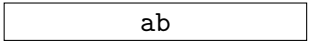
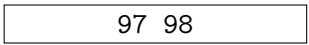
Recap: IO and Exceptions

- Input and output
  - Streams: Pipes through which data flows
    - When something has consumed, it's no longer there
    - Need to be flushed and closed at the end
  - InputStream/OutputStream: byte-wise operations
  - Readers/Writers: Used on top of streams to operate on characters
- Things can go wrong, even if our program works well
  - Many error sources with I/O: Files, disks, networks can fail
  - Exception handling
    - Mechanism to handle unexpected errors
    - `try {} catch (EX) {}`
  - Exceptions are objects of class `java.lang.Exception`

a file “file.txt”



a file “file.txt”

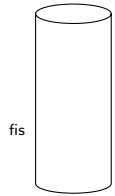


a file “file.txt”

0110000101100010

a file “file.txt”

0110000101100010



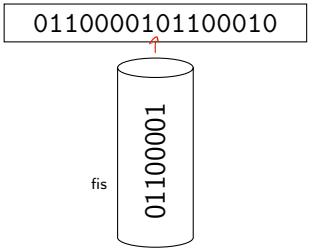

```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```

01100001

64 32 16 8 4 2 1

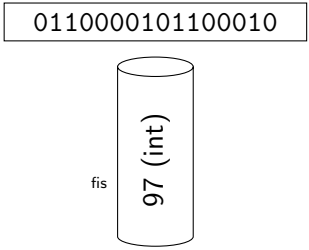
64 + 32 + 1 = 97

a file “file.txt”




```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```

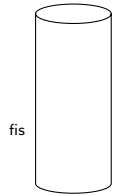
a file “file.txt”




```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```

a file “file.txt”

0110000101100010

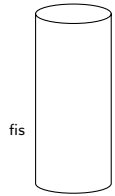


i 97

```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```

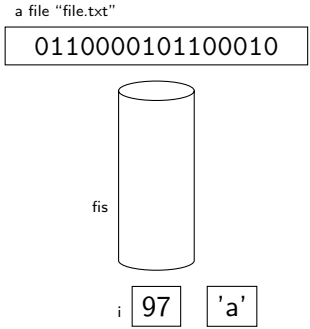
a file “file.txt”

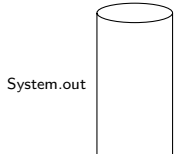
0110000101100010



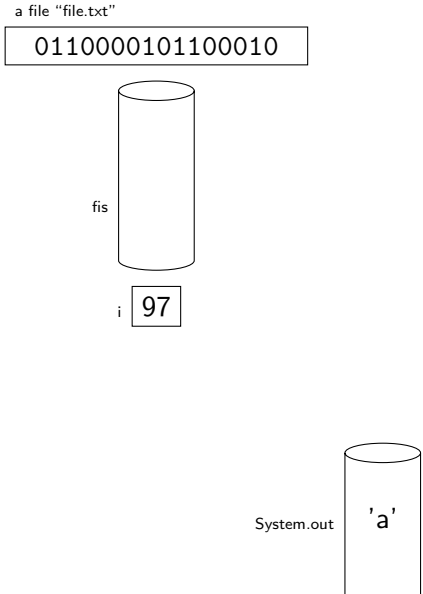
i 97

```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```

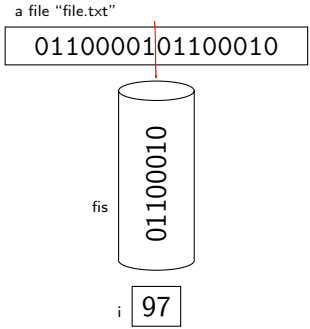


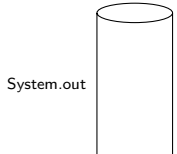



