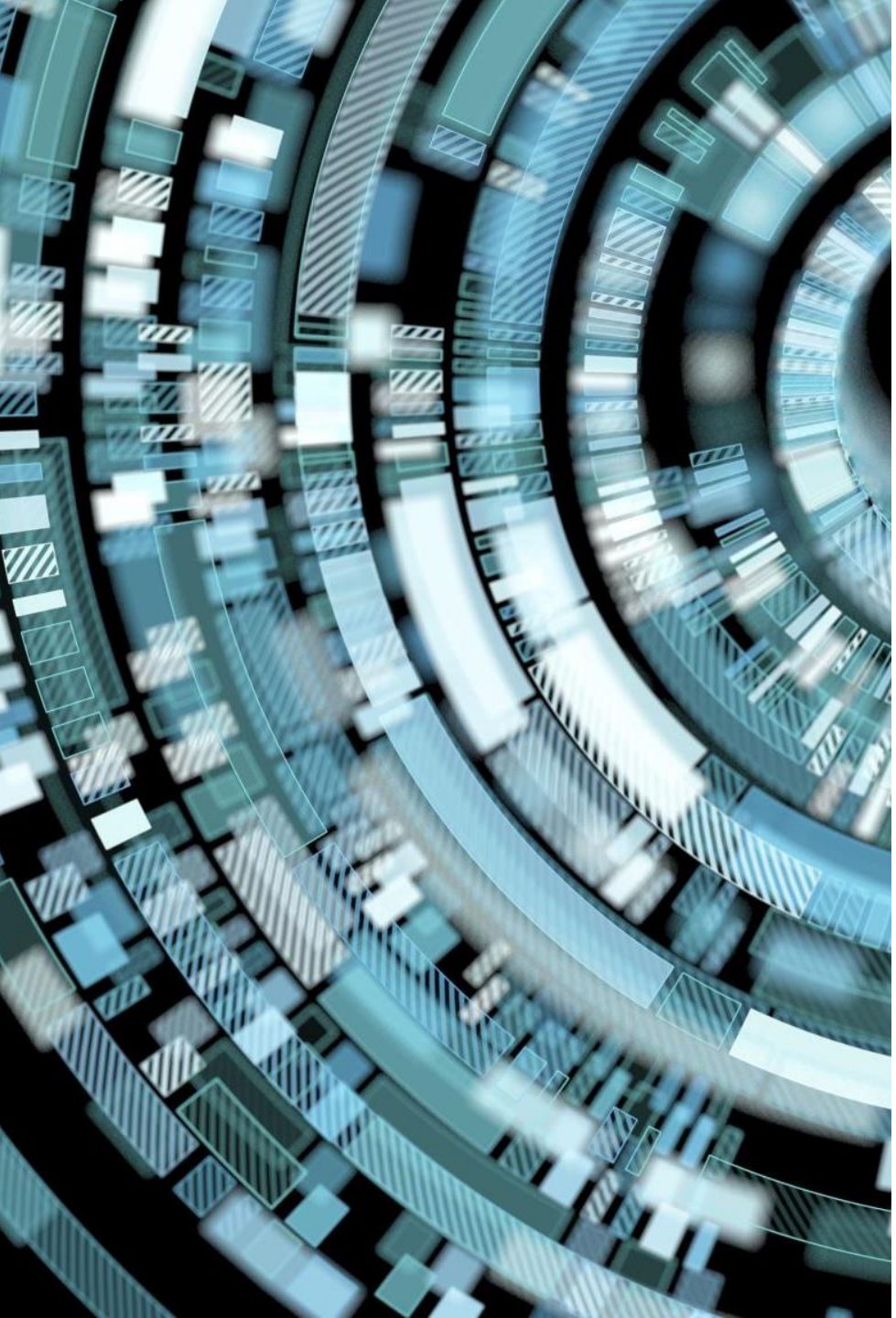
Computer System Design & Application

计算机系统设计与应用A

陶伊达 （TAO Yida）
taoyd@sustech.edu.cn

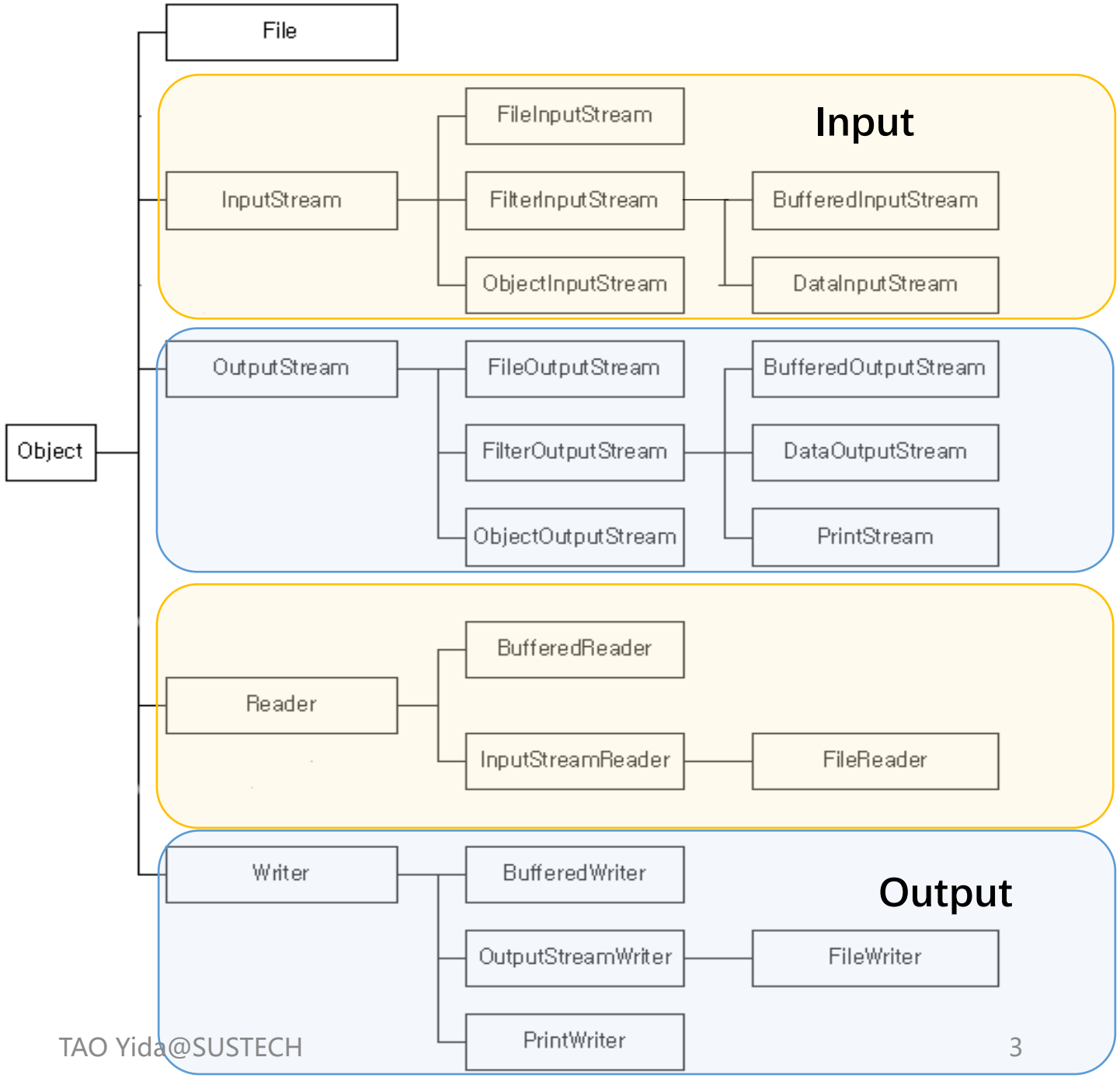


Lecture 5

- I/O Overview
- i18n & Character Encoding
- Byte Streams & Character Streams
- Combining Stream Filters
- Reading/Writing Text Input/Output
- I/O from Command Line

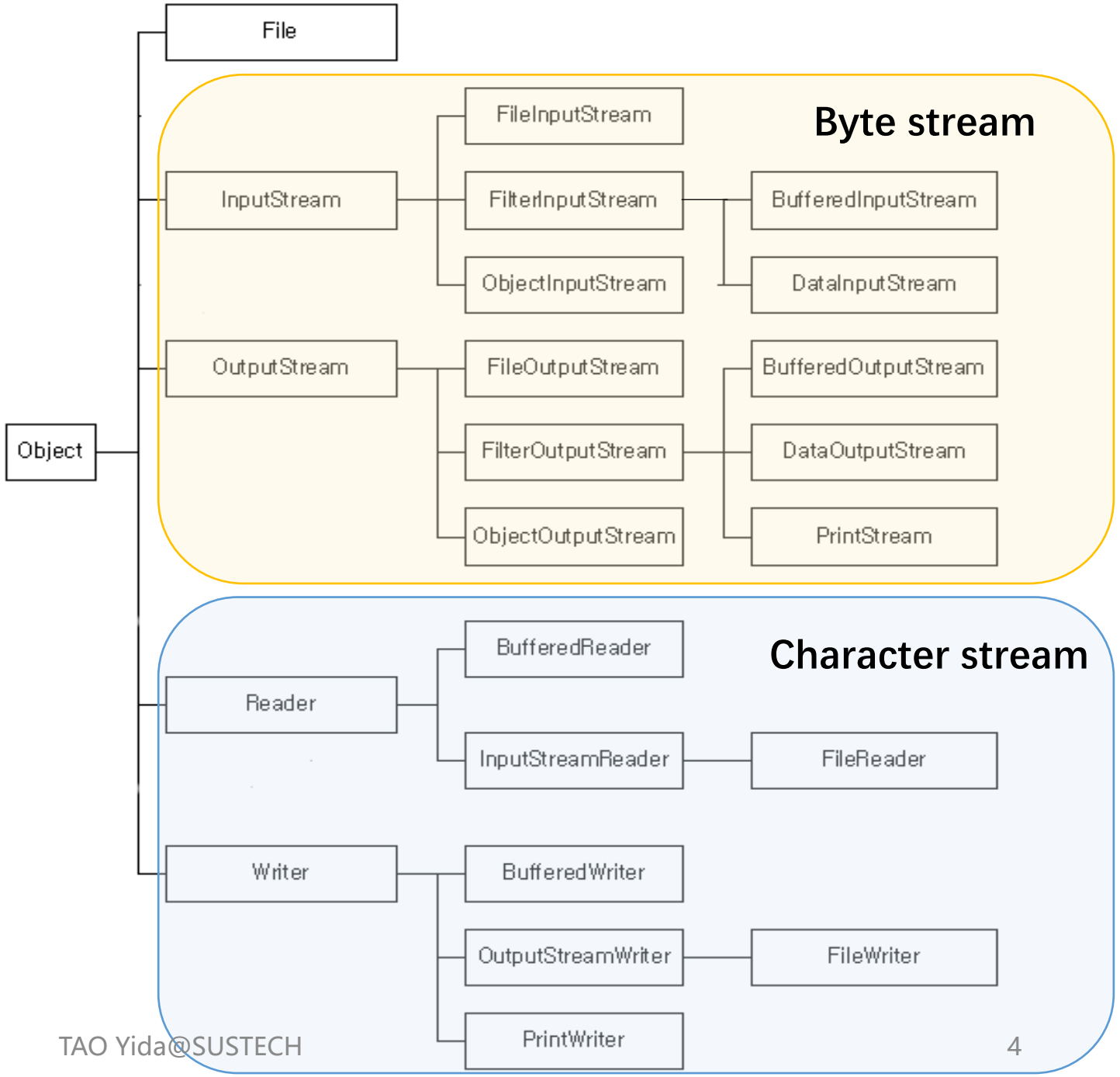
I/O Overview

- Java I/O and File are in `java.io` package
- I/O classification
  - Input and output
  - Byte stream vs Character stream



