



# PRACTICAL JAVA FOR IT PROFESSIONALS

KRISTOF WERLING  
KWERLINGIT GMBH  
1 SEPT 2023



## WHO AM I

- Kristof Werling
- 31 years of experience at HP:  
DevOp, Developer, Architect
- Started KwerlingIT GmbH
- Focus: IT / Cyber Security
- Security Audits, IT consulting,  
Trainings, Software creation



KwerlingIT GmbH

[kristof.werling@kwerlingit.com](mailto:kristof.werling@kwerlingit.com)

<https://www.linkedin.com/in/kristof-werling/>

<https://www.kwerlingit.com>

# TABLE OF CONTENT

- Git Training (separate set of slides)
- Familiarization with Java
- Networking (Sockets, REST Webservices, Message Bus)
- Databases (Relational DBs, SQL, KV-DB)
- Web Uis (Bootstrap)



# MODULE A

## INTRODUCTION/WARMING UP

JUST WRITING SOME JAVA CODE AND GETTING TO KNOW SOME  
OF THE ODDITIES.

# CONTENT

- Command line parameters
- Working with Strings
- Garbage Collector
- Command Line Parameters
- Reverse Polish Calculator
- Unit testing

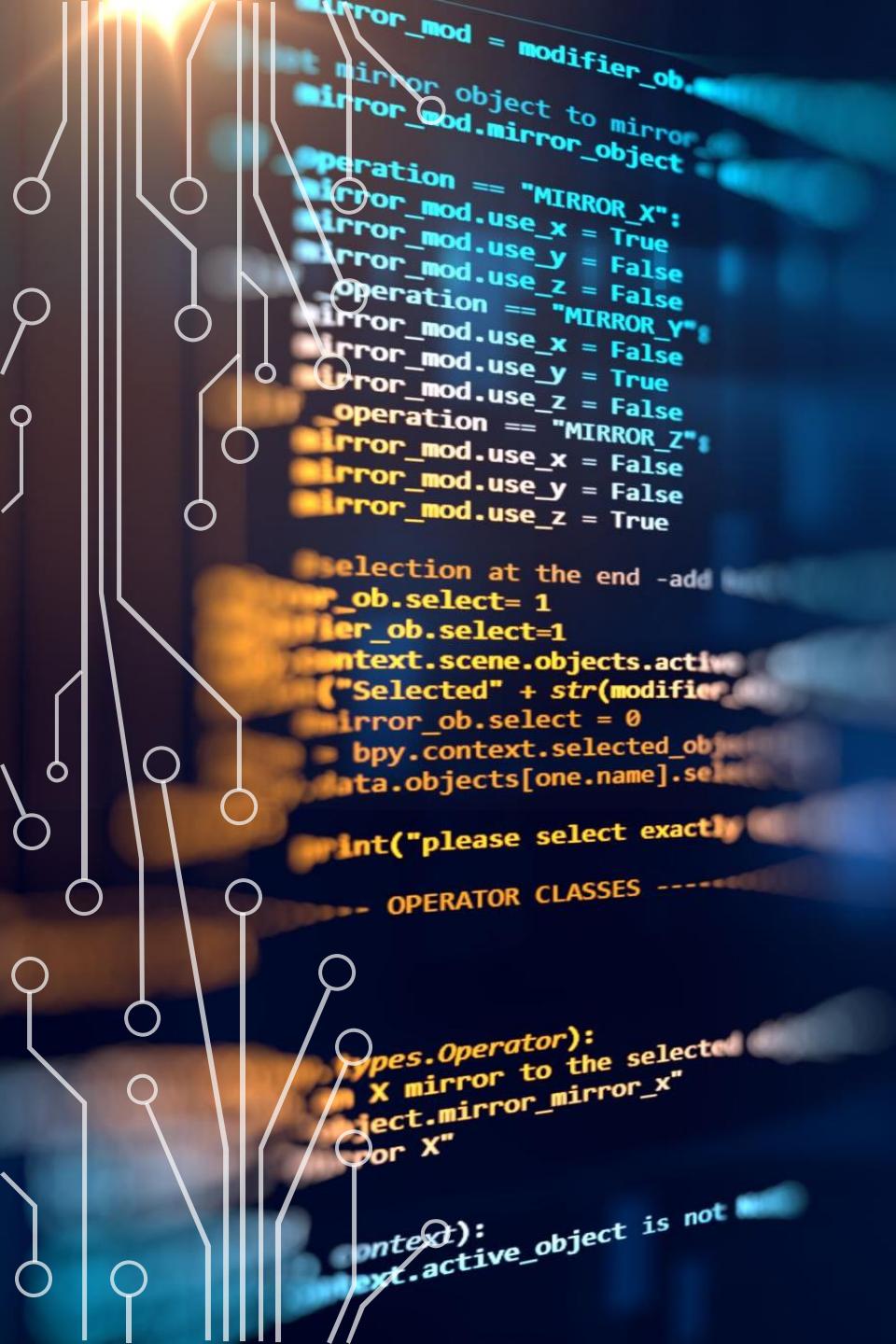
# COMMAND LINE PARAMETERS

- IntelliJ provides one with some default java program
- The "args" array contains the command line parameters

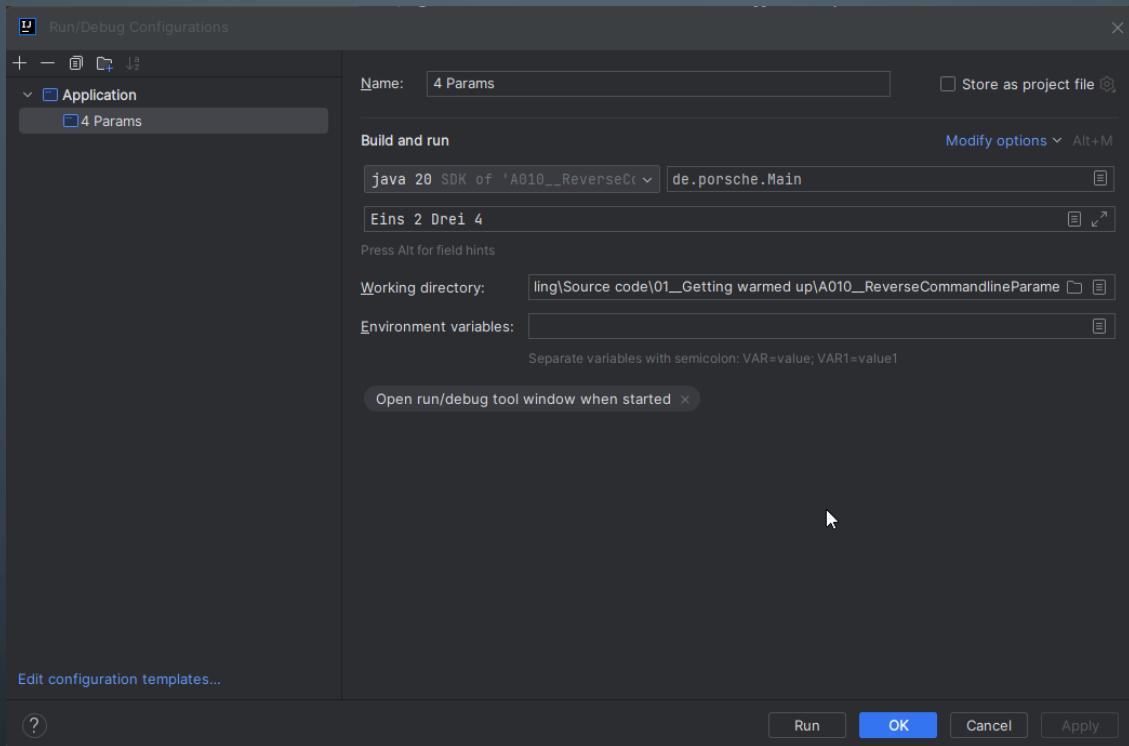
```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello world!");  
    }  
}
```

# EXERCISE A010

- Write a program, which prints out the command line parameters in reverse order
- Make sure that the source code is in the name space "de.porsche"



# SETTING RUN CONFIGURATION



# WORKING WITH STRINGS

(some) String class method	Functionality
<a href="#"><u>String toLowerCase()</u></a>	Returns all lowercase string.
<a href="#"><u>String toUpperCase()</u></a>	Returns all UPPERCASE string.
<a href="#"><u>String trim()</u></a>	Removes beginning and ending spaces of this string.
<a href="#"><u>int indexOf(String substring)</u></a>	Returns a substring.
<a href="#"><u>String[] split(String regex)</u></a>	Splits string at matching regex. Returns an array.
<a href="#"><u>boolean contains(CharSequence s)</u></a>	Returns if character sequence is contained in string.
<a href="#"><u>int length()</u></a>	It returns the string length. Compare to Array.length !!
<a href="#"><u>String substring(int beginIndex, int endIndex)</u></a>	Returns a substring.

# WORKING WITH STRINGS

RegEx character	What is it?	What does it match
[abcA-Z0-3]	Character class	Any character in the brackets. 0-3 stands for 0, 1, 2, 3 (range)
.	Metacharacter	Any character but linefeed
* ? + {2} {2,} {2,4}	Quantifiers	How often: zero or more, zero or one, one or more, exactly 2 times, at least 2 times, 2, 3 or 4 times
\t \r \n	White space	Tab, carriage return, linefeed
^ \$	Anchors	Beginning and End of line

There is much more to know about RegEx. Please do your own research online.

# WORKING WITH STRINGS

RegEx	
H[ao]use?	Matches : Haus, House, aber auch Hause
Han{2}a	Matches: Hanna
Gr[ae]y	Matches: Grey or Gray
Cat\$	Line ends with the word Cat
^Here we go	Line starts with: Here we go
^\t]*Here we go	Line starts with or without whitespaces followed by the text: Here we go

There is much more to know about RegEx. Please do your own research online.

# WORKING WITH STRINGS

Part 1

```
package de.porsche;

public class Main {

    public static void main(String[] args) {

        System.out.println("Explain the following results:");

        // Bad : Variables are to be declared on top of the block

        System.out.println( "" ); System.out.println( "" );
        String S1 = "Hello World";
        String S2 = S1;
        System.out.println( " S1 and S2 are equal      :" + ( S1 == S2 ) );
        System.out.println( " S1 and S2 have the same content:" + ( S1.compareTo(S2) == 0 ) );

        System.out.println( "" ); System.out.println( "" );
        String S3 = "Hello World";
        System.out.println( " S1 and S3 are equal      :" + ( S1 == S3 ) );
        System.out.println( " S2 and S3 are equal      :" + ( S2 == S3 ) );
        System.out.println( " S1 and S3 have the same content:" + ( S1.compareTo(S3) == 0 ) );
        System.out.println( " S2 and S3 have the same content:" + ( S2.compareTo(S3) == 0 ) );
    }
}
```

# WORKING WITH STRINGS

Part 2

```
System.out.println( "" ); System.out.println( "" );
String S4 = new String("Hello World");
System.out.println( " S1 and S4 are equal      : " + ( S1 == S4 ) );
System.out.println( " S1 and S4 have the same content: " + ( S1.compareTo(S4) == 0 ) );

System.out.println( "" ); System.out.println( "" );
String S5 = "Hello " + "World";
System.out.println( " S1 and S5 are equal      : " + ( S1 == S5 ) );
System.out.println( " S1 and S5 have the same content: " + ( S1.compareTo(S5) == 0 ) );

System.out.println( "" ); System.out.println( "" );
String S6 = "Hello ";
S6 = S6 + "World";
System.out.println( " S1 and S6 are equal      : " + ( S1 == S6 ) );
System.out.println( " S1 and S6 have the same content: " + ( S1.compareTo(S6) == 0 ) );
```

# WORKING WITH STRINGS

Part 3

```
System.out.println( "" ); System.out.println( "" );
System.out.println( " S1 identity hash code      : " + System.identityHashCode(S1));
System.out.println( " S2 identity hash code      : " + System.identityHashCode(S2));
System.out.println( " S3 identity hash code      : " + System.identityHashCode(S3));
System.out.println( " S4 identity hash code      : " + System.identityHashCode(S4));
System.out.println( " S5 identity hash code      : " + System.identityHashCode(S5));
System.out.println( " S6 identity hash code      : " + System.identityHashCode(S6));

System.out.println( "" ); System.out.println( "" );
S5 = "Another text";
System.out.println( " S5 identity hash code      : " + System.identityHashCode(S5));
}
```

# EXERCISE A020

- Run the program in the 3 slides before and explain the result

# WORKING WITH STRINGS

Part 1

```
package de.porsche;

public class Main {

    static long startTime = 0;
    static long endTime = 0;

    static void startTimer() {
        startTime = System.nanoTime();
    }

    static void endTimer() {
        endTime = System.nanoTime();
    }

    static void timerResults() {
        long duration = endTime - startTime;
        System.out.println("The operation took \n"
            + duration + " nano seconds\n"
            + duration / 1000 + " micro seconds\n"
            + duration / 1000000 + " milli seconds\n"
            + duration / 1000000000 + " seconds\n");
    }
}
```

# WORKING WITH STRINGS

Part 2

```
public static void main(String[] args) {
    String sourceString = "1234".repeat(256);
    System.out.println("Source string length: " + sourceString.length());
    String resultString = "";

    // To get some idea of how long a loop takes
    System.out.println("Empty For-Loop, 100 iterations");
    startTimer();
    for (int i = 0; i < 100; i++) {
    }
    endTimer();
    timerResults();
    System.out.println("\n\n");

    System.out.println("Empty For-Loop, 1 million iterations");
    startTimer();
    for (int i = 0; i < 1000000; i++) {
    }
    endTimer();
    timerResults();
    System.out.println("\n\n");
```

# WORKING WITH STRINGS

```
System.out.println("Concat string, 100 iterations");
resultString = "";
startTimer();
for (int i = 0; i < 100; i++) {
    resultString = resultString + sourceString;
}
endTimer();
System.out.println("Result string length: " + resultString.length());
timerResults();
System.out.println("\n\n");

System.out.println("Concat string, 1000 iterations");
resultString = "";
startTimer();
for (int i = 0; i < 1000; i++) {
    resultString = resultString + sourceString;
}
endTimer();
System.out.println("Result string length: " + resultString.length());
timerResults();
System.out.println("\n\n");
```

# WORKING WITH STRINGS

Part 4

```
System.out.println("Concat string, 10000 iterations");
resultString = "";
startTimer();
for (int i = 0; i < 10000; i++) {
    resultString = resultString + sourceString;
}
endTimer();
System.out.println("Result string length: " + resultString.length());
timerResults();
System.out.println("\n\n");

// System.out.println("Concat string, 100000 iterations");
// resultString = "";
// startTimer();
// for (int i = 0; i < 100000; i++) {
//     // resultString = resultString + sourceString;
//     // Took 1010 secs on my machine.
// }
// endTimer();
// System.out.println("Result string length: " + resultString.length());
// timerResults();
// System.out.println("\n\n");
```

# WORKING WITH STRINGS

Part 5

```
System.out.println("Concat string -- StringBuffer , 100000 iterations");
resultString = "";
StringBuffer sbuf = new StringBuffer();
startTimer();
for (int i = 0; i < 100000; i++) {
    sbuf.append(sourceString);
}
endTimer();
resultString = sbuf.toString();
System.out.println("Result string length: " + resultString.length());
timerResults();
System.out.println("\n\n");
```

# WORKING WITH STRINGS

Part 6

```
System.out.println("Concat string -- StringBuffer 32, 100000 iterations");
resultString = "";
sbuf = new StringBuffer(32);
startTimer();
for (int i = 0; i < 100000; i++) {
    sbuf.append(sourceString);
}
endTimer();
resultString = sbuf.toString();
System.out.println("Result string length: " + resultString.length());
timerResults();
System.out.println("\n\n");
```

# WORKING WITH STRINGS

Part 7

```
System.out.println("Concat string -- StringBuilder , 100000 iterations");
resultString = "";
StringBuilder sbuild = new StringBuilder();
startTimer();
for (int i = 0; i < 100000; i++) {
    sbuild.append(sourceString);
}
endTimer();
resultString = sbuild.toString();
System.out.println("Result string length: " + resultString.length());
timerResults();
System.out.println("\n\n");
```

# EXERCISE A030

- Run the program in the 7 slides before and explain the result

# GARBAGE COLLECTOR

Part 1

```
package de.porsche;

import java.util.Date;

class Main {

    private Thread thread;
    private boolean running;

    public Main() {
        running = true;
        thread = new Thread(() -> {
            while (running) {
                System.out.println(getTime() + "\t" + "Hello from the Class");
                Main.waitForSecs(10);
            }
        });
        thread.start();
    }

    static String getTime() {
        Date date = new Date();
        return date.toString();
    }

    static void waitForSecs(int secs) {
        try {
            Thread.sleep(secs * 1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

# GARBAGE COLLECTOR

```
@Override  
protected void finalize() throws Throwable {  
    running = false;  
    thread.interrupt();  
    thread.join();  
    System.out.println("Class is finished");  
}  
  
String getTime() {  
    String retVal = "";  
    Date date = new Date();  
    long timeInMillis = date.getTime();  
  
