# Files and Streams

## Using Streams, Files, Serialization


**SoftUni Team**

**Technical Trainers**

Software University

SoftUni

Software University

https://softuni.bg

# Table of Contents

# Have a Question?

Software University

sli.do

#java-advanced

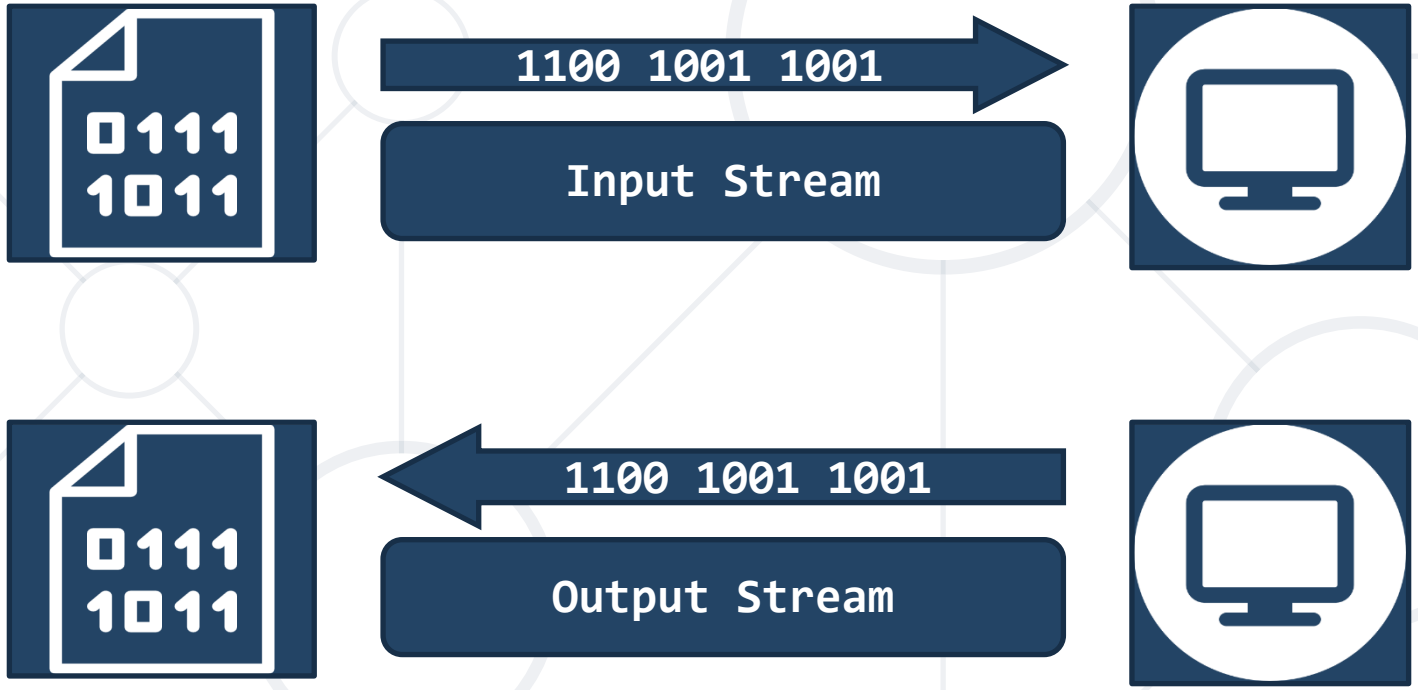
# Streams

# What is Stream?

- **Streams** are used to **transfer data**
- We **open a stream** to:
  - **Read** a file
  - **Write** to a file

# Streams Basics

- Two fundamental types of streams:




**Streams are unidirectional!**

# Opening a File Stream

```
String path = "C:\\input.txt";

FileInputStream fileStream =
                new FileInputStream(path);

int oneByte = fileStream.read();
while (oneByte >= 0) {
  System.out.print(oneByte);
  oneByte = fileStream.read();
}
```

Returns -1 if empty

# Closing a File Stream (1)

- Using **try-catch-finally**

```
InputStream in = null;
try {
    in = new FileInputStream(path);
} catch (IOException e) {
    // TODO: handle exception
} finally {
    if (in != null) {
        in.close();
    }
}
```

close() can also throw an exception

Always free resources!

