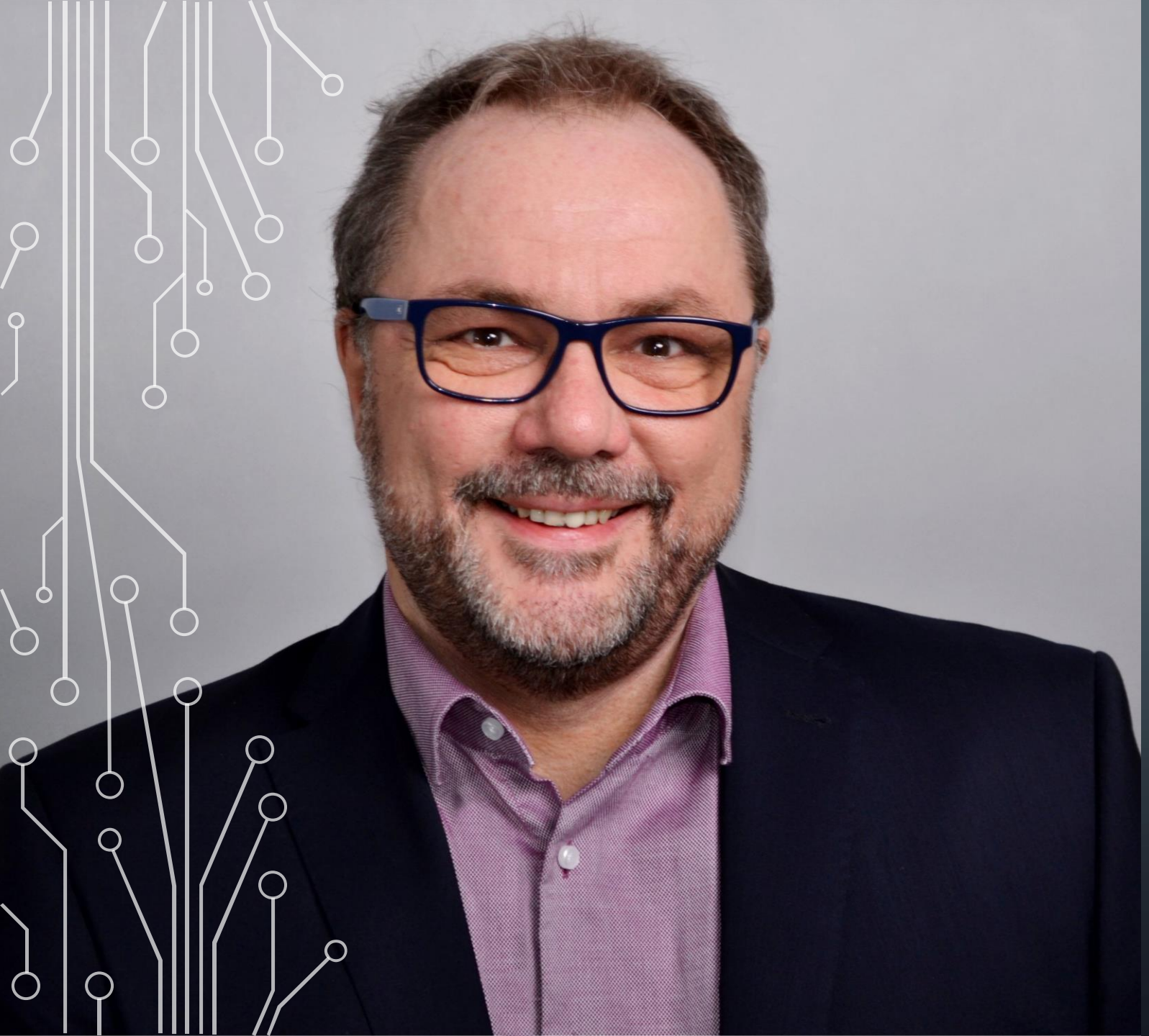
PRACTICAL JAVA FOR IT PROFESSIONALS

KRISTOF WERLING

KWERLINGIT GMBH

1 SEPT 2023



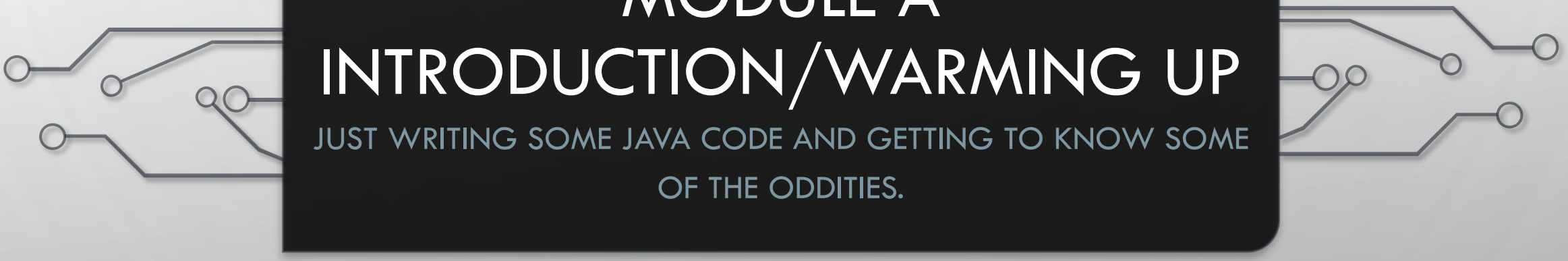
WHO AM I

- 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
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

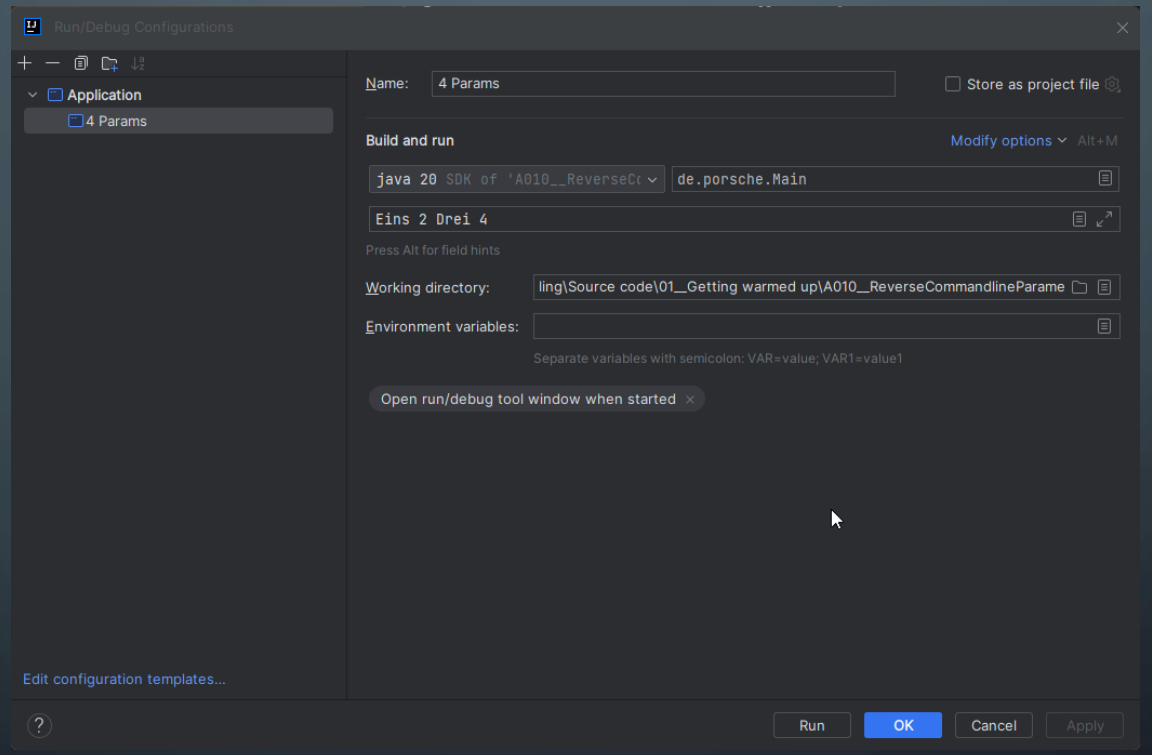
- IntelliJ provides one with a default Java program
- The "args" array contains the command line parameters

```
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



SOLUTION A010

```
package  de.porsche;

public class Main {
    public static void main(String[] args) {
        for(int i = args.length - 1;  i >= 0; i--) {
            System.out.println("Parameter #" + i + "\t" + args[i]);
        }
    }
}
```

WORKING WITH STRINGS

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

Part 3

```
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();
   }
```

GARBAGE COLLECTOR

Part 2

```
@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 explicitely -> 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

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

String paramter for Integer.parse	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

COMMAND LINE PARAMETERS 2

Part 1

```
package de.porsche;

public class Main {

   public static void main(String[] args) {
      boolean isInteger = false;
      int intVal = -1;
      for( int i = args.length; i>0;  i--) {
         try {
            intVal = Integer.parseInt( args[i - 1]) + 10;
            isInteger = true;
         } catch( Exception ex ) {
            isInteger = false;
         }
         if( isInteger ) {
            System.out.println("Param #" + i +":   " + intVal + "    INTEGER");
         } else {
            System.out.println("Param #" + i + ":   " + args[i - 1]);
         }
      }
   }
}
```

REVERSE POLISH NOTATION (UPN)

- Example : 3 + 4 * 5 = 23

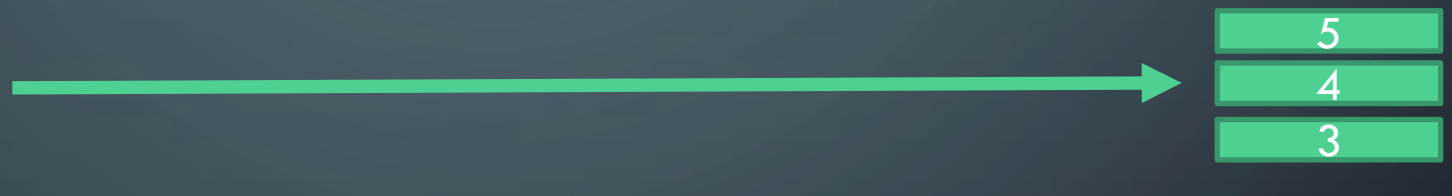
  In RPN it looks like this: 3 4 5 * +

  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 syntax is correct – no error checking is done
- The last step of the program is to print out the top of the stack (result)

REVERSE POLISH NOTATION (UPN)

Part 1

```
package de.porsche;

public class Main {
   public static void main(String[] args) throws Exception {
      String expression = "1 2 + 3 *";

      RpnCalculator calculator = new RpnCalculator();
      int result = calculator.evaluate(expression);

      System.out.println(result); //
   }
}
```

REVERSE POLISH NOTATION (UPN)

Part 2

```
package de.porsche;

import java.util.Stack;

interface Operator {

   int apply(int operand1, int operand2);
}

public class RpnCalculator {

   private final Stack<Integer> stack;

   public RpnCalculator() {
      stack = new Stack<>();
   }
```

REVERSE POLISH NOTATION (UPN)

Part 3

```
public int evaluate(String expression) throws Exception {
    String[] tokens = expression.split("\\s+");
    for (String token : tokens) {
        if (isOperator(token)) {
            Operator operator = getOperator(token);
            int operand2 = stack.pop();
            int operand1 = stack.pop();
            int result = operator.apply(operand1, operand2);
            stack.push(result);
        } else {
            int number = Integer.parseInt(token);
            stack.push(number);
        }
    }

    if (stack.size() != 1) {
        throw new Exception("Invalid RPN expression");
    }

    return stack.pop();
}
```

REVERSE POLISH NOTATION (UPN)

Part 4

```
private boolean isOperator(String token) {
   return token.equals("+") || token.equals("-") || token.equals("*") || token.equals("/");
}

private Operator getOperator(String token) throws Exception {
   return switch (token) {
      case "+" -> new AdditionOperator();
      case "-" -> new SubtractionOperator();
      case "*" -> new MultiplicationOperator();
      case "/" -> new DivisionOperator();
      default -> throw new Exception("Invalid operator: " + token);
   };

   /*  switch (token) {
         case "+":   return new AdditionOperator();
         case "-":   return new SubtractionOperator();
         case "*":   return new MultiplicationOperator();
         case "/":   return new DivisionOperator();
         default:    throw new Exception("Invalid operator: " + token);
      }
    */
}
```

REVERSE POLISH NOTATION (UPN)

Part 5

```
class AdditionOperator implements Operator {

   @Override
   public int apply(int operand1, int operand2) {
      return operand1 + operand2;
   }
}

class SubtractionOperator implements Operator {

   @Override
   public int apply(int operand1, int operand2) {
      return operand1 - operand2;
   }
}
```

REVERSE POLISH NOTATION (UPN)

Part 6

```
class MultiplicationOperator implements Operator {

   @Override
   public int apply(int operand1, int operand2) {
      return operand1 * operand2;
   }
}

class DivisionOperator implements Operator {

   @Override
   public int apply(int operand1, int operand2) {
      return operand1 / operand2;
   }
}
```

REVERSE POLISH NOTATION (UPN)

- Discuss the provided solution

REVERSE POLISH NOTATION (UPN)

Part 1

```
package de.porsche;

import java.util.Stack;

public class RPNCalculator {

    public static void main(String[] args) {
        // Create a stack to store the operands
        Stack<Integer> stack = new Stack<>();

        processCommandLineParams(args, stack);

        // The result of the calculation is now on the top of the stack
        int result = stack.pop();

        // Print the result to the console
        System.out.println("The result is: " + result);
    }
```

REVERSE POLISH NOTATION (UPN)

Part 2

```
private static void processCommandLineParams(String[] args, Stack<Integer> stack) {
    // Iterate over the command line parameters
    for (String arg : args) {
        // If the argument is an operand, push it onto the stack
        if (isOperand(arg)) {
            stack.push(Integer.parseInt(arg));
        } else {
            // If the argument is an operator, pop the top two elements of the stack, perform the operation, and push the result back onto the stack
            doCalculation(stack, arg);
        }
    }
}

private static void doCalculation(Stack<Integer> stack, String arg) {
    int operand2 = stack.pop();
    int operand1 = stack.pop();
    int result = performOperation(arg, operand1, operand2);
    stack.push(result);
}
```

REVERSE POLISH NOTATION (UPN)

Part 3

```
    private static boolean isOperand(String arg) {
        // An operand is a string that does not contain any of the following characters: +, -, *, /
        return !arg.contains("+") && !arg.contains("-") && !arg.contains("*") && !arg.contains("/");
    }

    private static int performOperation(String operator, int operand1, int operand2) {
        // Perform the specified operation on the two operands and return the result
        switch (operator) {
            case "+":
                return operand1 + operand2;
            case "-":
                return operand1 - operand2;
            case "*":
                return operand1 * operand2;
            case "/":
                return operand1 / operand2;
            default:
                throw new IllegalArgumentException("Unknown operator: " + operator);
        }
    }
}
```

REVERSE POLISH NOTATION (UPN)

Part 3

# of runs	Time procedural [sec]	Time OOP [sec]	P faster than OOP [%]
1.000.000	1.892	2.561	35
100.000.000	111	185	66

In addition to the runtime difference, there is a difference in size of the binaries:

The OOP binaries are almost twice the size of the procedural approach.

Exercise: Explain why that might be the case.

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 testting:

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.

