

A background image showing several cyclists in motion on a road, with a focus on a cyclist in the foreground wearing a red jersey and a white helmet. The image is slightly blurred to convey a sense of speed.

Java network concurrency

<https://github.com/heig-vd-dai-course>

[Web](#) • [PDF](#)

L. Delafontaine and H. Louis, with the help of GitHub Copilot.

Based on the original course by O. Liehti and J. Ehrensberger.

This work is licensed under the [CC BY-SA 4.0](#) license.

Objectives

- Learn what is concurrency
- Learn the different ways to handle multiple clients at the same time:
 - Multi-processing
 - Multi-threading
 - Asynchronous programming
- Learn how to implement and manage concurrency in Java network applications



Disclaimer

More details for this section in the [course material](#). You can find other resources and alternatives as well.

Disclaimer

- **This is not a course on concurrency**
- Many many things are not covered
- Focus on concurrency for network applications
- Other ways and mechanisms to handle concurrency will be covered in other courses



Explore the code examples

More details for this section in the [course material](#). You can find other resources and alternatives as well.

Explore the code examples

Individually, or in pair/group, **take 10 minutes to explore and discuss the code examples.**

Answer the questions available in the course material:

- How do the code examples work?
- What are the main takeaways of the code examples?
- What are the main differences between the code examples?

If needed, use the theoretical content to help you.

Concurrency: an introduction

More details for this section in the [course material](#). You can find other resources and alternatives as well.

Concurrency: an introduction

- Ability to handle multiple tasks at the same time
- Differs from parallelism, the ability to execute multiple tasks simultaneously
- Before we dive into concurrency, let's take a step back and remember what a processor is



What is a processor?

- Piece of hardware that executes millions of instructions every second
- Gives the illusion of executing multiple tasks at the same time
- However, it can execute only one instruction at a time
- When managing multiple tasks, the processor switches between tasks



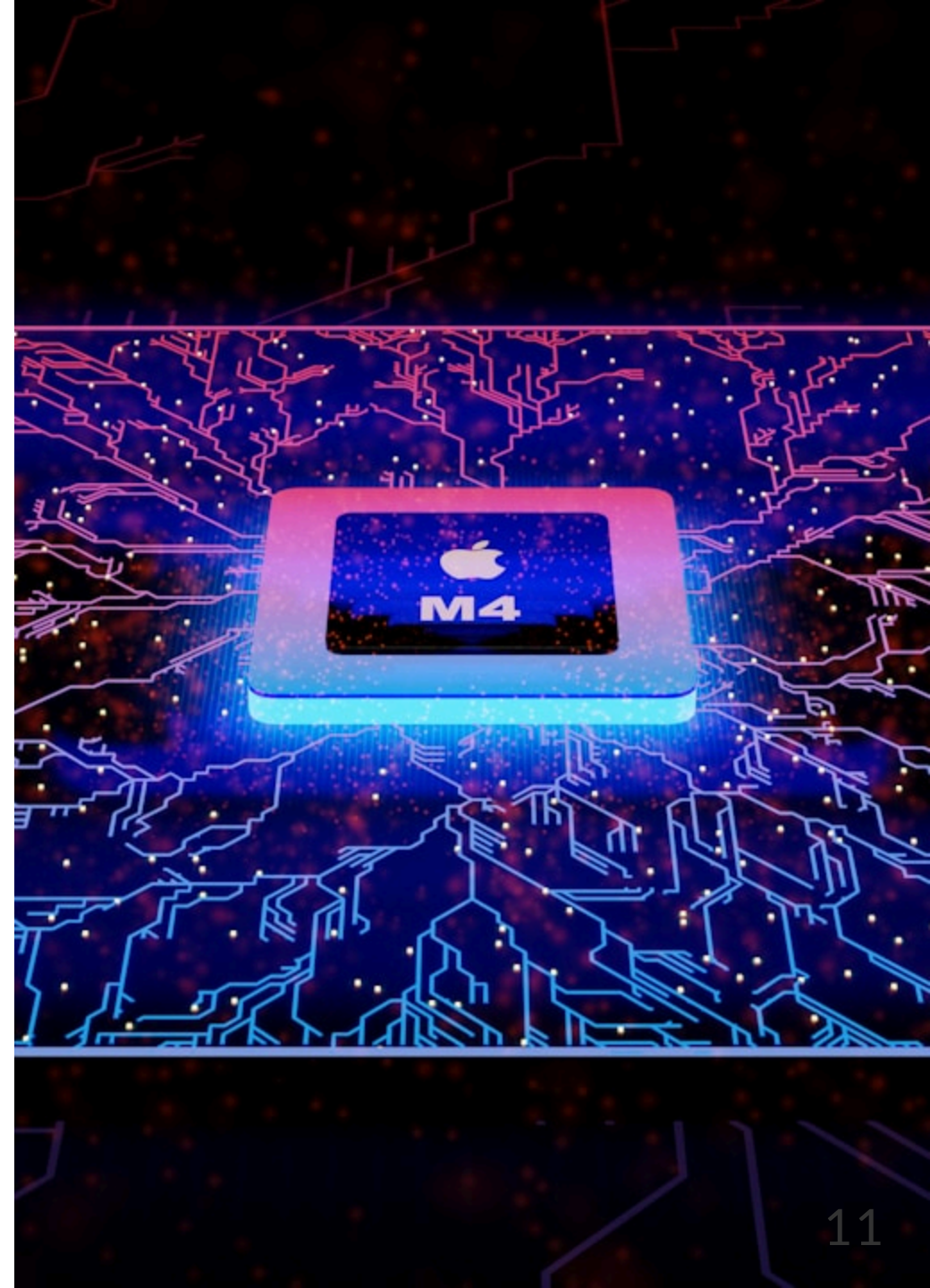
What is a core?

