Introduction to Programming

L01 Introduction

Stephan Krusche

18 October 2023
Technical University of Munich

Welcome to
Introduction to Programming (ITP)

Our teaching concept: hybrid and blended learning

- **Synchronous** 🔄
  - Lecture with exercises (livestream)
  - Tutor groups with presentations
- **Asynchronous** 🕒
  - Lecture recordings, lecture slides, readings
  - Homework for self-study
- **Great support** 🤗 2 exercise instructors and 47 tutors for ~600 students
- Digital learning tools 🛠️💻 and many different activities

→ **Best possible learning experience** 🌟🎓 for everyone

Our expectations

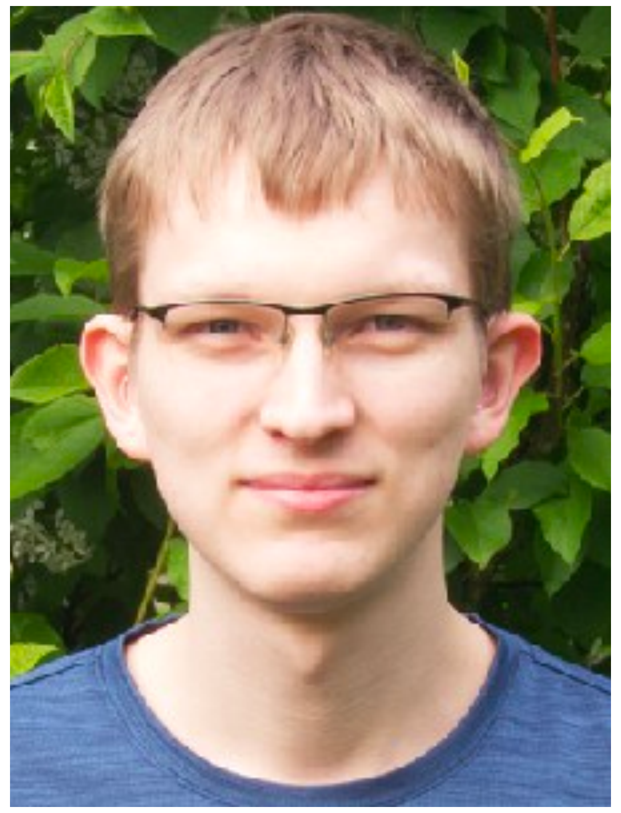
- Be **open** to new technologies, learning methods and teaching approaches
- Be **active** and **curious**, learn to **organize** yourself
- Be **kind** and **patient**
- **Bear with us** if something does not work out as expected

Our teaching goals

- You improve your **problem solving** and **soft skills**
- You understand **programming concepts** and learn **how to program**
- You achieve the **best possible grade** in the course

Course team

Stephan (Krusche)
Lecturer



Timor (Morrien)
Exercise instructor

Patrick (Bassner)
Exercise instructor

Michael (Allgaier)
Exercise instructor

Markus (Paulsen)
Exercise instructor

+ 46 tutors

© 2023 Stephan Krusche

Abdelhadi

Adriano

Aleksandar

Andrea

Aniruddh

Anton

Arman

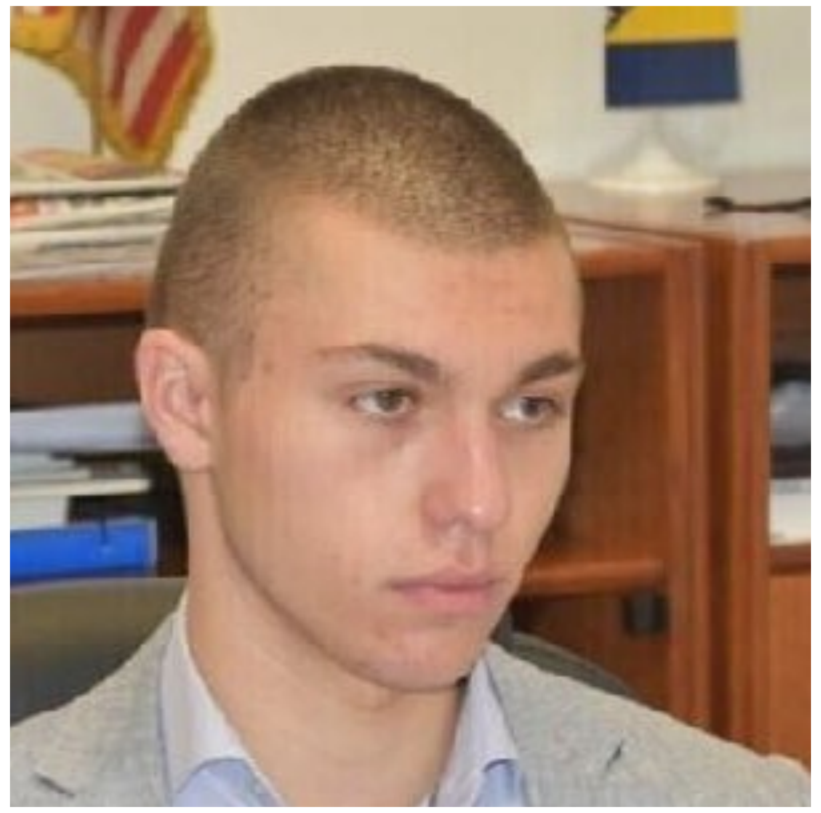
August

Berke

Diego

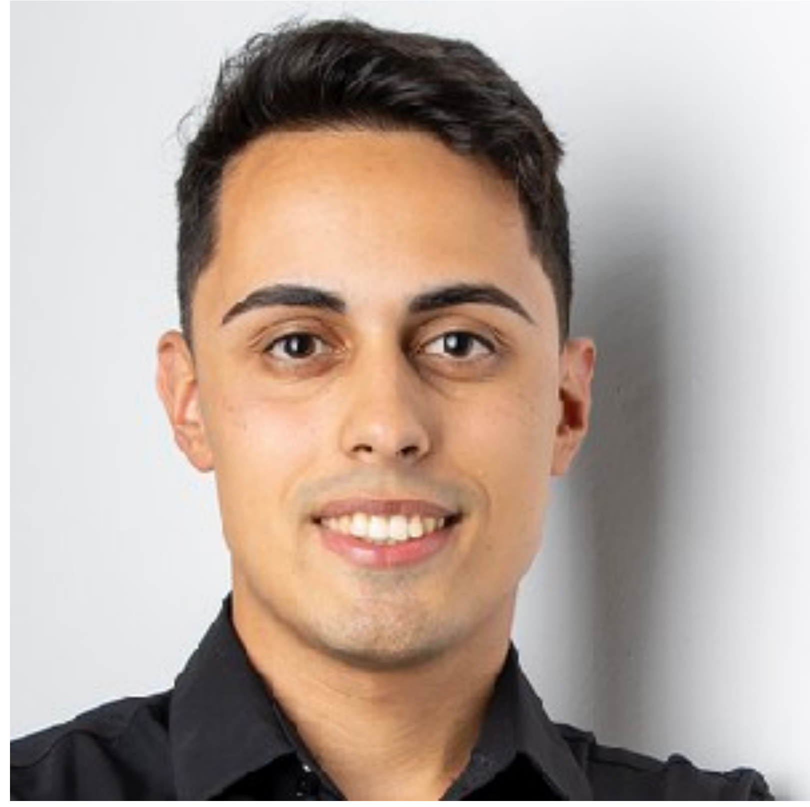
Ege

Egemen

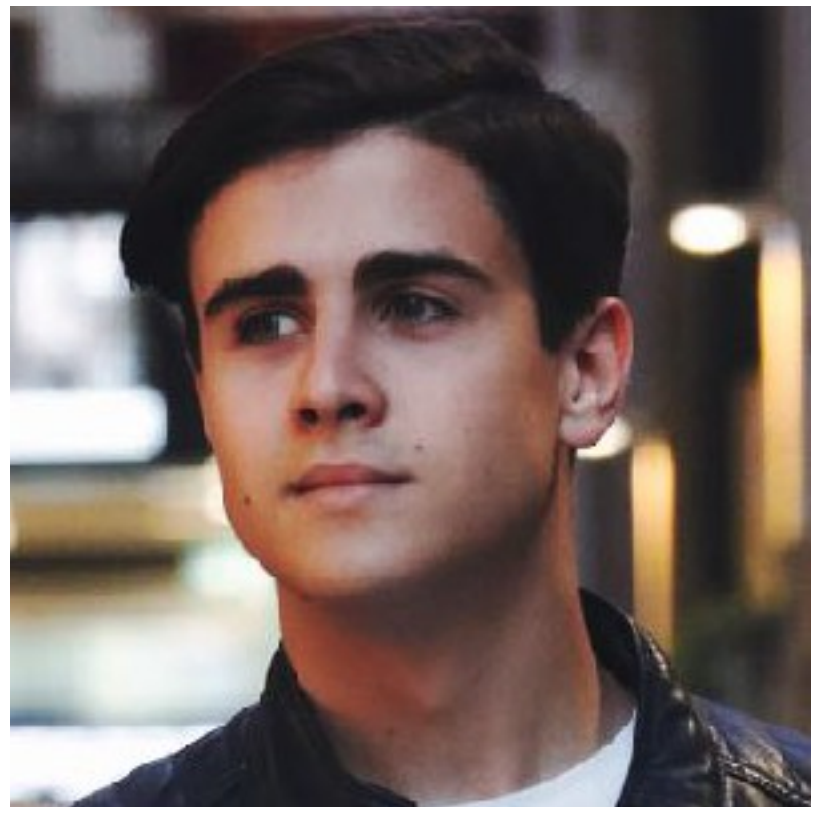


Faris

Feryal



Francisco



Georgy

Hassan

Jana

Jan

Jasmina

Konrad

Lagun

Lena

Liam

Luca

Mariya

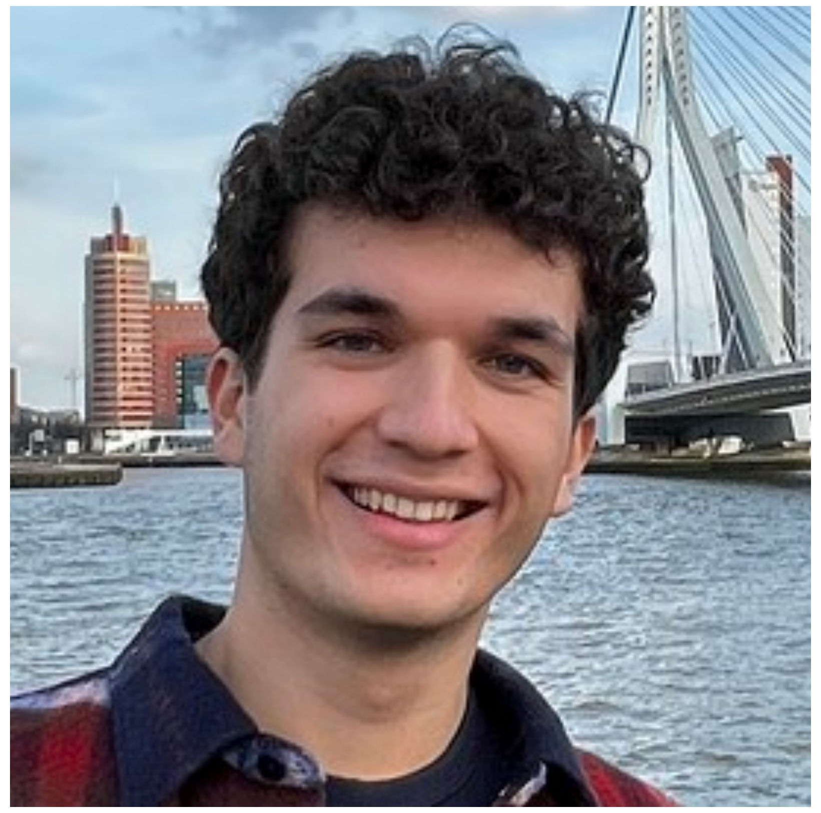
Maximilian

Michael

Napassakorn

Natallia

Nicolas



Ömercan

Philipp

Sarp

Tobias

Ulkar

Vincent

Yulia

Course concept 🌟

- **Lecture** 🗓️
  - Onsite lectures with small in-class exercises (livestream + recordings)
  - 4 x 40min lecture units (with breaks in-between)
  - Focus on interaction and activation
  - Digital learning tools to create the **best possible learning experience**
- **Tutor groups** 👥
  - 3 hour slots (with small breaks) in small onsite tutor groups (~15 students) with intense supervision
  - Group exercises and student presentations
- **Homework** 🛠️
  - Weekly homework exercises which must be solved independently
- **Assessment** 📊
  - Multiple computer-based tests, presentations and a project

Requirements for this course

- **Assumptions**
  - You want to learn how to program
  - You want to learn how to solve problems
- **Prerequisites**
  - You have your own laptop and use it during the lectures and exercises!
- **Beneficial**
  - You have practical experience with programming
  - You have already participated in a software project

Main learning goals

- **Understand** and **apply** the fundamental concepts of computer science at a basic, practical, but scientific level: algorithms, syntax, semantics, efficiency
- **Solve** algorithmic **problems**
- **Program** simple applications in the object-oriented language **Java**
- **Understand** the underlying **concepts** and **models** of programming languages to be able to learn other languages independently
- **Apply problem solving concepts** based on data structures, recursion, objects, classes, methods, lists, queues

Learning goals (detailed)

- **Apply** basic notions to solve problems using algorithms and programs
- **Understand** imperative programming constructs
- **Differentiate** syntax and semantics
- **Understand** the syntax of programming languages based on regular expressions and context-free grammar
- **Understand** the semantics of programs using control-flow graphs
- **Understand** numbers, strings, arrays, insertion sort
- **Understand** and **apply** recursion (binary search, patterns of recursion)
- **Implement** classes, attributes, methods and **instantiate** objects
- **Use** data structures such as lists, stacks, queues
- Object-oriented programming: **apply** inheritance, abstract classes, interfaces, polymorphism
- **Understand** the perspectives for programming in the large (software engineering)

© 2023 Stephan Krusche

Roadmap

- **Context**
  - You have your own laptop and can use it during the lecture!
  - You are motivated to learn programming
- **Learning goals**
  - Understand the organization of lectures and exercises
  - Interact with us using Artemis and Discord
  - Participate in on-site tutor meetings
  - Understand a first small program
  - Create simple classes with attributes and methods
  - Instantiate objects
  - Understand the most important data types

Outline

- **General organization of the course**
- Object oriented programming
- Basic types and their operations
- Methods
- UML class diagrams

Tools

➡ **Discord**: https://discord.gg/3PVut934ca

  - Communicate within tutor groups
  - Discuss with other students
  - Read announcements
  - Ask questions about the general organization
  - Post questions and answers related to specific lectures and exercises

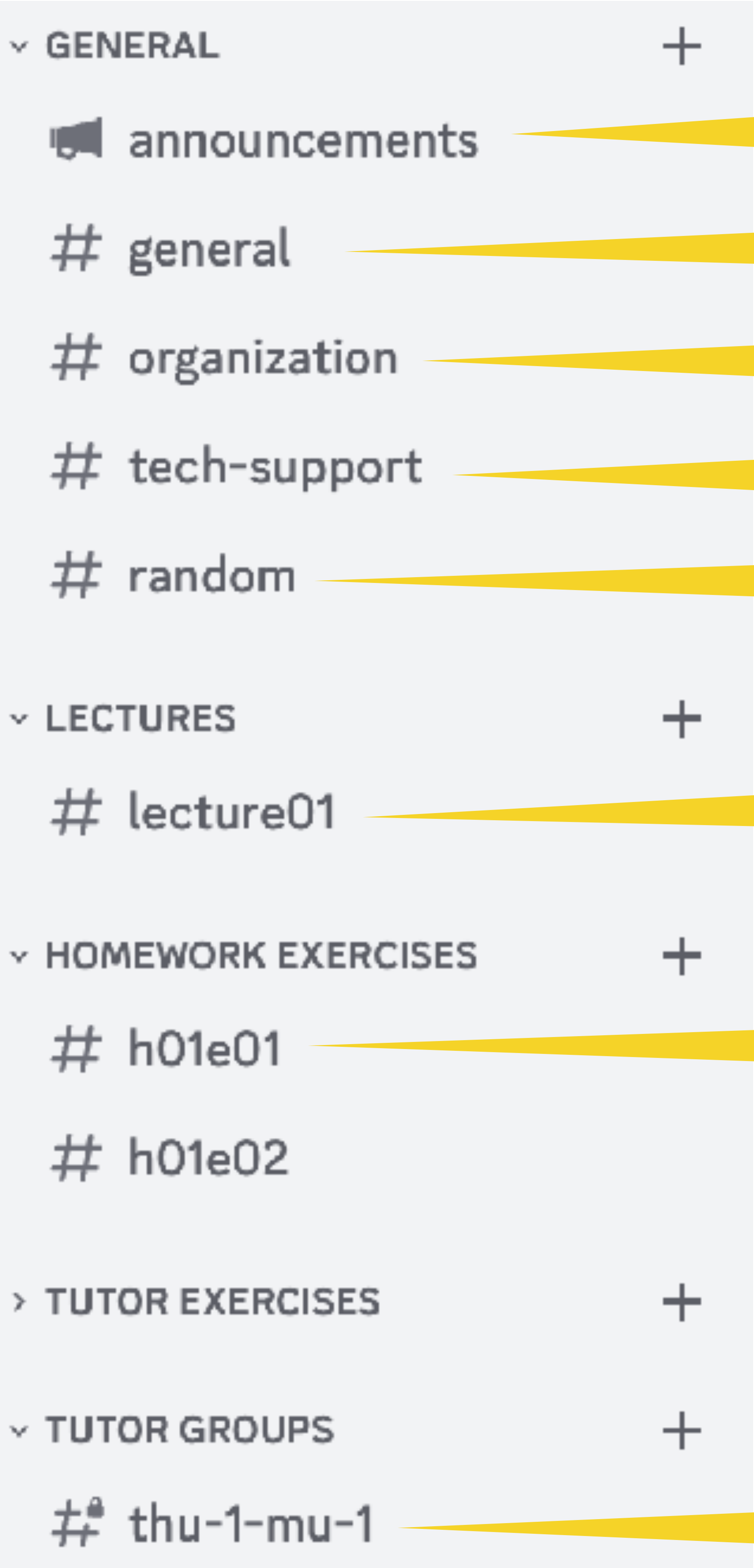
- **Artemis**: https://artemis.cit.tum.de
  - Download PDF slides before the lecture starts
  - Participate in exercises: programming, modeling, quiz, text
  - Review your individual exercise results including feedback
  - Participate in exams

Registration for the Discord workspace

- Please join https://discord.gg/3PVut934ca

Create a new account if needed

- Overview of channels



GENERAL +

announcements

general

organization

tech-support

random

LECTURES +

lecture01

HOMEWORK EXERCISES +

h01e01

h01e02

TUTOR EXERCISES +

TUTOR GROUPS +

thu-1-mu-1

Announcement from instructors

General questions and comments

Questions regarding the organization

Questions regarding tools and technical issues

For your fun

Questions specific to lecture 01

Questions specific to this homework exercise

Questions specific to this tutor groups

→ Use these channels for their **dedicated purposes**

→ The purpose is to **enable communication**, ask questions, and share the answers with everyone

