

# Practical work 2

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

[Web](#) • [PDF](#)

L. Delafontaine and H. Louis, with the help of [GitHub Copilot](#).

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

# Objectives

- Define and implement a network application with TCP and/or UDP
- Package, publish and run a network application with Docker
- You can choose what the network application will do (you can be creative!)
  - a chat application, a chess game, a shopping list application, ...





# Demo 1

A simple file transfer application made with TCP

## Compile the project:

```
./mvnw clean package
```

## Run the CLI without any arguments:

```
java -jar target/practical-work-2-demo-1-1.0-SNAPSHOT.jar
```

```
Missing required subcommand
```

```
Usage: DaiFileTransfer [COMMAND]
```

```
A simple file transfer application
```

```
Commands:
```

```
    server  Start a TCP server to serve files
```

```
    client  Start a client to download files from a server
```

Start the server:

```
java -jar target/practical-work-2-demo-1-1.0-SNAPSHOT.jar server -p 12345 -t 4
```

Start the client:

```
java -jar target/practical-work-2-demo-1-1.0-SNAPSHOT.jar client -p 12345
```

```

      -----
|  _ _ \  /\  |  _ _ |  |  _ _ \  |  _ _ |  _ _ |
|  |  |  | /  \  |  |  |  |  _ _ )  |  _ _ _ _ _ |  |  _ _ _ _ _ |  _ _ _ _ _ |
|  |  |  | /  /\  \  |  |  |  |  _ _ /  ' _ _ /  _ \  |  _ _ /  _ \  /  _ _ /  _ \  |
|  |  _ _ | /  _ _ _ \  _ |  |  |  |  |  ( _ )  |  |  ( _ )  |  ( _ |  ( _ )  |  |
|  _ _ _ _ /  _ /  _ \  _ \  _ _ _ _ |  |  _ _ |  _ _ |  _ _ \  _ _ _ _ /  _ _ \  _ _ _ _ /  |  _ _ |

```

You are connected to server on port 12345 with IP address localhost.

Available commands:

LS - list available files on server

GET <file> - get file from server

QUIT - quit the client

>

List available files:

```
> ls
Available files on server:
demo.txt
my-passwords-in-clear.txt
rzs-sc2.exe
```

Get one of the available files:

```
> GET my-passwords-in-clear.txt
Downloading file from server...
File saved to my-passwords-in-clear.txt
```

Get a file that does not exist:

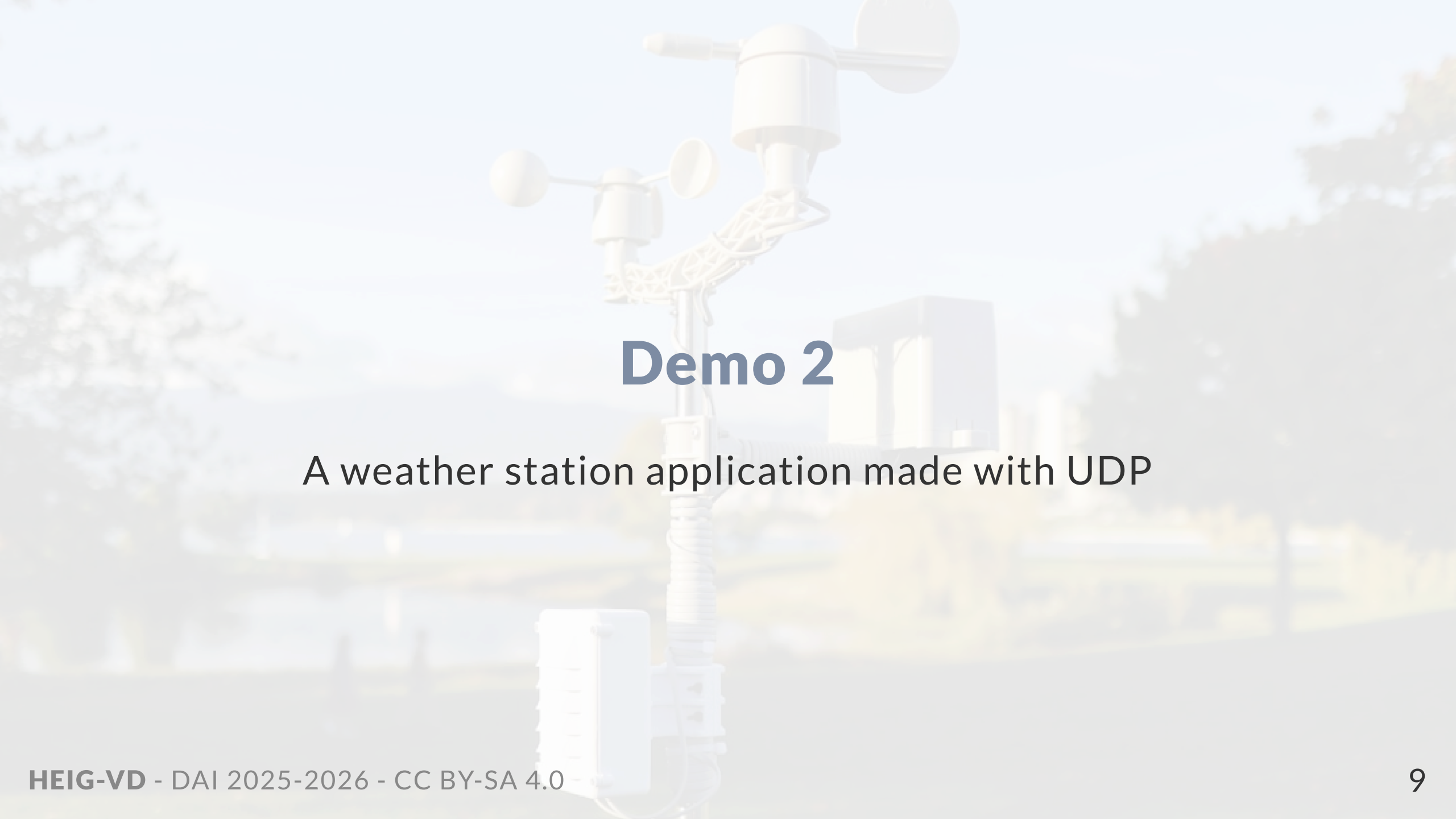
```
> GET not-found.txt  
The specified file was not found on the server.
```

Quit:

```
> QUIT  
Bye.
```

If a client tries to connect to the server when no thread is available, the client has to wait to be served.