    // Calculate the hours, minutes, and milliseconds  
    int hours = (int) date.getHours();  
    int minutes = (int) ((timeInMillis / (1000 * 60)) % 60);  
    int seconds = (int) ((timeInMillis / 1000) % 60);  
    int milliseconds = (int) (timeInMillis % 1000);  
  
    retVal = String.format("%02d", hours) + ":" + String.format("%02d", minutes) + ":"  
        + String.format("%02d", seconds) + "." + String.format("%03d", milliseconds);  
  
    return retVal;  
}
```

# GARBAGE COLLECTOR

Part 3

```
static void waitForSecs( int secs) {  
    try {  
        Thread.sleep(secs * 1000);  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}  
  
public static void main(String[] args) {  
    Main instance = new Main();  
    System.out.println("After instantiation -> Start waiting for 30 seconds.");  
    Main.waitForSecs(30);  
    System.out.println("Wait is over.");  
    instance = null; // Collect the object for garbage collection  
    System.out.println("After setting reference variable to null -> Start waiting for 30 seconds.");  
    Main.waitForSecs(30);  
    System.out.println("Wait is over.");  
    System.gc();  
    System.out.println("After calling Garbage Collector explicitly -> Start waiting for 30 seconds.");  
    Main.waitForSecs(30);  
    System.out.println("Wait is over.");  
}
```

# EXERCISE A040

- Run the program in the 3 slides before and explain the result

# COMMAND LINE PARAMETERS 2

Part 1

- The static method `Integer.parseInt(<String>)` returns the integer value of the String, or, if String is no integer, throws an exception
- Examples:

String parameter for <code>Integer.parseInt</code>	Result
"34"	Integer value: 34
"3987"	Integer value: 3987
"Vier"	Throws an exception

# EXERCISE A050

- Write a program, which prints out the command line parameters in reverse order
- If the parameter is a valid integer, then add 10 to it before printing it out

# REVERSE POLISH NOTATION (UPN)

- Example :  $3 + 4 * 5 = 23$

In RPN it looks like this: 3 4 5 \* +

RPN calculations work with a stack, which growth upwards. The rules for processing expressions are these:

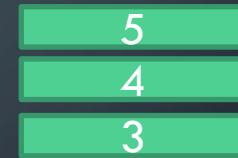
1. If parameter is a number push it onto the stack into the top position
2. If parameter is an operator, pull the required number from the stack, do the calculation and push the result back onto the stack.

# REVERSE POLISH NOTATION (UPN)

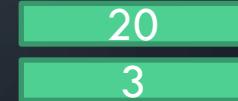
- Example : 3 4 5 \* +

1. Push 3
2. Push 4
3. Push 5

}



4. Pull 5 and Pull 4, multiply them and push 20 back on the stack



5. Pull 20 and Pull 3, add them, and push 23 onto the stack



# EXERCISE A060

- Write a reverse polish notation calculator.
- Take the input from the command line parameters
- Assume that the input is correct – no error checking there
- The last step of the program is to print out the top of the stack (result)

# UNIT TESTING

- Usually white-box testing
- Usually written by the developer, who wrote( or will write) code under test
- No dependency to other parts of the system
- They should run fast
- They should cover all of the code in the unit under test
- Each test should only test one thing and one thing only

# UNIT TESTING

YES, I know of projects, where the amount of test code by far exceeded the production code!

# UNIT TESTING

- There are two approaches to unit testing:
  1. Write the test code after the production code is written.  
When the code to test already exists then there is no other option.
  2. Let the tests drive your development (TDD): Write tests first and then write as little as possible code to have all the tests pass.

# UNIT TESTING

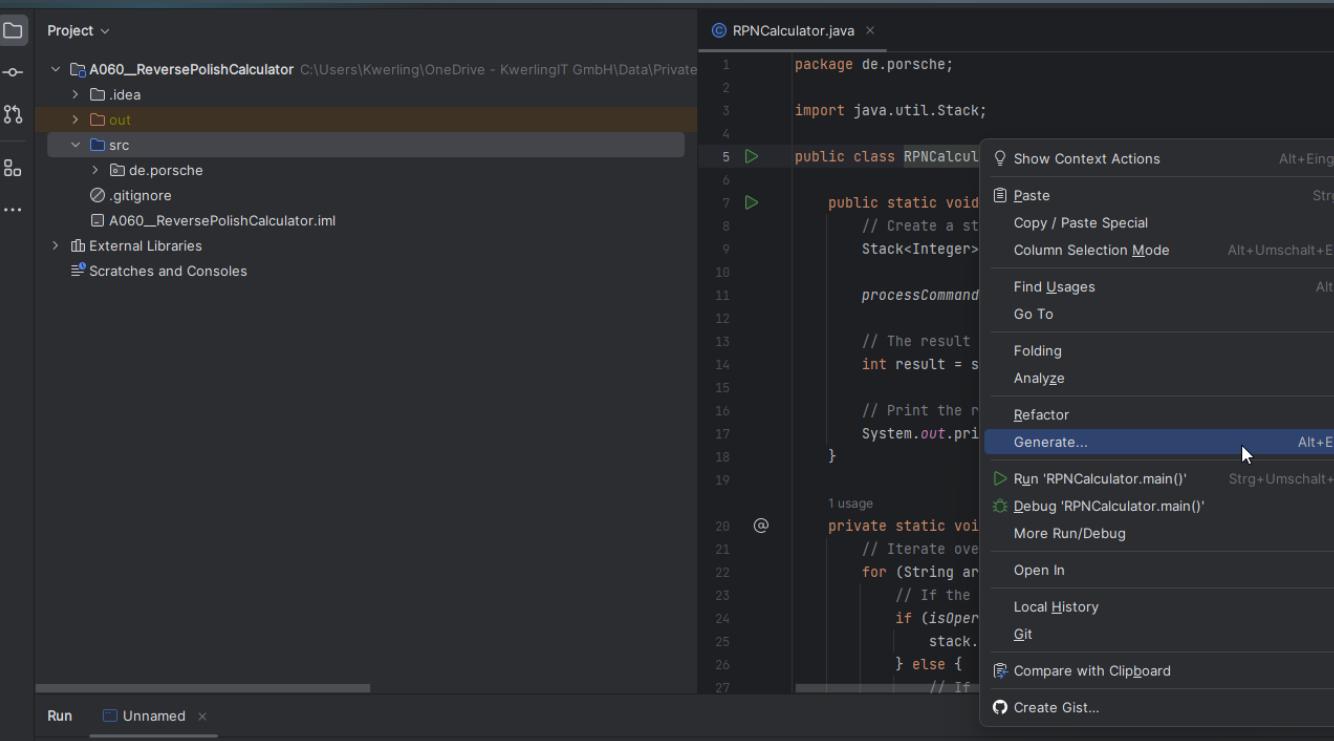
- Advantages to gain through unit testing:
  1. Improved code quality
  2. Tests document the expected behavior of the code
  3. Confidence in your code
  4. Easier refactoring or enhancing of code: Assured that code still behaves as it did before the changes.

# UNIT TESTING

- Advantages to gain through unit testing:
  1. Improved code quality
  2. Tests document the expected behavior of the code
  3. Confidence in your code
  4. Easier refactoring or enhancing of code: Assured that code still behaves as it did before the changes.

# UNIT TESTING - INTELLIJ

Part 1



The screenshot shows the IntelliJ IDEA interface. On the left is the Project tool window, displaying a project named "A060\_ReversePolishCalculator" with a "src" folder containing "de.porsche" and "A060\_ReversePolishCalculator.iml". The right side is the Editor pane, showing the file "RPNCalculator.java" with the following code:

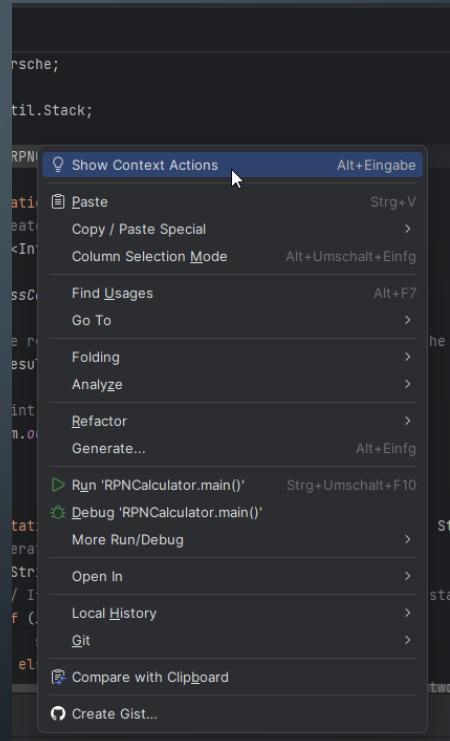
```
package de.porsche;
import java.util.Stack;

public class RPNCalculator {
    public static void processCommand() {
        Stack<Integer> stack = new Stack<Integer>();
        int result = 0;
        System.out.println(result);
    }

    private static void for (String arg) {
        if (isOper(stack)) {
            } else {
                // If
        }
    }
}
```

A context menu is open over the code, with the "Generate..." option highlighted.

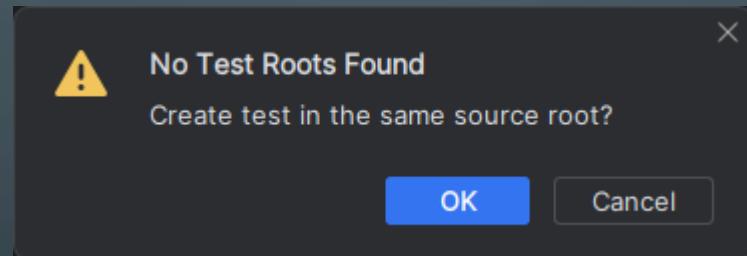
OR



The screenshot shows the IntelliJ IDEA interface with the Editor pane. A context menu is open over the code, with the "Run 'RPNCalculator.main()'" option highlighted.

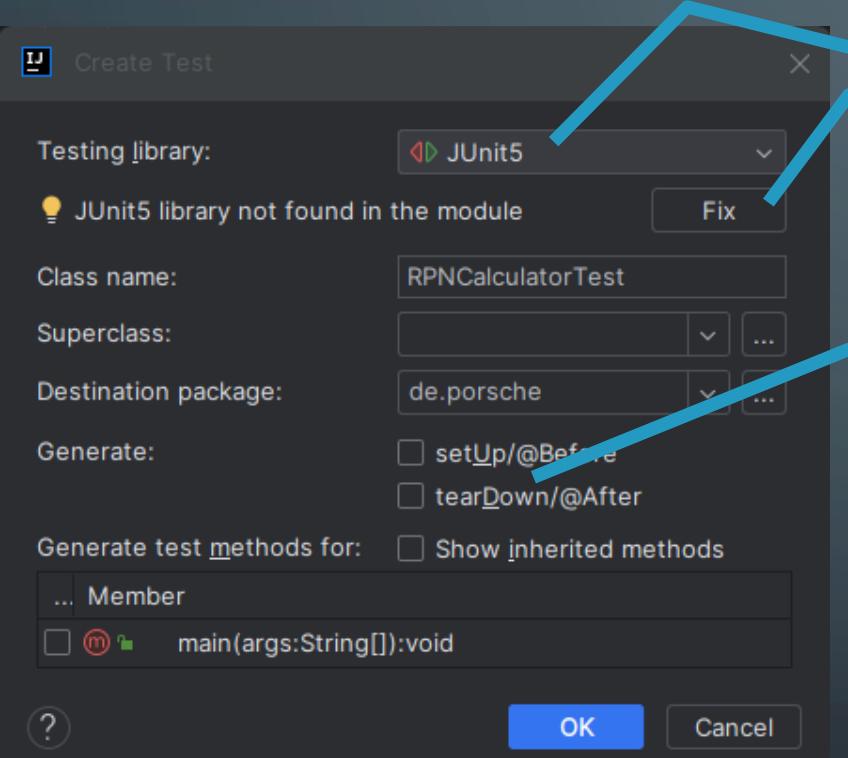
# UNIT TESTING - INTELLIJ

Part 2



# UNIT TESTING - INTELLIJ

Part 3



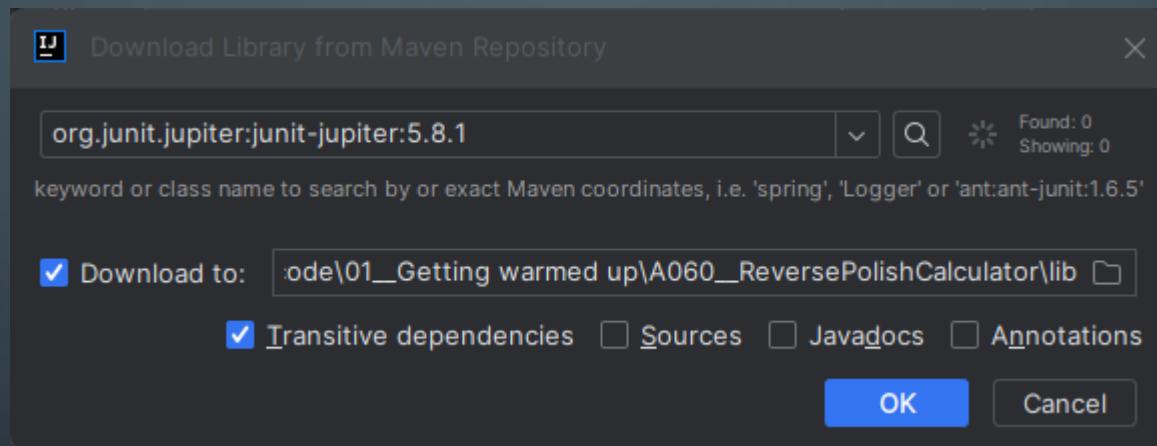
Use Junit 5

Fix it (see next slide)

Generate setup / teardown

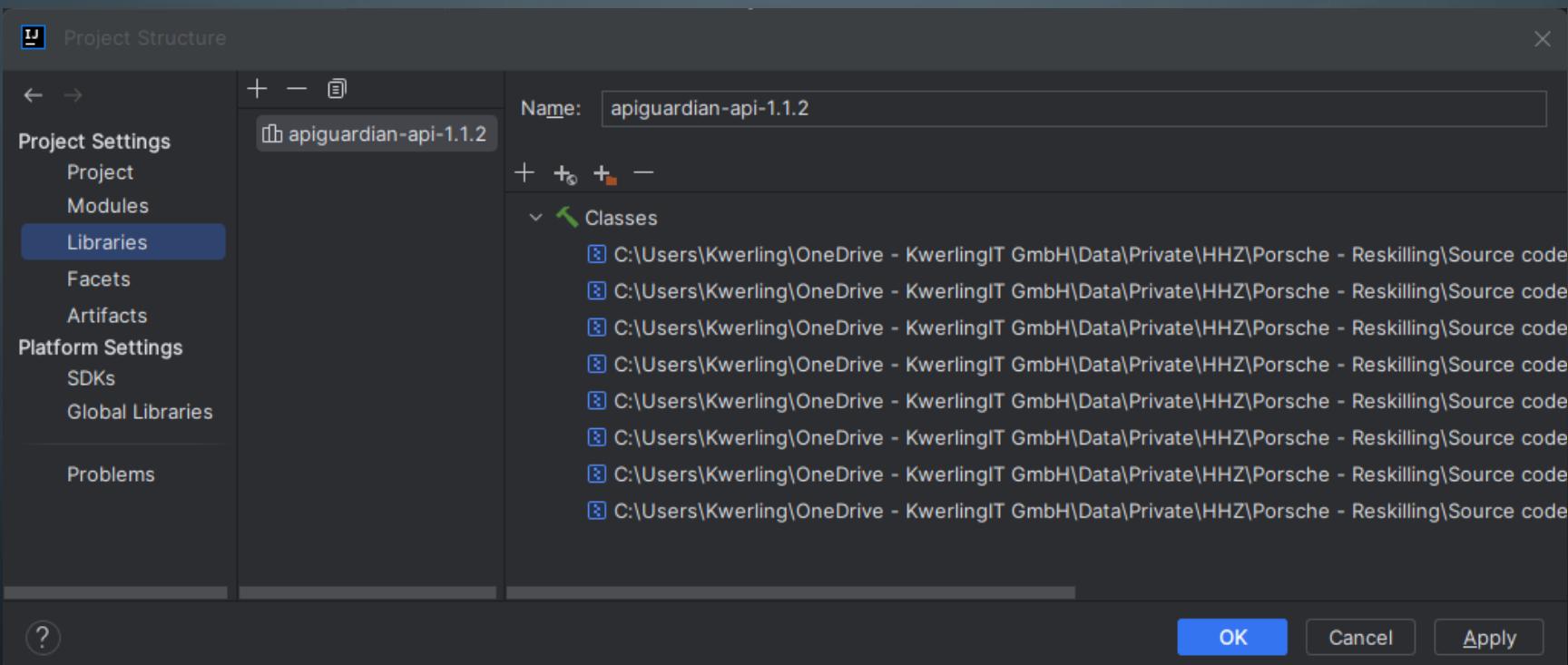
# UNIT TESTING - INTELLIJ

Part 4



# UNIT TESTING - INTELLIJ

Part 5



# UNIT TESTING - INTELLIJ

Part 5

