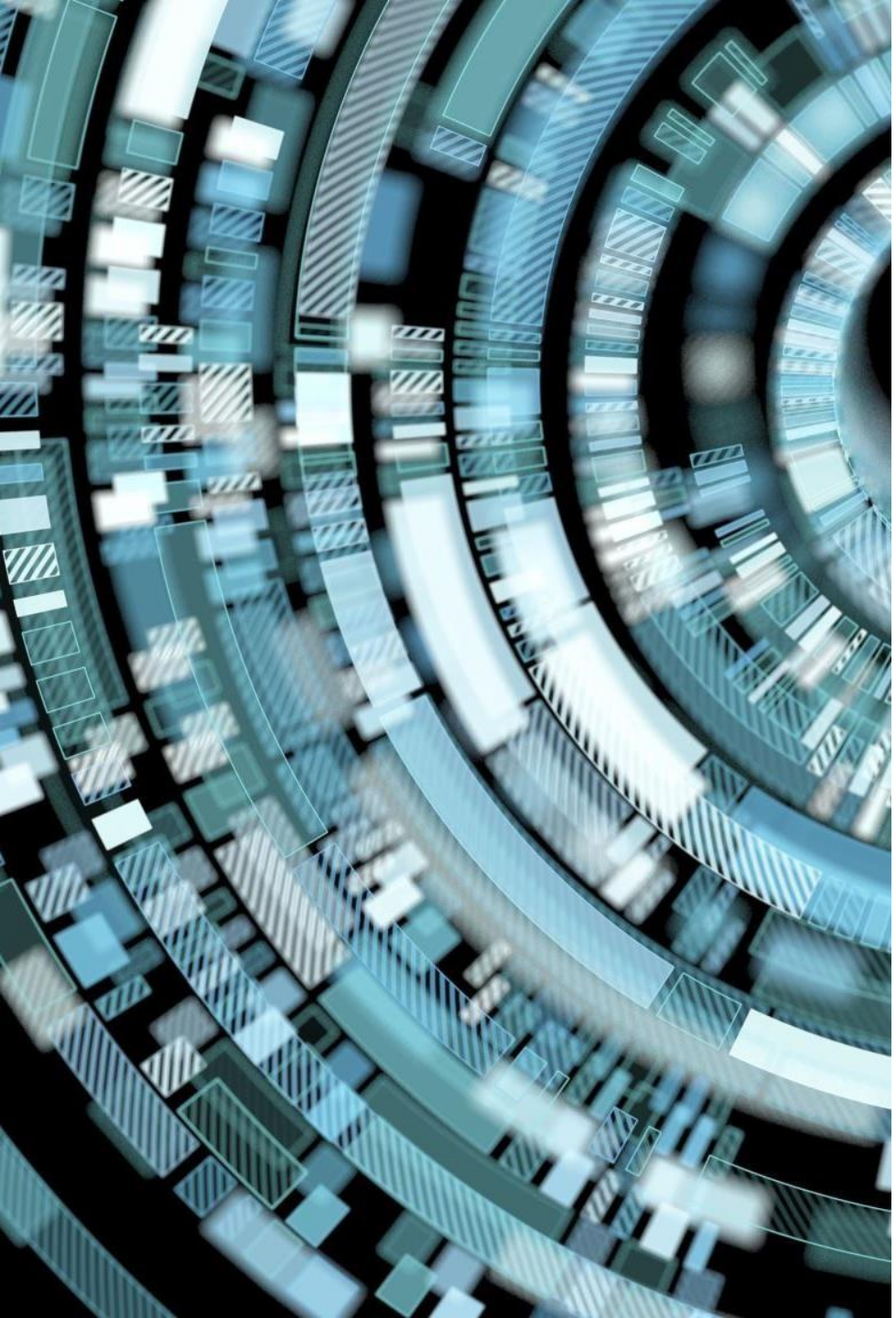
Computer System Design & Application

计算机系统设计与应用A



Lecture 2

- Exception Handling
- I/O and Encoding
- NIO and Files
- Persistence and Serialization

Exception

- An exception indicates that a problem occurs during a program's execution
- An exception disrupts the normal flow of the program

Happy Path

Files are always there
Network is always okay
Memory is always enough
User input is always valid
……

Unhappy Path

Files are not found
Network breaks down
Memory is not enough
User input is invalid
……

Exception Handling

- A mechanism to handle runtime errors (gracefully) in order to maintain the normal flow of the program

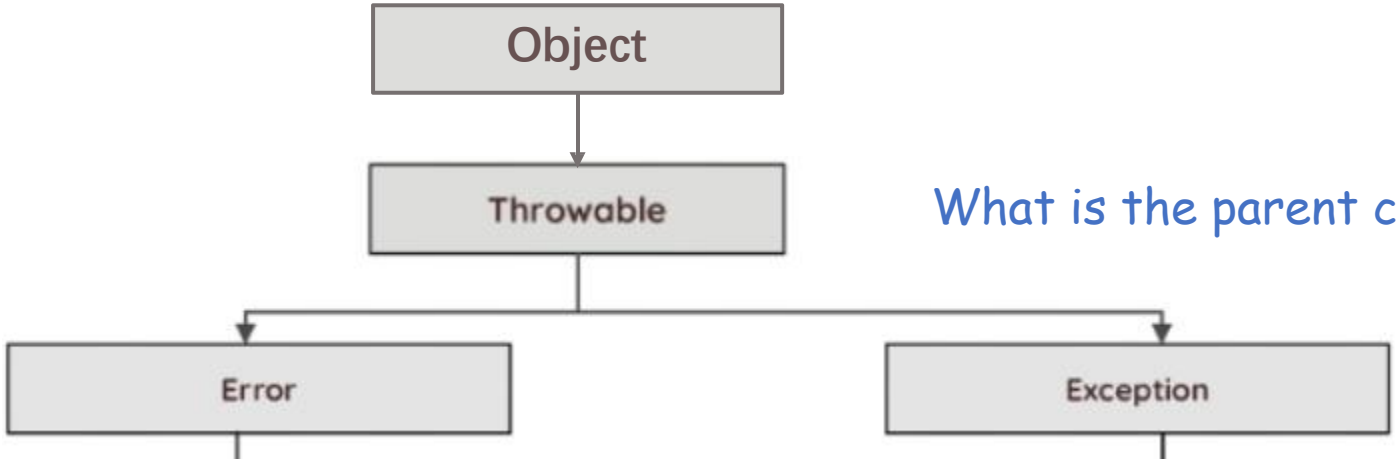
Handling
Passing control from the point of error detection to a handler that can deal with the error

```
try {
    File text = new File("C:/temp/test.txt");
    Scanner s = new Scanner(text);
} catch (FileNotFoundException e) {
    System.err.println("file not found.");
}
```

Detection
Scanner class could detect the error, but cannot handle it

Java Exception Hierarchy

- The `Throwable` class is at the top of the Java exception class hierarchy; has two direct subclass: `Error` and `Exception`
- Only `Throwable` or its subclasses
  - Can be thrown by JVM or the `throw` keyword
  - Can be caught by the `catch` keyword



What is the parent class of Throwable?

Error

- An error indicates serious problems that a reasonable application **should not** try to catch
- E.g., `OutofMemoryError`, `StackOverflowError`

Mostly thrown by JVM in a scenario considered fatal; no way for the application program to recover from that error

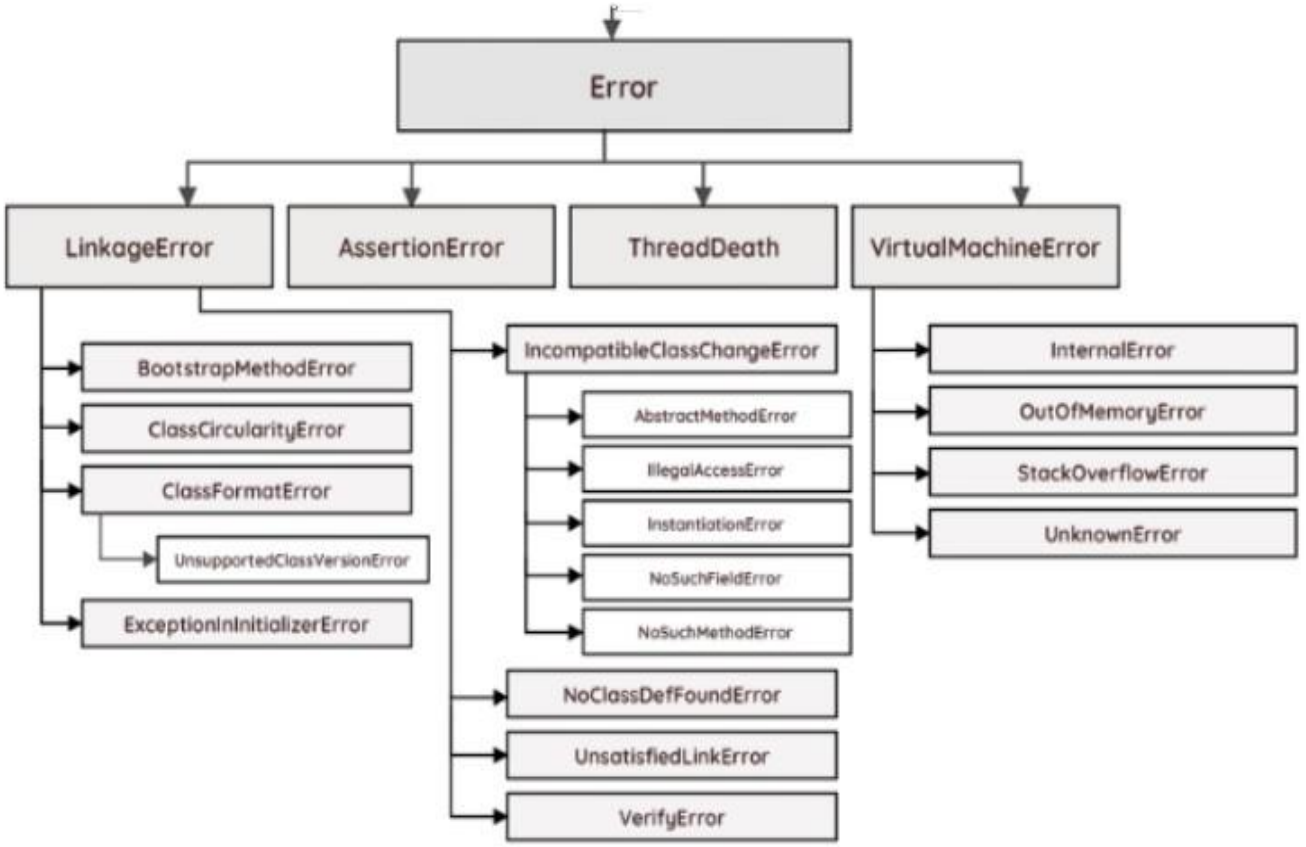


Image source: https://rollbar.com/blog/java-exceptions-hierarchy-explained

Example

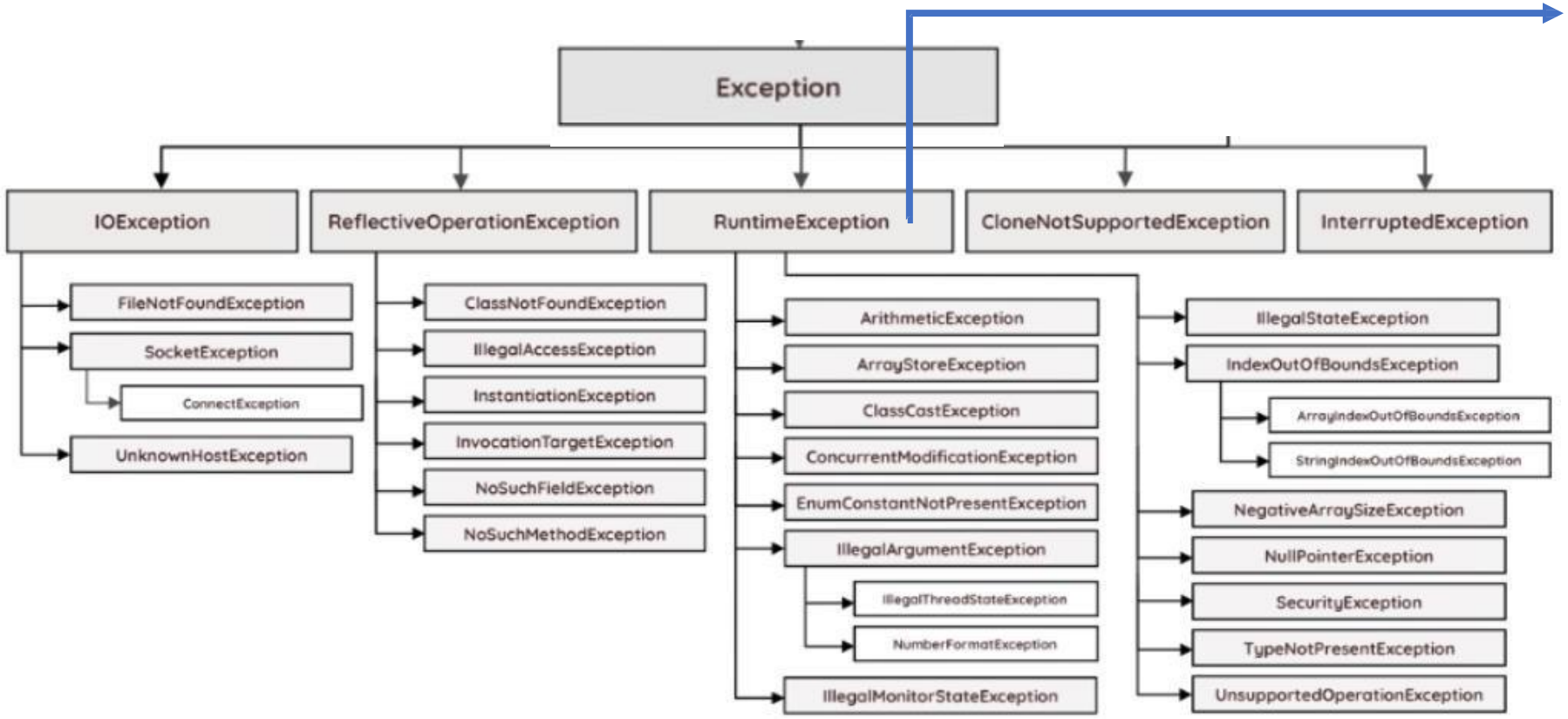
- What is the problem with `foo(String s)`?
- Stack is exhausted, leading to `StackOverflowError`
- No recovery during execution, just let it terminate
- Fixing the code or increasing JVM stack size (-Xss)

```
public void foo(String s)
{
        foo(s);
}
```

```
Exception in thread "main" java.lang.StackOverflowError
        at examples.foo(examples.java:58)
        at examples.foo(examples.java:58)
        at examples.foo(examples.java:58)
        at examples.foo(examples.java:58)
        at examples.foo(examples.java:58)
        at examples.foo(examples.java:58)
        at examples.foo(examples.java:58)
```