→ Do **not** offend and bully each other

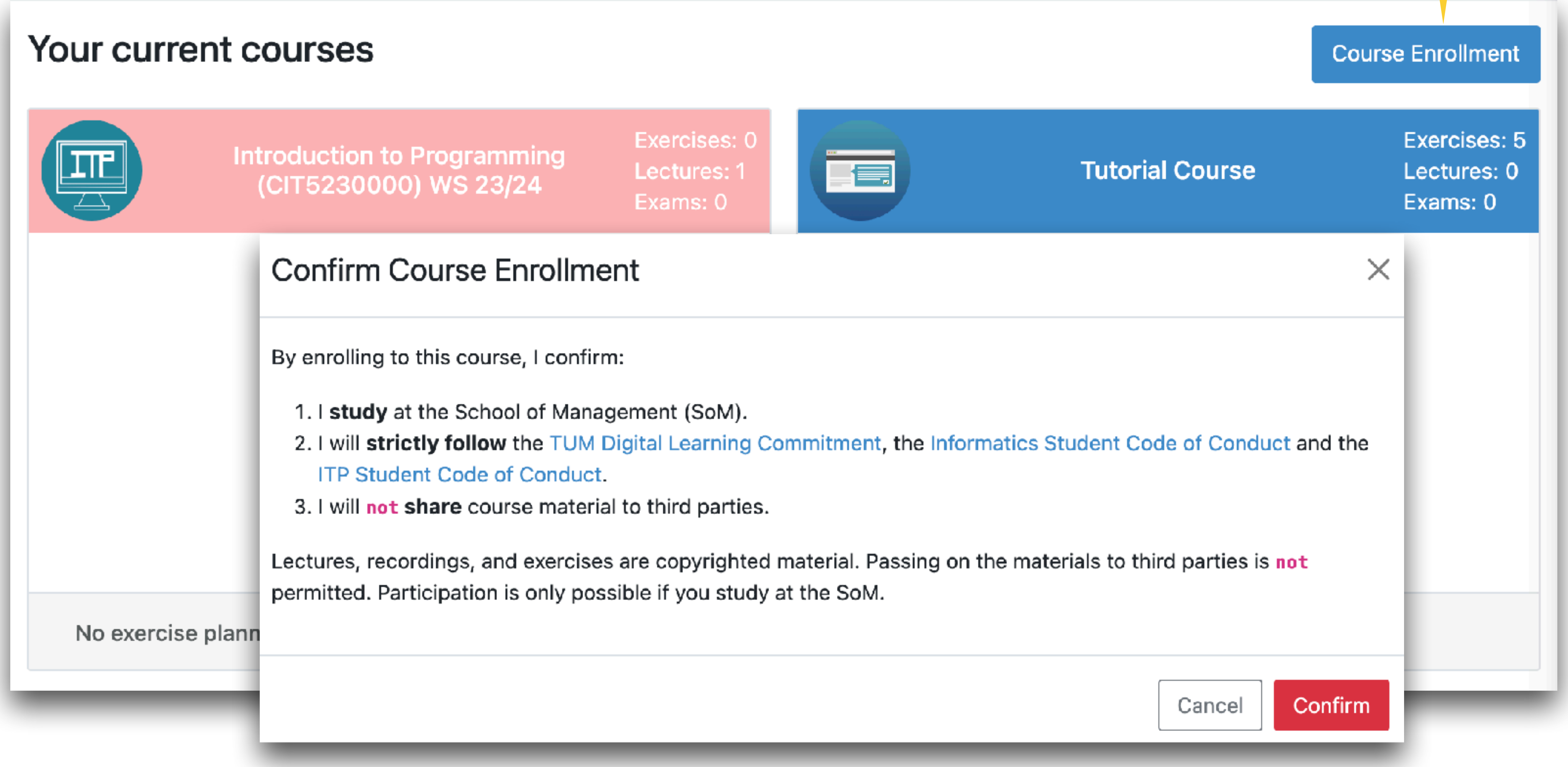
Follow the code of conduct

Tools

- **Discord**: https://discord.gg/3PVut934ca
  - Communicate within tutor groups
  - Discuss with other students
  - Read announcements
  - Ask questions about the general organization
  - Post questions and answers related to specific lectures and exercises
- **Artemis**: https://artemis.cit.tum.de
  - Download PDF slides before the lecture starts
  - Participate in exercises: programming, modeling, quiz, text
  - Review your individual exercise results including feedback
  - Participate in exams

Task 1: Navigate into the course

1. Enroll for the course if you do not see it



Your current courses

Confirm Course Enrollment

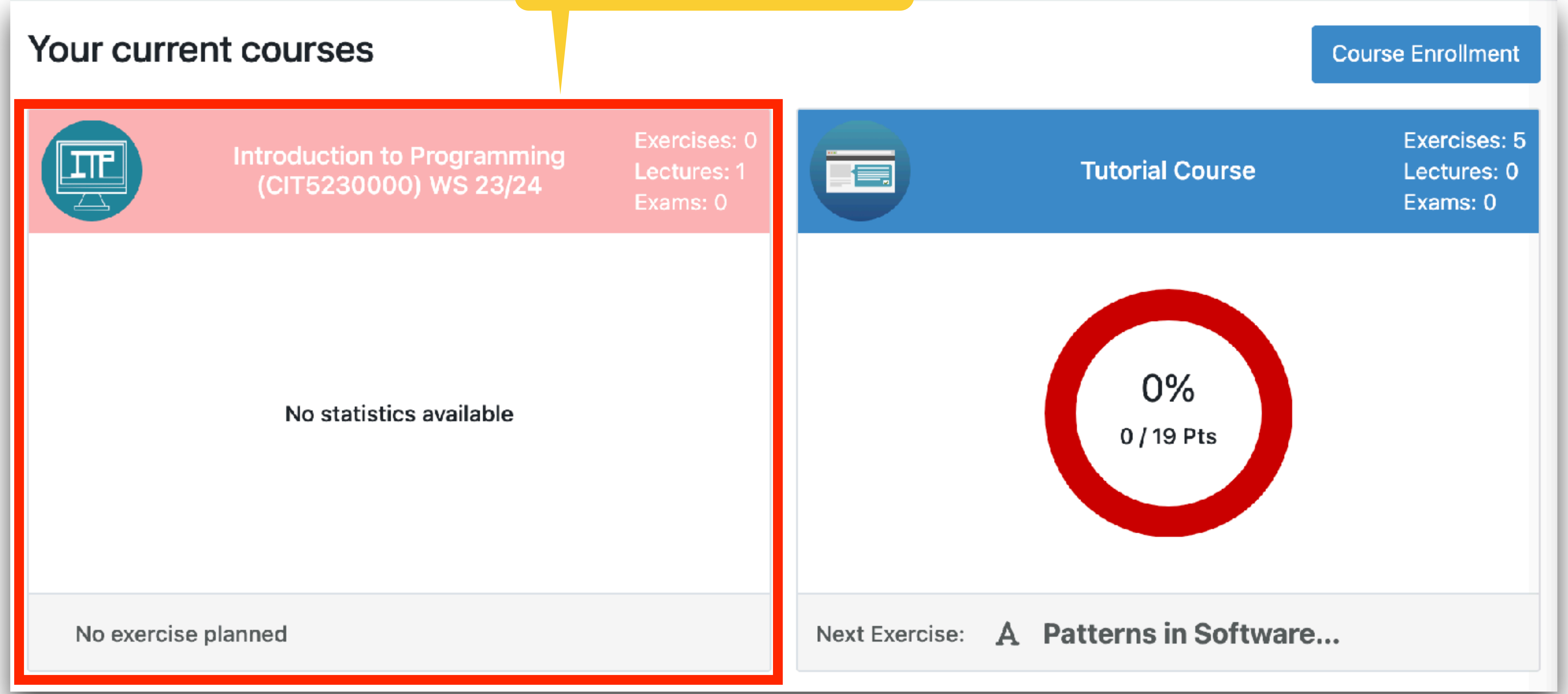
By enrolling to this course, I confirm:

1. I **study** at the School of Management (SoM).
2. I will **strictly follow** the TUM Digital Learning Commitment, the Informatics Student Code of Conduct and the ITP Student Code of Conduct.
3. I will `not` **share** course material to third parties.

Lectures, recordings, and exercises are copyrighted material. Passing on the materials to third parties is `not` permitted. Participation is only possible if you study at the SoM.

Task 1: Navigate into the course

2. Navigate into the course



Your current courses

Introduction to Programming (CIT5230000) WS 23/24

Exercises: 0
Lectures: 1
Exams: 0

No statistics available

No exercise planned

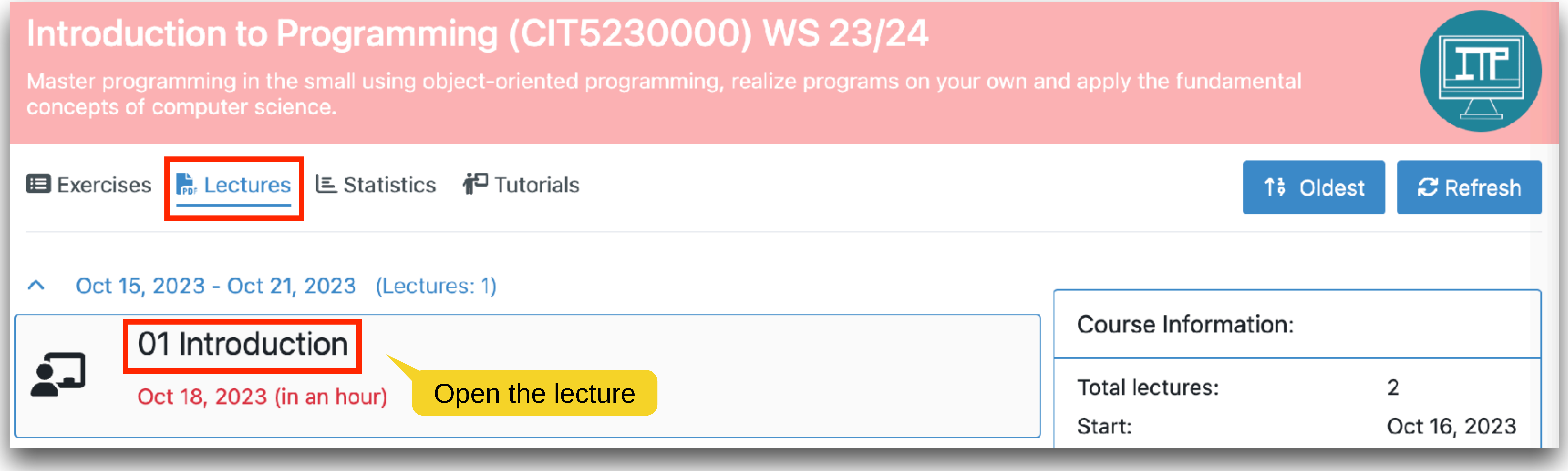
Tutorial Course

Exercises: 5
Lectures: 0
Exams: 0

0%
0 / 19 Pts

Next Exercise: Patterns in Software...

Task 2: Navigate into L01 Introduction



Introduction to Programming (CIT5230000) WS 23/24

Master programming in the small using object-oriented programming, realize programs on your own and apply the fundamental concepts of computer science.

Oct 15, 2023 - Oct 21, 2023 (Lectures: 1)

01 Introduction

Oct 18, 2023 (in an hour)

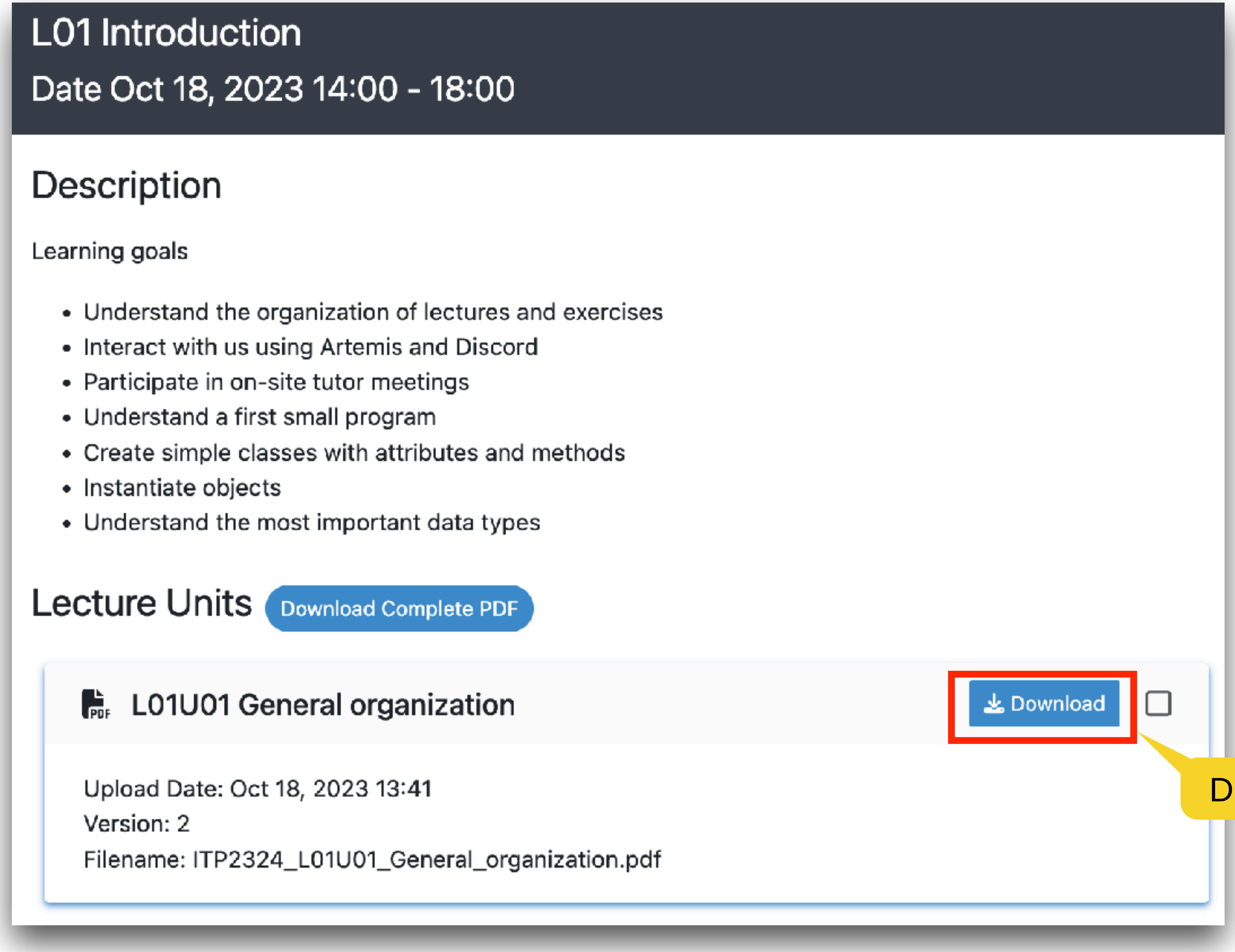
Open the lecture

Course Information:

Total lectures: 2

Start: Oct 16, 2023

Task 3: Download the slides



L01 Introduction

Date Oct 18, 2023 14:00 - 18:00

Description

Learning goals

- Understand the organization of lectures and exercises
- Interact with us using Artemis and Discord
- Participate in on-site tutor meetings
- Understand a first small program
- Create simple classes with attributes and methods
- Instantiate objects
- Understand the most important data types

Lecture Units

L01U01 General organization

Upload Date: Oct 18, 2023 13:41

Version: 2

Filename: ITP2324_L01U01_General_organization.pdf

Download the slides

© 2023 Stephan Krusche

Schedule

#	Date	Subject
1	**18.10.23**	**Introduction**
1b	25.10.23	Central exercise
	01.11.23	**No lecture**
2	08.11.23	Control Structures
3	15.11.23	Data Types
4	22.11.23	Object Orientation I
5	29.11.23	Object Orientation II
6	06.12.23	Object Orientation III
7	13.12.23	Algorithms
	20.12.23	**No lecture**
8	10.01.24	Programming Languages
9	17.01.24	Graphical User Interfaces
10	24.01.24	Recursion
11	31.01.24	Beyond Programming
12	07.02.24	Course Review

Teaching philosophy

“Tell me and I will forget.
Show me and I will remember.
Involve me and I will understand.
Step back and I will act.”

— Chinese Proverb

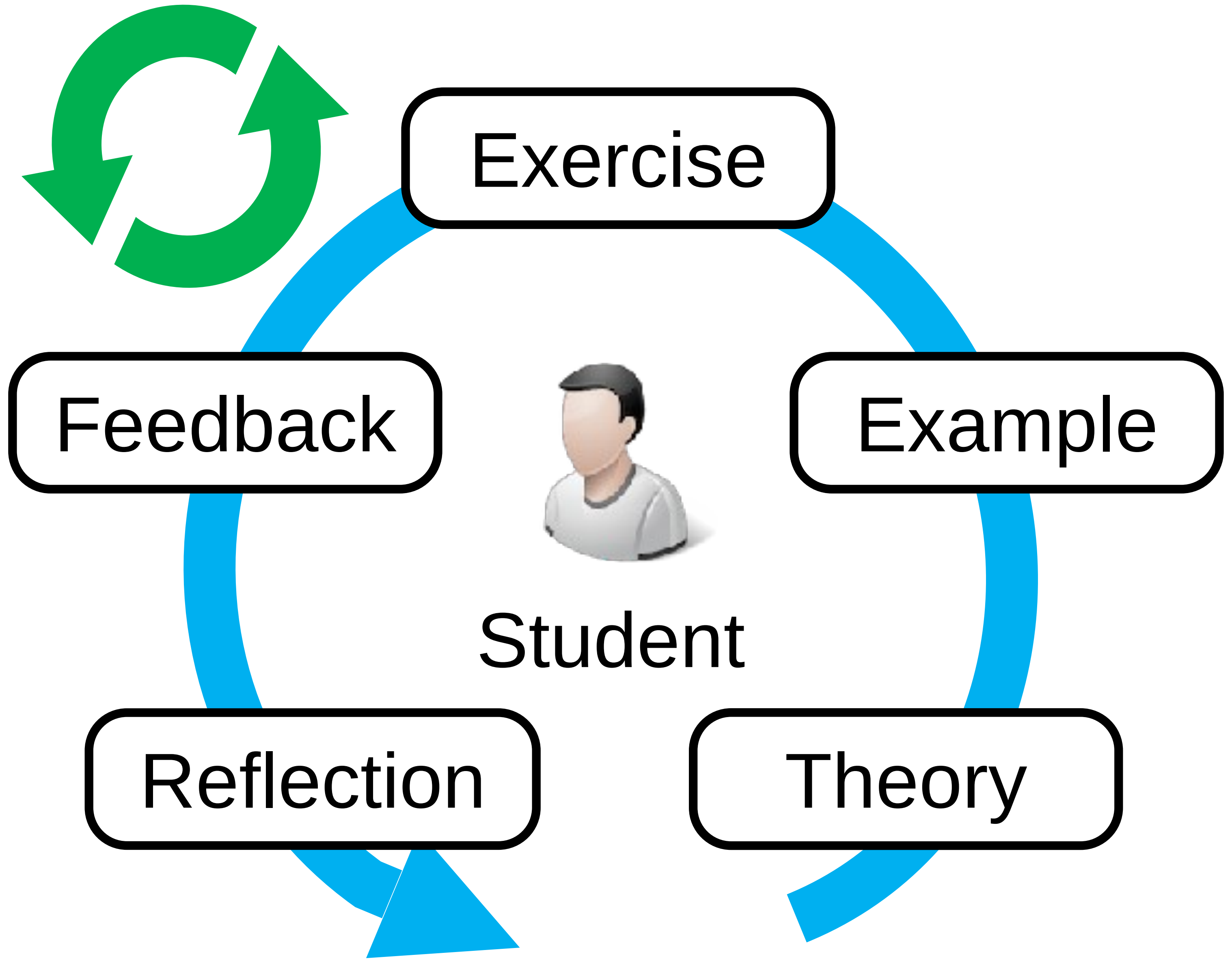
Interactive learning

- **Goal:** directly involve students in the learning process using digital tools
  - Engage students in two aspects – **doing** things and **thinking** about those things
- Lecture time slot: **Wednesday 14:00 - 18:00** with 4 x 40 min each week

1) We teach you one (or more) concepts
2) You apply these concepts in small **in-class exercises**
3) You reflect on the concepts you are applying

- Artemis supports **interactive learning**
- Tutor groups deepen the knowledge
  - Group exercises
  - Homework presentation

Soft skills: **presentation** and **communication**



Lecture

Quiz

14:00 14:10

Lecture unit 1 – 40 min – ~14:50

Break 10 min – ~15:00

Lecture unit 2 – 40 min – ~15:40

Break 30 min – ~16:10

Lecture unit 3 – 40 min – ~16:50

Break 10 min – ~17:00

Lecture unit 4 – 40 min – ~17:40

Exercise, Example, Theory, Reflection, Feedback – Student

Exercise, Example, Theory, Reflection, Feedback – Student

Exercise, Example, Theory, Reflection, Feedback – Student

Exercise, Example, Theory, Reflection, Feedback – Student

Copyright

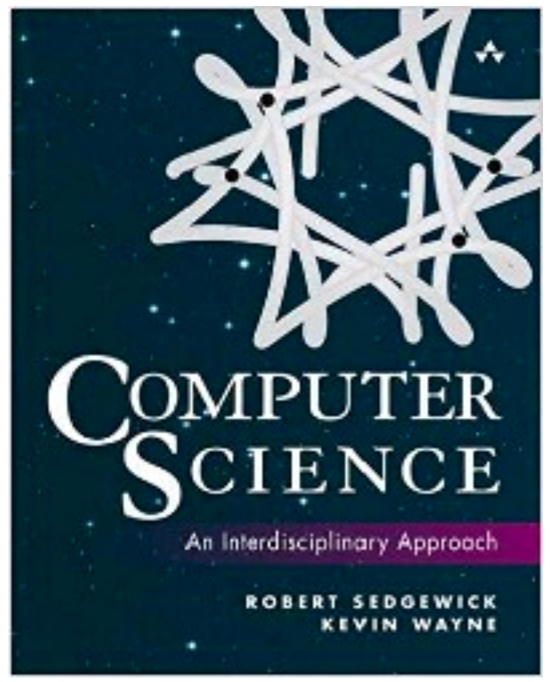
- Lecture slides and exercise resources are copyright protected
- Lectures are based on material by Helmut Seidl, Alexander Pretschner and Marco Kuhrmann
- You are **not** allowed to distribute them to other people
- You are **not** allowed to publish them on the internet

All material of this course is protected by copyright and has been copied by and solely for the educational purposes of the university under license. You may not sell, alter or further reproduce or distribute any part of this material to any other person. Where provided to you in electronic format, you may only print it for your own private study and research.