- A physical unit inside a processor that executes instructions (always one at a time)
- A processor can have multiple cores, called multi-core processors
- Cores can execute multiple instructions at the same time (parallelism) - not covered in this course



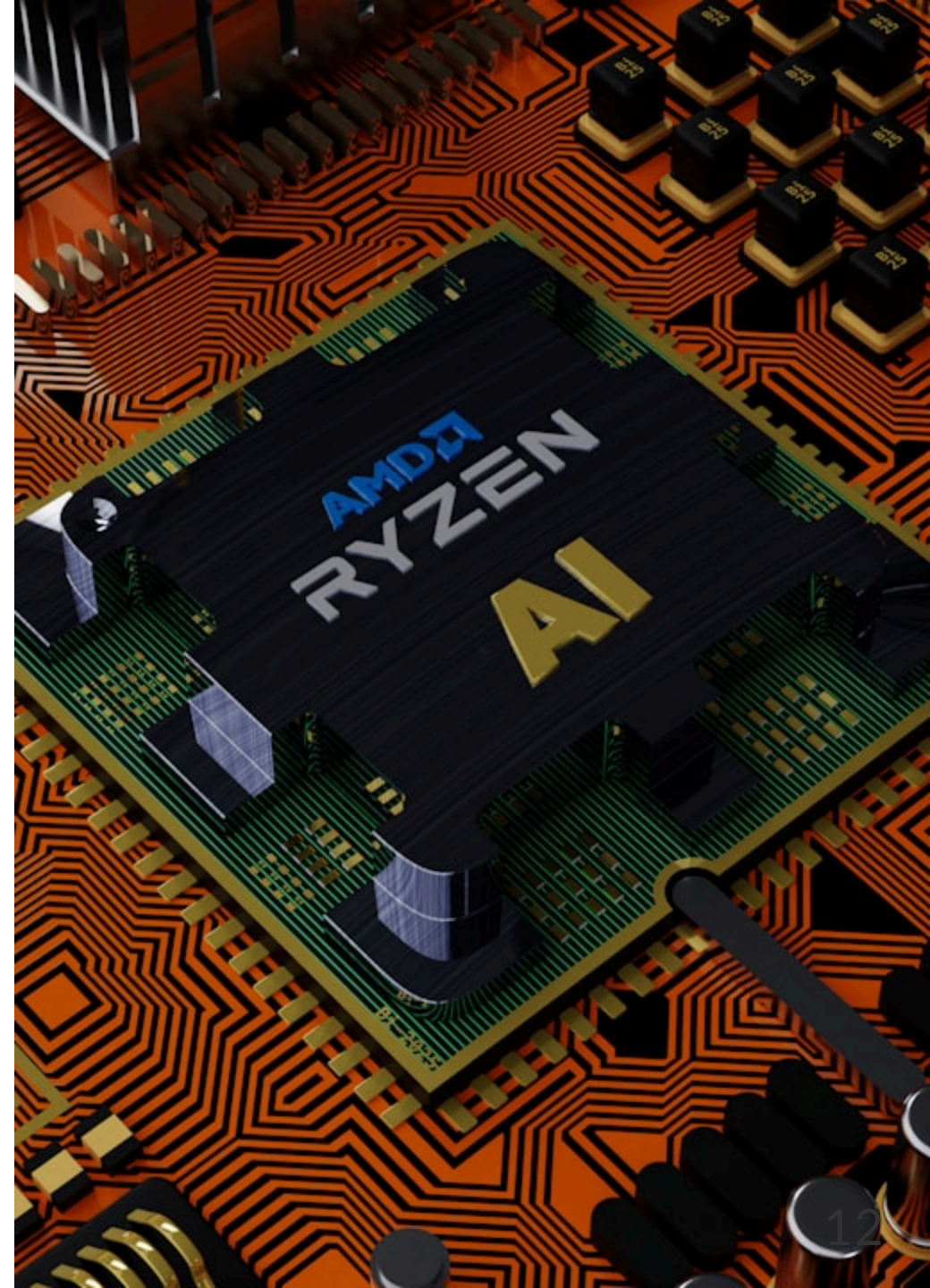
What is a process?

- A process is an instance of a program that is being executed
- A process can have multiple threads (such as picocli)
- Processes are isolated from each other and have their own memory space
- Heavier than threads: expensive to create and manage