I/O Overview

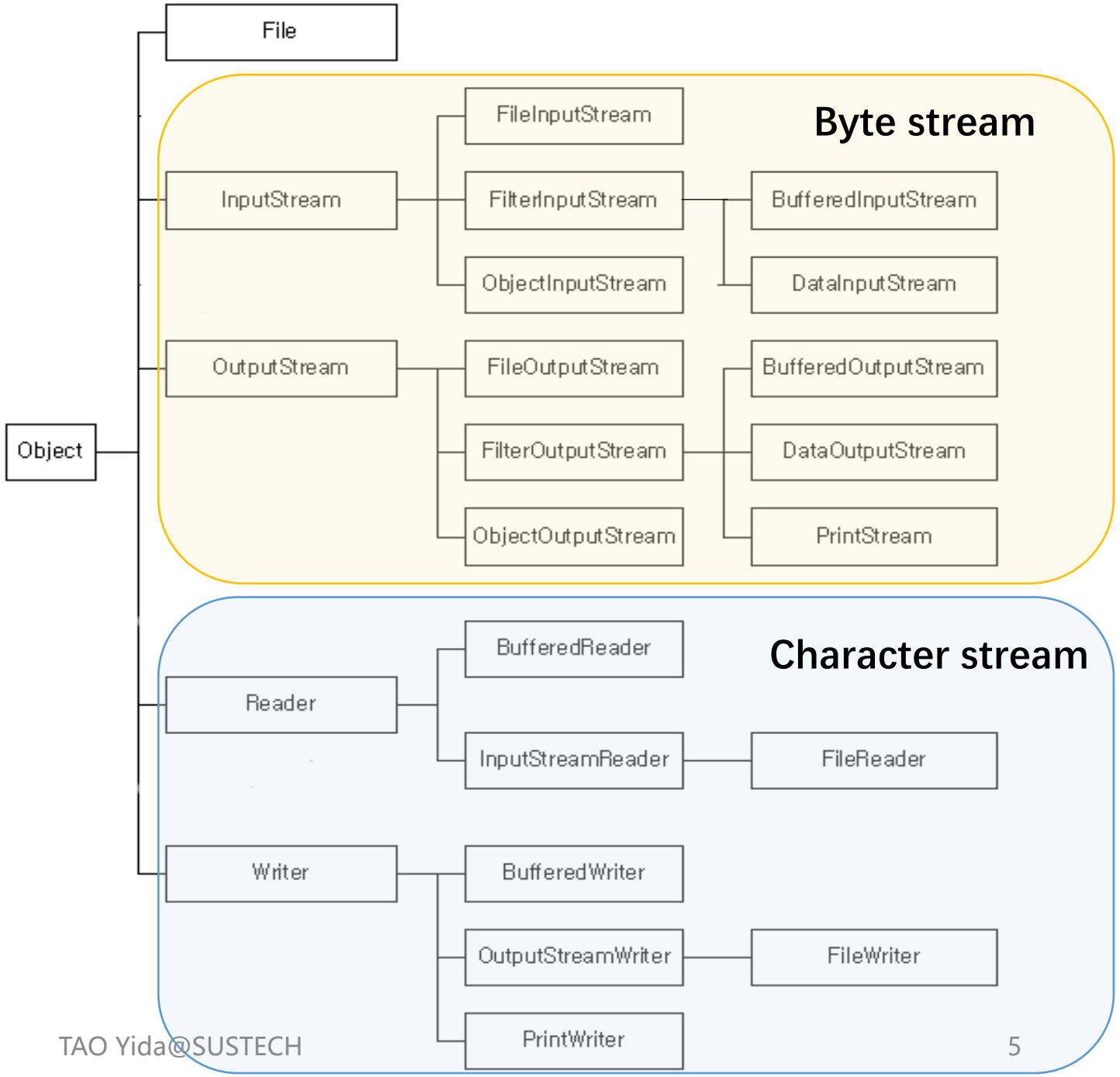
- Java I/O and File are in `java.io` package
- I/O classification
  - Input and output
  - Byte stream vs Character stream



Overview

- Java I/O and File are in java.io package
- I/O classification
  - Input and output
  - Byte stream vs Character stream

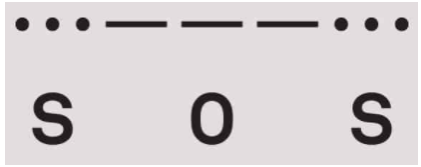
Character Stream is used to handle Internationalization (i18n), where character encoding makes software systems international (language/location independent).



Encoding

Convert characters (字符) to other formats, often numbers, in order to store and transmit them more effectively

The International Morse Code (摩斯电码, 1837) encodes A-Z, numbers, and some other characters.



International Morse Code

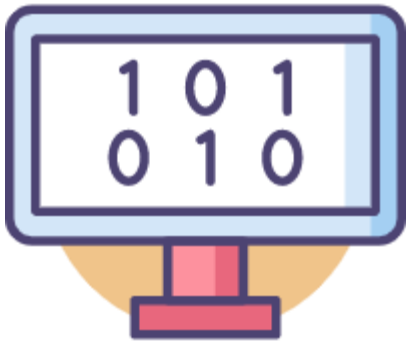
1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

Chinese Telegraph Code (中文电码, 1872)

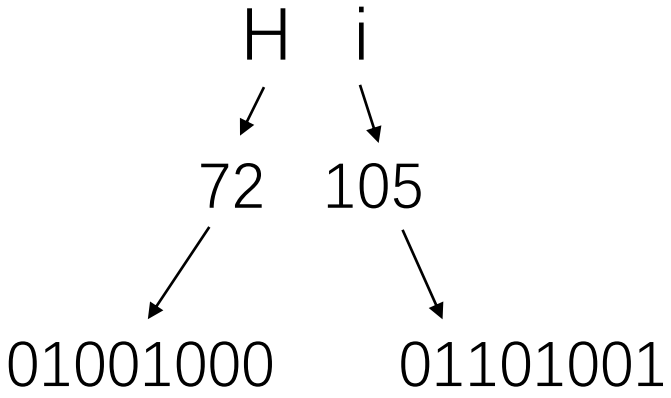
One-to-one mapping between Chinese characters and four-digit numbers from 0000 to 9999

ASCII

- Represent text in computers
- Using 7 bits to represent 128 characters
- Extended ASCII uses 8 bits for 256 characters



Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

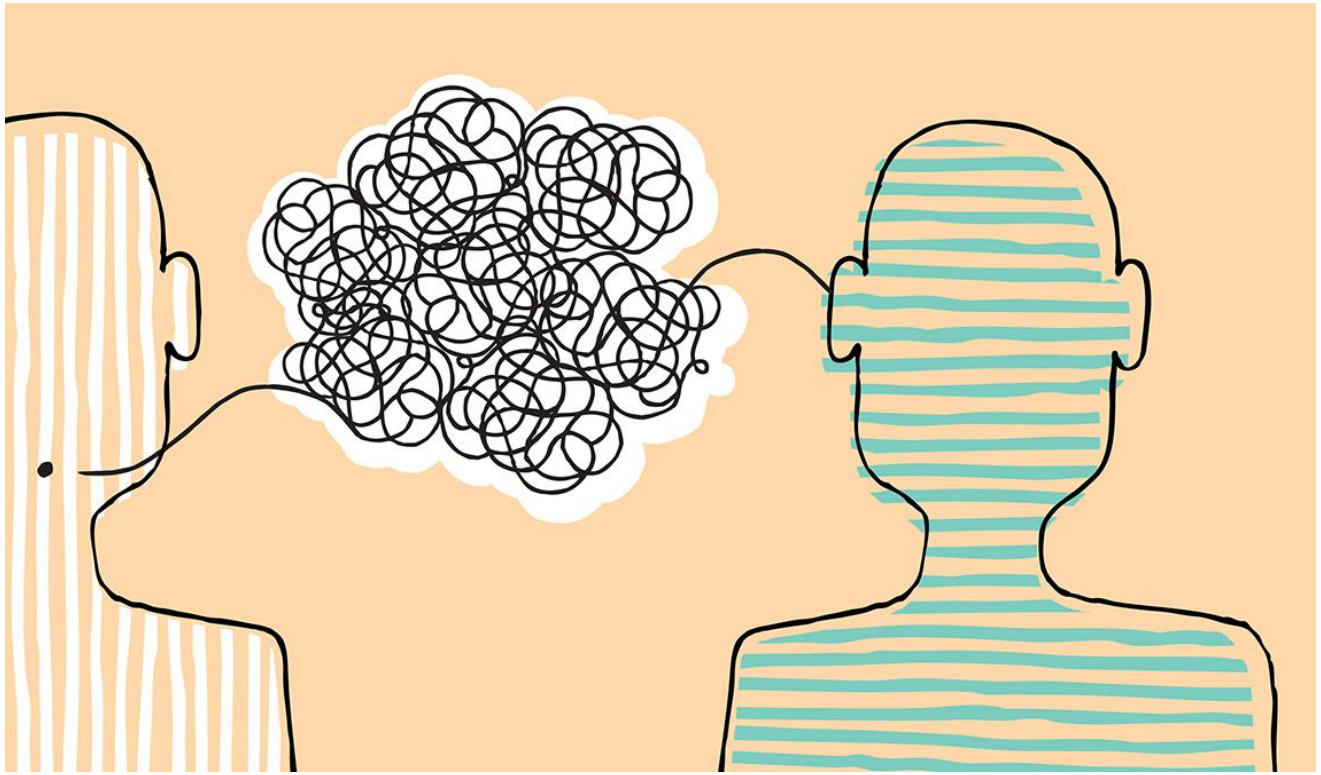


GB2312, GBK, GB18030

- GB stands for 国标
- GB2312 uses 2 bytes (cover 99% daily usages)
- GBK (国标扩展) extends GB2312 to encode more characters
- GB18030 extends GBK

编	码
B1E0	C2EB
1011000111100000	1100001011101011

Problems?