Exception

- An exception indicates a condition that a reasonable application might **want to** catch.



- `RuntimeException` and its subclasses are unchecked exceptions
- Others are checked exceptions (think of it as "checked" by compiler)

Checked Exceptions

- Checked Exceptions cannot be ignored at the time of compilation
- Compilers will enforce programmers to handle them

- Two fixes: throw or catch

```
public void processFile() {
    File text = new File("C:/test.txt");
    Scanner s = new Scanner(text);
}
```

Unhandled exception type FileNotFoundException

2 quick fixes available:

Add throws declaration

Surround with try/catch

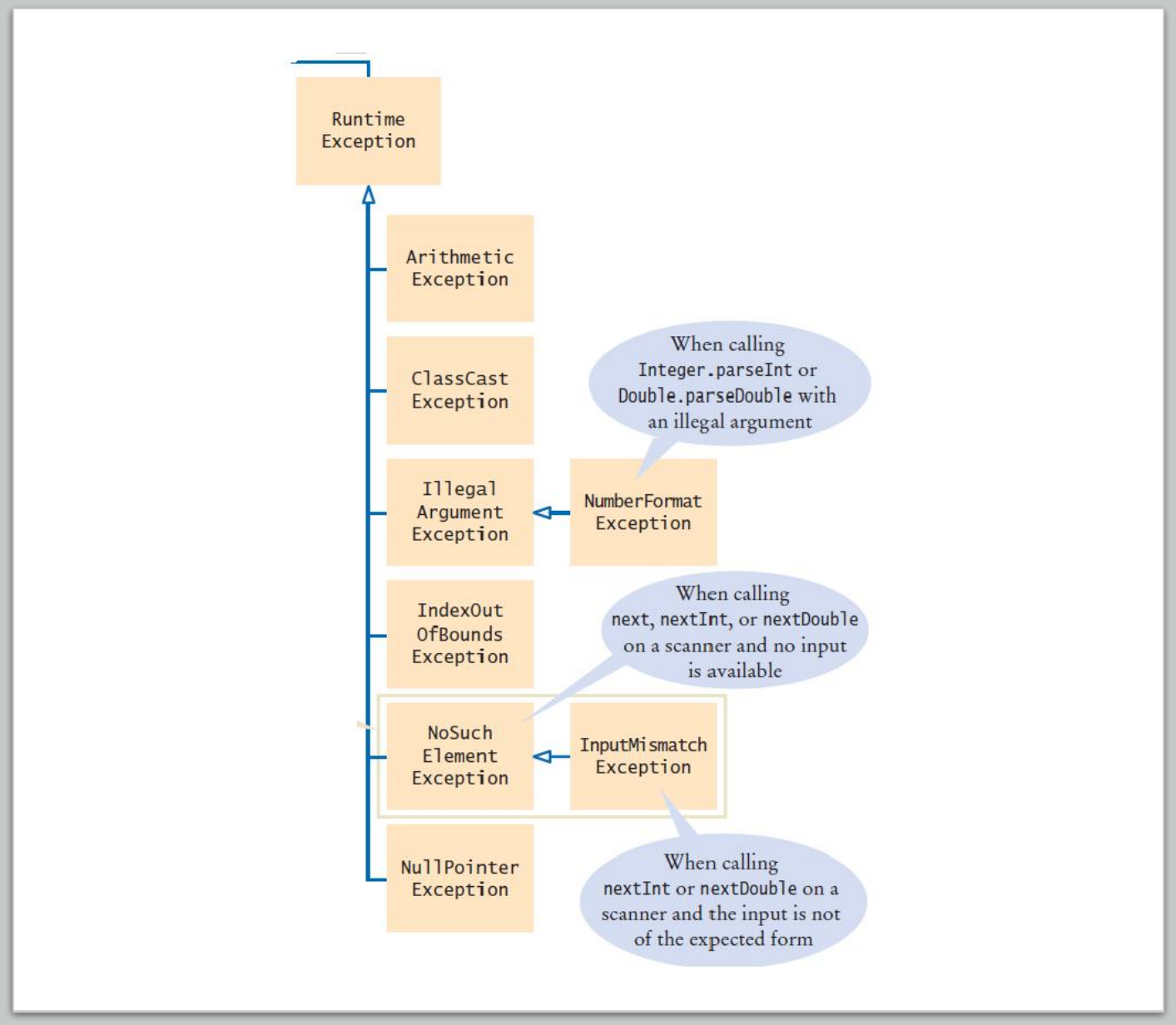
Press 'F2' for focus

```
public void processFile() throws FileNotFoundException {
    File text = new File("C:/test.txt");
    Scanner s = new Scanner(text);
}
```

```
public void processFile() {
    File text = new File("C:/test.txt");
    try {
        Scanner s = new Scanner(text);
    } catch (FileNotFoundException e) {
        System.out.println("Cannot find file xxx.");
    }
}
```

Unchecked Exceptions

- Will not be checked by compilers; Occur at runtime
- Usually caused by logic errors in programming
- E.g., `NullPointerException`, `IndexOutOfBoundsException`



Catching Multiple Exceptions

```
try {
        // some code
} catch(FileNotFoundException e) {
    logger.log(e);

} catch(SQLException e)
    { logger.log(e);

} catch(SocketException e) {
    logger.log(e);
}
```

Why not the simpler code?

```
try {
        // some code

} catch(Exception e) {
    logger.log(e);

}
```

`catch (Exception e)` is often considered a bad practice

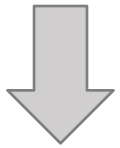
- It may not properly handle logics that required for specific exceptions
- It may catch unexpected exceptions
- It may mask the actual error and impeding debugging

Catching Multiple Exceptions

```
try {
        // some code
} catch(FileNotFoundException e) {
    logger.log(e);

} catch(SQLException e)
    { logger.log(e);

} catch(SocketException e)
    { logger.log(e);
}
```



```
try {
        // some code

} catch(FileNotFoundException | SQLException | SocketException e)
    { logger.log(e);

}
```

In Java 7 and later, a single catch block can handle multiple types of exception

- Reduce code duplication
- Avoid using overly broad exception

try-with-resources

- `try-with-resources` statement ensures that a resource (e.g., InputStream, JDBC connection) is automatically closed after the program is finished with it

```
Scanner scanner = null;
try {
    scanner = new Scanner(new File("test.txt"));
    while (scanner.hasNext()) {
        System.out.println(scanner.nextLine());
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} finally {
    if (scanner != null) {
        scanner.close();
    }
}
```

Before Java 7

```
try (Scanner scanner = new Scanner(new File("test.txt"))) {
    while (scanner.hasNext()) {
        System.out.println(scanner.nextLine());
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

Java 7 and later

try-with-resources

- `try-with-resources` statement ensures that a resource (e.g., InputStream, JDBC connection) is automatically closed after the program is finished with it

Syntax Sugar

- Syntax in a programming language that is designed to make things easier to express
- Compiler automatically inserts a "close()" when compiling the source to bytecode

```
try (Scanner scanner = new Scanner(new File("test.txt"))) {
    while (scanner.hasNext()) {
        System.out.println(scanner.nextLine());
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

try-with-resources

- Define custom resource: Any object that implements the `AutoCloseable` interface and overrides its `close()` method could be used as a resources

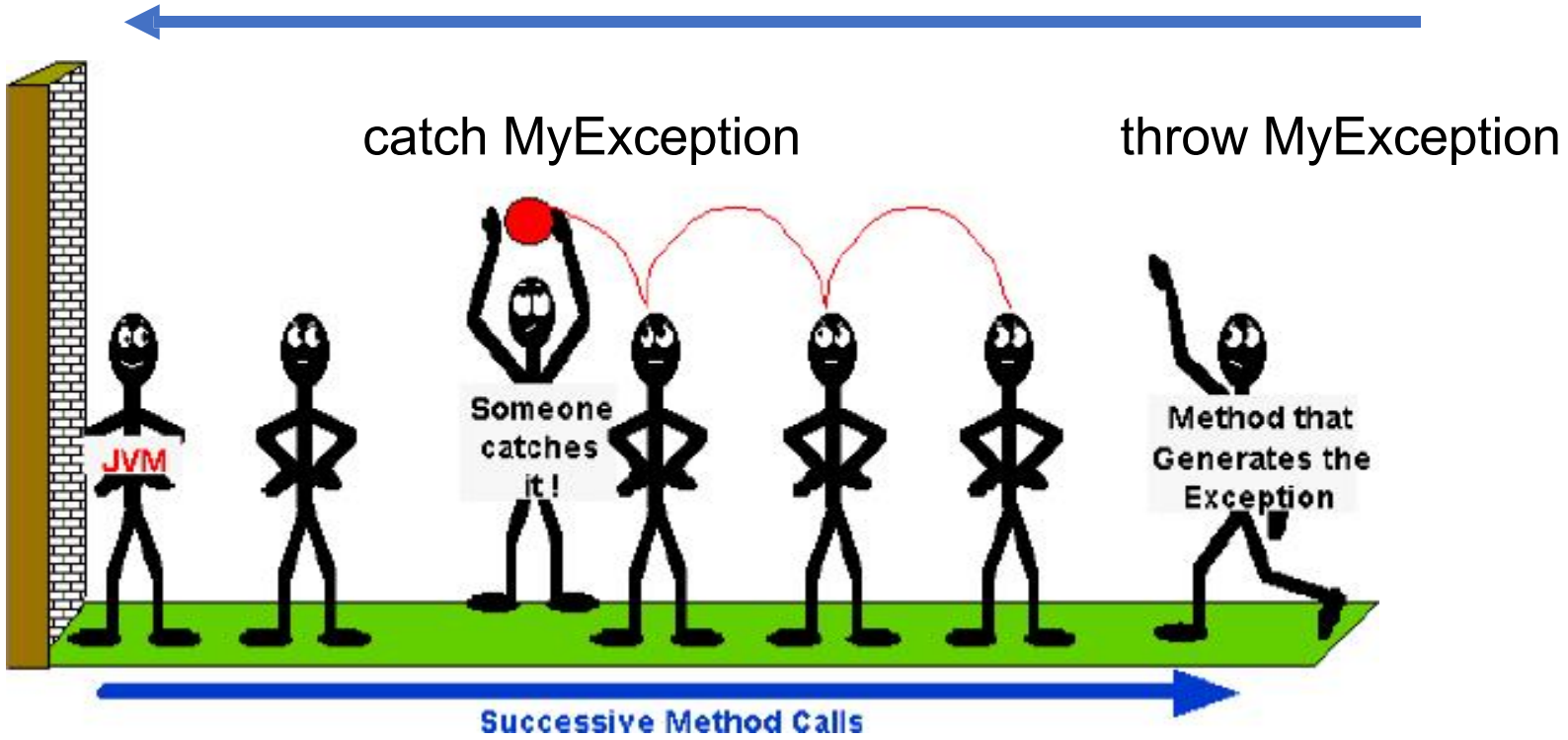
```
try (Scanner scanner = new Scanner(new File("test.txt"))) {
    while (scanner.hasNext()) {
        System.out.println(scanner.nextLine());
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

