Object-Oriented Programming in Java

Lecture 9 - Input and Output

Emily Lucia Antosch

HAW Hamburg

15.08.2025

Inhaltsverzeichnis

1. Introduction

1.1 Where are we now?

- In the last lecture, we dealt with handling exceptions
- You can now
  - throw and catch exceptions,
  - handle exceptions with `try` and `catch`
  - and define your own exception types.
- Today we continue with **Input and Output**.

1.1 Where are we now?

1. Imperative Concepts
2. Classes and Objects
3. Class Libraries
4. Inheritance
5. Interfaces
6. Graphical User Interfaces
7. Exception Handling
8. **Input and Output**
9. Multithreading (Parallel Computing)

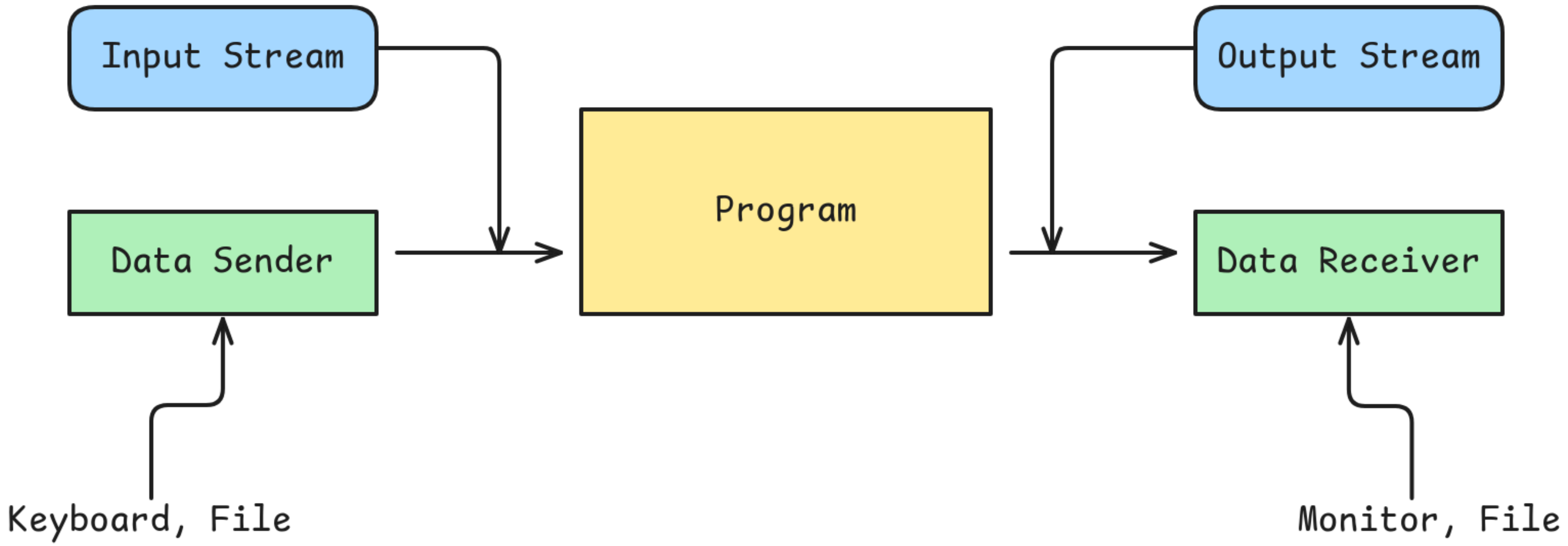
1.2 The goal of this chapter

- You read characters, strings and numeric values from the keyboard.
- You chain and use input and output streams contained in the Java SDK for input and output of bytes, characters and text lines.
- You read and write strings from and to text files.