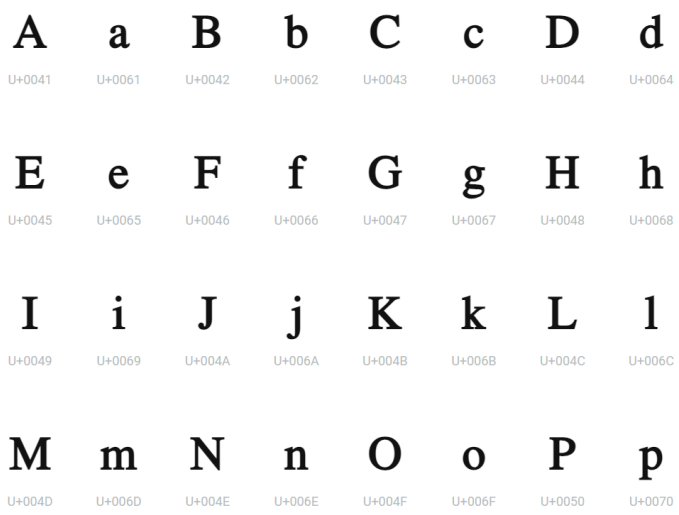
Communication

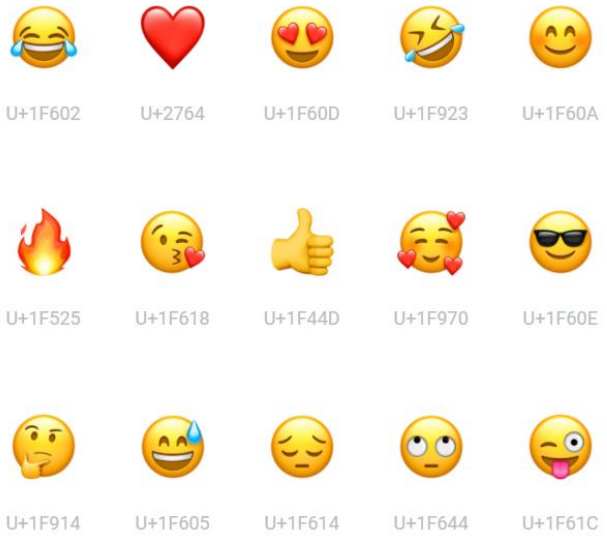
Different countries with different language systems implement their own character encoding

(e.g., sending text message)

Unicode (统一码、万国码、单一码) https://unicode-table.com/en/

- Motivated by the need to encode characters in all languages consistently without conflicts
- A character maps to something called a code point (A: U+0041)
- Unicode comprises 1,114,112 code points in the range 0_{hex} to $10FFFF_{hex}$





Encoding Scheme

- Unicode is a standard (defines the mapping to code point)
- An encoding scheme follows the Unicode standard and defines how code points are stored in memory
- There are different encoding schemes for packaging Unicode code points into bytes, e.g., UTF-8, UTF-16, UTF-32

UTF-8
- Uses a minimum of 1 byte, but if the character is bigger, then it can use 2, 3 or 4 bytes.
- is compatible with the ASCII table

UTF-16
- uses a minimum of 2 bytes. UTF-16 can not take 3 bytes, it can either take 2 or 4 bytes
- is not compatible with the ASCII table

UTF-32
- always uses 4 bytes
- is not compatible with the ASCII table

character	encoding	bits
A	UTF-8	01000001
A	UTF-16	00000000 01000001
A	UTF-32	00000000 00000000 00000000 01000001
あ	UTF-8	11100011 10000001 10000010
あ	UTF-16	00110000 01000010
あ	UTF-32	00000000 00000000 00110000 01000010

UTF-8

- UTF-8 stands for “Unicode Transformation Format – 8-bit”
- Characters are encoded with varied lengths (1~4 bytes)
  - For example: "T" in UTF-8 is "01010100"
  - "汉" in "UTF-8" is "11100110 10110001 10001001 “

Character Range	Encoding
0...7F	$0a_6a_5a_4a_3a_2a_1a_0$
80...7FF	$110a_{10}a_9a_8a_7a_6\ 10a_5a_4a_3a_2a_1a_0$
800...FFFF	$1110a_{15}a_{14}a_{13}a_{12}\ 10a_{11}a_{10}a_9a_8a_7a_6\ 10a_5a_4a_3a_2a_1a_0$
10000...10FFFF	$11110a_{20}a_{19}a_{18}\ 10a_{17}a_{16}a_{15}a_{14}a_{13}a_{12}\ 10a_{11}a_{10}a_9a_8a_7a_6\ 10a_5a_4a_3a_2a_1a_0$

Image: Core Java Volume II, 2.2.4

UTF-8

https://stackoverflow.com/a/27939161/636398

A Chinese character:	汉
its Unicode value:	U+6C49
convert 6C49 to binary:	01101100 01001001

1st Byte	2nd Byte	3rd Byte	4th Byte	Number of Free Bits
0xxxxxxx				7
110xxxxx	10xxxxxx			(5+6)=11
1110xxxx	10xxxxxx	10xxxxxx		(4+6+6)=16
11110xxx	10xxxxxx	10xxxxxx	10xxxxxx	(3+6+6+6)=21

Header	Place holder	Fill in our Binary	Result
1110	xxxx	0110	11100110
10	xxxxxx	110001	10110001
10	xxxxxx	001001	10001001

To computer, its simply 0110110001001001
(don't know whether its 1 or 2 character)

Add header to free bits

11100110 10110001 10001001

Java char

```java
int v1 = 0x0454; // Hex
System.out.printf("%c\n", v1); //є
System.out.printf("%c\n", (char)v1); //є

int v2 = 1108; // Decimal
System.out.printf("%c\n", v2); //є
System.out.printf("%c\n", (char)v2); //є

int v3 = 0x10454; // Hex
System.out.printf("%c\n", v3); //𐑔
System.out.printf("%c\n", (char)v3); //є
```

Java `char` implementation

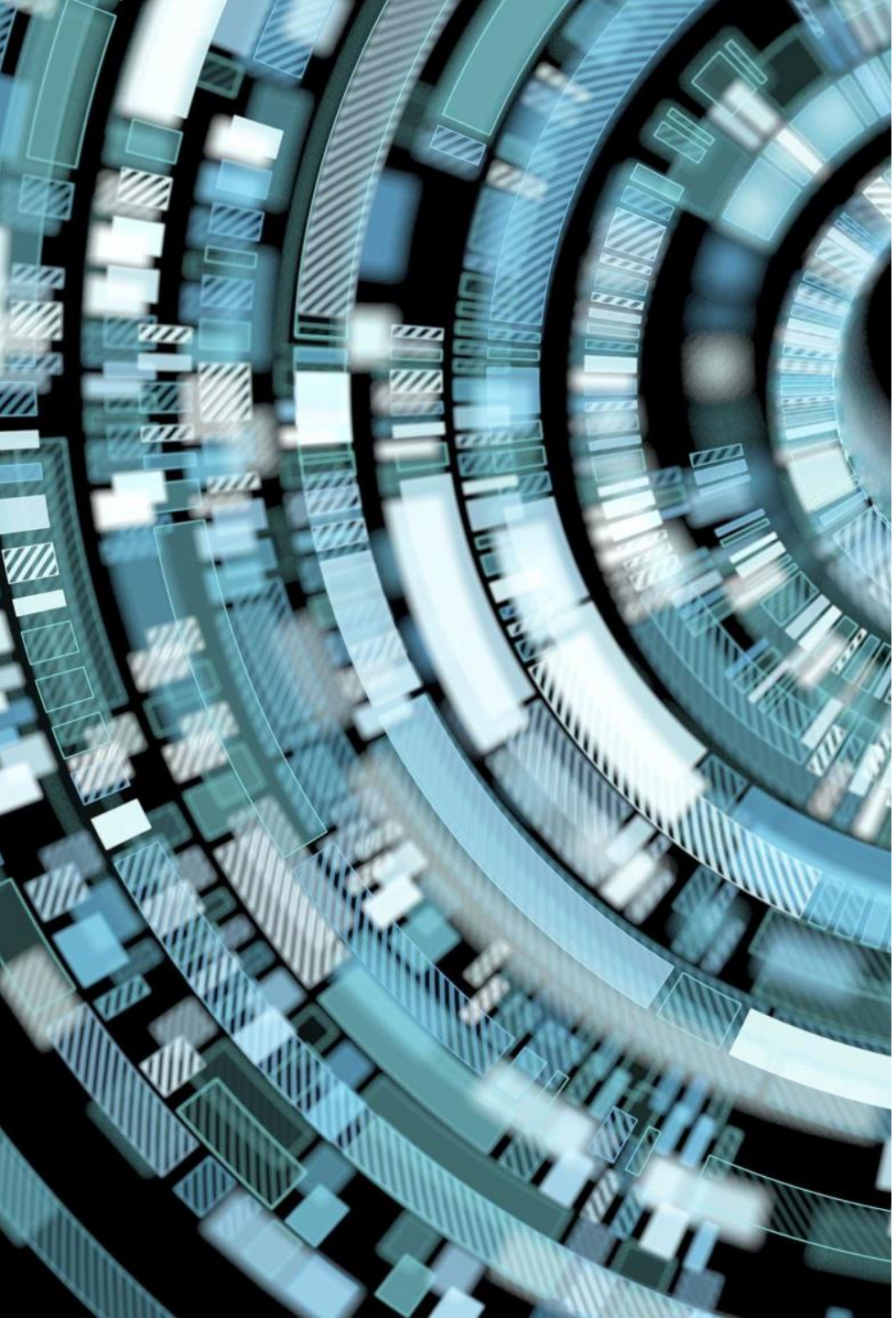
- 16-bit unsigned int (U+0000~U+FFFF), corresponding to Unicode code points
- Conversion between `int` and `char` refers to the Unicode mapping

https://unicode-table.com/en

Java char

- Unicode legal range now: U+0000 to U+10FFFF
- Characters whose code points are greater than U+FFFF are called supplementary characters
- Supplementary characters are represented as a pair of char values (16 + 16 = 32 bits / 4 bytes)

```
int v1 = 0x0454;
int v3 = 0x10454;
char[] c1 = Character.toChars(v1); // length 1
char[] c2 = Character.toChars(v3); // length 2
System.out.println(c1); //є
System.out.println(c2); //𐑔
```