```
public class MyResource implements AutoCloseable {
    @Override
    public void close() throws Exception {
        System.out.println("My resource is closed.");
    }

}
```

Exception & Call Stack

JVM searches backward through the call stack to find a matching exception handler (i.e., catch)



A() → B() → C() → D() → E() → F()

Y. Li et al., "EH-Recommender: Recommending Exception Handling Strategies Based on Program Context," 2018 (ICECCS)

Where to throw, where to catch?

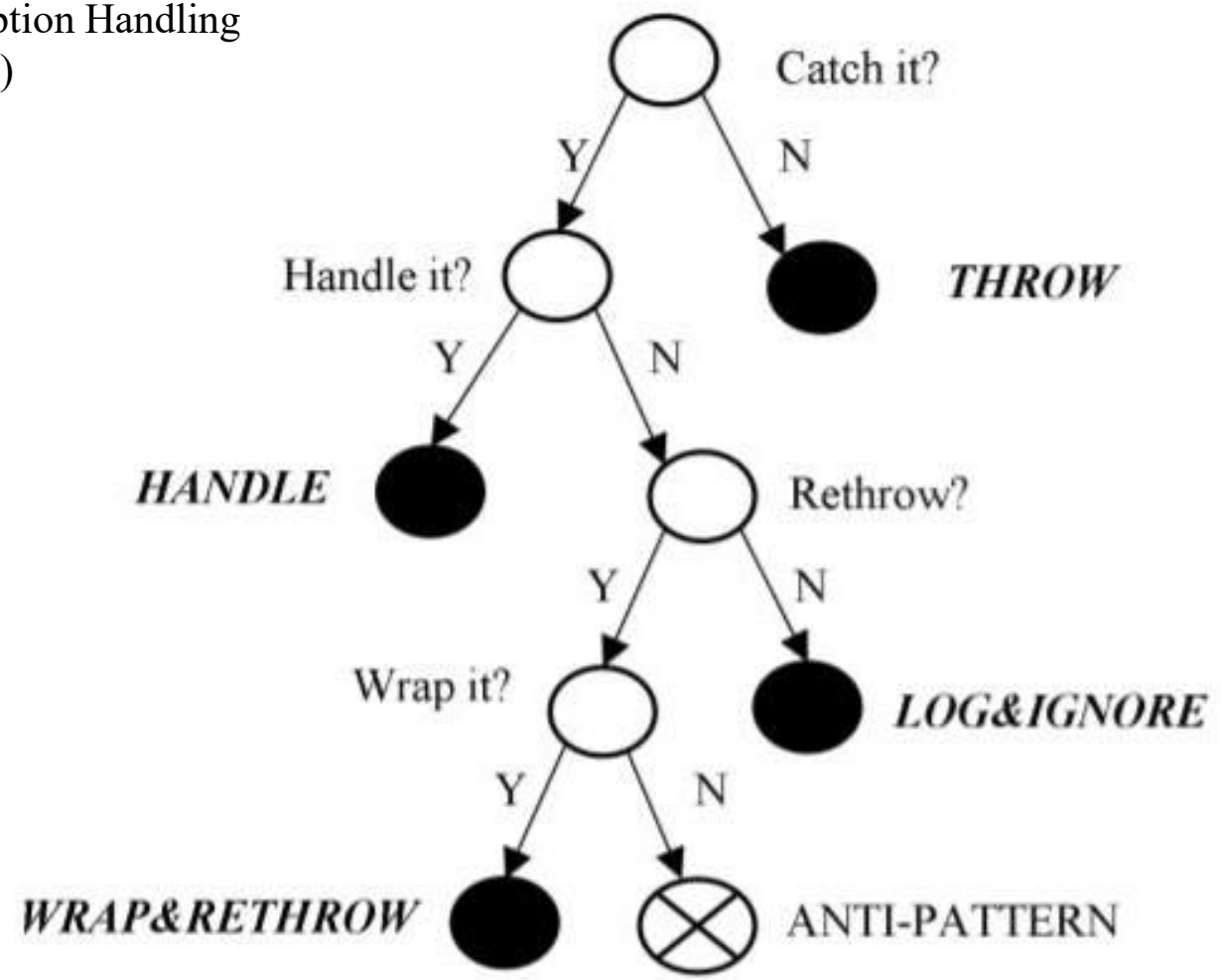


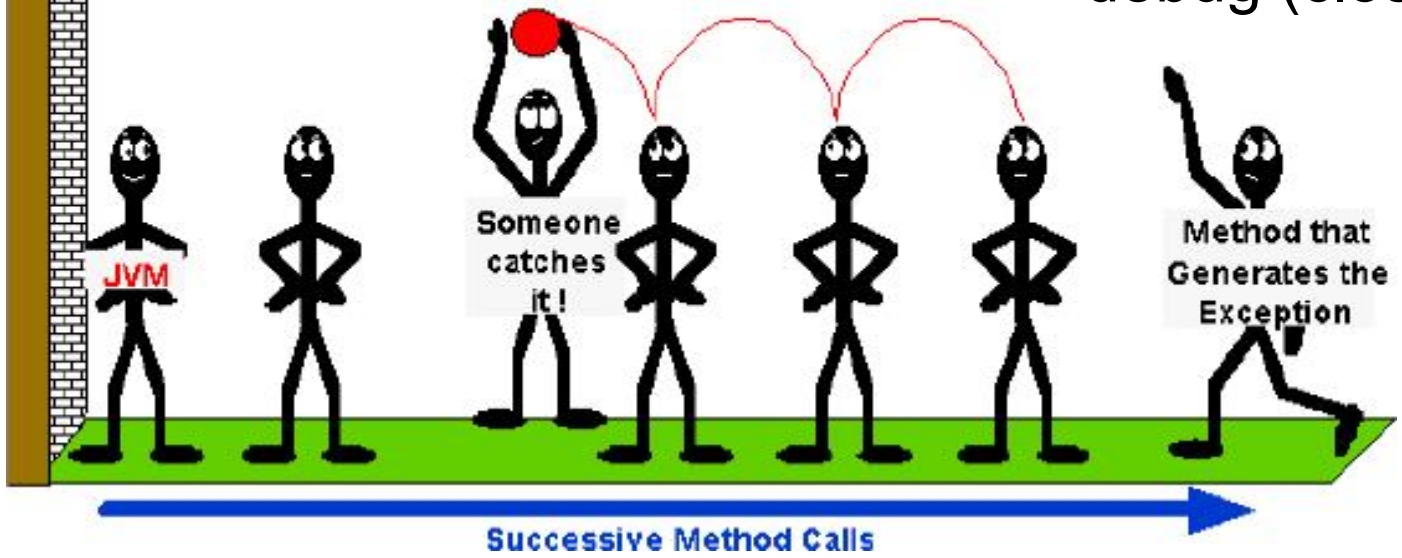
Fig. 1. Decision Process and Exception Handling Strategies

Where to throw, where to catch?

Throw Early, Catch Late

catch at higher level to decide proper actions

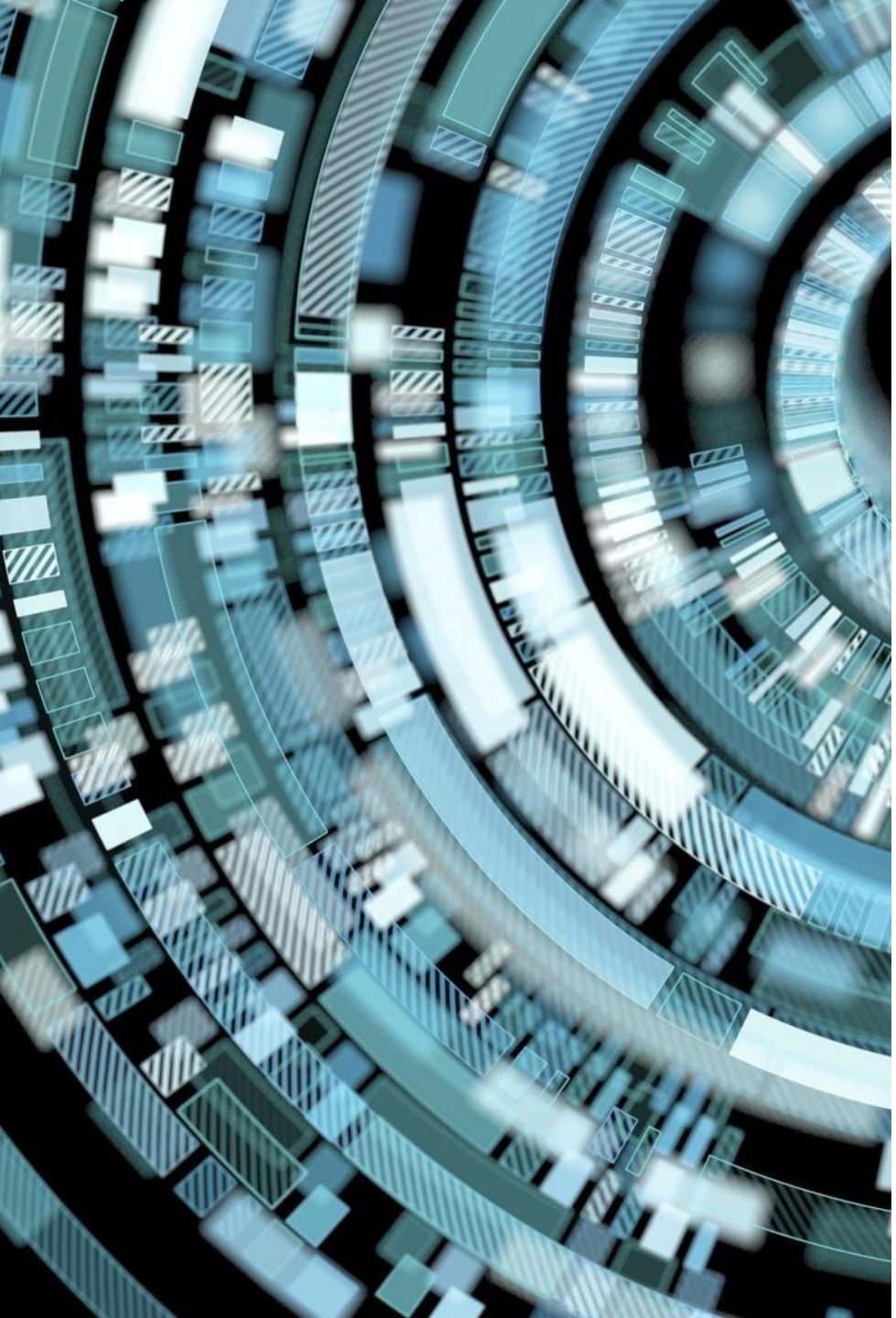
throw at where the error occurs to have more useful context to debug (closer to the cause)



A() → B() → C() → D() → E() → F()

Where to throw, where to catch?

- Further reading
  - Ebert, Felipe, Fernando Castor, and Alexander Serebrenik. "An exploratory study on exception handling bugs in Java programs." Journal of Systems and Software 106 (2015): 82-101.
  - Suman Nakshatri, Maithri Hegde, and Sahithi Thandra. Analysis of exception handling patterns in Java projects: an empirical study. MSR'2016
  - Y. Li et al., "EH-Recommender: Recommending Exception Handling Strategies Based on Program Context," ICECCS'2018
  - Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Yanjun Pu, and Xudong Liu. Learning to Handle Exceptions. ASE'2020



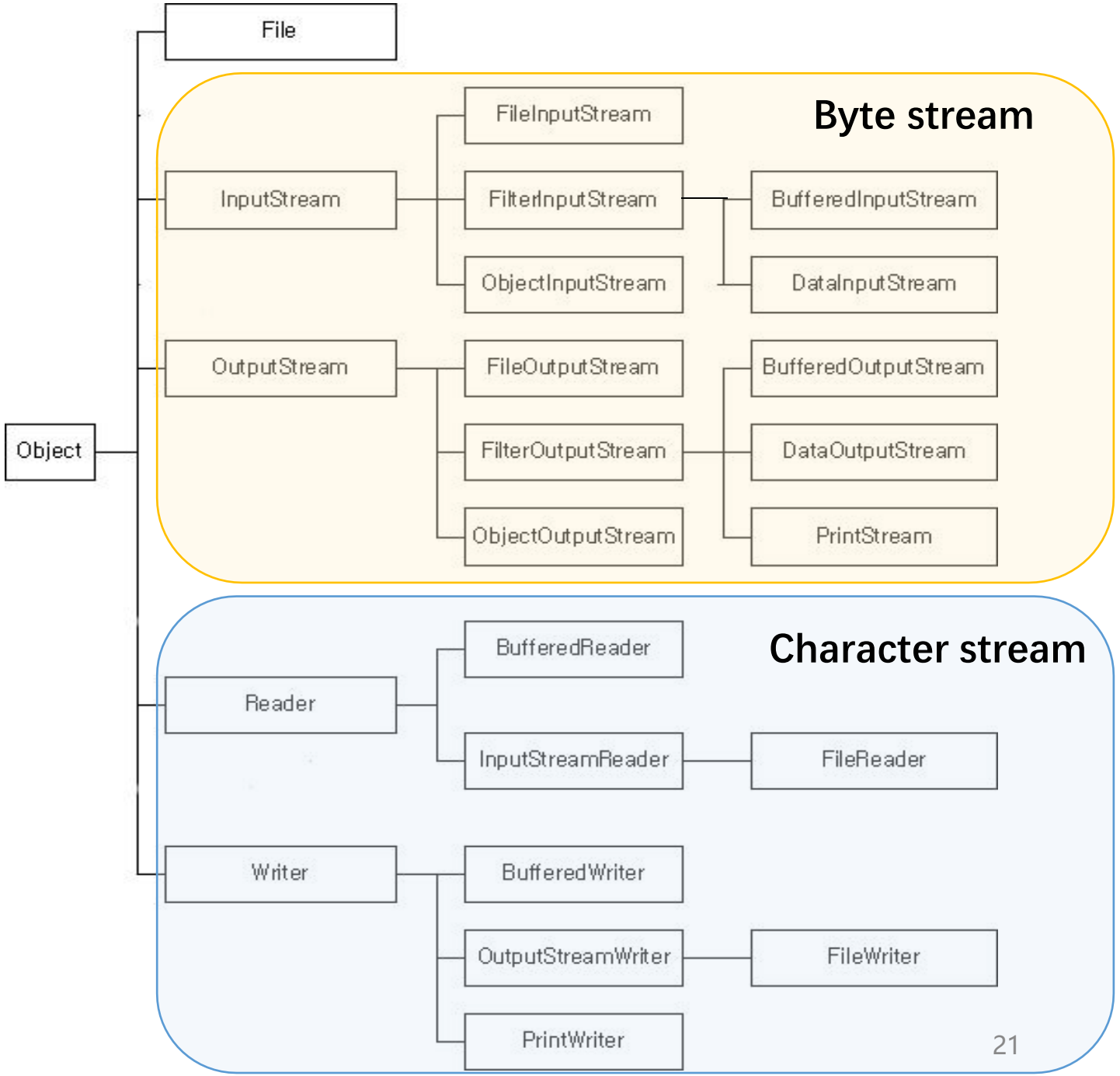
Lecture 2

- Exception Handling
- I/O and Encoding
- NIO and Files
- Persistence and Serialization

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



Internationalization (i18n)

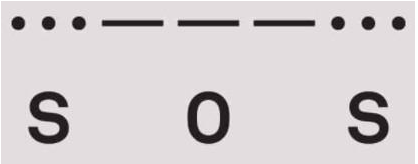