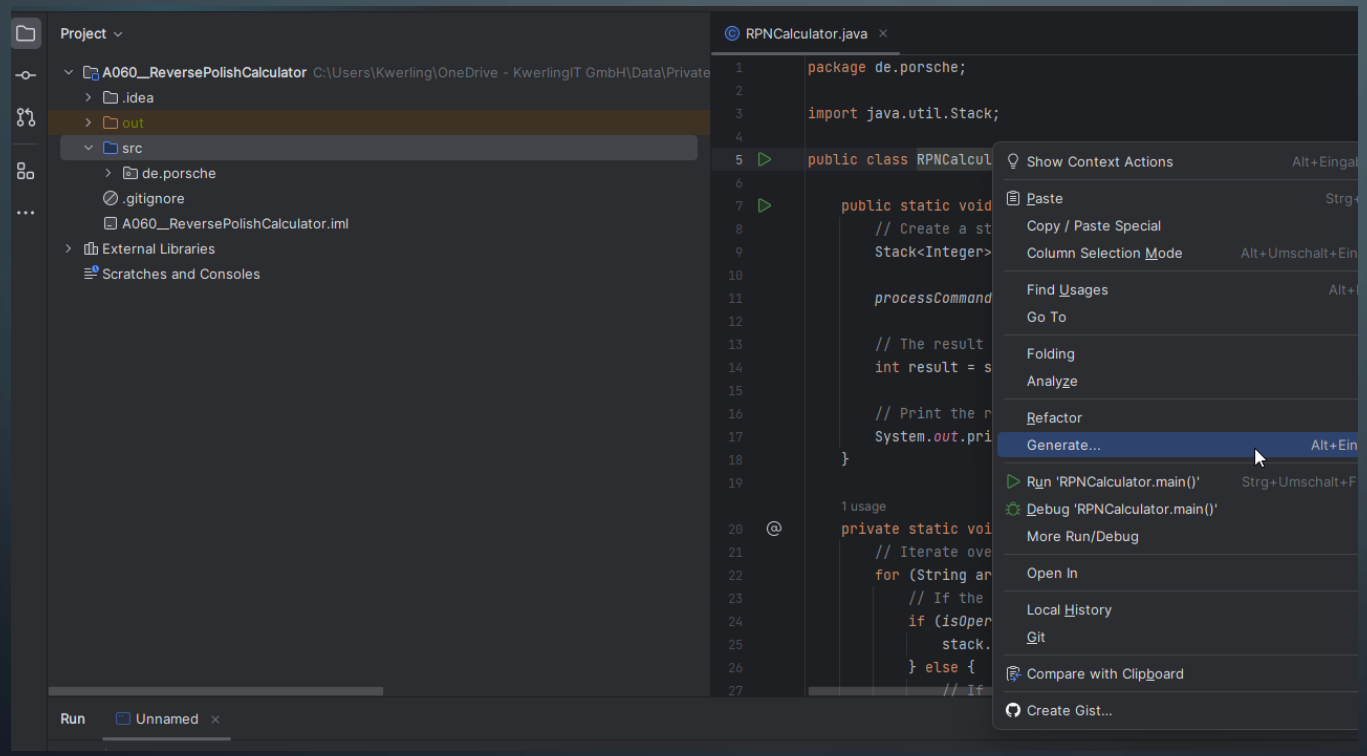
UNIT TESTING

- Advantages to gain through unit testing:

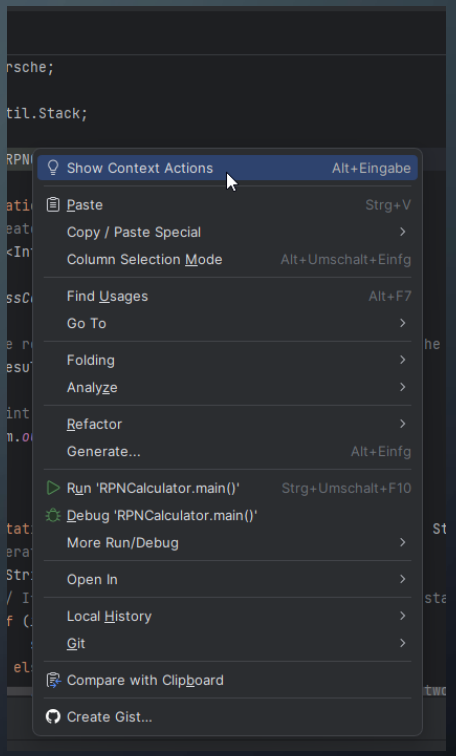
1. Improved code quality
2. Tests document the expected behavior of the code
3. Confidence in your code
4. Easier refactoring or enhancing of code: Assured that code still behaves as it did before the changes.

UNIT TESTING - INTELLIJ

Part 1

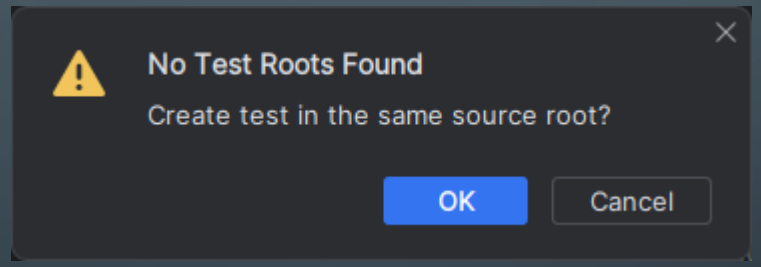


OR

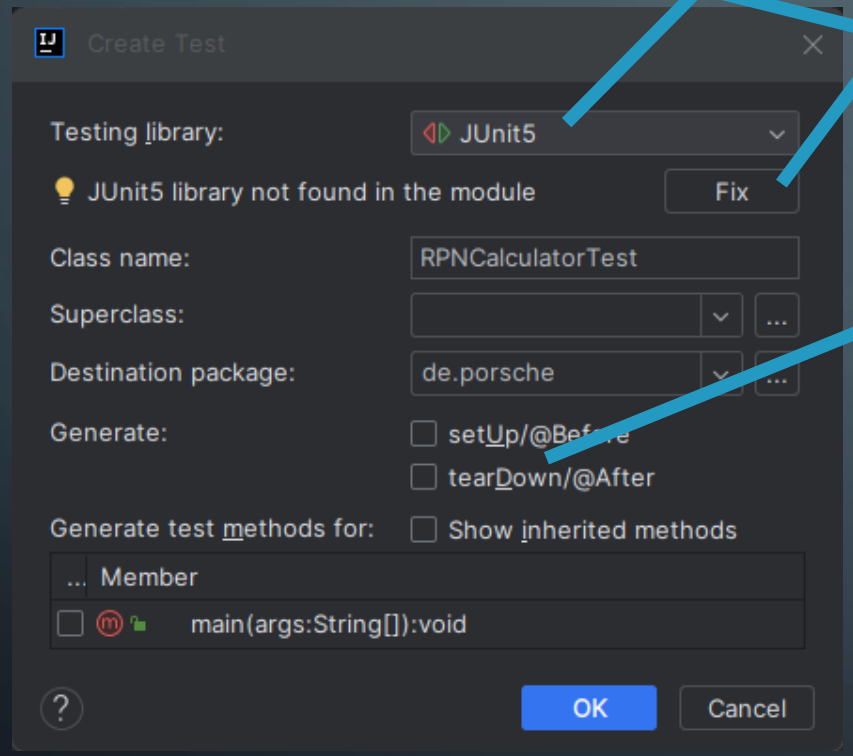


UNIT TESTING - INTELLIJ

Part 2



UNIT TESTING - INTELLIJ



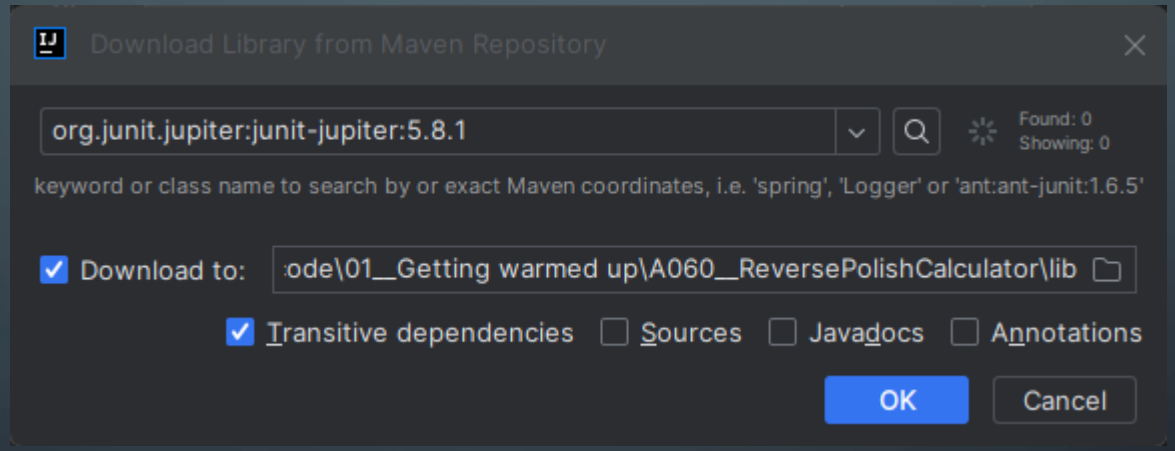
Use Junit 5

Fix it (see next slide)

Generate setup / teardown

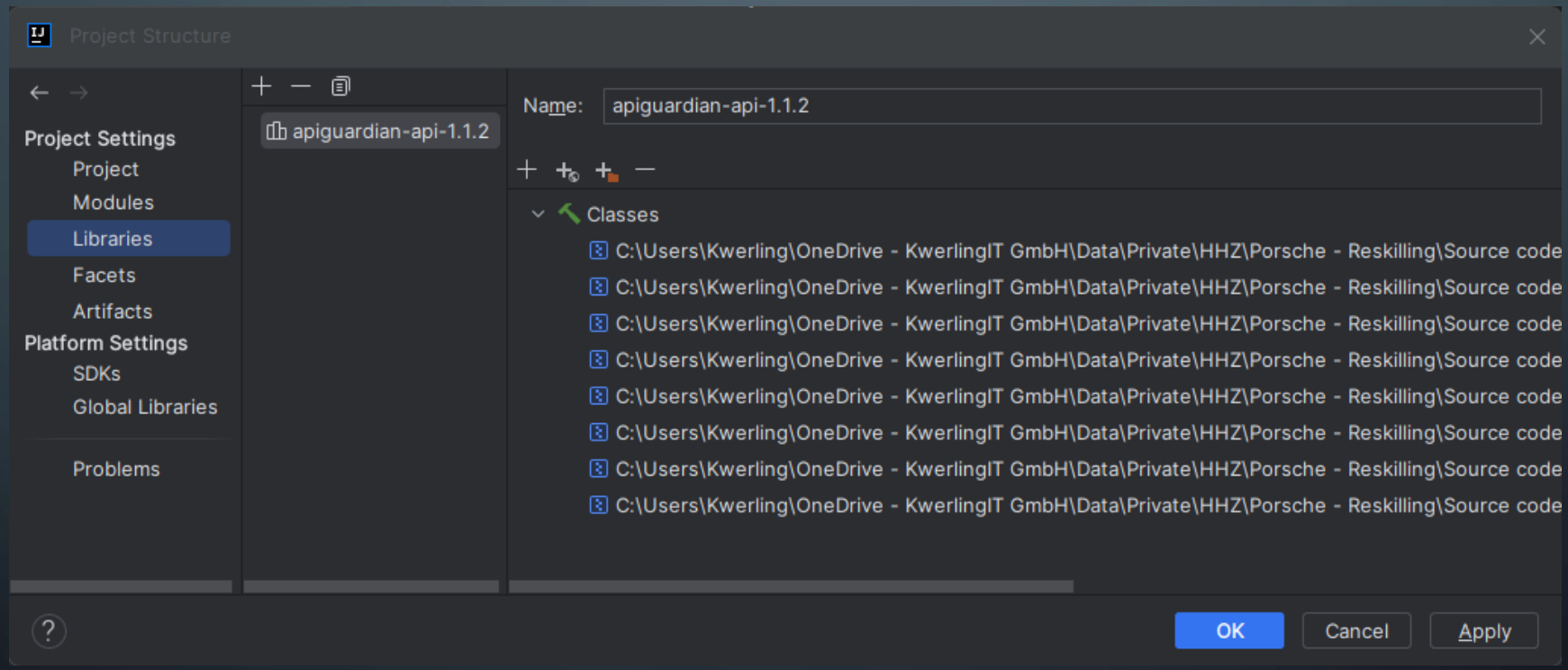
UNIT TESTING - INTELLIJ

Part 4



UNIT TESTING - INTELLIJ

Part 5



UNIT TESTING - INTELLIJ

Part 5

```
package de.porsche;

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

class RPNCalculatorTest {

    @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
assertEquals(exp, act [, optionalMsg])	exp is required to be the same as act
assertTrue(cond [, optionalMsg])	cond is required to be true
assertNotNull(obj [, optionalMsg])	obj is required to not be null

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

REVERSE POLISH NOTATION (UPN)

Part 1

```
package de.porsche;

import java.util.Stack;

public class RPNCalculator {

    public static void main(String[] args) {
        RPNCalculator rpnCalculator = new RPNCalculator();

        int result = rpnCalculator.processCommandLineParams(args);



        // Print the result to the console
        System.out.println("The result is: " + result);
    }
```

REVERSE POLISH NOTATION (UPN)

Part 2

```
int processCommandLineParams(String[] args) {
   // Create a stack to store the operands
   Stack<Integer> stack = new Stack<>();

   // Iterate over the command line parameters
   for (String arg : args) {
      // If the argument is an operand, push it onto the stack
      if (isOperand(arg)) {
         stack.push(Integer.parseInt(arg));
      } else {
         // If the argument is an operator, pop the top two elements of the stack, perform the operation, and push the result back onto the stack
         doCalculation(stack, arg);
      }
   }
   // The result of the calculation is now on the top of the stack
   return stack.pop();
}
```

REVERSE POLISH NOTATION (UPN)

Part 3

```
void doCalculation(Stack<Integer> stack, String arg) {
   int operand2 = stack.pop();
   int operand1 = stack.pop();
   int result = performOperation(arg, operand1, operand2);
   stack.push(result);
}

boolean isOperand(String arg) {
   // An operand is a string that does not contain any of the following characters: +, -, *, /
   return !arg.contains("+") && !arg.contains("-") && !arg.contains("*") && !arg.contains("/");
}
```

REVERSE POLISH NOTATION (UPN)

Part 4

```
  int performOperation(String operator, int operand1, int operand2) {
    // Perform the specified operation on the two operands and return the result
    switch (operator) {
      case "+":
        return operand1 + operand2;
      case "-":
        return operand1 - operand2;
      case "*":
        return operand1 * operand2;
      case "/":
        return operand1 / operand2;
      default:
        throw new IllegalArgumentException("Unknown operator: " + operator);
    }
  }
}
```

REVERSE POLISH NOTATION (UPN) TESTS

Part 1

```
package de.porsche;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import java.util.EmptyStackException;
import java.util.Stack;

class RPNCalculatorTest {

    RPNCalculator rpnCalculator = null;

    @org.junit.jupiter.api.BeforeEach
    void setUp() {
        rpnCalculator = new RPNCalculator();
    }

    @org.junit.jupiter.api.AfterEach
    void tearDown() {
        rpnCalculator = null;
    }
```

REVERSE POLISH NOTATION (UPN) TESTS

Part 2

```
@Test
public void testIsOperand() {
   assertTrue(              rpnCalculator.isOperand("1"));
   assertEquals(true, rpnCalculator.isOperand("123"));
   assertFalse(             rpnCalculator.isOperand("+"));
   assertEquals(false, rpnCalculator.isOperand("-"));
   assertEquals(false, rpnCalculator.isOperand("*"));
   assertEquals(false, rpnCalculator.isOperand("/"));
}