```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```

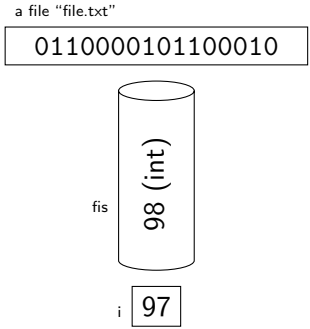


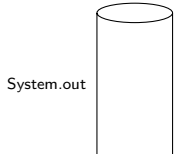

```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```



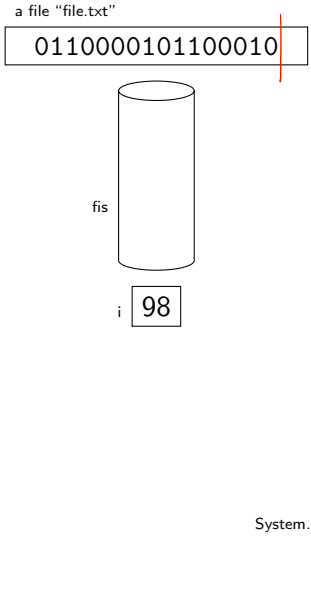



```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```

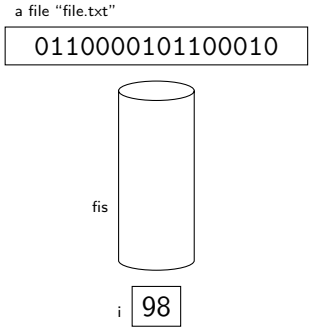


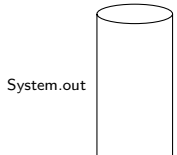



```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```

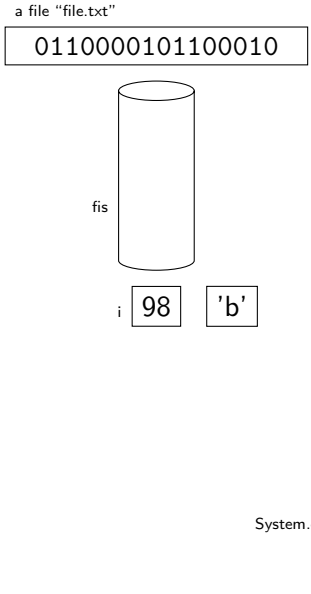



```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```

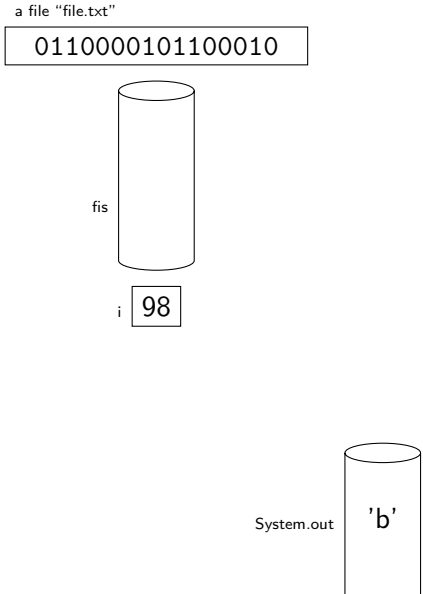




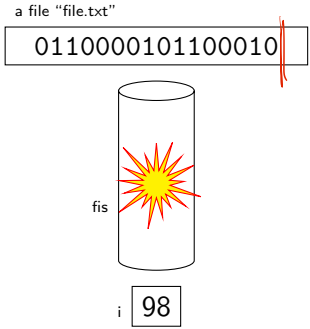

```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```

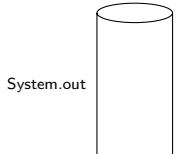