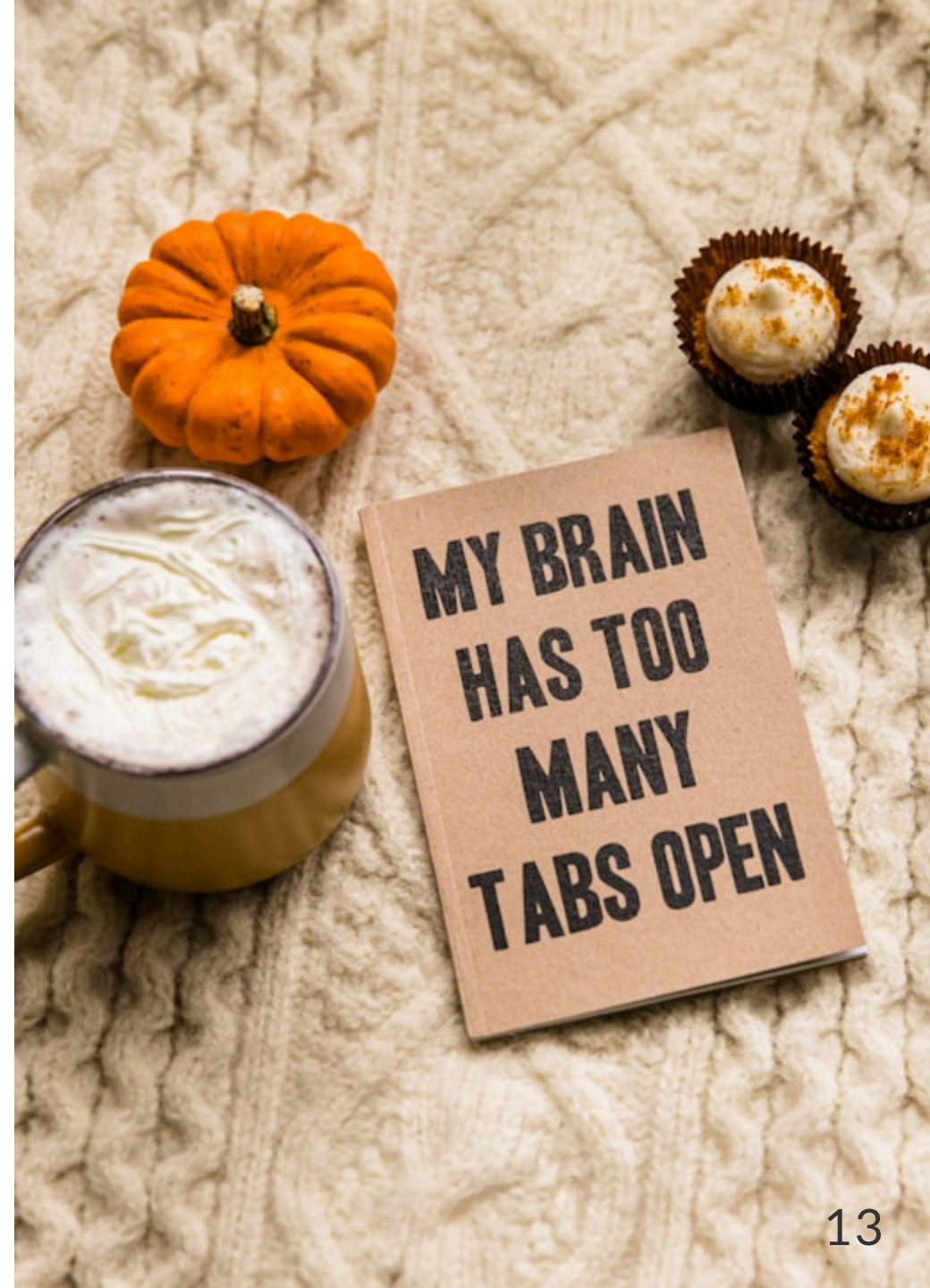
What is a thread?

- A thread is a sequence of instructions that can be executed by a core
- A core can manage multiple threads at the same time
- The main thread is the thread that starts when the application starts
- More lightweight than processes and share the same memory space



What problems can concurrency cause?

- What happens when a client connects to the server?
- How to isolate the data sent by one client from the others?
- What happens when multiple clients want to access the same resource (variables) at the same time?



What happens when multiple threads access the same resource?

- The processor has to switch between threads
- The order in which the threads are executed is not guaranteed
- The threads can access the same resource at the same time



Handling one client at a time

More details for this section in the [course material](#). You can find other resources and alternatives as well.

Handling one client at a time

1. Create a socket to listen to incoming connections
2. Create a new socket for the client
3. Handle the connection

Analogy: a restaurant with one table

➡ Simple but quite useless...



Handling multiple clients with concurrency

More details for this section in the [course material](#). You can find other resources and alternatives as well.

Handling multiple clients at the same time

Handling multiple clients at the same time is called concurrency.

Concurrency can be achieved with:

- Multi-processing
- Multi-threading
- Asynchronous programming



Multi-processing

- Create an entirely new process for each client
- Heavyweight and slow
- Not recommended
- Analogy: a new restaurant for each customer, including the kitchen, waiters, etc.