```
package de.porsche;

import static org.junit.jupiter.api.Assertions.*;

class RPNCcalculatorTest {

    @org.junit.jupiter.api.BeforeEach
    void setUp() {
    }

    @org.junit.jupiter.api.AfterEach
    void tearDown() {
    }

    @org.junit.jupiter.api.Test
    void someTest() {
    }
}
```

# UNIT TESTING

- There are a number of Assert-Methods supporting the testing. Examples:

Assertion	Explanation
<code>assertEquals( exp, act [, optionalMsg])</code>	<code>exp</code> is required to be the same as <code>act</code>
<code>assertTrue( cond [, optionalMsg])</code>	<code>cond</code> is required to be true
<code>assertNotNull( obj [, optionalMsg])</code>	<code>obj</code> is required to not be null

There are a number of more Assert-Methods, but these 3 should get you going.

## EXERCISE A060, PART 2

- Write some unit tests for the different methods of RPNCalculator.
- In order to do so it is much better to have the methods of RPNCalculator as non-static – Refactor, if you have the methods all static as it is in the code shown a couple of slides before.

# MODULE B

# NETWORKING

# CONTENT

- Sockets
- Rest(ful) APIs
- Message Bus

# SOCKETS

- Communication end points
- Logical constructs – there is no physical representation
- The OS provides a little over 65500 Sockets (why that number?)
- Well known sockets are numbered up to 1024 (again, why that number?)

# SOCKETS

- Examples for well known sockets:

Socket number	Protocol served
20	FTP
22	SSH
25	SMTP
53	DNS
80	HTTP
443	HTTPS
...	

# SOCKETS

- For communicating over a network one needs two types of programs:

1. Server

This program waits for connections and services them

2. Client

This program initiates the connection and uses the service of the server program

# SOCKETS

Part 1

- Single Connection Server example
- How it works:
  - Starts
  - Listens on the server socket
  - When connected
    - Reads incoming message
    - Writes message back
  - Closes socket

```
public class EchoServer {  
    private ServerSocket serverSocket;  
    private Socket connectionSocket;  
    private PrintWriter out;  
    private BufferedReader in;  
  
    public void start(int port) throws IOException {  
        serverSocket = new ServerSocket(port);  
        connectionSocket = serverSocket.accept();  
        out = new PrintWriter(connectionSocket.getOutputStream(), true);  
        in = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));  
        String str = in.readLine();  
        out.println("Echo: " + str);  
    }  
  
    public void stop() throws IOException {  
        in.close();  
        out.close();  
        connectionSocket.close();  
        serverSocket.close();  
    }  
}
```

# SOCKETS

Part 2

- Here is the Main class for the server
- It instantiates the EchoServer class
- It starts listening on port 7788
- When server is finished it stops all server activities

```
package de.porsche;  
  
import java.io.IOException;  
  
public class Main {  
  
    public static void main(String[] args) {  
        EchoServer server=new EchoServer();  
        try {  
            server.start(7788);  
            server.stop();  
        } catch (IOException e) {  
            System.err.println(e.getMessage());  
        }  
    }  
}
```

# SOCKETS

Part 1

- Single Connection Client example
- How it works:
  - Build up connection to server
  - Sends its message
  - Receives back answer from server
  - Closes connection

```
public class EchoClient {  
    private Socket clientSocket;  
    private PrintWriter out;  
    private BufferedReader in;  
  
    public void startConnection(String ip, int port) throws IOException {  
        clientSocket = new Socket(ip, port);  
        out = new PrintWriter(clientSocket.getOutputStream(), true);  
        in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));  
        System.out.println("Server is listening at port:" + clientSocket.getPort());  
    }  
  
    public String sendMessage(String msg) throws IOException {  
        out.println(msg);  
        String resp = in.readLine();  
        return resp;  
    }  
  
    public void stopConnection() throws IOException {  
        in.close();  
        out.close();  
        clientSocket.close();  
    }  
}
```

# SOCKETS

Part 2

- Here is the Main class for the client
- It sets the port to 7788 on the local machine and establishes the connection
- The answer received from the server is printed

```
package de.porsche;

import java.io.IOException;

public class Main {

    public static void main(String[] args) {
        EchoClient client = new EchoClient();
        try {
            client.startConnection("127.0.0.1", 7788);
            String response = client.sendMessage("hello server");
            System.out.println(response);
            client.stopConnection();
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }
}
```

# SOCKETS

- How can a server serve multiple connections at the same time?
- How to know which socket to pick for a server? How to verify if it is free?
- How to connect to a server which ip-address & socket I do not know?

# EXERCISES

- Take the server & client code and get it to work on your computer
- Enhance the server, so that it will not end processing after just one connection.  
Maybe include the possibility of a command, which, when sent will stop the server.
- Use your client to connect to a server on another computer.

# RESTFUL APIs – WEB SERVICES

- In the following slides we look at REST clients.
- Reasoning for it:
  - It is by far more likely that you will have to connect to a REST API than that you have to write one
  - One can do REST APIs in Java, but often another technology is used (Ex.: JavaScript)
  - Even in Java there are many libraries / tools to do REST APIs

# RESTFUL APIs – WEB SERVICES

- As Restful API for all our clients we use  
<https://postman-echo.com>
- This is a freely available REST API on the internet  
The examples use the Http-Client, which is contained in the JDK. Again, there are many other alternatives available.

# RESTFUL APIs – WEB SERVICES

- What is Rest?
  - Http based
  - Stateless
  - Rest request:
    - Http-Verb: GET, POST, PUT, DELETE (to name the more common ones)
    - Header: Accept field (Response type), Authorization ...
    - Path to the resource (Ex.: mysrv.com/Customers/9/Orders/17)
    - Message body (optional): Additional data

# RESTFUL APIs – WEB SERVICES

- How to do a Http-Request?
  - HttpRequest uses the builder pattern
  - One needs to make use of multiple methods of the class the construct the request:

```
HttpRequest request = HttpRequest.newBuilder()  
    .uri(new URI("https://postman-echo.com/get"))  
    .GET()  
    .build();
```

# RESTFUL APIs – WEB SERVICES

- Execute the request:
  - It is the `HttpClient`, which executes the request and provides the response:

```
HttpClient client = HttpClient.newHttpClient();
HttpResponse<String> response = client.send(request,
BodyHandlers.ofString());
```

- The result is found in `response.body`

# EXERCISE B030

- Write a Java program, which can do the following:
  1. Retrieve the response from the Postman-Echo GET web service and print it
  2. Retrieve and print your IP-address from the Postman-Echo IP web service
  3. Use the Postman-Echo POST web service. Read up on HttpClient and the Postman-Echo web site in order to write the code.

# MESSAGE BUS

- Assumptions for the following slides:

- Docker Desktop is installed on your computer

With this the command line interface for docker is included, too. All the needed docker activities in this presentation can be executed this way.

Please refer to Appendix A for instructions on how to install Docker Desktop.

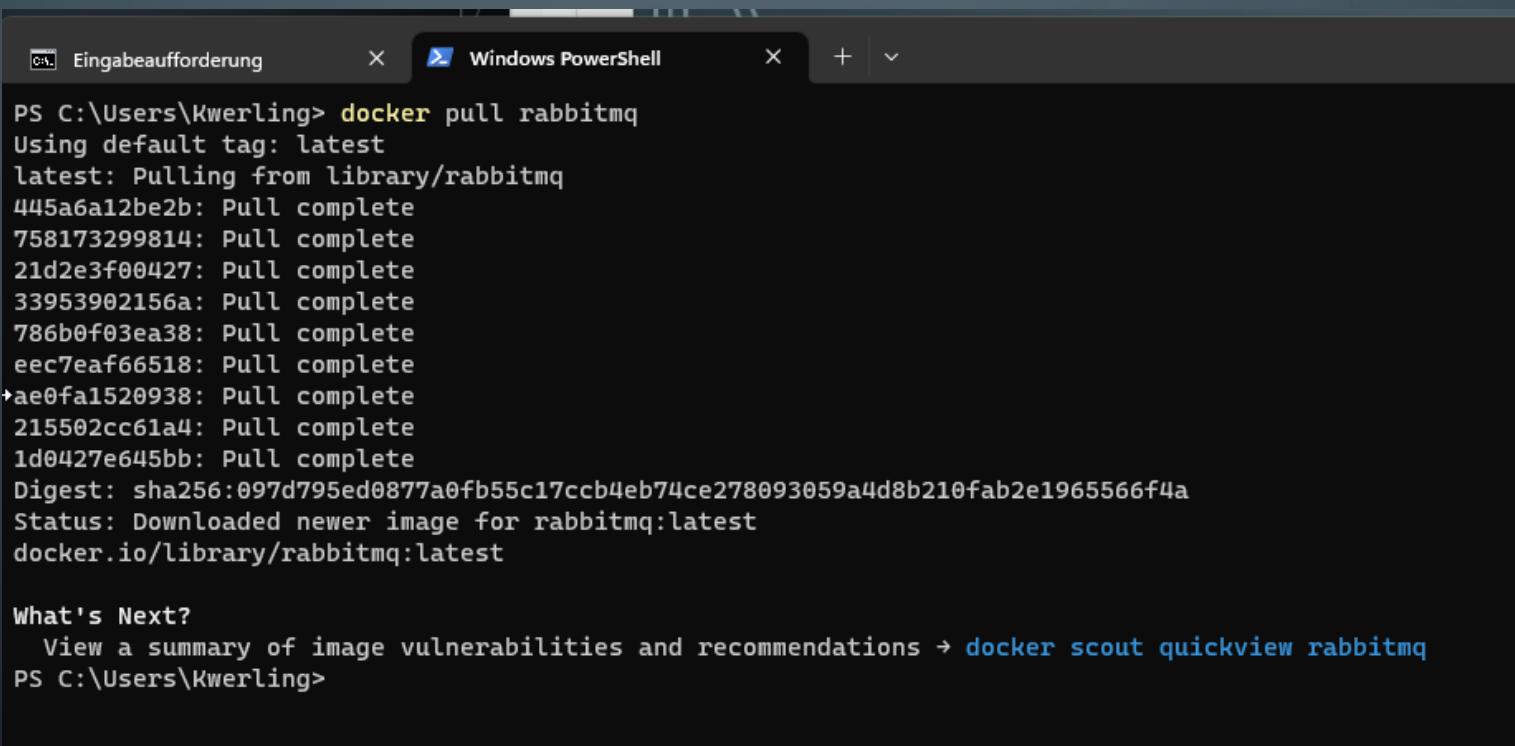
# MESSAGE BUS

- Install RabbitMQ on your machine

Command line:      `docker pull rabbitmq`

- This will take a little while

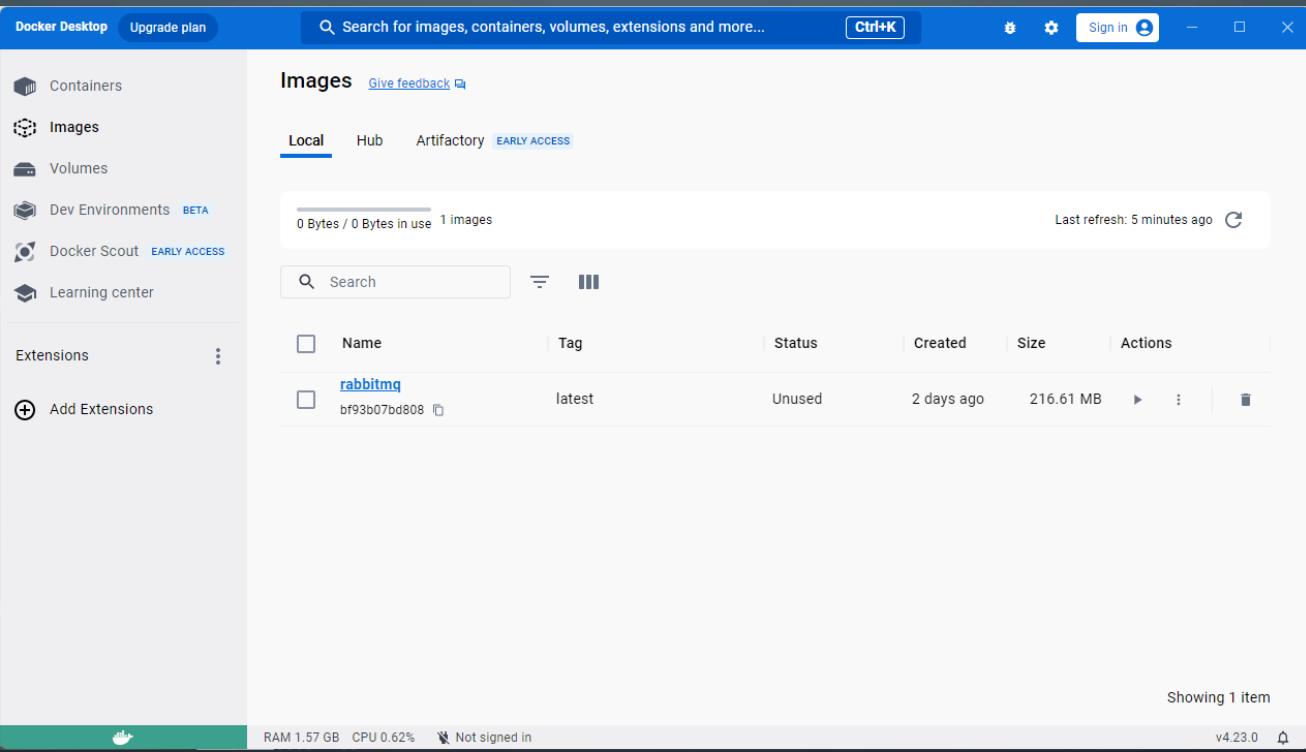
# MESSAGE BUS



```
PS C:\Users\Kwerling> docker pull rabbitmq
Using default tag: latest
latest: Pulling from library/rabbitmq
445a6a12be2b: Pull complete
758173299814: Pull complete
21d2e3f00427: Pull complete
33953902156a: Pull complete
786b0f03ea38: Pull complete
eec7eaf66518: Pull complete
ae0fa1520938: Pull complete
215502cc61a4: Pull complete
1d0427e645bb: Pull complete
Digest: sha256:097d795ed0877a0fb55c17ccb4eb74ce278093059a4d8b210fab2e1965566f4a
Status: Downloaded newer image for rabbitmq:latest
docker.io/library/rabbitmq:latest