```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```

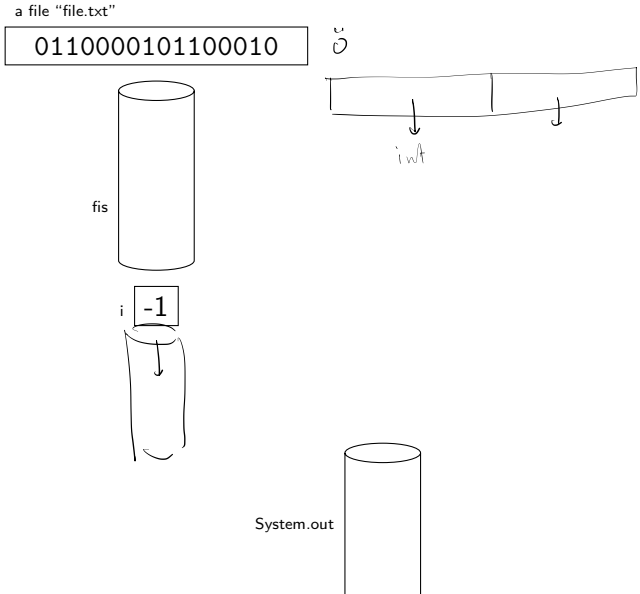



```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```

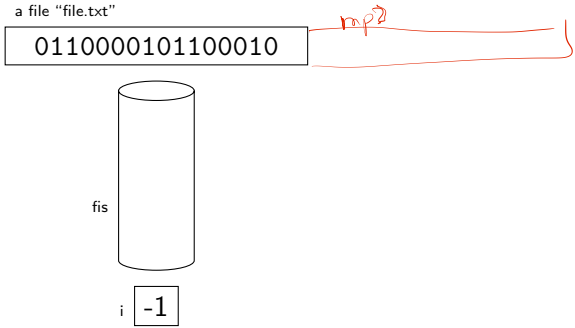





```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```




```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```