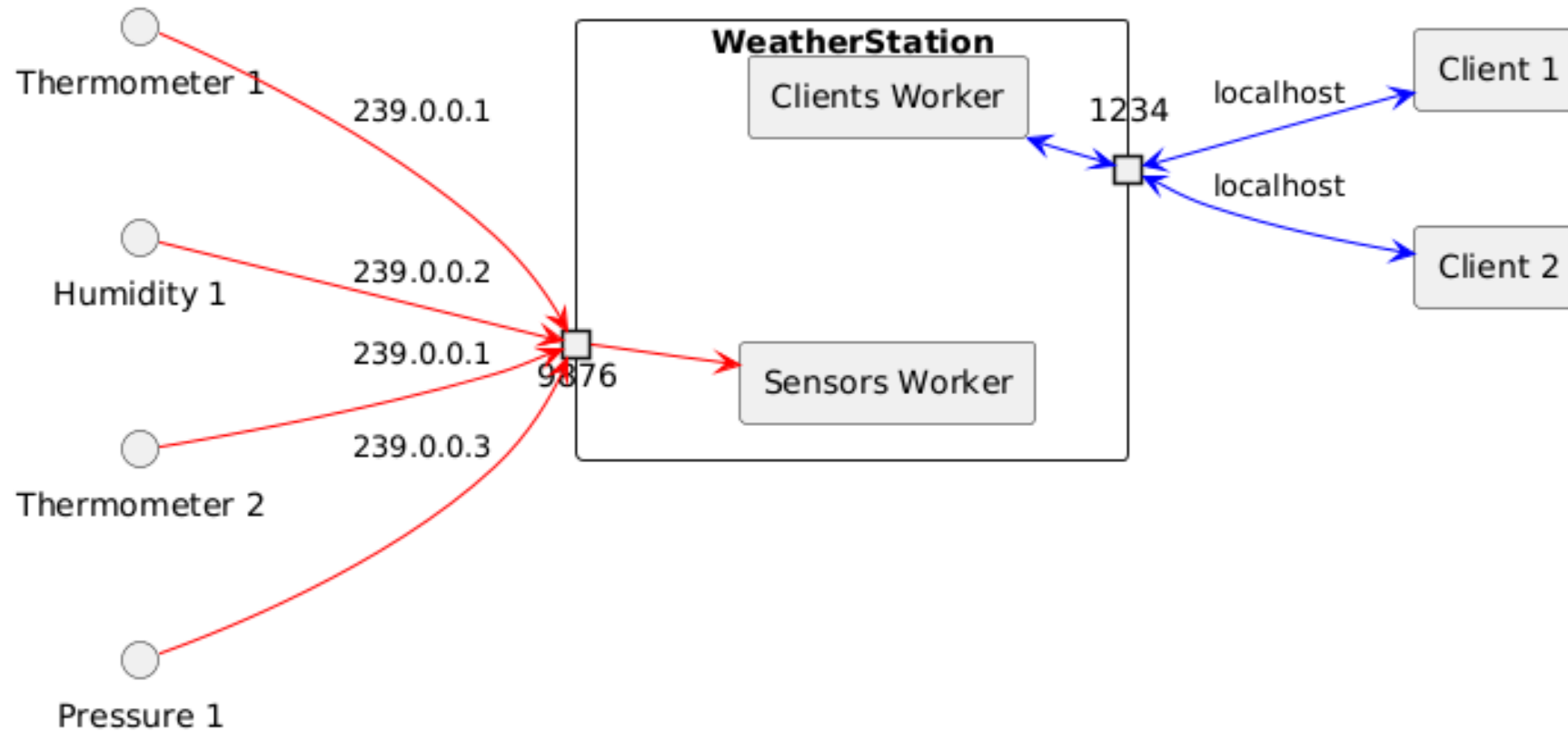




## Demo 2

A weather station application made with UDP

## Practical Work 2 - UDP network application - Architecture



| Color | Type                        |
|-------|-----------------------------|
| Red   | Multicast - Fire and Forget |
| Blue  | Unicast - Request Response  |

Compile the project:

```
./mvnw clean package
```

Run the CLI without any arguments:

```
java -jar target/practical-work-2-demo-2-1.0-SNAPSHOT.jar
```

Missing required subcommand

Usage: practical-work-2-demo-2-1.0-SNAPSHOT.jar [-hV] [COMMAND]

DAI Weather Station

-h, --help Show this help message and exit.

-V, --version Print version information and exit.

Commands:

thermometer-emitter Emits temperature values.

pressure-emitter Emits pressure values.

humidity-emitter Emits humidity values.

weather-station Start an UDP multicast receiver.

weather-client Start an UDP weather client.

list-network-interfaces List all available network interfaces.