- Internationalization refers to making programs that can take input and output that is tailored to different locations and languages.
- As different languages contain a wide variety of letters and characters, character encoding is an important element to make software systems international or language/location independent.

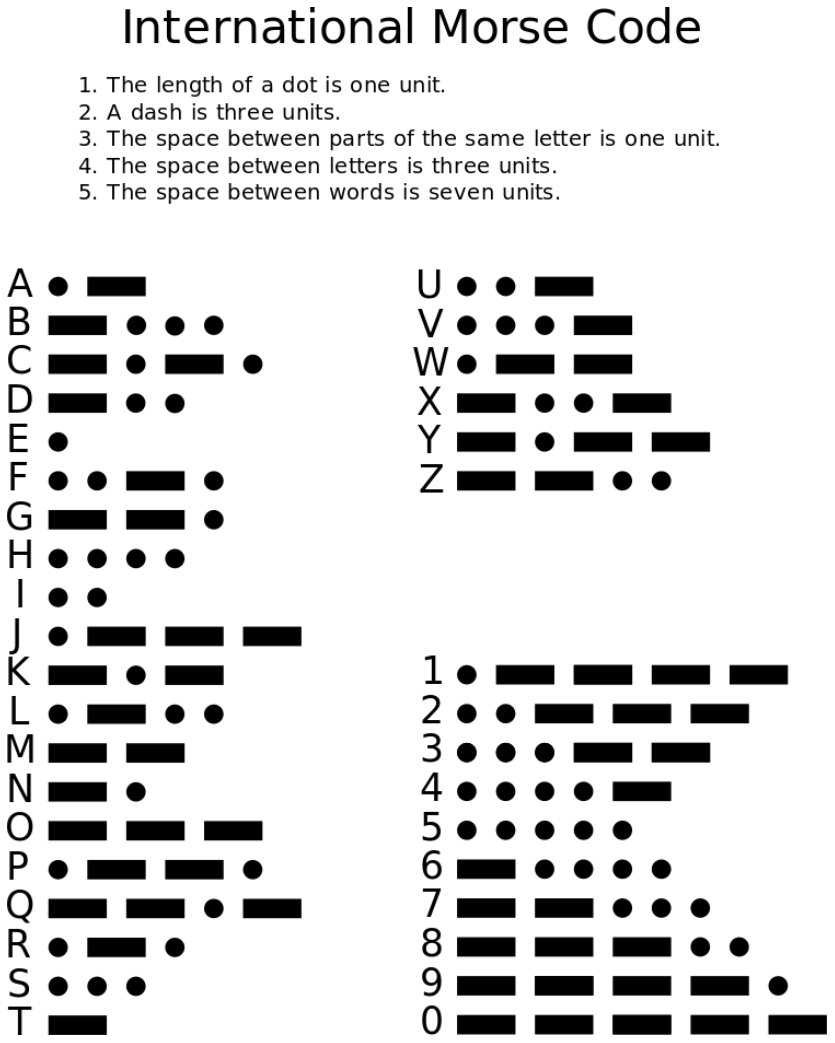
Image source: https://bilingua.io/how-to-say-hello-in-100-languages

Character 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.



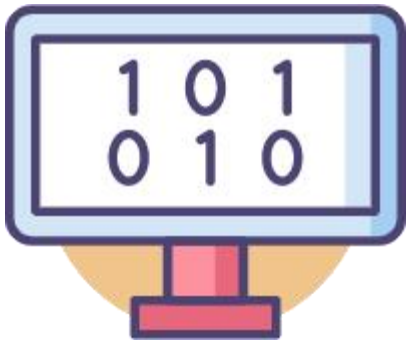


Chinese Telegraph Code (中文电码, 1872)

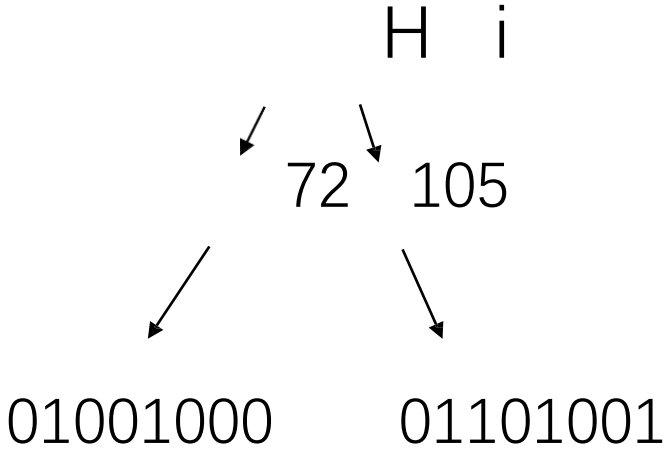
One-to-one mapping between Chinese characters and four-digit numbers from 0000 to 9999

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

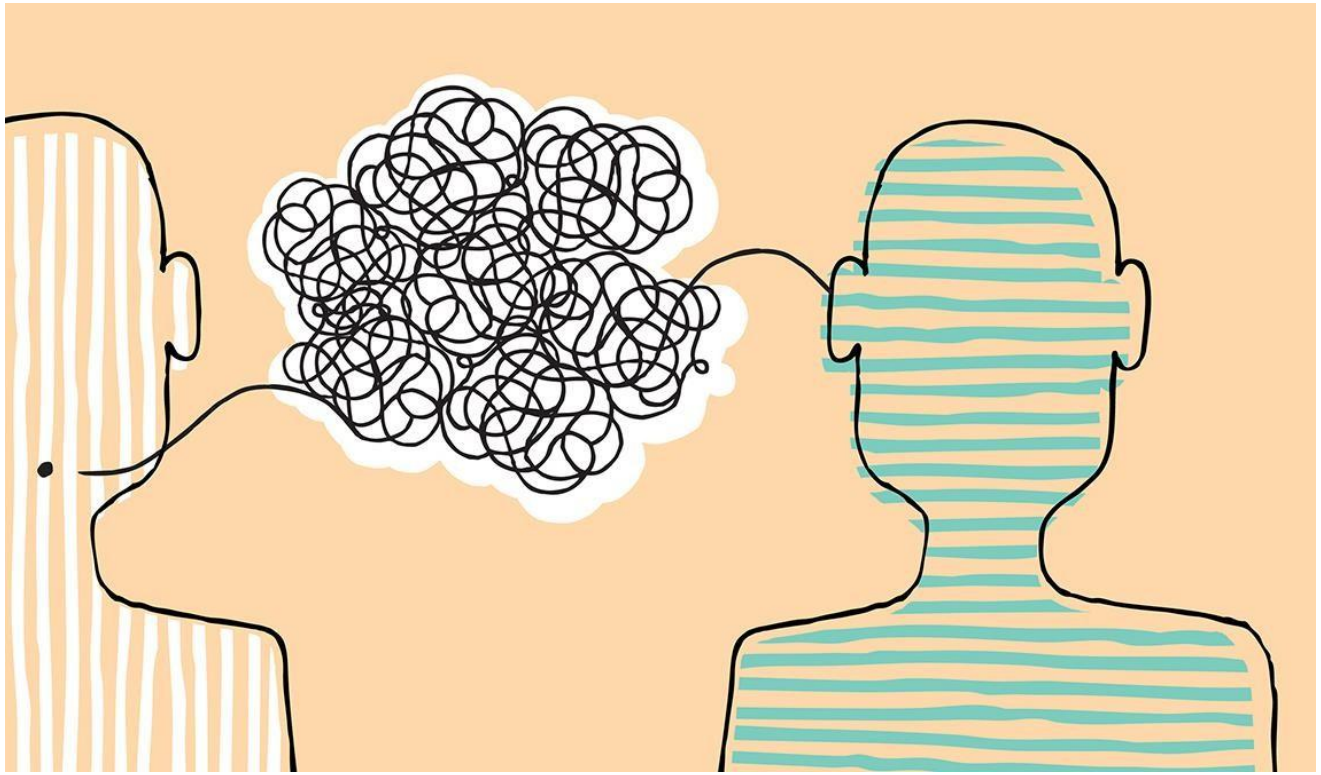


GB2312, GBK

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

编	码
B1E0	C2EB
1011000111100000	1100001011101011

Problems?



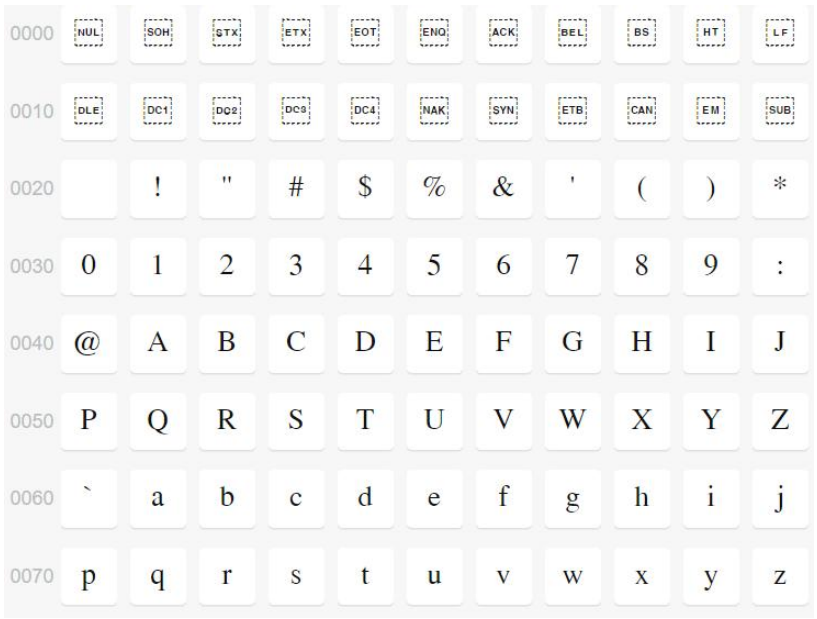
Communication

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)

- Motivated by the need to encode characters in all languages without conflicts
- A standard to consistent encoding of text in most of the languages; It covers 144,697 characters and keeps evolving



Playing Cards in Unicode

U+2660	U+2665	U+2666	U+2663
♠	♥	♦	♣
Black Spade Suit	Black Heart Suit	Black Diamond Suit	Black Club Suit
&spades;	&hearts;	&diams;	&clubs;
U+2664	**U+2661**	**U+2662**	**U+2667**
♤	♡	♢	♧
White Spade Suit	White Heart Suit	White Diamond Suit	White Club Suit

U+1F0A1	U+1F0B1	U+1F0C1	U+1F0D1
🂡	🂱	🃁	🃑
Ace of Spades	Ace of Hearts	Ace of Diamonds	Ace of Clubs
U+1F0A2	**U+1F0B2**	**U+1F0C2**	**U+1F0D2**
🂢	🂲	🃂	🃒
Two of Spades	Two of Hearts	Two of Diamonds	Two of Clubs
U+1F0A3	**U+1F0B3**	**U+1F0C3**	**U+1F0D3**
🂣	🂳	🃃	🃓
Three of Spades	Three of Hearts	Three of Diamonds	Three of Clubs

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 “