Lecture 5

- I/O Overview
- i18n & Character Encoding
- Byte Streams & Character Streams
- Combining Stream Filters
- Reading/Writing Text Input/Output
- I/O from Command Line

Java I/O Streams

- A Stream is a continuous flow of data that can be accessed sequentially (not like an array for which we could use index to move back and forth)
- A Stream, as a data container, is linked to a data source and a data destination

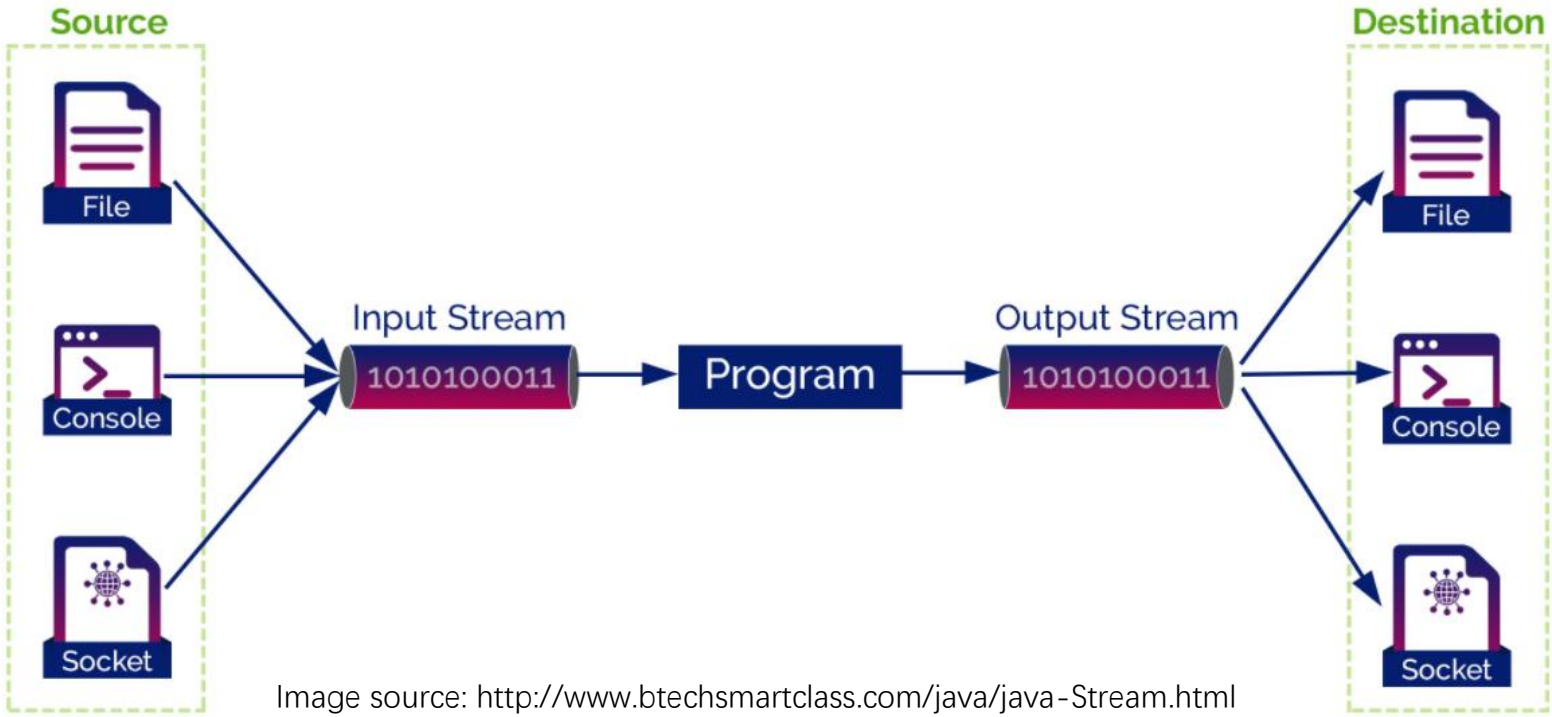
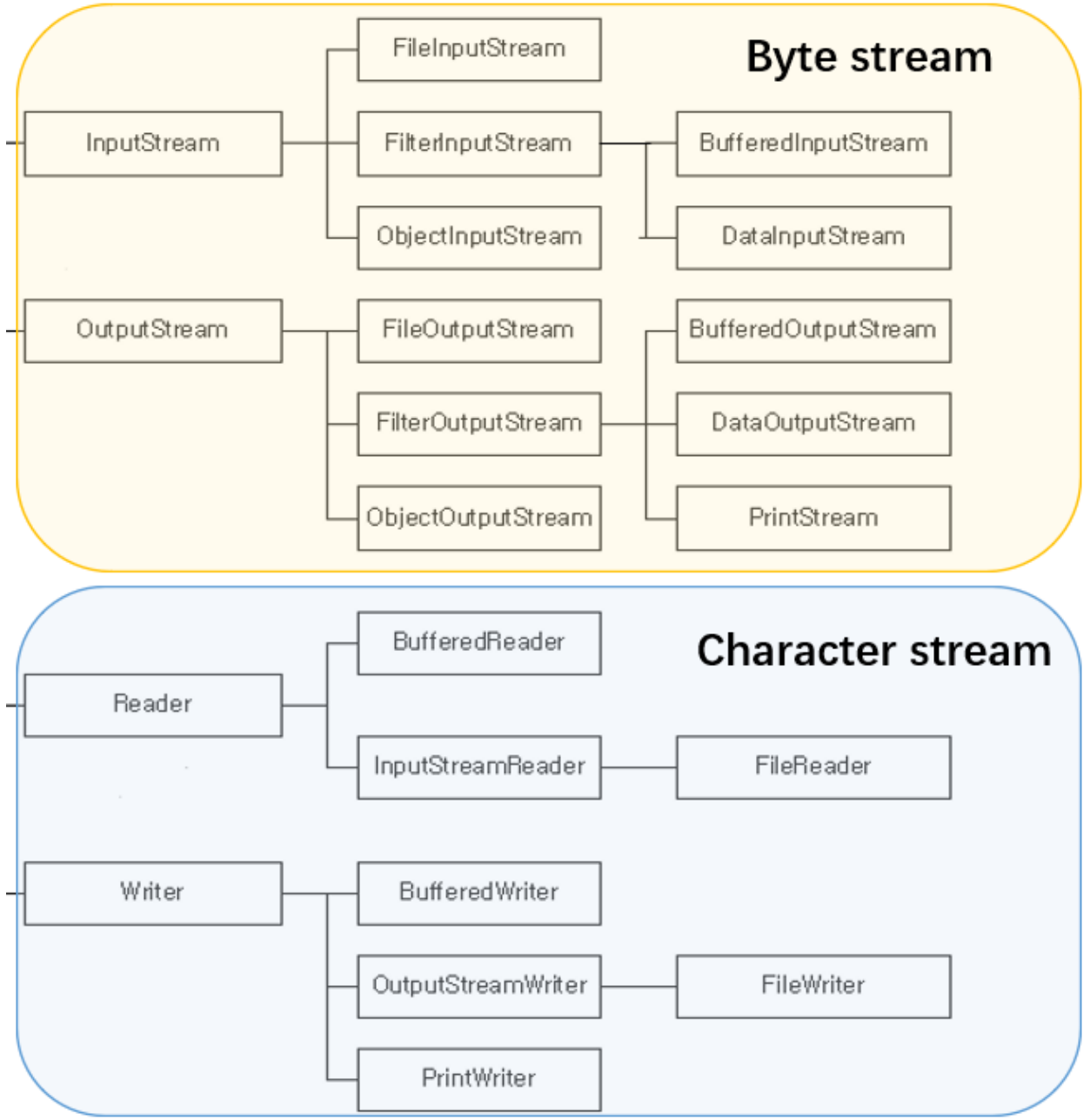


Image source: http://www.btechsmartclass.com/java/java-Stream.html

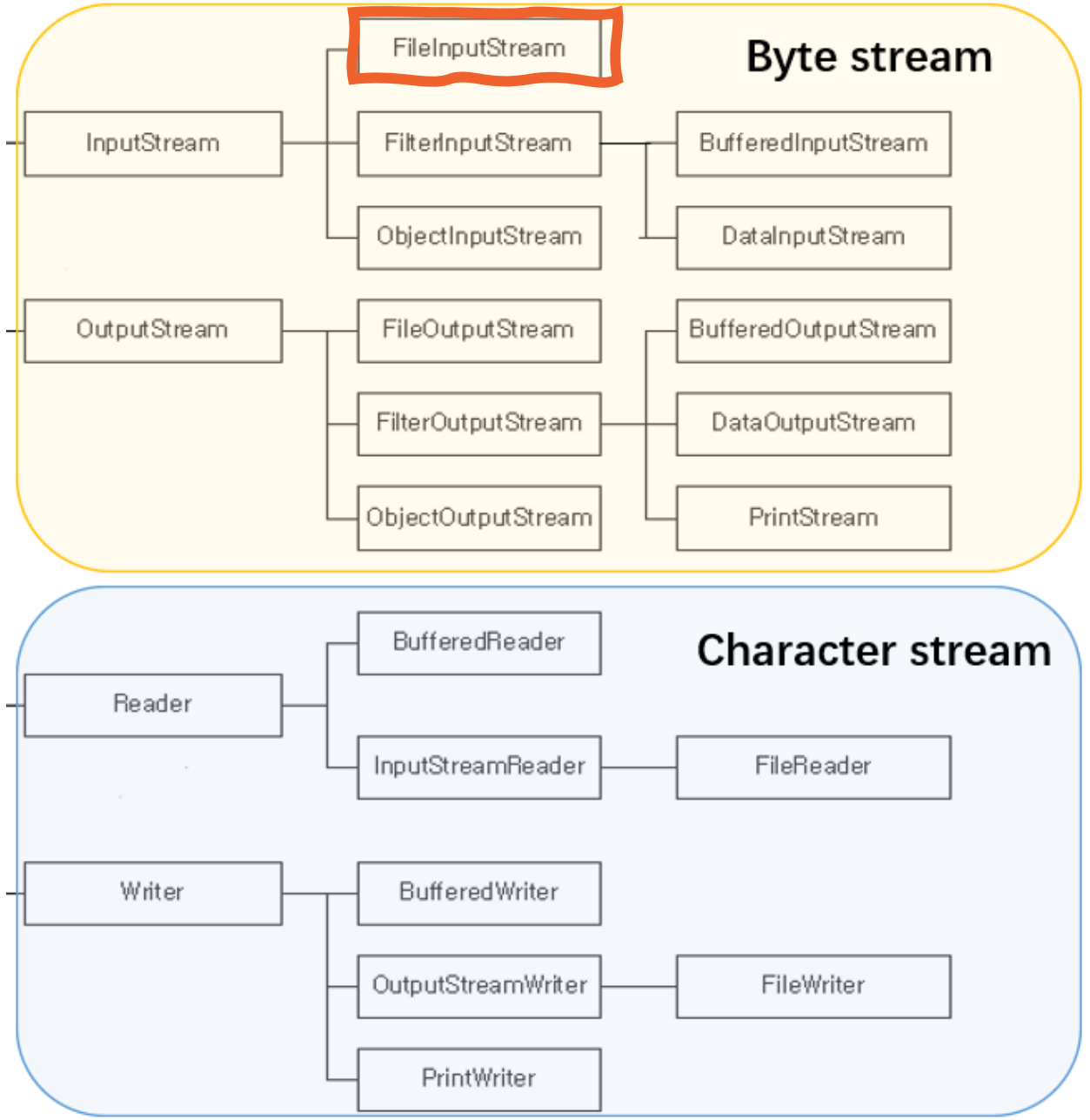
Java I/O Streams

- Byte Stream
  - *Input stream:* an object from which we can read a sequence of bytes
  - *Output stream:* an object to which we can write a sequence of bytes
  - Byte streams are inconvenient for processing info stored in Unicode
- Character Stream
  - A separate hierarchy provides classes, inheriting from `Reader` and `Writer`, for processing Unicode characters
  - These classes have `read` and `write` operations that are based on `char` values rather than `byte` values



