Practical Java

Kristof Werling

April 2022

Organizational Info

Each day of the lecture

- 2 Parts
  - Part 1:
    - Learn about new Java concepts and functionality
    - Easy and short exercises to enhance the understanding of the material
  - Part 2:
    - Use the new concepts and functionality of Part 1 to enhance and extend the Converter project

Part 1 - more details

- There will be exercises
- Each execise comes with a solution
- Each exercise will be discussed with the whole group and problems / issues will be addressed
- The solution provided also will be discussed

Depth of the information provided

- For the most part the information provided is sufficient to work out the solution to the execises
- For most of the concepts and functionality shown there is a vast body of knowledge we cannot explore in any kind of practical manner.

Project for this lecture

- Converter: Markdown to Latex and later to Html
- Simple solution.
- Each lecture works on one aspect of the solution (like: GUI, DB, …)

Markdown Tags

Headings	Text Formatting	Rendered Output
# Heading level 1	**This is bold text**	Link to [Google](https://www.google.com/)
## Heading level 2	__This is bold text__	- Unordered List Item 1 - Unordered List Item 2 - Unordered List Item 3
### Heading level 3	*This text is italicized*	
#### Heading level 4	~~Strikethrough text~~	
##### Heading level 5	***Bold and italics text***	1. Ordered List Item 1 1. Ordered List Item 2
###### Heading level 6	**Bold and *nesting italics* text**	

Source: https://www.markdownguide.org/basic-syntax
Or
https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

All the gory details (specs): https://github.github.com/gfm/

Latex

Boiler Plate for a Latex Document	\documentclass[12pt, a4paper] {article} \begin{document} Here goes the document. \end{document}

Source: https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes

Latex

Bold	This is \textbf{bold text}
Italic	This is \textit{text in italic}
Strikethrough	
Unordered List	\begin{itemize} \item Item 1 \item ... \end{itemize}
Ordered List	\begin{enumerate} \item Item 1 \item ... \end{enumerate}
Link	Use the hyperref package

Source: https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes

Latex

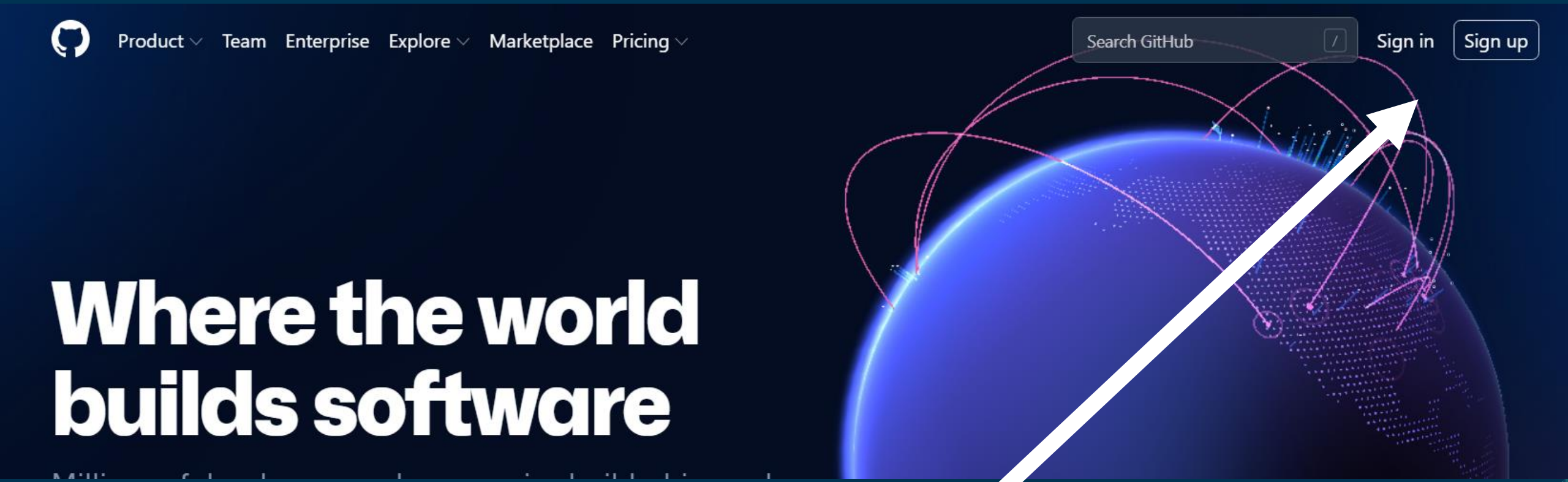
Heading 1	**\section{section}**
Heading 2	**\subsection{subsection}**
Heading 3	**\subsubsection{subsubsection}**
Heading 4	**\paragraph{paragraph}**
Heading 5	**\subparagraph{subparagraph}**

Source: https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes

What we need to get started

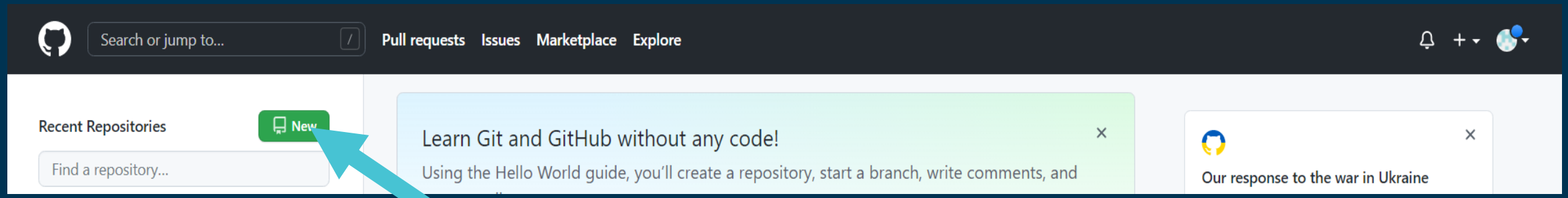
- IntelliJ Community Edition (Download here)
- MiKTex (or any other Tex that can process Latex) (Download here)
- OpenJDK Java 18 (Download here)
- Markdown Viewer (For example: Windows Markdown Viewer)
- Git for Windows (Download here)