Start the weather station:

```
java -jar target/practical-work-2-demo-2-1.0-SNAPSHOT.jar weather-station \  
-i eth0 \  
--port 9876 \  
--server-port 1234
```

Output:

```
Multicast receiver started (10.11.12.25:9876)  
Unicast receiver started (10.11.12.25:1234)
```

Start the emitters:

```
java -jar target/practical-work-2-demo-2-1.0-SNAPSHOT.jar thermometer-emitter \  
  --delay=0 \  
  --frequency 10000 \  
  -i eth0 \  
  -p 9876
```

```
java -jar target/practical-work-2-demo-2-1.0-SNAPSHOT.jar pressure-emitter \  
  --delay=0 \  
  --frequency 10000 \  
  -i eth0 \  
  -p 9876
```

## Output:

```
Emitter of type thermometer started (10.11.12.25:9876).
```

```
Multicasting measure : 30.0 to 239.0.0.1:9876 on interface eth0
```

```
Multicasting measure : 23.0 to 239.0.0.1:9876 on interface eth0
```

```
Multicasting measure : 28.0 to 239.0.0.1:9876 on interface eth0
```

```
Multicasting measure : 19.0 to 239.0.0.1:9876 on interface eth0
```

```
Multicasting measure : 15.0 to 239.0.0.1:9876 on interface eth0
```

```
Emitter of type pressure started (10.11.12.25:9876).
```

```
Multicasting measure : 982.0 to 239.0.0.2:9876 on interface eth0
```

```
Multicasting measure : 1027.0 to 239.0.0.2:9876 on interface eth0
```

```
Multicasting measure : 970.0 to 239.0.0.2:9876 on interface eth0
```

```
Multicasting measure : 996.0 to 239.0.0.2:9876 on interface eth0
```

```
Multicasting measure : 982.0 to 239.0.0.2:9876 on interface eth0
```

Start the weather client:

```
java -jar practical-work-2-demo-2-1.0-SNAPSHOT.jar weather-client
```

Output:

```
Welcome to the weather client!  
Please enter the measures you want to get  
1. Temperature  
2. Humidity  
3. Pressure  
4. Quit  
>
```

You can then select the measures you want to get.



```
> 1
List all measures (1) or only the average? (2)
> 1
-----
Received measures: [30.0, 23.0, 28.0, 19.0, 15.0]
-----
Please enter the measures you want to get
1. Temperature
2. Humidity
3. Pressure
4. Quit
>
```

The client will then display all the measures received from the weather station.

```
> 3
List all measures (1) or only the average? (2)
> 2
-----
Received average: 991.11
-----
Please enter the measures you want to get
1. Temperature
2. Humidity
3. Pressure
4. Quit
> 4
```