2. Stream concept & Screen output

2.1 Stream concept

- Stream: Transports data from sender („source“) to receiver („sink“)
- Input: Reading data into a program
- Output: Data leaves a program
- Class library contains about 50 classes for all important input and output variants



2.1 Stream concept

- With what we have already learned:
  - What are the components of `System.out.println()`?
- Only this makes sense:
  - `System`: Class (since no variable System is declared)
  - `out`: Class variable of System, references an object
  - `println()`: Method of the object referenced via `out`
- Output stream:
  - `System.out` references object of class `PrintStream`
  - Object is connected to the screen

2.1 Stream concept

- Selected methods of the PrintStream class:

Methods	**Meaning**
`println(String message)`	Output with line break(`print line`)
`print(String message)`	Output without line break
`printf(String format, Object... arg)`	Formatted Output (see `String.format()`)
`format(String format, Object... arg)`	Formatted Output (see `String.format()`)

Tabelle 1: Formats and Flags

2.1 Stream concept

Frage

- What is output?

Java

```java
	public static void main(String[] args) {
	double tempHawaiiCelsius = 15.97;
	double tempHamburgCelsius = 22.71;
	String.format("Hawaii: %.1f °C", tempHawaiiCelsius);
System.out.printf("Hamburg:%.1f °C", tempHamburgCelsius);
	}
```

2.1 Stream concept

Streams referenced in `System`:

Reference	**Data Type**	**Meaning**
`System.out`	`PrintStream`	Output on screen
`System.err`	`PrintStream`	Error output on screen
`System.in`	`InputStream`	Input from keyboard

Tabelle 2: Formats and Flags

3. Keyboard input

3.1 Class Scanner

- Provides methods for reading texts and simple data types (e.g. int)
- Text input is analyzed and interpreted („parsing“, e.g. converting to integer)
- Creation and termination:
  - Scanner object is connected to input stream in constructor
  - The connection should be terminated via the Scanner method close().

3.1 Class Scanner

Beispiel

Java

```java
public class ScannerLine {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Bitte einen Satz eingeben: ");
        System.out.println(scanner.nextLine());
        scanner.close();
    }
}
```

3.1 Class Scanner

Frage

- Oops, what happens here?

```java
public class ScannerToken {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Please enter a sentence: ");
        System.out.println(scanner.next());
        scanner.close();
    }
}
```

- Method `next()`: Only first word instead of entire sentence is read and output
- Words and lines are distinguished.

3.1 Class `Scanner`

- Separators of multiple inputs:
  - Token: Individual words or values (e.g. integer)
  - Tokens in input separated by separators
  - Default separator is a whitespace (i.e. space, tab, line break)
- Methods:
  - Separator changeable via method useDelimiter()
  - Via method hasNext() query whether tokens are still available

Aufgabe 1

- Write a program that reads a sentence via next().

3.1 Class Scanner

Java

```java
public class ScannerNext {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Bitte einen Satz eingeben: ");
        while (scanner.hasNext()) {
            System.out.println(scanner.next());
        }
        scanner.close();
    }
}
```

3.1 Class Scanner

? Frage

- What happens if you replace scanner.hasNext() with true?
- How does next() behave once all words have been read?

3.1 Class Scanner

- Spezielle Methoden für einfache Datentypen:
  - Einlesen: `nextBoolean()`, `nextInt()`, `nextDouble()`, …
  - Abfrage: `hasNextBoolean()`, `hasNextInt()`, `hasNextDouble()`, …

? Frage

– Welche Ausgaben werden für die Eingaben „127“, „128“ und „Hamburg“ erzeugt?

Java
```java
public class ScannerByte1 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Please enter a byte value: ");
        System.out.println("Entered: " + scanner.nextByte());
        scanner.close();
    }
}
```

3.1 Class `Scanner`

- Parsing errors:
  - Inputs „128“ and „Hamburg“: Exception of type `InputMismatchException`
  - Has base class `RuntimeException` (exception handling not mandatory)

Aufgabe 2

- The program should not be terminated by an exception:
  - Find two different ways to avoid this.
  - Implement these approaches.