- Unicode vs UTF-8
  - Unicode is a standard (defines the mapping)
  - UTF-8 follows the Unicode standard and defines how digits are stored in memory

Unicode vs 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

Encoding Support for Java

- Every implementation of the Java platform is required to support the following standard charsets. Consult the release documentation for your implementation to see if any other charsets are supported.

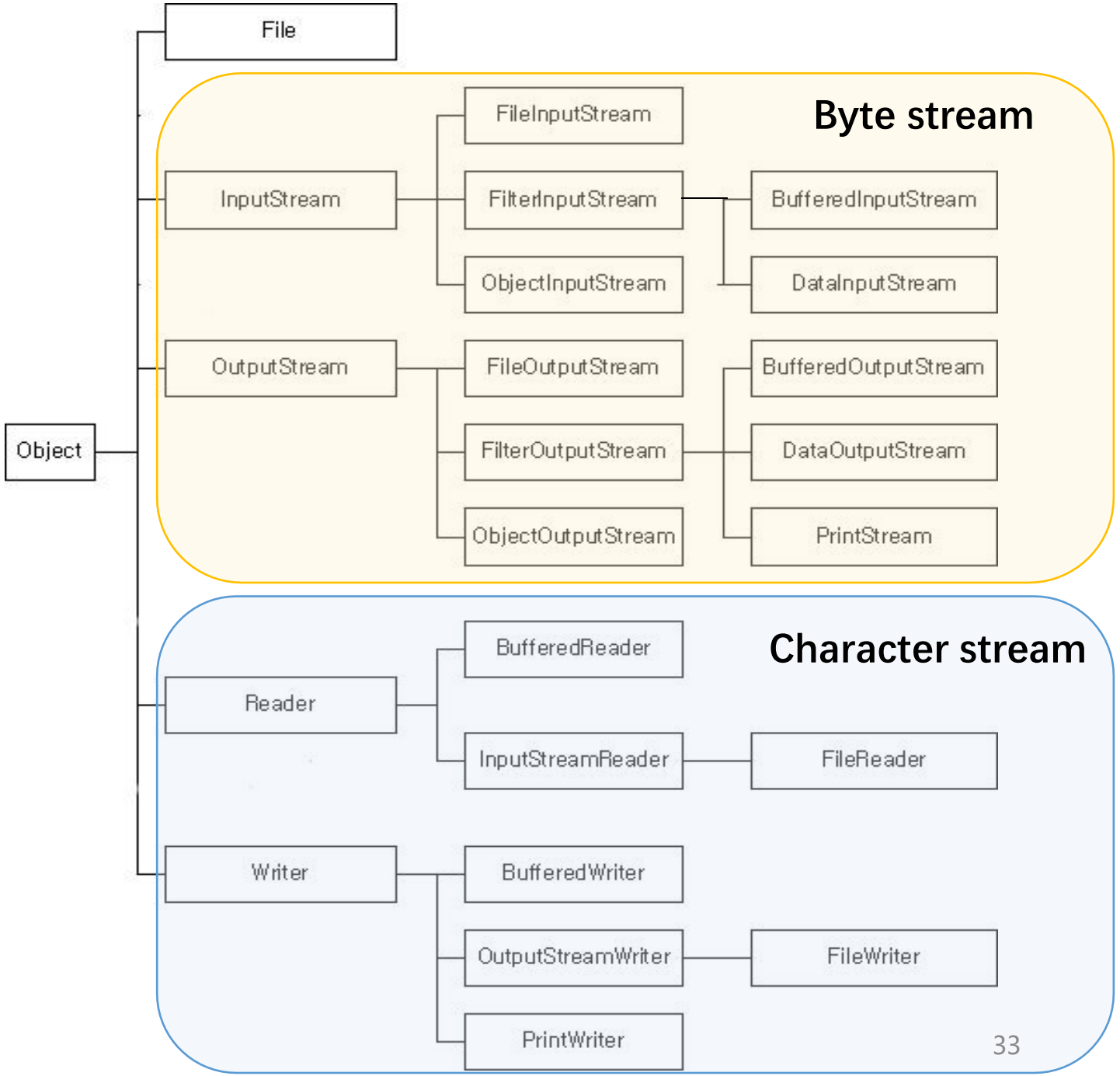
Charset	Description
US-ASCII	Seven-bit ASCII, a.k.a. ISO646-US, a.k.a. the Basic Latin block of the Unicode character set
ISO-8859-1	ISO Latin Alphabet No. 1, a.k.a. ISO-LATIN-1
UTF-8	Eight-bit UCS Transformation Format
UTF-16BE	Sixteen-bit UCS Transformation Format, big-endian byte order
UTF-16LE	Sixteen-bit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

- Use `Charset.defaultCharset().displayName()` to check

Overview

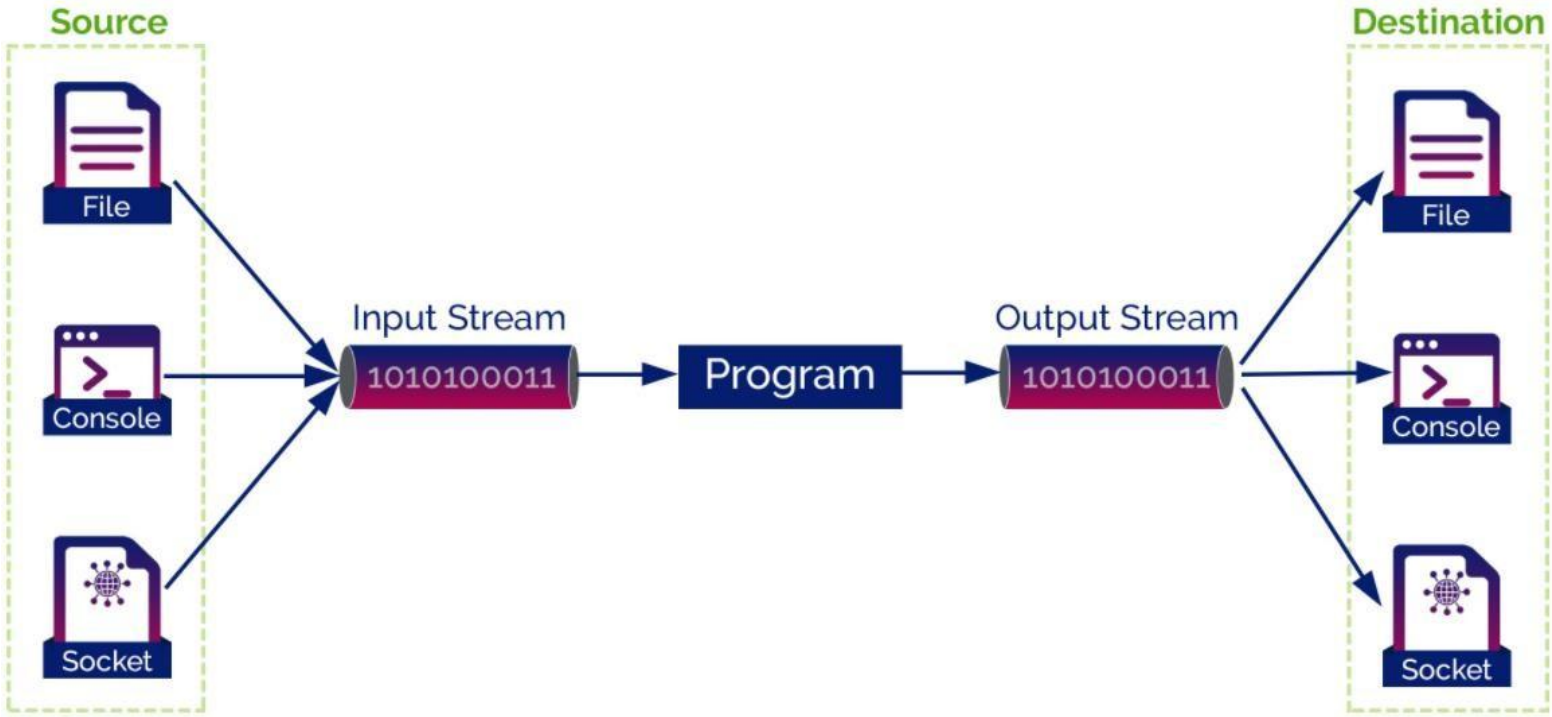
- Java I/O and File are in java.io package
- I/O classification
  - Input and output
  - Byte stream vs Character stream

Character Stream is used to handle
Internationalization



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
- Java provides Byte Stream and Character Stream

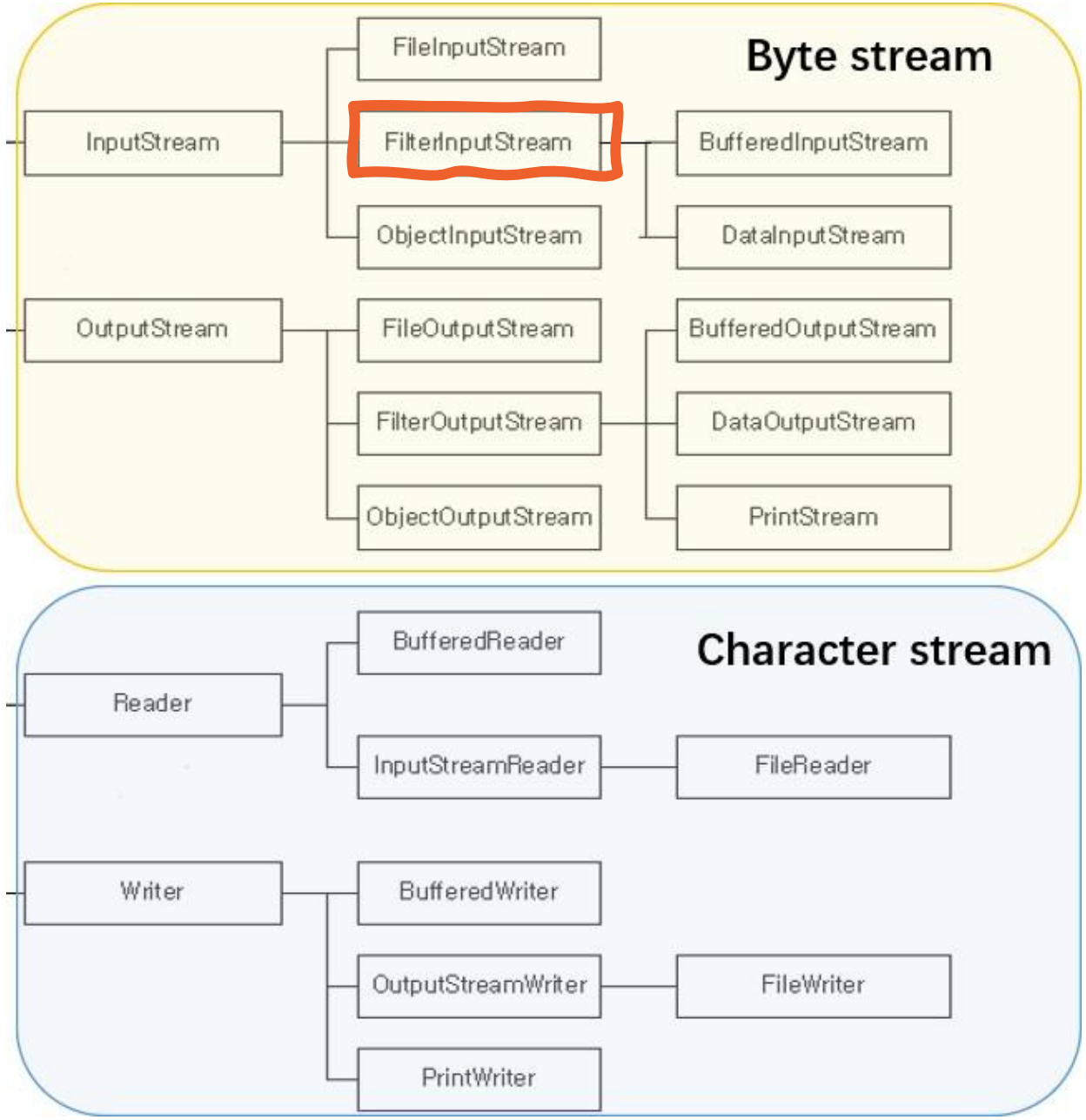


://www.btechsmartclass.com/java/java-Stream.html

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()`

Talk about the differences later!



FileInputStream

Throw IOException

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

try-with-resource

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

What is the output when test.txt contains the text “Hello World”?

72 101 108 108 111 32 87 111 114 108 100

FileInputStream

- Get meaningful character by using (char) or StringBuilder

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