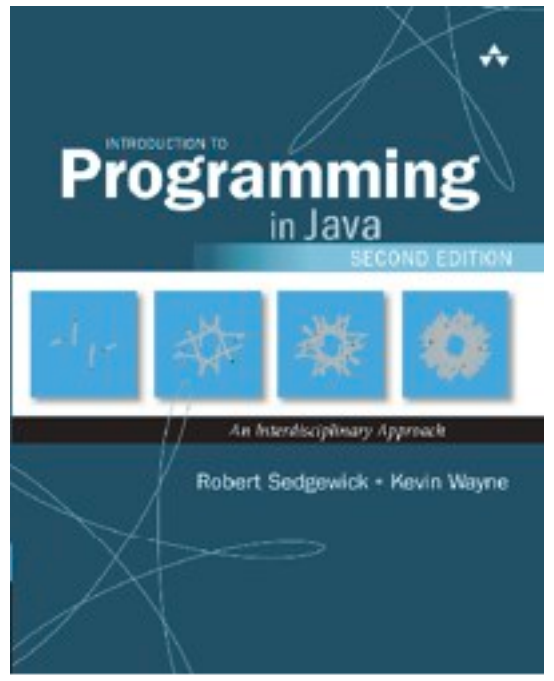
Failure to comply with the terms of this warning may expose you to legal action for copyright infringement and / or disciplinary action by the university.

Literature

- English books on computer science and Java programming



Sedgewick, Robert, and Kevin Wayne.
Computer science: An interdisciplinary approach.
Addison-Wesley Professional, 2016.



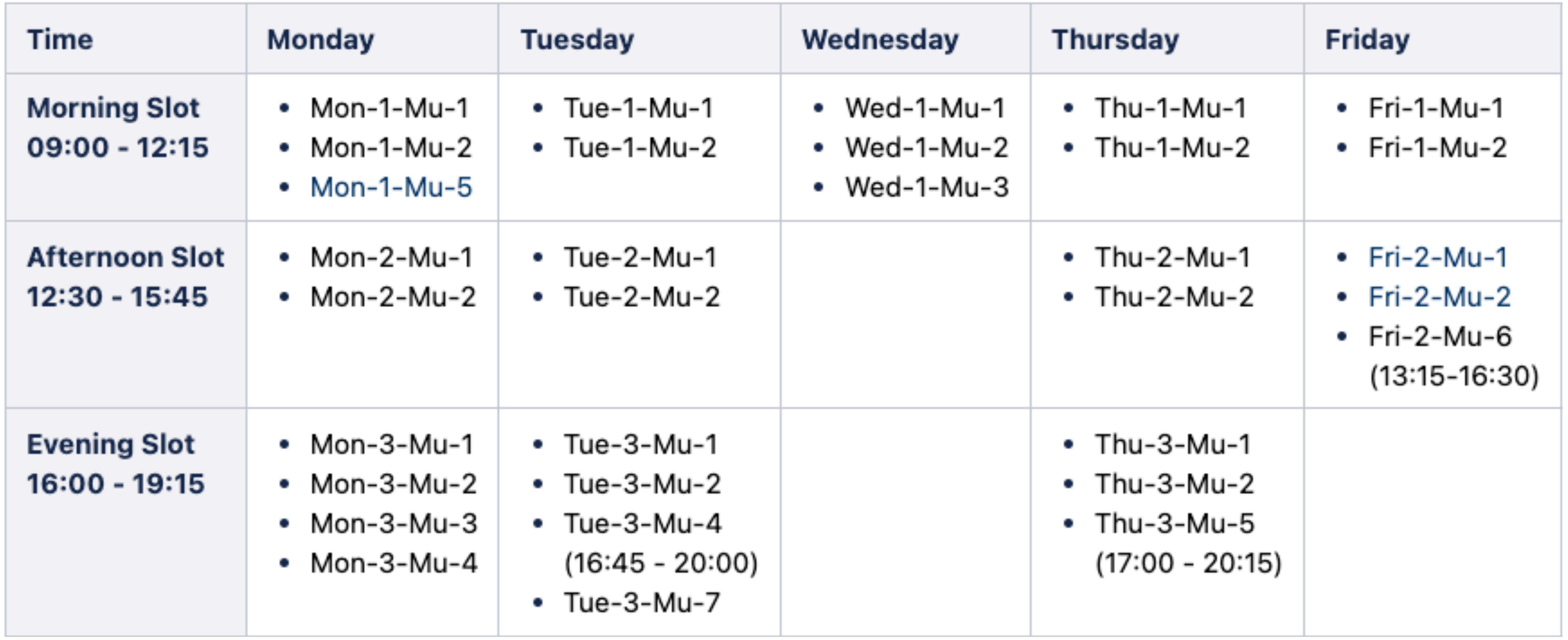
Sedgewick, Robert, and Kevin Wayne.
Introduction to programming in Java: an interdisciplinary approach.
Addison-Wesley Professional, 2017.

- Corresponding material can be found on https://introcs.cs.princeton.edu/java/home
- Specific references to literature are added in the slides if needed!

© 2023 Stephan Krusche

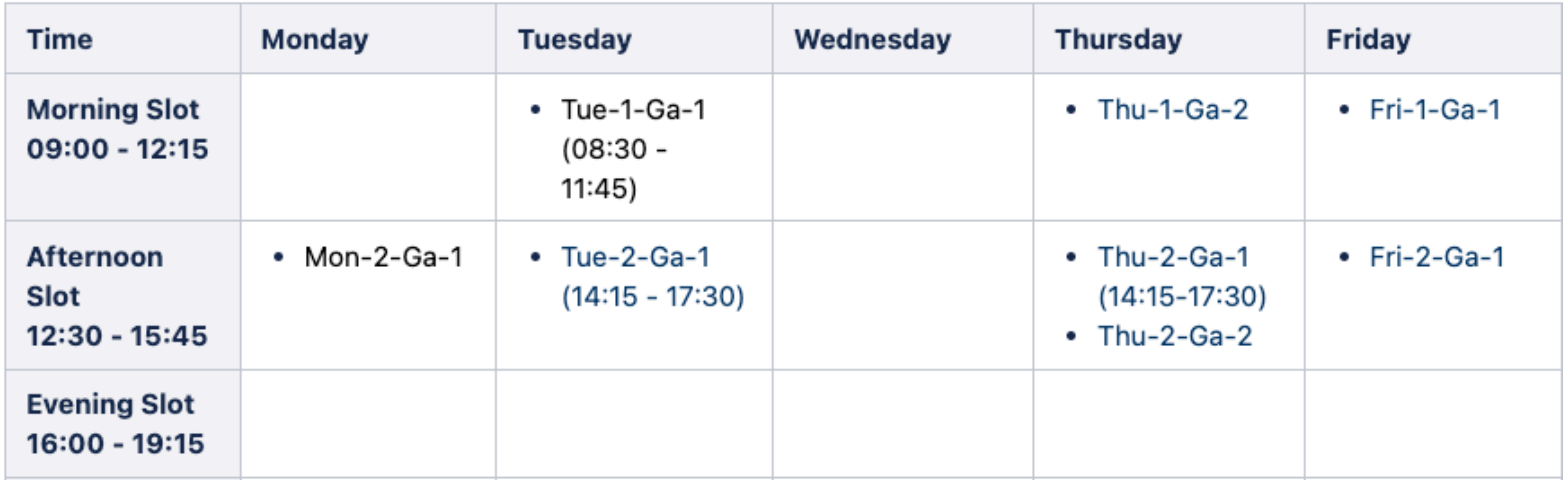
Overview of 40 tutor group slots

Munich (city campus)



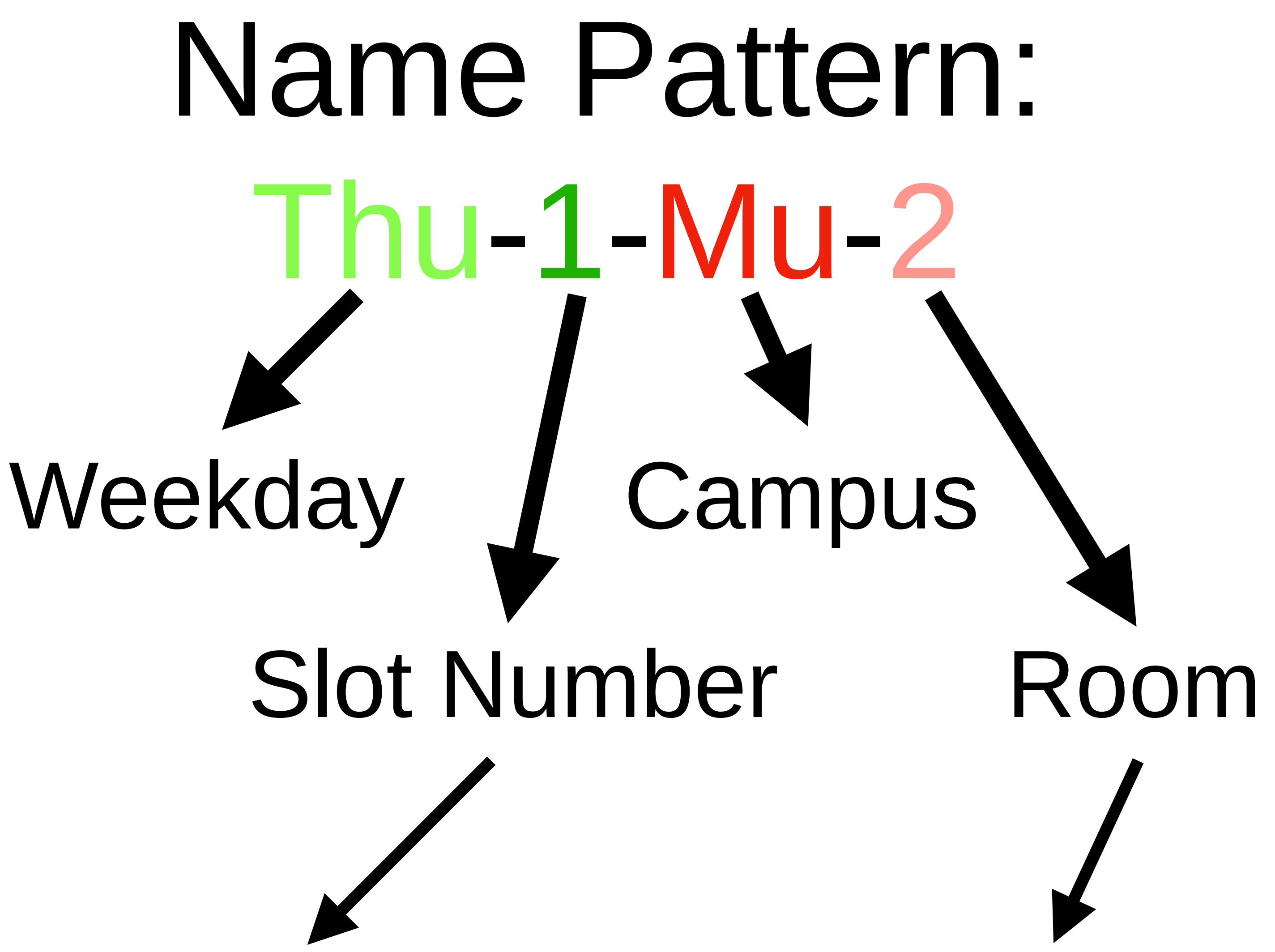
Time	Monday	Tuesday	Wednesday	Thursday	Friday
Morning Slot **09:00 - 12:15**	Mon-1-Mu-1 Mon-1-Mu-2 Mon-1-Mu-5	Tue-1-Mu-1 Tue-1-Mu-2	Wed-1-Mu-1 Wed-1-Mu-2 Wed-1-Mu-3	Thu-1-Mu-1 Thu-1-Mu-2	Fri-1-Mu-1 Fri-1-Mu-2
Afternoon Slot **12:30 - 15:45**	Mon-2-Mu-1 Mon-2-Mu-2	Tue-2-Mu-1 Tue-2-Mu-2		Thu-2-Mu-1 Thu-2-Mu-2	Fri-2-Mu-1 Fri-2-Mu-2 Fri-2-Mu-6 (13:15-16:30)
Evening Slot **16:00 - 19:15**	Mon-3-Mu-1 Mon-3-Mu-2 Mon-3-Mu-3 Mon-3-Mu-4	Tue-3-Mu-1 Tue-3-Mu-2 Tue-3-Mu-4 (16:45 - 20:00) Tue-3-Mu-7		Thu-3-Mu-1 Thu-3-Mu-2 Thu-3-Mu-5 (17:00 - 20:15)	

Garching



Time	Monday	Tuesday	Wednesday	Thursday	Friday
Morning Slot **09:00 - 12:15**		Tue-1-Ga-1 (08:30 - 11:45)		Thu-1-Ga-2	Fri-1-Ga-1
Afternoon Slot **12:30 - 15:45**	Mon-2-Ga-1	Tue-2-Ga-1 (14:15 - 17:30)		Thu-2-Ga-1 (14:15-17:30) Thu-2-Ga-2	Fri-2-Ga-1
Evening Slot **16:00 - 19:15**					

Name Pattern:



Thu-1-Mu-2

Weekday

Slot Number

Campus

Room

ID	Time
1	09:00 - 12:25
2	12:30 - 15:45
3	16:00 - 19:15

ID	Room
Mu-1	0510
Mu-2	1237
Mu-3	Z510
Mu-4	0514
Mu-5	Z538
Mu-6	0670ZG
Mu-7	3149
Ga-1	01.07.014
Ga-2	01.13.010

© 2023 Stephan Krusche

Registration for tutor groups

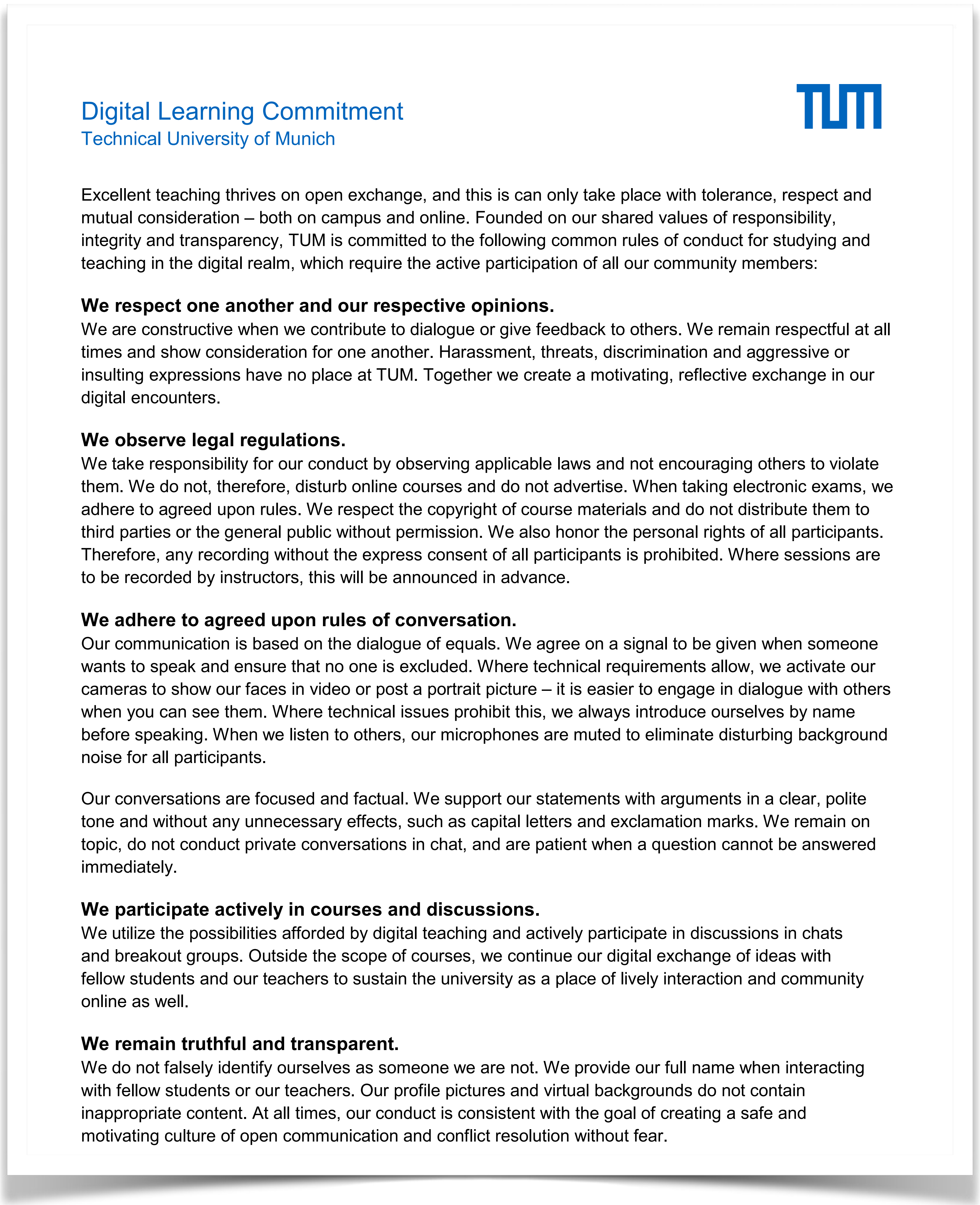
- URL to register and enter your priorities:
  https://campus.tum.de/tumonline/ee/ui/ca2/app/desktop/#/slc.tm.cp/student/courses/950701484
- Timeline
  - Enter your tutor group preferences **before today, 15:30 (October 18)**
  - Receive your assignment today on TUMonline
  - The first meetings take place **tomorrow, Thursday, October 19**
- Exception: if you missed the matching, could not be matched or want to change your group
  - Late registration or a change of the tutor group will be possible until **Friday (October 27)** on TUMonline (there is a daily matching to assign new students)
  - Contact your fellow students on **Discord** if you want to switch
  - **Please note:** changing a group is only possible with a good reason

© 2023 Stephan Krusche

Tutor group and exercise rules

- Make sure to attend the tutor groups
- In case your tutor group meeting is cancelled (SVV, FVV, holiday): **reallocate yourself to another tutor group**
- You can only submit solutions to exercises on Artemis
- Each exercise has a due date
  - If you miss the deadline, your solution will **not** be graded
- **Important:** exercises must be completed and submitted **independently**
  - You need a unique solution to the exercises
  - Plagiarism, i.e. copying blindly without understanding the concepts and without applying the skills, might be tempting, but you basically betray yourself

Code of conduct



Digital Learning Commitment

Technical University of Munich

Excellent teaching thrives on open exchange, and this is can only take place with tolerance, respect and mutual consideration – both on campus and online. Founded on our shared values of responsibility, integrity and transparency, TUM is committed to the following common rules of conduct for studying and teaching in the digital realm, which require the active participation of all our community members:

We respect one another and our respective opinions.
We are constructive when we contribute to dialogue or give feedback to others. We remain respectful at all times and show consideration for one another. Harassment, threats, discrimination and aggressive or insulting expressions have no place at TUM. Together we create a motivating, reflective exchange in our digital encounters.

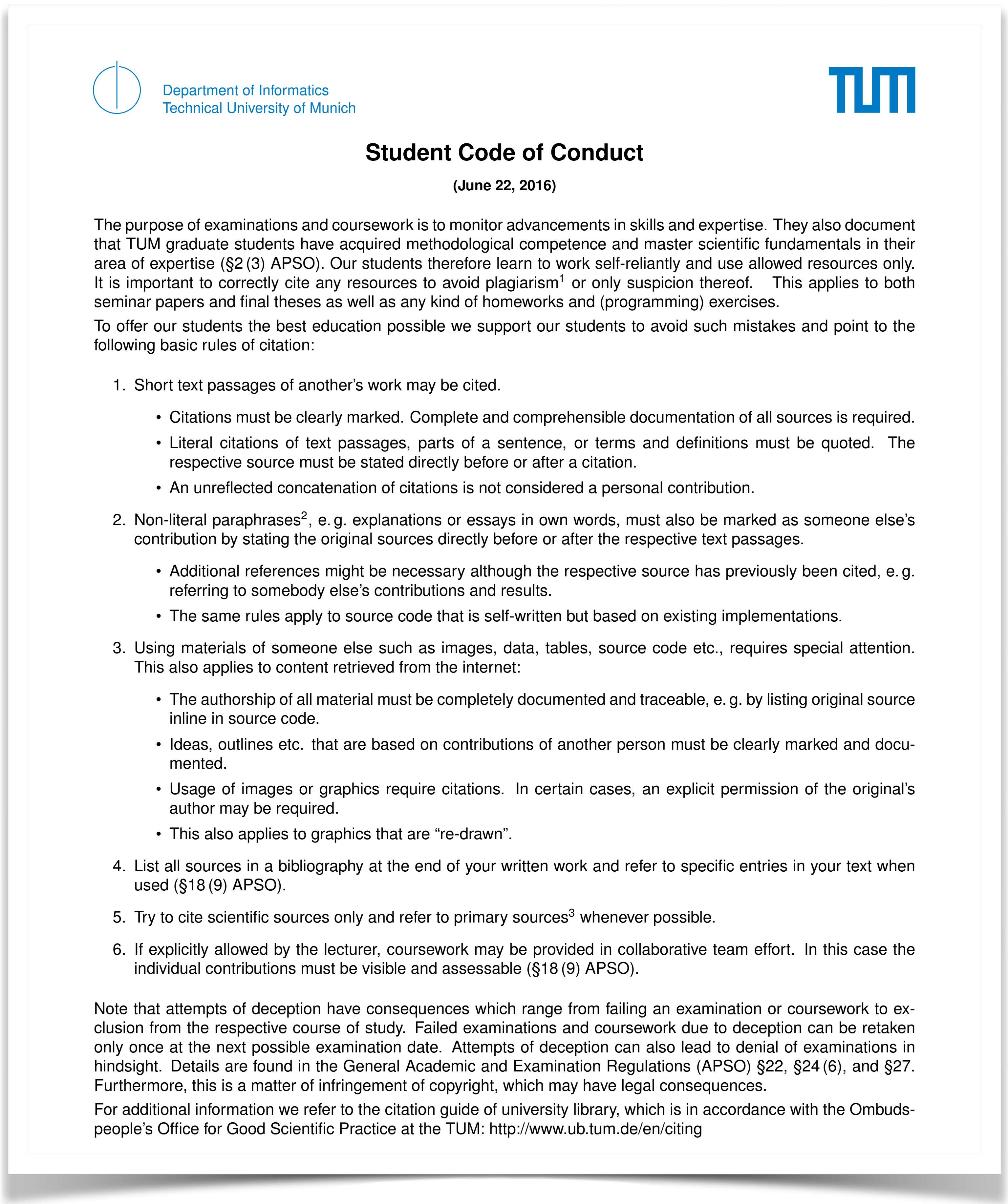
We observe legal regulations.
We take responsibility for our conduct by observing applicable laws and not encouraging others to violate them. We do not, therefore, disturb online courses and do not advertise. When taking electronic exams, we adhere to agreed upon rules. We respect the copyright of course materials and do not distribute them to third parties or the general public without permission. We also honor the personal rights of all participants. Therefore, any recording without the express consent of all participants is prohibited. Where sessions are to be recorded by instructors, this will be announced in advance.

We adhere to agreed upon rules of conversation.
Our communication is based on the dialogue of equals. We agree on a signal to be given when someone wants to speak and ensure that no one is excluded. Where technical requirements allow, we activate our cameras to show our faces in video or post a portrait picture – it is easier to engage in dialogue with others when you can see them. Where technical issues prohibit this, we always introduce ourselves by name before speaking. When we listen to others, our microphones are muted to eliminate disturbing background noise for all participants.

Our conversations are focused and factual. We support our statements with arguments in a clear, polite tone and without any unnecessary effects, such as capital letters and exclamation marks. We remain on topic, do not conduct private conversations in chat, and are patient when a question cannot be answered immediately.

We participate actively in courses and discussions.
We utilize the possibilities afforded by digital teaching and actively participate in discussions in chats and breakout groups. Outside the scope of courses, we continue our digital exchange of ideas with fellow students and our teachers to sustain the university as a place of lively interaction and community online as well.

We remain truthful and transparent.
We do not falsely identify ourselves as someone we are not. We provide our full name when interacting with fellow students or our teachers. Our profile pictures and virtual backgrounds do not contain inappropriate content. At all times, our conduct is consistent with the goal of creating a safe and motivating culture of open communication and conflict resolution without fear.



Department of Informatics
Technical University of Munich

Student Code of Conduct

(June 22, 2016)

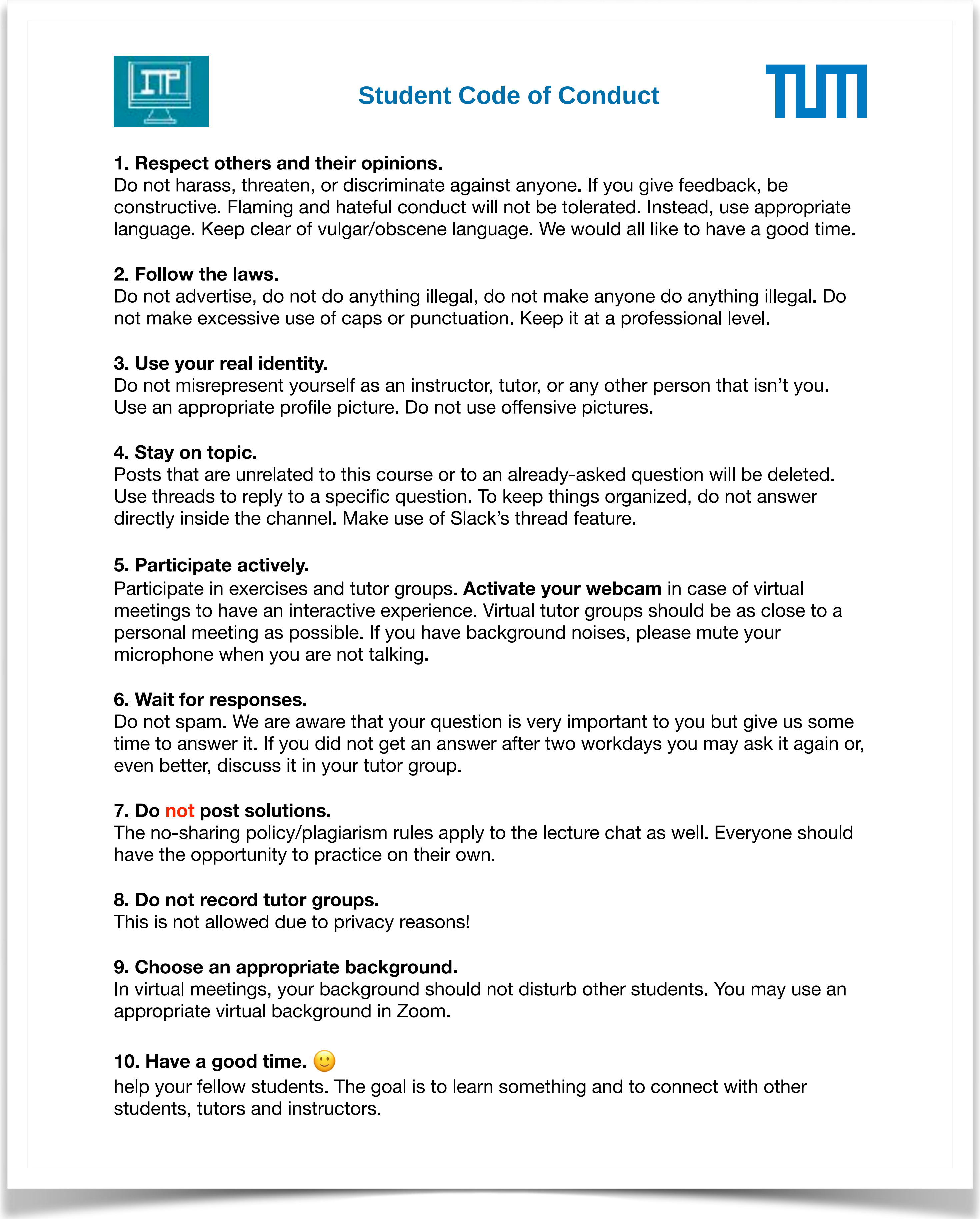
The purpose of examinations and coursework is to monitor advancements in skills and expertise. They also document that TUM graduate students have acquired methodological competence and master scientific fundamentals in their area of expertise (§2 (3) APSO). Our students therefore learn to work self-reliantly and use allowed resources only. It is important to correctly cite any resources to avoid plagiarism[1] or only suspicion thereof. This applies to both seminar papers and final theses as well as any kind of homeworks and (programming) exercises.

To offer our students the best education possible we support our students to avoid such mistakes and point to the following basic rules of citation:

1. Short text passages of another's work may be cited.
   - Citations must be clearly marked. Complete and comprehensible documentation of all sources is required.
   - Literal citations of text passages, parts of a sentence, or terms and definitions must be quoted. The respective source must be stated directly before or after a citation.
   - An unreflected concatenation of citations is not considered a personal contribution.
2. Non-literal paraphrases[2], e. g. explanations or essays in own words, must also be marked as someone else's contribution by stating the original sources directly before or after the respective text passages.
   - Additional references might be necessary although the respective source has previously been cited, e. g. referring to somebody else's contributions and results.
   - The same rules apply to source code that is self-written but based on existing implementations.
3. Using materials of someone else such as images, data, tables, source code etc., requires special attention. This also applies to content retrieved from the internet:
   - The authorship of all material must be completely documented and traceable, e. g. by listing original source inline in source code.
   - Ideas, outlines etc. that are based on contributions of another person must be clearly marked and documented.
   - Usage of images or graphics require citations. In certain cases, an explicit permission of the original's author may be required.
   - This also applies to graphics that are "re-drawn".
4. List all sources in a bibliography at the end of your written work and refer to specific entries in your text when used (§18 (9) APSO).
5. Try to cite scientific sources only and refer to primary sources[3] whenever possible.
6. If explicitly allowed by the lecturer, coursework may be provided in collaborative team effort. In this case the individual contributions must be visible and assessable (§18 (9) APSO).

Note that attempts of deception have consequences which range from failing an examination or coursework to exclusion from the respective course of study. Failed examinations and coursework due to deception can be retaken only once at the next possible examination date. Attempts of deception can also lead to denial of examinations in hindsight. Details are found in the General Academic and Examination Regulations (APSO) §22, §24 (6), and §27. Furthermore, this is a matter of infringement of copyright, which may have legal consequences.

For additional information we refer to the citation guide of university library, which is in accordance with the Ombudspeople's Office for Good Scientific Practice at the TUM: http://www.ub.tum.de/en/citing



Student Code of Conduct

1. Respect others and their opinions.
Do not harass, threaten, or discriminate against anyone. If you give feedback, be constructive. Flaming and hateful conduct will not be tolerated. Instead, use appropriate language. Keep clear of vulgar/obscene language. We would all like to have a good time.

2. Follow the laws.
Do not advertise, do not do anything illegal, do not make anyone do anything illegal. Do not make excessive use of caps or punctuation. Keep it at a professional level.

3. Use your real identity.
Do not misrepresent yourself as an instructor, tutor, or any other person that isn't you. Use an appropriate profile picture. Do not use offensive pictures.

4. Stay on topic.
Posts that are unrelated to this course or to an already-asked question will be deleted. Use threads to reply to a specific question. To keep things organized, do not answer directly inside the channel. Make use of Slack's thread feature.

5. Participate actively.
Participate in exercises and tutor groups. **Activate your webcam** in case of virtual meetings to have an interactive experience. Virtual tutor groups should be as close to a personal meeting as possible. If you have background noises, please mute your microphone when you are not talking.

6. Wait for responses.
Do not spam. We are aware that your question is very important to you but give us some time to answer it. If you did not get an answer after two workdays you may ask it again or, even better, discuss it in your tutor group.

7. Do not post solutions.
The no-sharing policy/plagiarism rules apply to the lecture chat as well. Everyone should have the opportunity to practice on their own.

8. Do not record tutor groups.
This is not allowed due to privacy reasons!

9. Choose an appropriate background.
In virtual meetings, your background should not disturb other students. You may use an appropriate virtual background in Zoom.

10. Have a good time. 🙂
help your fellow students. The goal is to learn something and to connect with other students, tutors and instructors.

Violations may result in a permanent exclusion from attending tutor groups, interacting in Artemis, or even participating in the course!

You can find the full version of all three code of conducts as PDF on https://artemis.cit.tum.de/courses/274/lectures/738

© 2023 Stephan Krusche

Student code of conduct (in this course)

1. Respect others and their opinions
2. Follow the laws
3. Use your real identity
4. Stay on topic
5. Participate actively
6. Wait for responses
7. Do **not** post solutions
8. Do not record tutor groups
9. Have a good time 🙂

Violations may result in a permanent exclusion from attending tutor groups, interacting in Artemis, or even participating in the course!

You can find the full version of all three code of conducts as PDF on https://artemis.cit.tum.de/courses/274/lectures/738

Typical schedule of the tutor group meetings

1. **Short review (max. 5 min)** of the lecture content (**no repetition**)
    - Your tutor quickly shows the roadmap, outline and summary slides of the lecture
    - You can ask questions, if something is unclear (prepare them in advance)
2. **Presentation and discussion of previous homework (20 - 30 min)**
    - Present your solution to your tutor and the other students
3. **Group exercises (120 - 150 min)**
    - Solve exercises in a group and present your solution
4. **Short discussion of the new homework (max. 5 min)**
    - Your tutor quickly shows the new homework
    - You can ask questions, if something is unclear (prepare them in advance)

© 2023 Stephan Krusche

Assessment

Activity	Weight
Intermediate exam 1 (Monday, 20.11.2023, 19:00 - 20:40)	20%
Intermediate exam 2 (Monday, 18.12.2023, 19:00 - 20:40)	20%
Project work: development of a game (January)	20%
Final exam (Tuesday, 20.02.2024, 15:30 - 17:10)	30%
Presentations (homework + tutor exercise)	10%
Total	**100%**
Bonus for selected lecture and homework exercises	up to 10%

Computer-based exam in the lecture hall

Everything on Artemis with a bonus category

More information will be announced later

Possible grading key*

- Sum of weighted scores (based on previous slide)

Score in %	➡	Grade
[0.0, 40.0)	➡	5.0
[40.0, 45.0)	➡	4.7
[45.0, 50.0)	➡	4.3
[50.0, 55.0)	➡	4.0
[55.0, 60.0)	➡	3.7
[60.0, 65.0)	➡	3.3
[65.0, 70.0)	➡	3.0
[70.0, 75.0)	➡	2.7
[75.0, 80.0)	➡	2.3
[80.0, 85.0)	➡	2.0
[85.0, 90.0)	➡	1.7
[90.0, 95.0)	➡	1.3
[95.0, 100.0]	➡	1.0

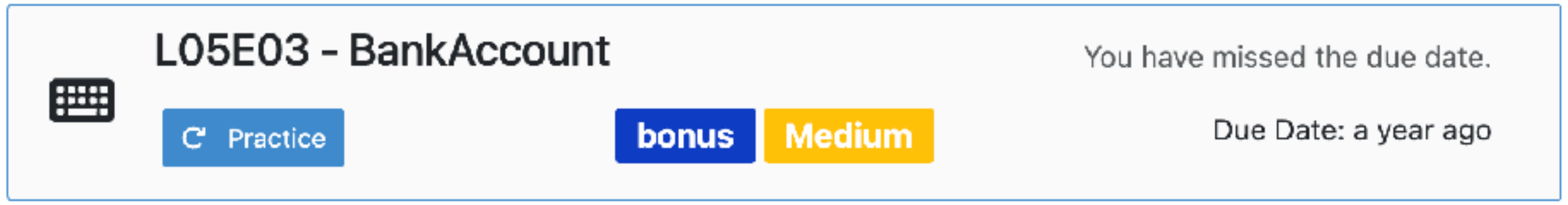
$(a, b] = \{ x \mid a < x \leq b \}$ $[a, b) = \{ x \mid a \leq x < b \}$

Rounding: to the nearest first decimal place (half up)

∗ subject to change

Bonus system

- Participate in lecture and homework exercises to receive points 🏆
- Exercises are marked with a bonus category

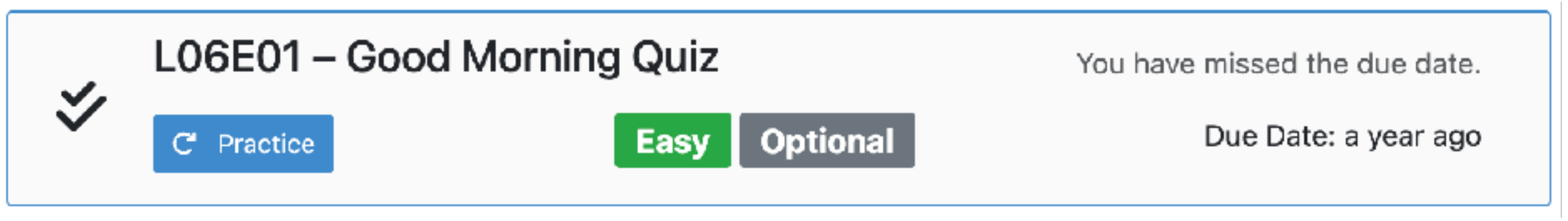


L05E03 - BankAccount

You have missed the due date.

Due Date: a year ago

- Other (non-bonus) exercises are marked as optional



L06E01 – Good Morning Quiz

You have missed the due date.

Due Date: a year ago

- Mapping of **exercise score** to **grade score**
  - Prerequisite: you have passed the course

➡ **Important:** the bonus is **not** applicable to the retake

Exercise score	➡	Additional grade score
[0%, 5%)	➡	0.0%
[5%, 10%)	➡	0.5%
[10%, 15%)	➡	1.0%
...	...	...
[90%, 95%)	➡	9.0%
[95%, 100%)	➡	9.5%
[100%,...%	➡	10.0%

$(a, b] = \{ x \mid a < x \leq b \}$

$[a, b) = \{ x \mid a \leq x < b \}$

Exercise types

✓ **1. Lecture exercises (e.g. L01E01)**

✓ **2. Group exercises (e.g. T01E01)**

- Solve exercises in a group during the tutor group meeting
- Presented in the same tutor groups
- Prepare for homework exercises and for the final exam

➡ **3. Homework exercises (e.g. H01E01)**

- Published every **Wednesday evening**: to be solved within one week at home
- Assessed automatically (real time feedback)
- Presented and discussed in the subsequent tutor group

Assessment and complaints

- Artemis uses double blind assessments to improve the fairness
  - The tutor does not know the identity of the student
  - The student does not know the identity of the tutor
- If you think the assessment is wrong, you can complain on Artemis
  - After you have received the result on Artemis, you have 7 days to complain
  - You have 3 open complaints for the whole semester
  - Each complaint will be reviewed by a second tutor

Break

10 min

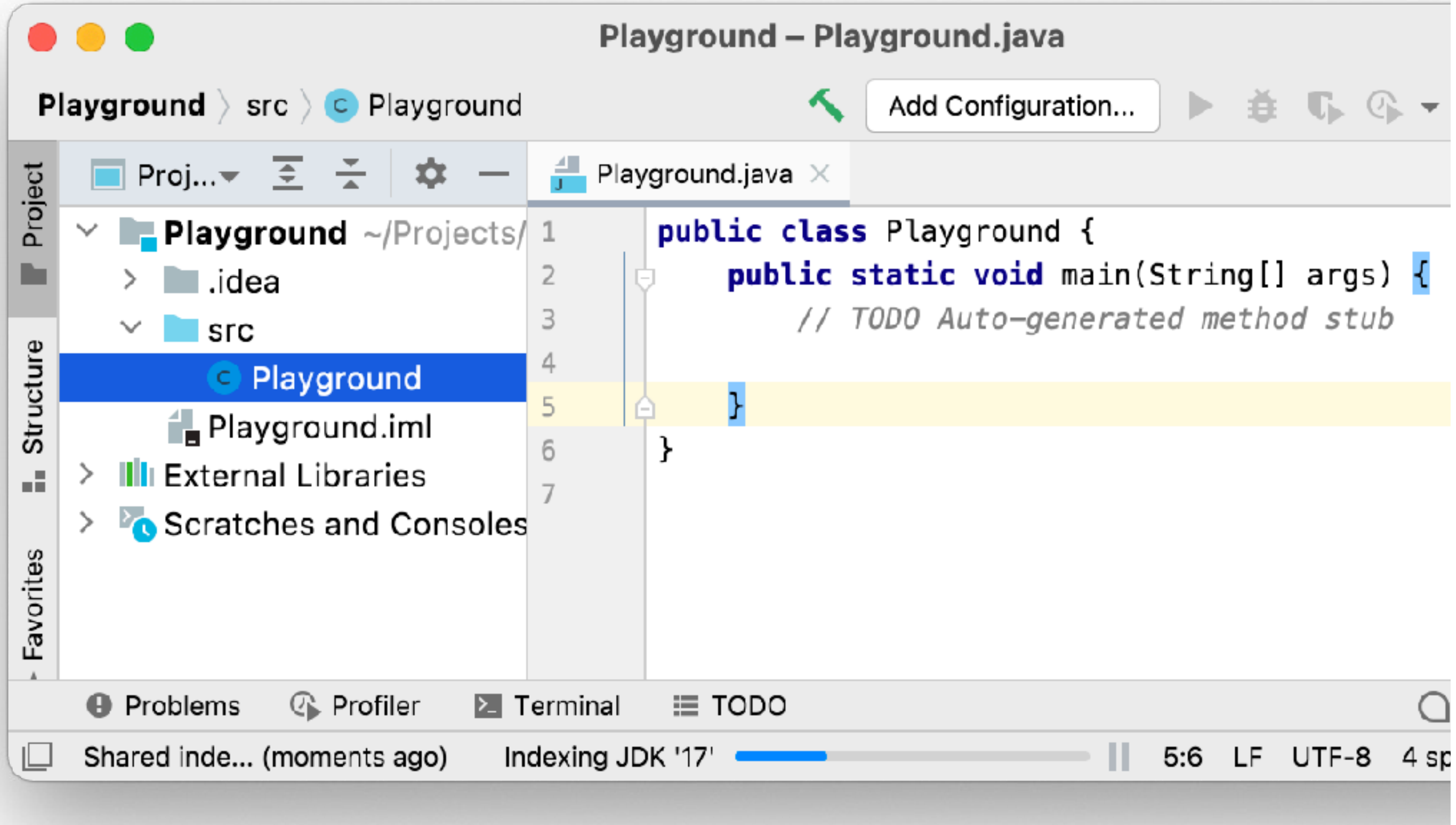
The lecture will continue at **15:10**

Outline

- General organization of the course
- ➡ **Object oriented programming**
- Basic types and their operations
- Methods
- UML class diagrams

IntelliJ playground

- Let’s open IntelliJ
- Create a new Java project called **Playground** with a new class **`Playground.java`**



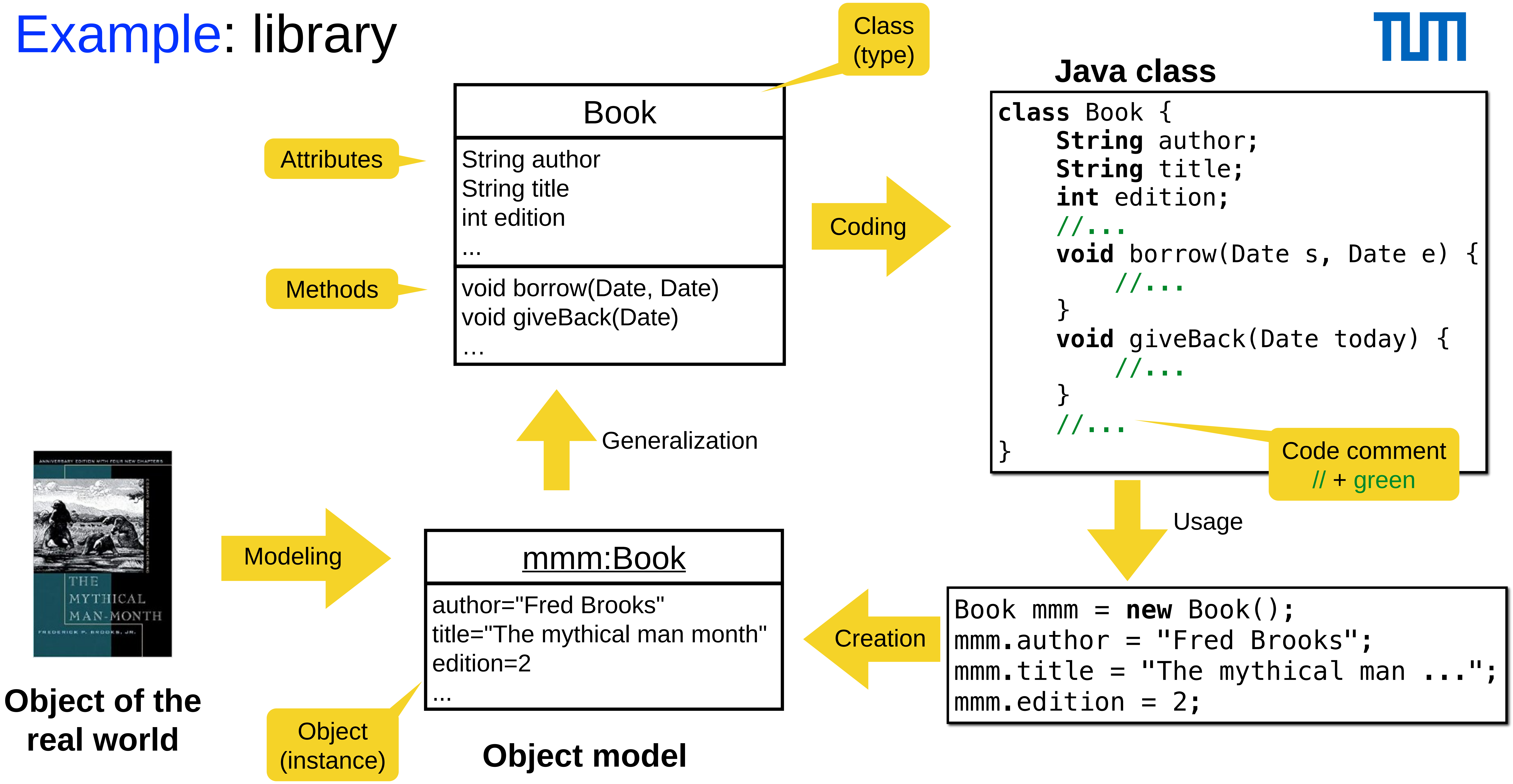
```java
public class Playground {
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }
}
```

→ Please code the examples from the slides into the Playground project

Object oriented programming

- The world consists of objects
- Basic idea
  - Every object in reality has a virtual counterpart
  - Objects cooperate through data exchange and invocation of services
- Perspective
  - "Who does what?” → Properties and capabilities of objects
  - Algorithmic perspective: "How is it done?”

Example: library


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.

Object of the real world

Modeling

Object model

Object (instance)

mmm:Book

author="Fred Brooks"
title="The mythical man month"
edition=2
...

Generalization

Class (type)

Book

Attributes

String author
String title
int edition
...

Methods

void borrow(Date, Date)
void giveBack(Date)
…

Coding

Java class

```java
class Book {
    String author;
    String title;
    int edition;
    //...
    void borrow(Date s, Date e) {
        //...
    }
    void giveBack(Date today) {
        //...
    }
    //...
}
```

Code comment
// + green

Usage

```java
Book mmm = new Book();
mmm.author = "Fred Brooks";
mmm.title = "The mythical man ...";
mmm.edition = 2;
```

Creation

Definition: object and class

- Three aspects characterize objects
  - **Identity:** properties can change, identity is constant
    Ensured by **name** (reference)
  - **State:** set of properties at a point in time
    Realized by **attributes**
  - **Behavior:** execution of actions that change the state
    Realized by **methods**
- A class is a "blueprint" for similar objects and determines
  - Which attributes the objects have
  - Which methods the objects have

Classes in Java

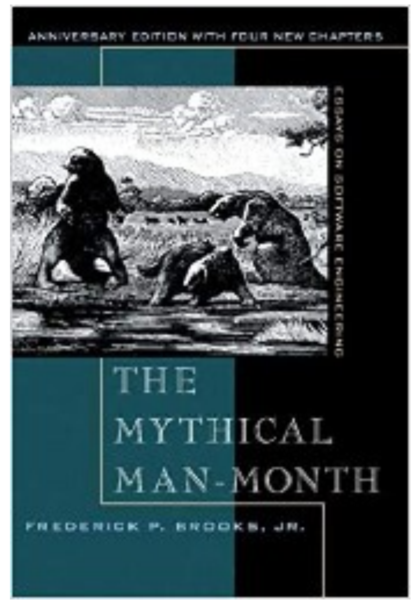
```java
import java.util.Date;

class Book {
    // Attributes
    String author;
    String title;
    int edition;
    //...

    // Constructor
    Book() {
        //... initialize the object
    }

    // Methods
    void borrow(Date s, Date e) {
        //...
    }
    void giveBack(Date today) {
        //...
    }
    //...
}
```

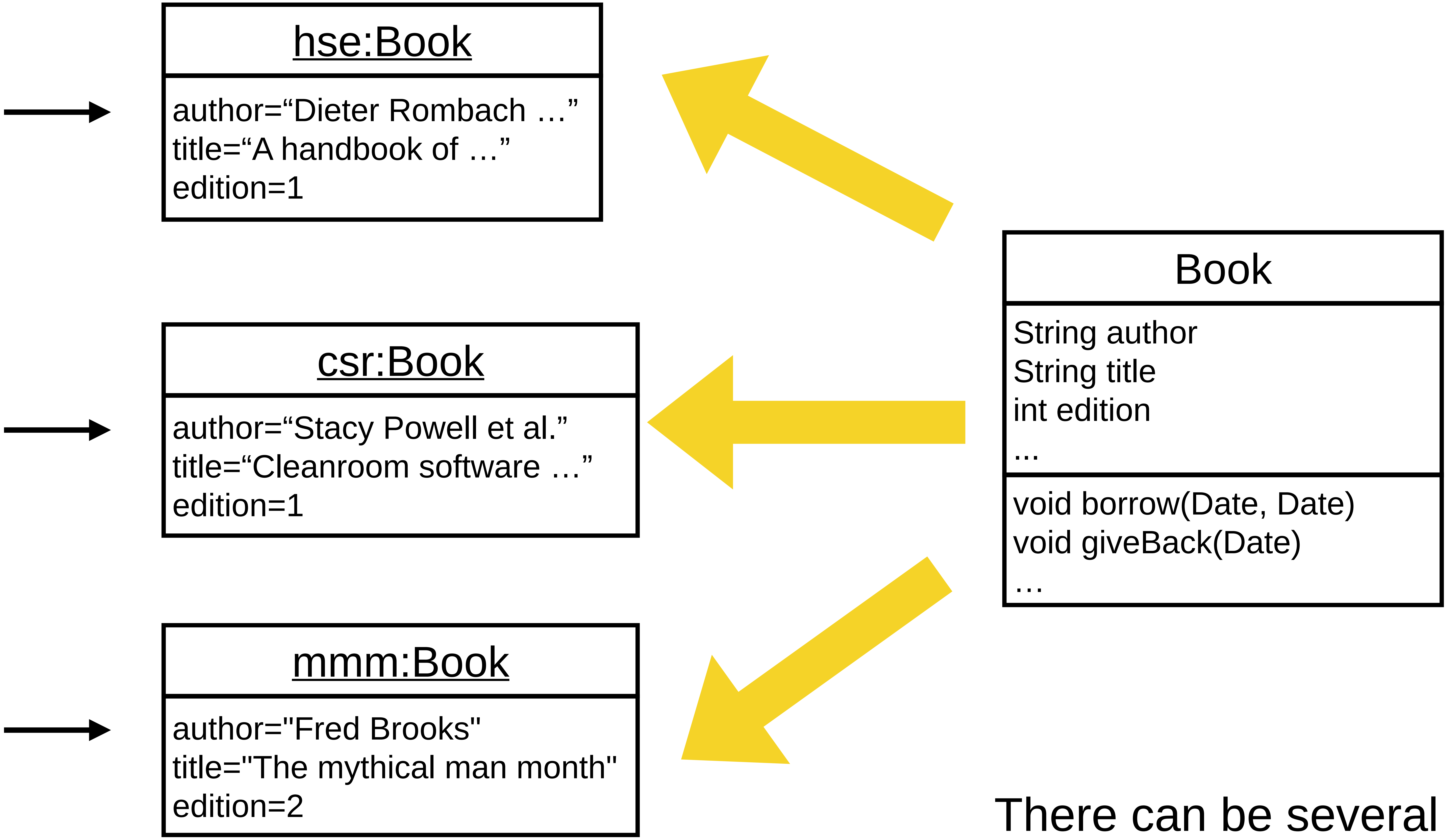
Schema

```
class Book {
  body
}
```

© 2023 Stephan Krusche

Example: library





<u>hse:Book</u>

author=“Dieter Rombach …”
title=“A handbook of …”
edition=1

<u>csr:Book</u>

author=“Stacy Powell et al.”
title=“Cleanroom software …”
edition=1

<u>mmm:Book</u>

author="Fred Brooks"
title="The mythical man month"
edition=2

Book

String author
String title
int edition
...

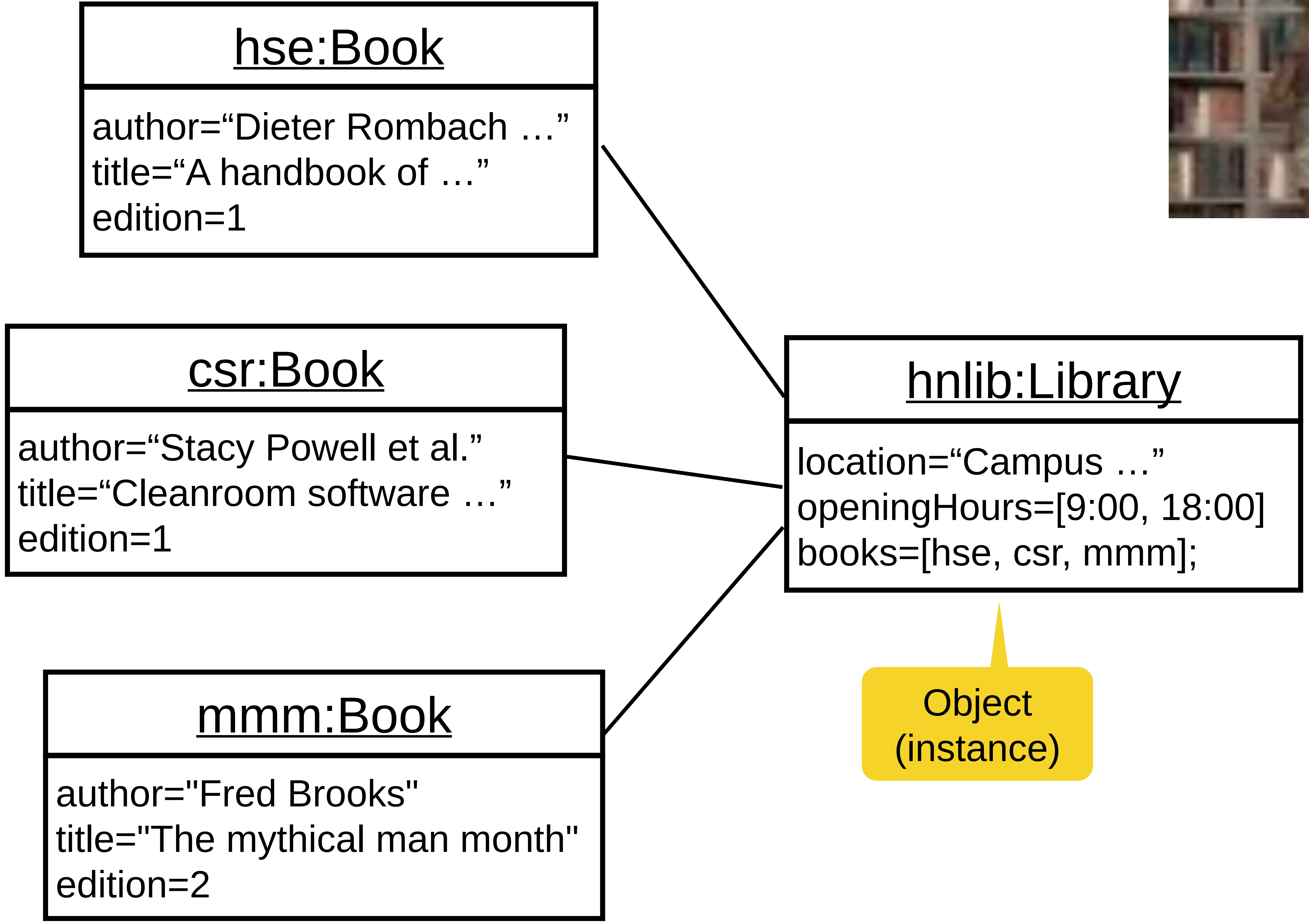
void borrow(Date, Date)
void giveBack(Date)
…

Object model

There can be several objects (instances) of a class (type) !

Example: library

Object of the real world



hse:Book

author="Dieter Rombach …"
title="A handbook of …"
edition=1

csr:Book

author="Stacy Powell et al."
title="Cleanroom software …"
edition=1

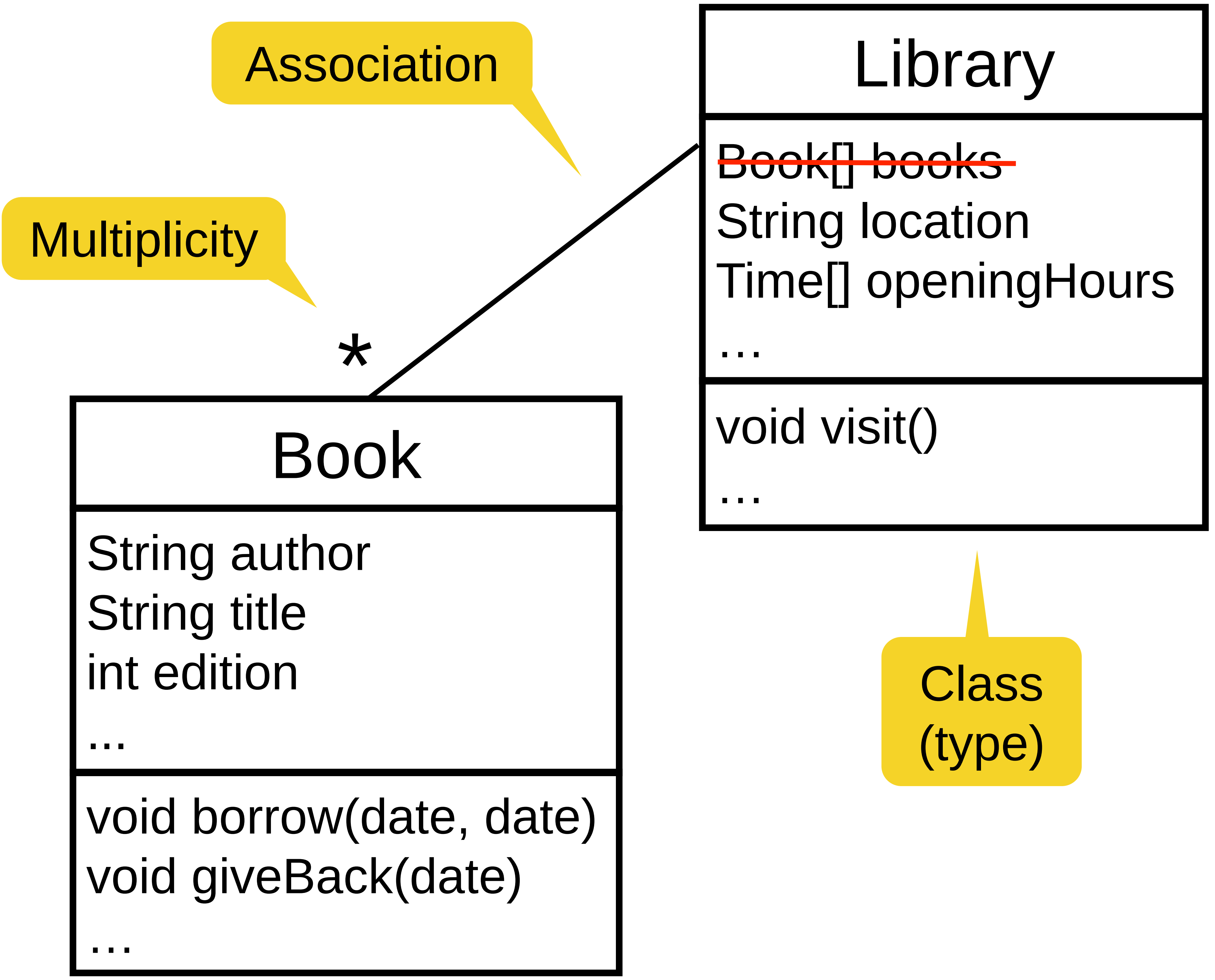
mmm:Book

author="Fred Brooks"
title="The mythical man month"
edition=2

hnlib:Library

location="Campus …"
openingHours=[9:00, 18:00]
books=[hse, csr, mmm];

Object (instance)



Association

Multiplicity

*

Library

~~Book[] books~~
String location
Time[] openingHours
…

void visit()
…

Book

String author
String title
int edition
...

void borrow(date, date)
void giveBack(date)
…

Class (type)

Different objects belong together

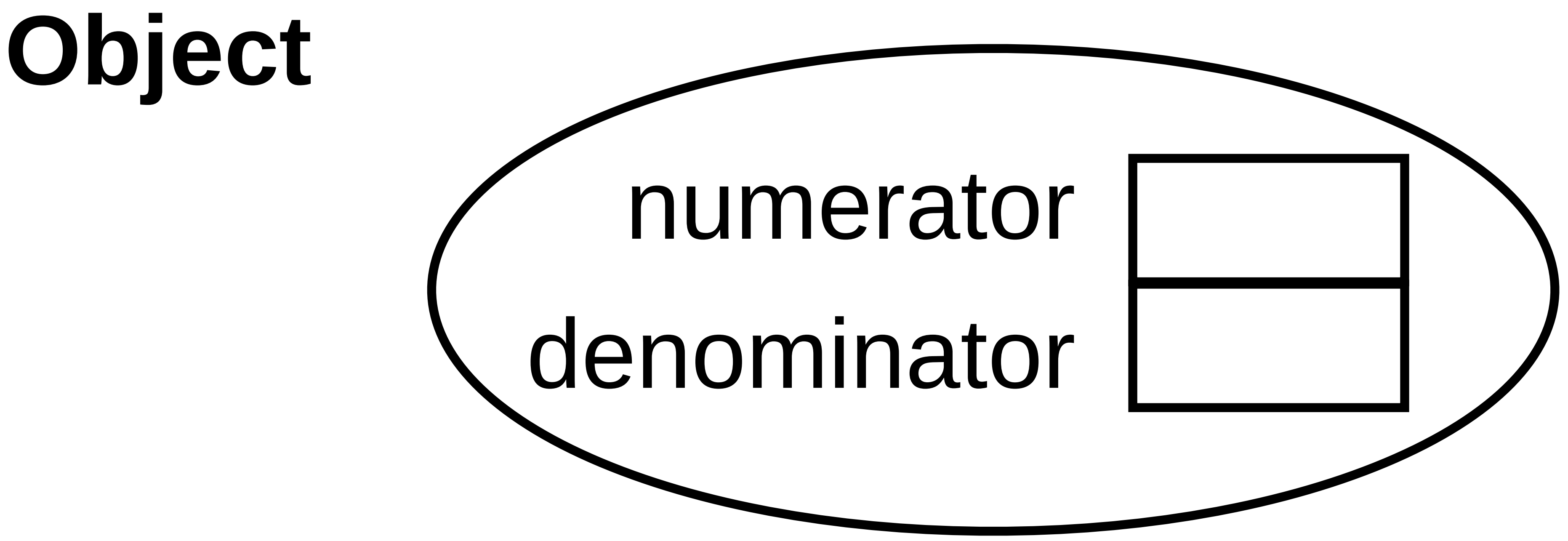
© 2023 Stephan Krusche

Variables

- The attributes (data components) of an object are also called instance variables
- Later we will learn about other variables that are defined for
  - Classes
  - Methods (locally defined)
- Variables must be declared
- A constructor must be called (except for so-called basic types)

Example: rationale numbers

- A rationale number q ∈ Q has the form q = $\frac{x}{y}$, where $x, y \in Z$
- x and y are called numerator and denominator of q
- An object of type **`Rationale`** should contain two `int` variables **`numerator`** and **`denominator`** as components (attributes)



Example: rationale numbers

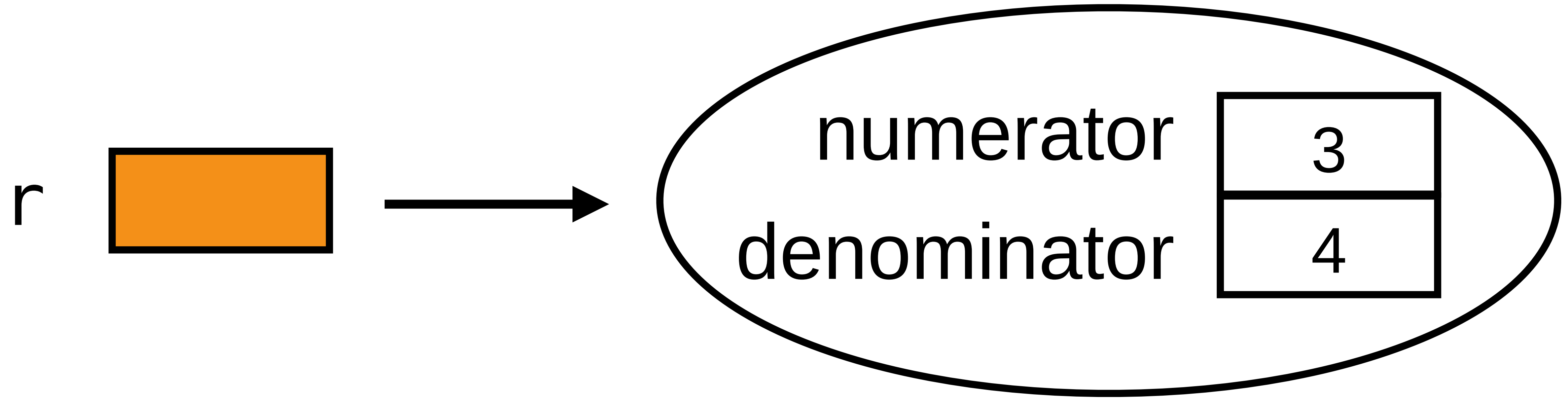
Code

```
class Rationale {
    int numerator;       // Declaration of an int attribute
    int denominator;     // Declaration of an int attribute

    // constructor Rationale() exists automatically

    // methods . . .
}
```

Declaration and constructor

- **`Rationale name;`** declares a variable for objects of the class `Rationale`
- The command `new Rationale(...)` creates the object, calls a constructor for this object and returns the new object

```
Rationale r = new Rationale();
r.numerator = 3;
r.denominator = 4;
```



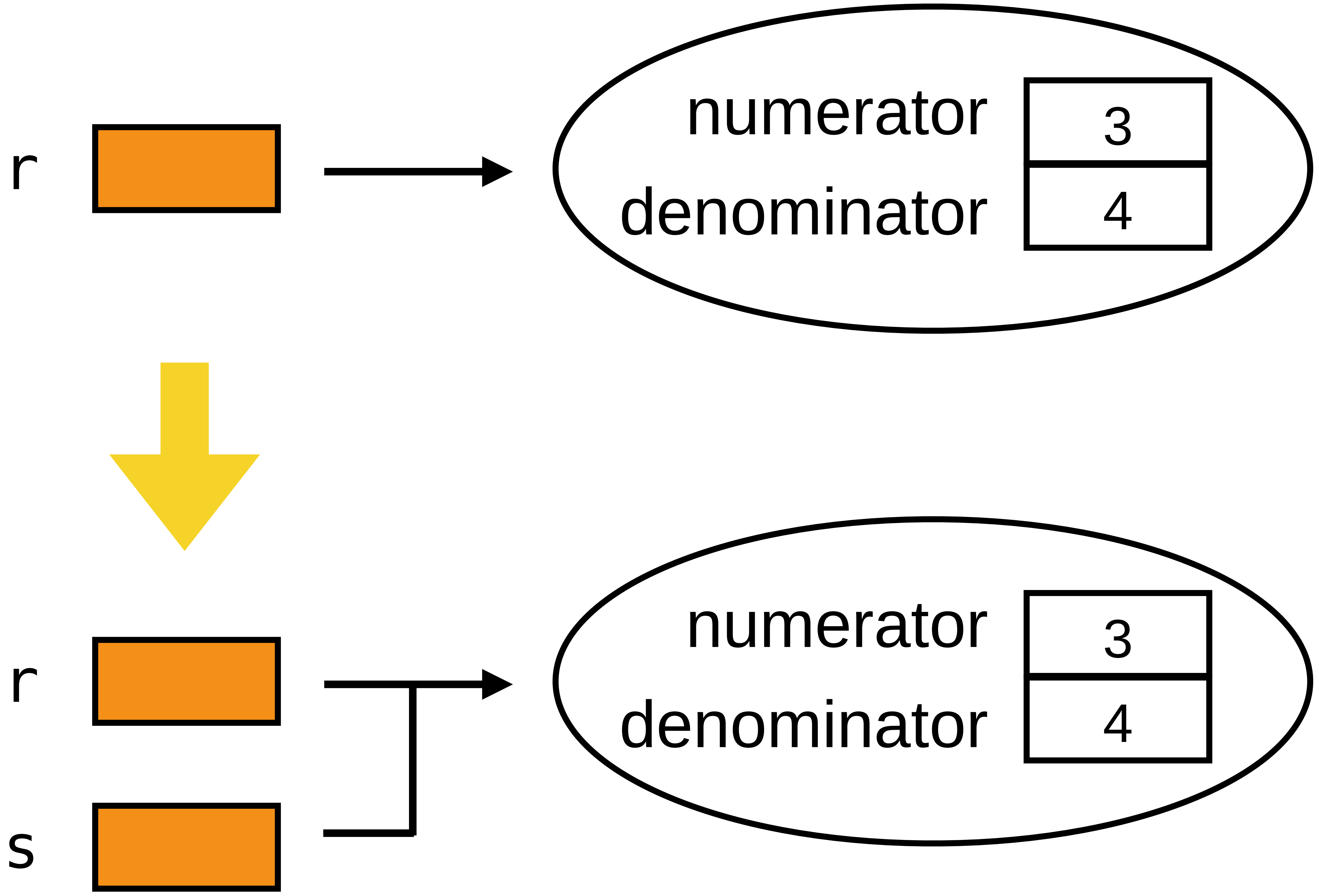
Constructors

- The constructor is a special method that reserves memory in the context of **`new`** (and can initialize attributes of the new object)
- Constructors always carry the name of their class
- A default constructor **`ClassName()`** in the class **`ClassName`** is provided automatically (if no other constructor is defined)
- Constructors can also have arguments
- If you add a custom constructor (with parameters), the default one is not automatically available anymore

Object references

- The value of a **`Rationale`** variable is a reference
- `Rationale s = r;` copies the reference from `r` to the variable `s`

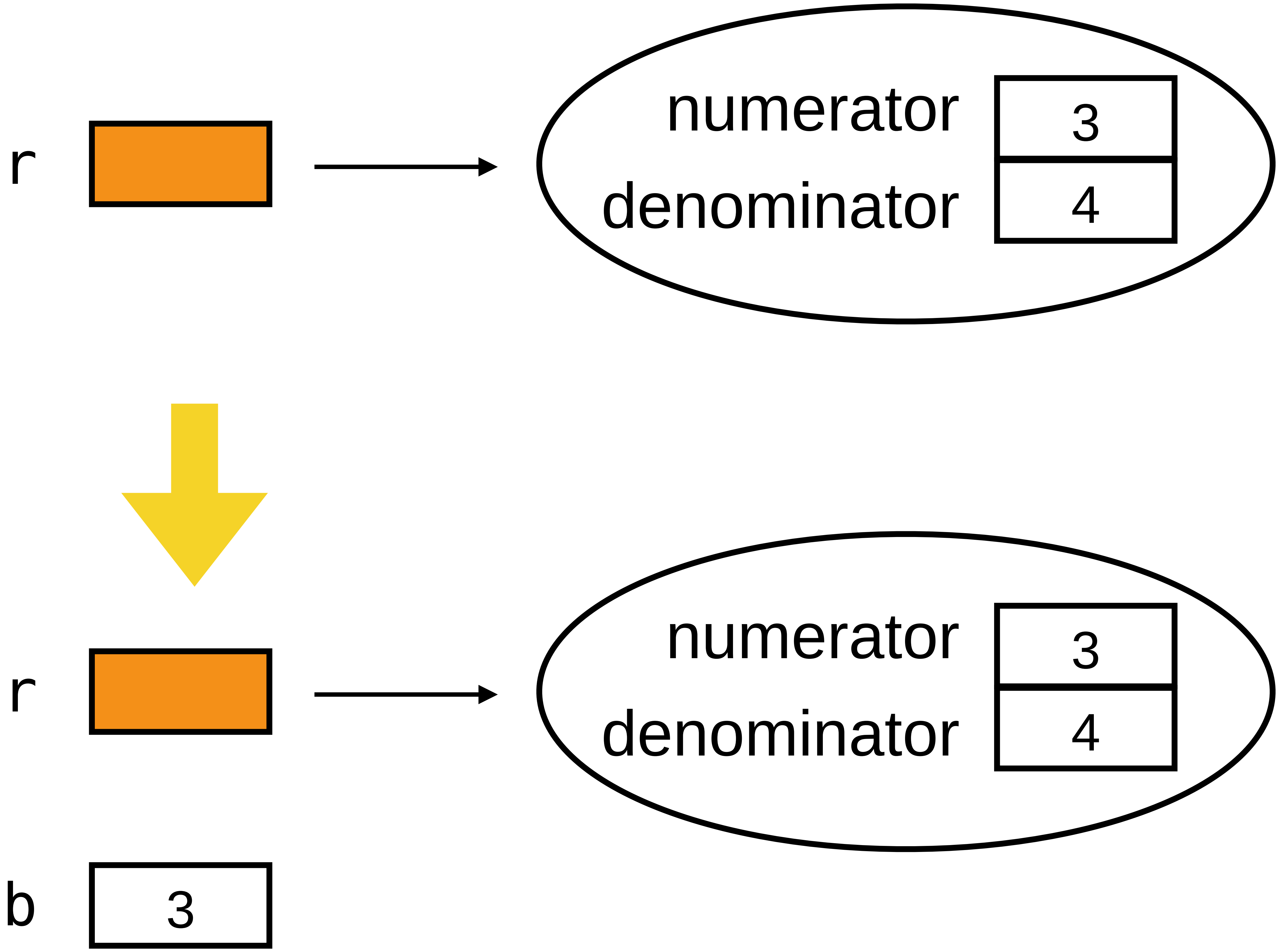
```
Rationale s = r;
```



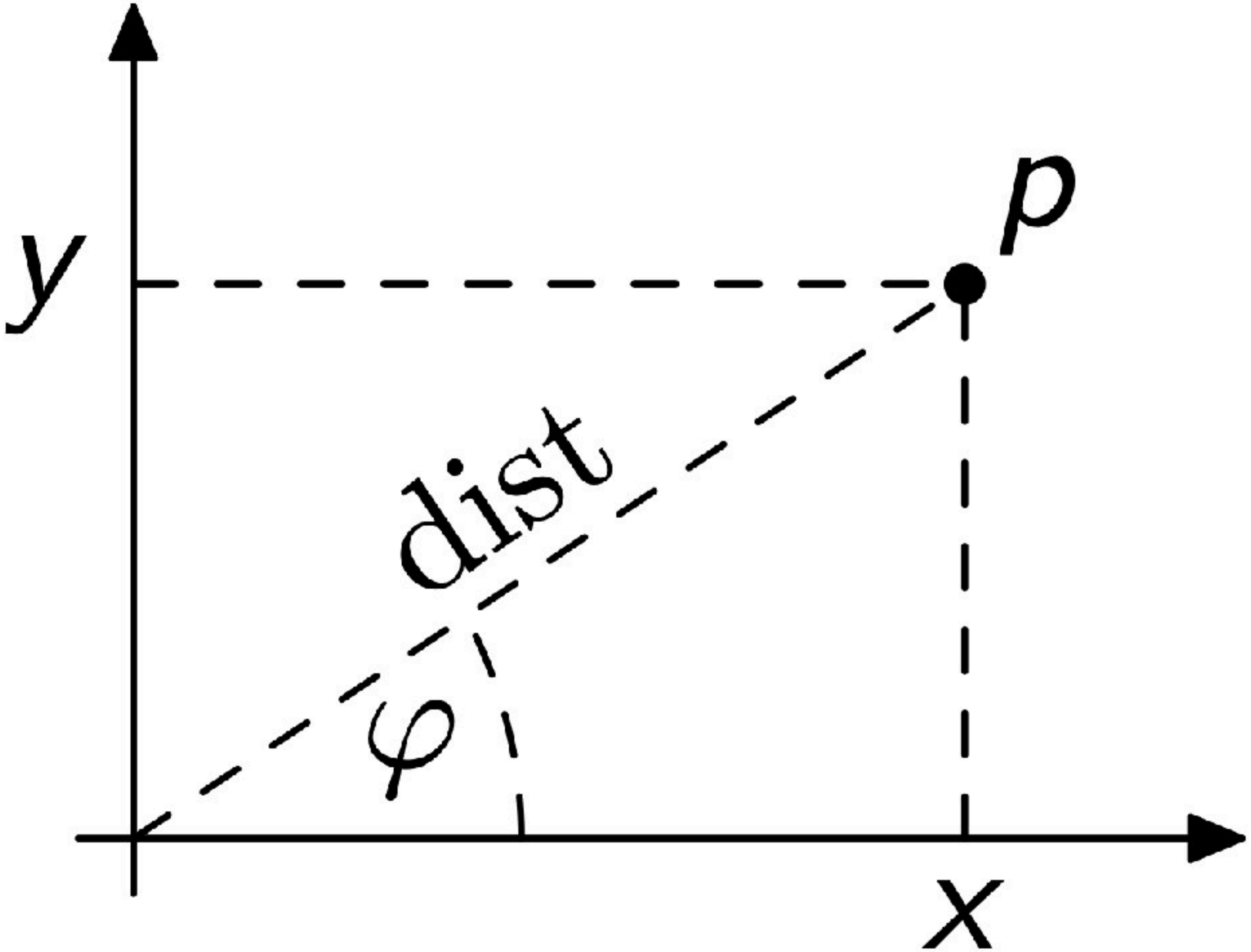
Access to attributes

- `r.numerator` returns the value of the `numerator` attribute of the object `r`

```
int b = r.numerator;
```



Example: points in R^2



Example: points in R^2

Code

```
class Point {
    double x;
    double y;

    // constructor Point() exists automatically

    // methods . . .
}
```

Create points / references

- Create points with new
  - `Point p = new Point(); // create Point p`
  - `Point q = new Point(); // create Point q`
  - `p, q` are names for references to new objects of the class **`Point`**
- A point can have multiple identifiers
  - `Point r = p; // r, p reference the same object`

Functionality of references (1)

- Consider two declarations of local variables
  - `Point p;`
  - **`int`** `n;`
- Both variables are not (yet) initialized

Question: what is their current (default) value?

- Initialize
  - `p =` **`new`** `Point();`
  - `n = 5;`
- `new Point()` creates an object of type `Point` (on the heap) and returns a reference to the object
- This reference is stored in the reference variable p
- `n = 5` stores the value 5 directly in the variable n

Functionality of references (2)

- Consider two other variable declarations

```java
Point p = new Point();
Point q = p;
```

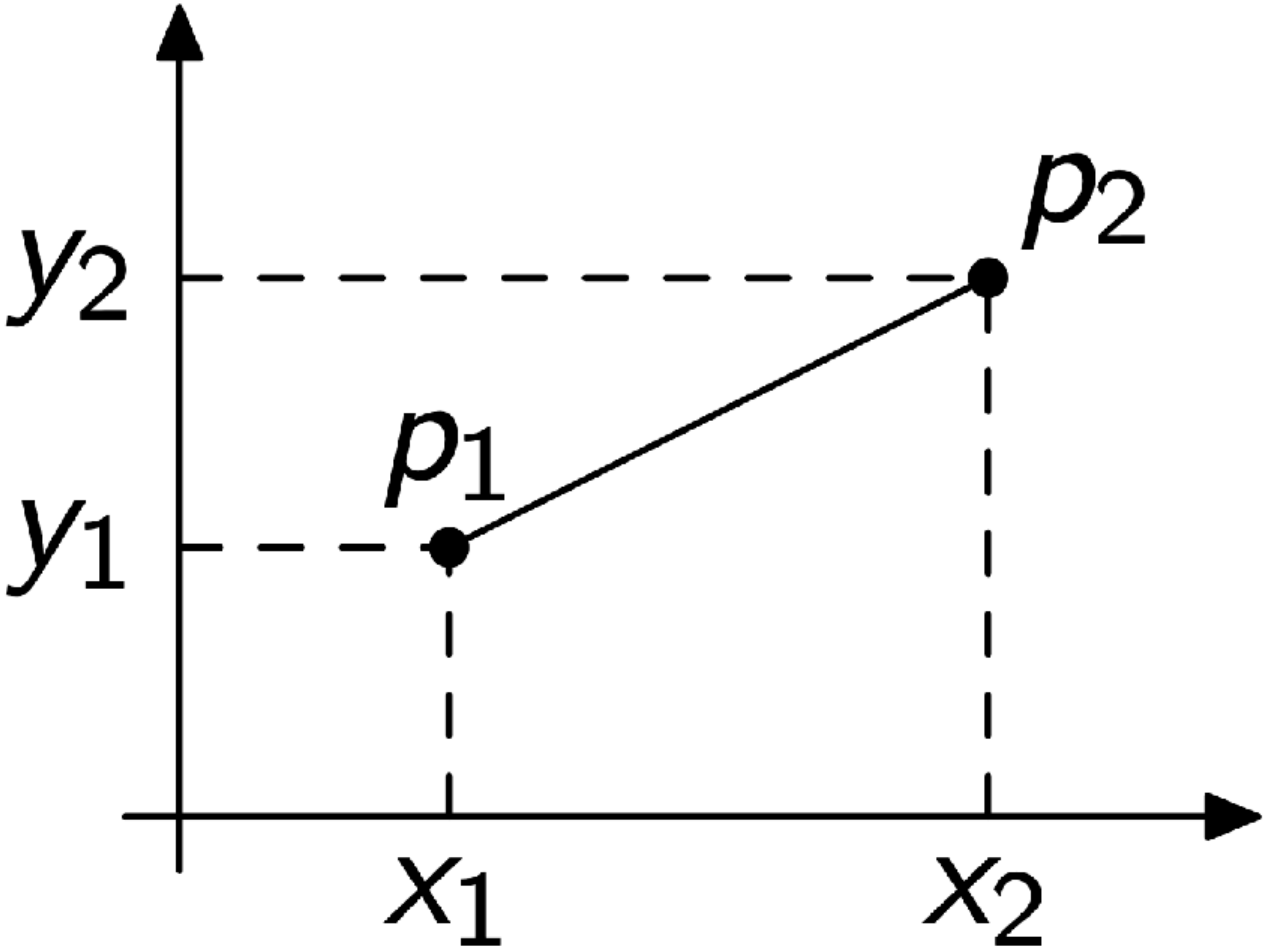
Store the **value** of p in the variable q

```java
int n = 5;
int k = n;
```

Store the **value** of n in the variable k

- In case of a reference this is the reference to the respective object
- In case of a primitive (basic) type, it is the value itself
- → p and q refer to one and the same object

Example: objects as attributes of objects: lines in R²



Example: objects as attributes of objects: lines in R^2

Code

```
class Line {
    // attributes: end points
    Point p1;
    Point p2;

    // Default constructor Line() no longer exists
    // if another constructor is defined.
    // Here: constructor explicitly with two arguments!
    Line(Point p, Point q) {
        p1 = p;
        p2 = q;
    }

    // other methods . . .
}
```

© 2023 Stephan Krusche

Example: objects as attributes of objects: lines in R^2

Code: creation of objects

```java
Point p1 = new Point();
p1.x = 2.0;
p1.y = 2.7;

Point p2 = new Point();
p2.x = 5.8;
p2.y = 16.0;

Line l = new Line(p1, p2);
```

Improved example with a constructor

```java
Point p1 = new Point(2.0, 2.7);
Point p2 = new Point(5.8, 16.0);
Line l = new Line(p1, p2);
```

```java
class Point {
    double x;
    double y;

    Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    // methods . . .
}
```

Refers to the current object

© 2023 Stephan Krusche

`null`

- The default value for all reference types is **`null`**
- If a reference variable or a reference attribute have not been initialized, then it has the value **`null`**
- If a variable has the value **`null`**, it does not refer to any object
  - Example: `Point p; // now applies: p == null`
  - Access to **`p.x`** and **`p.y`** with **`p == null`** leads to runtime error!

NullPointerException

Information hiding

- Information hiding is an important concept in computer science
  - Why are endpoints not represented by two coordinate pairs `x1, y1` and `x2, y2` in class **`Line`**?
  - Change of implementation of **`Point`** would require a change of implementation of **`Line`** (e.g. when changing to polar coordinates)

→ The implementation of **`Point`**`s` is the secret of the class **`Point`** alone

- This is the only way to limit the effects of changes locally (**principle of locality**)

Intermediate summary

- The (virtual) world consists of objects
- Objects are instances of classes, classes are blueprints / templates for objects
- Objects encapsulate data (attributes) and behavior (methods)
- Objects are accessed via references
  - References are defined by attribute declarations
  - Attributes can be objects themselves
- Objects are created / initialized by **`new`**
  - **`new`** always calls a constructor that performs initializations
- Two references can point to the same memory area (aliasing)

Break

30 min

The lecture will continue at **16:20**

Outline

- General organization of the course
- Object oriented programming
- ➡ **Basic types and their operations**
- Methods
- UML class diagrams

Basic types

- A data type (short: type) defines a set of possible values and the possible operations (methods) on these types
- All variables and attributes have a type (and methods have a return type and types for the arguments)
- In Java, classes define data types
- Java also provides eight basic types (e.g. **`int`** and **`double`**)
- Each value of a basic type requires the same amount of space to represent it in the computer
- The space is measured in bits
  (How many values can be represented with n bits?)

Integer based types

Type	Space	Smallest value	Highest value
`byte`	8	−128	127
`short`	16	−32 768	32 767
`int`	32	−2 147 483 648	2 147 483 647
`long`	64	−9 223 372 036 854 775 808	9 223 372 036 854 775 807

Using smaller types like byte or short saves space

Note: Java does **not** warn in case of overflow or underflow!

© 2023 Stephan Krusche

Overflow example

```
int x = 2147483647; // largest int value
x = x + 1;            // overflow
System.out.println(x);
```

```
prints:    -2147483648
```

Smallest **int** value

- You can assign a first value to a variable when you declare it directly (initialization)
- You can even declare it at the point where you need it for the first time

Floating point numbers

- There are two kinds of floating point numbers

Type	Space	Smallest value	Highest value	Precision
`float`	`32`	`−3.4e+38`	`3.4e+38`	6 to 7 decimal digits
`double`	`64`	`−1.7e+308`	`1.7e+308`	15 to 16 decimal digits

- Both double and float are approximate types, they are **not** precise
- Overflow/Underflow returns the values **`Infinity`** or **`−Infinity`**
- Consider the accuracy of the result to select the right type
- Floating point constants in the program are understood as `double`
- To differentiate, append the letter **`f`** (or **`F`**) or **`d`** (or **`D`**)

```
Example: 3.14f
```

Other basic types

Type	Space	Values
`boolean`	1	`true, false`
`char`	16	`all` **`unicode`** `characters`

- Unicode is a character set that includes all alphabets commonly used anywhere in the world, for example
  - The characters of our keyboard (including umlauts)
  - Chinese characters
  - Egyptian hieroglyphics
  - ...
- Char constants are written with single quotation marks: `'A', ';', '\n'`

Basic types and constructors

- Because they are used often and their implementation is efficient, no constructor has to be called
- It is sufficient, for example, to write **`int x = 2;`**
- Only one memory area is reserved (for the value), not two as in the case of reference types (reference + object)

Equality and identity

- When are two objects of a basic type "equal"?
  - For basic types the value is stored directly, without reference
- Check the equality of the value for basic types therefore with **==**
- For reference types the reference is compared with **==**

Example

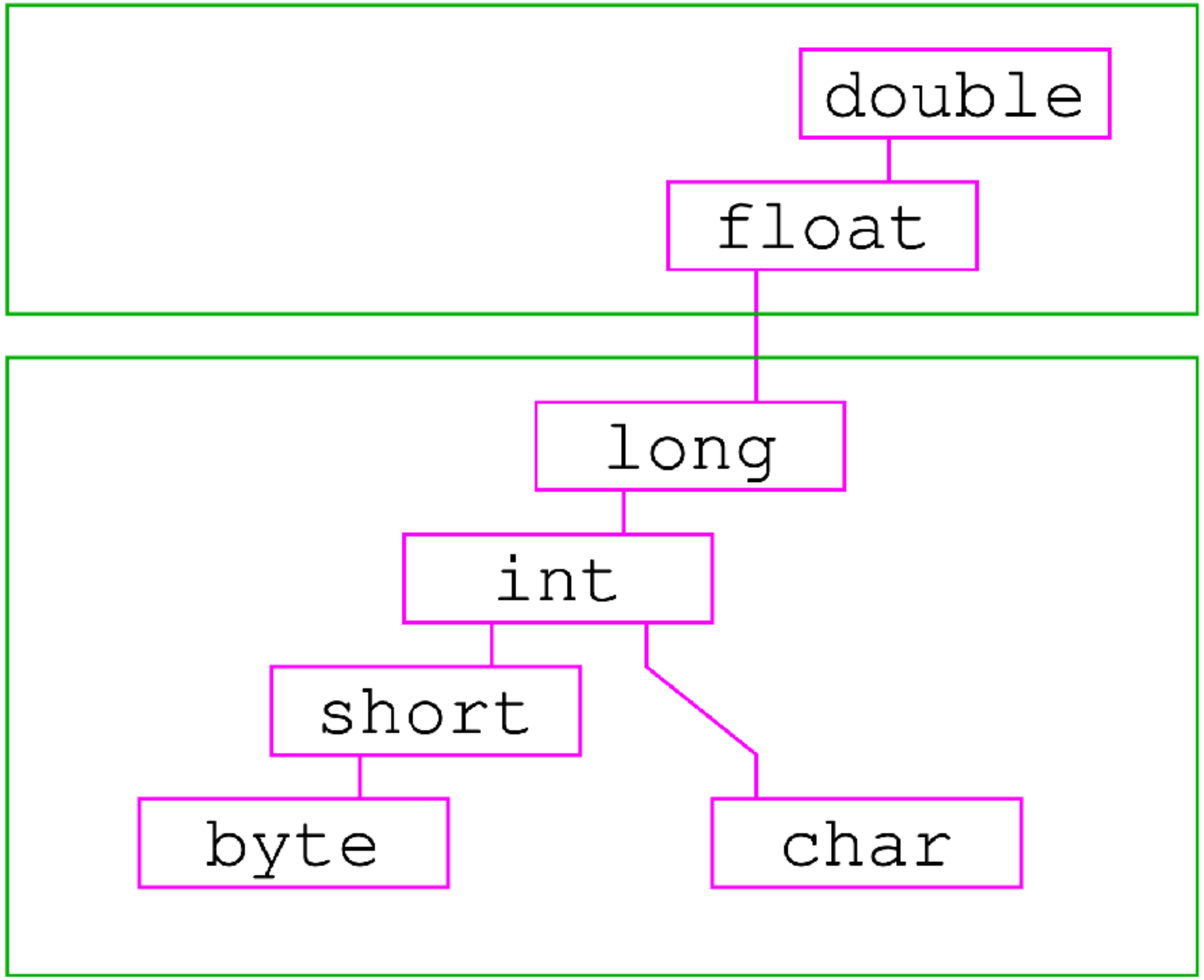
```
int i1 = 5;
int i2 = 6;
int i3 = 5;
System.out.println(i1 == i2);   // false
System.out.println(i1 == i3);   // true
```

```
Point p1 = new Point();
Point p2 = new Point();
Point p3 = p1;
System.out.println(p1 == p2);   // false
System.out.println(p1 == p3);   // true
```

Basic operations

- The operators `+, -, *, /` and `%` exist for each of the number types
- If they are applied to a pair of arguments of different types, the more specific one is converted to the more general one beforehand (implicit type cast)

Overview of basic types



Floating point numbers

Integers

Example

```
short xs = 1;
int x = 9999999;
System.out.println(x + xs);
```

Prints the **`int`** value `10000000`

```
float xs = 1.0f;
int x = 999999999;
System.out.println(x + xs);
```

Can output floating point numbers

Prints the **`float`** value `1.0E9`

Elementary operations

Precedence	Operator	Description
	Arithmetic and comparison operators	
1	+x, -x	unary plus/minus
2	x * y, x / y, x % y	multiplication, division, modulo
3	x + y, x - y	addition, subtraction
4	x < y, x <= y, x > y, x >= y	size comparisons
6	x == y, x != y	equality, inequality
	Operators on integers	
1	˜x	Bitwise complement (NOT)
	Operators on integers and truth values	
7	x & y	Bitwise AND
8	x ˆ y	Bitwise either/or (XOR)
9	x | y	Bitwise or (OR)
	Operators on truth values	
1	!x	NOT
10	x && y	Sequential AND
11	x || y	Sequential OR
	Operators on integers	
4	x << y	Left shift
4	x >> y	Right shift (sign conform)
4	x >>> y	Right shift (without sign)

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

Example

```
String s0 = "";
String s1 = "Hel";
String s2 = "lo Wo";
String s3 = "rld!";
System.out.println(s0 + s1 + s2 + s3);
```

... writes Hello World! on the output

Implicit conversion to **String**

- Every value in Java has a representation as a **String**
- If the operator "+" is applied to a value of type **String** and another value x, then x is automatically converted to its **String** representation

- This is a simple way to output float or double

- Example

```
double x = -0.55e13;
System.out.println("A floating point number: " + x);
```

Output on the console: `A floating point number: -5.5E12`

Exercise: what will the following program output?

```java
public class Test {
    public static void main(String[] args) {
        int x = 2, y = 5;

        int exp1 = (x * y / x);
        int exp2 = (x * (y / x));

        System.out.println(exp1);
        System.out.println(exp2);
    }
}
```

Solution

© 2023 Stephan Krusche

Exercise: what will the following program output?

```
public class Test {
    public static void main(String[] args) {
        int x = 10, y = 5;

        int exp1 = (y * (x / y + x / y));
        int exp2 = (y * x / y + y * x / y);

        System.out.println(exp1);
        System.out.println(exp2);
    }
}
```

Solution

Exercise: what will the following program output?

```
public class Test {
    public static void main(String[] args) {
        int x, y, z;
        x = y = z = 2;
        x += y;
        y -= z;
        z /= (x + y);
        System.out.println(x + " " + y + " " + z);
    }
}
```

Solution

Break

10 min

The lecture will continue at **17:10**

Outline

- General organization of the course
- Object oriented programming
- Basic types and their operations
- ➡ **Methods**
- UML class diagrams

Methods

Methods realize the dynamic behavior of objects and define functions

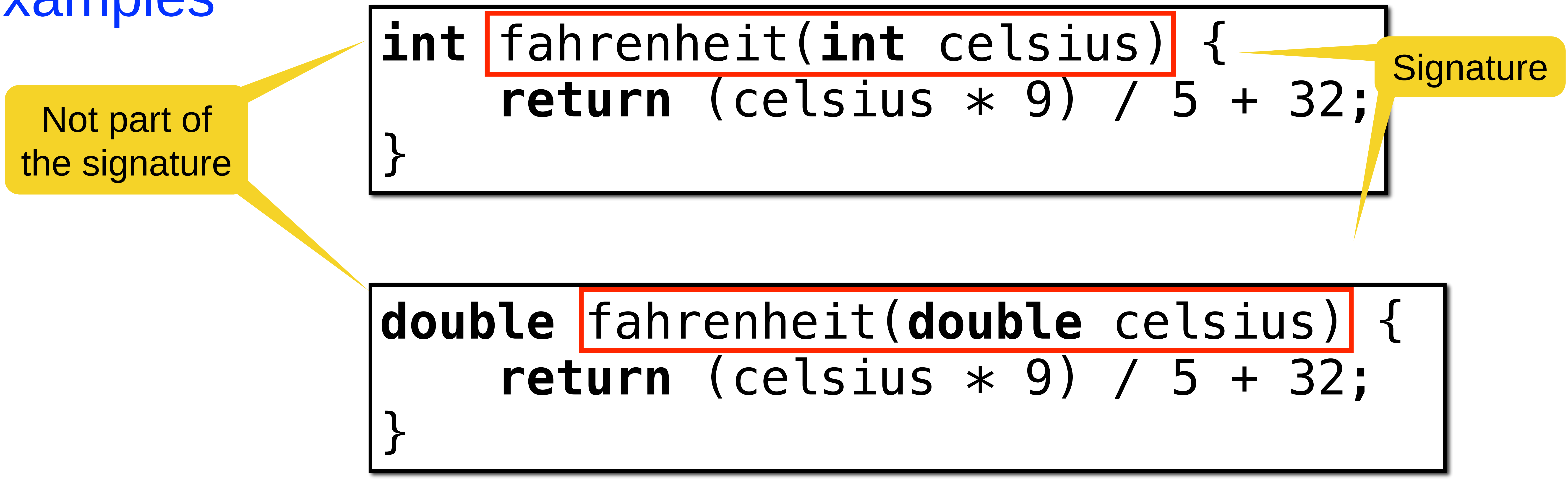
```
int average(int v1, int v2) {
    return (v1 + v2) / 2;
}
```

```
int fahrenheit(int celsius) {
    return (celsius * 9) / 5 + 32;
}
```

```
double voltage(double resistance, double current) {
    return resistance * current;
}
```

Method signature

- The interface of a mathematical function consists of the name of the function, an ordered list of (formal) parameters and their types, and the return type
- A method signature in Java consists of
  - Name of the method
  - Ordered list of (formal) parameters and their types (this determines the order and number)
  - Note: in Java, the return type is not part of the method signature
- Examples



```
int fahrenheit(int celsius) {
    return (celsius ∗ 9) / 5 + 32;
}
```

```
double fahrenheit(double celsius) {
    return (celsius ∗ 9) / 5 + 32;
}
```

Not part of the signature

Signature

Method call

- **Example**: Ohm's law

Formal parameters

```
double v(double r, double i) {
    return r * i;
}
```

```
v: voltage
i: current
r: resistance
```

- **Call:**

```
double r1 = 3.0;
double v1 = v(r1, 2.0);
```

Actual parameters

- **Syntax:** `methodname(actualparameter)`
- **Result:** An element with the return type of the method

→ What is the value of `v1`?

© 2023 Stephan Krusche

Example: cylinder volume

- Mathematical notation $volume(r, h) = r^2 * \pi * h$

- Java declaration

```
double circleArea(double radius) {
    return radius * radius * 3.1416;
}
```

You can also use **Math.PI**

```
double cylinderVolume(double height, double radius) {
    return height * circleArea(radius);
}
```

- **`circleArea(2.0)`** returns the value `12.5664`
- **`cylinderVolume(1.5, 2.0)`** evaluates the expression **`1.5 * circleArea(2.0)`** and returns the value `18.8496`

© 2023 Stephan Krusche

Constructors

- Constructors are special methods that initialize attributes after reserving memory with **new**
- Constructors do not define a **return type** (you do not return anything, because they will automatically create a new object and return it)
- Formal parameters can be used for initialization
- Example

```
class Rationale {
    double numerator, denominator;

    Rationale(double n, double d) {
        numerator = n;
        denominator = d;
    }
}
```

© 2023 Stephan Krusche

The **void** return type

- The void return type is used when a method does not return a value
- The method is then called procedure
- Example

```
void alarm(String message) {
    System.out.println("DANGER: " + message);
}
```

  - No **`return`** necessary
  - Procedures can still change the state of an object (i.e. have side effects)
  - Declaration

```
void methodName(formalParameterList) {
    // body
}
```

  - Usage

```
methodName(actualParameterList);
```

© 2023 Stephan Krusche

Current object

- Normally the method depends on the state of the current object

→ This related object must be specified when calling

- This means that within a method it must be possible to address this reference object ("itself")
- For this purpose there is the keyword **`this`**

Example: using **this** in the constructor

```
class Point {
    double x, y;
    Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

Use of object state in methods

- Example: define method **`shift`** in class **`Point`**

```java
class Point {
    double x, y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    void shift(double dx, double dy) {
        this.x = this.x + dx;
        this.y = this.y + dy;
    }
}
```

- Call

```java
Point p = new Point(3.0, 4.0);
p.shift(2.0, 4.0);
```

- Call syntax

```java
object.methodName(parameter);
```

If no object is specified → **`this`** is always used implicitly

Reference class - static methods

- A static method is independent of the state of concrete objects
- Example

```
class Weather {
    static int fahrenheit(int celsius) {
        return (celsius * 9) / 5 + 32;
    }
}
```

Question: can static methods have side effects?

- Call

```
Weather.fahrenheit(15);
```

- Syntax

```
static returnType methodName(parameter) {
    // method body
}
```

- Call syntax

```
ClassName.methodName(parameters)
```

© 2023 Stephan Krusche

The main method

- One static method is special: the **`main()`** method

```
class K {
    public static void main(String[] args) {
        //...
    }
}
```

Field / array

- **`main(p1, ..., pn)`** in class K will be called when you execute `java K` after compiling K
- The arguments **`p1, ..., pn`** are stored in a field called **`args`**

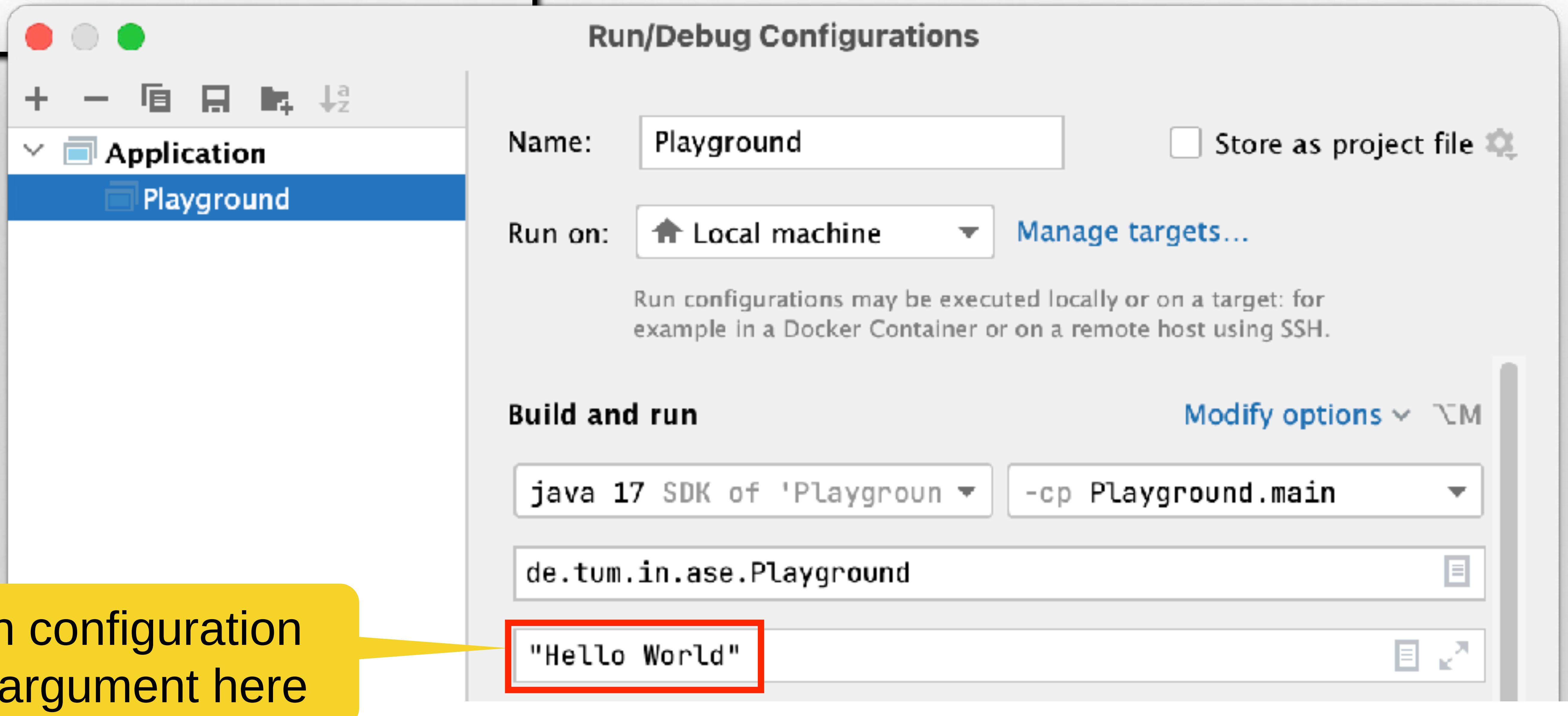
Covered later

Example

Playground.java

```
public class Playground {
    public static void main(String[] args) {
        if (args.length > 0) {
            System.out.println("First argument: " + args[0]);
        }
        else {
            System.out.println("No arguments provided");
        }
    }
}
```

Run the application with an argument



Name: Playground

□ Store as project file

Run on: Local machine

Manage targets...

Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.

Build and run

Modify options ⌥M

java 17 SDK of 'Playgroun

-cp Playground.main

de.tum.in.ase.Playground

"Hello World"

Open the run configuration and add the argument here

Example: Point

Move points

```
class Point {

    double x, y;

    void shift(double dx, double dy) {
        this.x = this.x + dx;
        this.y = this.y + dy;
    }
}
```

Current coordinates

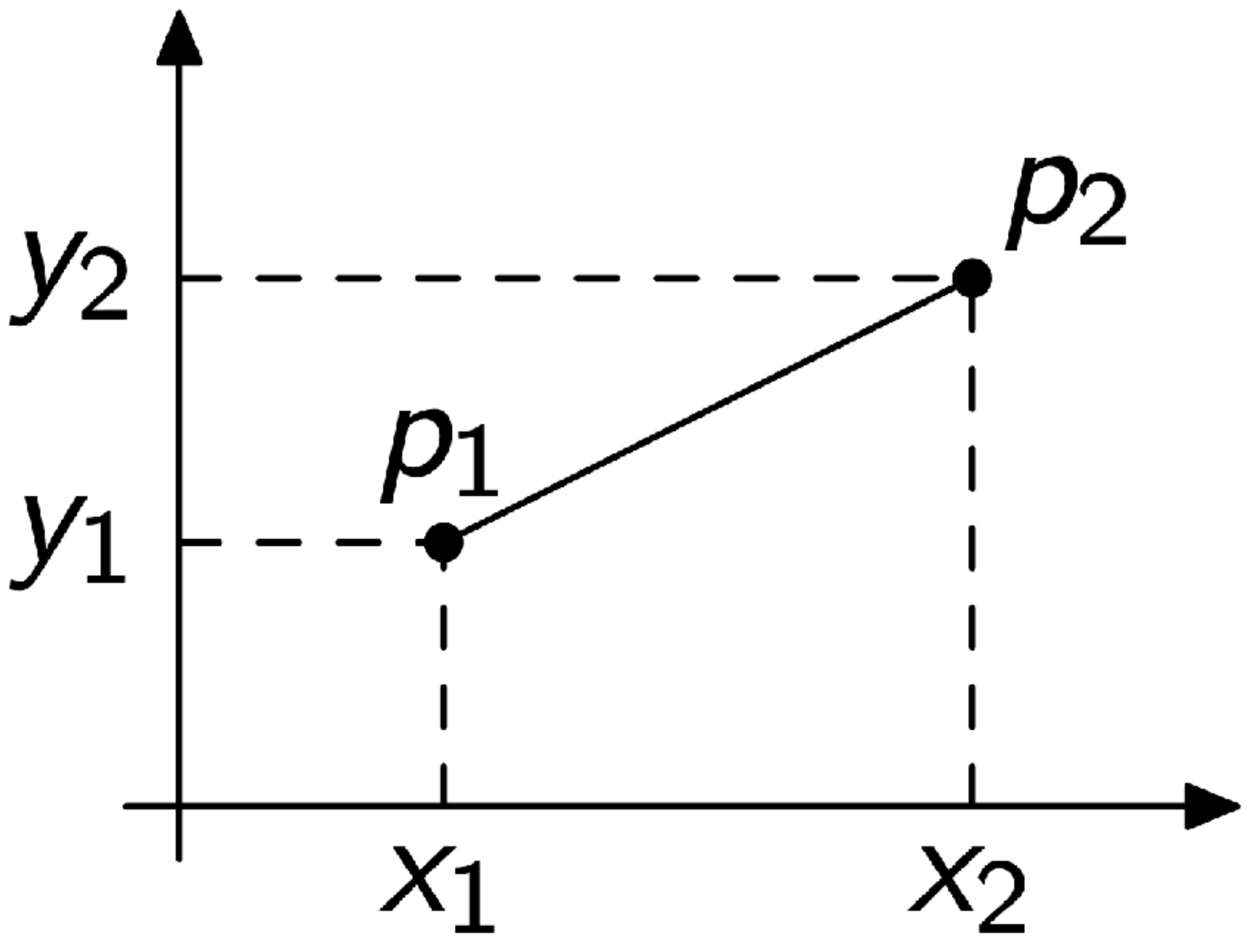
```
double getX() {
    return this.x;
}
```

```
double getY() {
    return this.y;
}
```

Move lines

Example: lines in R^2

Concept



```java
class Line {
    Point p1;
    Point p2;

    Line(Point p1, Point p2) {
        this.p1 = p1;
        this.p2 = p2;
    }

    void shift(double dx, double dy) {
        this.p1.shift(dx, dy);
        this.p2.shift(dx, dy);
    }
}
```

References and method calls

- In Java, only call-by-value is used for parameter passing in method calls

- Definition of call-by-value: when calling methods, all statements in the parameters are first evaluated completely
  - Then, the values of those statements are passed as a copy to the called method

- **Note:** values of non-primitive types (i.e. objects) are references

  → A copy of the reference is passed
  - This can be used to still change the state of the referenced object

Local variables (1)

- Serve as auxiliary memory for intermediate results in methods
- Are stored on the so-called runtime stack
- Exist only as long as the method is processed

- Example (calculation of the area of a circle)

```
double circleArea(double radius) {
    double radiusSquare = radius * radius;
    return radiusSquare * 3.1416;
}
```

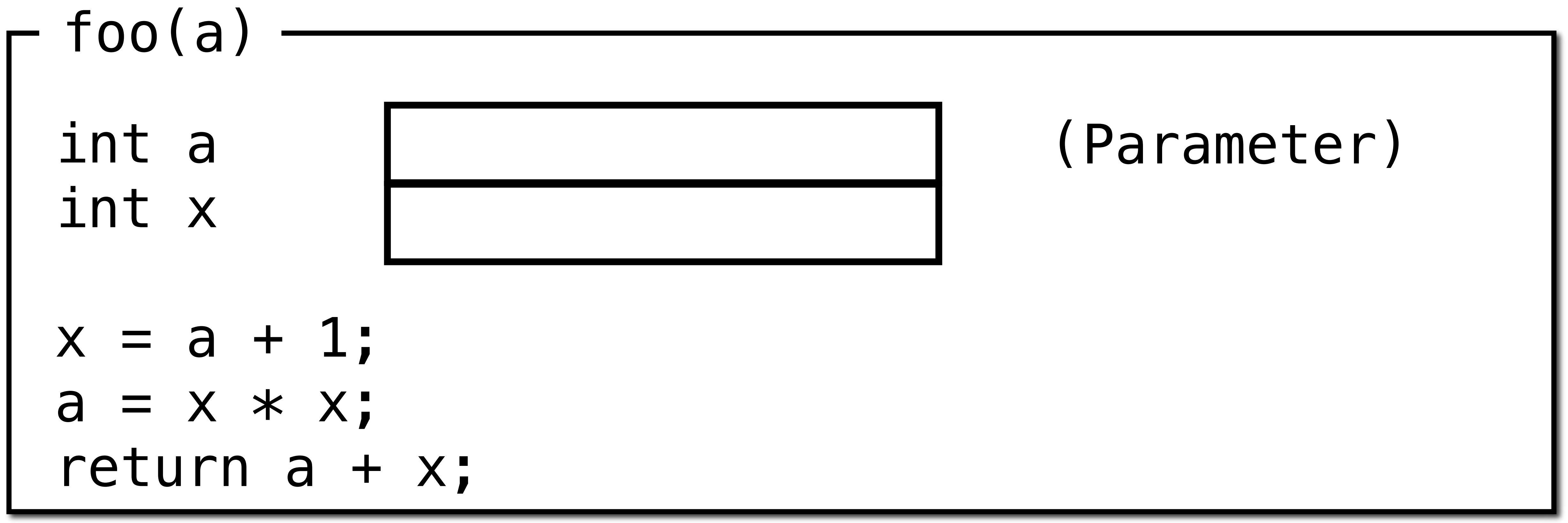
© 2023 Stephan Krusche

Local variables (2)

- In Java, **parameters** are local variables that are initialized when the method is called (call-by-value)
- Example

```
int foo(int a) {
    int x = a + 1;
    a = x * x;        // bad programming style
    return a + x;
}
```

You should **not** change input parameters in a method



```
foo(a)

int a            (Parameter)
int x

x = a + 1;
a = x * x;
return a + x;
```

Attention: simple argument types **(`int, float, ...`)** are handled differently than complex types **(`Book, Author, ...`)**

Local variables - differences to attributes

	Local variable	Attribute
Declaration	Within methods	Outside of methods
Lifetime	Method call	Lifetime of the associated object
Accessible	Only within a method	For all methods of the class
Purpose	Cache for values	State of the object

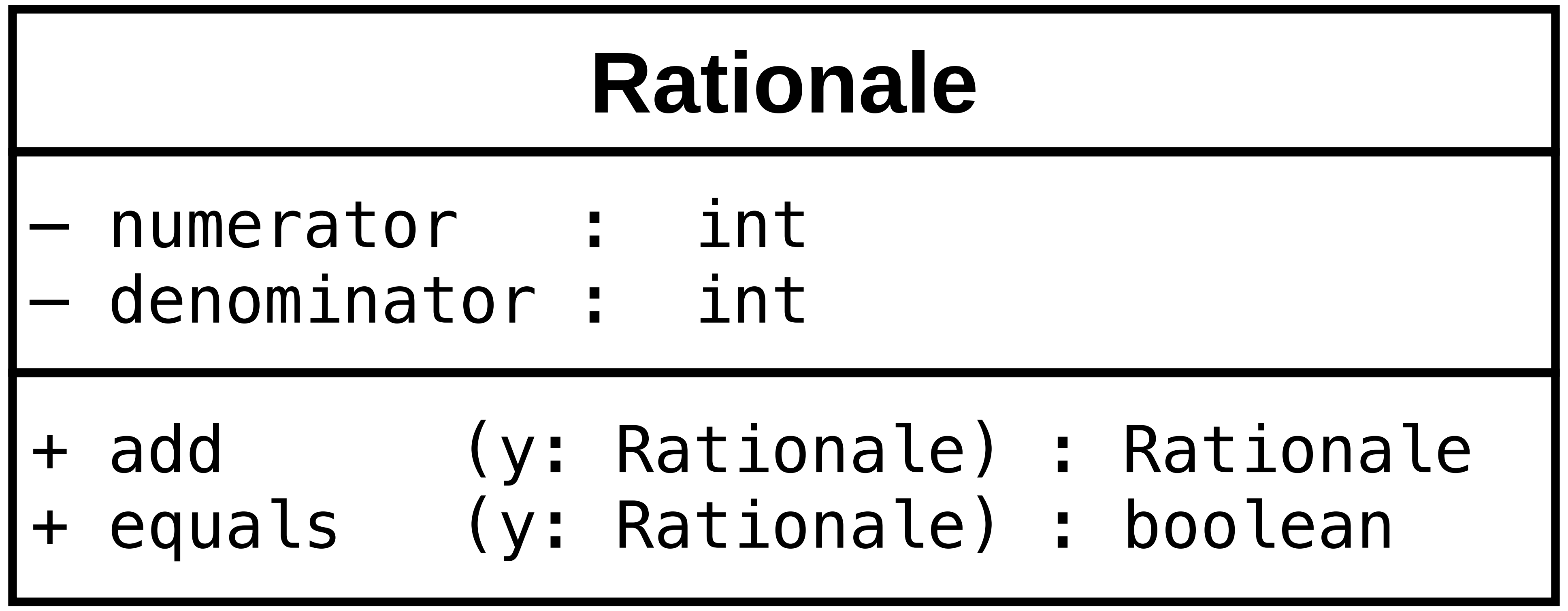
Outline

- General organization of the course
- Object oriented programming
- Basic types and their operations
- Methods

➡ **UML class diagrams**

Class diagrams: classes

- A graphical visualization of the **`Rationale`** class, considering only the essential functionality, might look like this



Rationale

```
– numerator   :  int
– denominator :  int
```

```
+ add      (y: Rationale) : Rationale
+ equals   (y: Rationale) : boolean
```

Class diagrams: discussion and outlook

- Such diagrams are provided by UML (Unified Modeling Language), to design software systems

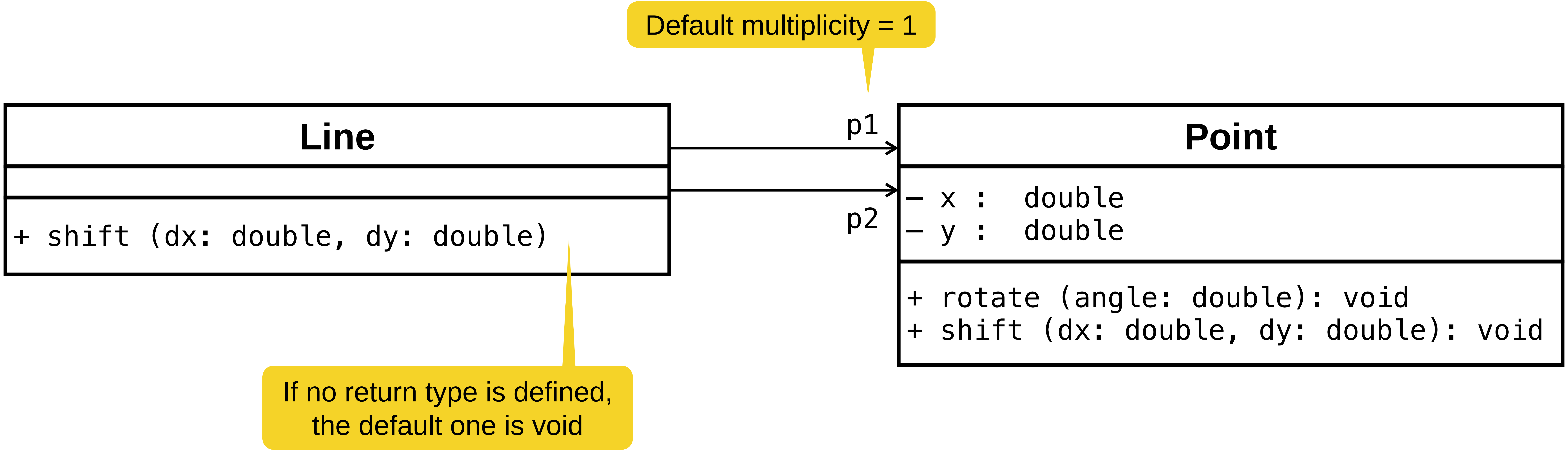
More on UML in the course **Introduction to Software Engineering**

- For a single class, such a diagram is not really worthwhile
- However, if a system consists of many classes, it can be used to illustrate the relationships between different classes
- UML was not developed specifically for Java
  - Types are notated differently
  - Also, some ideas cannot be modeled at all or only poorly

© 2023 Stephan Krusche

Class diagrams: associations

- Reference attributes are often noted as references (lines between classes)
- In the example below, a line is connected to two points p1 and p2


Default multiplicity = 1
Line
+ shift (dx: double, dy: double)
p1
p2
Point
– x : double
– y : double
+ rotate (angle: double): void
+ shift (dx: double, dy: double): void
If no return type is defined, the default one is void

© 2023 Stephan Krusche

Next steps

- Visit a tutor group
  - T01E01 - Hello World
  - T01E02 - Calculator
- Solve the homework independently
  - H01E01 - TUM Module Quest
  - H01E02 - Lectures
- Read the following articles to find out more about object oriented programming in Java
  - https://www.w3schools.com/java/java_oop.asp
  - https://www.w3schools.com/java/java_methods.asp

Click on **Next** several times

→ Due until **Wednesday, October 25, 13:00**

© 2023 Stephan Krusche

Summary

- **Object oriented programming** is the elementary thinking in Java and many other programming languages
  - **Attributes** allow to model the state of objects
  - **Constructors** initialize objects
  - **Methods** implement the logic (behavior) of objects
- Eight **basic data types**: `boolean, byte, short, int, long, float, double, char`
- **Methods** determine the behavior of objects
  - Static methods belong to classes
- **UML class diagrams** allow to visualize the structure and relationships of classes

References

- https://www.javatpoint.com/difference-between-object-and-class
- https://www.javatpoint.com/object-and-class-in-java
- https://www.javatpoint.com/method-in-java
- https://www.javatpoint.com/java-constructor
- https://www.javatpoint.com/static-keyword-in-java
- https://www.javatpoint.com/this-keyword
- https://www.javatpoint.com/java-oops-concepts
- https://www.geeksforgeeks.org/difference-between-class-and-object