- Approaches:
  - Catch the exception
  - Query via `hasNextByte()`

3.1 Class Scanner

- Catch exception:

Java

```java
public class ScannerByte2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Please enter a byte value: ");
        try {
            System.out.println("Entered: " + scanner.nextByte());
        } catch (InputMismatchException e) {
            System.out.println("Input is not a byte value.");
        } finally {
            scanner.close();
        }
    }
}
```

3.1 Class Scanner

- Query data type:

Java

```java
public class ScannerByte3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Please enter a byte value: ");
        if (scanner.hasNextByte()) {
            System.out.println("Entered: " + scanner.nextByte());
        } else {
            System.out.println("Not a byte value: " + scanner.next());
        }
        scanner.close();
    }
}
```

3.1 Class `Scanner`

Aufgabe 3

- Read the components of a vector (data type `int`)
- Read components until another token (e.g. a letter) is entered
- Output the vector as well as the magnitude

Beispiel

Integer components (terminate with different character): 7 4 0 15 End

$a = [7, 4, 0, 15]^T$

$\|a\| = 17,03$

3.1 Class Scanner

Java

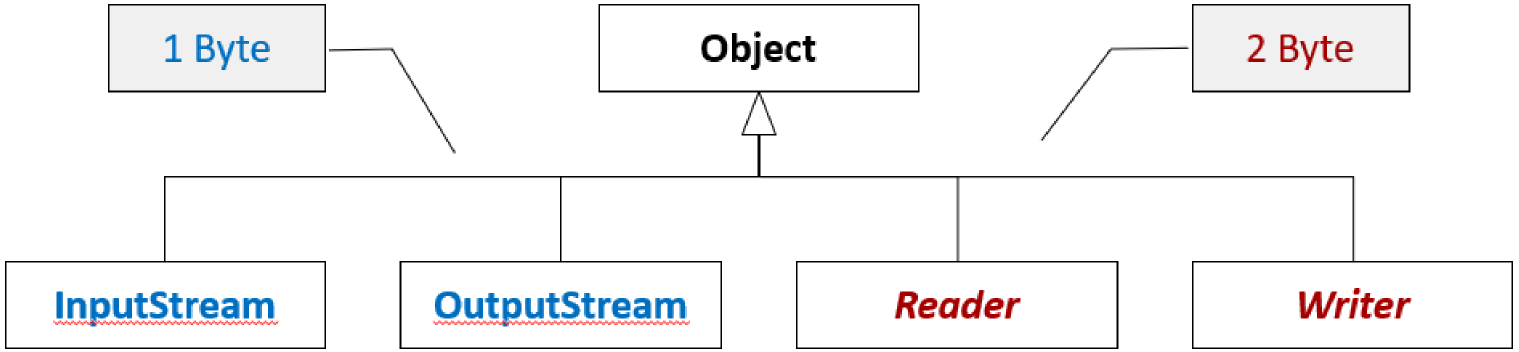
```java
public class ScannerVektor {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ArrayList<Integer> vector = new ArrayList<Integer>();
        System.out.print("Integer components (terminate with different character): ");
        while (scanner.hasNextInt())
            vector.add(scanner.nextInt());
        scanner.close();
        if (vector.size() > 0) {
            System.out.print("a = [" + vector.get(0));
            long sumOfSquares = vector.get(0) * vector.get(0);

            for (int i = 1; i < vector.size(); i++) {
                System.out.print(", " + vector.get(i));
                sumOfSquares += vector.get(i) * vector.get(i);
            }
            System.out.println("]^T");
            System.out.printf("||a|| = %.2f\n", Math.sqrt(sumOfSquares));
        }
    }
}
```