```
try (InputStream input = new FileInputStream("src/test.txt")) {
    int n;
    StringBuilder sbuilder = new StringBuilder();
    while ((n = input.read()) != -1) {
        sbuilder.append((char) n);
    }
   String text = sbuilder.toString();
   System.out.println(text);
}
```

FileInputStream

- 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);
    }
}
```

```
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)

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

Reading 1 byte at a time (split the 2 bytes for 1 word, meaningless)

```
? ? ? ? ? ú ? ? ? ?
```

FileReader

- What if `test.txt` contains “计算机系统” with GBK file encoding (consistent with **my** Java default encoding)?

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

```
 35745 31639 26426 31995 32479
```

```
try (Reader reader = new FileReader("src/test.txt")) {
    int n;
    while ((n = reader.read()) != -1) {
        System.out.print(" " + (char)n);
    }
}
```

```
 计 算 机 系 统
```

- `FileReader` is used to read stream of character (instead of stream of raw bytes as `FileInputStream`)
- Return the read character as an integer (range 0 to 65535)

FileReader

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

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

```
 29825 65284 30075 37832 34425 37108 32513 65533
```

UTF-8 encoding has varied length;
Normal Chinese character often take 3 bytes

```
try (Reader reader = new FileReader("src/test.txt")) {
    int n;
    while ((n = reader.read()) != -1) {
        System.out.print(" " + (char)n);
    }
}
```

```
 璁 ＄ 瞱 鏈 虹 郴 缁 ？
```

FileReader

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

```
try (Reader reader = new FileReader("src/test.txt", StandardCharsets.UTF_8)) {
    int n;
    while ((n = reader.read()) != -1) {
        System.out.print(" " + (char)n);
    }
}
```

计 算 机 系 统

Set FileReader encoding to be consistent with the file encoding

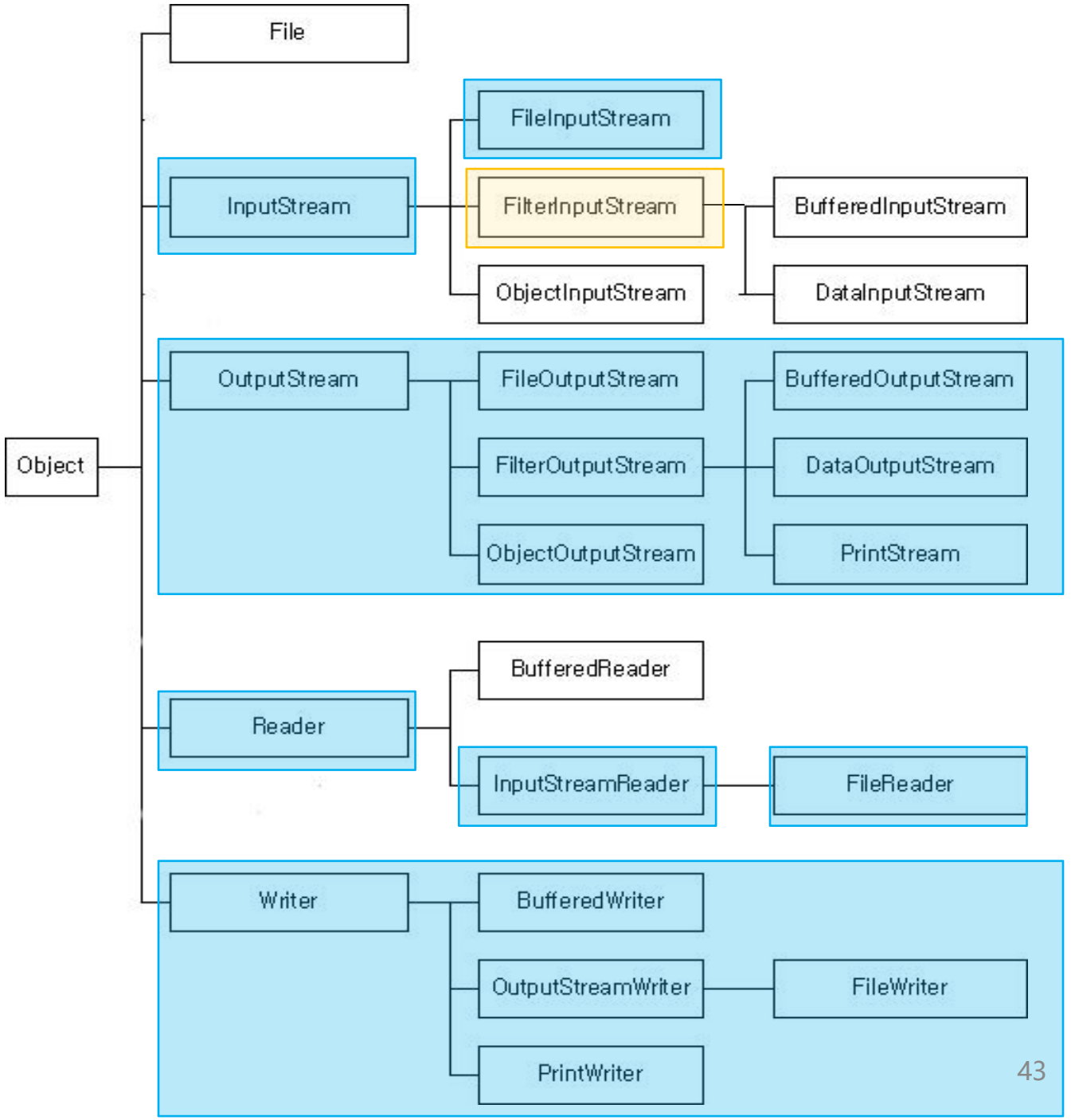
InputStream to Reader

- 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

Where are we?


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

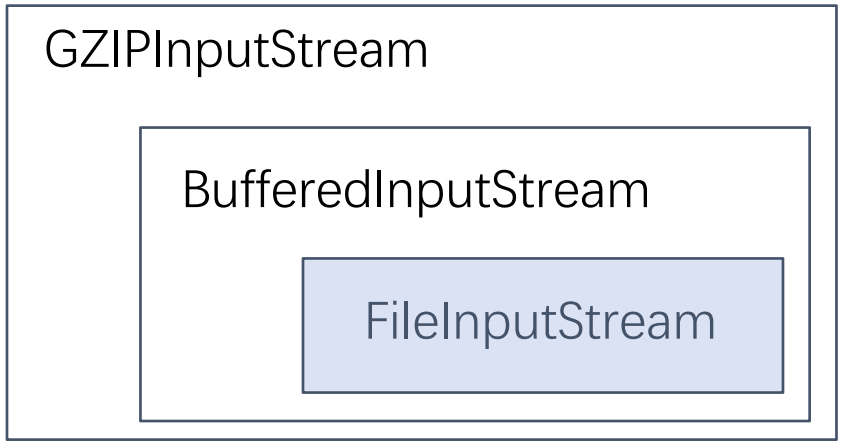
FilterInputStream

- Contains another InputStream as a basic source of data
- Provide additional functionality on top of the original stream

+ gzip functionality

+ buffered functionality

Basic 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

- Direct Known Subclasses
  - BufferedInputStream
  - CheckedInputStream
- CipherInputStream
- DataInputStream
- DeflaterInputStream
- DigestInputStream
- InflaterInputStream
- LineNumberInputStream
- ProgressMonitorInputStream
- PushbackInputStream

System I/O

```
System.out.println("Hello World!");
```

- The `System` class
  - A subclass of Object
  - Is a `final` class that cannot be extended by other classes
  - The constructor is `private`; cannot create an instance of it

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

```
/**
 * This class is uninstantiable.
 */
private System()
{
}
```

System I/O

- Three static fields: in, out, err

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.

Sound familiar?

System.in

```
public static final InputStream in
```

- Standard input, often read keyboard input

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.in with Scanner

java.lang.Object
java.util.Scanner

- Parse the input into different primitive types and strings

```
Scanner input = new Scanner(System.in);

System.out.println("Enter an int: ");
int data = input.nextInt();
System.out.println("Get int: " + data);

System.out.println("Enter a float: ");
float data2 = input.nextFloat();
System.out.println("Get float: " + data2);

System.out.print("Enter a word: ");
String value = input.next();
System.out.println("Get word: " + value);

input.close();
```

```
Enter an int:
2
```

System.out

```
public static final PrintStream out
```

- A `PrintStream` often used to write to command line console
- 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?");
```

System.out.println

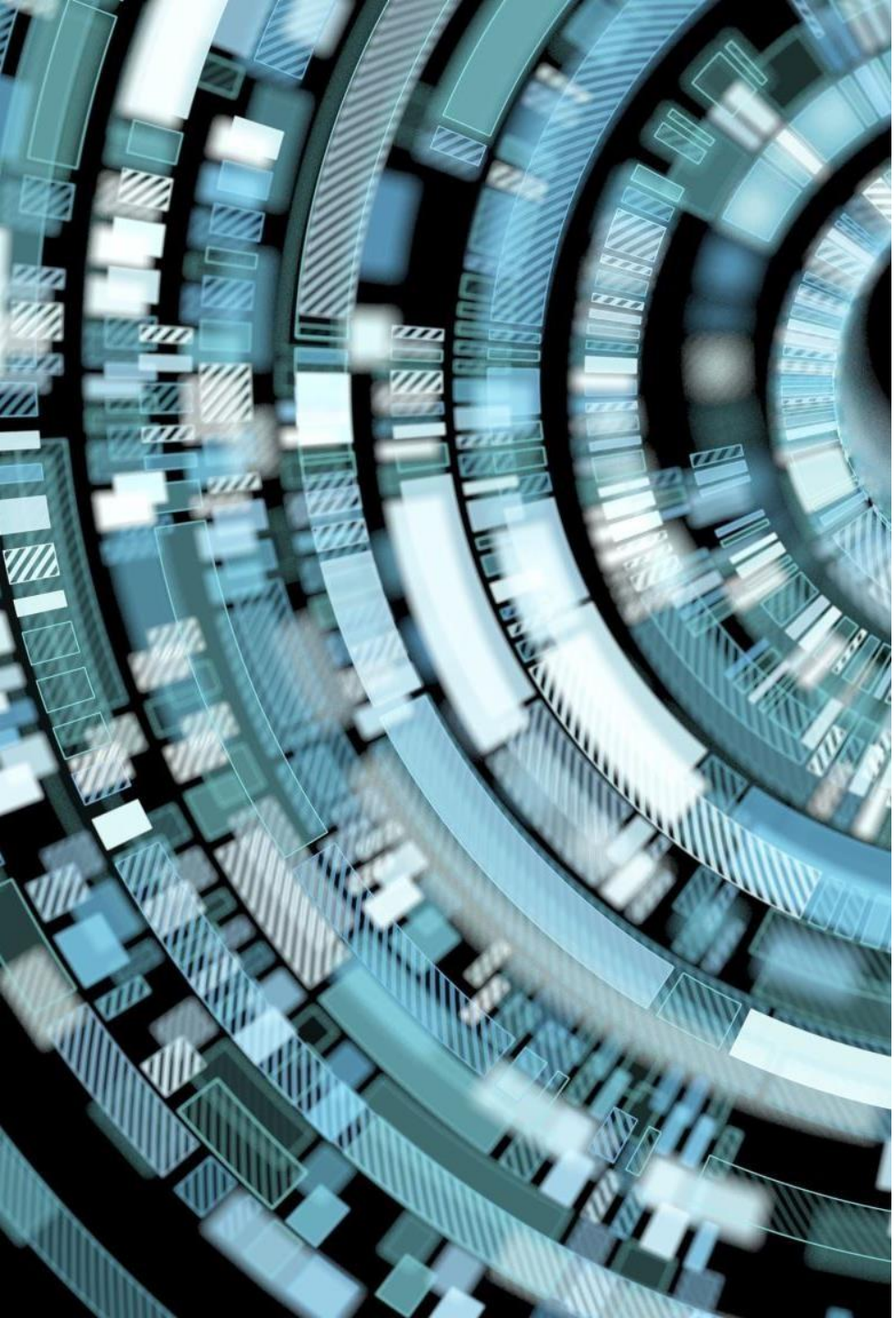
- Performance could be affected for many println()
- All things will be printed with no filter (flooded console)
- Alternatives: logging
  - java.util.logging
  - Open-source logging framework: Log4J, SLF4J, etc.