# Closing a File Stream (2)

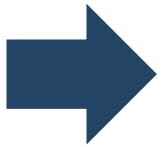
- Using **try-with-resources**

```
try (InputStream in = new FileInputStream(path)) {
  int oneByte = in.read();
  while (oneByte >= 0) {
    System.out.print(oneByte);
    oneByte = in.read();
  }
} catch (IOException e) {
  // TODO: handle exception
}
```

# Problem: Read File

- You are given a file
- Read and print all of its contents **as a sequence of bytes**
- Submit in Judge **only the output** of the program

```
Two households, both
alike in dignity,
In fair Verona, where we
lay our scene,
```



```
1010100 1110111 1101111
100000 1101000 1101111
1110101 1110011 1100101
1101000…
```

Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Solution: Read File

```
String path = "D:\\input.txt";

try (InputStream in = new FileInputStream(path)) {
  int oneByte = in.read();
  while (oneByte >= 0) {
    System.out.printf("%s ",
      Integer.toBinaryString(oneByte));
    oneByte = in.read();
  }
}
catch (IOException e) {
  e.printStackTrace();
}
```

# Problem: Write to File

- Read a file and write all its content while **skipping any punctuation** (skip ',', '.', '!', '?')
- Submit in Judge **only the output** of the program

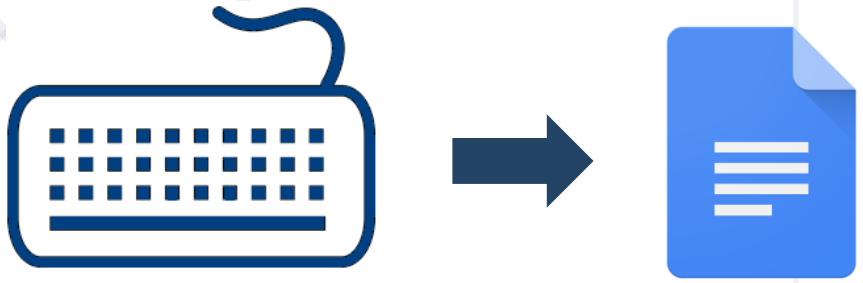
```
Two households, both
alike in dignity.
In fair Verona, where
we lay our scene.
```

```
Two households both
alike in dignity
In fair Verona where
we lay our scene
```

Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Solution: Write to File (1)

```
String inputPath = "D:\\input.txt";
String outputPath = "D:\\output.txt";

List<Character> symbols = new ArrayList<>();
Collections.addAll(symbols, '.', ',', '!', '?');

// continues…
```



Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Solution: Write to File (2)

```
try (InputStream in = new FileInputStream(inputPath);
     OutputStream out = new FileOutputStream(outputPath))
{
  int oneByte = 0;
  while ((oneByte = in.read()) >= 0) {
    if (!symbols.contains((char)oneByte)) {
      out.write(oneByte);
    }
  }
} // TODO: handle exceptions
```

Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Basic Stream Types in Java

## Byte and Character

# Byte Stream

- Byte streams are the **lowest level streams**
  - Byte streams can read or write **one byte at a time**
  - All byte streams **descend** from **InputStream** and **OutputStream**

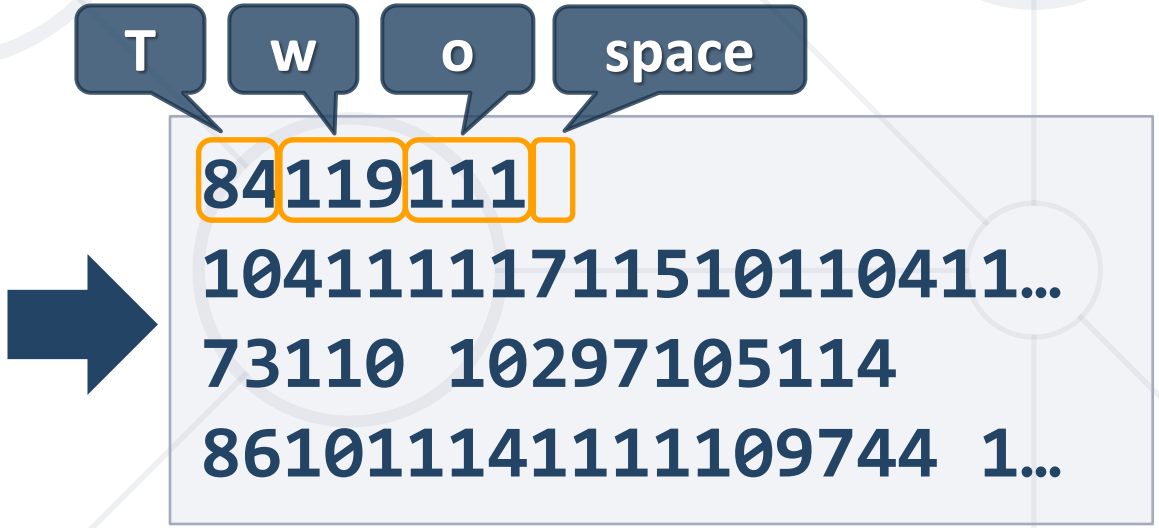
| InputStream | 100101 | 111111 | 100011 | -1 |
|---|---|---|---|---|

| OutputStream | 100101 | 111111 | 100011 | |
|---|---|---|---|---|

# Problem: Copy Bytes

- **Read a file** and **copy** its contents **to another text file**
- Write characters **as bytes** in decimal
- Write **every space or new line as it is**, e.g. as a space or new line

```
Two households, both
alike in dignity.
In fair Verona, where
we lay our scene.
```




Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Solution: Copy Bytes

```
// TODO: Open input and output streams
int oneByte = 0;
while ((oneByte = in.read()) >= 0) {
  if (oneByte == 10 || oneByte == 32) {
    out.write(oneByte);
  } else {
    String digits = String.valueOf(oneByte);
    for (int i = 0; i < digits.length(); i++)
      out.write(digits.charAt(i));
  }
} // TODO: handle exceptions
```

Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Character Streams

- All character streams descend from `FileReader` and `FileWriter`

```
String path = "D:\\input.txt";

FileReader reader = new FileReader(path);
```

# Combining Streams

- Character streams are often "**wrappers**" for byte streams
  - **FileReader** uses **FileInputStream**
  - **FileWriter** uses **FileOutputStream**

```
String path = "D:\\input.txt";

Scanner reader =
  new Scanner(new FileInputStream(path));
```

Wrapping a Stream

# Problem: Extract Integers

- **Read a file** and **extracts all integers** in a separate file
- Get only numbers that are **not a part of a word**
- Submit in Judge **only the output** of the program

```
2 households, 22 alike
in 3nity,
In fair Verona, where
we lay our scene
```

```
2
22
```

Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

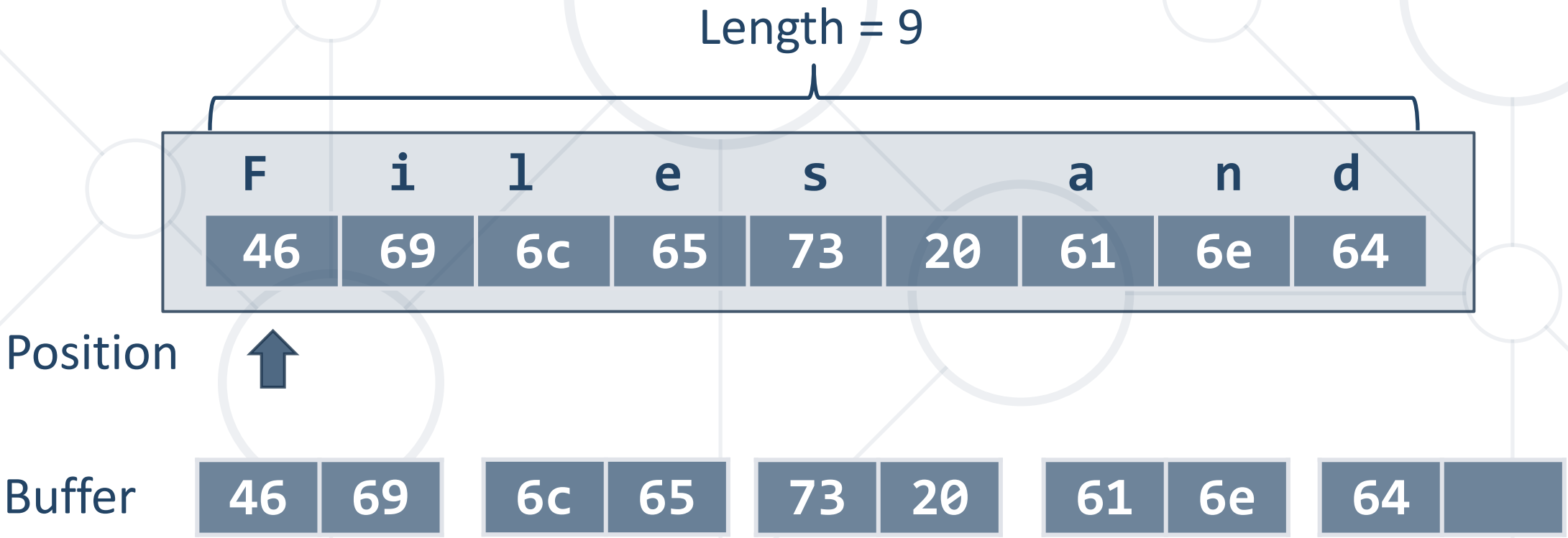
# Solution: Extract Integers

```
Scanner scanner =
      new Scanner(new FileInputStream(inputPath));

PrintWriter out =
      new PrintWriter(new FileOutputStream(outputPath));

while (scanner.hasNext()) {
  if (scanner.hasNextInt())
    out.println(scanner.nextInt());

  scanner.next();
}
out.close();
```

Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Buffered Streams

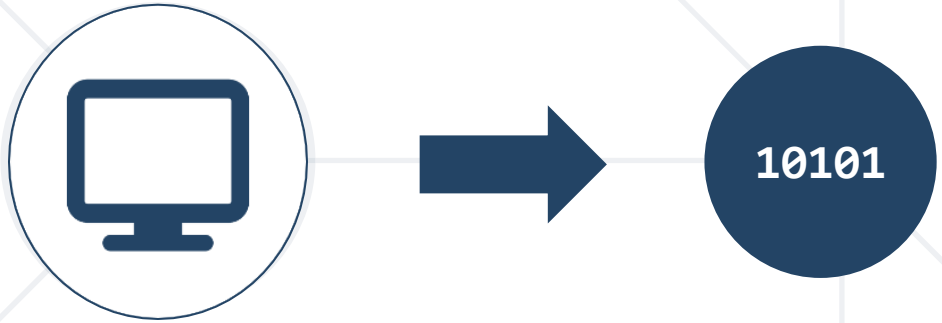
- Reading information in **chunks**
- Significantly **boost performance**



# Problem: Write Every Third Line

- Read a file and **write all lines which number is divisible by 3** in a separate file
- Line numbers start from **one**

```
Two households, both
alike in dignity,
In fair Verona, where
we lay our scene,
From ancient grudge
break to new mutiny...
```




Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Solution: Write Every Third Line

```
try (BufferedReader in =
      new BufferedReader(new FileReader(inputPath));
     PrintWriter out =
      new PrintWriter(new FileWriter(outputPath))) {
  int counter = 1;
  String line = in.readLine();
  while (line != null) {
    if (counter % 3 == 0)
      out.println(line);
    counter++;
    line = in.readLine();
  }
} // TODO: handle exceptions
```

Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

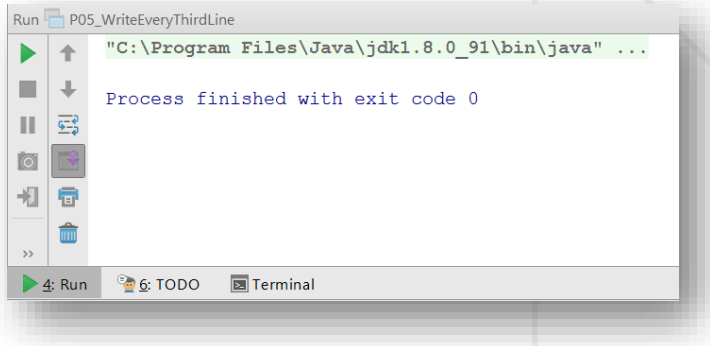
# Command Line I/O (1)

- Standard Input - **System.in**
- Standard Output - **System.out**
- Standard Error - **System.err**

Input Stream

```
Scanner scanner = new Scanner(System.in);
String line = scanner.nextLine();
System.out.println(line);
```

Output Stream



# Command Line I/O (2)

```
public static void main(String[] args) throws IOException {
  BufferedReader reader =
        new BufferedReader(new InputStreamReader(System.in));

  String hello = reader.readLine(); // Hello BufferedReader
  System.out.println(hello);        // Hello BufferedReader
}
```