4. Byte & Character Streams

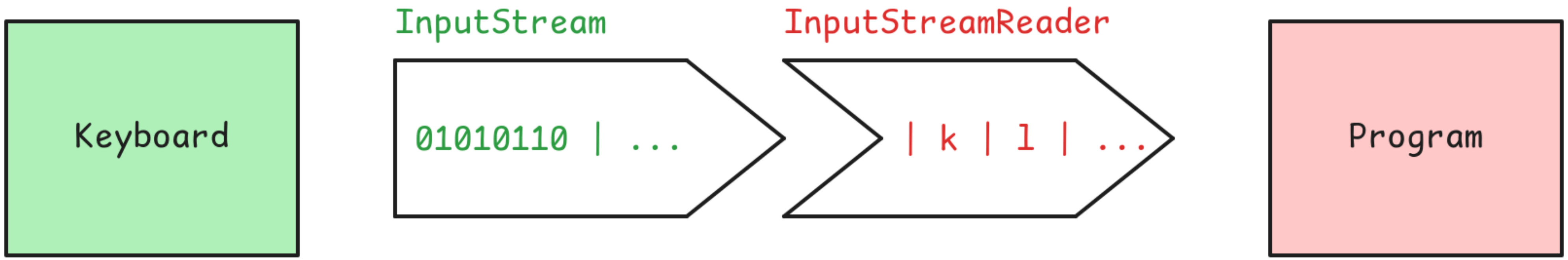
4.1 Byte & Character Streams

- What was the special feature of characters in Java again?
  - All characters encoded as 2 bytes (Unicode)
  - Distinguish: Streams that transport elements of 1 byte or 2 bytes („characters“)
- Byte streams (byte-oriented streams):
  - Transport individual bytes
  - Classes InputStream and OutputStream as well as classes derived from them
- Character streams (character-oriented streams):
  - Transport characters of 2 bytes each
  - Abstract classes Reader and Writer as well as classes derived from them



4.1 Byte & Character Streams

- Keyboard delivers stream of individual bytes (e.g. `System.in` of data type `InputStream`)
  - Java characters consist of 2 bytes
  - Connect byte stream with character stream
- Notes:
  - Goal in the following: Illustration of stream chaining
  - Yes, keyboard inputs (code $l = 255$) you wouldn't need to chain with a character stream.
  - Yes, feel free to use Scanner for keyboard inputs.



4.1 Byte & Character Streams

Java

```java
public class KeyboardReader1 {
    public static void main(String[] args) throws IOException {
        InputStreamReader reader = new InputStreamReader(System.in);

        System.out.print("Please enter a character: ");
        System.out.println(reader.read());
        System.out.println(reader.read());
        System.out.println(reader.read());
        reader.close();
    }
}
```

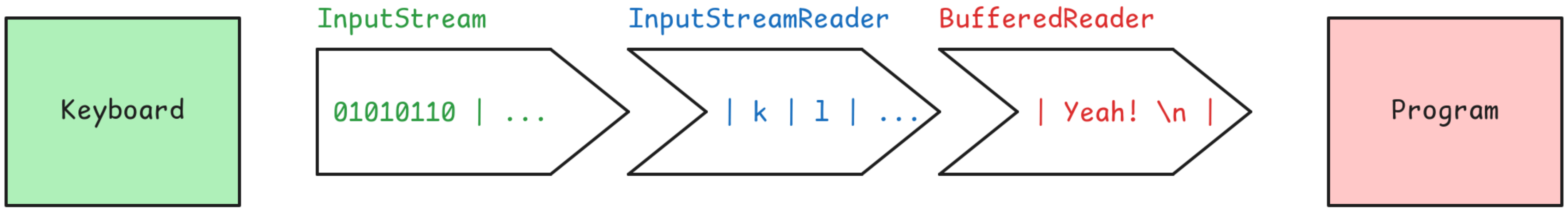
4.1 Byte & Character Streams

? Frage

- Why is `read()` called three times?
- Why are the second and third outputs always 13 and 10?

4.1 Byte & Character Streams

- `BufferedReader` reads a character stream and buffers the characters
- Provides e.g. method `readLine()` for reading out a line
- Analogously, class `BufferedWriter` outputs entire line via `newLine()`



Aufgabe 4

- Modify the previous example as follows:
  - Read two lines via
  - BufferedReader Then output both lines

4.1 Byte & Character Streams

Java

```java
public class KeyboardReader2 {
    public static void main(String[] args) throws IOException {
        InputStreamReader reader = new InputStreamReader(System.in);
        BufferedReader bufferedReader = new BufferedReader(reader);

        System.out.print("Please enter first line:  ");
        String line1 = bufferedReader.readLine();
        System.out.print("Please enter second line: ");
        String line2 = bufferedReader.readLine();

        System.out.println(line1);
        System.out.println(line2);
        reader.close();
    }
}
```