Log4j software bug is 'severe risk' to the entire internet

A flaw in a commonly used piece of software has left millions of web servers vulnerable to exploitation by hackers

TECHNOLOGY 13 December 2021

Level
FATAL
ERROR
WARNING
INFO
DEBUG
TRACE



Lecture 2

- Exception Handling
- I/O and Encoding
- NIO and Files
- Persistence and Serialization

java.io.File

java.lang.Object
java.io.File

- Support various operations w.r.t. files and directories

Method	Type	Description
`canRead()`	Boolean	Tests whether the file is readable or not
`canWrite()`	Boolean	Tests whether the file is writable or not
`createNewFile()`	Boolean	Creates an empty file
`delete()`	Boolean	Deletes a file
`exists()`	Boolean	Tests whether the file exists
`getName()`	String	Returns the name of the file
`getAbsolutePath()`	String	Returns the absolute pathname of the file
`length()`	Long	Returns the size of the file in bytes
`list()`	String[]	Returns an array of the files in the directory
`mkdir()`	Boolean	Creates a directory

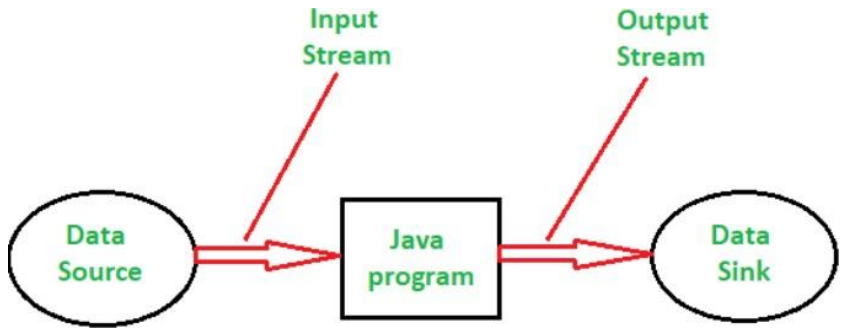
https://www.w3schools.com/java/java_files.asp

Java NIO

- Stands for “New IO” or “Non-blocking IO”
- Alternative IO API for Java introduce from JDK 4

`java.io`
- Stream Oriented
- Blocking I/O operations

`java.nio`
- Buffer Oriented
- Nonblocking I/O Operations



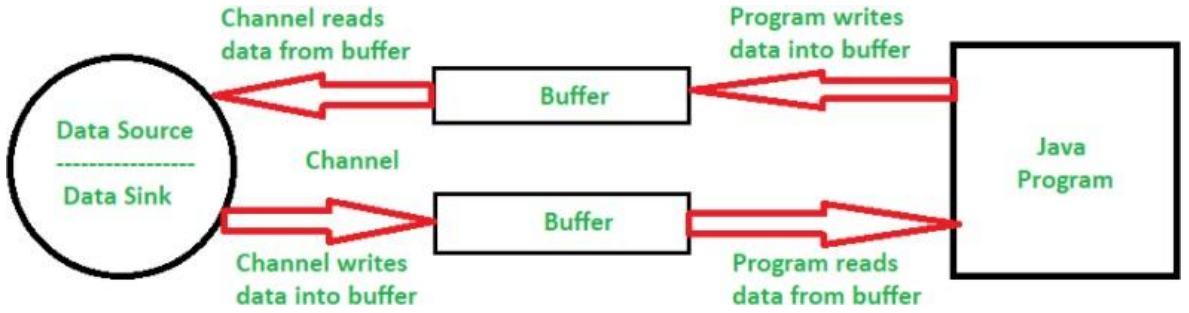


Image source: https://www.geeksforgeeks.org/difference-between-java-io-and-java-nio/

Create a File

```
public void createFile() throws IOException {
    File f = new File("src/myfile.txt");
    boolean success = f.createNewFile();
}
```

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public void createFile() throws IOException {
    Path newFilePath = Paths.get("src/myfile.txt");
    Files.createFile(newFilePath);
}
```

List all files in a directory

```java
public void listFiles() {
    File directory = new File("src/");
    File[] filelist = directory.listFiles();
    for (File file : filelist) {
        System.out.println(file.getAbsolutePath());
    }
}
```

What if there are subdirectories and we want to list files in subdirectories as well?

```
\---folder
    |   file1.txt
    |   file2.txt
    |
    \---subfolder
            file3.txt
            file4.txt
```

List all files in a directory

```
public void listFiles(File directory) {
    File[] filelist = directory.listFiles();
    for (File file : filelist) {
        if(file.isDirectory()) {
            listFiles(file);
        }
        else {
            System.out.println(file.getAbsolutePath());
        }

    }
}
```

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
public void listFiles() throws IOException{
    try (Stream<Path> paths = Files.walk(Paths.get("src/"))) {
        paths.filter(Files::isRegularFile)
            .forEach(System.out::println);
    }
}
```

Files.walk() traverses the directory (file tree) in a depth-first manner

Streams will be introduced in Lecture 4!

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.)

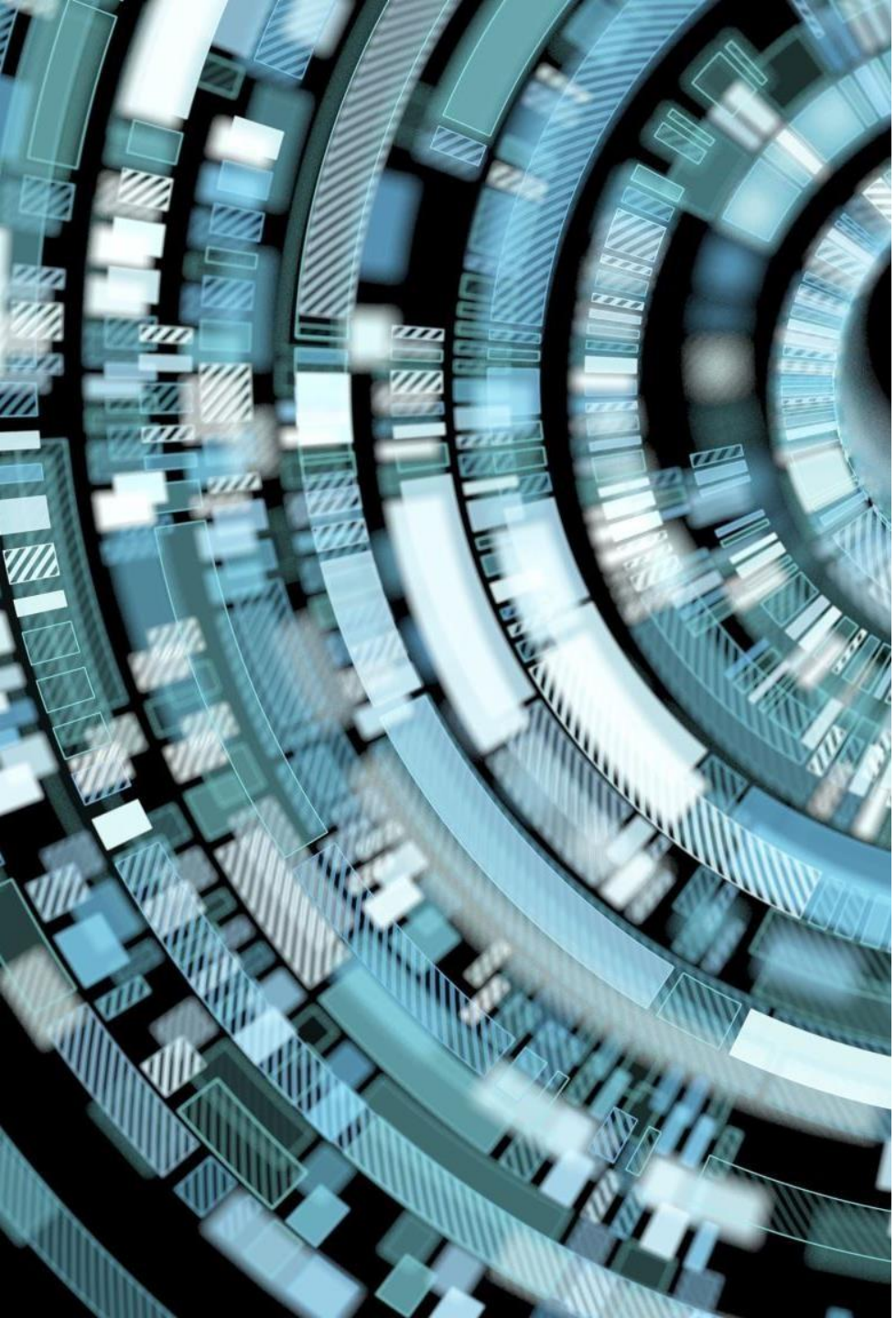
You must simply remember to use File objects in the Scanner constructor:

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

```
public class File
extends Object
implements Serializable, Comparable<File>
```

This lecture

See Lecture 1



Lecture 2

- Exception Handling
- I/O and Encoding
- NIO and Files
- Persistence and Serialization

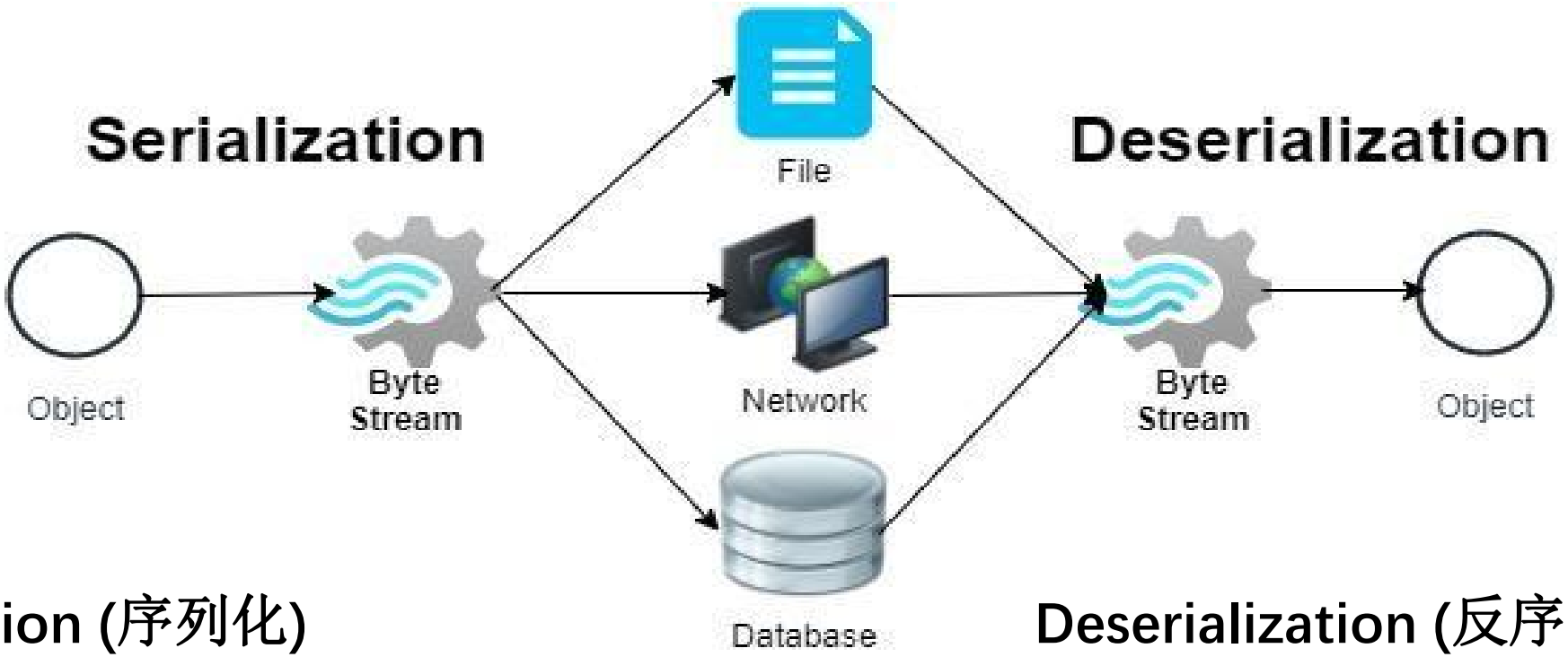
Data Persistence (数据持久化)

- Objects created in Java programs live in memory；they are removed by the garbage collector once they are not used anymore
- What if we want to persist the objects?

Data survives after the process that created it has ended.

Reuse the data without having to executing the program all over again to reach that state.

Data persistence
Store it on a disk, send
it over the network



Serialization (序列化)
Converting the state of an object
into a byte stream

Deserialization (反序列化)
Using the byte stream to recreate
the object in the same state

The `Serializable` Interface

- Classes need to implement the `serializable` interface for their instances to be serialized or deserialized
- The `serializable` interface is an empty interface, without any method or field
- Classes implementing `serializable` do not have to implement any methods
- The `serializable` interface is called a *marker* interface or *tagging* interface (like putting a tag on the class, so the compiler and JVM, when seeing the tag, knows that the object of the class could be serialized)

Example

```
import java.io.Serializable;

public class Student implements Serializable {
    String name;
    String dept;

    public String getName() {
        return name;
    }

    public String getDept() {
        return dept;
    }

    public Student(String name, String dept) {
        this.name = name;
        this.dept = dept;
    }
}
```

Example (cont.)