@Test
public void testPerformOperation() {
   assertEquals(3, rpnCalculator.performOperation("+", 1, 2));
   assertEquals(1, rpnCalculator.performOperation("-", 2, 1));
   assertEquals(4, rpnCalculator.performOperation("*", 2, 2));
   assertEquals(2, rpnCalculator.performOperation("/", 4, 2));
}
```

REVERSE POLISH NOTATION (UPN) TESTS

Part 3

```
@Test
public void testProcessCommandLineParams() {
    String[] args = new String[] { "1", "2", "+" };
   assertEquals(3, rpnCalculator.processCommandLineParams(args));


   args = new String[] { "2", "3", "*", "4", "+" };
   assertEquals(10, rpnCalculator.processCommandLineParams(args));
}

@Test
public void testProcessCommandLineParams_InvalidOperator() {
   String[] args = new String[] { "1", "2", "%" };

   assertThrows(IllegalArgumentException.class, () -> {
      rpnCalculator.processCommandLineParams(args);
   });
}
```

REVERSE POLISH NOTATION (UPN) TESTS

Part 4

```
@Test
public void testProcessCommandLineParams_OneOperand() {
    String[] args = new String[] { "1" };

   assertEquals(1,  rpnCalculator.processCommandLineParams(args));
}

@Test
public void testDoCalculation() {
   Stack<Integer> stack = new Stack<>();

   stack.push(1);
   stack.push(2);

   rpnCalculator.doCalculation(stack, "+");

   assertEquals(3, stack.pop());
}
```

REVERSE POLISH NOTATION (UPN) TESTS

Part 5

```
    @Test
    public void testDoCalculation_EmptyStack() {
        Stack<Integer> stack = new Stack<>();

        assertThrows(EmptyStackException.class, () -> {
            rpnCalculator.doCalculation(stack, "+");
        });
    }

    @Test
    public void testDoCalculation_OneOperand() {
        Stack<Integer> stack = new Stack<>();

        stack.push(1);

        assertThrows(EmptyStackException.class, () -> {
            rpnCalculator.doCalculation(stack, "+");
        });
    }
}
```

REVERSE POLISH NOTATION CALCULATOR

Project name	Explanation
A059 …	RPN Calculator with Classes and Interfaces
A060 …	RPN Calculator (almost) procedurally
A061 …	Enhanced RPN Calculator with Classes: • 1 Level If Statement • Comparison Operators: < <= > >= = == • Commands: . D. DUP DROP
A062 …	Enhanced RPN Calculator with Classes: • Nested If Statements • Commands: ." " CR
A063 …	Enhanced RPN Calculator with Classes: • Nested For loops • Commands: SP TB I I-(1..5)

REVERSE POLISH NOTATION COMMANDS

Command	Explanation

1
10
<=
If
   <commands>
Else
   <commands>
Then
	Compare 1 <= 10
	
	Put result (0=>false, other value0>true) on stack
	If true
	   run <commands>
	Else
	   run <commands>
	End of if statement

REVERSE POLISH NOTATION COMMANDS

Command	Explanation
1 dup 10 <= If 5 <= If <commands> Then <commands> Else <commands> Then	Of course, If statements can be nested

REVERSE POLISH NOTATION COMMANDS

Command	Explanation
11 1 Do <commands> Loop	Limit Index Start For Loop Run <commands> End For Loop The Loops runs from one (index) up in steps of 1 to 10. The limit is is 1 greater than the last loop Value

REVERSE POLISH NOTATION COMMANDS

Command	Explanation
11 1 Do 11 1 Do <commands> Loop <commands> Loop	For Loops can be nested, too

REVERSE POLISH NOTATION COMMANDS

Command	Explanation
11 1 Do ." Run # " I . CR <commands> Loop	I Puts the index (running counter of the For Loop) onto the stack. This is different from FORTH: I-1, I-2, I-3, I-4 and I-5 Allow to access the index variables of nested For Loops (see example on next page)

REVERSE POLISH NOTATION COMMANDS

Command	Explanation
11 1 Do 11 1 Do I I-1 * . Loop Loop	Print the results of the multiplication table ("Kleines 1 mal 1")

REVERSE POLISH NOTATION COMMANDS

Command	Explanation
11 1 Do 11 1 Do I I-1 * . Loop Loop	Print the results of the multiplication table ("Kleines 1 mal 1")

REVERSE POLISH NOTATION CALCULATOR

- Further enhancements can be added as an exercise:
  - While or Repeat-Until loops
  - Register to store values. Ex: Sto-A Rcl-A AddTo-A SubFrom-A (Registers: A – Z)
    For improvement allow for flexible (multi-chars) names for registers.
  - Other mathematical functions
  - Add new Word dynamically – These are similar to functions in other languages

Try to get inspiration from the FORTH programming language. Most of the constructs implemented are close to or exactly like their counterparts in FORTH.

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

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

Part 1

```
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

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

Part 1

```
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 an another computer.

RESTRICTIONS OF THE CODE

- The server shown here is a single connection server.
- Connections can only be established when there is no existing one
- For a multi-connection server one needs to make use of threads
- The concepts still remain the same:
  - The server is listing on a port on a computer for client connections
  - The client connection is established
  - The client is served
  - The connection get closed when the server is done

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 man 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 to 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.

RESTFUL APIS – WEB SERVICES

- Very simple Main class
- Everything happens inside of the PostmanEchoClient class
- There is even no need to have the instance assigned to a local variable

Part 1

```
package de.porsche;

public class Main {

    public static void main(String[] args) {

        new PostmanEchoClient();

    }
}
```

RESTFUL APIS – WEB SERVICES

- It is the constructor calling all the methods in the class

Part 2

```
package de.porsche;

import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;

public class PostmanEchoClient {

    public PostmanEchoClient() {
        HttpClient client = HttpClient.newHttpClient();
        try {
            peGet(client);
            peGetWithParameter(client);
            peMyIP(client);
            pePost(client);
        } catch (Exception ex) {
            System.err.println(ex.getMessage());
            ex.printStackTrace();
        }
    }
```

RESTFUL APIS – WEB SERVICES

- printResult is a helper routine, which is used by every other method.
- As it does not provide any service outside the class it could (or should?) have been made private.

Part 3

```
void printResult( String methodName, HttpResponse<String> response) {
   System.out.println(methodName + ": " + response.body());
   System.out.println(methodName + " status code: " + response.statusCode());
   System.out.println();
}

 void peGet(HttpClient client) throws URISyntaxException, IOException, InterruptedException {
   HttpRequest request = HttpRequest.newBuilder()
         .uri( new URI("https://postman-echo.com/get") )
         .GET()
         .build();

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

   printResult("peGet", response);
}
```

RESTFUL APIS – WEB SERVICES

- Another two GET requests

Part 4

```
void  peGetWithParameter(HttpClient client) throws URISyntaxException, IOException, InterruptedException
{
   HttpRequest request = HttpRequest.newBuilder()
         .uri( new URI("https://postman-echo.com/get?Eins=1") )
         .GET()
         .build();

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

   printResult("peGetWithParameter", response);
}

void  peMyIP(HttpClient client) throws URISyntaxException, IOException, InterruptedException {
   HttpRequest request = HttpRequest.newBuilder()
         .uri( new URI("https://postman-echo.com/ip") )
         .GET()
         .build();

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

   printResult("peMyIP", response);
}
```

RESTFUL APIS – WEB SERVICES

Part 5

- POST requests require special handling recording the data to be posted.

```
    void pePost(HttpClient client) throws URISyntaxException, IOException, InterruptedException {
      HttpRequest request = HttpRequest.newBuilder()
            .uri( new URI("https://postman-echo.com/post") )
            .headers("Content-Type", "text/plain;charset=UTF-8")
            .POST(HttpRequest.BodyPublishers
                  .ofString("Greeting to the Postman Web Service."))
            .build();