Multi-threading

- Create a new thread for each client
- More lightweight and faster than multi-processing
- Recommended with thread pool to limit resource usage
- Not enough or too many threads can slow down the system
- Analogy: a new or limited number of tables for each customer



Asynchronous programming

- Handle multiple clients with a single thread
- Very performant!
- Analogy: a food truck - you get a ticket, wait, get your food and leave
- Out of scope for this course but interesting to know! Node.js is a good example for this approach



Handling multiple threads in Java

More details for this section in the [course material](#). You can find other resources and alternatives as well.

Handling multiple threads in Java

- Java provides the `ExecutorService` interface to manage threads
- It uses different `Executors` implementations
- Each implementation has its own pros and cons
- You have used threads with picocli (each sub-command is a thread)



```
@Override
public Integer call() {
    try (ExecutorService executorService = Executors.newFixedThreadPool(2); ) {
        executorService.submit(this::emittersWorker);
        executorService.submit(this::operatorsWorker);
    } catch (Exception e) {
        System.out.println("[Receiver] Exception: " + e);

        return 1;
    }

    return 0;
}

public Integer emittersWorker() {
    // Manage emitters
}

public Integer operatorsWorker() {
    // Manage operators
}
```


Concurrent safe variable types and data structures in Java

More details for this section in the [course material](#). You can find other resources and alternatives as well.

Concurrent safe variable types and data structures in Java

- A thread-safe variable type and/or data structure is a variable type/data structure that can be accessed by multiple threads at the same time
- Java provides several thread-safe variable types and data structures



- **Variables types:**

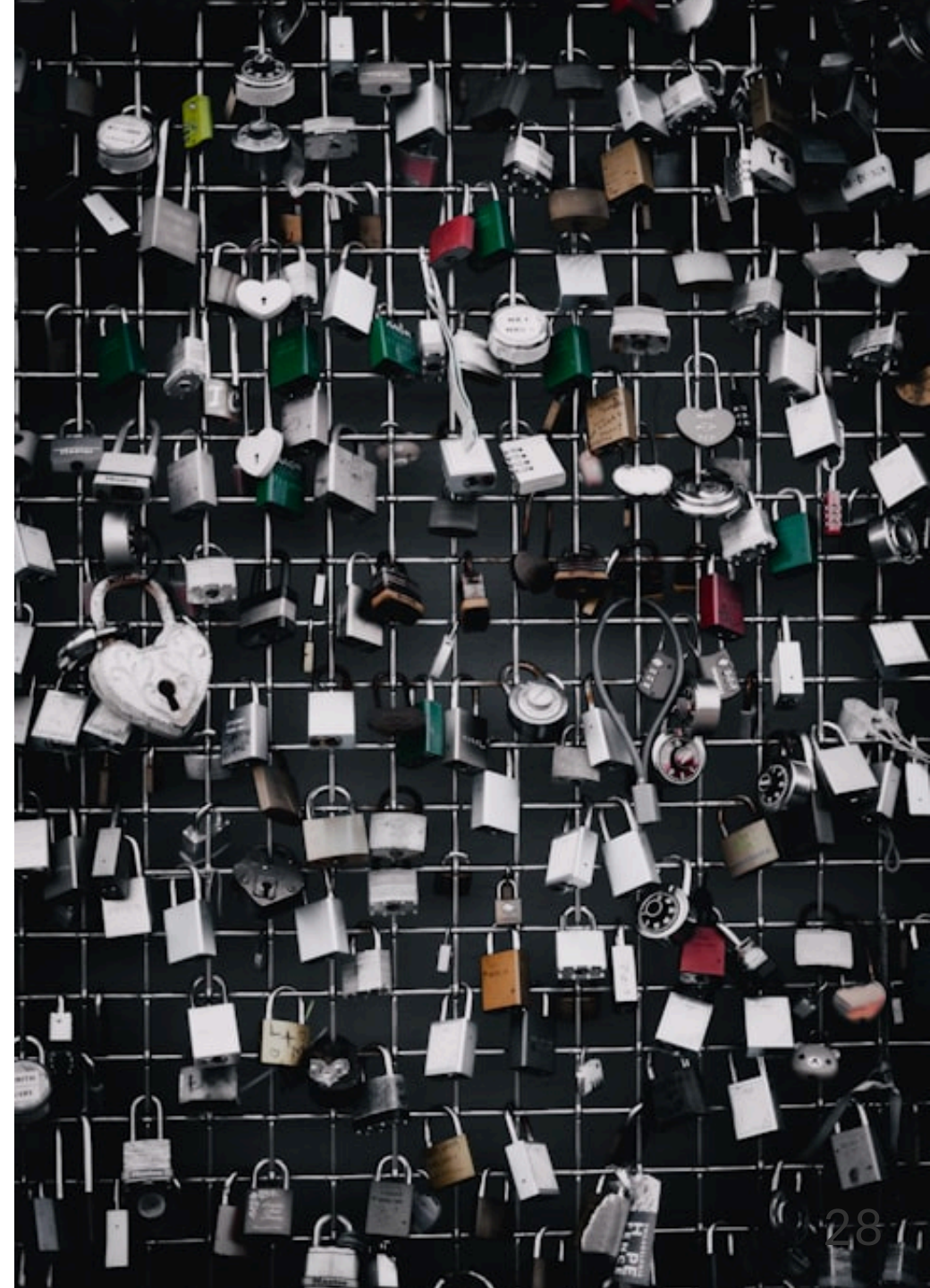
- `AtomicBoolean` instead of `boolean` / `Boolean`
- `AtomicInteger` instead of `int` / `Integer`
- `AtomicLong` instead of `long` / `Long`