Similarity

- `InputStream` & `OutputStream`, `Reader`& `Writer` are abstract classes
- Subclasses are all called “xxxStream” or “xxxReader” & “xxxWriter”
- Subclasses for `InputStream` or `Reader` must implement `read()`
- Subclasses for `OutputStream` or `Writer` must implement `write()`



FileInputStream

Used for reading streams of raw bytes

```
public void readFile() throws IOException {
    try (InputStream input = new FileInputStream("src/test.txt")) {
        int n;
        while ((n = input.read()) != -1) {
            System.out.println(n);
        }
    }
}
```

Reading 1 byte a time until there is no more data (-1)

What is the output when test.txt contains the text “Hello World”? (e.g., file encoding is UTF-8)

```
72 101 108 108 111 32 87 111 114 108 100
```

FileInputStream

Used for reading streams of raw bytes

- What if `test.txt` contains “计算机系统”？

```
try (InputStream input = new FileInputStream("src/test.txt")) {
    int n;
    while ((n = input.read()) != -1) {
        System.out.print(" " + n);
    }
}
```

If file encoding is UTF-8

```
232 174 161 231 174 151 230 156 186
231 179 187 231 187 159
```

In UTF-8, normal Chinese characters often take 3 bytes

If file encoding is GBK

```
188 198 203 227 187 250 207 181 205 179
```

1 Chinese Character requires more than 1 byte to store (2 bytes for GBK encoding)

GBK encoding for 计：BCC6

FileInputStream

Used for reading streams of raw bytes

- How to get meaningful characters? Can we directly cast bytes to char?

```
try (InputStream input = new FileInputStream("src/test.txt")) {
    int n;
    while ((n = input.read()) != -1) {
        System.out.print((char)n);
    }
}
```

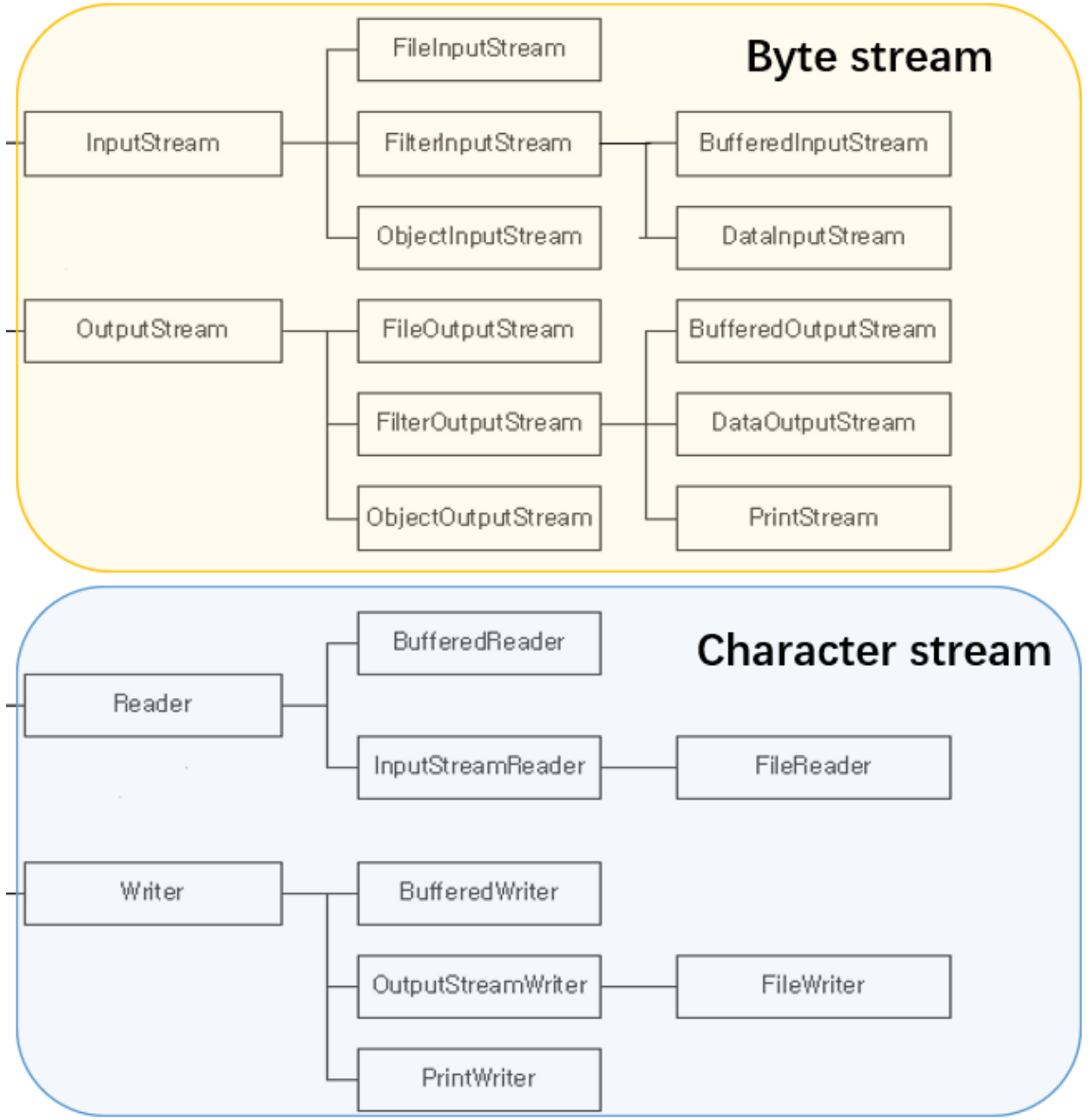
Works only for “Hello World” in which each character can be encoded using only 1 byte
Not working for “计算机系统”, which takes 2 bytes (GBK) or 3 bytes (UTF8) for 1 character

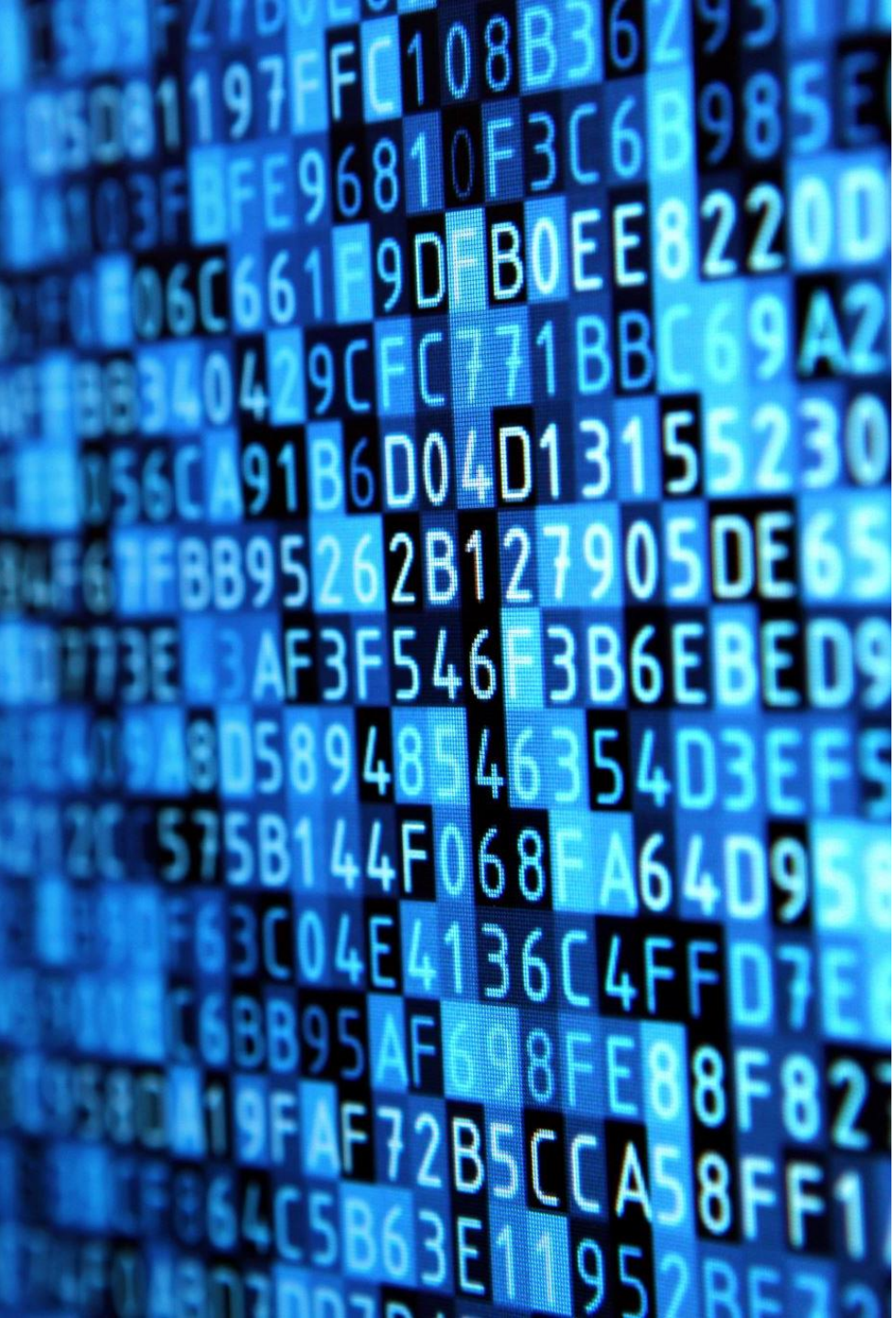
FileReader

Used for reading streams of characters (instead of streams of bytes)

Q: Data is still read as streams of 0s and 1s. How to decide the corresponding character?