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

      printResult("pePost", response);
    }

}
```

DOCKER

Docker is a lightweight, portable containers for packaging and running applications

DOCKER

Feature	Explanation
Containerization	Packages applications and their dependencies into self-sufficient units called containers.
Lightweight	Containers share the host operating system's kernel, making them more efficient than VMs.
Portability	Containers run consistently across different environments, from local machines to cloud platforms.
Isolation	Containers are isolated from each other, preventing conflicts and resource contention.
Scalability	Containers can be easily scaled up or down based on demand.
Reproducibility	Applications deployed in containers are always deployed in the same environment, ensuring reproducibility.
Automation	Provides tools for automating the build, deployment, and management of containers.
Popular ecosystem	Has a large and active community with a wide range of tools and integrations.
Wide adoption	Used by millions of developers and organizations worldwide.

COMPARISON OF DOCKER WITH VMS

Feature	Docker	Virtual Machine
Resource usage	Lightweight, shares the host kernel	More resource-intensive, runs its own operating system
Portability	Highly portable, runs consistently across environments	Less portable, may require environment configuration
Isolation	Isolated from each other, preventing conflicts	Isolated from each other (more overhead, separate OS)
Startup time	Fast startup time	Slower startup time due to OS initialization
Scalability	Easily scalable up or down	Scalable, but may require more resource management
Reproducibility	High reproducibility due to consistent environment	Reproducibility depends on underlying OS
Complexity	Simpler to manage and deploy	More complex to manage and deploy
Ecosystem	Large and active ecosystem	Wide range of tools and integrations

MESSAGE BUS

- Assumptions for the following slides:
  - Docker Desktop is installed on your computer

    With Docker Desktop 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.

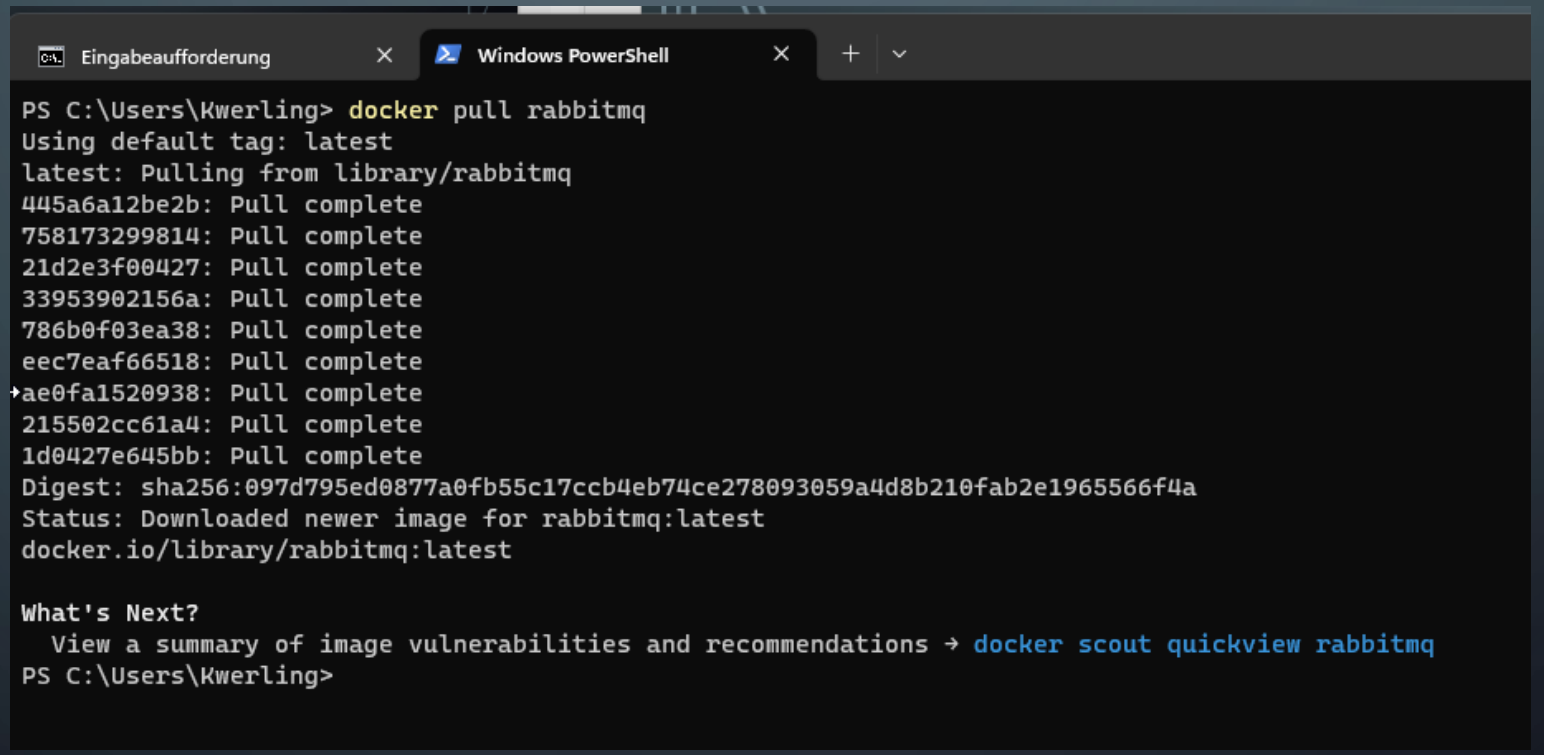
MESSAGE BUS

- Install RabbitMQ on your machine

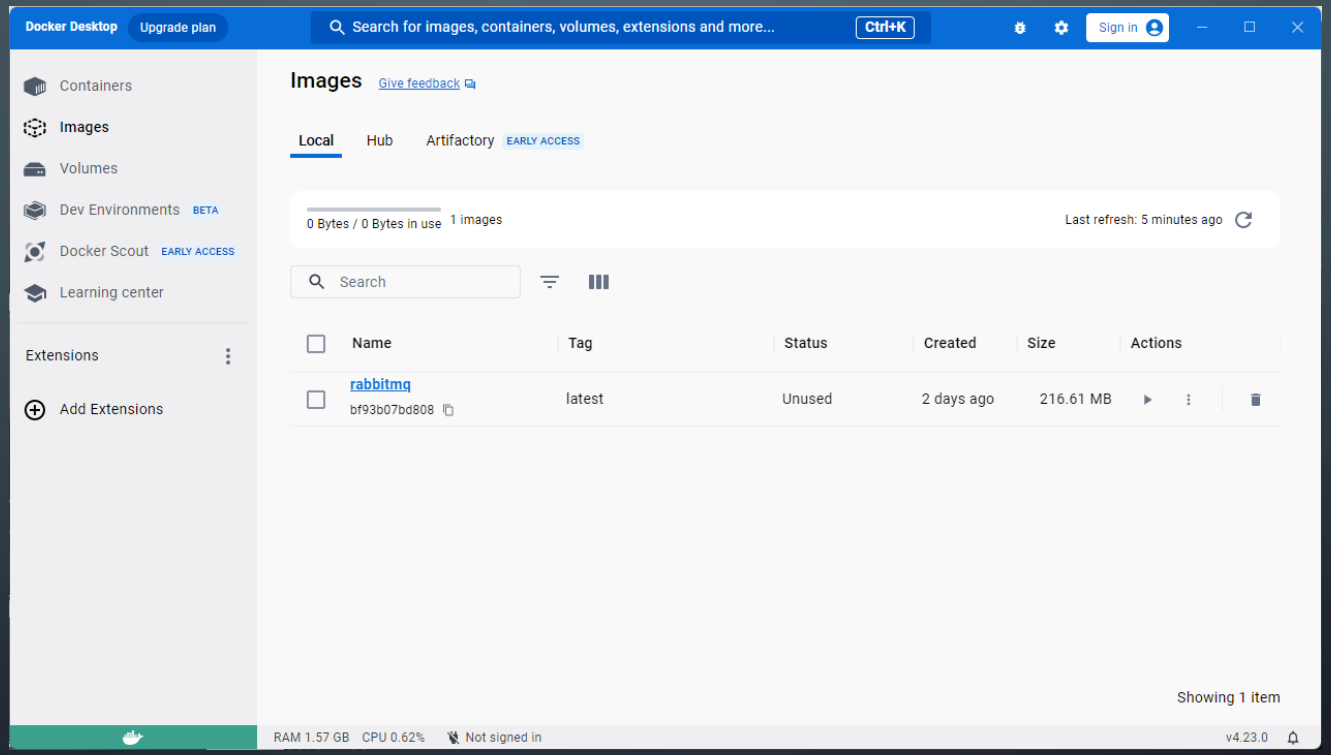
Command line: docker pull rabbitmq

- This will take a little while

MESSAGE BUS


Eingabeaufforderung
Windows PowerShell
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


Docker Desktop
Upgrade plan
Search for images, containers, volumes, extensions and more...
Ctrl+K
Sign in
Containers
Images
Volumes
Dev Environments
BETA
Docker Scout
EARLY ACCESS
Learning center
Extensions
Add Extensions
Images
Give feedback
Local
Hub
Artifactory
EARLY ACCESS
0 Bytes / 0 Bytes in use
1 images
Last refresh: 5 minutes ago
Search
Name
Tag
Status
Created
Size
Actions
rabbitmq
bf93b07bd808
latest
Unused
2 days ago
216.61 MB
Showing 1 item
RAM 1.57 GB
CPU 0.62%
Not signed in
v4.23.0

MESSAGE BUS

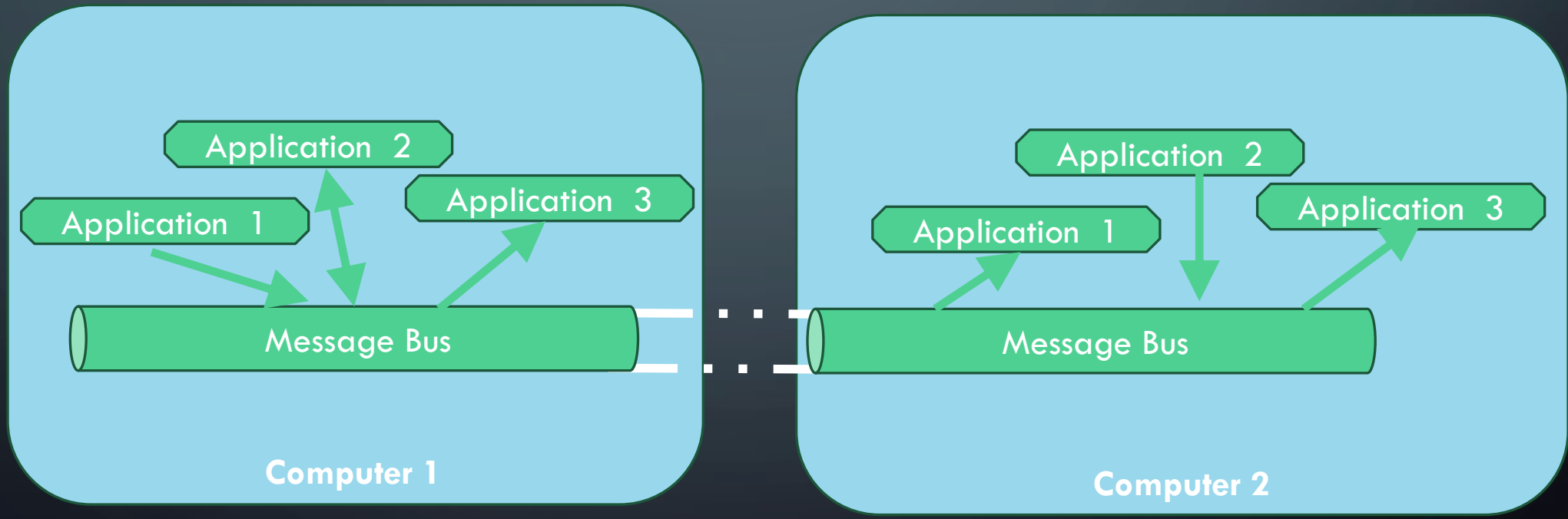
- Features:
  - Applications communicate through messages via the bus
  - The bus architecture contains a message broker as central element
  - The broker receives, routes, and delivers the messages
  - Because of the broker the message sender / receiver are only loosely coupled
  - The communication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

MESSAGE BUS

Either	Or	Or
Sender	Producer	Publisher
Receiver	Consumer	Subscriber

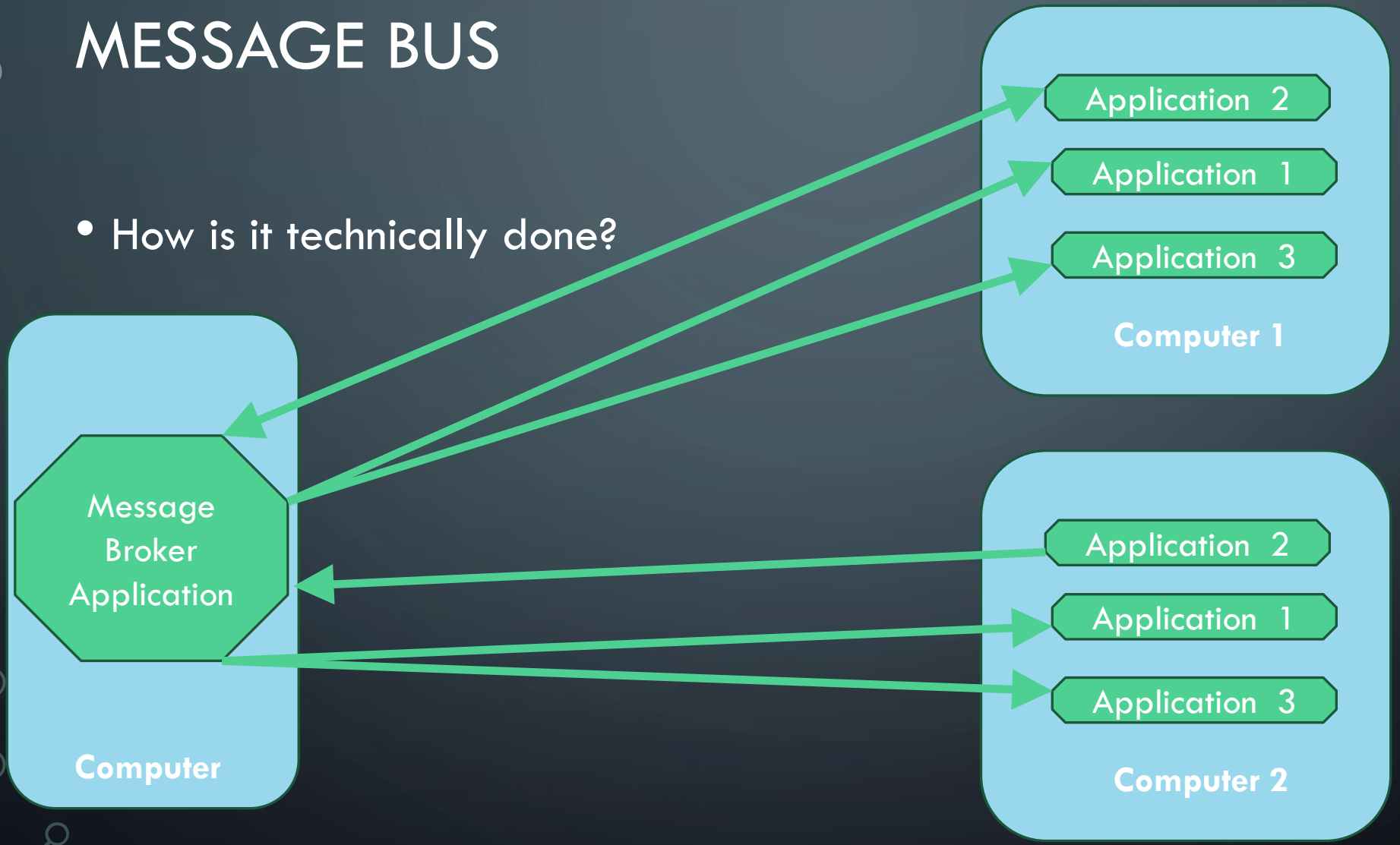
MESSAGE BUS

- How to picture a message bus?



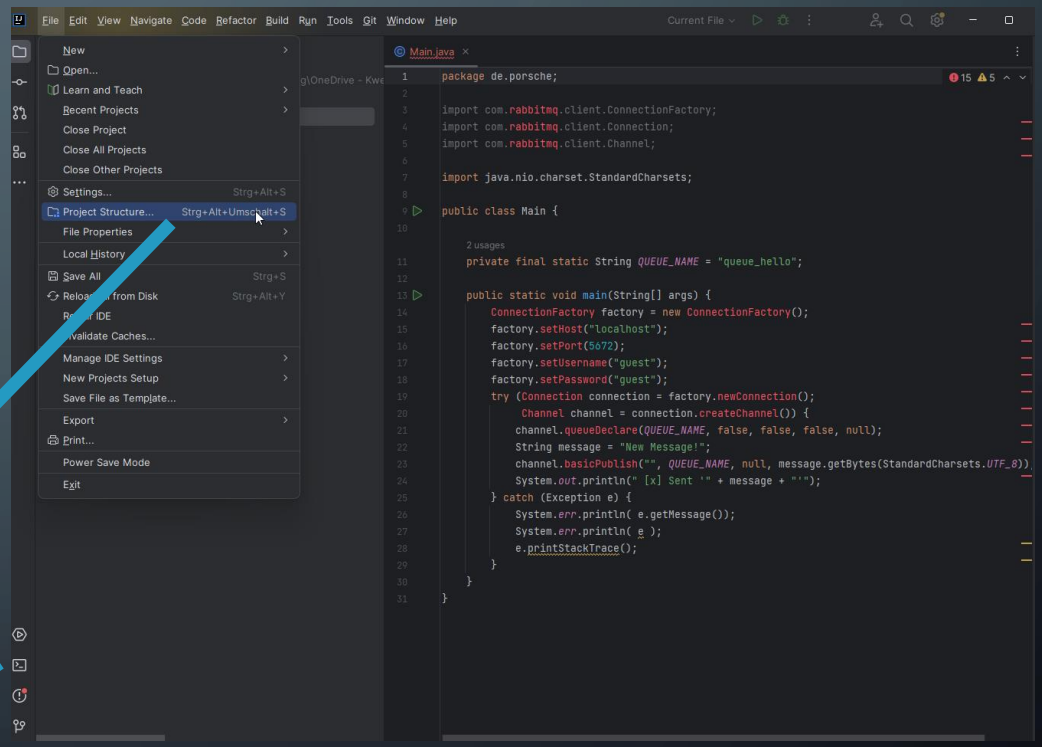
MESSAGE BUS

- How is it technically done?



MESSAGE BUS

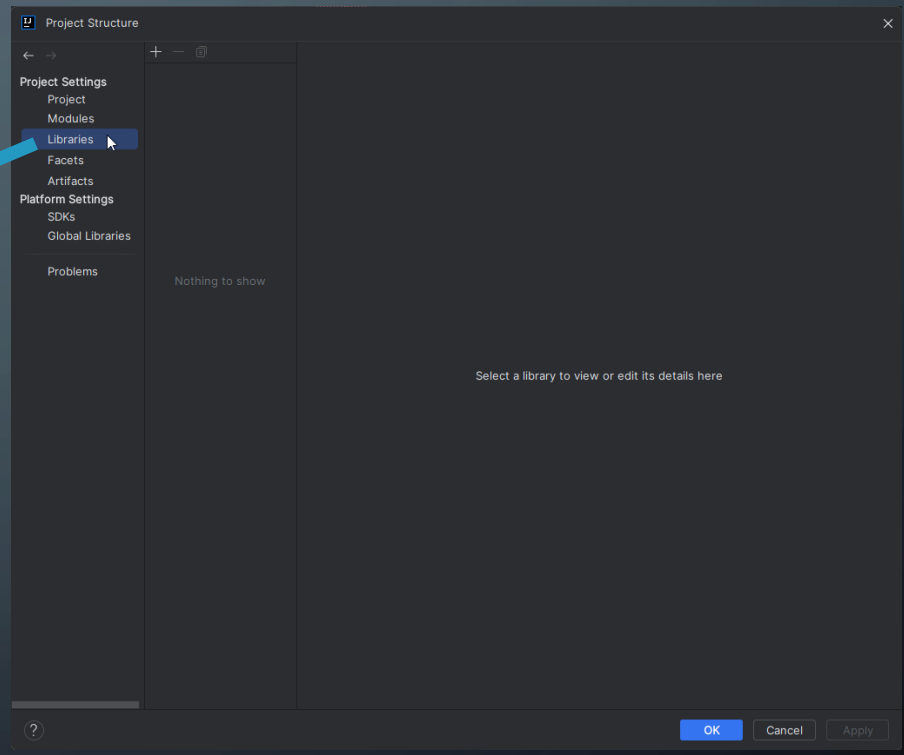
- In order to run the RabbitMQ Consumer and Producer code, the RabbitMQ java library needs to be added to the project.



File -> Project Structure

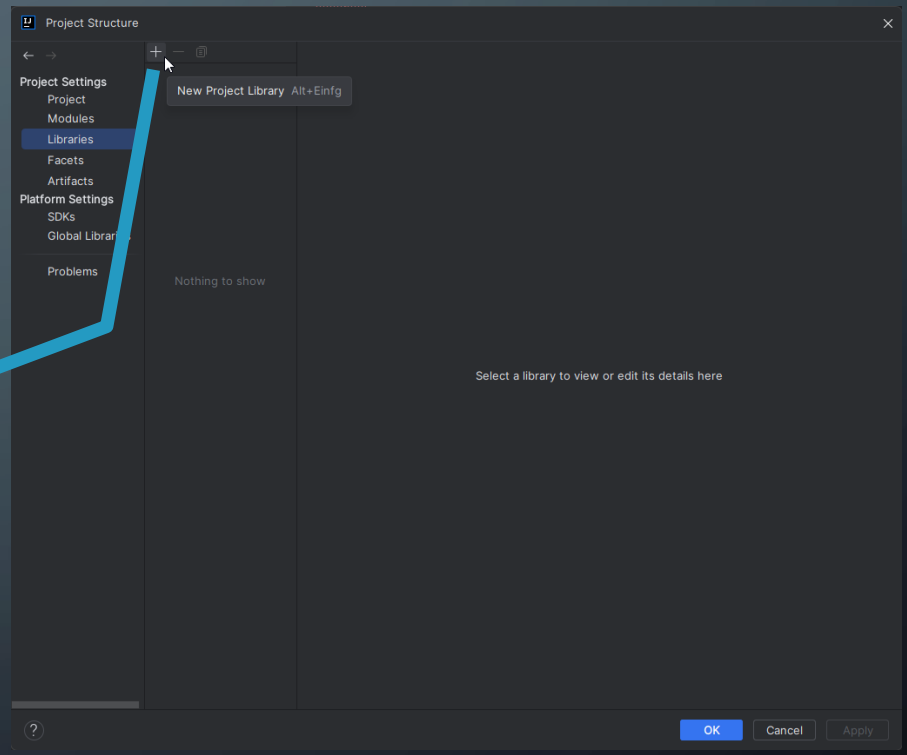
MESSAGE BUS

Libraries



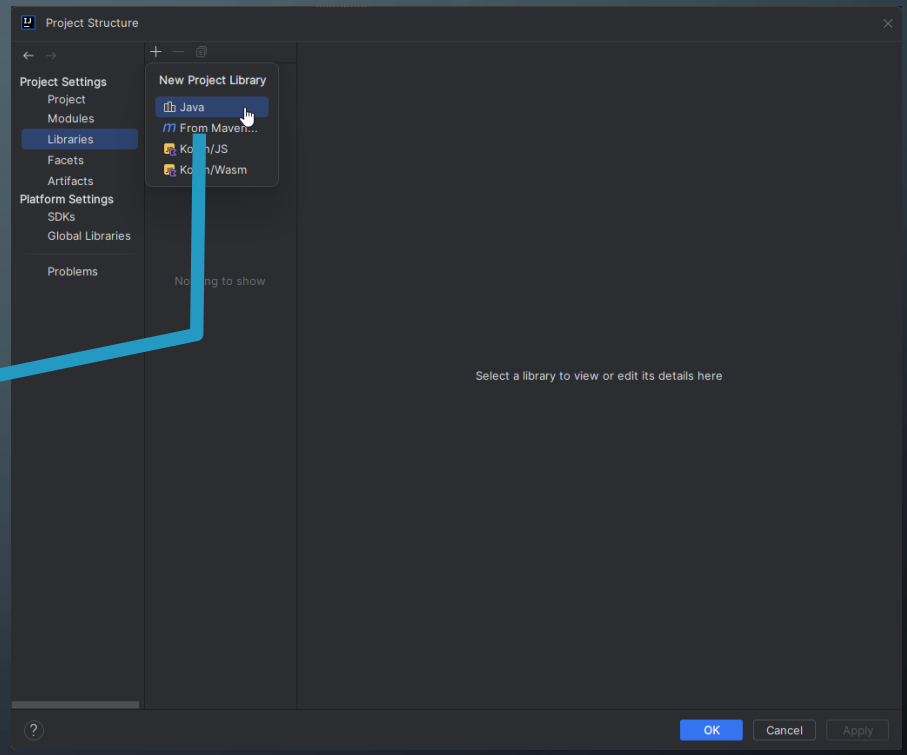
MESSAGE BUS

Add (+) button



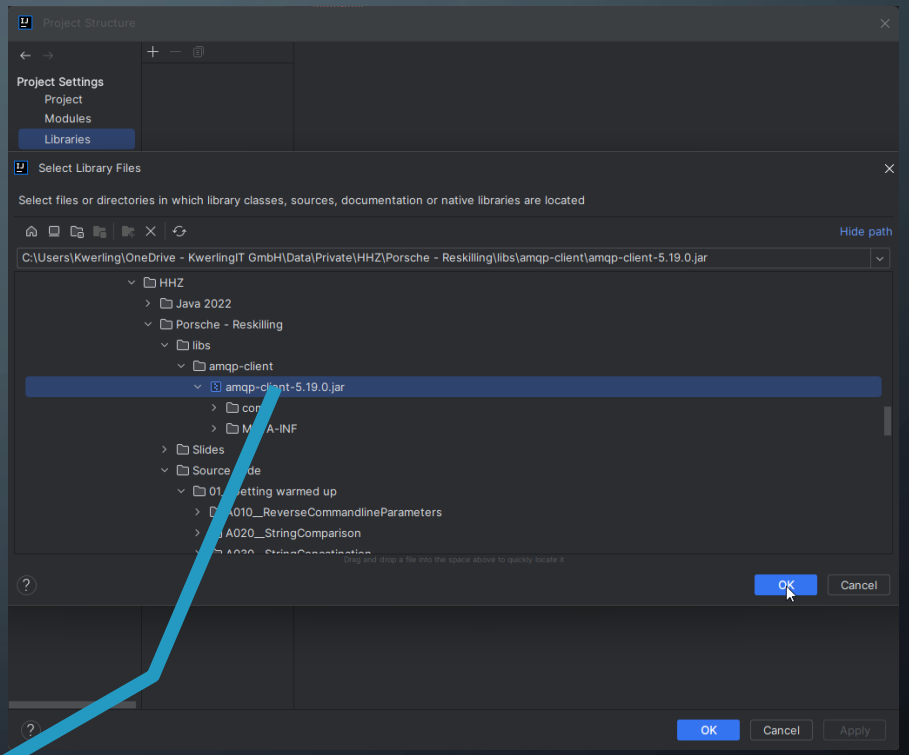
MESSAGE BUS

Choose Jave library



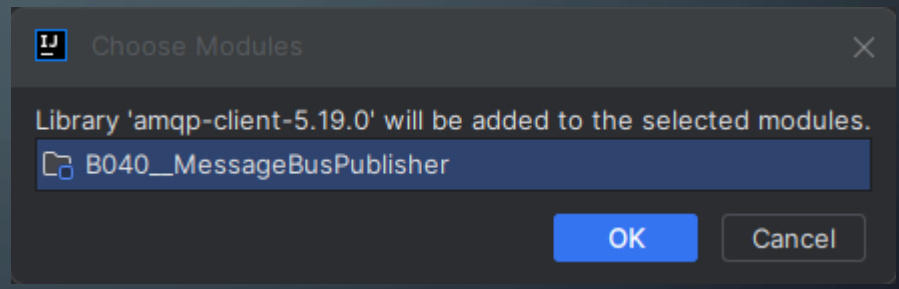
MESSAGE BUS

- 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

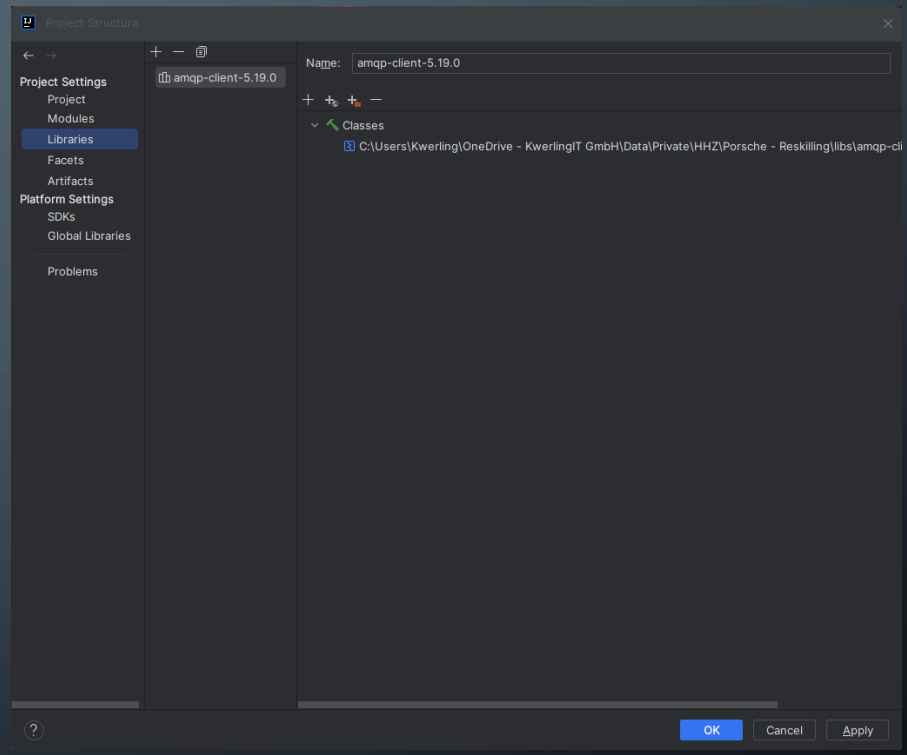


Libraries

MESSAGE BUS

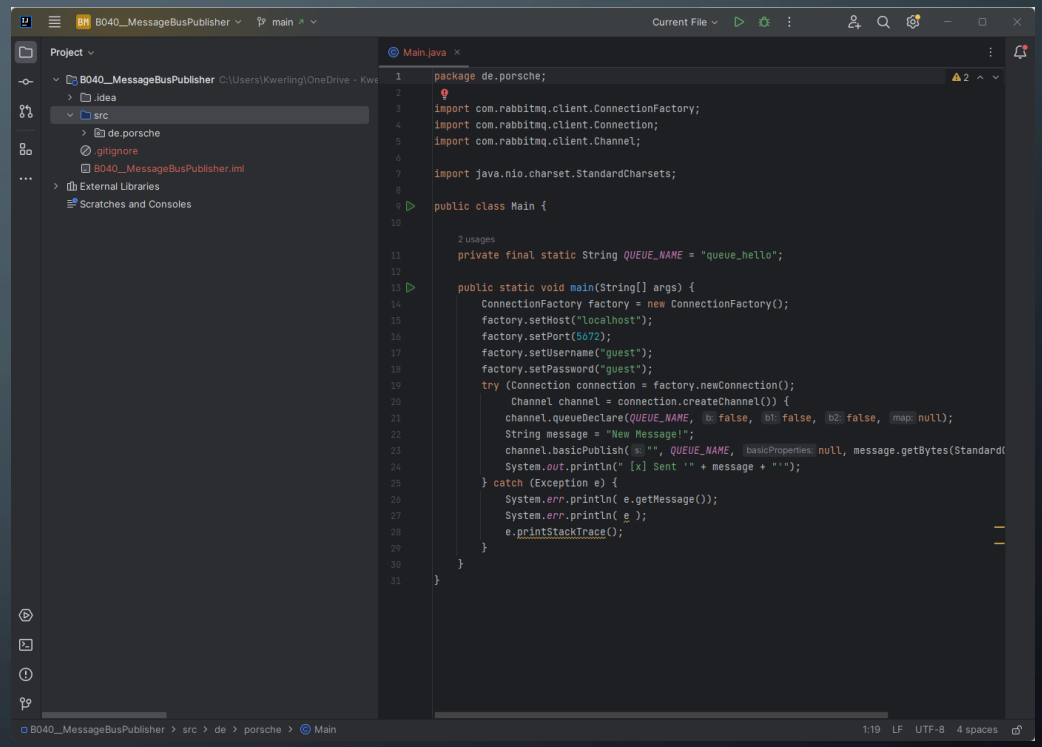

Choose Modules
Library 'amqp-client-5.19.0' will be added to the selected modules.
B040__MessageBusPublisher
OK
Cancel

MESSAGE BUS


Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
amqp-client-5.19.0
Name:
amqp-client-5.19.0
Classes
C:\Users\Kwerling\OneDrive - KwerlingIT GmbH\Data\Private\HHZ\Porsche - Reskilling\libs\amqp-cli
OK
Cancel
Apply

MESSAGE BUS

- All classes and methods should be known now.



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";
```