5. Files

5.1 Files and Directories

- Class `File` represents file or directory
  - Objects contain information about file, not its content
  - IntelliJ uses the project directory as root directory for reading/writing.

```java
public class CreateFile {
    public static void main(String[] args) throws IOException {
        File file = new File("Testdatei.txt");
        boolean isExists = file.exists();

        if (!isExists) {
            System.out.println("Create file");
            isExists = file.createNewFile();
        }

        if (isExists && file.isFile()) {
            System.out.println("Read: " + file.canRead());
            System.out.println("Write: " + file.canWrite());
            file.delete();
        }
    }
}
```

5.1 Files and Directories

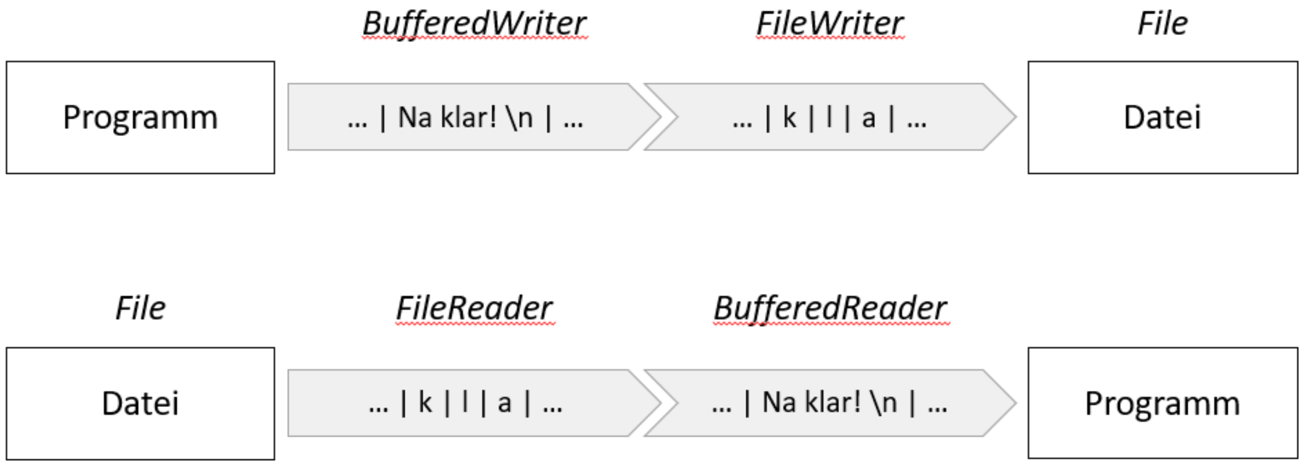
Java

```java
public class ListDirectory {
    public static void main(String[] args) {
        File directory = new File(".");

        if (directory.isDirectory()) {
            String[] children = directory.list();
            for (String child : children) {
                System.out.println(child);
            }
        }
    }
}
```

5.1 Files and Directories

- Byte streams:
  - Read files via FileInputStream classes and write via FileOutputStream
- Character streams (e.g. text files):
  - Read files via FileReader and write via FileWriter
  - Buffered character streams via BufferedReader and BufferedWriter



5.1 Files and Directories

- Let's apply this:
  - Create a program that writes a text file.
  - Create another program that reads the content of the text file and outputs it.

Java

```java
public class WriteFile {
        public static void main(String[] args) throws IOException {
                File file = new File("Testdatei.txt");
                FileWriter writer = new FileWriter(file);
                BufferedWriter bufferedWriter = new BufferedWriter(writer);

                bufferedWriter.write("This is the first line.");
                bufferedWriter.newLine();
                bufferedWriter.write("And here comes the second line.");
                bufferedWriter.newLine();
                bufferedWriter.close();
        }
}
```

5.1 Files and Directories

Java

```java
public class ReadFile {
    public static void main(String[] args) throws IOException {
        File file = new File("Testdatei.txt");
        FileReader reader = new FileReader(file);
        BufferedReader bufferedReader = new BufferedReader(reader);

        while (bufferedReader.ready()) {
            System.out.println(bufferedReader.readLine());
        }
        bufferedReader.close();
    }
}
```

6. License Notice

6.1 Attribution