A: We need to specify an encoding scheme. If not specified, use the default encoding scheme.





Java Default Encoding

- Java system/platform default encoding
  - The default encoding when JVM starts (i.e., used when deciding bytes for a character)
  - Differs from OS and language settings (e.g., GBK on 中文操作系统)
  - Could be changed (environment variable, IDE, code)
- File encoding
  - Independent from Java
  - Could be changed
  - When using Java to read a file, the Java system default encoding and the file encoding should be consistent

Encoding Support for Java

- Every implementation of the Java platform must support the following basic standard charsets (see `StandardCharsets`)

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

FileReader

- What if the input `txt` contains “计算机系统” and has UTF-8 file encoding? (consistent with **my** Java default encoding UTF-8)

```
try(Reader reader = new FileReader( fileName: "src/io/sample2.txt")){
    int n;
    while((n=reader.read())!=-1){
        System.out.printf("%d %c\n", n, n);
    }
}
```

```
35745 计
31639 算
26426 机
31995 系
32479 统
```

Return the read char as an integer (range 0 to 65535 or 2^{16})　　计：U+8BA1

FileReader

- What if the input `txt` contains “计算机系统” and has GBK file encoding? (**inconsistent** with **my** Java default encoding UTF-8)

```
try(Reader reader = new FileReader( fileName: "src/io/sample3.txt")){
    int n;
    while((n=reader.read())!=-1){
        System.out.printf("%d %c\n", n, n);
    }
}
```

```
65533 
65533 
65533 
65533 
65533 
1013 ϵ
883 т
```

Malformed UTF-8 bytes are replaced by default string;
GBK bytes that happen to be valid UTF-8 bytes are mapped to different characters.

FileReader

- What if the input `txt` contains “计算机系统” and has GBK file encoding? (**inconsistent** with **my** Java default encoding UTF-8)

```
try(Reader reader = new FileReader( fileName: "src/io/sample3.txt", Charset.forName("gb2312"))){
    int n;
    while((n=reader.read())!=-1){
        System.out.printf("%d %c\n", n, n);
    }
}
```

Set FileReader encoding to be consistent with the file encoding

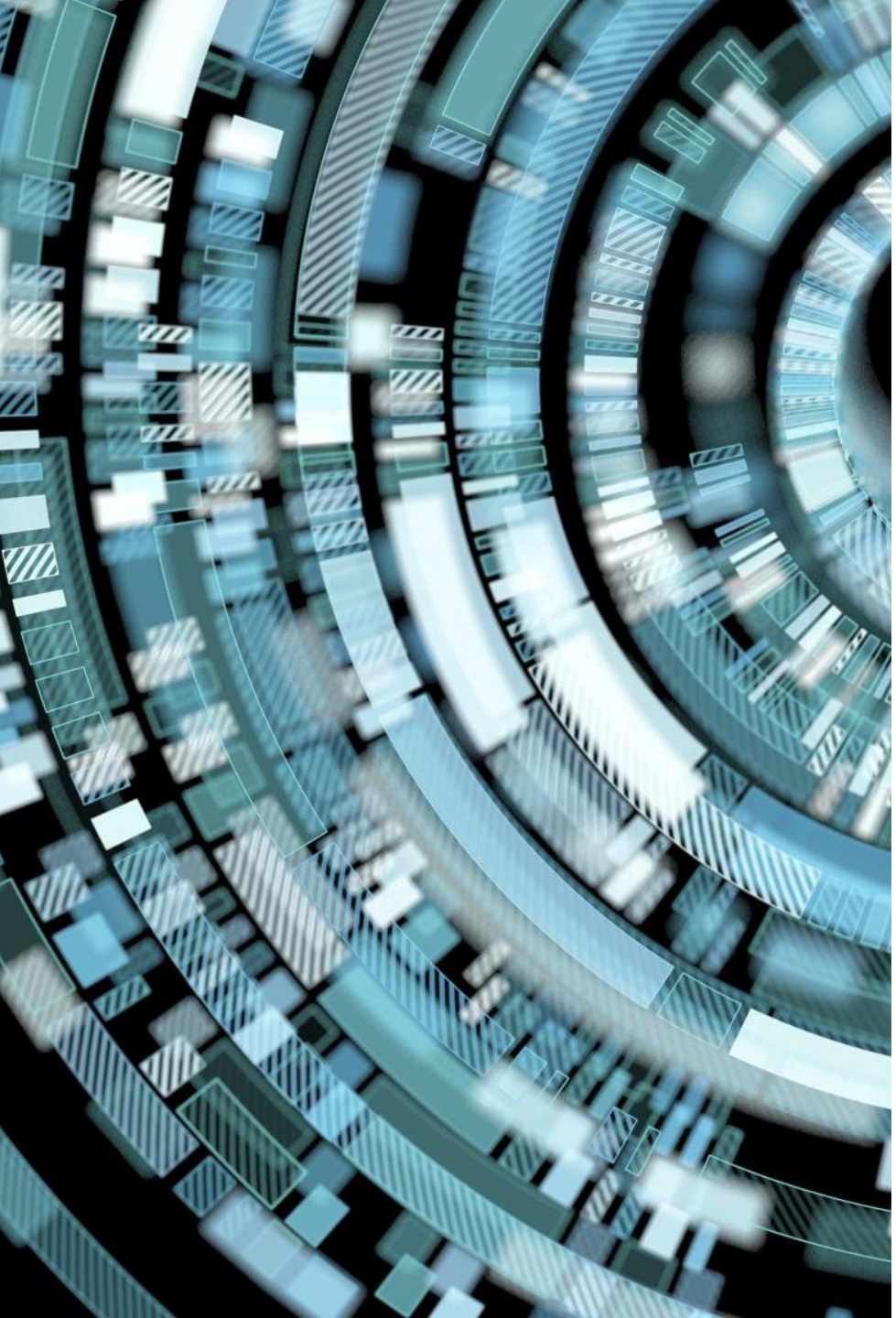
```
35745 计
31639 算
26426 机
31995 系
32479 统
```

InputStream to Reader

- FileReader under the hood: using FileInputStream for reading bytes, then convert them to characters based on the given encoding
- Use InputStreamReader to transform InputStream to Reader

```
// create FileInputStream
InputStream input = new FileInputStream("src/test.txt");
// convert to FileReader by specifying encoding
Reader reader = new InputStreamReader(input, "UTF-8");
```

OutputStream and Writer have the same pattern

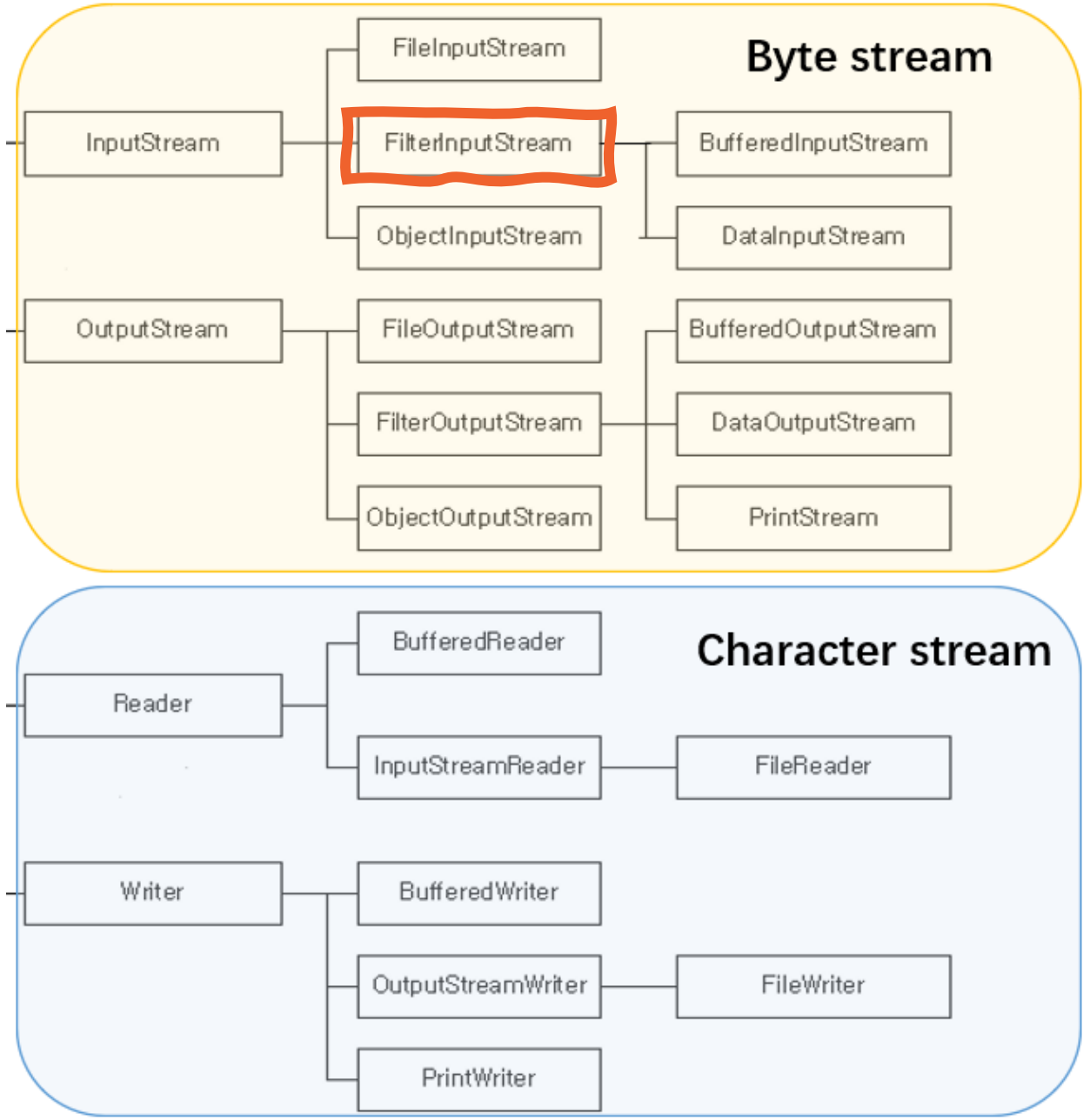


Lecture 5

- I/O Overview
- i18n & Character Encoding
- Byte Streams & Character Streams
- Combining Stream Filters
- Reading/Writing Text Input/Output
- I/O from Command Line