Github.Com Account



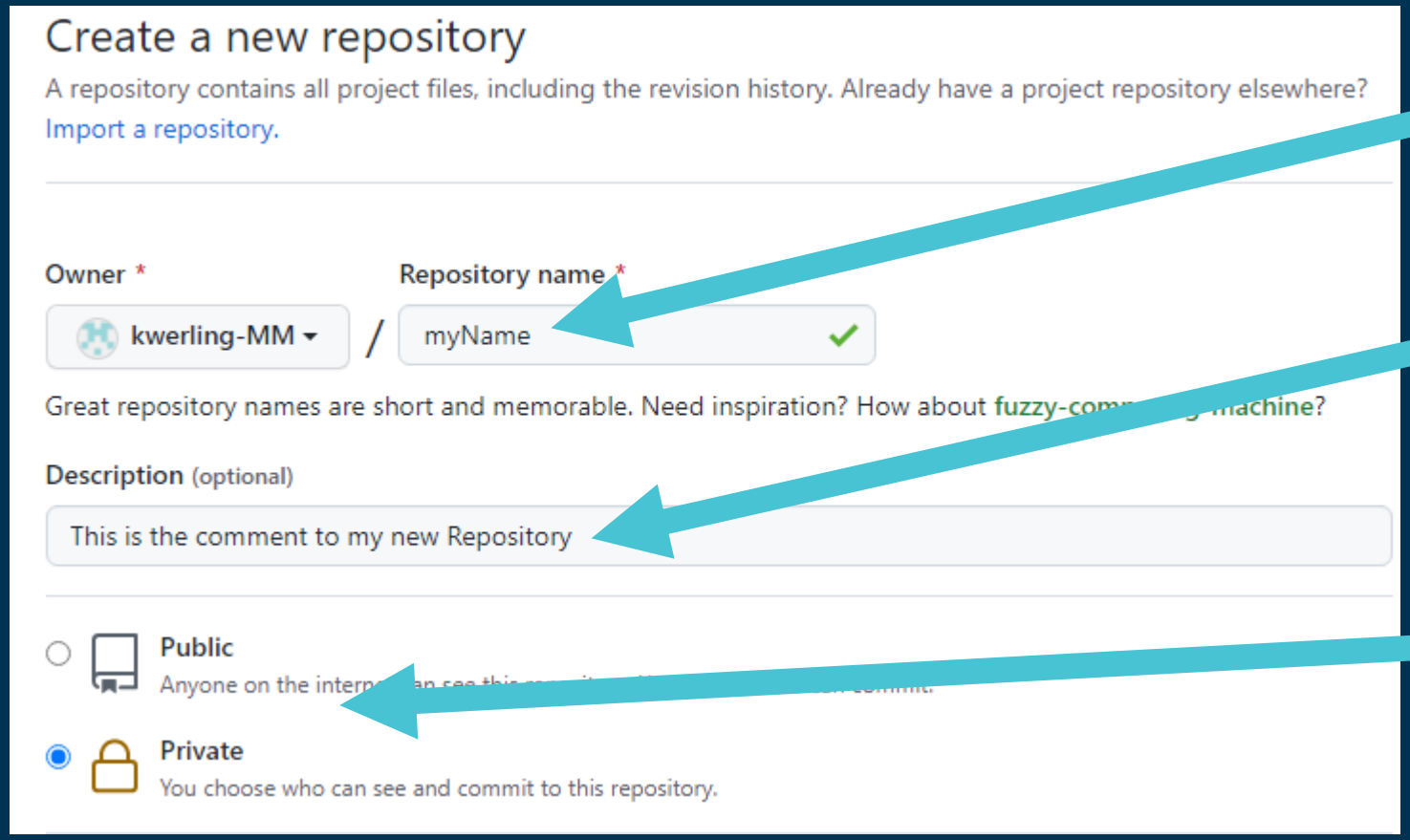
Sign in or Sign up

Create new Repository



New Repository

Create new Repository 1 of 2

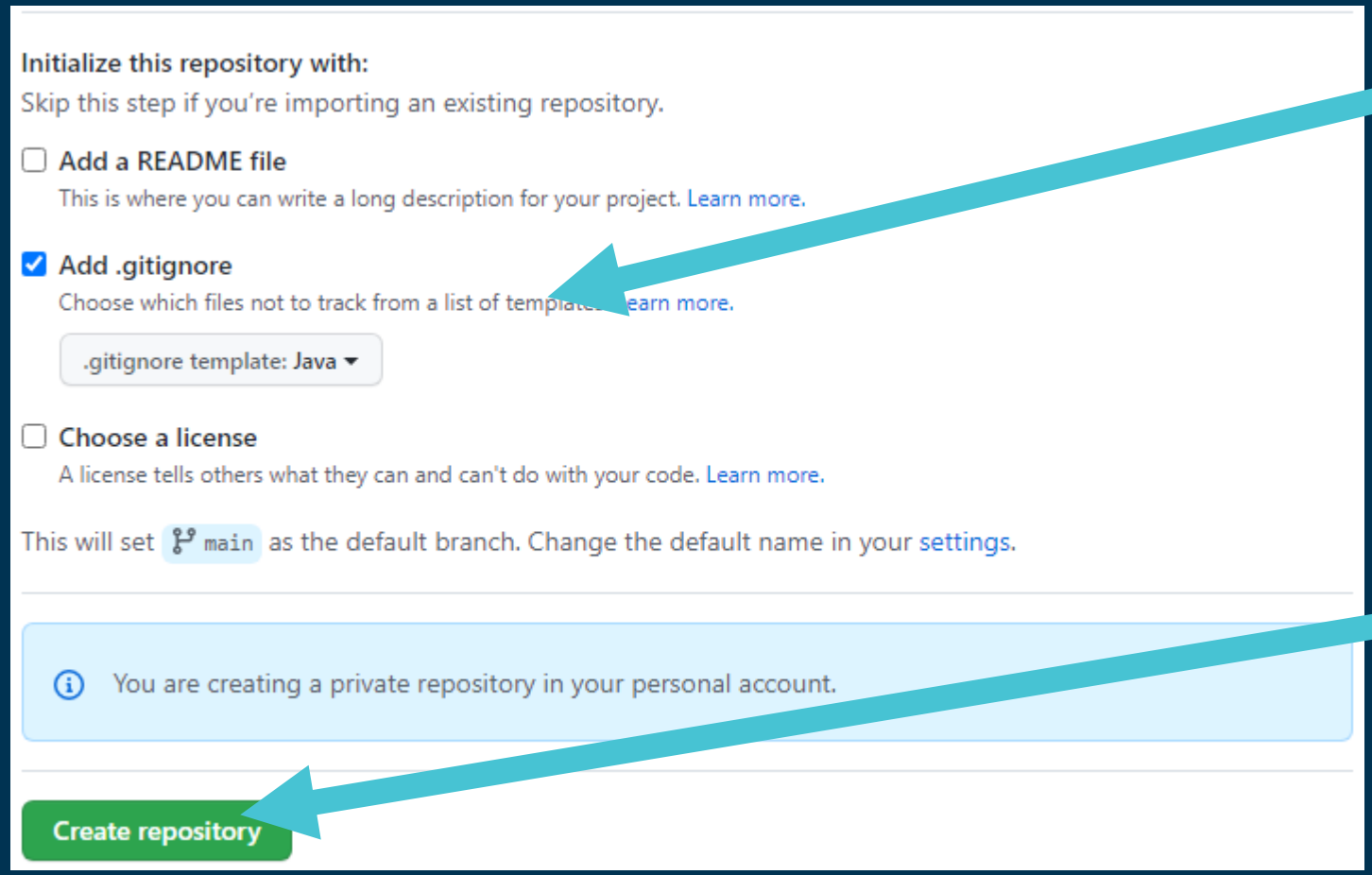


Repository name

Comment, if wished

Private or Public access

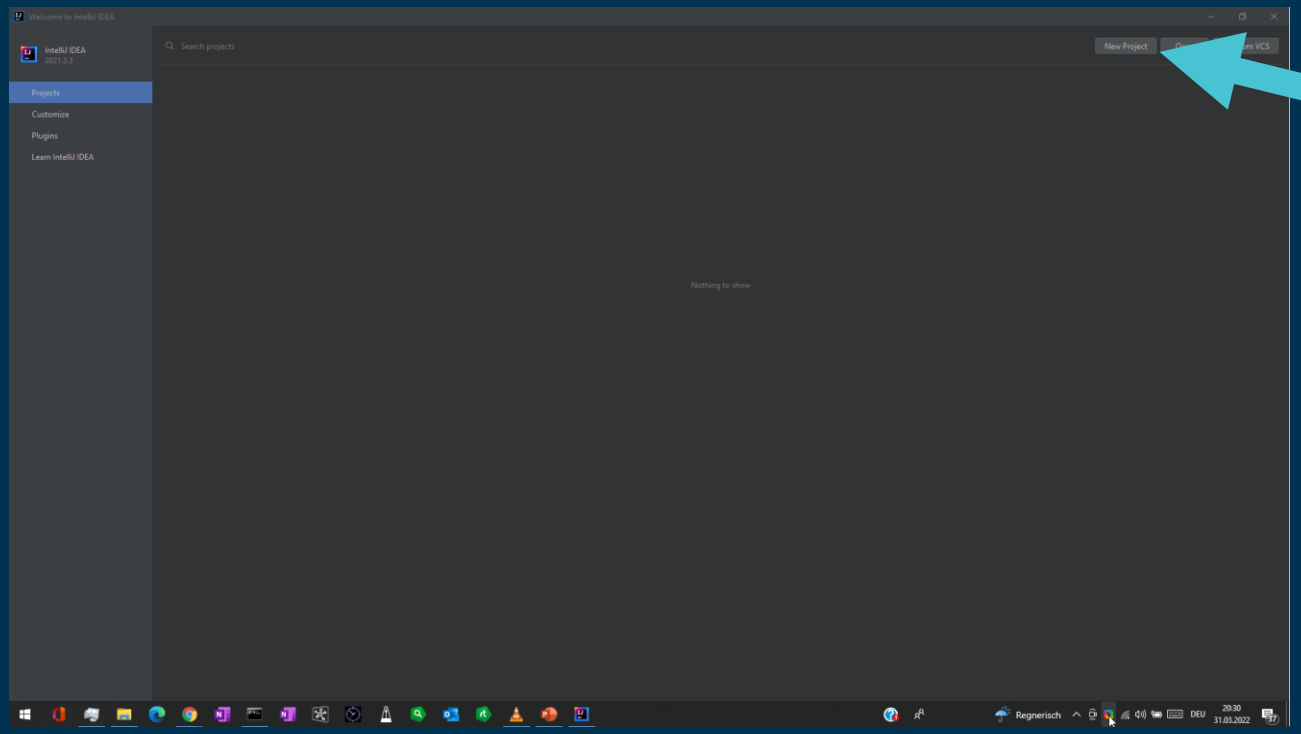
Create new Repository 2 of 2



Add .gitignore for JAVA

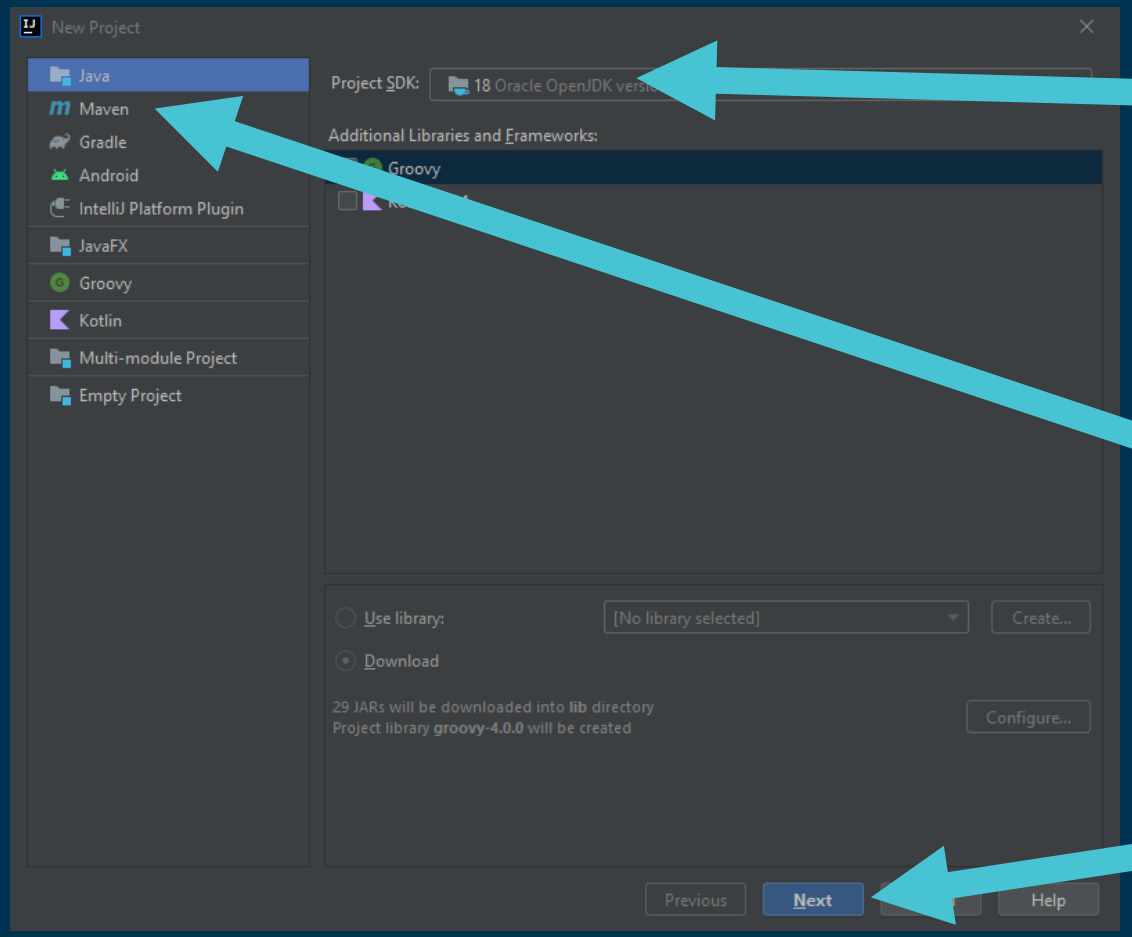
Create it

Creating a new project 1 of 5



New Project

Creating a new project 2 of 5

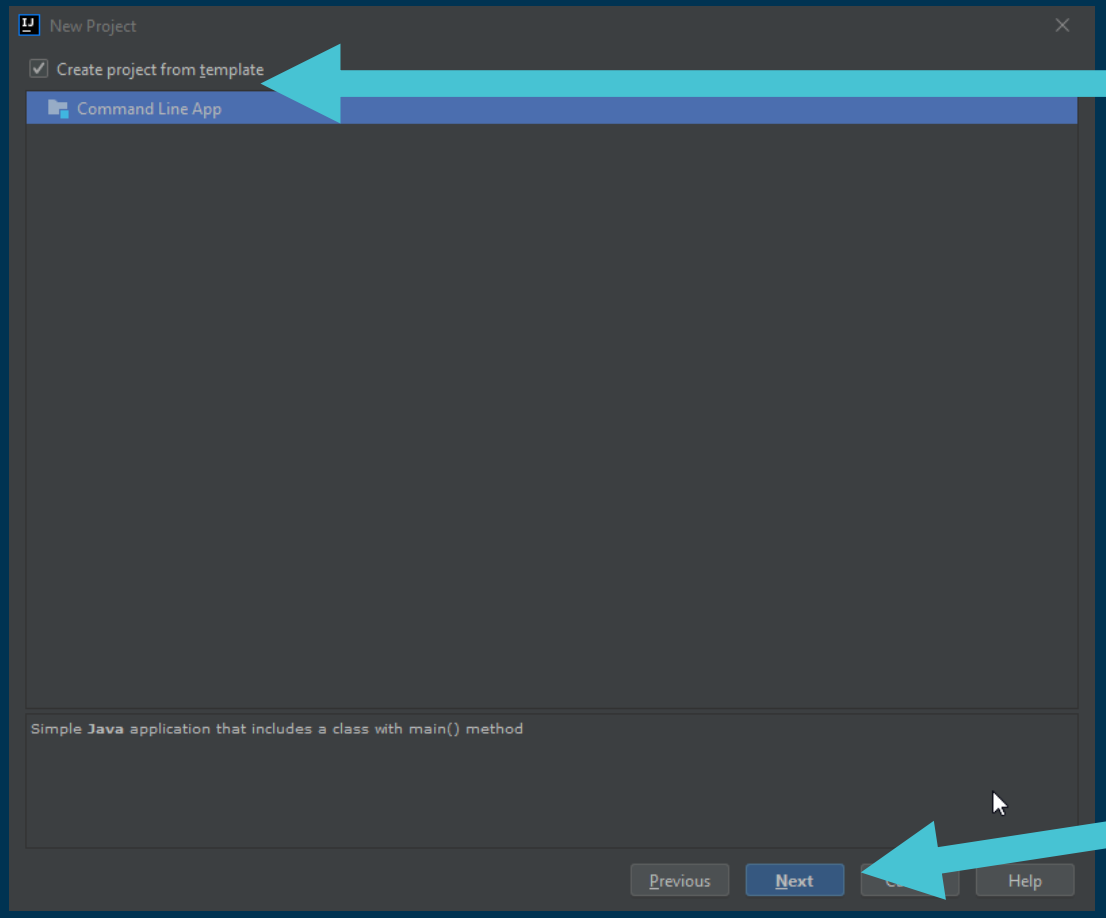


Make sure your JDK is listed here

Java

Press Next

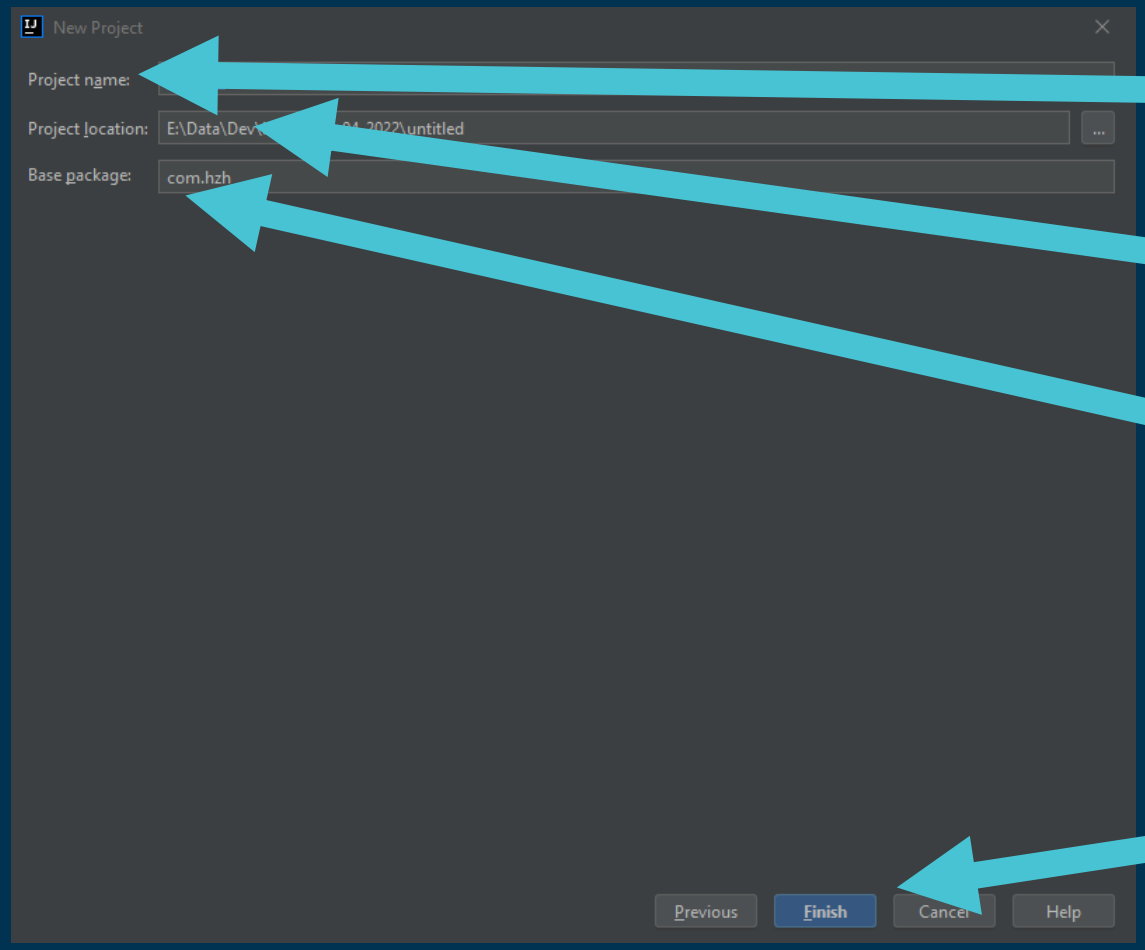
Creating a new project 3 of 5



Choose the Commend Line Template to get started

Press Next

Creating a new project 4 of 5



Give it a name

Local directory

Name space

Press Next

Creating a new project 5 of 5

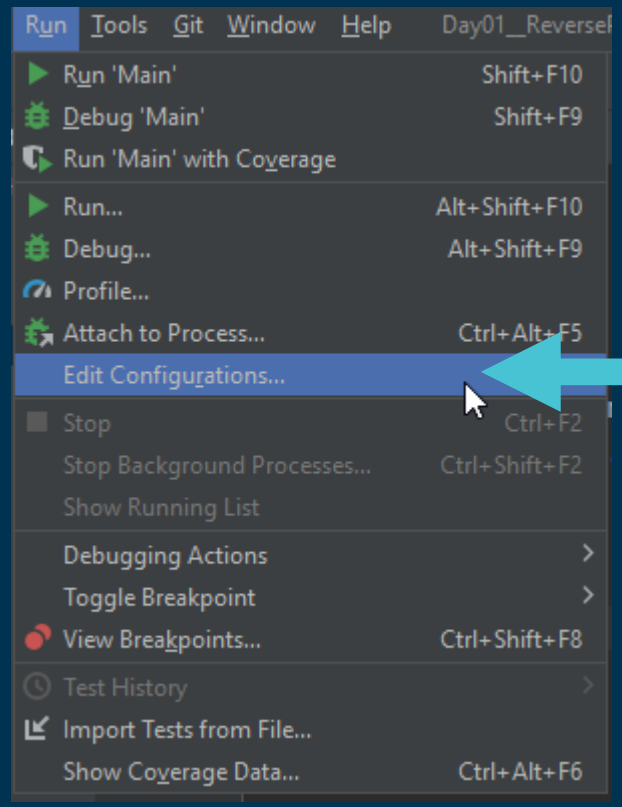
```
package com.hhz;

public class Main {

    public static void main(String[] args) {
    // write your code here
    }
}
```

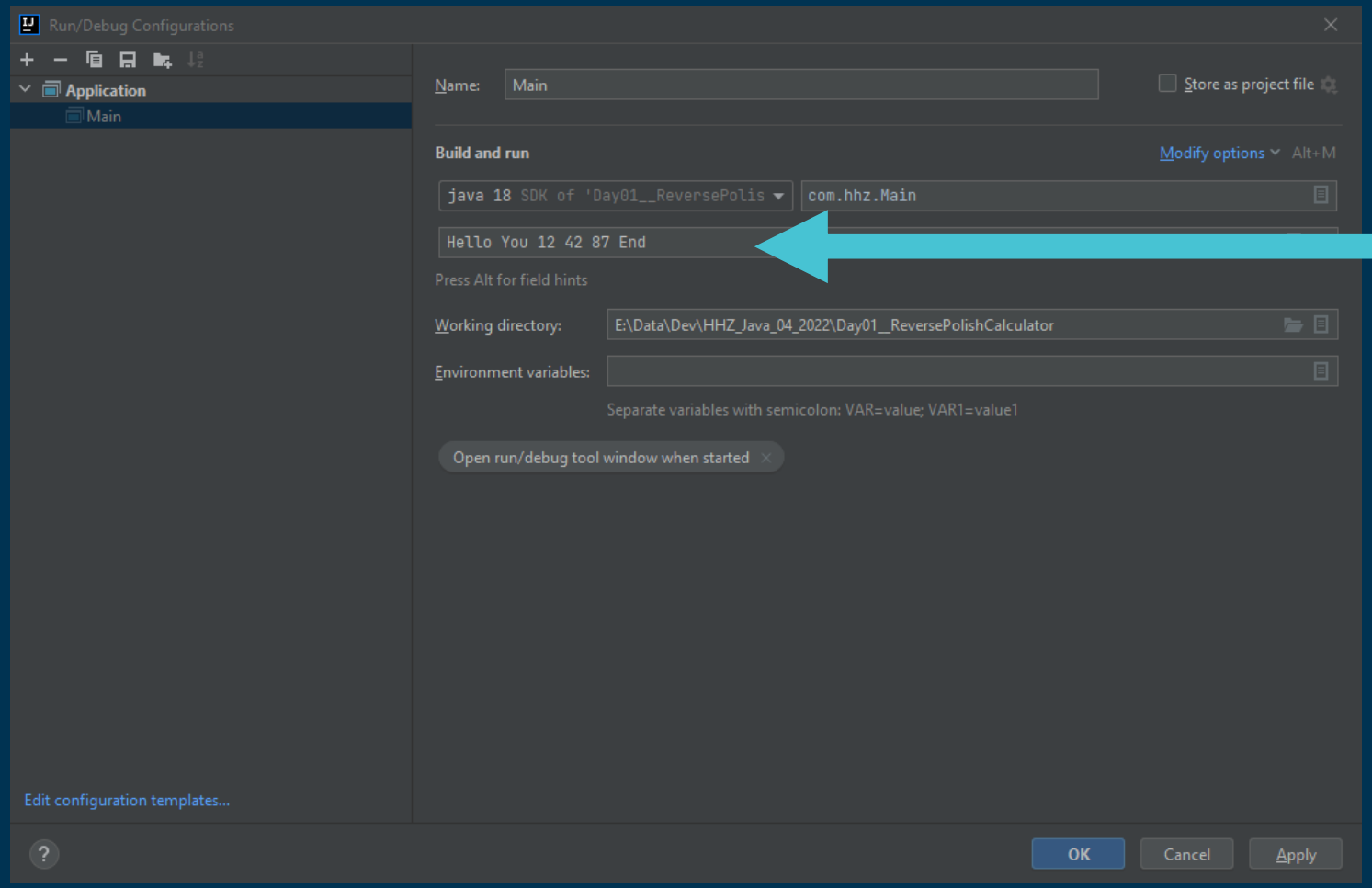
Resulting project

Run with Command Line Params 1 of 2



Adjust the configuration

Run with Command Line Params 1 of 2



Here go the command Line params

Table of Content

Command line parameters

- The main function takes the command line parameters in an String array
- Each parameter is passed on as a String (String array after all)
- In case of no command line parameters the String array is empty

Exercise 01

- Write a program, which prints out the command line parameters in reverse order

Exercise 01 - Solution

```java
package com.hhz;

public class Main {

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

Some methods of the String class

(some) String class method	Functionality
String toLowerCase()	It returns a string in lowercase.
String toUpperCase()	It returns a string in uppercase.
String trim()	It removes beginning and ending spaces of this string.
int indexOf(String substring)	It returns the specified substring index.
String[] split(String regex)	It returns a split string matching regex.
boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
int length()	It returns string length. Compare to Array.length !!
String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.

Exercise 02 – Playing with String comparison

- Take the code on this slide
- Run it
- Explain the results

Main.java

Exercise 03 – Playing with String concatination

- Take the code on this slide
- Run it
- Explain the results

Main.java

Integer class – parsing of text

- int Integer.parseInt(String)

  tries to convert the String into an integer value. Throws an exception if that not possible.

  Ex: Integer. parseInt(„411“) → 411
  Integer. parseInt(„Axx“) → Throws exception

Try - catch - finally

- In order to control code, which might throw exceptions it is enclosed in a try-catch (-finally) construct:

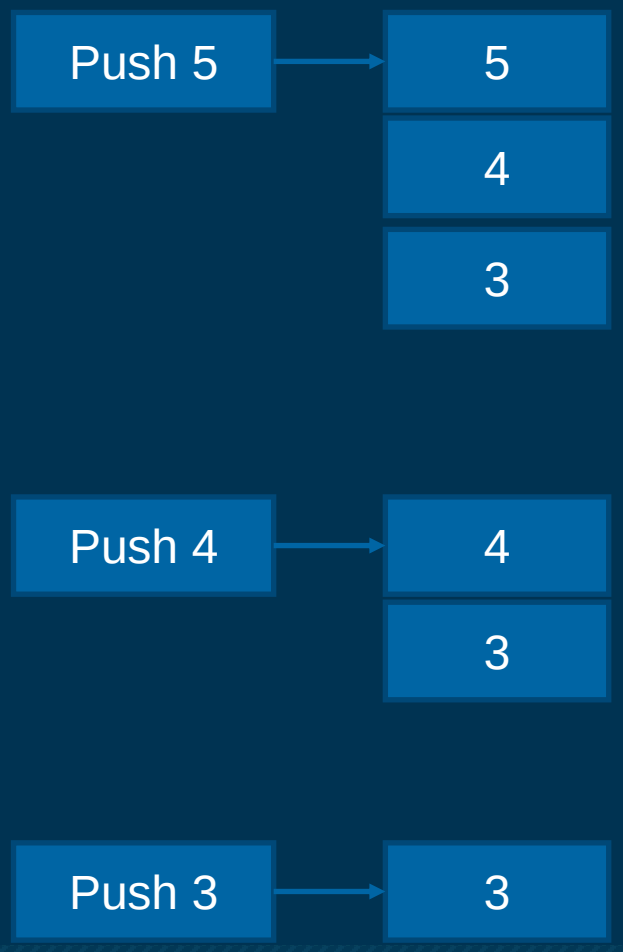
```
try {
        // Code, which might throw exeptions
} catch( Exception ex ) {
        // Code to run if an exeption happened
} finally {
        // Code, which runs wether an exeption was thrown
}
```

Exercise 04

- Write a program, which prints out the command line parameters in reverse order.
- Add 10 to each integer value in the list before printing it out.

Class Stack

- Growths (aka Push operation) upwards
- Shrinks (aka Pop operation) downards
- There are only the push and pop operations for accessing the stack.

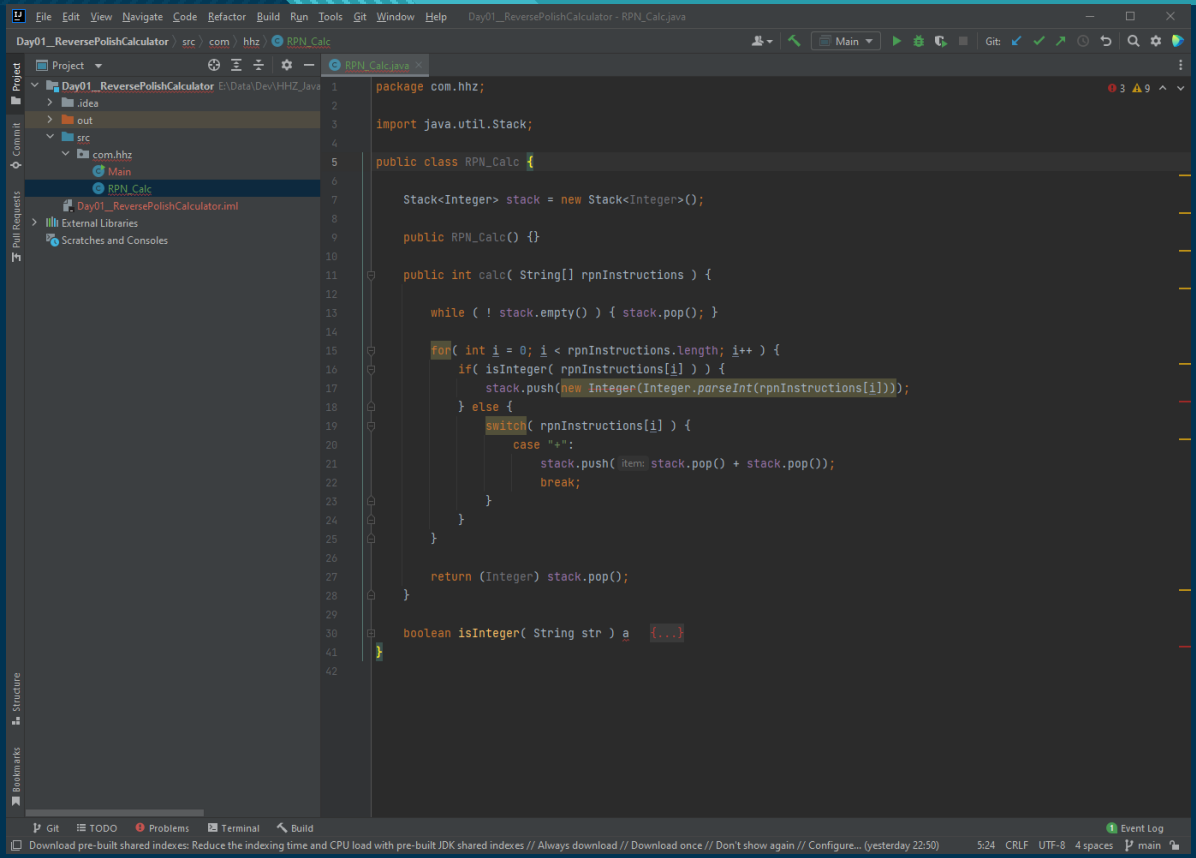


Unit testing

- Small test of parts of code
- Always test one thing and one thing only
- Expected to run fast
- YES, I know of projects where the code for testing exceeded the code under test.

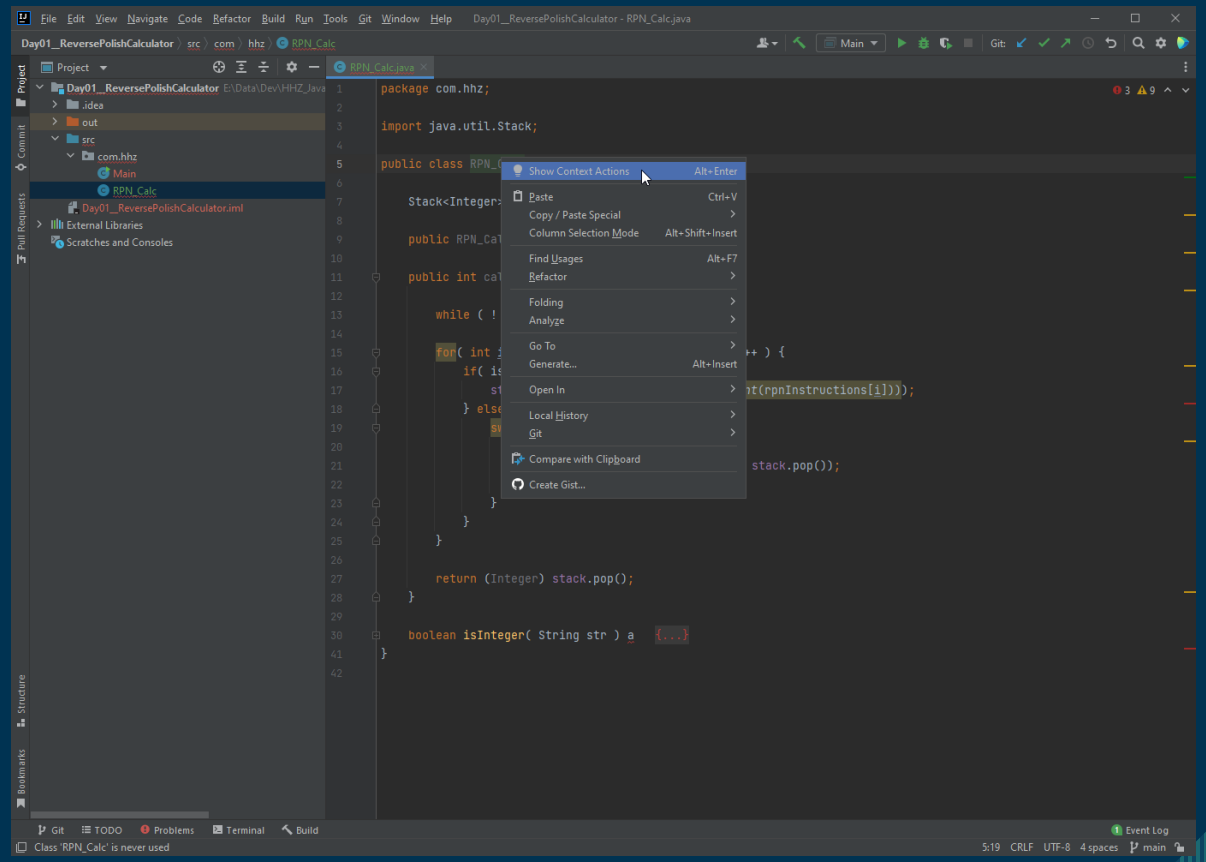
Unit testing

- In order to create Unit tests move the cursor over the class name and press the right mouse button → context menu



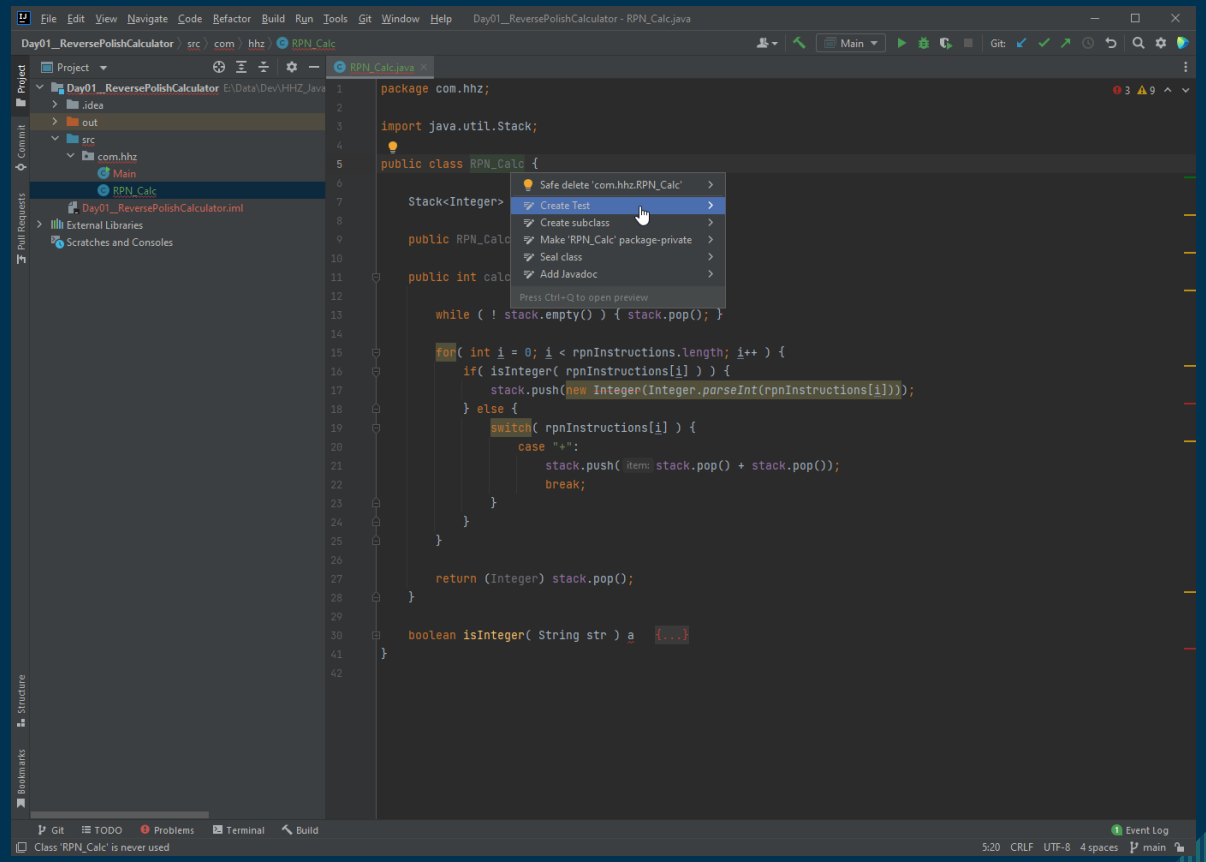
Unit testing

- Choose „Show Context Action“



Unit testing

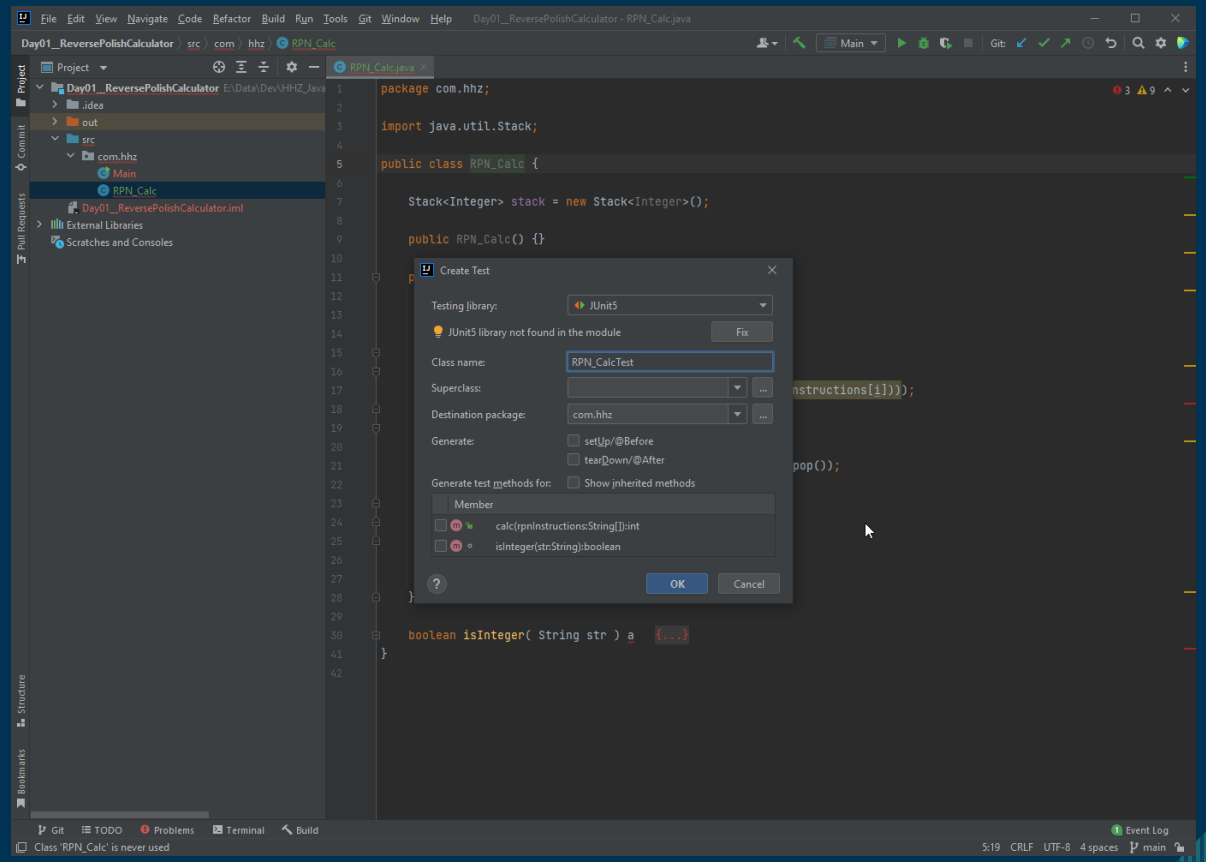
- Choose „Create Test“



Unit testing

- When done for the first time the Junit jar file needs to be added to the project.

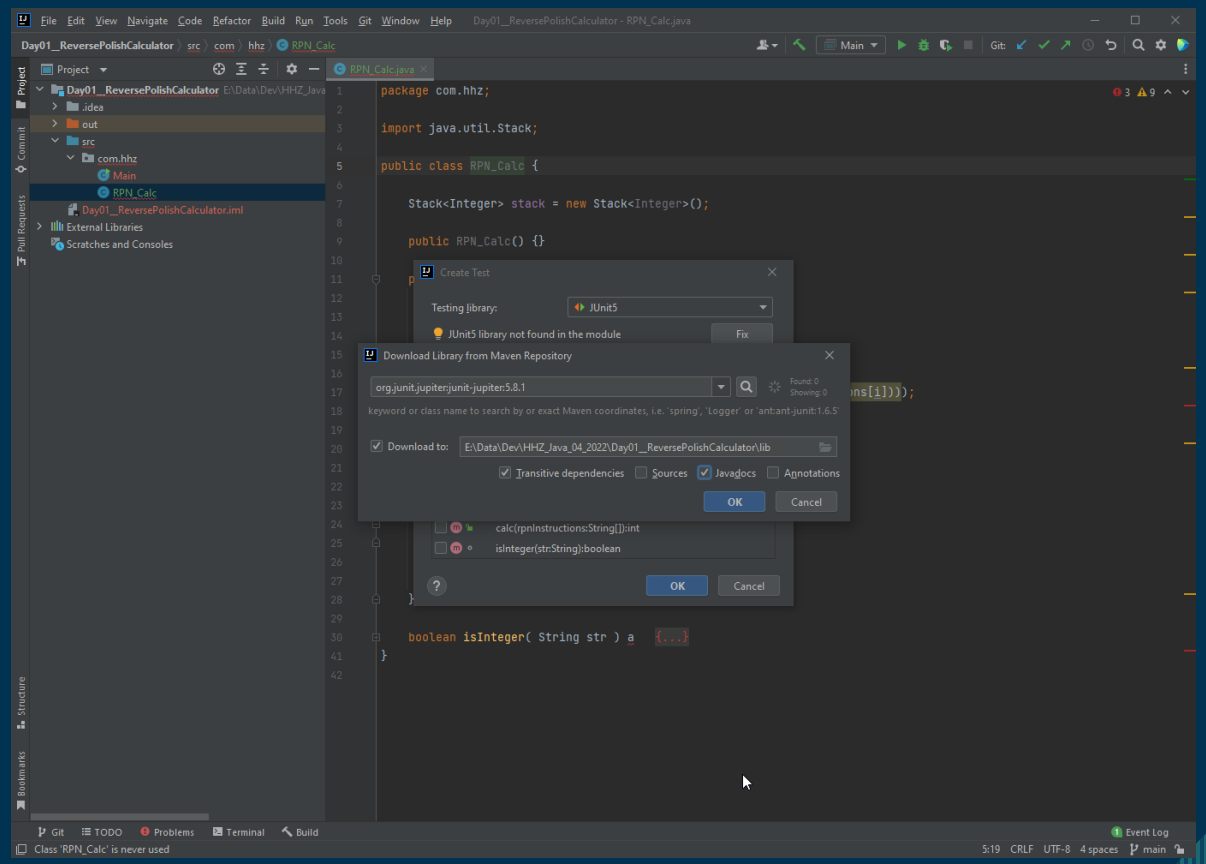
  Press „Fix“



Unit testing

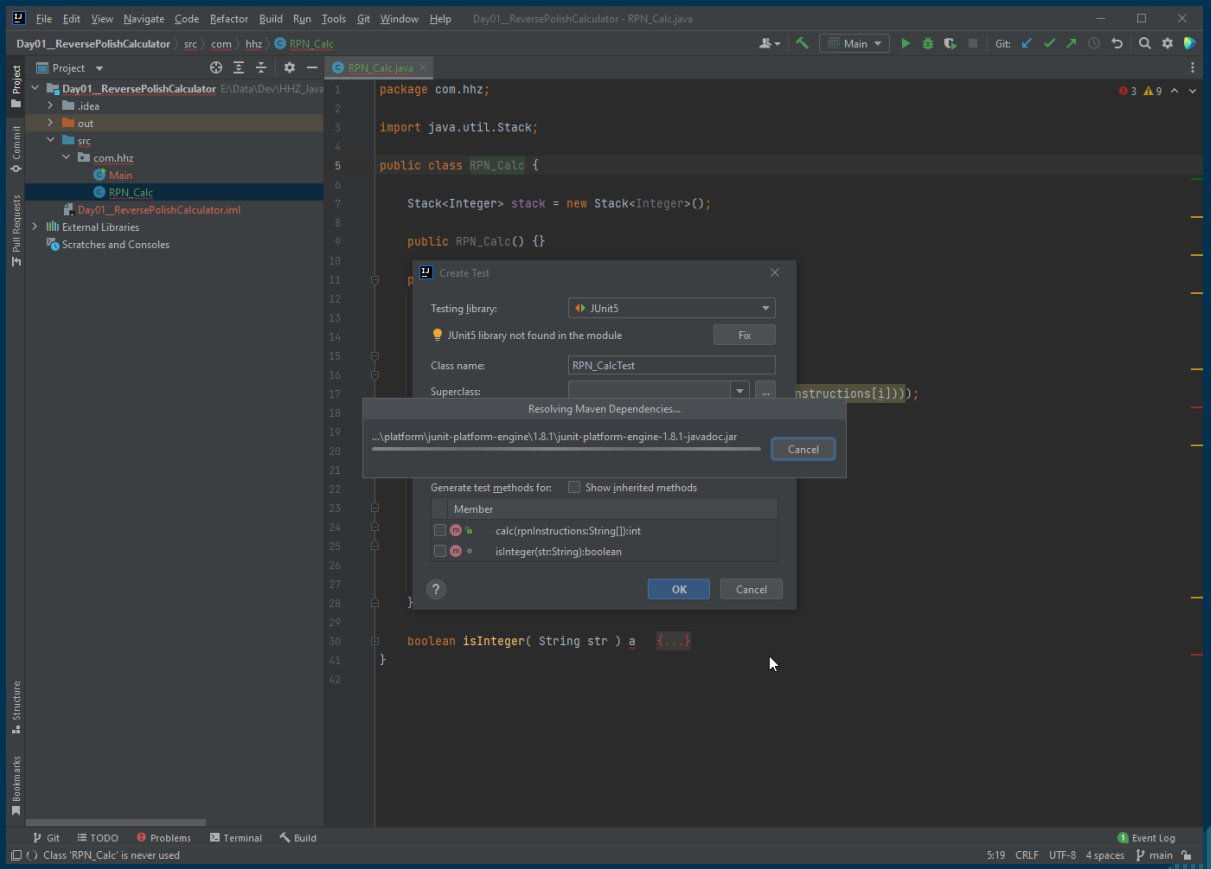
- Add the Junit jar file

  Download Javadoc as well.



Unit testing

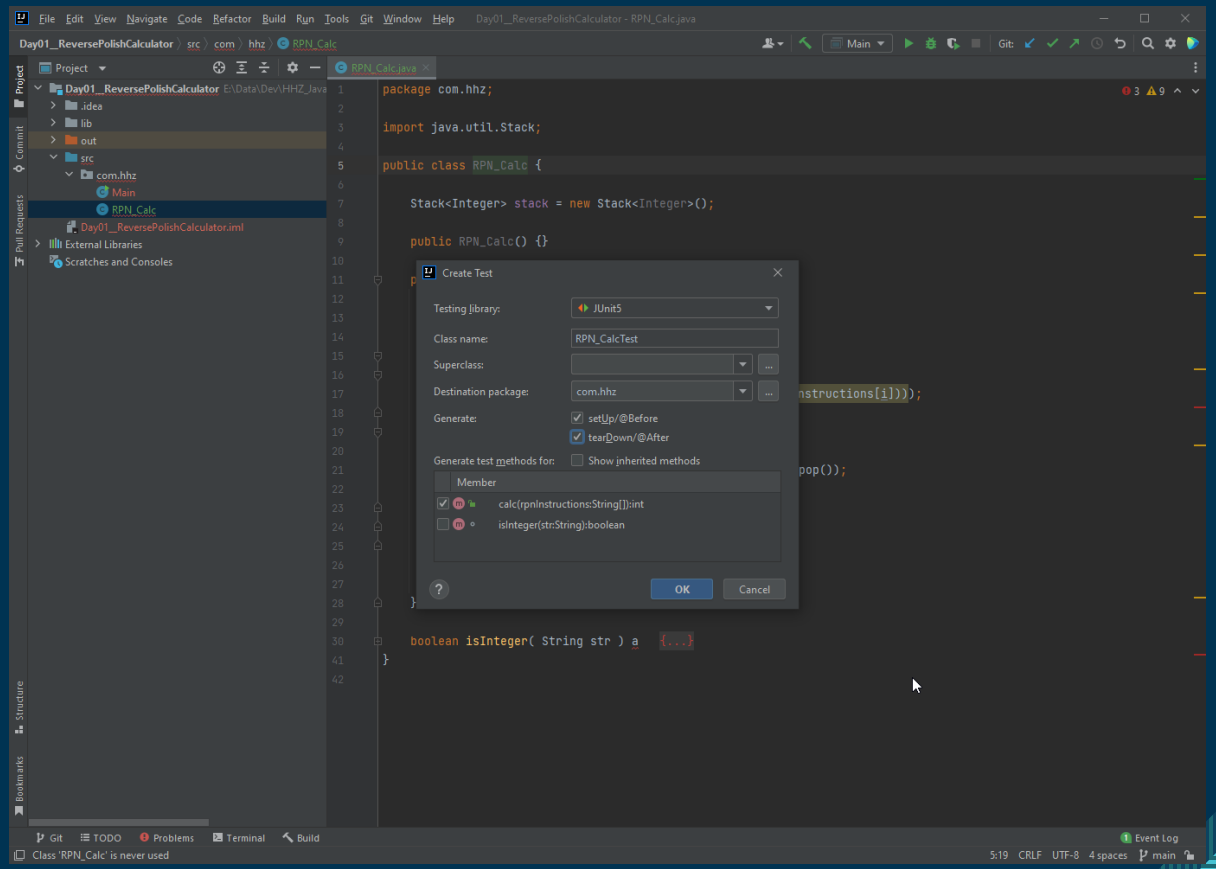
- IntelliJ downloads the required files



Unit testing

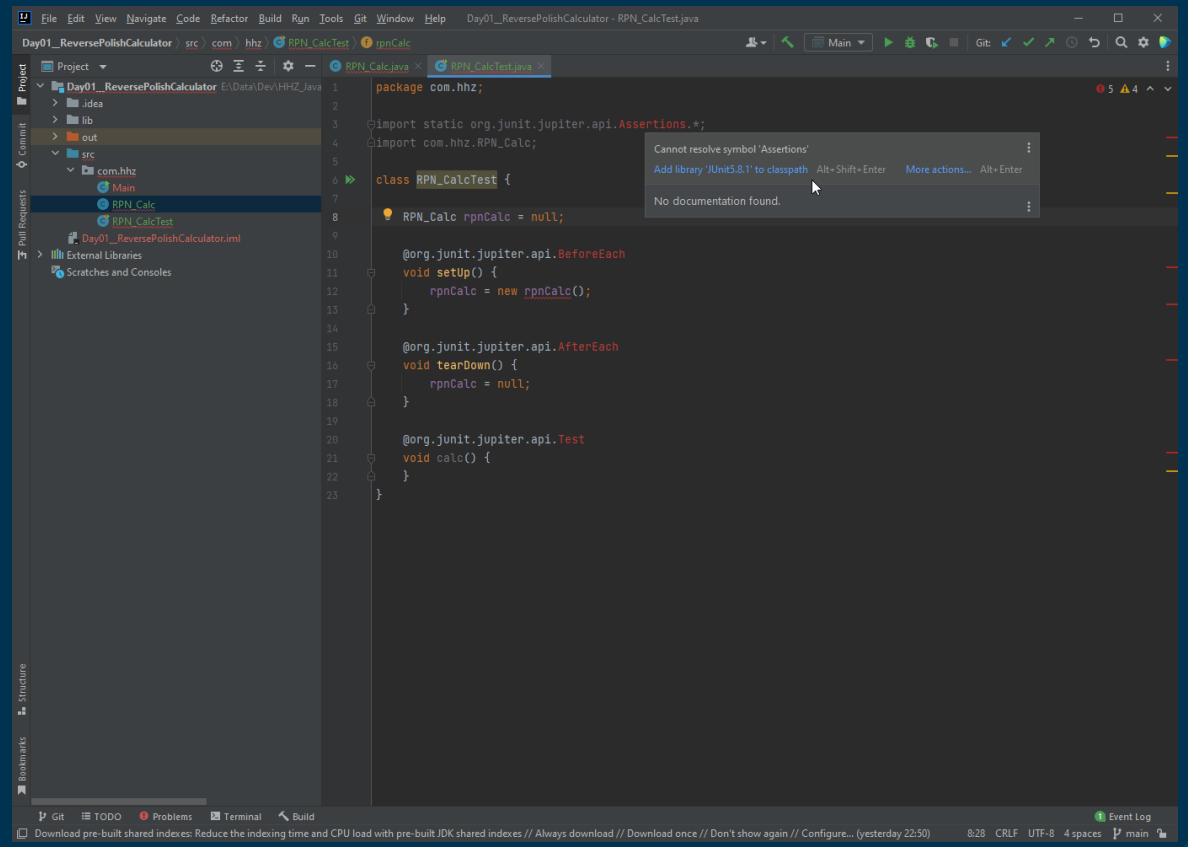
- Create the Junit test skeleton for the calc method

  Create the Setup & TearDown methods



Unit testing

- Need to add the Junit library to the classpath.



Exercise

- Create a class Calc
  - With one method int add(int a, int b), which returns the sum of a and b
  - Create multiple tests for this method

UPN

- Add 10 and 20:

10 20 +

- 3*(4+2)

4 2 + 3 *

Exercise 05

- Create a class RPN, which has this method:
  - int calc(String [ ] rpnInstructions);
- It takes a string array as input, which contains the operands and the operators in correct order for the calculation
- It returns the result of the calculation
- Only integer values are used in the calculation
- Add Unit testing for verification of the correctness of the class

Exercise Converter

- Create a Converter class, which is able to take in a text string in Markdown text and convert it into Latex
- For starters we need to translate the Headings first
- The first character in a line is a ‚#‘,maybe followed by more of them
- The first non-‘#‘-character to the end of the line is the Header text
- All other text (Markdown tags or not) is simply copied to the Latex file.
- Create a Main-method, which takes the Markdown-Filename from the command line.
- The Latex file has the same file name as the Markdown file, just ending in ‚.latex‘
- Feel free to convert other Makrdown tags as well

Exercise Converter - Hints

- ArrayList<String>
- String class:
  - charAt
  - indexOf
  - substring
  - trim

SWING - User Interface

A simpe Swing application

- JFrame is the class providing a window for an application.
- Inside of the window there will be components for user information and interaction

Excercise 1

- Build an applicaton, which shows an empty window
- Play with it. Is there any difference to other windows from other applications?

Hints

- JFrame.size(int, int) allows to set the dimensions of the window.

- In order to end the application when the window gets closed one needs to add an event listener:

```
window.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
        }
    });
```

Components

- There are many different components available.
- We will start out with JButton and JLable

Excercise 2

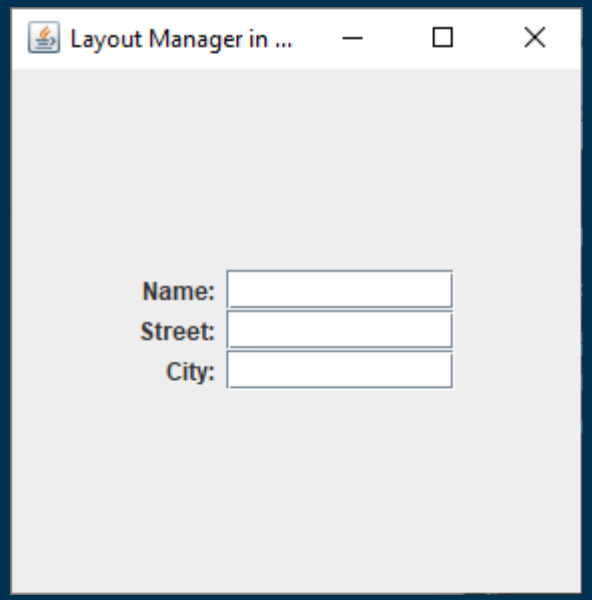
- Add a button to the window
- Anything the matter?
- Try adding a second button
- What happens then?

Layout manager

- In order to achieve control over the positioning of the different components in a window Swing offers different Layout managers.
- Layout managers can be nested.
- The different Layout manages are shown here.

Excercise 3

- Use the GridbagLayout Manager to construct the following screen with JLables and JTextFields(width 10)



Working with events

- In order to react to the push of a button it (the button) needs to listen to such an event.

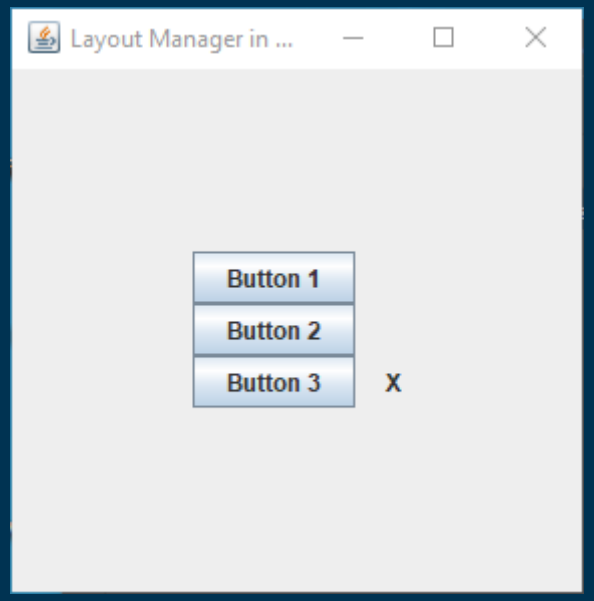
```
button.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            System.out.println( ((JButton)e.getSource()).getText() );
        }
    });
```

See Different ways to implement a listener

Excercise 4

- Create a little application, which shows an X next to the button, which was clicked last:

  Example:



Picking a file name

- JFileChooser brings up a dialog box for choosing a file name

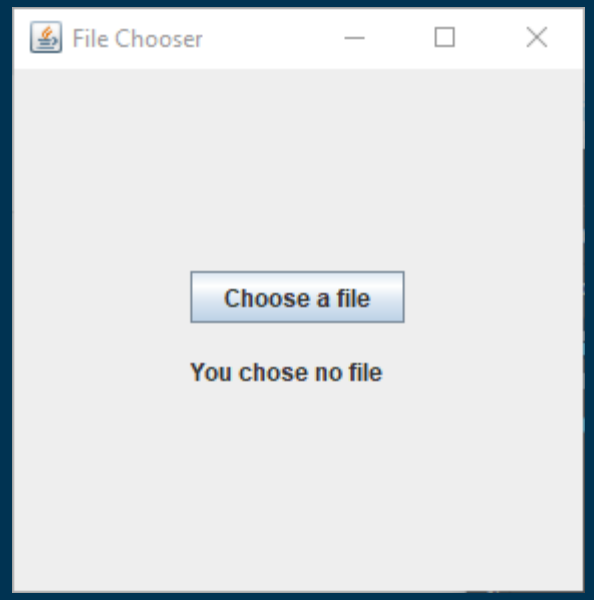
```
JFileChooser jfc = new JFileChooser(new File(System.getProperty("user.home")));
        int result = jfc.showOpenDialog(parent_window);
        if (result == JFileChooser.APPROVE_OPTION) {
            // jfc.getSelectedFile()  gets the file name
            // Work with the file name here
        }
```

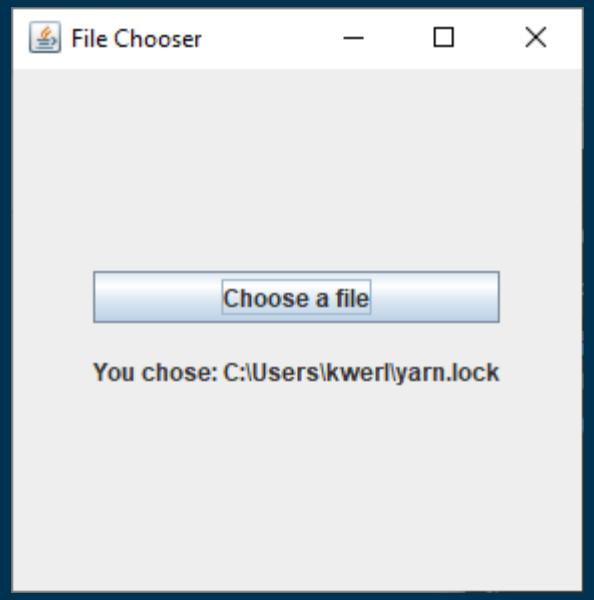
See Different ways to implement a listener

Excercise 5

- Create an application, which shows the file name chosen from a JFileChooser component:

Example:





Excercise 6

- Write a UI, which asks the Markdown file name from the user
- Then it calls the converter code from Day 1
- The output is saved in a „.latex“ file

- If time permits do the latex to pfd translation
- Show the pdf in a viewer.
  (This part we have not talked about an you will have work out how an external process can be called (and controlled) from Java) →