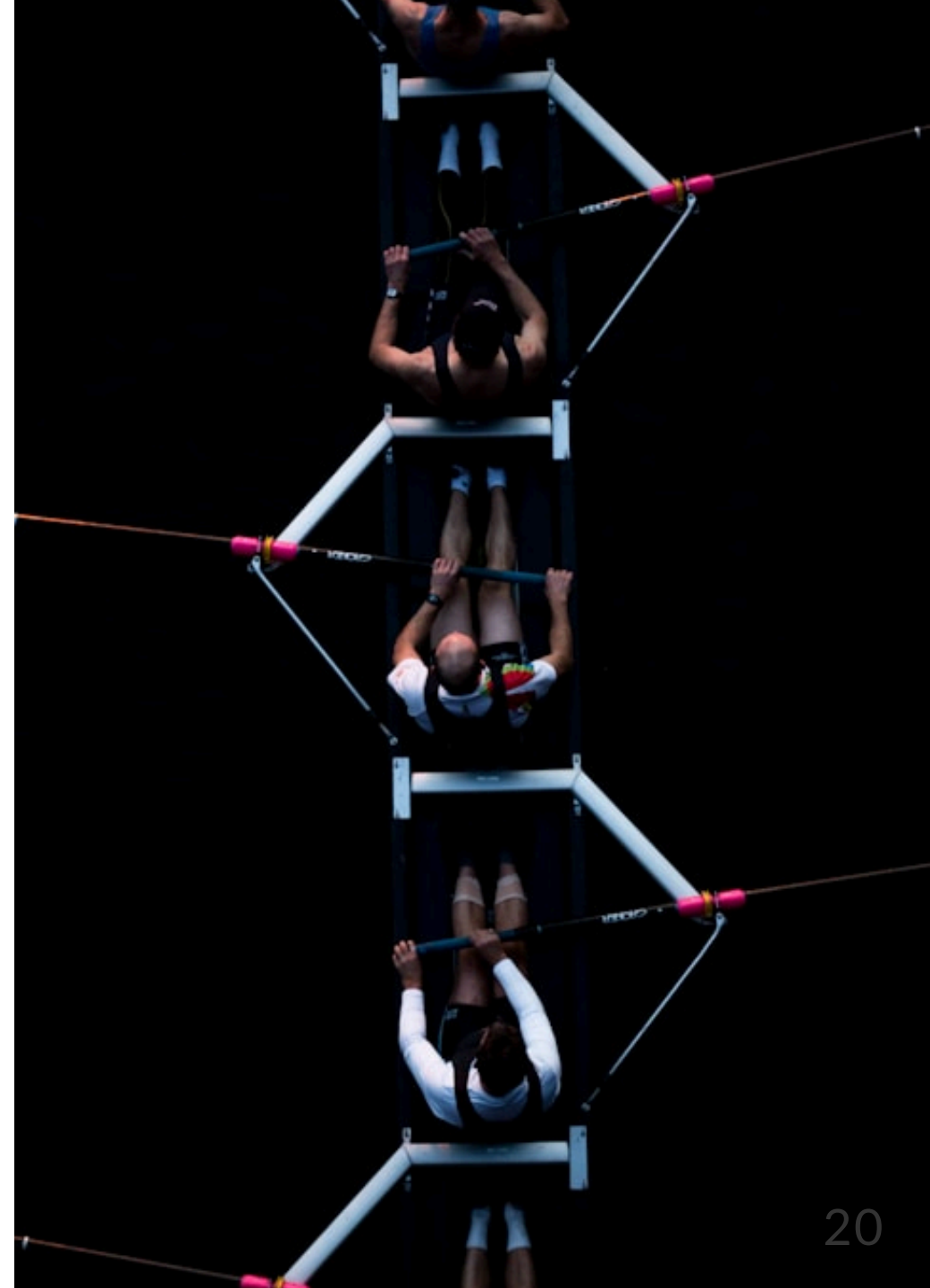
The client can then quit the application.

# Group composition

More details for this section in the [course material](#).

# Group composition

- Two (2) or three (3) students per group.
- Create a GitHub Discussion to:
  - Announce your group members.
  - Announce your idea (even a draft is fine).
- **You must do it before next week,** otherwise you will be penalized (check the [Constraints](#) for details).



# Idea validation

More details for this section in the [course material](#).

# Idea validation

- You must state your idea on your GitHub Discussion.
- We might ask you to change your idea if it is too simple or too complex.
- We will help you to find a good idea if needed.
- **You must do it before next week!**



# Grading criteria

More details for this section in the [course material](#).

# Grading criteria

You can find all the grading criteria in the [course material](#):

- 0 point - The work is insufficient
- 0.1 point - The work is done
- 0.2 point - The work is well done  
(without the need of being perfect)

Maximum grade:  $25 \text{ points} * 0.2 + 1 = 6$





# Constraints

More details for this section in the [course material](#).

# Constraints

- The application must be written in Java, compatible with Java 21
- The application must be built using Maven with the `maven-shade-plugin` plugin
- The application must use the picocli dependency
- You can only use the Java classes seen in the course
- Your application must be slightly more complex and slightly different than the examples presented during the course (we emphasize the word **slightly**, no need to shoot for the moon!)