# Files and Paths

## Easily Working With Files

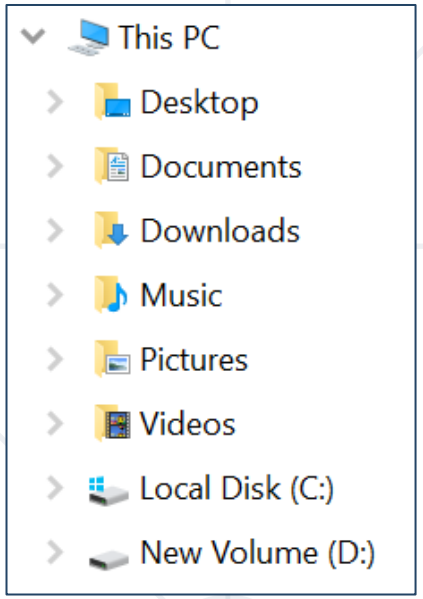
# Paths

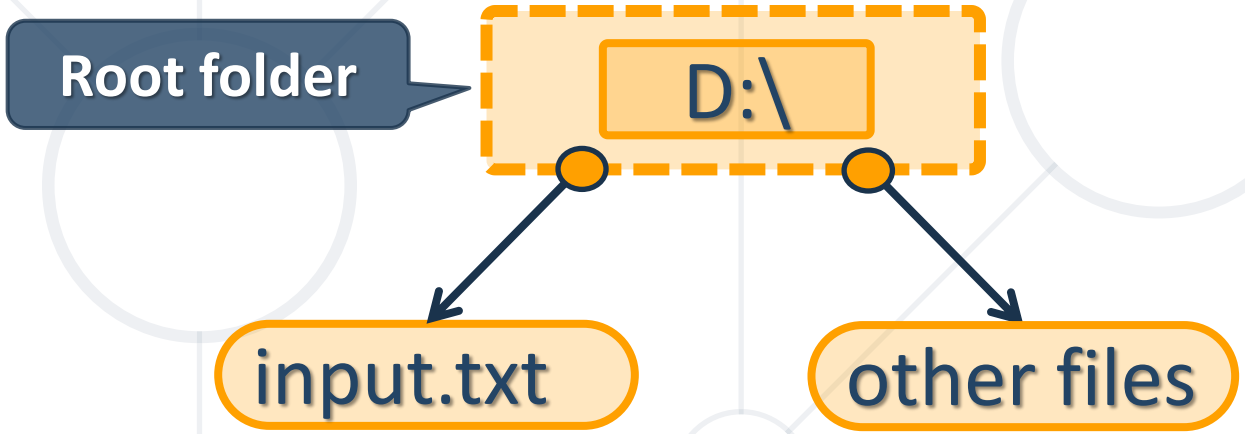
- The location of a file in the file system

```
D:\input.txt
```

- Represented in Java by the Path class

```
Path path = Paths.get("D:\\input.txt");
```



# Files (1)

- Provides **static methods** for **creating streams**

```
Path path = Paths.get("D:\\input.txt");

try (BufferedReader reader =
      Files.newBufferedReader(path)) {
  // TODO: work with file
} catch (IOException e) {
  // TODO: handle exception
}
```

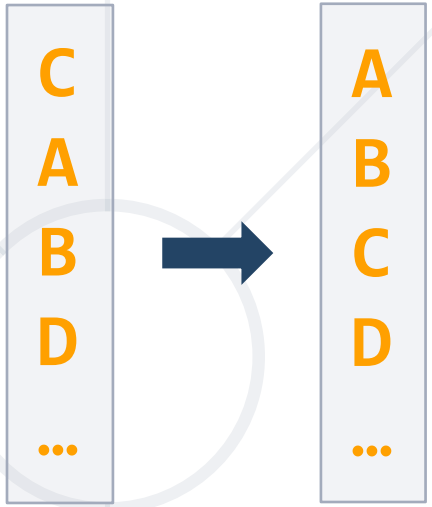
# Files (2)

- Provides **utility** methods for easy file manipulation

```
Path inPath = Paths.get("D:\\input.txt");
Path outPath = Paths.get("D:\\output.txt");

List<String> lines = Files.readAllLines(inPath);
Files.write(outPath, lines);

// TODO: handle exceptions
```

# Problem: Sort Lines

- **Read** a text **file** and **sort all lines**
- **Write** the result to another text **file**
- Use Paths and Files classes





Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Solution: Sort Lines

```
Path path = Paths.get("D:\\input.txt");
Path output = Paths.get("D:\\output.txt");

try {
  List<String> lines = Files.readAllLines(path);
  lines = lines.stream().filter(l ->
        !l.isBlank()).collect(Collectors.toList());
  Collections.sort(lines);
  Files.write(output, lines);
} catch (IOException e) {
  e.printStackTrace();
}
```