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview rabbitmq
PS C:\Users\Kwerling>
```

# MESSAGE BUS



# MESSAGE BUS

- Features:

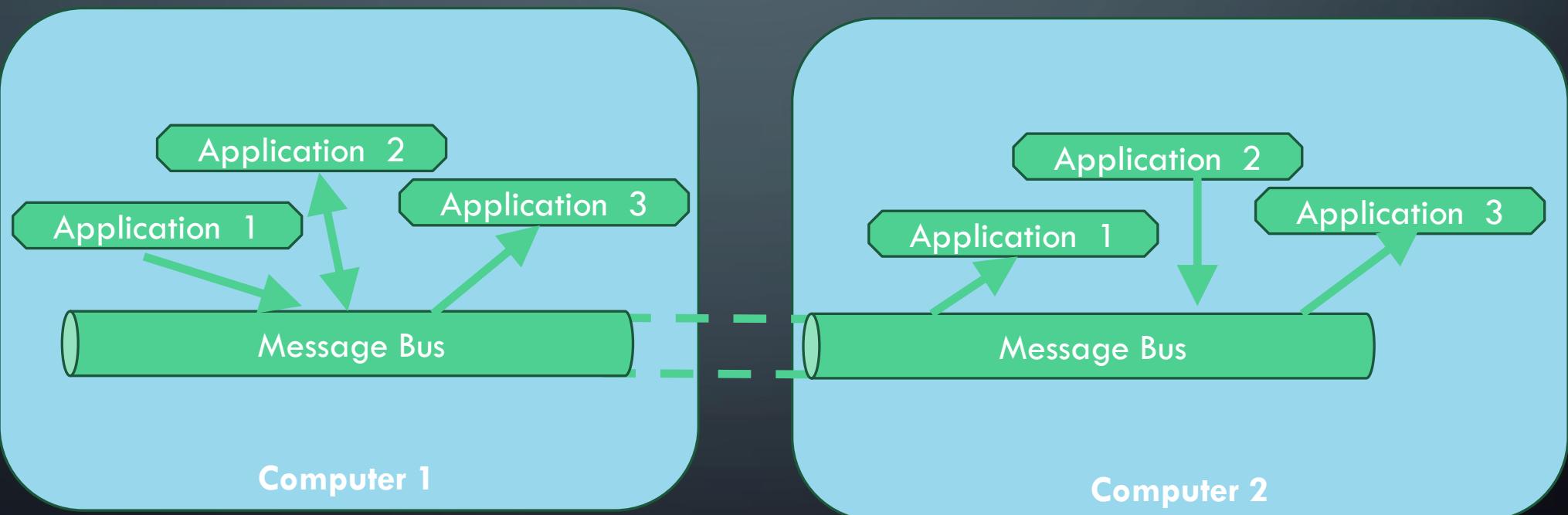
- Applications communicate through messages via the bus
- The bus architecture contains a message broker as central element
- The broker receives, routes, and delivers the messages
- Because of the broker the message sender / receiver are only loosely coupled
- The communication happens asynchronously

# MESSAGE BUS

<b>Either</b>	<b>Or</b>	<b>Or</b>
Sender	Producer	Publisher
Receiver	Consumer	Subscriber

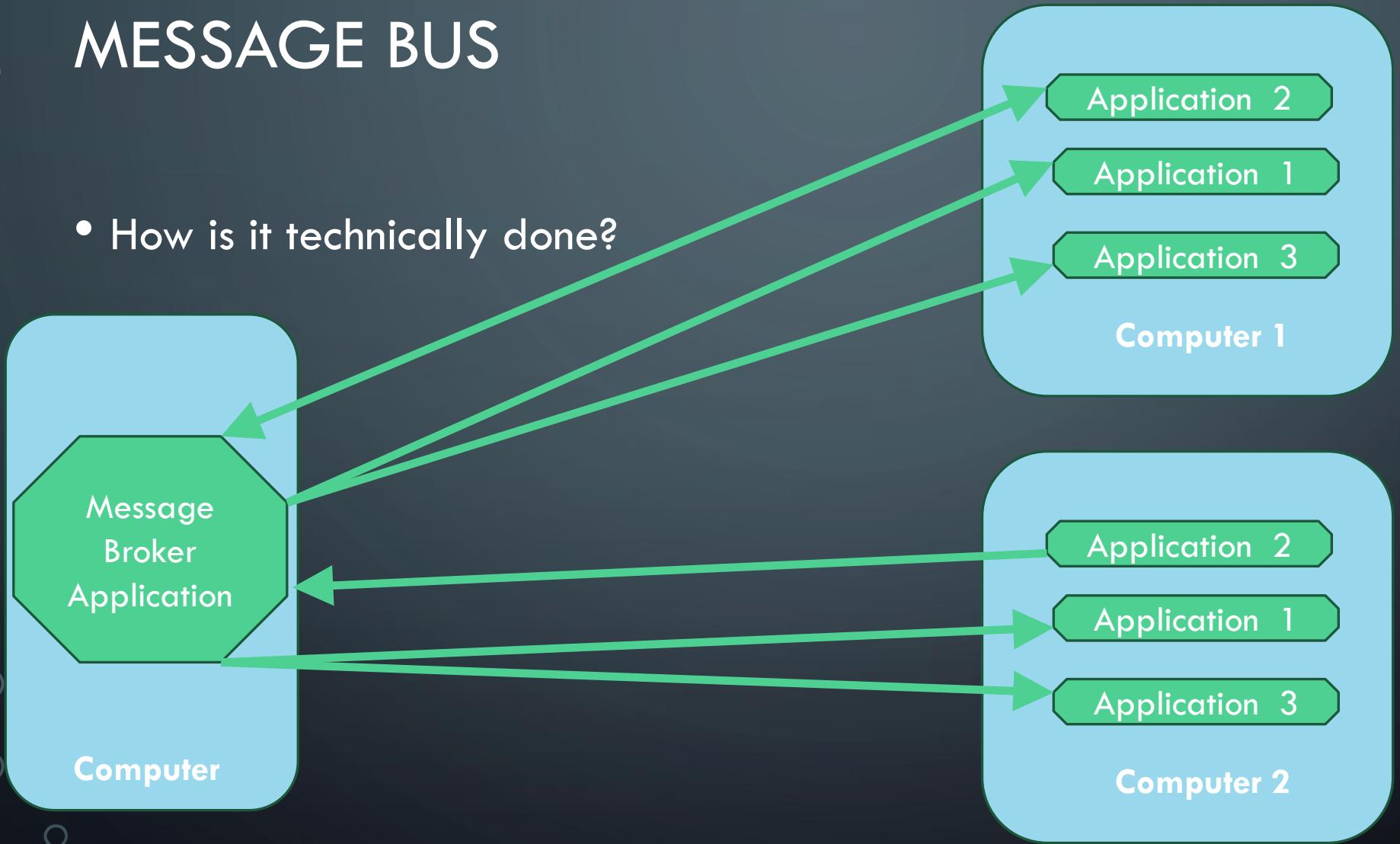
# MESSAGE BUS

- How to picture a message bus?



# MESSAGE BUS

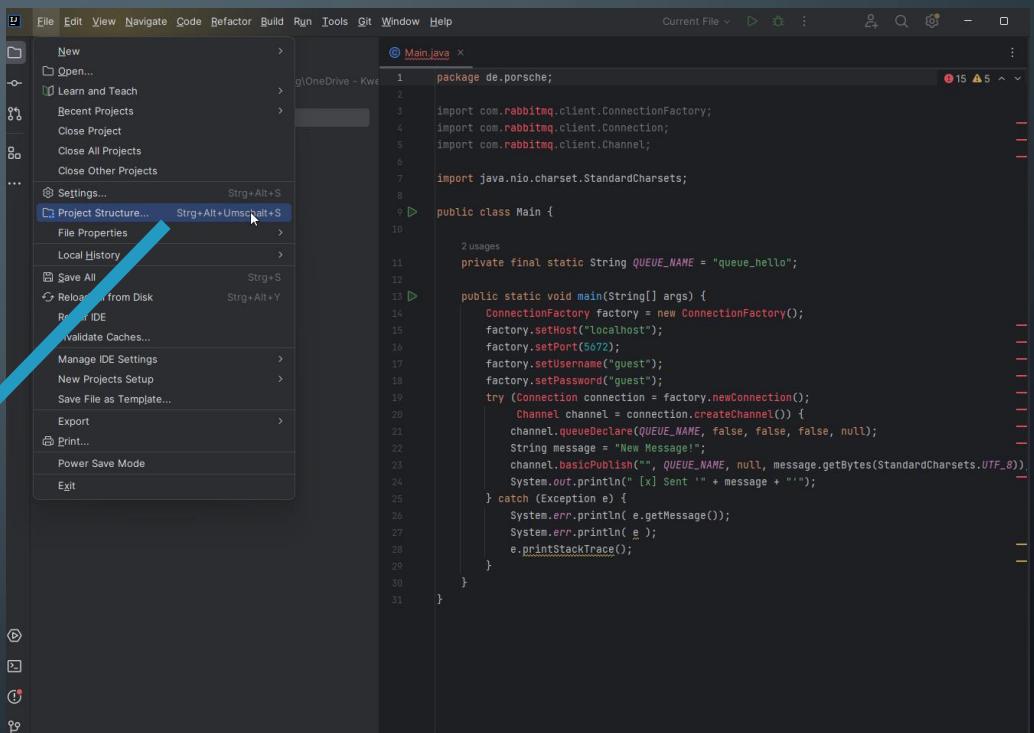
- How is it technically done?



# MESSAGE BUS

- In order to run the RabbitMQ Consumer and Producer code, the RabbitMQ java library needs to be added to the project.

File -> Project Structure



```
package de.porsche;

import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.Channel;

import java.nio.charset.StandardCharsets;

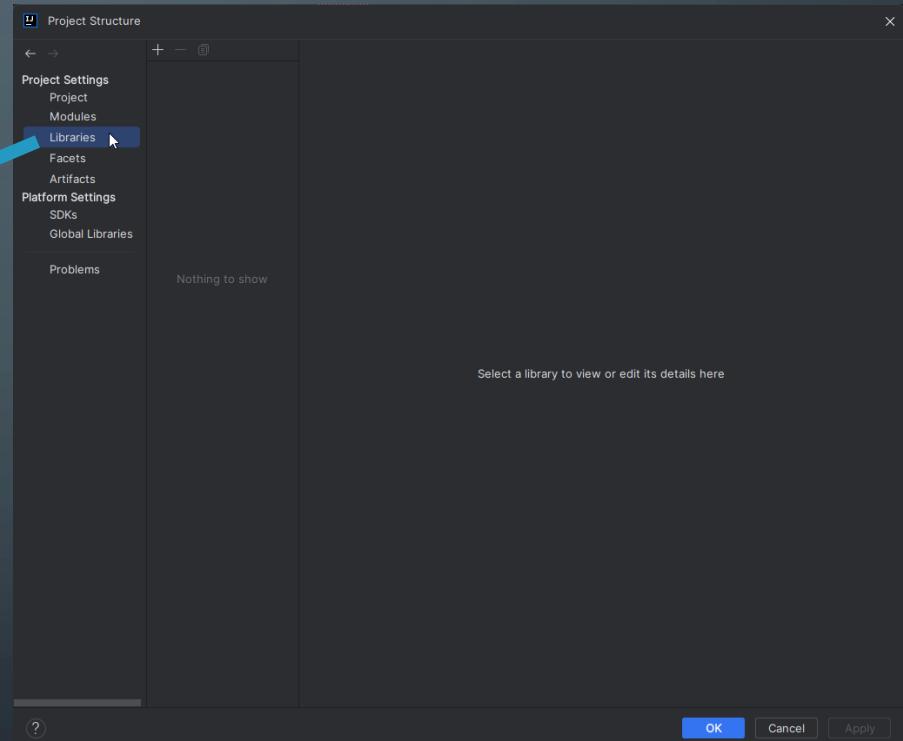
public class Main {

    private final static String QUEUE_NAME = "queue_hello";

    public static void main(String[] args) {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        factory.setPort(5672);
        factory.setUsername("guest");
        factory.setPassword("guest");
        try (Connection connection = factory.newConnection()) {
            Channel channel = connection.createChannel();
            channel.queueDeclare("", QUEUE_NAME, false, false, null);
            String message = "New Message!";
            channel.basicPublish("", QUEUE_NAME, null, message.getBytes(StandardCharsets.UTF_8));
            System.out.println(" [x] Sent " + message + "!");
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.err.println( e );
            e.printStackTrace();
        }
    }
}
```

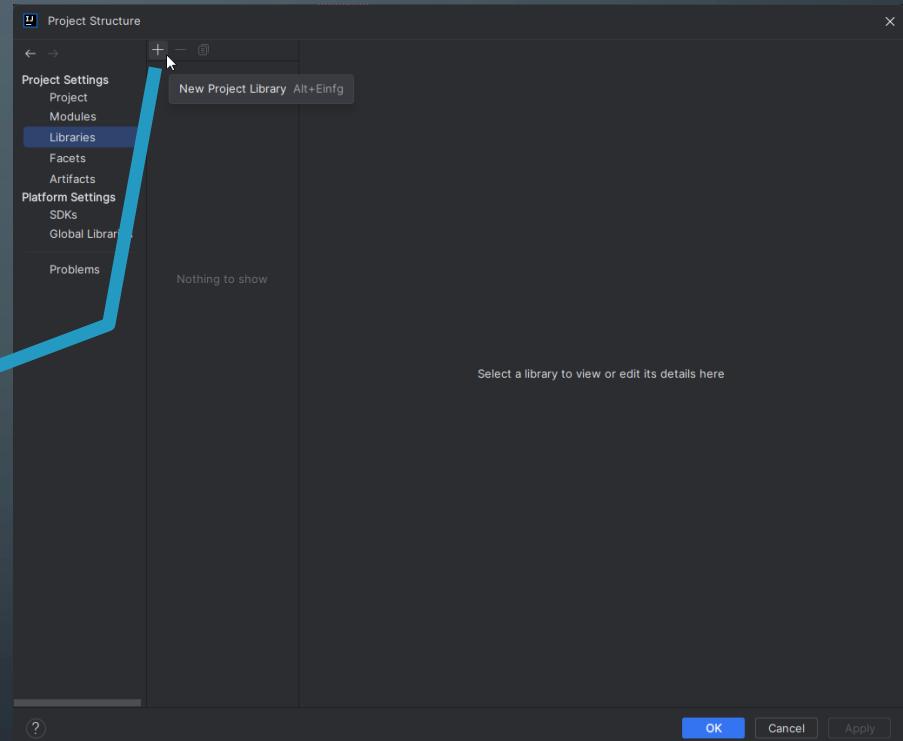
# MESSAGE BUS

Libraries



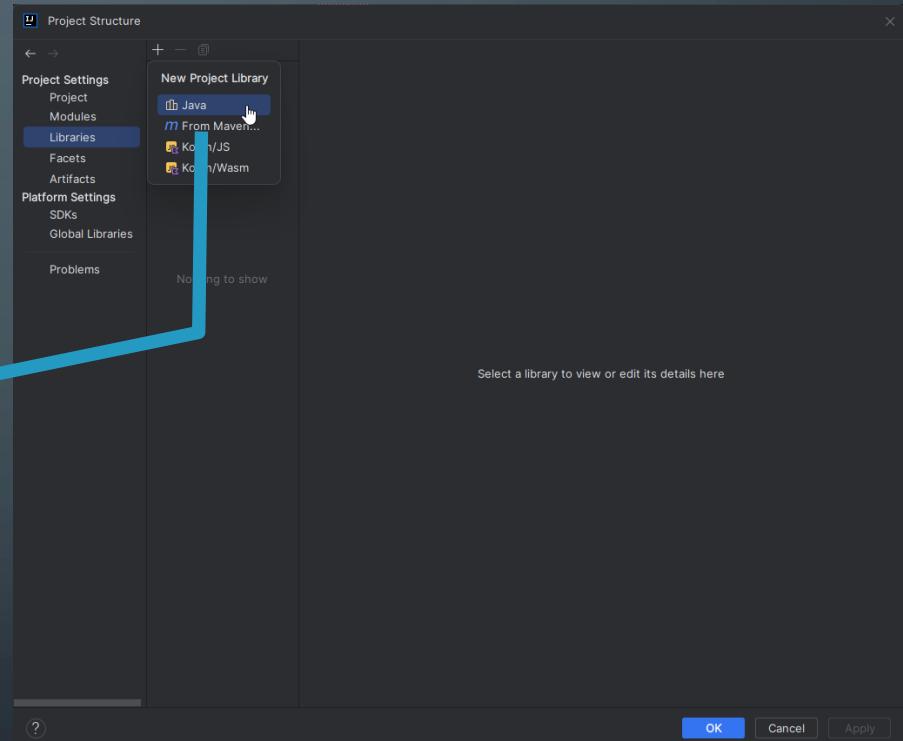
# MESSAGE BUS

Add ( + ) button



# MESSAGE BUS

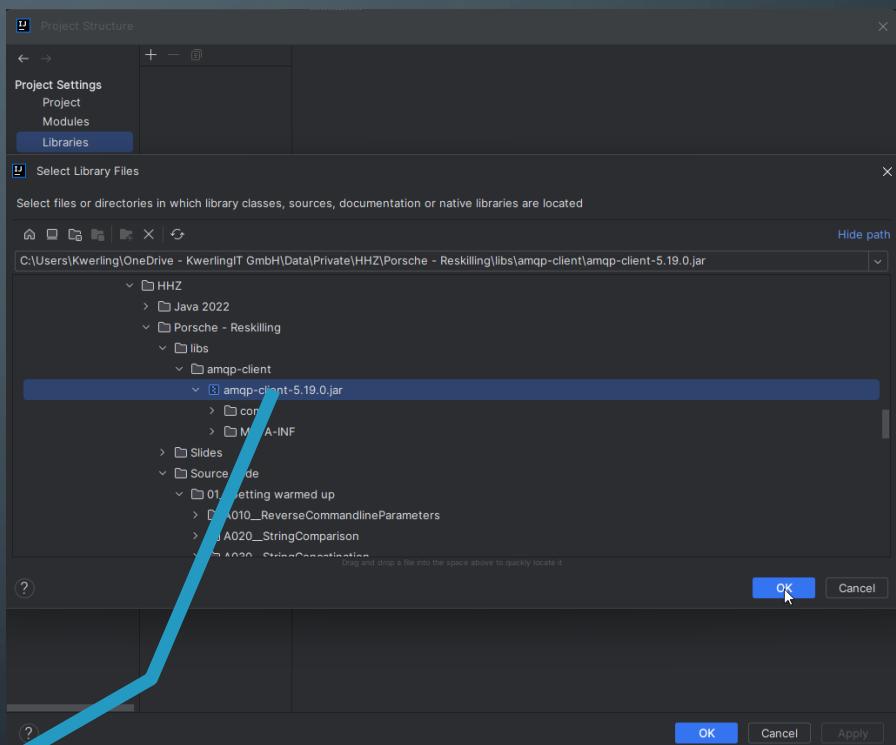
Choose Java library



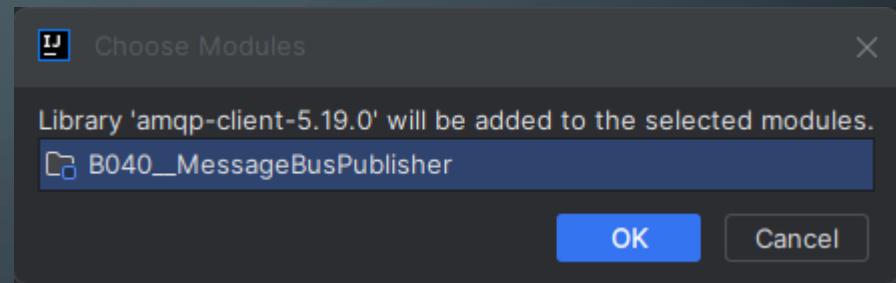
# MESSAGE BUS

- The repository contains a libs directory on the same level as the source code directory.
- In that directory, there is the amqp directory
- In that directory is the library needed

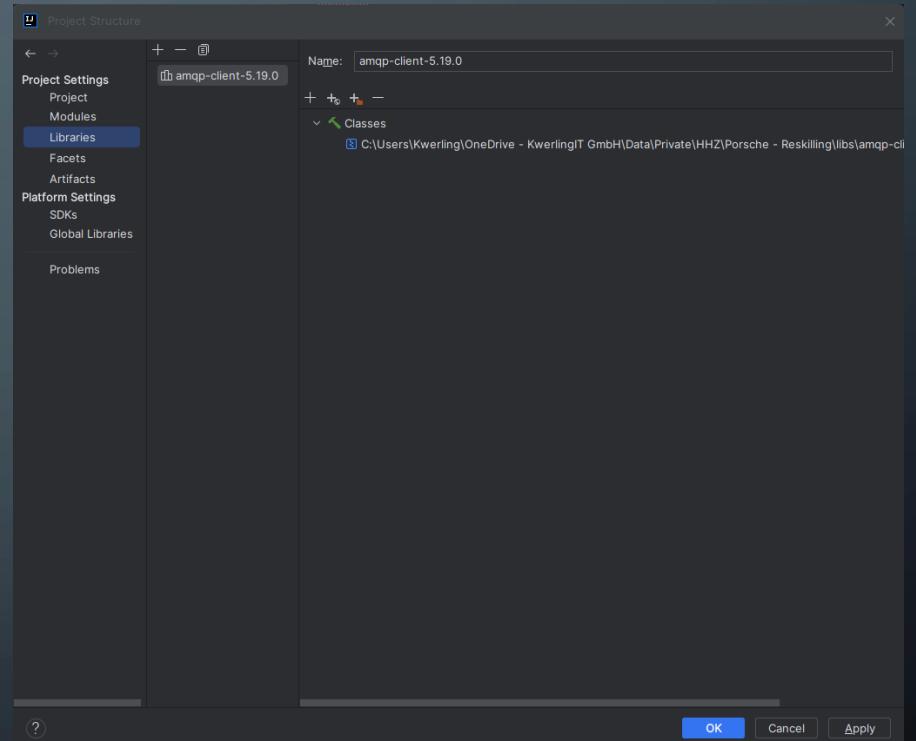
Libraries



# MESSAGE BUS

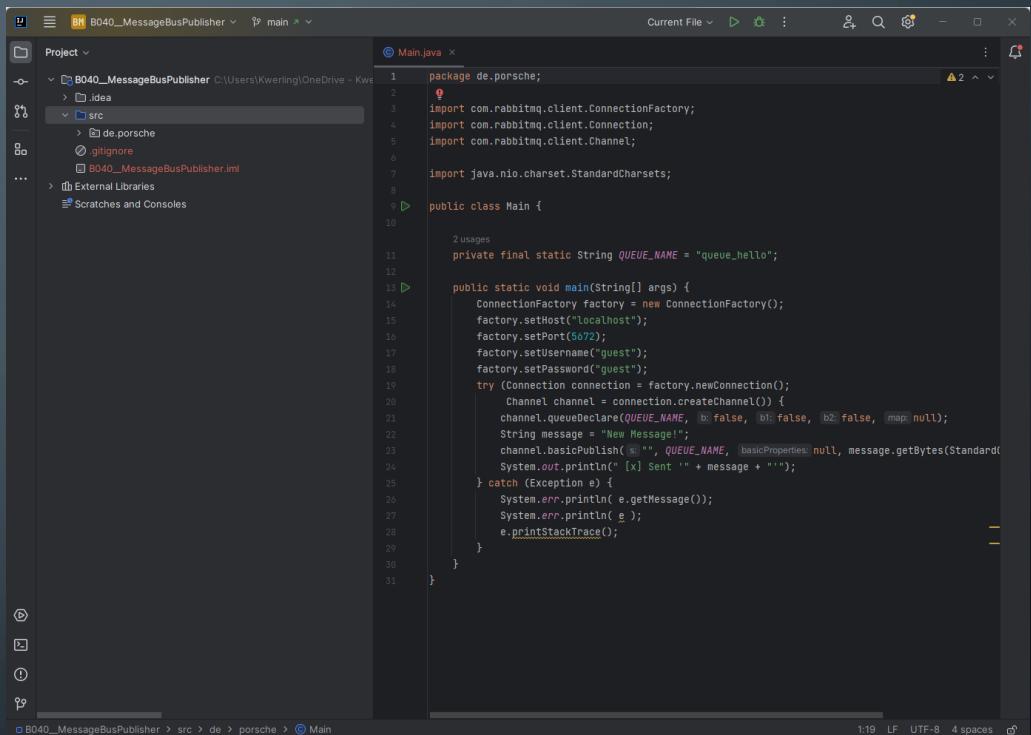


# MESSAGE BUS



# MESSAGE BUS

- All classes and methods should be known now.



The screenshot shows a Java application named "B040\_MessageBusPublisher" in an IDE. The project structure includes a "src" folder containing a "de.porsche" package with a file named "Main.java". The code in Main.java is as follows:

```
package de.porsche;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.Channel;
import java.nio.charset.StandardCharsets;

public class Main {

    private final static String QUEUE_NAME = "queue_hello";

    public static void main(String[] args) {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        factory.setPort(5672);
        factory.setUsername("guest");
        factory.setPassword("guest");
        try (Connection connection = factory.newConnection()) {
            Channel channel = connection.createChannel();
            channel.queueDeclare(QUEUE_NAME, false, false, false, null);
            String message = "New Message!";
            channel.basicPublish("", QUEUE_NAME, null, message.getBytes(StandardCharsets.UTF_8));
            System.out.println(" [x] Sent " + message);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.err.println(e);
            e.printStackTrace();
        }
    }
}
```

The code uses the RabbitMQ Java client library to declare a queue named "queue\_hello" and publish a message "New Message!" to it. The message bytes are obtained using StandardCharsets.UTF\_8.

# MESSAGE BUS

- First part of a simple publisher code.
- Every message is published into a queue.

```
package de.porsche;

import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.Channel;

import java.nio.charset.StandardCharsets;

public class Main {

    private final static String QUEUE_NAME = "queue_hello";
```

# MESSAGE BUS

1. Connect to the RabbitMQ broker
2. Authenticate
3. Connect to queue
4. Publish

```
public static void main(String[] args) {  
    ConnectionFactory factory = new ConnectionFactory();  
    factory.setHost("localhost");  
    factory.setPort(5672);  
    factory.setUsername("guest");  
    factory.setPassword("guest");  
    try (Connection connection = factory.newConnection()) {  
        Channel channel = connection.createChannel();  
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);  
        String message = "New Message!";  
        channel.basicPublish("", QUEUE_NAME, null, message.getBytes(StandardCharsets.UTF_8));  
        System.out.println("[x] Sent " + message + "");  
    } catch (Exception e) {  
        System.err.println( e.getMessage());  
        System.err.println( e );  
        e.printStackTrace();  
    }  
}
```

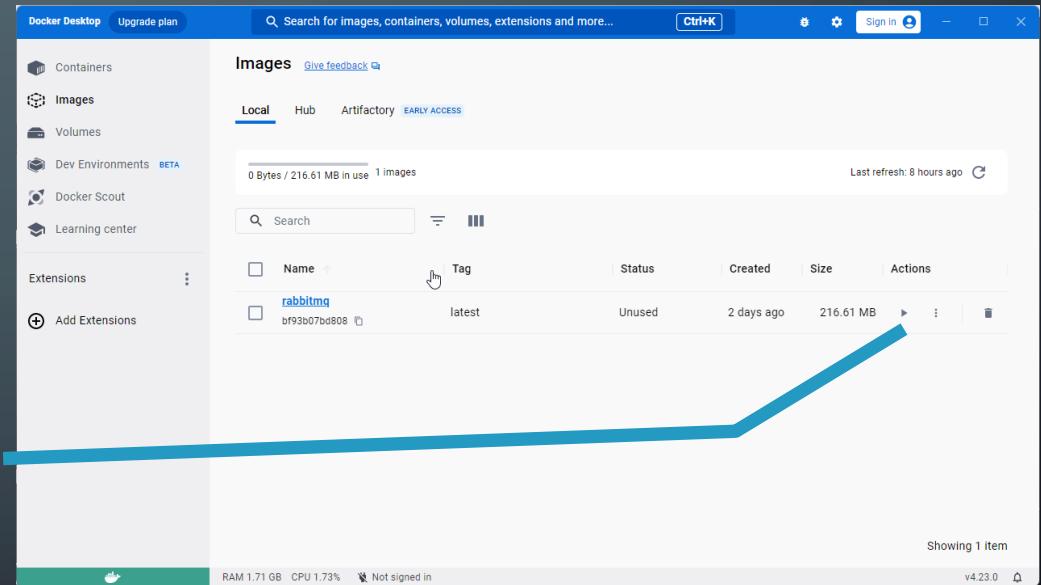
# MESSAGE BUS

- No compilation issues
- But when trying to run the error to the right appears
- Reason is, that there is a dependency on the slf4j libraries for logging
- Add them to the project

```
"C:\Program Files\OpenJDK\jdk-20.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.1\lib\native\jps-agent.jar" -Dfile.encoding=UTF-8 com.rabbitmq.clientConnectionFactory.<clinit>(ConnectionFactory.java:56)
at de.porsche.Main.main(Main.java:14)
Caused by: java.lang.ClassNotFoundException: org.slf4j.LoggerFactory
at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:521)
... 2 more
Process finished with exit code 1
```

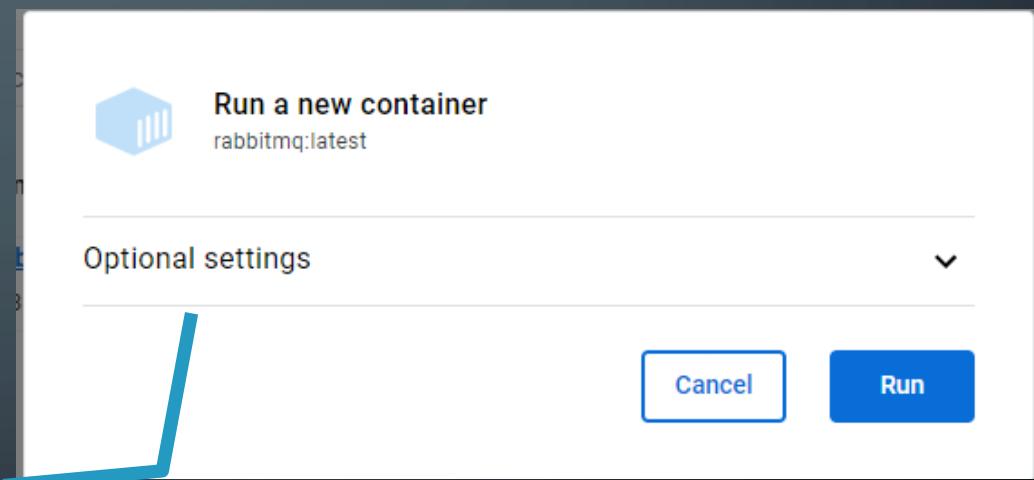
# MESSAGE BUS

- Now it is time to start RabbitMQ



Click the run button

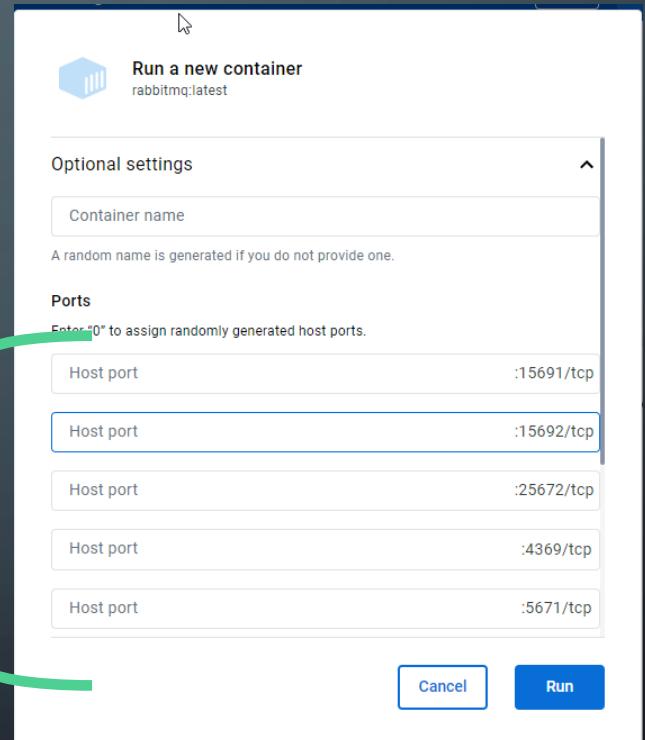
# MESSAGE BUS



Click on Optional settings

# MESSAGE BUS

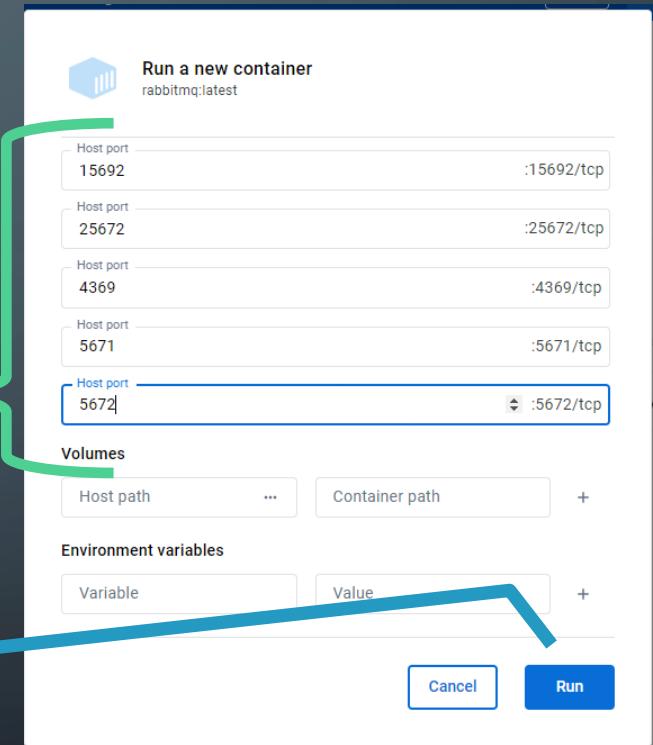
Enter the same local ports as RabbitMQ uses



# MESSAGE BUS

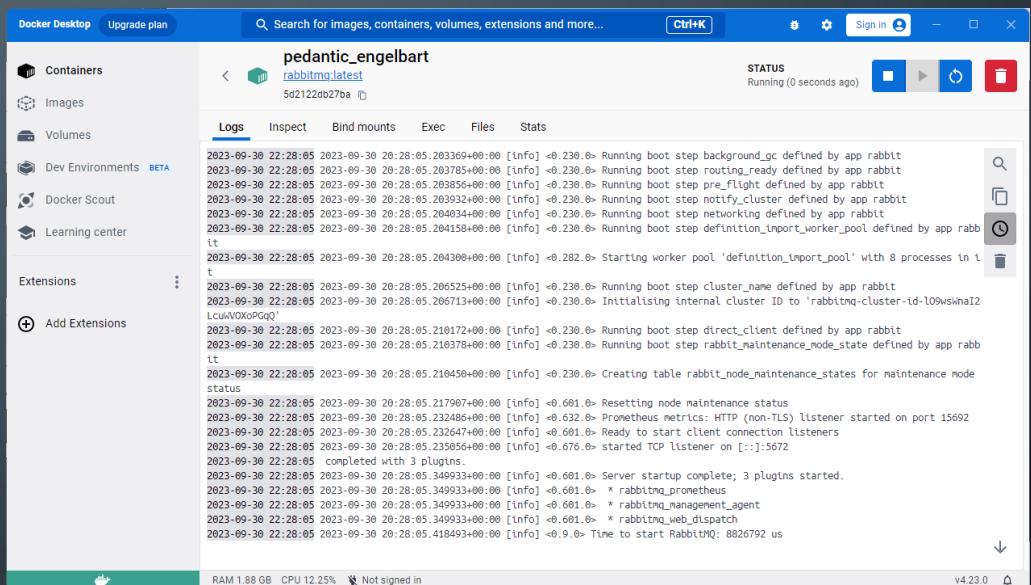
Continue – leave the other fields empty

Run



# MESSAGE BUS

- You'll see the log information from the RabbitMQ startup.
- CAREFUL: You might run into problems with your firewall -> allow access for the docker container



# MESSAGE BUS

- You should see this information when the program runs successfully

Process finished with exit code 0

# MESSAGE BUS

- Here is the first part of the Consumer code.
- The message queue of the consumer must be spelled exactly the same was as the one used in the publisher

```
package de.porsche;

import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.Channel;

import java.nio.charset.StandardCharsets;

public class Main {

    2 usages
    private final static String QUEUE_NAME = "queue_hello";
```

# MESSAGE BUS

- This consumer runs forever

```
private final static String QUEUE_NAME = "queue_hello";

public static void main(String[] args) {
    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");
    factory.setPort(5672);
    factory.setUsername("guest");
    factory.setPassword("guest");
    try (Connection connection = factory.newConnection()) {
        Channel channel = connection.createChannel();
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);
        String message = "New Message!";
        channel.basicPublish("", QUEUE_NAME, null, message.getBytes(StandardCharsets.UTF_8));
        System.out.println("[x] Sent '" + message + "'");
    } catch (Exception e) {
        System.err.println(e.getMessage());
        System.err.println(e);
        e.printStackTrace();
    }
}
```

# EXERCISE: MESSAGE BUS

- Get both, consumer and producer up and running
- Refactor the producer so, that the message is taken from the command line
- Try to connect to a RabbitMQ broker of your peers
- Play a little with different queue names
- Try to setup a chat application with one of the other groups  
(see next slide: Getting keyboard input in console app)

# GETTING KEYBOARD INPUT IN CONSOLE APP

- Use the `Console` class to read keyboard input

```
package de.porsche;

import java.io.Console;

public class Main {
    public static void main(String[] args) {

        Console con = System.console();

        if( null == con ) {
            System.err.println("Console read access unavailable!");
        } else {
            System.out.print("Enter some user input: ");
            String input = System.console().readLine();
            System.out.printf("You entered: %s", input);
        }
    }
}
```

# GETTING KEYBOARD INPUT IN CONSOLE APP

- Most likely there will be no access to the console from within IntelliJ
- Goto the project directory and then to the `out\production\B060__ReadingKey  
boardInputInConsoleApp` directory
- Run `java -cp . de.porsche.Main`

```
"C:\Program Files\OpenJDK\jdk-20.0.2\bin\java.exe" "-javaagent:C:\Program Files'  
Console read access unavailable!  
I  
Process finished with exit code 0
```



# MODULE C DATABASES



# DATABASES – TABLE OF CONTENT

- Relational Databases
- SQL
- JDBC
- ORM
- KV-DB

# DATABASES

Feature	Relational / SQL Database	No(n)-SQL Database
Data type	Structured	Structured / unstructured
Data storage	Tables	document, key-value, wide-column, and graph
Integrity / reliability	ACID: <ul style="list-style-type: none"><li>• Atomicity</li><li>• Consistency</li><li>• Isolation</li><li>• Durability</li></ul>	Some do, others don't.  There are NoSQL DBs, which sacrifice ACID for speed (wordplay ☺)
Database language	SQL: <ul style="list-style-type: none"><li>• Data Definition Language (DDL)</li><li>• Data Manipulation Language (DML)</li><li>• Data Control Language (DCL)</li><li>• Data Query Language (DQL)</li></ul>	There is no standard similar to SQL for No-SQL databases. Each Database Management System has its own language, may they be similar to each other in cases.

# DATABASES

Feature	Excel	SQL Database
Structure	Rows, which contain attributes in different columns	Records, which contain different attributes. More restrictive with the data stored in each column. Also, there are primary keys.
Data types	Very many different data types are supported, including formulae	Many different data types are supported, but also enforced for the attributes in a record
Relationships	Manually constructed between tables for user formulae	Standard part of SQL. Build very strong and enforced rules.
Queries	(Built-in) formulae	Powerful language, which selects or changes data in tables.

# DATABASES

Suppliers		
Id	Integer	PK
CompanyName	String	
ContactName	String	
City	String	
Country	String	
Phone	String	
Fax	String	

Products		
Id	Integer	PK
ProductName	String	
SupplierId	Integer	FK
UnitPrice	Double	
Package	String	
IsDiscontinued	Bit	

OrderItem		
Id	Integer	PK
OrderId	Integer	FK
ProductId	Integer	FK
UnitPrice	Double	
Quantity	Integer	

Customers		
Id	Integer	PK
FirstName	String	
LastName	String	
City	String	
Country	String	
Phone	String	

Orders		
Id	Integer	PK
OrderDate	Date	
CustomerId	Integer	FK
TotalAmount	Double	

# DATABASES

Suppliers		
Id	Integer	PK
CompanyName	String	
ContactName	String	
City	String	
Country	String	
Phone	String	
Fax	String	

Products		
Id	Integer	PK
ProductName	String	
SupplierId	Integer	FK
UnitPrice	Double	
Package	String	
IsDiscontinued	Bit	

Orders		
Id	Integer	PK
OrderDate	Date	
CustomerId	Integer	FK
OrderNumber	String	
TotalAmount	Double	

Customers		
Id	Integer	PK
FirstName	String	
LastName	String	
City	String	
Country	String	
Phone	String	

OrderItem		
Id	Integer	PK
OrderId	Integer	FK
ProductId	Integer	FK
UnitPrice	Double	
Quantity	Integer	

# EXERCISE: C010 - DATABASES

- Run the code in `C010_CreateAndFillDB`
- When it runs without error change the `DB_URL` to

```
static final String DB_URL = "jdbc:h2:./testdb/testdb";
```

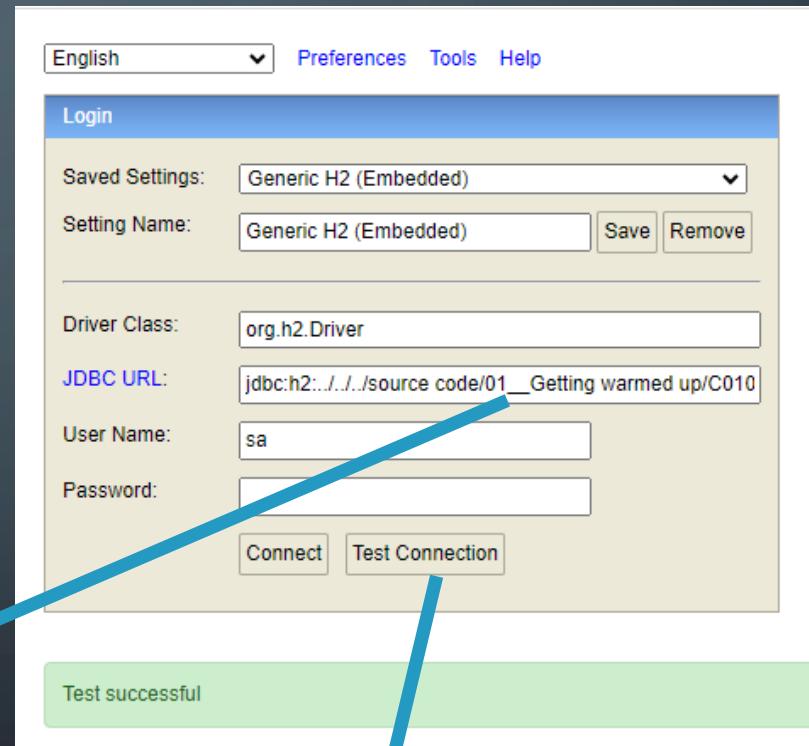
- Verify that in the project directory now there is a subdirectory called `testdb` and that in that subdirectory is a file called `testdb(.mv)`. This file contains the h2 database.

# DATABASES

- Goto to the libs/h2/bin directory
- Run `java -jar h2-2.2.224.jar`
- A new browser window opens

Enter path to DB (here relative from h2/bin directory)

Verify the correctness of data with "Test Connection"

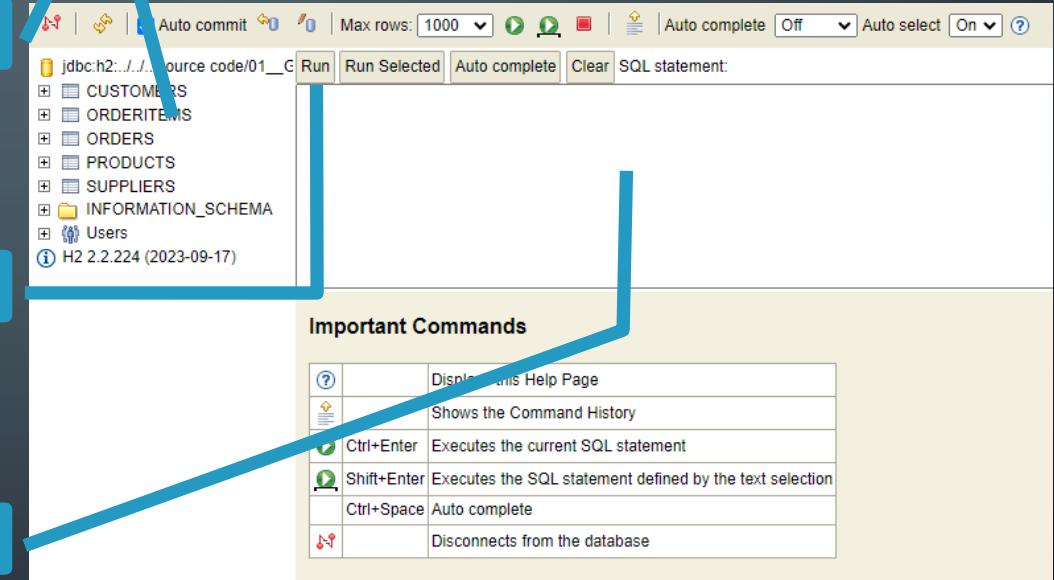


# DATABASES

Here are the tables of the DB

Click "Run" button to execute a SQL statement

Here you enter the SQL statements



# DATABASES

Details on the attributes of a table

Result of a run query

The screenshot shows a database interface with a sidebar containing a tree view of tables and schemas. The 'CUSTOMERS' table is selected, showing its attributes: ID (INTEGER), FIRSTNAME (CHARACTER VARYING(45)), LASTNAME, CITY, COUNTRY, and PHONE. Below the table structure, there is a list of indexes and foreign key relationships to ORDERITEMS, ORDERS, PRODUCTS, and SUPPLIERS. The 'INFORMATION\_SCHEMA' and 'Users' sections are also visible, along with a note about the version (H2 2.2.224 (2023-09-17)).

At the top of the main window, there are several buttons: Auto commit, Max rows: 1000, Run, Run Selected, Auto complete, Clear, and SQL statement: 'Select \* from customers;'. Below this, a results grid displays the data for the CUSTOMERS table:

ID	FIRSTNAME	LASTNAME	CITY	COUNTRY	PHONE
1	Maria	Anders	Berlin	Germany	030-0074321
2	Ana	Trujillo	México D.F.	Mexico	(5) 555-4729
3	Antonio	Moreno	México D.F.	Mexico	(5) 555-3932
4	Thomas	Hardy	London	UK	(171) 555-7788
5	Christina	Berglund	Luleå	Sweden	0921-12 34 65
6	Hanna	Moos	Mannheim	Germany	0621-08460
7	Frédérique	Citeaux	Strasbourg	France	88.60.15.31
8	Martín	Sommer	Madrid	Spain	(91) 555 22 82

# DATABASES – SELECT STATEMENT

- Select retrieves data from the database

Select statement part	Explanation
Select	Specifies the columns from the resultset to be returned
From	Specifies the tables to be used in the query, often also how they are joined
Where	Specifies the filter conditions
Group by	Specifies how to group rows in the resultset for running aggregate functions on them
Having	Specifies the filter criteria for the "Group by" groups in the resultset
Order by	Specifies how to sort the rows in the resultset
Limit	Limits the number of rows to be returned from the resultset

# DATABASES – SELECT STATEMENT

- String comparison in conditions:
  - **SQL wildcards:** Wildcards are special characters that represent one or more characters

Wildcard character	What it stands for
_	Underscore: stands for one single character
%	Percent sign: stands for zero or more characters

- **Escaping:** Wildcards are escaped by use of a backslash (e.g: '\\_ ' and '\%')
- **Example:**    `select * from Customers c where c.firstname like '%a%a'`

# DATABASES – SELECT STATEMENT

- select \* from Customers c where c.firstname <<comparison>> :

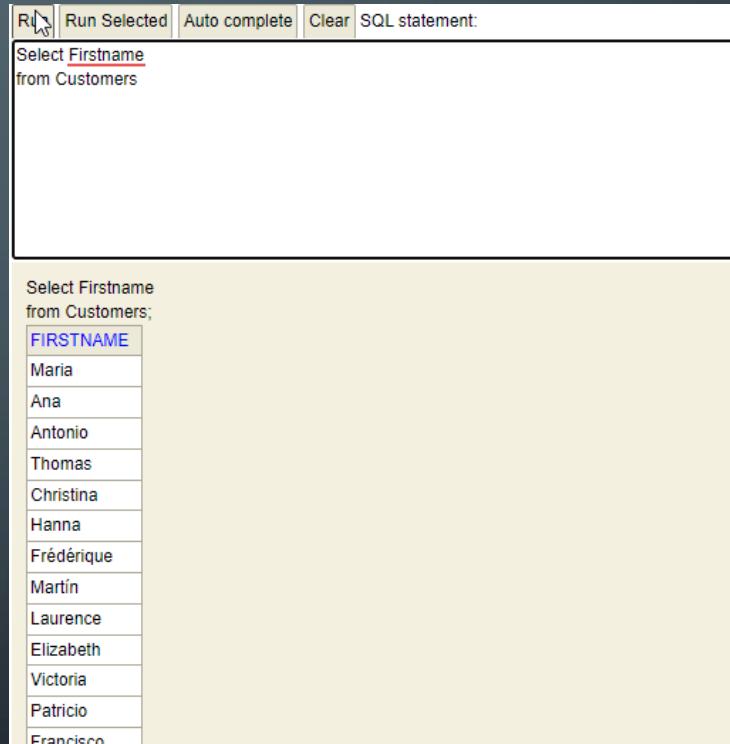
<<comparison>>	Result						
= 'Hanna'	<table border="1"><tr><td>6</td><td>Hanna</td><td>Moos</td><td>Mannheim</td><td>Germany</td><td>0621-08460</td></tr></table>	6	Hanna	Moos	Mannheim	Germany	0621-08460
6	Hanna	Moos	Mannheim	Germany	0621-08460		
Like 'Hanna'	<table border="1"><tr><td>6</td><td>Hanna</td><td>Moos</td><td>Mannheim</td><td>Germany</td><td>0621-08460</td></tr></table>	6	Hanna	Moos	Mannheim	Germany	0621-08460
6	Hanna	Moos	Mannheim	Germany	0621-08460		
Like 'Ha_na'	<table border="1"><tr><td>6</td><td>Hanna</td><td>Moos</td><td>Mannheim</td><td>Germany</td><td>0621-08460</td></tr></table>	6	Hanna	Moos	Mannheim	Germany	0621-08460
6	Hanna	Moos	Mannheim	Germany	0621-08460		
= 'Ha_na'	(no rows, 1 ms)						

# DATABASES – SELECT STATEMENT

- String comparison with Regex:
  - Comparison of strings can be done with Regex as well
- Example:
  - `select * from Customers c where c.lastname regexp 'M.*[n]+a'`

# DATABASES – SELECT STATEMENT

- Lists all first names from the customers table



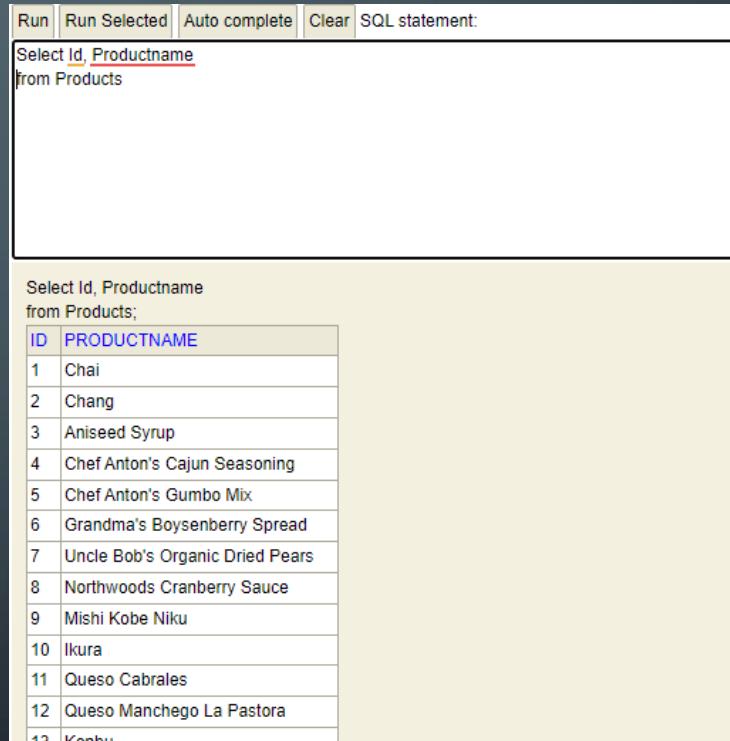
The screenshot shows a database query interface with the following details:

- Toolbar buttons: Run Selected, Auto complete, Clear, SQL statement.
- SQL Statement input field: "Select Firstname from Customers".
- Result table:

FIRSTNAME
Maria
Ana
Antonio
Thomas
Christina
Hanna
Frédérique
Martin
Laurence
Elizabeth
Victoria
Patricio
Francisco

# DATABASES – SELECT STATEMENT

- Lists all products with the internal id from the Products table



The screenshot shows a database interface with the following components:

- Toolbar:** Run, Run Selected, Auto complete, Clear, SQL statement.
- SQL Editor:** Select Id, Productname from Products;
- Result Grid:** A table showing product data with columns ID and PRODUCTNAME, containing 13 rows of data.

ID	PRODUCTNAME
1	Chai
2	Chang
3	Aniseed Syrup
4	Chef Anton's Cajun Seasoning
5	Chef Anton's Gumbo Mix
6	Grandma's Boysenberry Spread
7	Uncle Bob's Organic Dried Pears
8	Northwoods Cranberry Sauce
9	Mishi Kobe Niku
10	Ikura
11	Queso Cabrales
12	Queso Manchego La Pastora
13	Konbu

# DATABASES – SELECT STATEMENT

- List the first 10 rows of the supplier table sorted by the supplier name in ascending order

Run Run Selected Auto complete Clear SQL statement:

```
Select *
from Suppliers
order by Companyname ASC
limit 10;
```

Select \*
from Suppliers
order by Companyname ASC
limit 10;

ID	COMPANYNAME	CONTACTNAME	CONTACTTITLE	CITY	COUNTRY	PHONE	FAX
18	Aux joyeux: ecclésiastiques	Guylène Nodier	null	Paris	France	(1) 03.83.00.68	(1) 03.83.00.62
16	Bigfoot Breweries	Cheryl Saylor	null	Bend	USA	(503) 555-9931	null
5	Cooperativa de Quesos 'Las Cabras'	Antonio del Valle Saavedra	null	Oviedo	Spain	(98) 598 76 54	null
27	Escargots Nouveaux	Marie Delamare	null	Montceau	France	85.57.00.07	null
1	Exotic Liquids	Charlotte Cooper	null	London	UK	(171) 555-2222	null
14	Formaggi Fortini s.r.l.	Elio Rossi	null	Ravenna	Italy	(0544) 60323	(0544) 60603
29	Forêts d'éables	Chantal Goulet	null	Ste-Hyacinthe	Canada	(514) 555-2955	(514) 555-2921
24	G'day, Mate	Wendy Mackenzie	null	Sydney	Australia	(02) 555-5914	(02) 555-4873
28	Gai pâturage	Eliane Noz	null	Annecy	France	38.76.98.06	38.76.98.58
3	Grandma Kelly's Homestead	Regina Murphy	null	Ann Arbor	USA	(313) 555-5735	(313) 555-3349

(10 rows, 3 ms)

# DATABASES – SELECT STATEMENT

- List all orders where the totalamount is greater than 10000

Run Run Selected Auto complete Clear SQL statement:

```
Select *
from orders o
where o.TOTALAMOUNT > 10000
```

Select \*
from orders o
where TOTALAMOUNT > 10000;

ID	ORDERDATE	ORDERNUMBER	CUSTOMERID	TOTALAMOUNT
106	2012-11-13 00:00:00	542483	59	10741.60
125	2012-12-04 00:00:00	542502	62	12281.20
170	2013-01-16 00:00:00	542547	73	11283.20
177	2013-01-23 00:00:00	542554	51	11493.20
232	2013-03-19 00:00:00	542609	65	10495.60
268	2013-04-23 00:00:00	542645	63	10588.50
293	2013-05-19 00:00:00	542670	63	10191.70
444	2013-10-03 00:00:00	542821	63	10164.80
570	2014-01-06 00:00:00	542947	39	11490.70
618	2014-02-02 00:00:00	542995	63	17250.00
642	2014-02-16 00:00:00	543019	65	11380.00
650	2014-02-19 00:00:00	543027	37	10835.24
734	2014-03-27 00:00:00	543111	34	15810.00
783	2014-04-17 00:00:00	543160	71	16321.90

(14 rows, 2 ms)

# DATABASES – SELECT STATEMENT

- These are some aggregate functions:

Aggregate function	Explanation
count()	Number of rows
min()	Minimum of attribute column in resultset
max()	Maximum of attribute column in resultset
avg()	Average of attribute column in resultset
sum()	Sum of attribute column in resultset

# DATABASES – SELECT STATEMENT

- Display the minimum and maximum Totalamount in Orders
- The aggregated functions min and max are here used on all the records in the resultset as there is no group by clause.

Run Run Selected Auto complete Clear SQL statement:

```
Select min(totalamount) as Minimum, max(totalamount) as Maximum  
from orders o;
```

```
Select min(totalamount) as Minimum, max(totalamount) as Maximum  
from orders o;
```

MINIMUM	MAXIMUM
12.50	17250.00

(1 row, 18 ms)

# DATABASES – SELECT STATEMENT

- Display the minimum and maximum totalamount for each customer's orders

Run Run Selected Auto complete Clear SQL statement:

```
Select o.customerid, min(o.totalamount) as Minimum, max(o.totalamount) as Maximum  
from orders o  
group by o.customerid;
```

Select o.customerid, min(o.totalamount) as Minimum, max(o.totalamount) as Maximum  
from orders o  
group by o.customerid;

CUSTOMERID	MINIMUM	MAXIMUM	COUNT(*)
1	330.00	1086.00	6
2	88.80	514.40	4

# DATABASES – SELECT STATEMENT

- Display the minimum and maximum totalamount for each customer's orders for all customers with more than 22 orders

```
Run Run Selected Auto complete Clear SQL statement:  
Select o.customerid, min(o.totalamount) as Minimum, max(o.totalamount) as Maximum, count(*)  
from orders o  
group by o.customerid  
having count(*) >= 22;
```

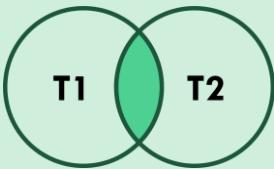
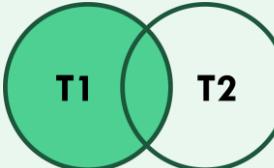
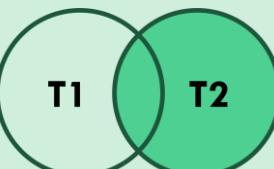
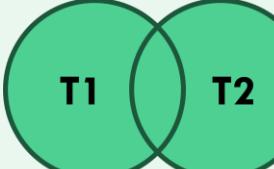
  

```
Select o.customerid, min(o.totalamount) as Minimum, max(o.totalamount) as Maximum, count(*)  
from orders o  
group by o.customerid  
having count(*) >= 22;
```

CUSTOMERID	MINIMUM	MAXIMUM	COUNT(*)
20	344.00	8623.45	30
63	82.40	17250.00	28
71	40.00	16321.90	31

(3 rows, 3 ms)

# DATABASES – SELECT STATEMENT

Type of Join	Explanation	
(inner) join	Returns records which have matching values in both tables	
left (outer) join	All records from the left table and the matched records from the right table	
right (outer) join	All records from the right table and the matched records from the left table	
full (outer) join	Returns all the records when there is a match in either table	

# DATABASES – SELECT STATEMENT

- Only orders with a matching customer for it

(Should be all of them)

Run	Run Selected	Auto complete	Clear	SQL statement:
SELECT * FROM customers cu INNER JOIN orders ON cu.id = orders.customerid;  select count(*) from orders				

# DATABASES – SELECT STATEMENT

- All customers, even those without orders

(Check out how the order data is represented when there is no order for a customer)

```
Run Run Selected Auto complete Clear SQL sta
SELECT *
FROM customers cu
LEFT JOIN orders ON cu.id = orders.customerid;
select count(*) from orders;
select count(*) from customers;
```

# DATABASES – SELECT STATEMENT

- All the orders, even those without a customer

(There better is none without a customer!)

```
Run Run Selected Auto complete Clear SQL statement:  
SELECT *  
FROM customers cu  
RIGHT JOIN orders ON cu.id = orders.customerid;  
  
select count(*) from orders;  
  
select count(*) from customers;
```

# DATABASES – SELECT STATEMENT

- Funny enough, the H2 SQL dialect does not support FULL JOINS

You will have to switch to a different dialect.

The screenshot shows an SQL editor interface with the following content:

```
Run Run Selected Auto complete Clear SQL statement:  
SELECT *  
FROM customers cu  
FULL JOIN orders ON cu.id = orders.customerid;  
  
select count(*) from orders;  
  
select count(*) from customers;
```

Below this, another section shows the same code with a red error message:

```
SELECT *  
FROM customers cu  
FULL JOIN orders ON cu.id = orders.customerid;
```

Syntax Fehler in SQL Befehl "SELECT \*\000d\000aFROM customers cu\000d\000aFU  
Syntax error in SQL statement "SELECT \*\000d\000aFROM customers cu\000d\000aF  
SELECT \*  
FROM customers cu  
FULL JOIN orders ON cu.id = orders.customerid [42000-224] 42000/42000 (Help)

# DATABASES – SELECT STATEMENT

- List all customers, which have placed orders.

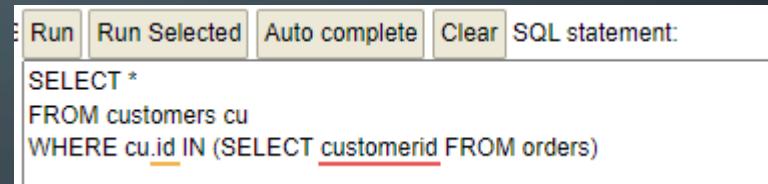
Since we use the \*, not only the customer's attributes, but also the order's attributes are shown

```
Run Run Selected Auto complete Clear SQL statement:  
SELECT *  
FROM customers cu  
INNER JOIN orders ON cu.id = orders.customerid;
```

# DATABASES – SELECT STATEMENT

- List all customers, which have placed orders.

This is another way, completely without joining tables. This solution uses a sub-select.



A screenshot of a SQL editor interface. At the top, there are buttons for 'Run', 'Run Selected', 'Auto complete', 'Clear', and 'SQL statement:'. Below the buttons is a text input field containing the following SQL code:

```
SELECT *  
FROM customers cu  
WHERE cu.id IN (SELECT customerid FROM orders)
```

The word 'cu' is underlined in orange, and 'customerid' is underlined in red, indicating they are identifiers in the query.

# DATABASES – SELECT STATEMENT

- Hangon .....

The "Join" version provides 830 result rows, while the "sub-select" version provides 89.

WHY is that?

# DATABASES – SELECT STATEMENT

- Modify the "Join" version slightly:

Distinct eliminates all duplicate rows  
in the final resultset

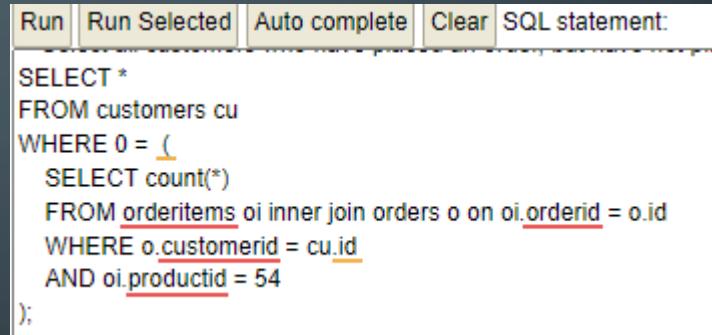
```
SELECT distinct cu.id, cu.firstname, cu.lastname  
FROM customers cu  
inner join orders on cu.id = orders.customerid
```

# DATABASES – SELECT STATEMENT

- Just for completeness:

Display all customers who have placed an order, but not for the product with the ID 54

The solution is based on a correlated sub-select



A screenshot of a SQL editor interface. At the top, there are several buttons: Run, Run Selected, Auto complete, Clear, and SQL statement. Below the buttons is a text input field containing a SQL query. The query is as follows:

```
SELECT *
FROM customers cu
WHERE 0 = (
    SELECT count(*)
    FROM orderitems oi inner join orders o on oi.orderid = o.id
    WHERE o.customerid = cu.id
    AND oi.productid = 54
);
```

# EXERCISES: DATABASES – SELECT STATEMENT

1. Show all products, that are with 100 or more items in stock
2. Show all orderitems for order #830
3. Show the first 3 orders with the most orderitems
4. Show all orderIDs, where the totalamount is different from the sum of the ordervalue of the orderitems
5. Show all customers without orders

# DATABASES – NULL VALUE

- NULL is the absence of data
- Any attribute can be null, if not declared "not null" in the create table statement
- NULL is different from an empty string or zero
- NULL cannot be compared
- To check for NULL use "is [not] null"
- To prevent NULL results use `coalesce(<attrib>, <value>)`
- Also, the `ifnull(<attrib>, <value>)` function can be used

# DATABASES – NULL VALUE

- Run "select \* from suppliers". Notice, that there are records with a NULL value for a fax number.
- How could that be done better?
- Now run "select \* from suppliers where fax = null"

Explain the resultset.

# DATABASES – TRANSACTIONS

- Transactions allow to treat multiple SQL statements as a single unit of work.
- Either all statements within a transaction succeed or none
- Databases are often set to auto-commit, which then commits every single statement
- If not, transaction start implicitly until they are explicitly finished.
- Transactions result in locks in the DB and therefore impact parallel access

# DATABASES – TRANSACTIONS

- Even though transactions start implicitly there are "Begin Transaction" or "Start Transaction" commands
- "Commit" finishes a transaction successfully and makes the changes permanent
- "Rollback" puts the DB in a state as if the transaction was never run
- "Savepoints" are steps in a transaction to which one can "Rollback" to, as long as the transaction has not been finished.

# DATABASES – SELECT STATEMENT

- Here is the order in which a DBMS processes a select statement:

Select statement part	Addtn'l info
From clause	Tables and Join instructions
Where clause	Filters the rows
Group by	Groups the rows of the resultset
Having	Filters the groups according the criteria
Select	Constructs the attributes shown
Order by	Orders the resultset
Limit	Restricts the number of rows in the resultset

# DATABASES – QUERY IN JAVA

- Database URL:
  - Jdbc      -> Protocol
  - h2          -> Driver to use
  - Rest        -> Path to DB (URL)
- User name ( sa -> System Admin)
- Password (1234 -> Standard (h2))

```
public class Main {  
    static final String DB_URL = "jdbc:h2:./testdb/testdb";  
    // static final String DB_URL = "jdbc:h2:mem:test";  
    // "jdbc:h2:mem:mymemdb:"  
    // "jdbc:h2:./testdb"  
    // "jdbc:h2:///c/temp/testdb"  
    static final String USER = "sa";  
    static final String PASS = "1234";
```

# DATABASES – QUERY IN JAVA

- The enclosed `getConnection` ensures that the DB is closed automatically
- `conn` is used for working with the DB

```
public static void main(String[] args) {  
  
    try (Connection conn = DriverManager.getConnection(DB_URL, USER, PASS)) {  
  
        <<SQL Statements to process>>  
  
        System.out.println("Successfully run the SQL statements.");  
  
    } catch (Exception ex) {  
        System.err.println("ERROR: " + ex.getMessage());  
    }  
}
```

# DATABASES – QUERY IN JAVA

- This is a simple select statement
- The results are in rs
- The while loop runs through the records
- Attribute values can be extracted if the data type is known

```
final String QUERY = "SELECT * FROM Products where ProductName like 'C%';";
try(Statement stmt = conn.createStatement()) {
    ResultSet rs = stmt.executeQuery(QUERY);
    while( rs.next() ) {
        System.out.println("Name: " + rs.getString("productname"));
        System.out.println("Price: " + rs.getDouble("unitprice"));
    }
}
```

# DATABASES – QUERY IN JAVA

- If the SQL statement stays the same, but the criteria values change between runs one uses:

## PreparedStatement

- The set method replace the ?s with the real value

```
final String QUERY = "SELECT * FROM Products"
    + " where ProductName like ? and unitprice > ?;";
try(PreparedStatement ps = conn.prepareStatement(QUERY)) {
    ps.setString(1, "C" + "%");
    ps.setDouble(2, 40.0);
    ResultSet rs = ps.executeQuery();
    while( rs.next()) {
        System.out.println("Name: " + rs.getString("productname"));
        System.out.println("Price: " + rs.getDouble("unitprice"));
    }
}
```

# EXERCISE: DATABASES – QUERY IN JAVA

- Write a Java program, which takes one or two parameters from the command line (Productname and Unitprice).
- Use the parameters to run the Select statement from the slides before
- Print the resulting lines to the console

(No solution will be provided)

# DATABASES – ORM

- Object-Relational Mapping (ORM) is an abstraction layer on top of the database layer
- It supports mapping of Classes (Objects) to Tables and vice versa
- Many ORM support lazy loading for improving performance
- Also, caching is supported by many ORM
- ORMs support transactions

# DATABASES – ORM EXAMPLE

- This code retrieves all customers
- The resultset is mapped into a Java List containing instances of the Customer class.

```
// Create a new ORM object  
EntityManager em = EntityManagerFactory.createEntityManager();  
  
// Create a query to retrieve all customers  
Query query = em.createQuery("SELECT c FROM Customer c");  
  
// Execute the query and get the results  
List<Customer> customers = query.getResultList();  
  
// Close the ORM object  
em.close();
```

# DATABASES – ORM EXAMPLE

- This code adds a new customer to the database.
- First one is setting the values in an instance of the customer class
- Then a call to persist will store the new customer into the Customer table in the DB

```
// Create a new ORM object  
EntityManager em = EntityManagerFactory.createEntityManager();  
  
// Create a new Java object  
Customer customer = new Customer();  
customer.setName("John Doe");  
customer.setEmail("john.doe@example.com");  
  
// Save the Java object to the database  
em.persist(customer);  
  
// Close the ORM object  
em.close();
```

# DATABASES – ORM EXAMPLE

- Pseudo code on the right
- q → Query  
cb → CriteriaBuilder
- This way the sql statement can be build step by step.

```
Query q = cb.createQuery( Student.class );
q.where( cb.greaterThan "studentID", 423 );
List<Student> lSt = q.execute();
```

# DATABASES – ORM ADVANTAGES

- Productivity gain: Writing DB code is much easier
- Improved Maintainability
- Increased portability

# DATABASES – KV-DB

- In a Key-Value DB each record is stored under a key
- There is (almost) no assumption on what the data is
- Keys are Strings
- Depending on the data a record might be a JSON String
- KV-DBs are often run in memory and therefore very fast

# DATABASES – KV-DB

- In our examples we make use of a KV-DB by the name of MemKV
- MemKV holds its data in memory
- It can be persistet to and loaded from disk
- Anything stored as data needs to be serializable
- Primitive datatypes are converted to their correlating class
- Data retrieved is always some object, if no specialized getter is used

# DATABASES – KV-DB

MemKV method	Explanation
public MemKV()	Constructor
int size()	Number of different keys in the DB
boolean isEmpty()	Does DB contain any keys
void clear()	Empty DB
void delete / void remove (<<key>>)	Remove record identified by <<key>>
boolean containsKey(<<key>>)	Does DB contain <<key>>
MemKV getCopyOfDB()	Copy Db into a new one
List<String> getListOfKeys()	Get a list of all keys in DB
void addOtherDB(MemKV memKV)	Add records (K and v) from another Db. Same key result in overwritten value
void addOtherDB(MemKV memKV, boolean onlyNewKeys)	Only take the records of a another DB for which the key does not exist in the actual DB
boolean containsValue( Object value )	Check if DB contains a specific value

# DATABASES – KV-DB

MemKV storage / retrieval method	Explanation
void put( <<key>>, Object / primitive datatype )	Data is stored under <<key>>, if data is primitive datatype it is converted in the corresponding Class Instance.
MemKV getMemKV( <<key>> )	Get MemKV data stored under key
Date getDate( <<key>> )	Get Date data stored under key
String getString( <<key>> )	Get String data stored under key
Double getDouble( <<key>> )	Get Double data stored under key
Float getFloat( <<key>> )	Get Float data stored under key
Boolean getBoolean( <<key>> )	Get Boolean data stored under key
Integer getInteger( <<key>> )	Get Integer data stored under key
BigInteger getBigInteger( <<key>> )	Get BigInteger data stored under key
Object get( <<key>> )	Get Object data stored under key. Generic get. Conversion by you.

# DATABASES – KV-DB

MemKV additional method	Explanation							
<code>void persist( &lt;&lt;filename&gt;&gt; )</code>	Save DB under filename to disk							
<code>void load( &lt;&lt;filename&gt;&gt; )</code>	Load Db from filename on disk							
<code>MemKV findEntriesByKey( searchPatter )</code>	<p>Get a new MemKV with only the records, which keys match the searchpattern.</p> <p>Searchpatter-Wildcards:</p> <table><tr><td>*</td><td>Zero or more characters</td></tr><tr><td>?</td><td>Single character</td></tr><tr><td>Any other char</td><td>Any character</td></tr></table>		*	Zero or more characters	?	Single character	Any other char	Any character
*	Zero or more characters							
?	Single character							
Any other char	Any character							

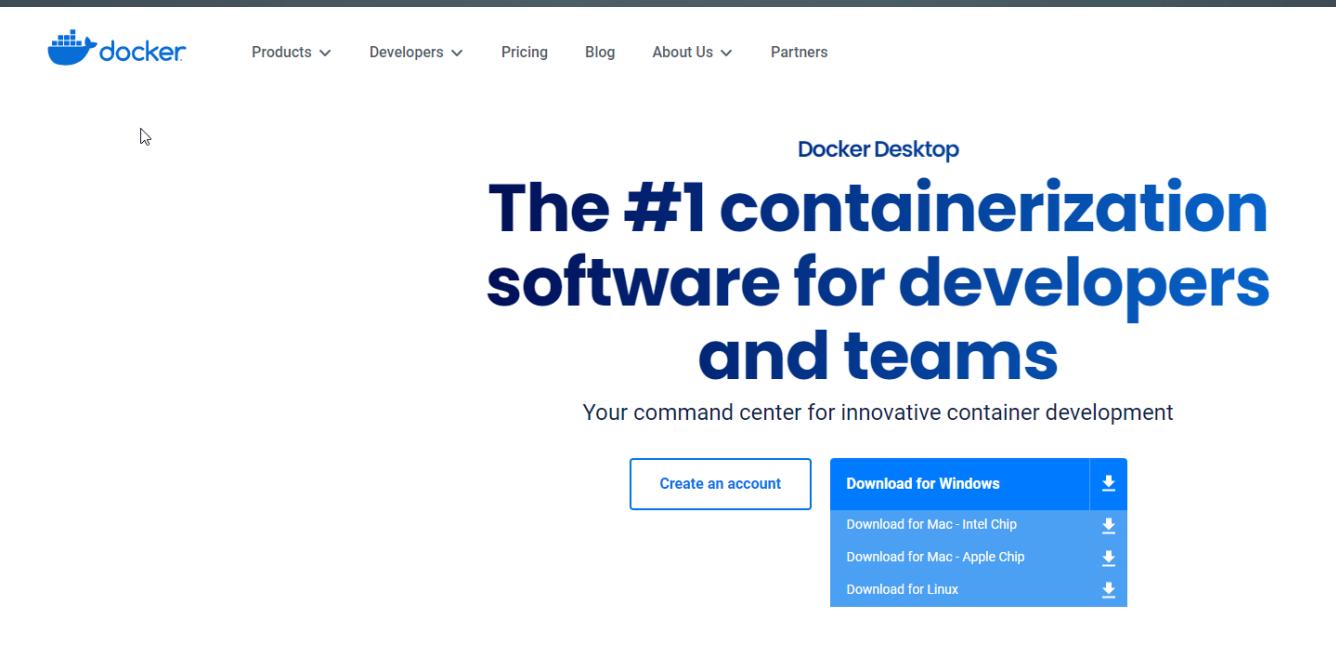
# DISCLAIMER

- The presentation was created on a MS-Windows-based Computer
- Screen-shots and explanations are based on the MS-Windows environment
- Wherever the author is aware of a significant difference between MS-Windows and the Macintosh operating environments the slides will point out these differences.
- The author uses [Github.com](https://github.com) for all GIT remote repository demonstration purposes

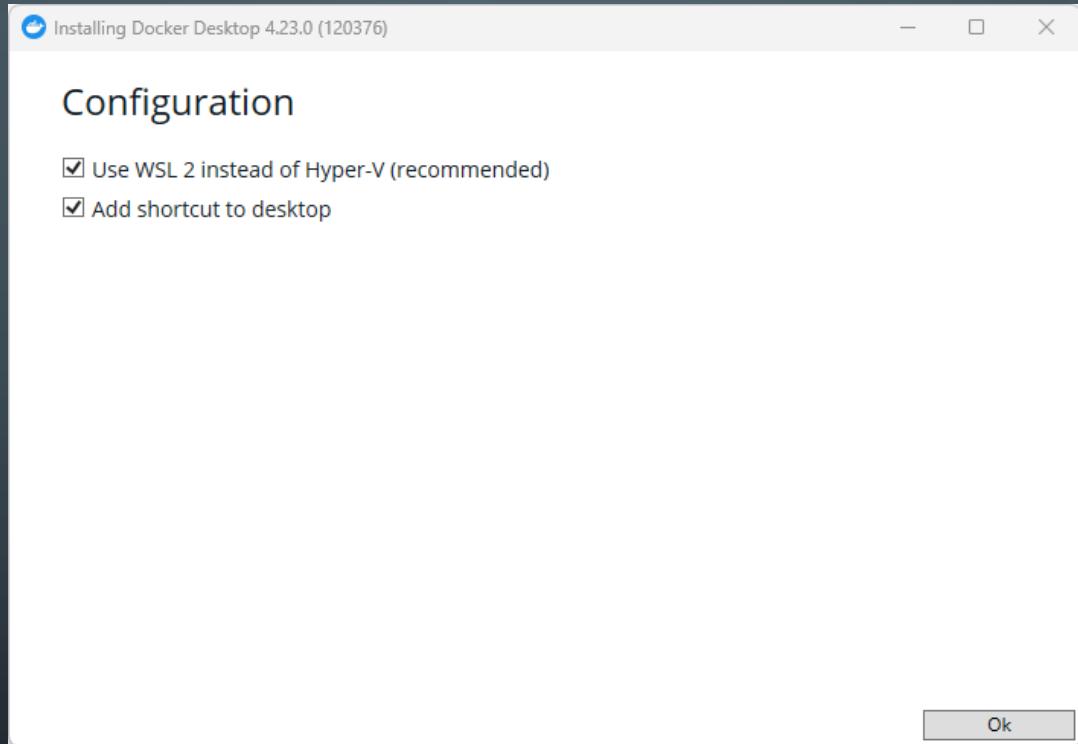
## APPENDIX A: DOCKER DESKTOP INSTALLATION

# DOWNLOAD DOCKER DESKTOP

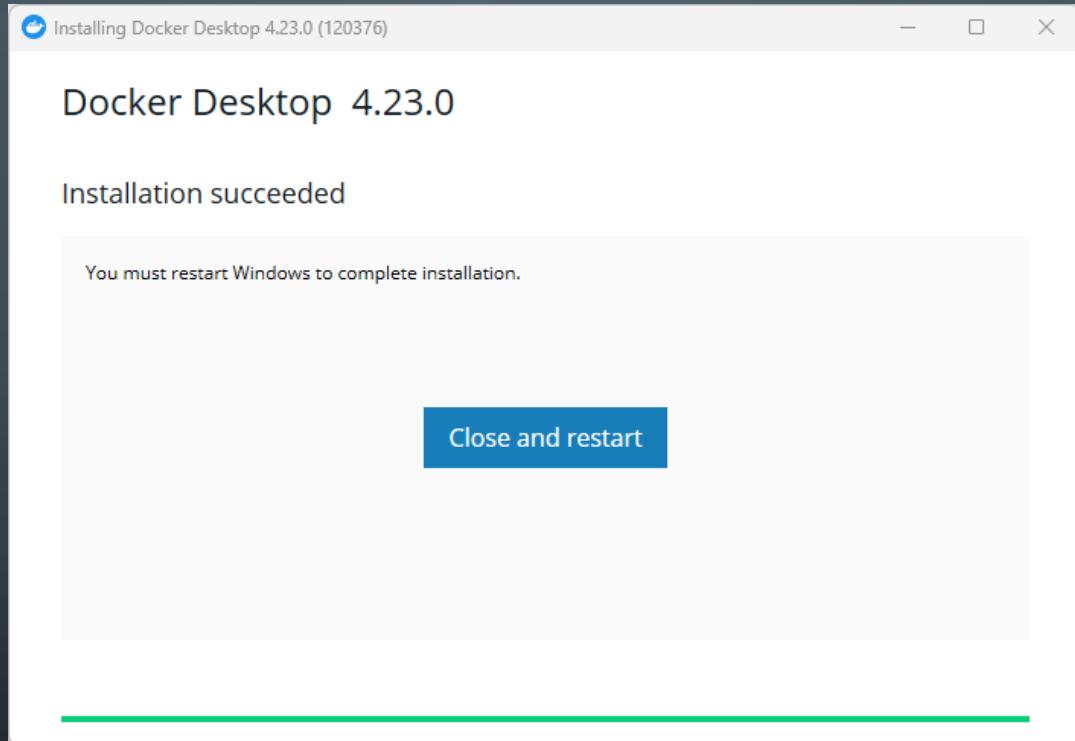
- Goto: <https://www.docker.com/products/docker-desktop/>



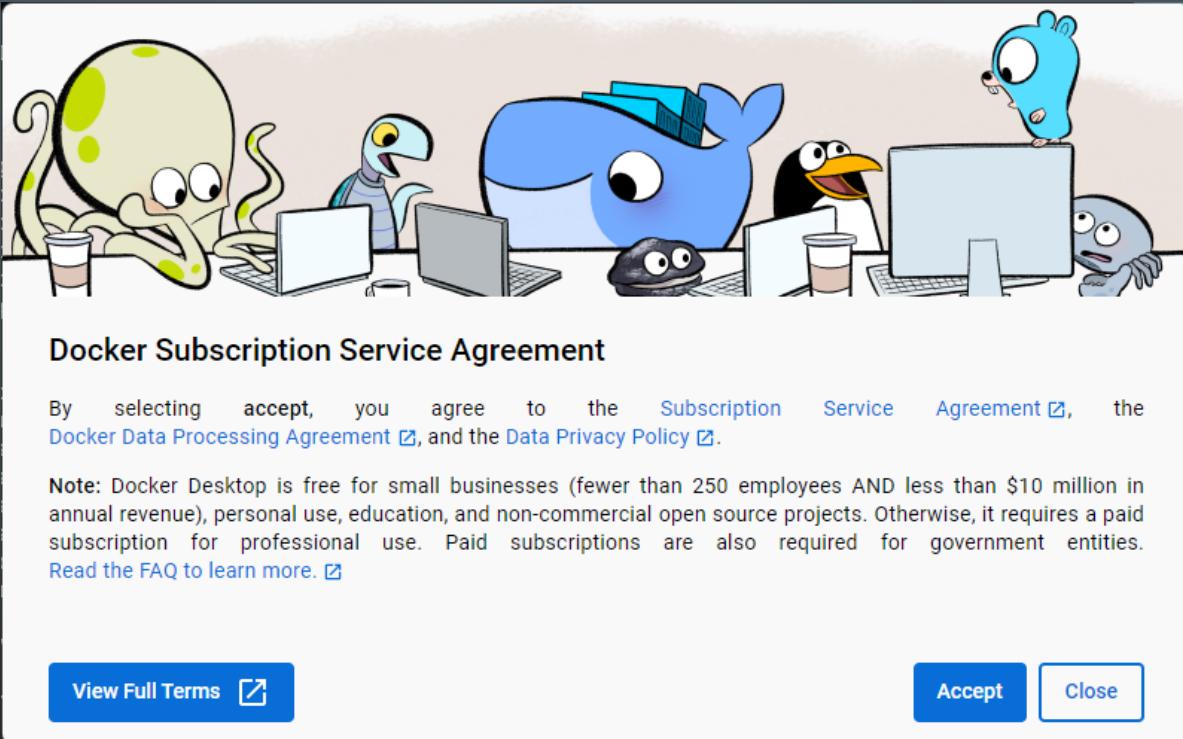
# START INSTALLATION



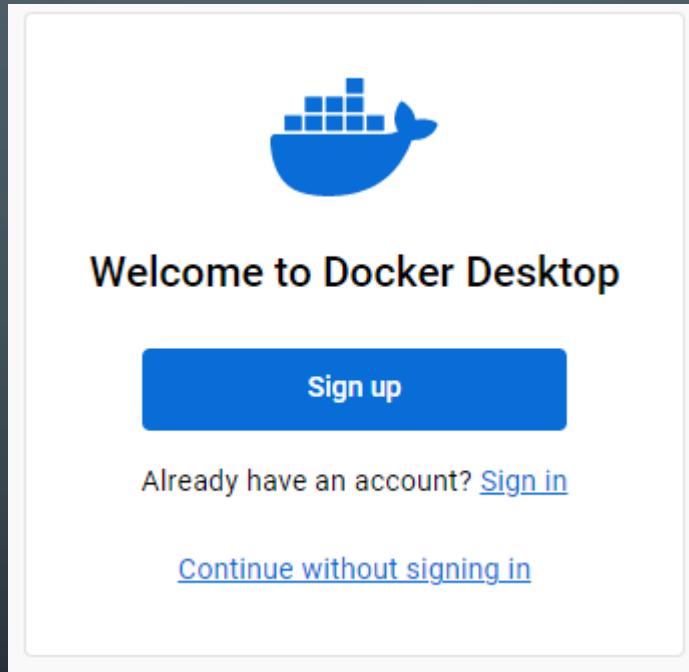
# SUCCESSFULLY FINISHED – REQUIRES RESTART



# AFTER RESTART



# SIGN IN / UP



# ON THE COMMAND LINE

- Docker Desktop might require a newer version of WSL:

Command line:    `wsl --update`

# FINALLY YOU SHOULD SEE THIS