- **Data structures:**

- `ConcurrentHashMap` instead of `HashMap`
- `ConcurrentLinkedQueue` instead of `LinkedList`
- `CopyOnWriteArrayList` instead of `ArrayList`
- `CopyOnWriteArraySet` instead of `HashSet`

- Multiple concurrent variable types and/or data structures can be co-dependent
- You must use them together with locks (the equivalent to mutexes and semaphores in C/C++) to avoid inconsistencies

Locks will not be covered (and not expected to be used) in this course but you can find additional information in the [course material](#).



Questions

Do you have any questions?

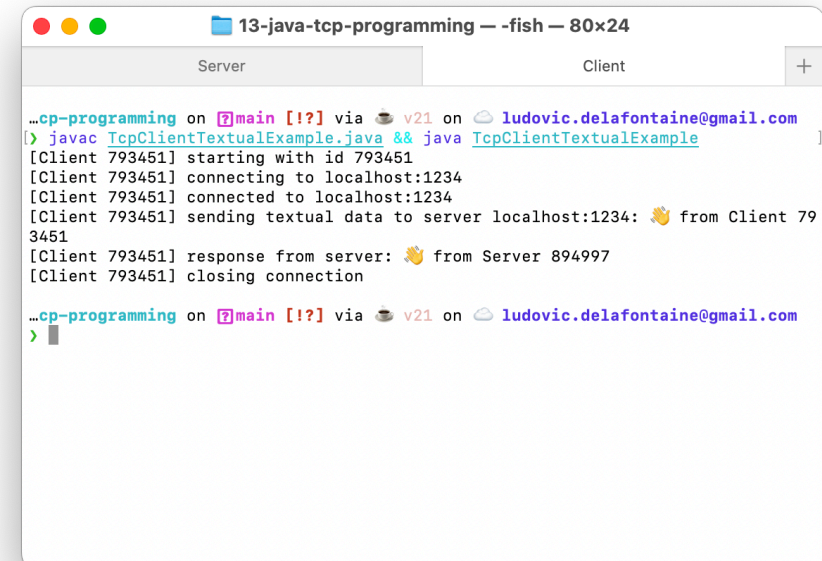
Practical content

What will you do?