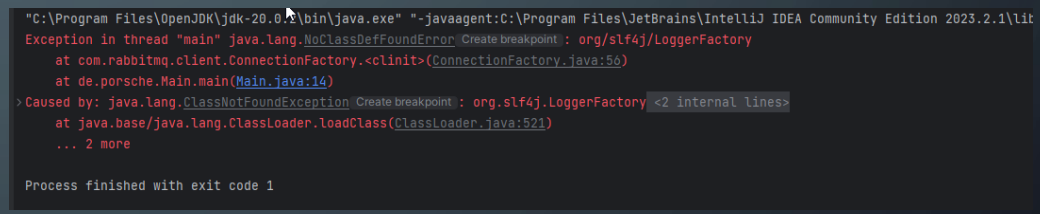
MESSAGE BUS

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

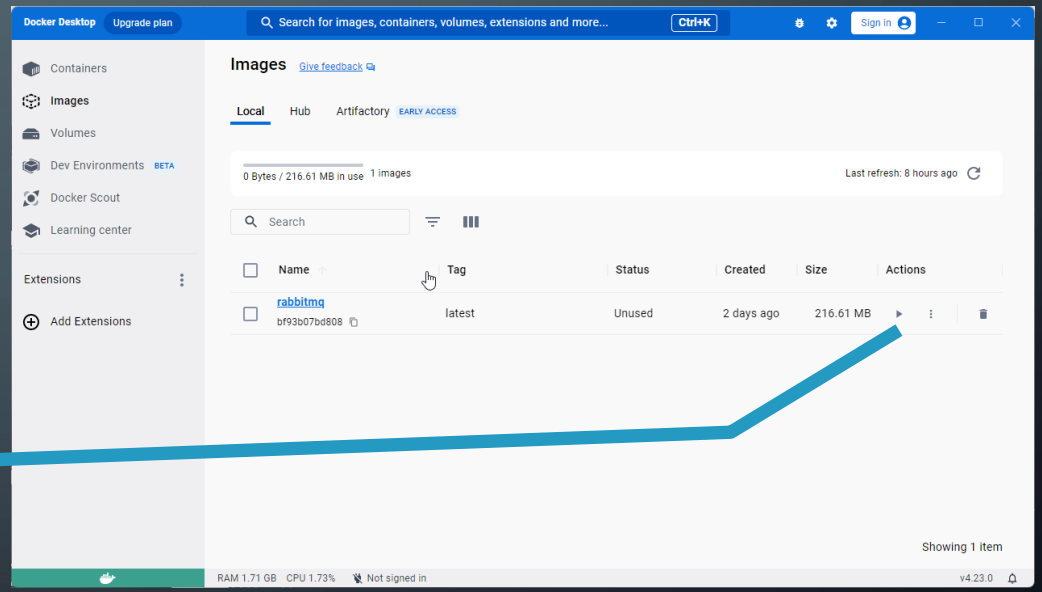
- No compilation issues
- But when trying to run the error to the right appears
- Reason is, that there is a dependency on the slf4j libraries for logging
- Add them to the project



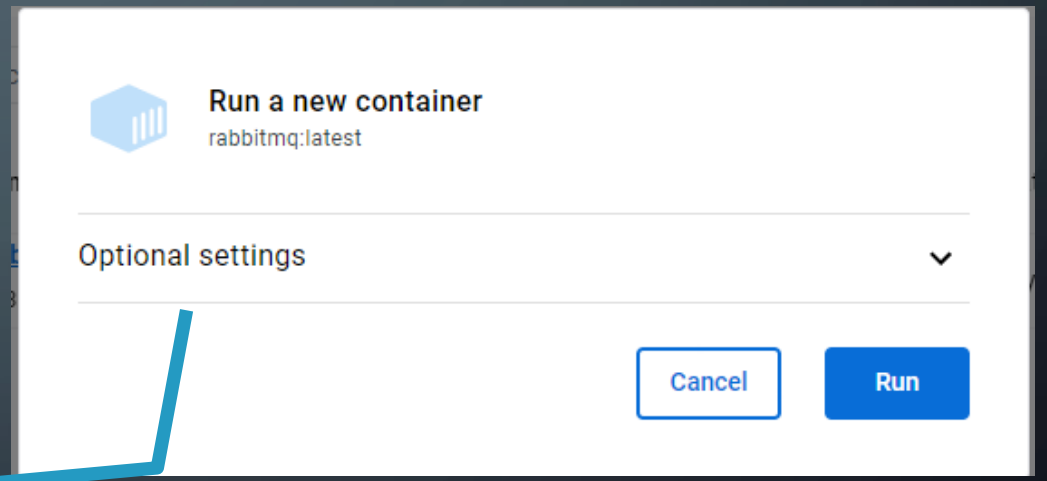
MESSAGE BUS

- Now it is time to start RabbitMQ

Click the run button



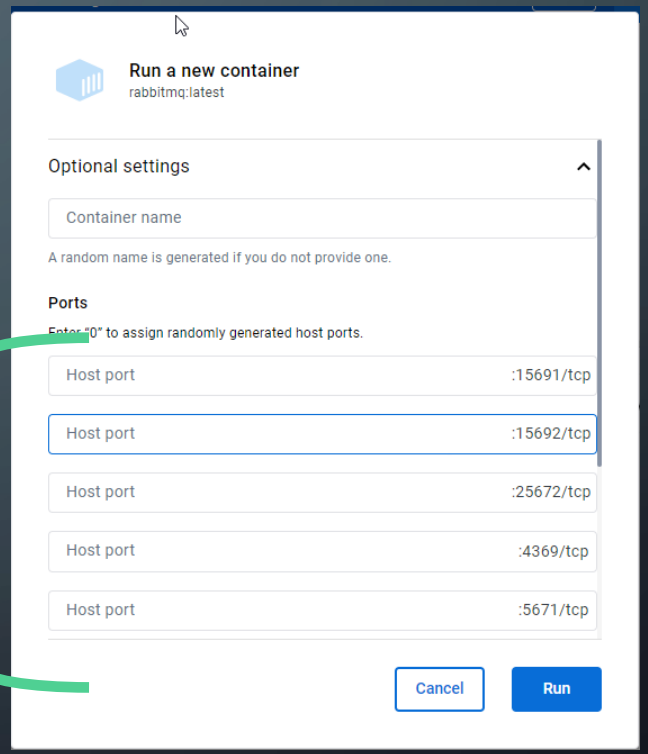
MESSAGE BUS



Click on Optional settings

MESSAGE BUS

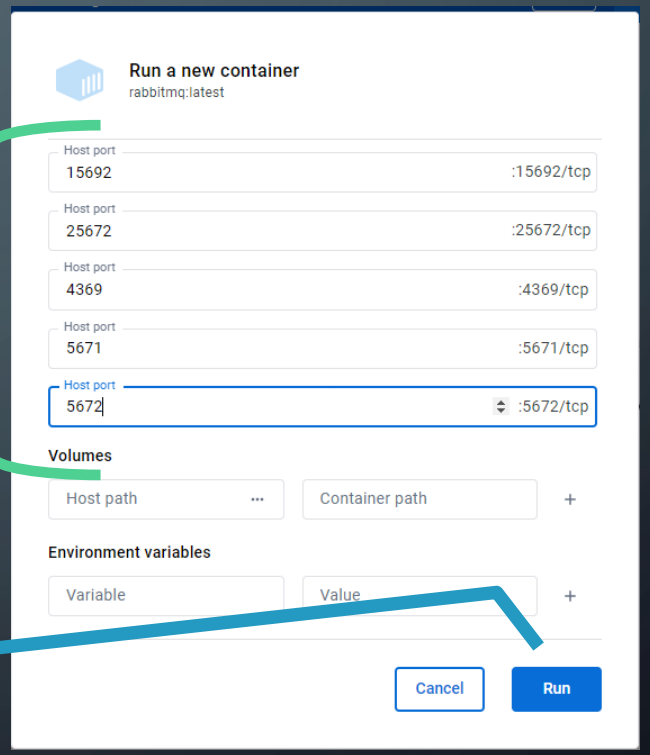
Enter the same local ports as RabbitMQ uses



MESSAGE BUS

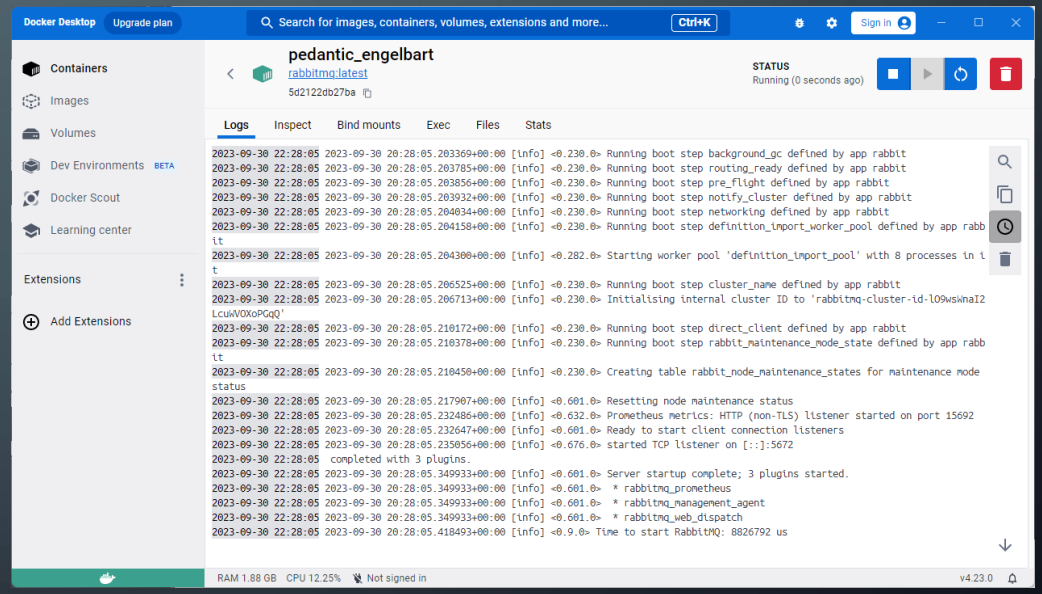
Continue – leave the other fields empty

Run



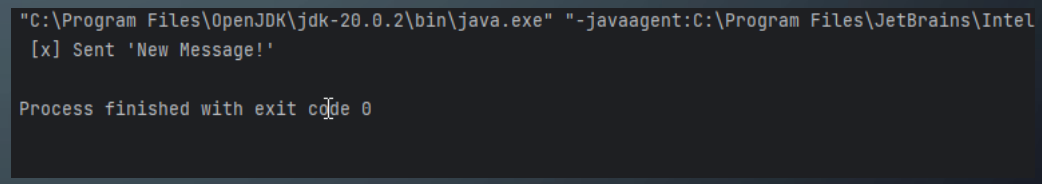
MESSAGE BUS

- 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



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, b: false, b1: false, b2: false, map: null);
            String message = "New Message!";
            channel.basicPublish( s: "", QUEUE_NAME, basicProperties: 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)

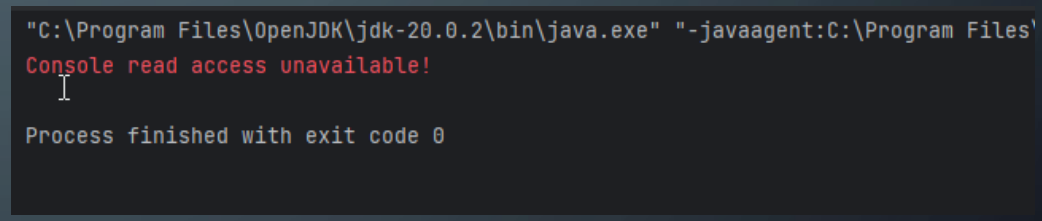
GETTING KEYBOARD INPUT IN CONSOLE APP

- Use the Console class to read keyboard input

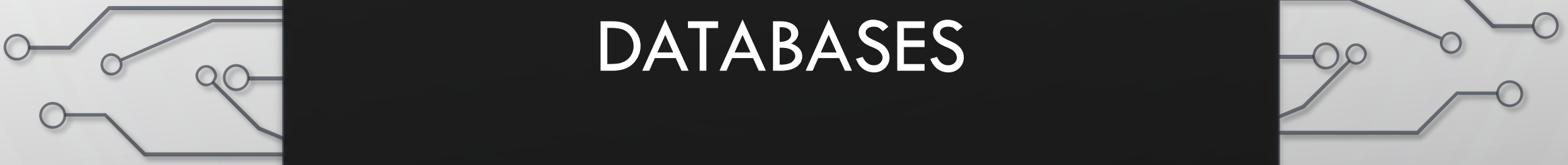
```
package de.porsche;

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__ReadingKeyboardInputInConsoleApp directory
- Run java –cp . de.porsche.Main



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: • Atomicity • Consistency • Isolation • Durabilty	Some do, others don't. There are NoSQL DBs, which sacrifice ACID for speed (wordplay ☺)
Database language	SQL: • Data Definition Language (DDL) • Data Manipulation Language (DML) • Data Control Language (DCL) • Data Query Language (DQL)	There is no standard similar to SQL for No-SQL databases. Each Database Management System hast ist 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	

Orders		
Id	Integer	PK
OrderDate	Date	
CustomerId	Integer	FK
TotalAmount	Double	

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

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	

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	

EXERCISE: C010 - DATABASES

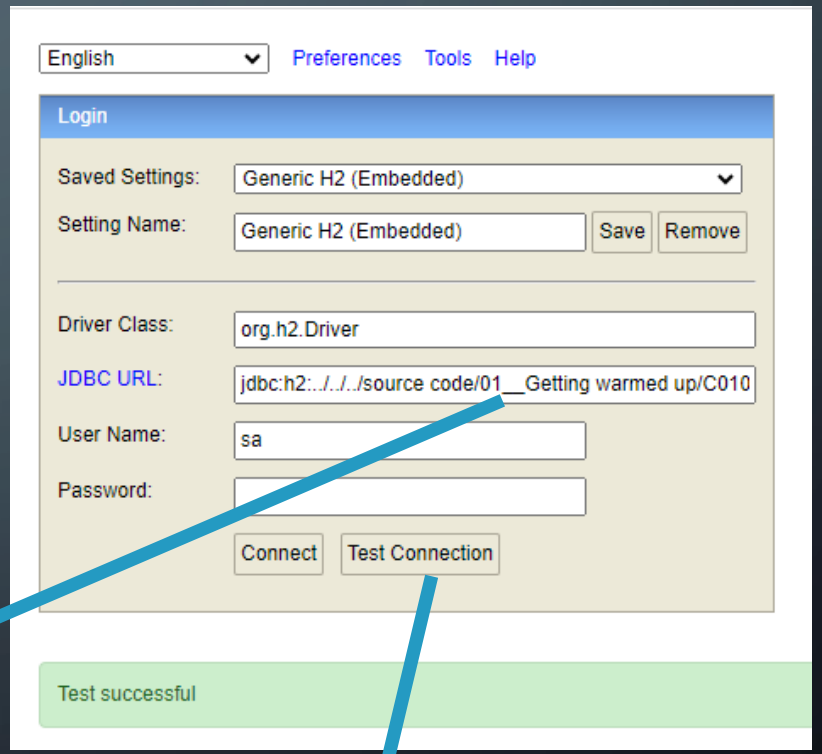
- Run the code in C010__CreateAndFillDB
- When it runs without error change the DB_URL to

```
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 databas.

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)

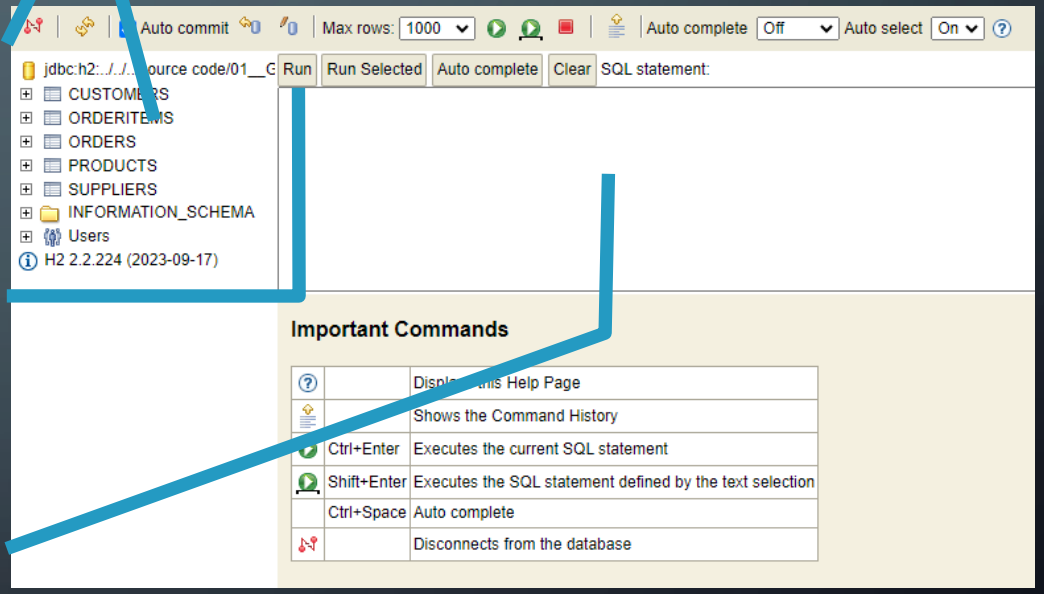
Verify the correctness of data with "Test Connection"

DATABASES

Here are the tables of the DB

Click "Run" button to execute a SQL statement

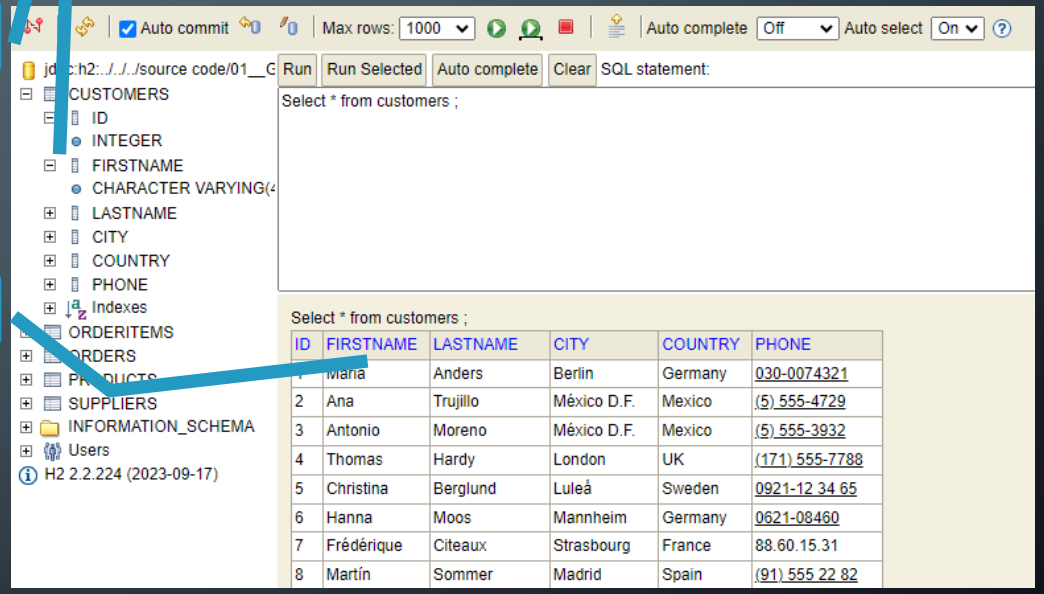
Here you enter the SQL statements



DATABASES

Details on the attributes of a table

Result of a run query



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 the 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>>** :

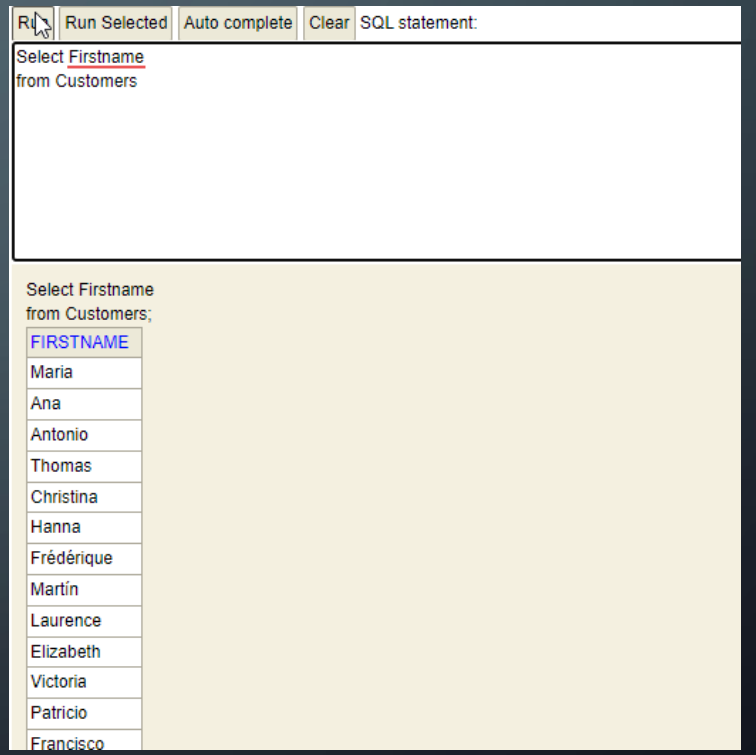
<<comparison>>		**Result**
=	'Hanna'	6 Hanna Moos Mannheim Germany 0621-08460
Like	'Hanna'	6 Hanna Moos Mannheim Germany 0621-08460
Like	'Ha_na'	6 Hanna Moos Mannheim Germany 0621-08460
=	'Ha_na'	(no rows, 1 ms)

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'

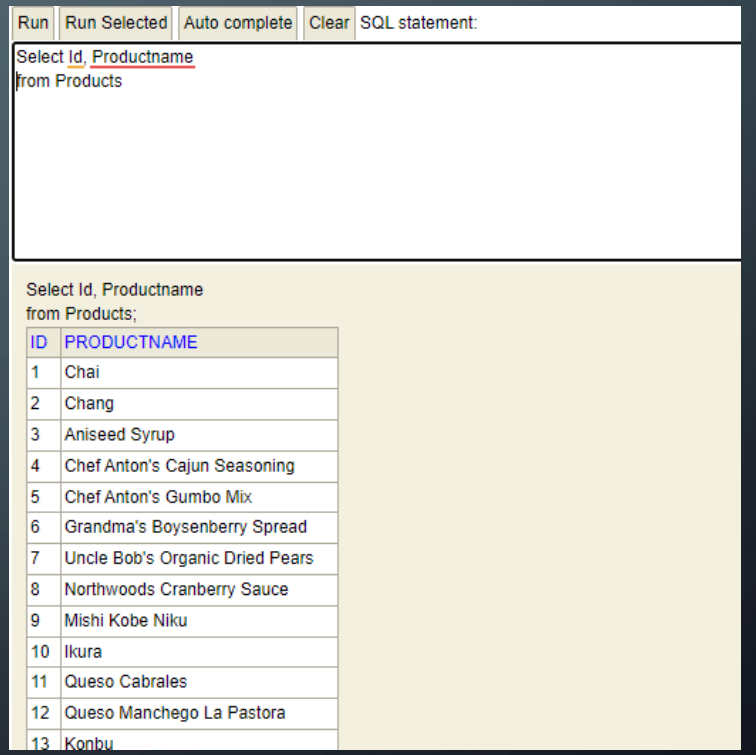
DATABASES – SELECT STATEMENT

- Lists all first names from the customers table



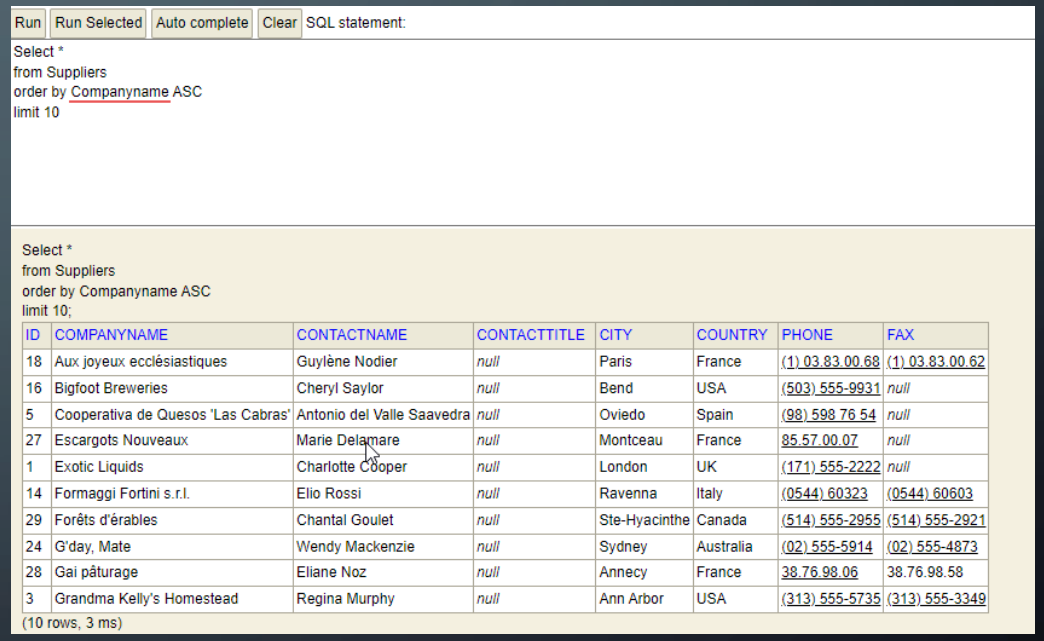
DATABASES – SELECT STATEMENT

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



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
```

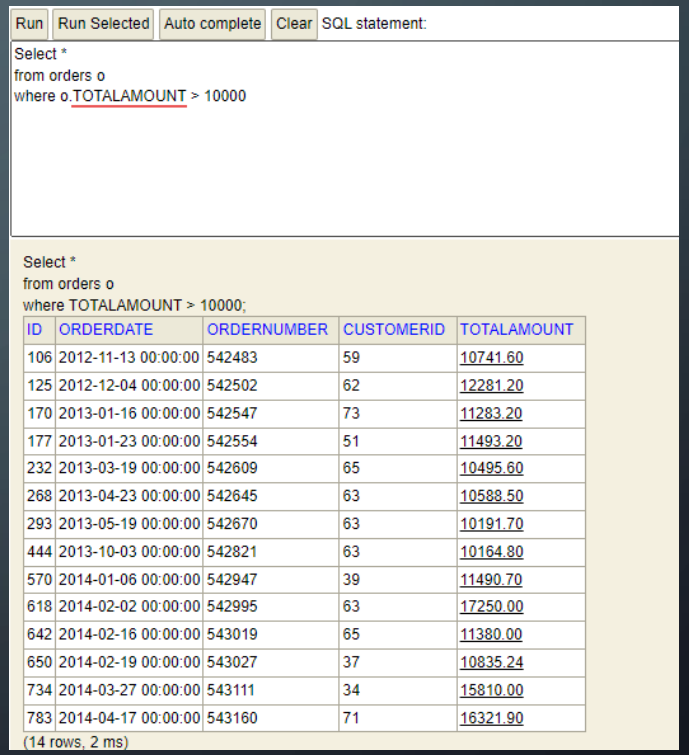
```
Select *
from Suppliers
order by Companyname ASC
limit 10;
```

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'érables	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



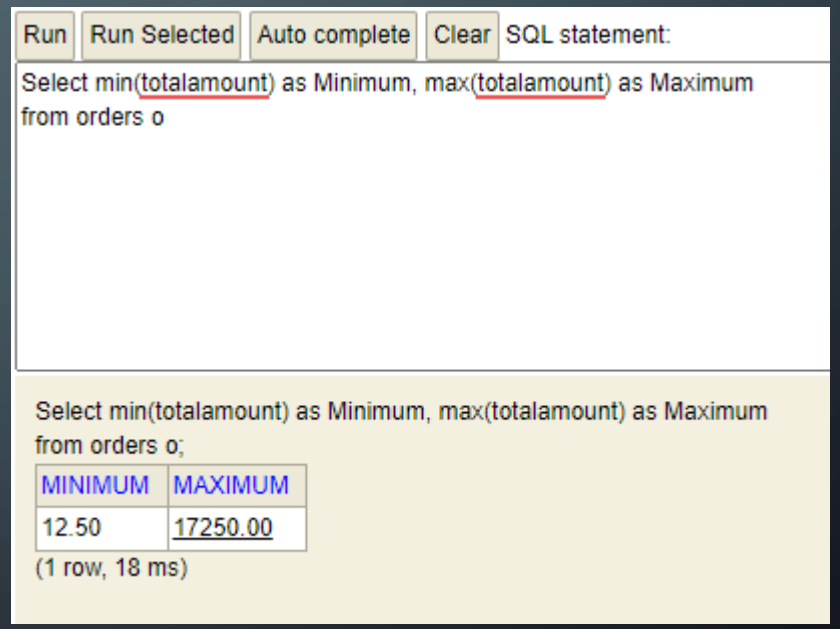
DATABASES – SELECT STATEMENT

- These are some aggregate functions:

Aggregate function	Explanantion
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

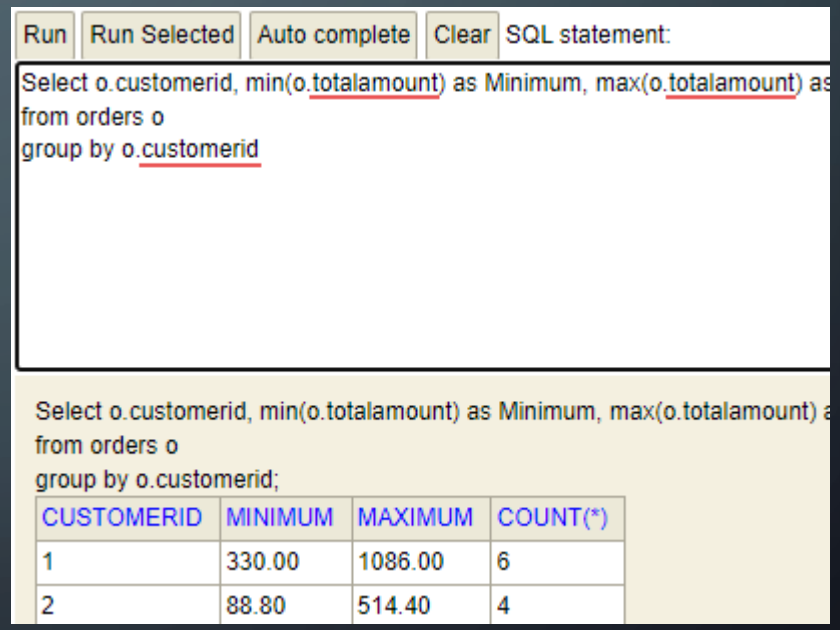
DATABASES – SELECT STATEMENT

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



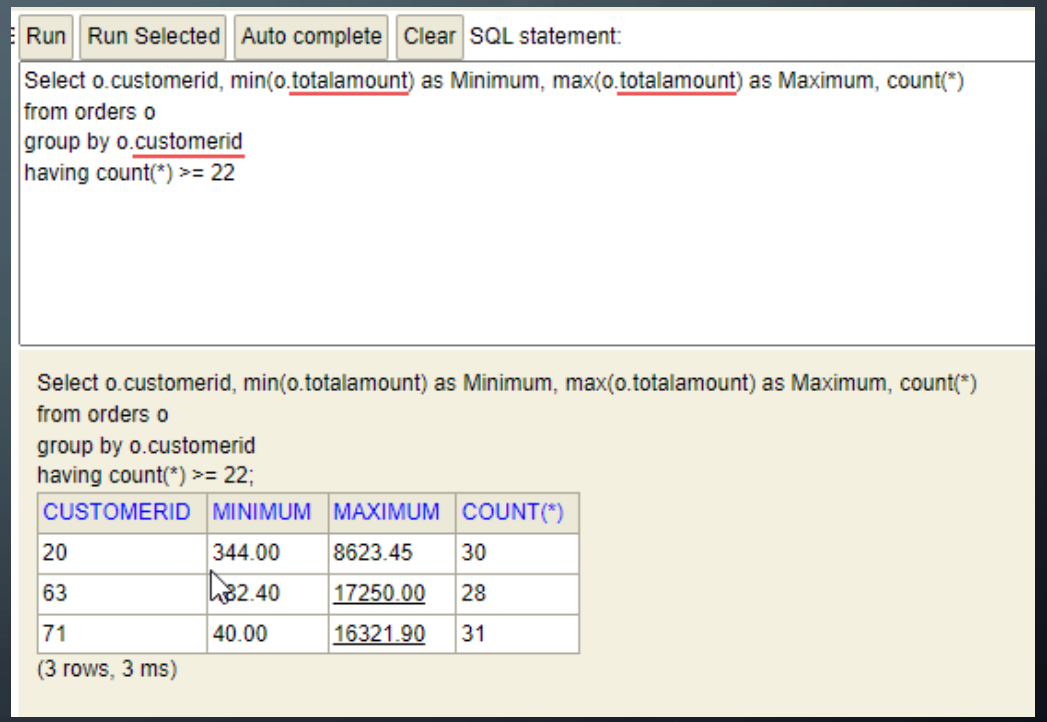
DATABASES – SELECT STATEMENT

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

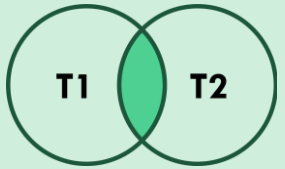
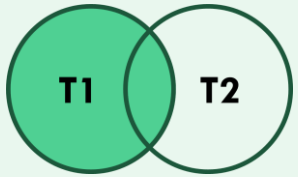
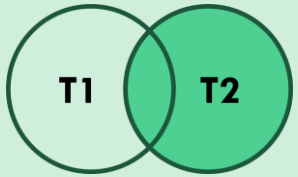
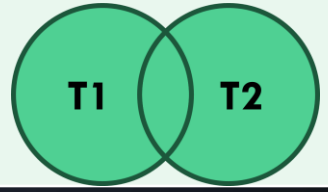


DATABASES – SELECT STATEMENT

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



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)

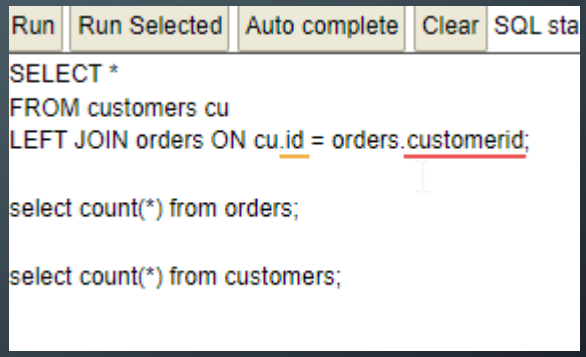
Run | Run Selected | Auto complete | Clear | SQL statement:

```
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)



DATABASES – SELECT STATEMENT

- All the orders, even those without a customer

  (There better is none without a customer!)

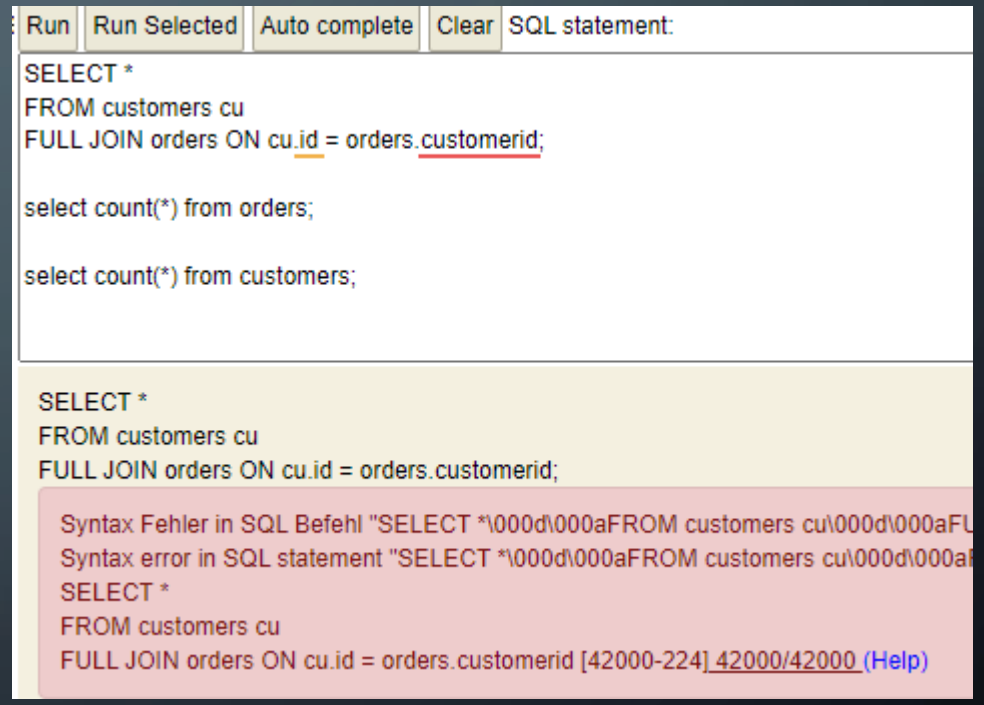
Run | Run Selected | Auto complete | Clear | SQL statement:

```
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

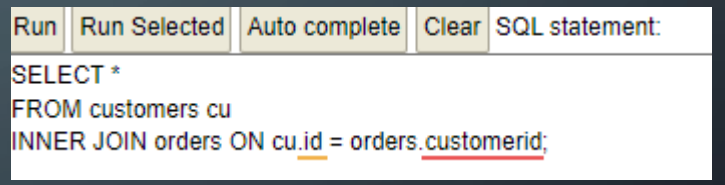
  You will have to switch to a different dialect.



DATABASES – SELECT STATEMENT

- List all customers, which have placed orders.

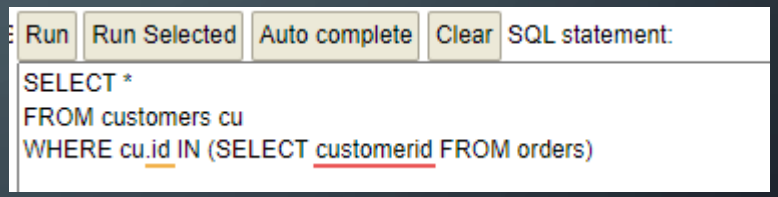
  Since we use the *, not only the customer's attributes, but also the order's attributes are shown



DATABASES – SELECT STATEMENT

- List all customers, which have placed orders.

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



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

Run | Run Selected | Auto complete | Clear | SQL statement:

```
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.
- Now run "select * from suppliers where fax = null"

  Explain the resultset.