Java I/O Streams

- Byte Stream
  - *Input stream:* an object from which we can read a sequence of bytes
  - *Output stream:* an object to which we can write a sequence of bytes
  - Byte streams are inconvenient for processing info stored in Unicode
- Character Stream
  - A separate hierarchy provides classes, inheriting from `Reader` and `Writer`, for processing Unicode characters
  - These classes have `read` and `write` operations that are based on `char` values rather than `byte` values



FilterInputStream an example of the Decorator design pattern

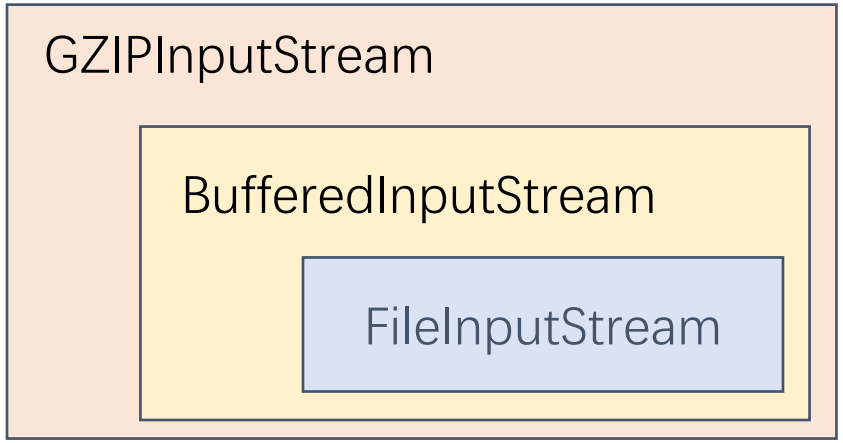
- Contains some other `InputStream` as its basic source of data
- Subclasses of `FilterInputStream` provide additional functionality on top of the original stream
- Direct known subclasses
  - `BufferedInputStream`
  - `DataInputStream`
  - `DigestInputStream`
  - `InflaterInputStream`
  - `LineNumberInputStream`
  - `PushbackInputStream`
  - etc……

FilterInputStream Example I

+ gzip functionality

+ buffered functionality

  original data




```
InputStream zfile = new
GZIPInputStream(bfile);

InputStream bfile = new
BufferedInputStream(file);

InputStream file = new
FileInputStream("src/test.zip");
```

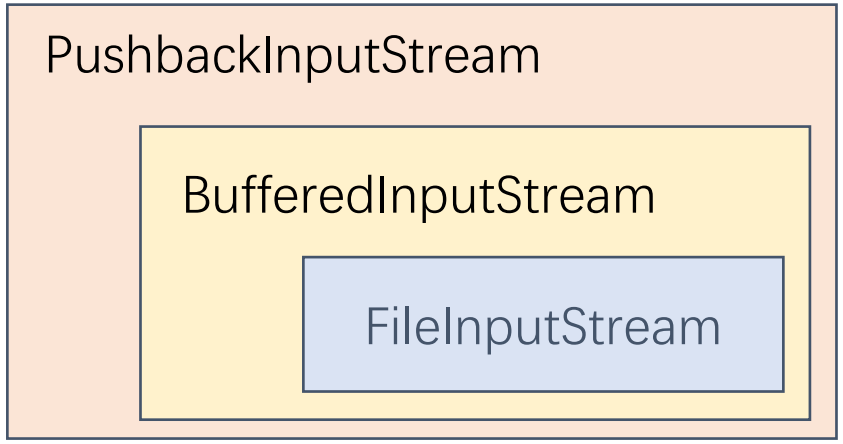

Example adopted from https://www.liaoxuefeng.com/

FilterInputStream Example II

+ pushback functionality

+ buffered functionality

original data



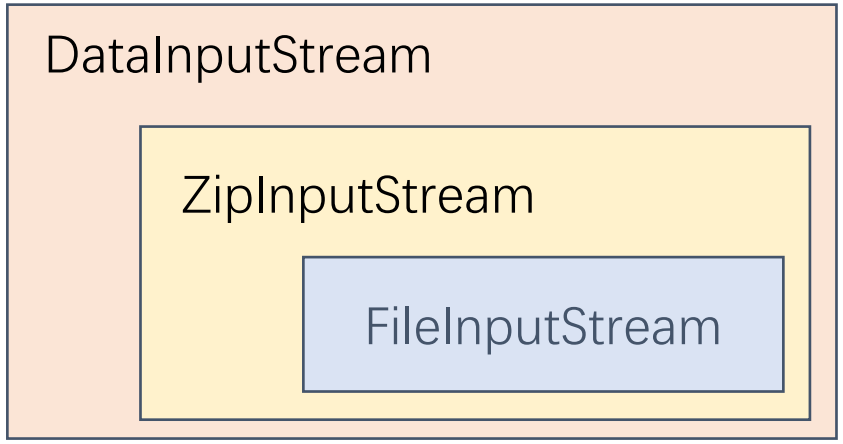

```
PushbackInputStream pbin = new PushbackInputStream(
                                    new BufferedInputStream(
                                        new FileInputStream("test.txt")));
int b = pbin.read();
pbin.unread(b);  // added functionality to push back bytes
```

FilterInputStream Example III

+ read-numbers functionality

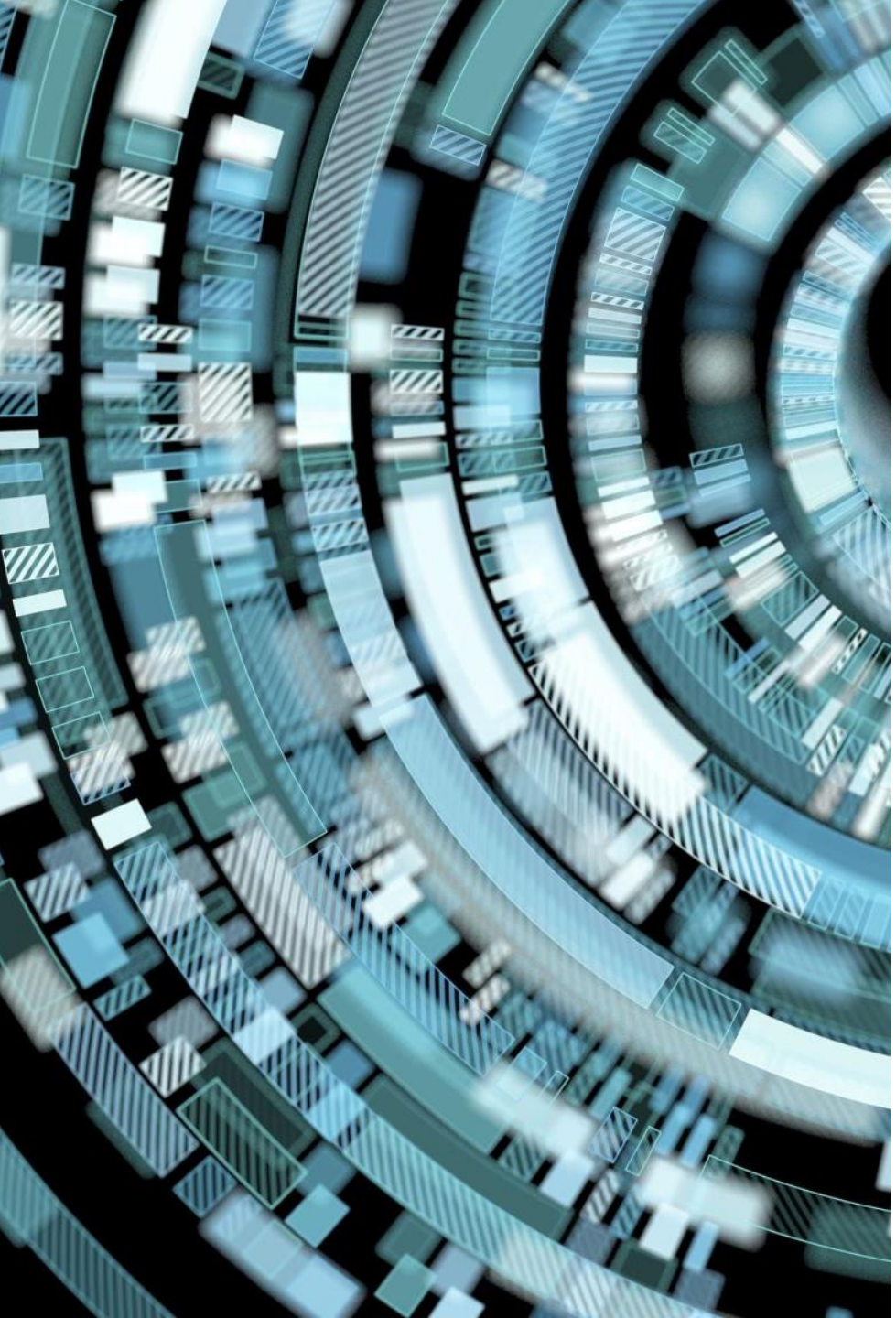
+ zip functionality

original data




```
DataInputStream din=new DataInputStream(
                                new ZipInputStream(
                                    new FileInputStream("test.zip")));

// added abilities to read numbers
din.readDouble();
din.readInt();
```



Lecture 5

- I/O Overview
- Character Encoding
- Byte Streams & Character Streams
- Combining Stream Filters
- Reading/Writing Text Input/Output
- I/O from Command Line

Reading/Writing Text Input/Output

- When working with I/O, we often work with human-readable text rather than binary data
- Java provide two APIs to assist working with text I/O
  - Scanning: useful for breaking down formatted input into tokens and translating individual tokens according to their data type (`Scanner`).
  - Formatting: assembles data into nicely formatted, human-readable form (`PrintWriter`)

Using Scanner for reading text files

To begin, construct a File object with the name of the input file:

```
File inputFile = new File("input.txt");
```

Then use the File object to construct a Scanner object:

```
Scanner in = new Scanner(inputFile);
```

This Scanner object reads text from the file input.txt. You can use the Scanner methods (such as nextInt, nextDouble, and next) to read data from the input file.

For example, you can use the following loop to process numbers in the input file:

```
while (in.hasNextDouble()) {
   double value = in.nextDouble();
   Process value.
}
```

Using `PrintWriter` for writing text files

To write output to a file, you construct a `PrintWriter` object with the desired file name, for example

```
PrintWriter out = new PrintWriter("output.txt");
```

If the output file already exists, it is emptied before the new data are written into it. If the file doesn't exist, an empty file is created.