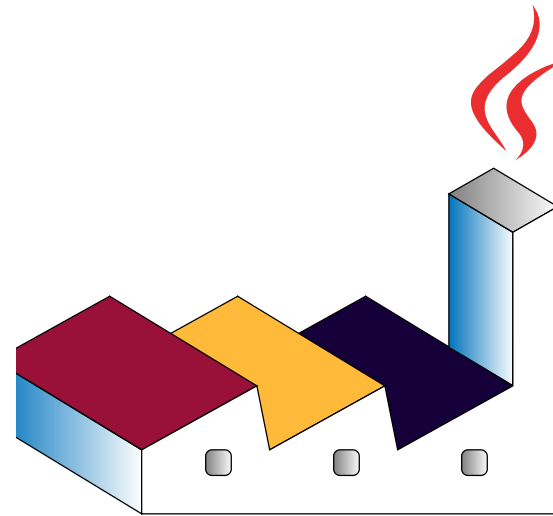
# Tips

More details for this section in the [course material](#).

# Create diagrams

- You can use any tools you want to create your diagrams:
  - [PlantUML](#)
  - [draw.io](#)
  - Scans from paper diagrams
  - Any other tools you want

PDF, PNG, SVG, etc. are all accepted formats in your repository.



# Extract the command and parameters from the message

The Short Message Service (SMS) protocol presented in the ["Define an application protocol"](#) chapter (accessible in the [examples repository](#) repository) defines the following message:

```
RECEIVE <message> <username>
```

This message is sent by the server to a client to inform them that they have received a message from another user.

The command is `RECEIVE` and the parameters are `<message>` and `<username>` .

The message can be up to 100 characters long.

You can use the following snippet of code to extract the command and the parameters from the message:

```

List<String> messageParts = Arrays.asList(emitterMessage.split(" ", 2));

String command = messageParts.get(0);
switch (command) {
    // Other cases can be defined here...
    case "RECEIVE" -> {
        if (messageParts.size() < 2) {
            // Invalid message, do something about it such as logging it or returning an error...
        }

        List<String> parameters = Arrays.asList(messageParts.get(1).split(" "));

        if (parameters.size() < 2) {
            // Invalid message, do something about it such as logging it or returning an error...
        }

        String user = parameters.removeLast();
        String message = String.join(" ", parameters);

        System.out.printf("Message from %s: %s\n", user, message);
    }
    // Other cases can be defined here...
}

```

# The POSIX standard

*“ The Portable Operating System Interface (POSIX) standard is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems. POSIX defines both the system and user-level application programming interfaces (APIs), along with command line shells and utility interfaces, for software compatibility (portability) with variants of Unix and other operating systems.*

*<https://en.wikipedia.org/wiki/POSIX>*

*”*



# Submission

More details for this section in the [course material](#).

# Submission

Your work is due as follows:

- DAI-TIC-C (Friday mornings): **05.12.2024 23:59**
- DAI-TIC-B (Monday mornings): **08.12.2024 23:59**

Update the GitHub Discussion with the link to your repository as mentioned in the course material.

**If you do not submit your work on time and/or correctly, you will be penalized (-1 point on the final grade for each day of delay).**

# Presentations

More details for this section in the [course material](#).

# Presentations

The practical work presentations will take place in **B38** on:

- DAI-TIC-C (Friday mornings): **06.12.2024**
- DAI-TIC-B (Monday mornings): **09.12.2024**

We only have **10 minutes per group**. You decide what you want to show us and how you want to present it.

**Come 5 minutes before your time slot** with your computer. You will have access to a video projector.

# Grades and feedback

Grades will be entered into GAPS, followed by an email with the feedback.

The evaluation will use exactly the same grading grid as shown in the course material.

Each criterion will be accompanied by a comment explaining the points obtained, a general comment on your work and the final grade.

If you have any questions about the evaluation, you can contact us!

# Questions

Do you have any questions?

# Find the practical work

You can find the practical  
work for this part 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 practical work?

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!



# Sources

- Main illustrations by [Rafael Rex Felisilda](#) on [Unsplash](#) and [Jorge Ramirez](#) on [Unsplash](#)
- Illustration by [Aline de Nadai](#) on [Unsplash](#)
- Illustration by [Josh Calabrese](#) on [Unsplash](#)
- Illustration by [Nicole Baster](#) on [Unsplash](#)
- Illustration by [Chris LaBarge](#) on [Unsplash](#)