```
1  FileInputStream fis =
     new FileInputStream("file.txt");
2  int i = fis.read();
3  while (i >= 0) {
4    System.out.println( (char) i );
5    i = fis.read();
6  }
```

UNIVERSITÄT
ZU KÖLN

User Input, Java Standard Library, Code Style, Closing Remarks

Softwaretechnologie: Java I

Nils Reiter
nils.reiter@uni-koeln.de

January 22, 2025

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

User Input

Introduction

- Last week: “Using `System.out.println()` uses a stream”
- Now: How does this work, exactly?

Introduction

- Last week: "Using `System.out.println()` uses a stream"
- Now: How does this work, exactly?
- Two directions
  - Program output written to console
  - Program input read from console

`System.out.println`

- `System`: A class with many static methods / fields java.lang.System
  - I.e., we can just use them: `System.exit()` calls static method exit in class System

`System.out.println`

- `System`: A class with many static methods / fields java.lang.System
  - I.e., we can just use them: `System.exit()` calls static method exit in class System
- Three stream-related fields:
  - `System.out` – a PrintStream
  - `System.err` – a PrintStream
  - `System.in` – an InputStream

PrintStream

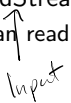
- java.io.PrintStream
- Inherits from `java.io.FilterOutputStream`, which inherits from `java.io.OutputStream`
  - I.e., `System.out` is an output stream, and we can call all OutputStream methods (e.g., `write(int byte)`)
- Class documentation:
  - Ability to print representations of various data values conveniently
  - PrintStream never throws an IOException; instead
  - PrintStream can be created so as to flush automatically

PrintStream

- java.io.PrintStream
- Inherits from `java.io.FilterOutputStream`, which inherits from `java.io.OutputStream`
  - I.e., `System.out` is an output stream, and we can call all OutputStream methods (e.g., `write(int byte)`)
- Class documentation:
  - Ability to print representations of various data values conveniently
  - PrintStream never throws an IOException; instead
  - PrintStream can be created so as to flush automatically
- System.out and System.err
  - System.out used for regular output (e.g., the answer that the program produces)
  - System.err intended for error messages (e.g., exception stuff)

`System.in`

- Used to read input from console
- Not very convenient with the bare input stream
- Two options:
  - InputStreamReader
    - Reads character-wise
    - Beware: `\n` is a single character
  - BufferedStreamReader (wrapped around an InputStreamReader)
    - Can read line-wise (which is usually what we want)



demo

Zoo/Exercise 13

Section 2

Java Standard Library

Introduction

- Programming language core: Rather small
- A few types, some statements, some syntactic elements

Introduction

- Programming language core: Rather small
- A few types, some statements, some syntactic elements
- Libraries
  - Collections of code, useful for all kinds of things
  - Many languages have such libraries
  - To avoid reinventing the wheel, we should use them

Java Standard Library

Interesting packages

- `java.io` – Input and output
- `java.lang` – Core functions
- `java.math` – Mathematical functions
- `java.net` – Handling networks and connections
- `java.text` – Simple text processing
- `java.util` – Various utility functions, in particular collections
  - Will be discussed in depth in the summer term
- `java.awt`, `javax.swing` – Classes for graphical user interfaces

Section 3

Code Style

Introduction

- Interaction between programmers is easier, if they adhere to common style
- Style: How to write and format variables, methods, classes etc.
- Java Code Style
  - No strict rules, but guidelines
- Offical document from 1997:
  `https://www.oracle.com/technetwork/java/codeconventions-150003.pdf`
- In Eclipse, you can select the code and use Source > Format to automatically format the code nicely

Java Code Style

- `CamelCase` is used for combining words (instead of underscore or dot)
- Class and interface names start with an upper case letter (`MyArray`) and are nouns
- Methods names start with a lower case letter (`get()`) and are verb phrases
- Variables start with a lower-case letter and are as long as it needs to be for clarity
  - Variable names like `a` are dispreferred
- Indentation should be used to make the structure of the program visible
  - Substatements of a statement or declaration should be indented
  - Indentation should be four spaces wide
- Avoid lines longer than 80 characters
- Files longer than 2000 lines are cumbersome and should be avoided.
- ...

Section 4

Closing Remarks

Learning Programming

- Learning to program is hard and takes time
- It helps to
  - Regularly do it
  - Talk about it
  - Be stubborn
  - Think formalistic
  - Be fearless and disrespectful
  - Read documentation
  - Try to understand your mistakes
- It's ok to make mistakes

On Programming in Real Life

- It's extremely rare to start from scratch
- Most of the time, we work with code that others have written
  - $60\,\%$ to $90\,\%$ of the lifetime cost of software goes to maintenance
- Software we start will likely be continued by others

Sources

On Programming in Real Life

- It's extremely rare to start from scratch
- Most of the time, we work with code that others have written
  - $60\,\%$ to $90\,\%$ of the lifetime cost of software goes to maintenance
- Software we start will likely be continued by others

→ Writing "good code" is not needed technically, but because it makes maintenance easier

Sources

On Programming in Real Life

- It's extremely rare to start from scratch
- Most of the time, we work with code that others have written
  - $60\,\%$ to $90\,\%$ of the lifetime cost of software goes to maintenance Sources
- Software we start will likely be continued by others
- ➔ Writing "good code" is not needed technically, but because it makes maintenance easier

Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?

Kernighan/Plauger (1978, 10)

Looking Ahead

What happens in the summer term

- Version control (= git)
- Recursion
- Data structures
- Unit testing
- Efficient programming
- Multithreading
- ...

Looking Ahead

What happens in the summer term

- Version control (= git)
- Recursion
- Data structures
- Unit testing
- Efficient programming
- Multithreading
- ...

Programming Ideas for the Break

- A simple game such as Tic Tac Toe
  - Turn-based games are simpler than real time games
- Birthday predictor (read in a list of birthdays, calculate the next round anniversaries)
- Make algorithmic art (e.g., ASCII art)