The `PrintWriter` class is an enhancement of the `PrintStream` class that you already know—`System.out` is a `PrintStream` object. You can use the familiar `print`, `println`, and `printf` methods with any `PrintWriter` object:

```
out.println("Hello, World!");
out.printf("Total: %8.2f\n", total);
```

Constructing a Scanner with a String

When you construct a `PrintWriter` with a string, it writes to a file:

```
PrintWriter out = new PrintWriter("output.txt");
```

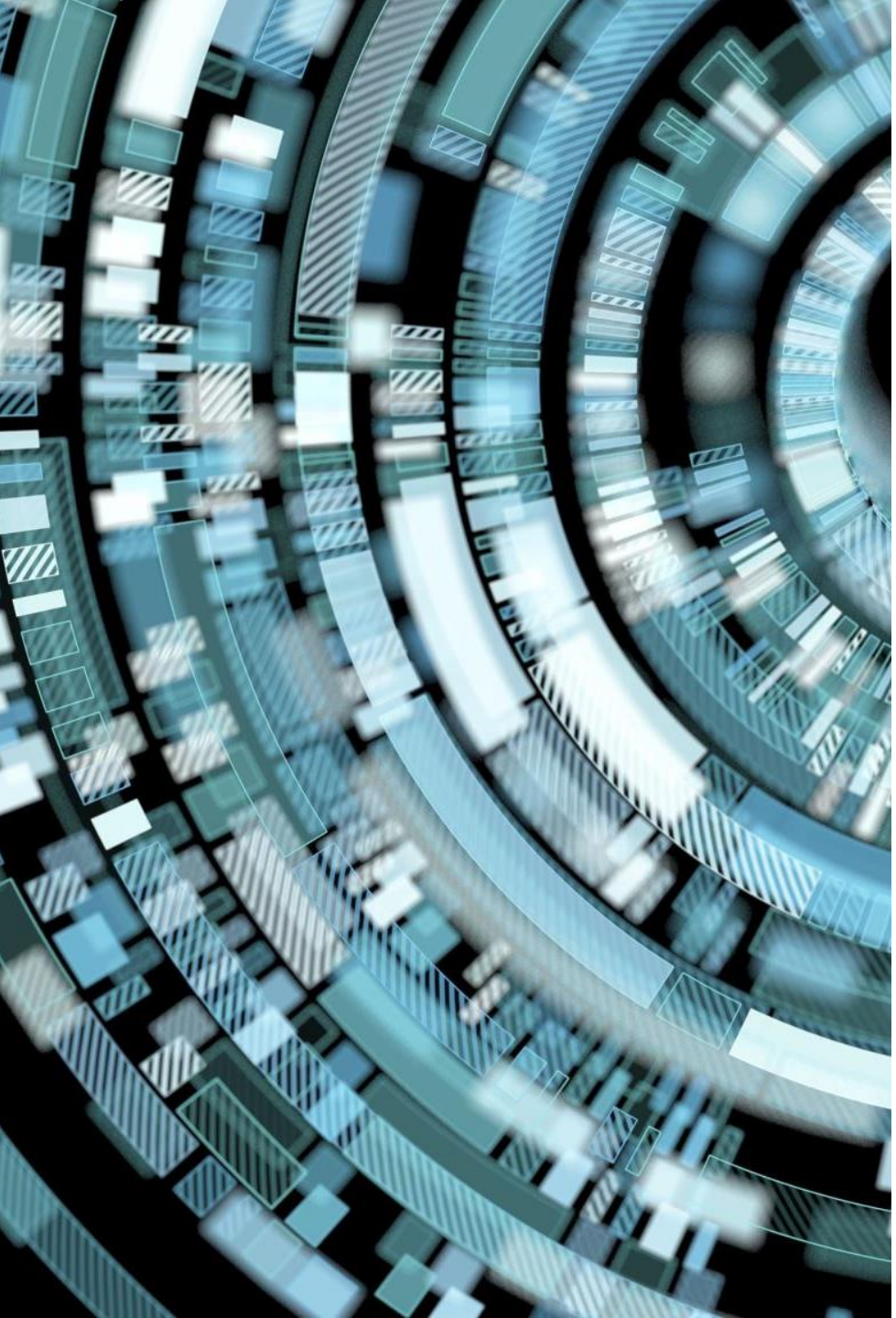
However, this does *not* work for a `Scanner`. The statement

```
Scanner in = new Scanner("input.txt"); // Error?
```

does *not* open a file. Instead, it simply reads through the string: `in.next()` returns the string `"input.txt"`. (This is occasionally useful.)

You must simply remember to use `File` objects in the `Scanner` constructor:

```
Scanner in = new Scanner(new File("input.txt")); // OK
```



Lecture 5

- I/O Overview
- Character Encoding
- Byte Streams & Character Streams
- Combining Stream Filters
- Reading/Writing Text Input/Output
- I/O from Command Line

I/O from the Command Line

- A program is often run from the command line and interacts with the user in the command line environment
- Java supports this kind of interaction in two ways:
  - Standard Streams (often used, e.g., `System.out`):
  - Console (more advanced, e.g., `System.console()`)

Standard Streams

```
java.lang.Object
    java.lang.System
```

```
public final class System
extends Object
```

- Standard streams read input from the keyboard and write output to the display.
- Java platform supports three Standard Streams

Fields

Modifier and Type	Field and Description
static PrintStream	err The "standard" error output stream.
static InputStream	in The "standard" input stream.
static PrintStream	out The "standard" output stream.

`System.err`

`System.in`

`System.out`

System.in

```
public static final InputStream in
```

- Standard input, often read keyboard input
- `System.in` is a byte stream with no character stream features. To use Standard Input as a character stream, wrap `System.in` in `InputStreamReader`.

Decorator To Reader InputStream

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String str = "";
while (!str.equals("quit")) {
    str = br.readLine();
    System.out.println(str);
}
```

`System.out`

```
public static final PrintStream out
```

- `System.out` is defined as a `PrintStream` object.

- Although it is technically a byte stream, `PrintStream` utilizes an internal character stream object to emulate many of the features of character streams (same for `PrintWriter`)
  - `print` and `println` format individual values in a standard way.
  - `format` formats almost any number of values based on a format string, with many options for precise formatting.

https://docs.oracle.com/javase/tutorial/essential/io/formatting.html

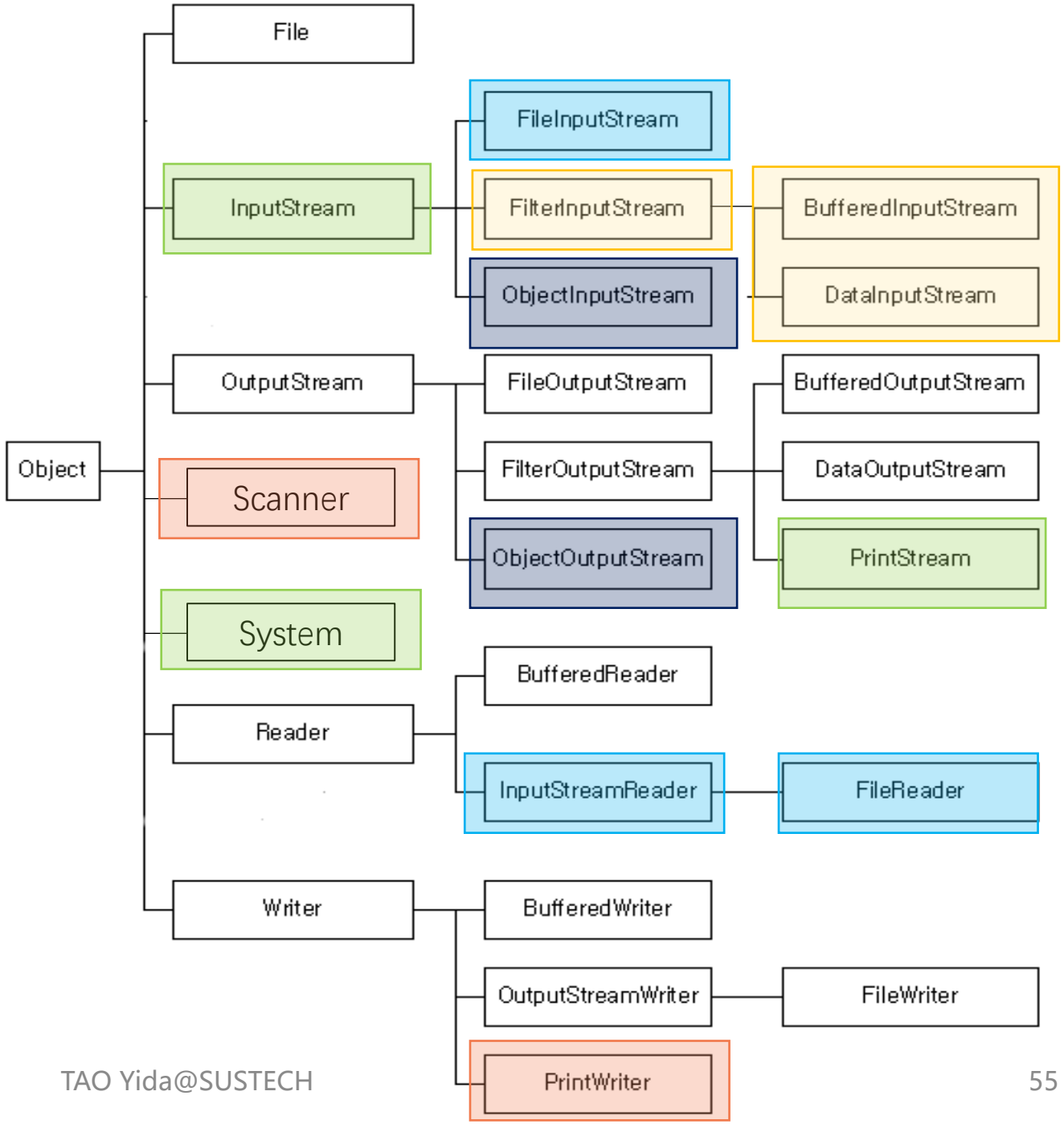
System.out

```
public static final PrintStream out
```

- Could use `setOut()` to redirect the output to other resources

```
// construct a new PrintStream with a specified file
PrintStream out = new PrintStream(new File("src/sysout.txt"));
// re-assign the standard output from console to file
System.setOut(out);
// this will be written to file
System.out.println("where am I?");
```

Let's Review


File
FileInputStream
InputStream
FilterInputStream
BufferedInputStream
ObjectInputStream
DataInputStream
OutputStream
FileOutputStream
BufferedOutputStream
Object
FilterOutputStream
DataOutputStream
Scanner
ObjectOutputStream
PrintStream
System
BufferedReader
Reader
InputStreamReader
FileReader
Writer
BufferedWriter
OutputStreamWriter
FileWriter
PrintWriter

Next Lecture

- Serialization
- Working with Files
- Exception Handlings