`ObjectOutputStream` writes primitive data types and Java objects to an `OutputStream`

```
Student student = new Student("Alice", "CS");

// Setup where to store the byte stream
FileOutputStream fos = new FileOutputStream("student.txt");
ObjectOutputStream oos = new ObjectOutputStream(fos);

// serialization
oos.writeObject(student);
```

`ObjectIutputStream` deserializes primitive data and objects previously written using an `ObjectOutputStream`.

```
//Setup where to read the byte stream
FileInputStream fis = new FileInputStream("student.txt");
ObjectInputStream ois = new ObjectInputStream(fis);

// deserialization
Student student2 = (Student)ois.readObject(); // down-casting object

System.out.println(student.getName() + " " + student2.getName());
System.out.println(student.getDept() + " " + student2.getDept());

oos.close();
ois.close();
```

Serialization

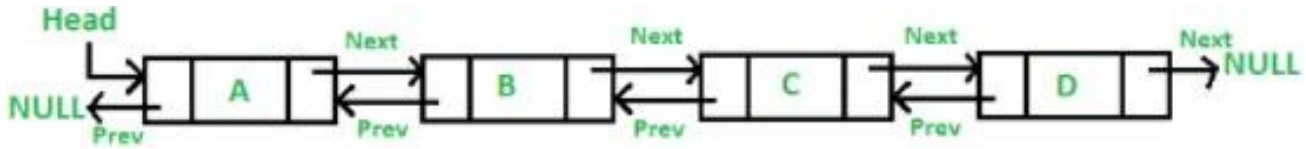
- ✓Enable data persistence
- ✓Easy to use and customize

Example I

```
// Awful candidate for default serialized form
public final class StringList implements Serializable {
    private int size = 0;
    private Entry head = null;

    private static class Entry implements Serializable {
        String data;
        Entry  next;
        Entry  previous;
    }

    ... // Remainder omitted
}
```



Doubly linked list

- The default serialization behavior will serialize every entry and all the links between them in both directions
- Take a long time and consume excessive space

Example from Effective Java, Chapter 12

Example I

"Serialize an object's logic data rather than its physical implementation"

- In the example, we only care about the size of the `StringList` and the data of each entry
- Implement customized `readObject()` and `writeObject()` inside the class to be serialized to replace the default behavior

```
private void writeObject(ObjectOutputStream s)
        throws IOException {
    s.defaultWriteObject();
    s.writeInt(size);

    // Write out all elements in the proper order.
    for (Entry e = head; e != null; e = e.next)
        s.writeObject(e.data);
}
```

- Implement inside the StringList Class
- Only write list size and the value of each entry

Example from Effective Java, Chapter7

Denial-of-Service (DoS) Attack

```
static byte[] bomb() {
    Set<Object> root = new HashSet<>();
    Set<Object> s1 = root;
    Set<Object> s2 = new HashSet<>();
    for (int i = 0; i < 100; i++) {
        Set<Object> t1 = new HashSet<>();
        Set<Object> t2 = new HashSet<>();
        t1.add("foo"); // Make t1 unequal to t2
        s1.add(t1);  s1.add(t2);
        s2.add(t1);  s2.add(t2);
        s1 = t1;
        s2 = t2;
    }
    return serialize(root); // Method omitted for
}
```

- root has 2 HashSet elements, each of which has 2 HashSet elements and so on, 100 level deep
- Deserializing a HashSet instance requires computing the hash code of all its elements
- 2^{100} invocations of the hashCode method, which takes forever

Providing this short byte stream to the target machine, which will take forever to deserialize and unable to provide other services

Example from Effective Java, Chapter 7 13 2

Example II

- Suppose we serialized a `Period` class, which describing valid time ranges, to a byte stream

```
// Byte stream couldn't have come from a real Period instance!
private static final byte[] serializedForm = {
  (byte)0xac, (byte)0xed, 0x00, 0x05, 0x73, 0x72, 0x00, 0x06,
  0x50, 0x65, 0x72, 0x69, 0x6f, 0x64, 0x40, 0x7e, (byte)0xf8,
  0x2b, 0x4f, 0x46, (byte)0xc0, (byte)0xf4, 0x02, 0x00, 0x02,
  0x4c, 0x00, 0x03, 0x65, 0x6e, 0x64, 0x74, 0x00, 0x10, 0x4c,
  0x6a, 0x61, 0x76, 0x61, 0x2f, 0x75, 0x74, 0x69, 0x6c, 0x2f,
  0x44, 0x61, 0x74, 0x65, 0x3b, 0x4c, 0x00, 0x05, 0x73, 0x74,
  0x61, 0x72, 0x74, 0x71, 0x00, 0x7e, 0x00, 0x01, 0x78, 0x70,
  0x73, 0x72, 0x00, 0x0e, 0x6a, 0x61, 0x76, 0x61, 0x2e, 0x75,
  0x74, 0x69, 0x6c, 0x2e, 0x44, 0x61, 0x74, 0x65, 0x68, 0x6a,
  (byte)0x81, 0x01, 0x4b, 0x59, 0x74, 0x19, 0x03, 0x00, 0x00,
  0x78, 0x70, 0x77, 0x08, 0x00, 0x00, 0x00, 0x66, (byte)0xdf,
  0x6e, 0x1e, 0x00, 0x78, 0x73, 0x71, 0x00, 0x7e, 0x00, 0x03,
  0x77, 0x08, 0x00, 0x00, 0x00, (byte)0xd5, 0x17, 0x69, 0x22,
  0x00, 0x78
};
```

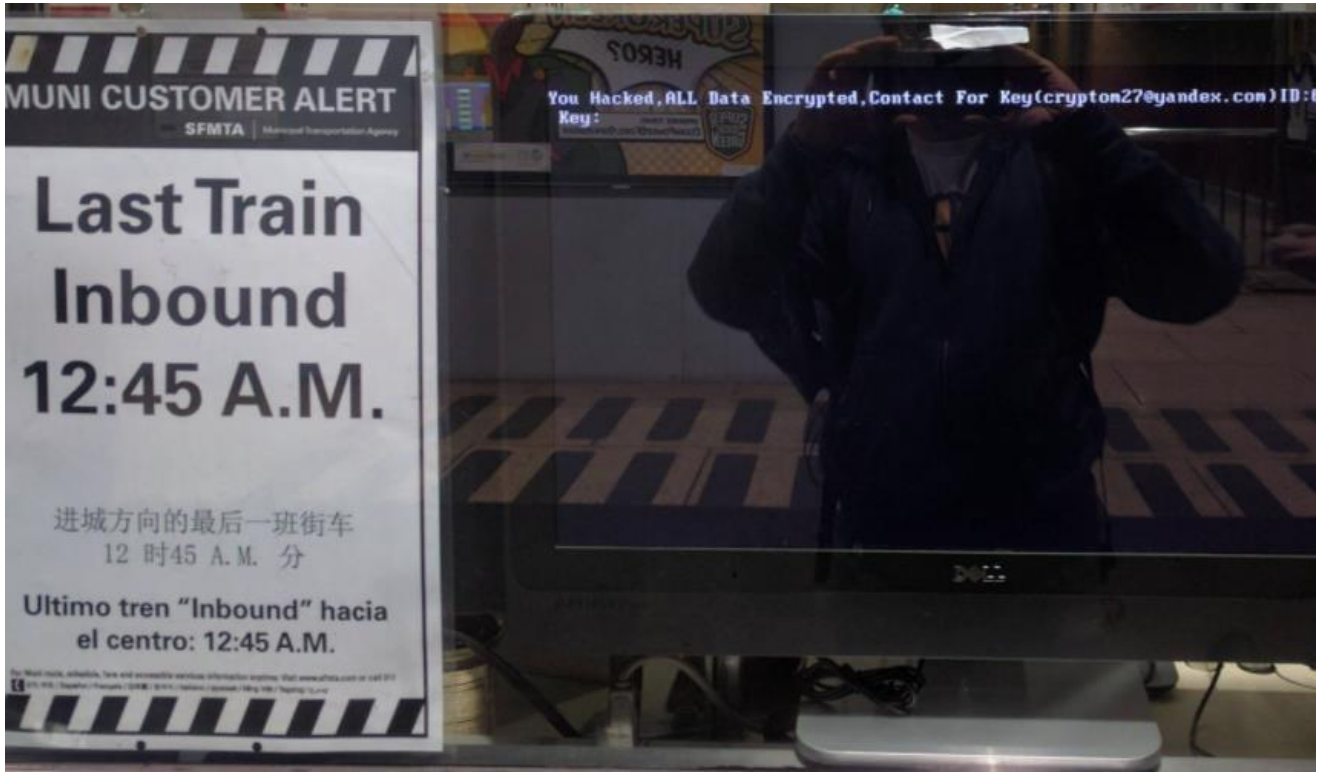
```
private void readObject(ObjectInputStream s)
        throws IOException, ClassNotFoundException {
    s.defaultReadObject();

    // Check that our invariants are satisfied
    if (start.compareTo(end) > 0)
        throw new InvalidObjectException(start +" after "+ end);
}
```

A bad guy modifies the byte stream; So after we deserialize it, we'll get an invalid time period (e.g., Fri. Jan 1 2021 to Sun Jan. 1 2021)

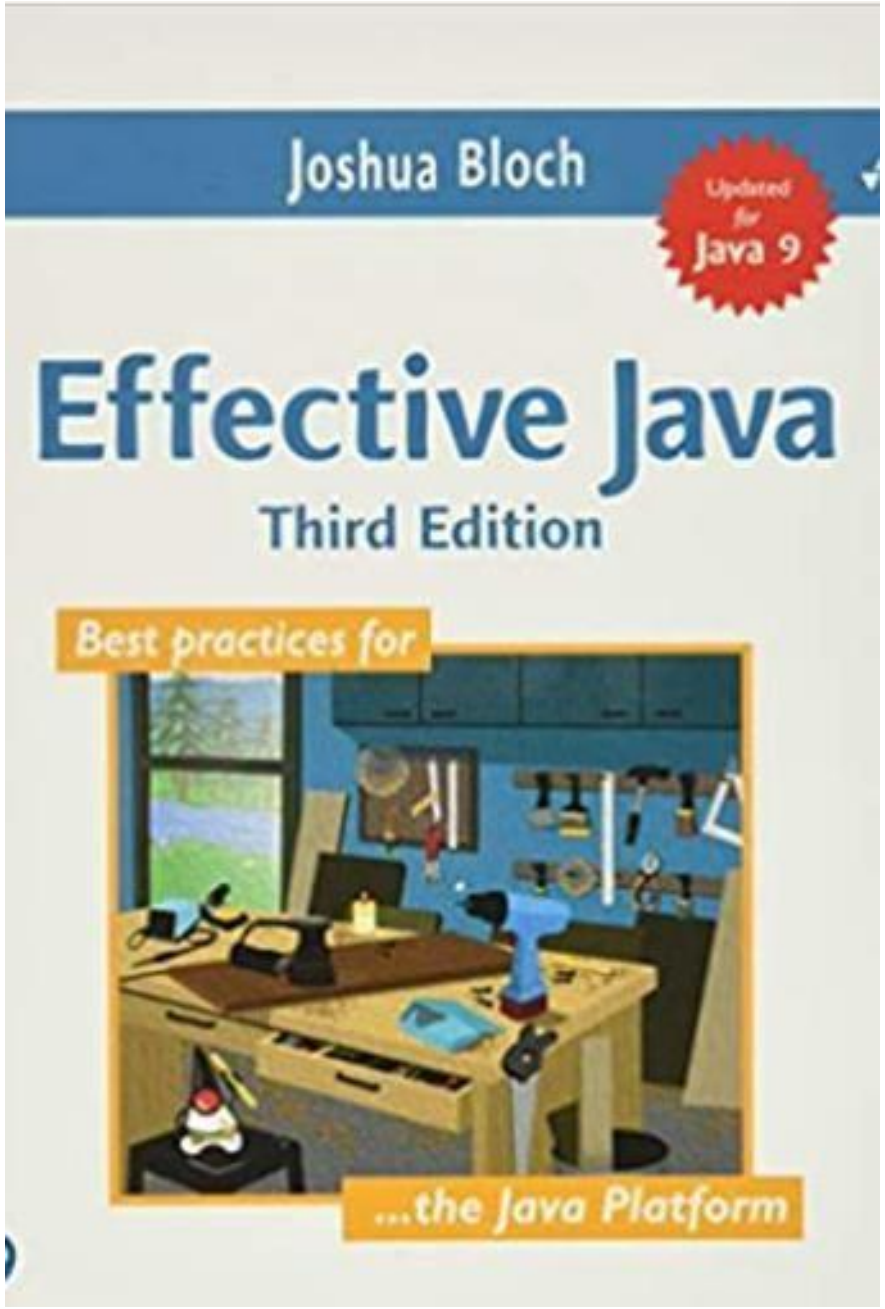
Example from Effective Java, Chapter 12

Attacks exploiting de/serialization



- Attackers could submit a carefully crafted byte stream for the target to deserialize, enable attackers to execute arbitrary code on the target machine (SFMTA Muni Attack)

Example from Effective Java, Chapter 12



Serialization

THIS chapter concerns *object serialization*, which is Java's framework for encoding objects as byte streams (*serializing*) and reconstructing objects from their encodings (*deserializing*). Once an object has been serialized, its encoding can be sent from one VM to another or stored on disk for later deserialization. This chapter focuses on the dangers of serialization and how to minimize them.

Further Reading

Next Lecture

- Generics
- ADT
- Collections