- Run full client/server examples and understand how concurrent clients are handled
- Explore and answer questions about some concurrent strategies
- Update previous game/application to handle multiple clients (optional)



```
13-java-tcp-programming — java TcpServerSimpleTextualExample — 80x...
Server Client +
> javac TcpServerSimpleTextualExample.java && java TcpServerSimpleTextualExample
[Server 894997] starting with id 894997
[Server 894997] listening on port 1234
[Server 894997] new client connection
[Server 894997] received textual data from client: 🍌 from Client 686573
[Server 894997] sending response to client: 🍌 from Server 894997
[Server 894997] closing connection
[Server 894997] new client connection
[Server 894997] received textual data from client: 🍌 from Client 866646
[Server 894997] sending response to client: 🍌 from Server 894997
[Server 894997] closing connection
[Server 894997] new client connection
[Server 894997] received textual data from client: 🍌 from Client 835722
[Server 894997] sending response to client: 🍌 from Server 894997
[Server 894997] closing connection
[Server 894997] new client connection
[Server 894997] received textual data from client: 🍌 from Client 444354
[Server 894997] sending response to client: 🍌 from Server 894997
[Server 894997] closing connection
[Server 894997] new client connection
[Server 894997] received textual data from client: 🍌 from Client 386144
[Server 894997] sending response to client: 🍌 from Server 894997
[Server 894997] closing connection
```



```
13-java-tcp-programming — -fish — 80x24
Server Client +
...cp-programming on [?]main [!?] via 🍌 v21 on ☁ ludovic.delafontaine@gmail.com
> javac TcpClientTextualExample.java && java TcpClientTextualExample
[Client 793451] starting with id 793451
[Client 793451] connecting to localhost:1234
[Client 793451] connected to localhost:1234
[Client 793451] sending textual data to server localhost:1234: 🍌 from Client 793451
[Client 793451] response from server: 🍌 from Server 894997
[Client 793451] closing connection
...cp-programming on [?]main [!?] via 🍌 v21 on ☁ ludovic.delafontaine@gmail.com
>
```

Find the practical content

You can find the practical content for this chapter on [GitHub](#).



Finished? Was it easy? Was it hard?

Can you let us know what was easy and what was difficult for you during this chapter?

This will help us to improve the course and adapt the content to your needs. If we notice some difficulties, we will come back to you to help you.

 [GitHub Discussions](#)

You can use reactions to express your opinion on a comment!

What will you do next?

- Learn about electronic mail protocols
- Experiment with the SMTP protocol
 - Start a local SMTP server for testing
 - Send an email using ncat
 - Send an email using a SMTP client written in Java



Sources

- Main illustration by [Brent Olson](#) on [Unsplash](#)
- Illustration by [Aline de Nadai](#) on [Unsplash](#)
- Illustration by [Bernard Hermant](#) on [Unsplash](#)
- Illustration by [alexey_turenkov](#) on [Unsplash](#)
- Illustration by [Andrey Matveev](#) on [Unsplash](#)
- Illustration by [BoliviaInteligente](#) on [Unsplash](#)
- Illustration by [BoliviaInteligente](#) on [Unsplash](#)
- Illustration by [BoliviaInteligente](#) on [Unsplash](#)

- Illustration by [That's Her Business](#) on [Unsplash](#)
- Illustration by [Jens Herrndorff](#) on [Unsplash](#)
- Illustration by [Elliott Stallion](#) on [Unsplash](#)
- Illustration by [K8](#) on [Unsplash](#)
- Illustration by [bckfwd](#) on [Unsplash](#)
- Illustration by [James Elchico](#) on [Unsplash](#)
- Illustration by [micheile henderson](#) on [Unsplash](#)
- Illustration by [Afrah](#) on [Unsplash](#)
- Illustration by [Pop & Zebra](#) on [Unsplash](#)
- Illustration by [Kateryna Miller](#) on [Unsplash](#)

- Illustration by [MChe Lee](#) on [Unsplash](#)