- How could that be done better?

DATABASES – TRANSACTIONS

- Transactions allow to 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 implicitely until they are explicitely finished.
- Transactions result in locks in the DB and therefore impact parallel access

DATABASES – TRANSACTIONS

- Even though transactions start implicitely 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"));
   }
}
```

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	
void persist(<<filename>>)	Save DB under filename to disk	
void load(<<filename>>)	Load Db from filename on disk	
MemKV findEntriesByKey(searchPatter)	Get a new MemKV with only the records, which keys match the searchpattern. Searchpatter-Wildcards:	
	*	Zero or more characters
	?	Single character
	Any other char	Any character

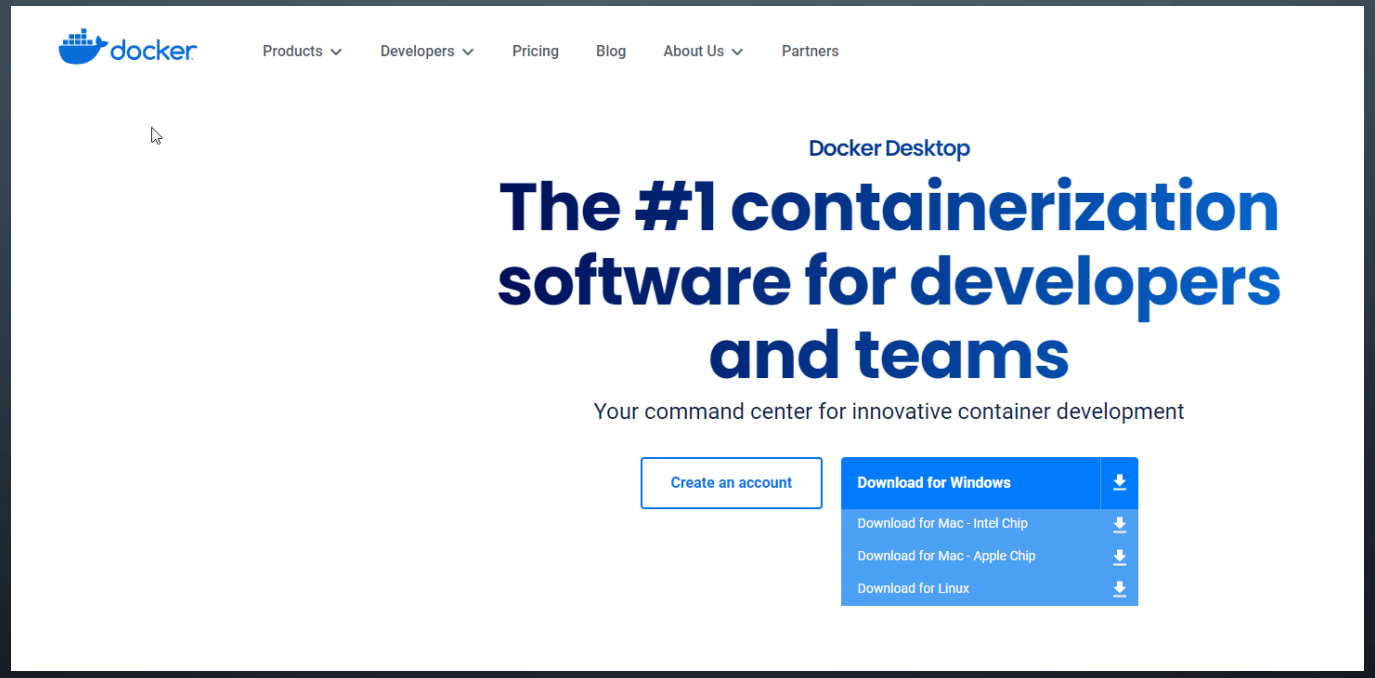
DISCLAIMER

- 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 for all GIT remote repository demonstration purposes

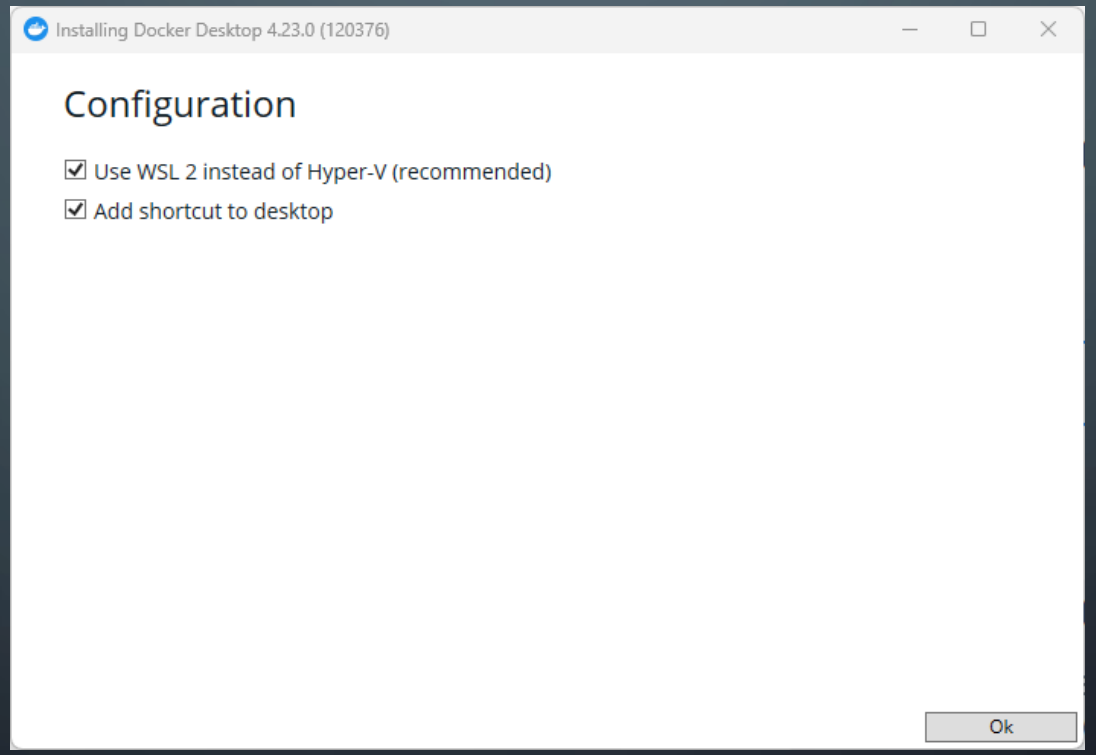
APPENDIX A: DOCKER DESKTOP INSTALLATION

DOWNLOAD DOCKER DESKTOP

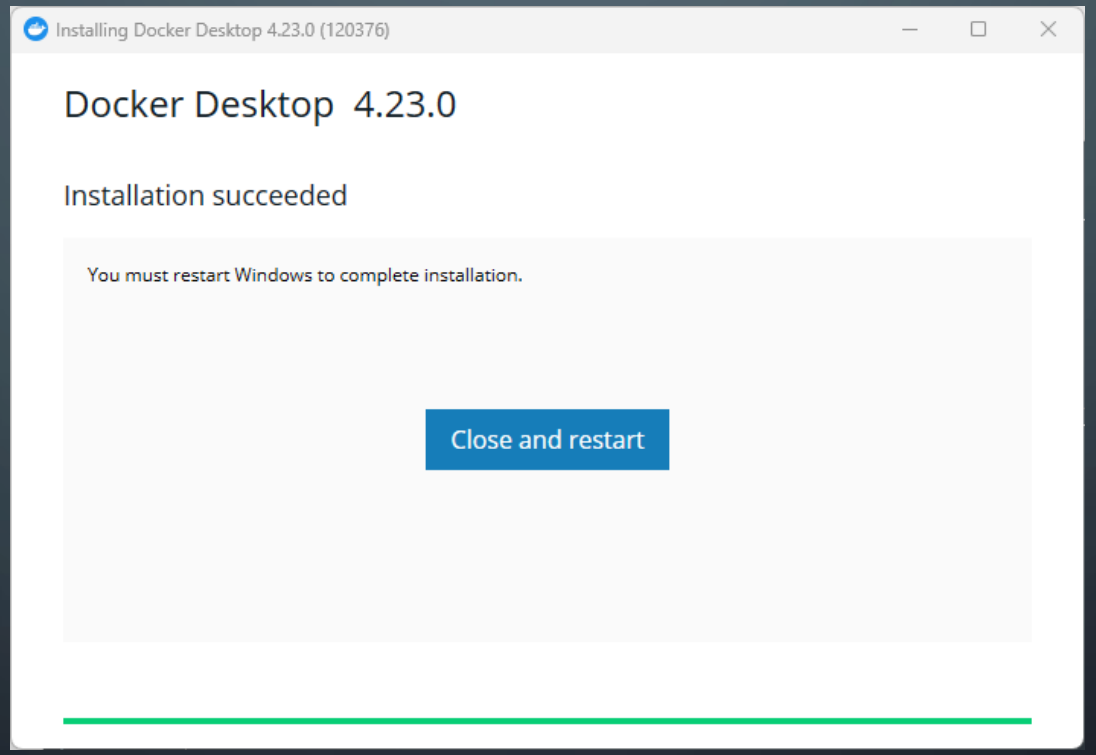
- Goto: https://www.docker.com/products/docker-desktop/



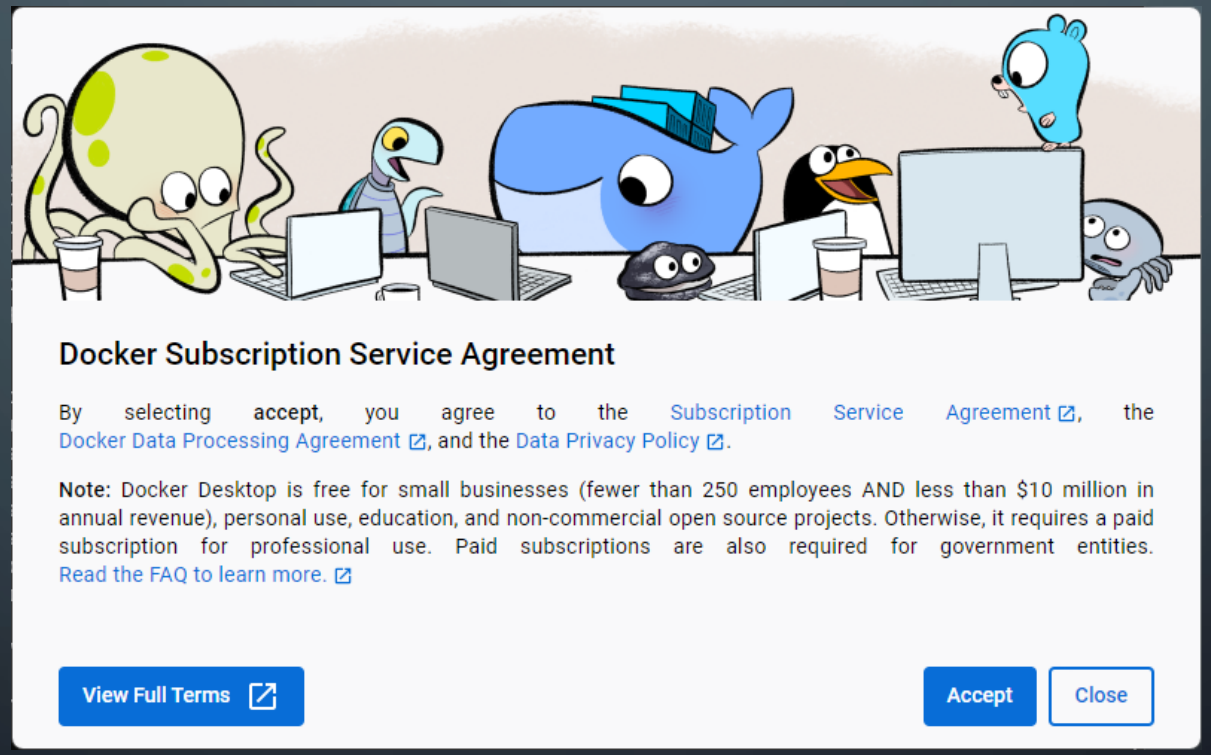
START INSTALLATION



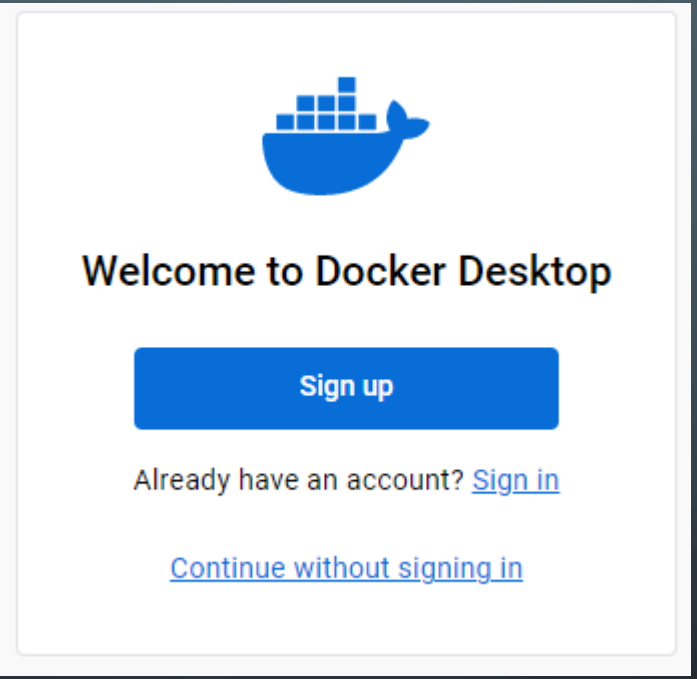
SUCCESSFULLY FINISHED – REQUIRES RESTART



AFTER RESTART



SIGN IN / UP


Welcome to Docker Desktop
Sign up
Already have an account? Sign in
Continue without signing in

ON THE COMMAND LINE

- Docker Desktop might require a newer version of WSL:

Command line: wsl --update

FINALLY YOU SHOULD SEE THIS