Don't use for large files

Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab


# File Class in Java

## Easily Working With Files

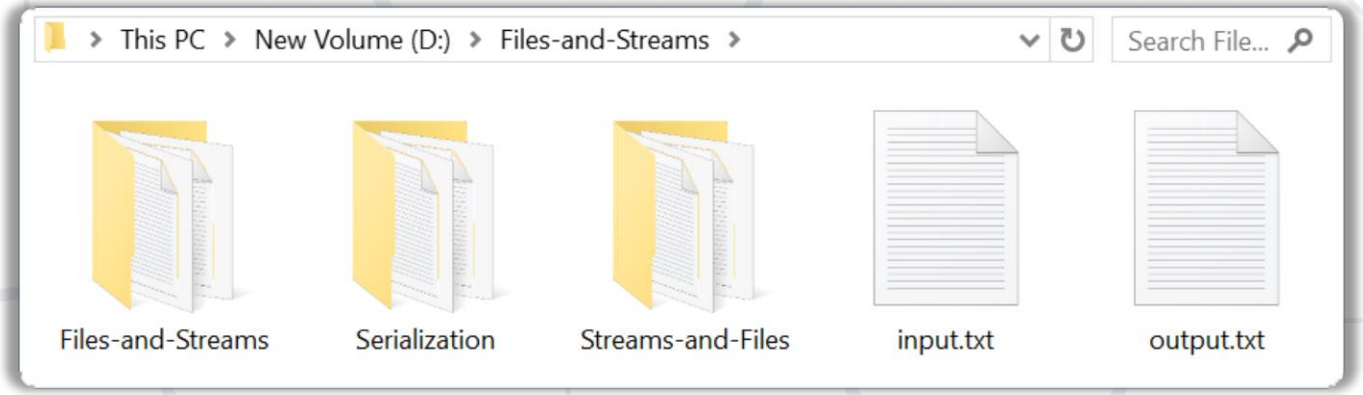
# File Class in Java

- Provides methods for quick and easy manipulation of files

```
import java.io.File;

File file = new File("D:\\input.txt");

boolean isExisting = file.exists();
long length = file.length();
boolean isDirectory = file.isDirectory();
File[] files = file.listFiles();
```

# Problem: List Files

- **Print names and sizes** of all files in "Files-and-Streams" directory
- Skip **child** directories

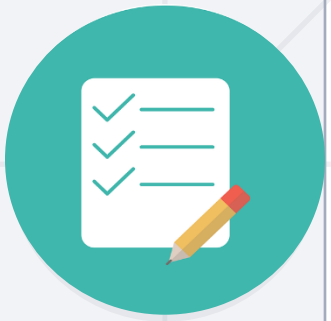



```
input.txt: [size in bytes]
output.txt: [size in bytes]
```

Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

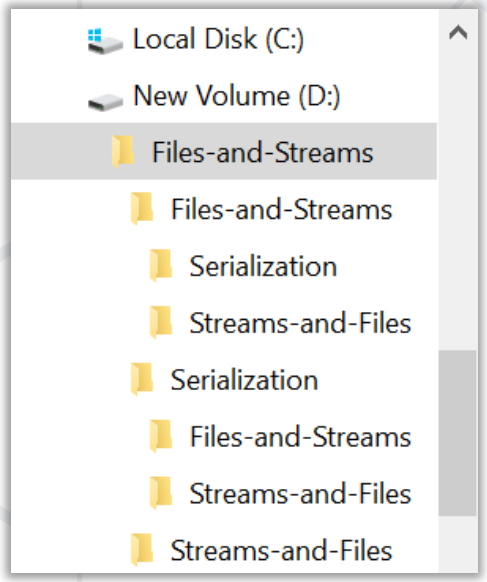
# Solution: List Files

```
if (file.exists()) {
   if (file.isDirectory()) {
      File[] files = file.listFiles();
      for (File f : files) {
         if (!f.isDirectory()) {
            System.out.printf("%s: [%s]%n",
                              f.getName(), f.length());
         }
      }
   }
}
```



# Problem: Nested Folders

- You are **given** a folder named "Files-and-Streams"
- **List all folder names**, starting with the root
- **Print folder count** on the last line (including the root)

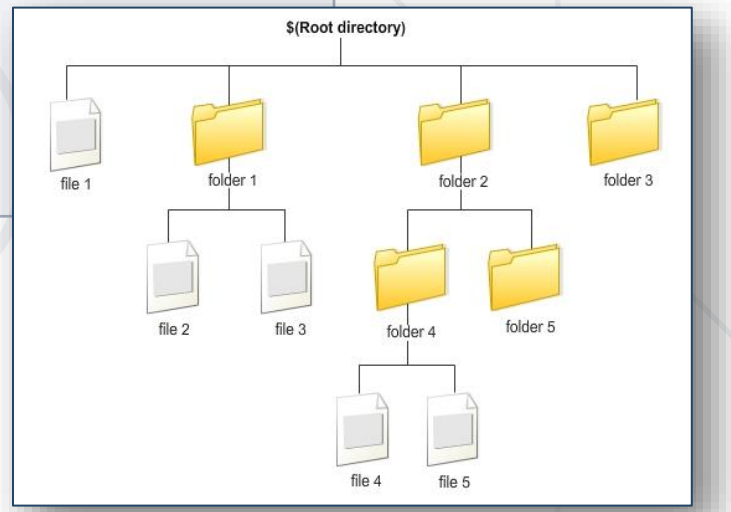



```
…
Streams-and-Files
Serialization
Streams-and-Files
[count] folders
```

Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Solution: Nested Folders (1)

```
String path = "D:\\Files-and-Streams";
File root = new File(path);

Deque<File> dirs = new ArrayDeque<>();
dirs.offer(root);

// continue…
```

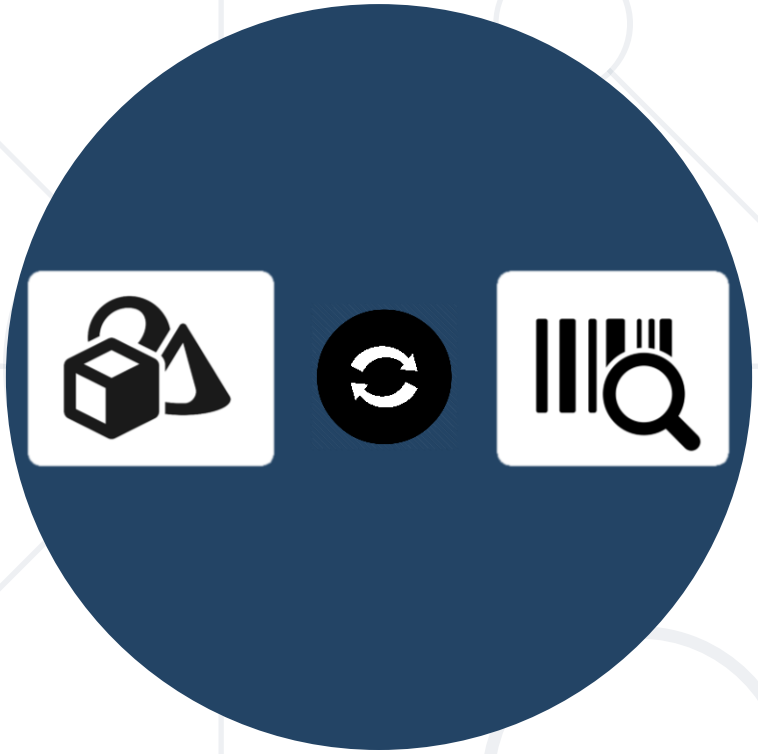



Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Solution: Nested Folders (2)

```
int count = 0;
while (!dirs.isEmpty()) {
  File current = dirs.poll();
  File[] nestedFiles = current.listFiles();
  for (File nestedFile : nestedFiles)
    if (nestedFile.isDirectory())
      dirs.offer(nestedFile);
  count++;
  System.out.println(current.getName());
}
System.out.println(count + " folders");
```

Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab



# Serialization

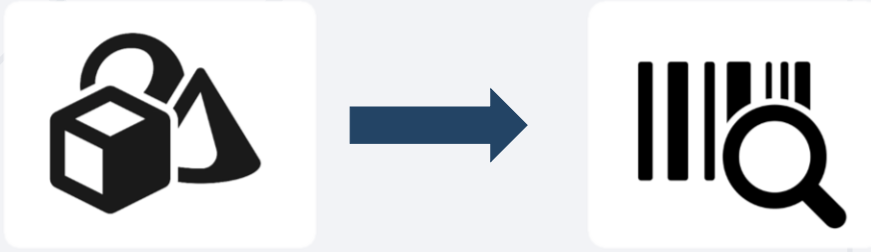
Serializing and Deserializing Objects

# Serialization

- **Save** objects to a file

```
List<String> names = new ArrayList<>();
Collections.addAll(names, "Mimi", "Gosho");

FileOutputStream fos = new FileOutputStream(path);
ObjectOutputStream oos =
          new ObjectOutputStream(fos);

oos.writeObject(names);

// TODO: handle exceptions
```

Save objects to .ser file



# Deserialization

- **Load** objects from a file

```
FileInputStream fis =
     new FileInputStream(path);
ObjectInputStream oos =
     new ObjectInputStream(fis);

List<String> names =
     (List<String>) oos.readObject();

// TODO: handle exceptions
```

# Serialization of Custom Objects

- Custom objects should **implement** the **Serializable** interface

```
class Cube implements Serializable {
  String color;
  double width;
  double height;
  double depth;
}
```

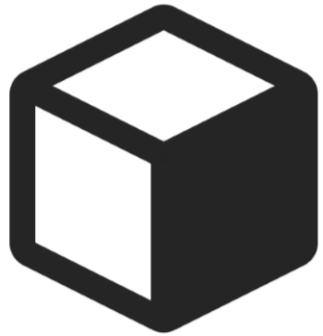
# Problem: Serialize Custom Object

- Create a **Cube class** with color, width, height and depth
- Create a cube – **color: "green", w: 15.3, h: 12.4** and **d: 3**

Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Solution: Serialize Custom Object (1)

```
class Cube implements Serializable {
   String color;
   double width;
   double height;
   double depth;
}
```



Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Solution: Serialize Custom Object (2)

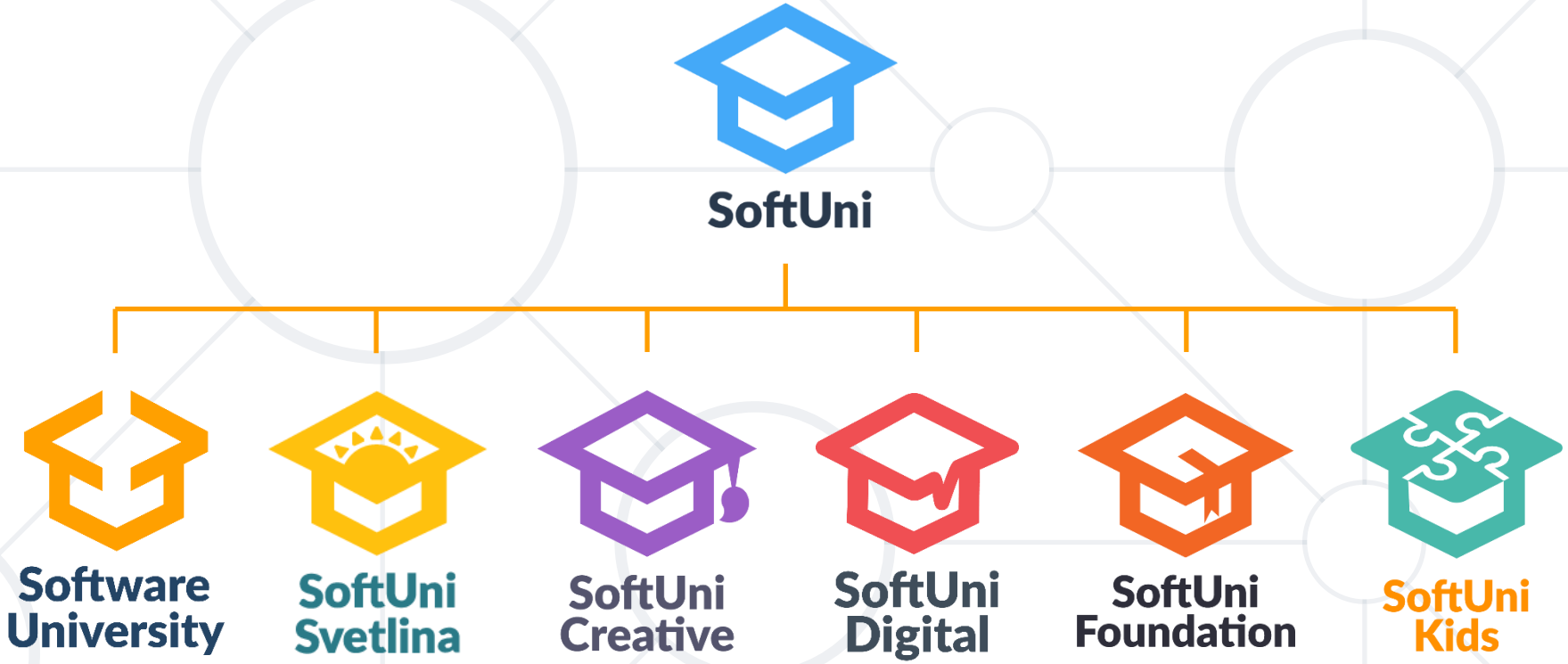
```
//TODO: Create Cube object

String path = "D:\\save.ser";
try (ObjectOutputStream oos = new ObjectOutputStream(
                    new FileOutputStream(path))) {
    oos.writeObject(cube);
} catch (IOException e) {
    e.printStackTrace();
}
```

Check your solution here: https://judge.softuni.bg/Contests/1493/Streams-Files-And-Directories-Lab

# Summary

- **Streams** are used to **transfer data**
- **Two** main **types** of streams
  - **Input** Streams
  - **Output** Streams
- **Buffered** streams boost **performance**
- Streams **can be chained together**
- You can **save objects** state into a file

# Questions?



# SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

SmartIT

XS SOFTWARE

zühlke
empowering ideas

INFRAGISTICS

Coca-Cola HBC Bulgaria

DRAFT KINGS

Postbank
Решения за твоето утре

SOFTWARE GROUP

# Educational Partners

Software University



# Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg


Software University

FORUM

# License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg