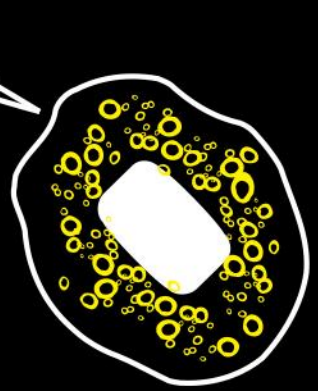
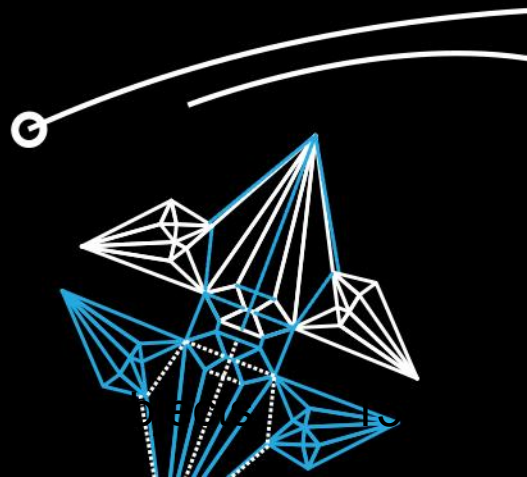
UNIVERSITY OF TWENTE.



Module 2 Programming Lectorial 1.1

Lecturer: Tom van Dijk



THREE PARTS

Introduction to Programming (3 EC)

- Weeks 1 - 3
- Exam in Week 7 (resit in Q3)

Advanced Programming (3 EC)

- Weeks 4 - 7
- Exam in Week 10 (resit in Q3)

Programming Project (3 EC)

- Weeks 8 - 10
- Sign-offs in Week 8 and Week 10

INTRODUCTION TO PROGRAMMING

- Procedural programming (Week 1)
  - Variables, methods
  - Arrays
  - Recursion
- Object-oriented programming (Weeks 2 and 3)
  - Encapsulation, abstraction, inheritance, polymorphism
- Documenting (Javadoc), debugging, and testing

ADVANCED PROGRAMMING

• Exceptions	Week 4
• Java Collections (Set, List, Map)	Week 4
• Software specification (JML preconditions and postconditions)	Week 4
• Streams and files	Week 4/5
• Simple game AI	Week 5
• Security	Week 5/7
• Design patterns (Strategy, Listener, MVC)	Week 5
• Basics of concurrency (race conditions, wait-notify mechanism)	Week 6
• Networking	Week 7

PROGRAMMING PROJECT

- Game reveal in week 6 (before the winter holiday)

- Expected to start reading the description before week 7
  - General description already available on Canvas
  - Week 7 has a Skills assignment on planning for the project

- Starts with initial software design (4 diagrams)
- Includes individual code deep-dive sign-off
- Ends with a grand AI tournament

SOME THINGS TO KNOW

- Mandatory feedback moments (sign-off points)
- Focus: how you approached the problem, not just whether the solution is good
- Exams: pure Remindo, **no IDE support**, just with syntax highlighting!!

SOME SUGGESTIONS TO FOLLOW

- Actually study (read/watch) the materials
- Do the practical exercises and sign-off on time!
  - So you can start on the project asap…
- Go to the diagnostic exam!!
- Work ***with*** your partner, not just ***near*** your partner
  - Preferably on one computer and mentally engaged (pair programming)
  - Also for the project!!

PAIR PROGRAMMING

- How to work together? (synergy)

- Worst case: you see your partner work and you only understand some parts
- Or: it looks sensible but you're not sure if you could do it

- By working together you learn more than doing it alone
- You learn from each other and with each other

PAIR PROGRAMMING

- Driver and navigator
  - Driver writes code
  - Navigator checks and considers the greater view
- Strong-style pair programming
  - Navigator instructs
  - Drive executes
  - Constant discussion
- It is intensive: take breaks

SOME MATERIALS TO STUDY

- Video topics (on Canvas)
- The book Introduction to Programming Using Java by David J. Eck
- The M2 manual (on Canvas)
- The lectures are complementarty: context, background, depth, Q&A

SOME MATERIALS TO STUDY

The book Introduction to Programming Using Java by David J. Eck

- Week 1: Chapter 1 – 4
- Week 2: Chapter 5.1 – 5.4
- Week 3: Chapter 5.5 – 5.8
- Week 4: Chapter 7.1 – 7.3, 8.1 – 8.3, 9.1 – 9.2, 10.1 – 10.4, 11.1 – 11.2
- Week 6: Chapter 12.1 – 12.3
- Week 7: Chapter 11.4

SOME CHALLENGES FOR BORED FOLKS

What if I already know how to program??

- Advent of Code challenges (Dec 1 – Dec 12, join the M2 leaderboard!)
- Optional challenges in the project (chatting, encryption, GUI, AI, etc)
- Still extra time? Try learning Kotlin, or Rust. Or learn more about advanced concurrency data structures in Java. Or about new Java language features.

Do you want to “prove you can do the basics” or “have fun learning new things”

WHAT ABOUT GENERATIVE AI?

- ChatGPT, Copilot, Grok, Cursor, Gemini, JetBrains AI Assistant…….

- **Simply not allowed!**

- Obviously not for doing practical exercises
- Also not for summarizing materials
- Also not for debugging problems
- Also not for general questions
- Also not for the project
- Also not for exam preparation
- Also not to argue with a TA

WHAT ABOUT GENERATIVE AI?

These tools are very attractive, yes?

- Needs very low mental effort to use *
- No need to be polite or phrase a nice question, always there for you <3
- It can do almost all the simple uncomplicated practice exercises *
- Potentially improves your work resulting in higher grades *
- Good vibes when using it *

WHAT ABOUT GENERATIVE AI?

What can we expect from a university graduate?

- The ability to read a dense text
- The ability to write at an academic level
- The ability to study a complex topic
- Critical thinking

- Only use Generative AI if you can distinguish a **good** answer from a **bad** answer
  - You can't use Generative AI critically when you're still learning!

WHAT ABOUT GENERATIVE AI?

- Generative AI create homogeneity and erases individuality / diversity
- Generative AI creates an illusion of increased productivity
- Generative AI erases your ability to reason and your critical thinking
- Generative AI confidentially tells half-truths and full lies (bias, hallucination)
- Generated code is typically of low quality and full of subtle errors

Any questions so far?

PROGRAMMING OVERVIEW

Week 1 Procedural programming Debugging	**Week 2** Classes and Objects Documenting and Testing	**Week 3** Interfaces and Inheritance Subtyping
Week 4 JML Specification Arrays and List/Set/Map Streams and Files Exceptions	**Week 5** Design Patterns User Interfaces	**Week 6** Concurrency Project kick-off
Week 7 Basic Networking Defensive programming Security	**Week 8/9** Project	**Week 10** Project Exam

Talking point

What is the difference between a programmer and a software engineer?

Talking point

What is programming?

What is programming?

• Precise reasoning	Details matter, computers don’t guess your intent
• Computational thinking	Breaking problems into smaller parts
• Communication	Code is read more by people than by computers
• Engineering	Building large complex systems ***in a systematic and reliable way***
• Learning	For deep understanding of concepts
• Creativity	Writing a program is an act of creation.
• Beauty	There are many elegant solutions
• Not just coding!	Code expresses ideas, theory, intent, communication

PROGRAMMING

A good program is…

- Functional
- Reliable: extensively tested with automated tests, does not crash
- Documented (in code: Javadoc, comments; in technical documentation)
- Formally specified/verified (especially for safety critical systems)
- Designed for change
  - new versions, changed requirements, bug fixing, etc.
  - change propagates throughout the code

PROGRAMMING

How to write programs?

- Computers are stupid: they do what they are told to do
- Programming is difficult!! It is a learned skill.

PROGRAMMING

How to write programs?

- Simplify
  - break down into step by step (decomposition)
  - write comments, then write code
  - talk to your rubber ducky
  - println statements and unit tests!
  - debugging!! stepping through, setting breakpoints, etc.

PROGRAMMING

How to write programs?

- Abstraction and adding structure
  - Work on small parts of the software system
  - Don’t need to think about all the other parts of the system

- This week: variables and methods
  - Later weeks: further levels of abstraction: classes, class hierarchy, interfaces
  - Even more levels of abstraction: design patterns, architecture, libraries

JAVA

- One of the most popular object-oriented programming languages:
  - TIOBE index 2025: 1) Python 2) C 3) C++ 4) Java 5) C#
  - IEEE Spectrum 2025: 1) Python 2) Java 3) C++ 4) SQL 5) C#
- Same style as C/C++
- Object-oriented, strongly typed
- Platform independent via the Java Virtual Machine
- First appeared in 1995

- See video topic on differences Python and Java

VARIABLES

Oldest programs: just assembly and *one big block of memory*, no explicit structure

Variables and types structure memory

```
pushq   %rbp
movq    %rsp, %rbp
pushq   %r12
pushq   %rbx
subq    $32, %rsp
movl    %edi, -20(%rbp)
movq    %rsi, -32(%rbp)
movq    %rdx, -40(%rbp)
movq    -40(%rbp), %rdx
movq    -32(%rbp), %rcx
movl    -20(%rbp), %eax
movq    %rcx, %rsi
movl    %eax, %edi
call    _M2_init
movq    -40(%rbp), %rdx
movq    -32(%rbp), %rcx
movl    -20(%rbp), %eax
movq    %rcx, %rsi
movl    %eax, %edi
call    _M2_fini
movl    $0, %eax
```

BASIC (PRIMITIVE) TYPES IN JAVA

Integer types:

• byte	8 bits	-128 to 127
• short	16 bits	-32,768 to 32,767
• int *	32 bits	-2,147,483,648 to 2,147,483,647
• long	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Floating-point types:

• float	32 bits	7 significant digits, E-48 to E+38
• double *	64 bits	15 significant digits, E-324 to E+308

Other types:

• boolean	1 bit	only values **true** and **false**
• char	16 bits	single Unicode character

ARRAYS

Array: a fixed-length sequence of elements, all of the same type, accessible by index

```
int numbers = new int[5];
numbers[0] = 19;
numbers[4] = 31;
System.out.println("the fourth number is " + numbers[3]);
System.out.println("there are " + numbers.length + " numbers");
```

STRUCTURE IN CODE

- Simplest program:
  - 1 main entry point (`public static void main` in Java)
  - 1 list of instructions, one after another, from top to bottom

- Control / code flow:
  - jumps: **goto** and **jump** (not in Java, but in C, C++, BASIC, Assembly, etc)
  - conditionals: **if** and **else**, **switch**
  - loops: **for**, **while**, **do-while**, **break**, **continue**

STRUCTURE IN CODE

Variables and types structure memory

Procedures structure code

PROCEDURES

- Called methods in Java
- A method has a name, parameters, and a return type
- Method signature: method name + parameter type list
- Method overloading: same name, different parameters

PROCEDURES

Example of method overloading: compute the average of several values

```
double average(int a, int b) {
     return (a + b) / 2.0;
}

double average(int a, int b, int c) {
     return (a + b + c) / 3.0;
}

System.out.println(average(4, 8));
System.out.println(average(4, 8, 10));
```

WHY PROCEDURES?

- Avoid repetition (allow reuse of code)
- Improve readability (smaller methods, lower complexity)
- Easier to maintain (designed for change)
- Easier to test and debug (more reliable)

RECURSIVE PROCEDURES

- Special kind of method: recursive methods
- A recursive method calls itself
- Used to compute something by first computing something smaller
- For example: **mergesort**:
  - Split problem into two smaller problems
  - Call **mergesort** on the smaller problems
  - Combine the two solutions

OVERVIEW OF WEEK 1 EXERCISES

1 – 6	Introduction to IntelliJ and debugging
7 – 8	Using Math functions
9 – 11	Using control structures: if, else, for, while
12 – 14	Methods and Javadoc
15 – 17	Strings and arrays
18 – 20	More arrays
21 – 23	Recursion