

# 252-0027

## Einführung in die Programmierung Übungen

### Woche 3: Einführung Java

Timo Baumberger

**Departement Informatik**

**ETH Zürich**



# Organisatorisches




- Mein Name: Timo Baumberger
- Bei Fragen: [tbaumberger@student.ethz.ch](mailto:tbaumberger@student.ethz.ch)  
(*Discord: troxhi*)
  - Mails bitte mit «[EProg25]» im Betreff
- Meine Website: [timobaumberger.com](http://timobaumberger.com)
- Neue Aufgaben: **Dienstag Abend** (im Normalfall)
- Abgabe der Übungen bis **Dienstag Abend (23:59)** Folgewoche
  - Abgabe immer via Git
  - Lösungen in separatem Projekt auf Git

# Inhalt

- **Theorie**
  - Operator Precedence
  - Kurzschlussauswertung
- **Aufgaben**
- **Praxisbeispiele**
- **Kahoot**
- **Nachbesprechung Übung 2**
- **Vorbesprechung Übung 3**

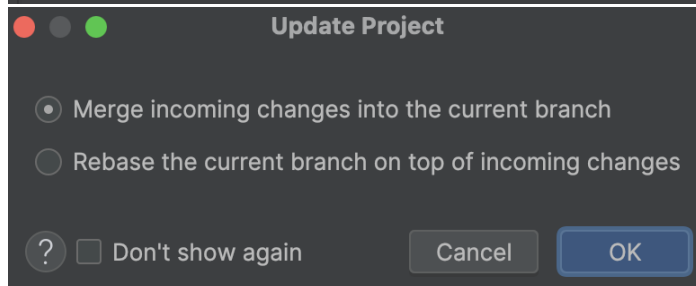
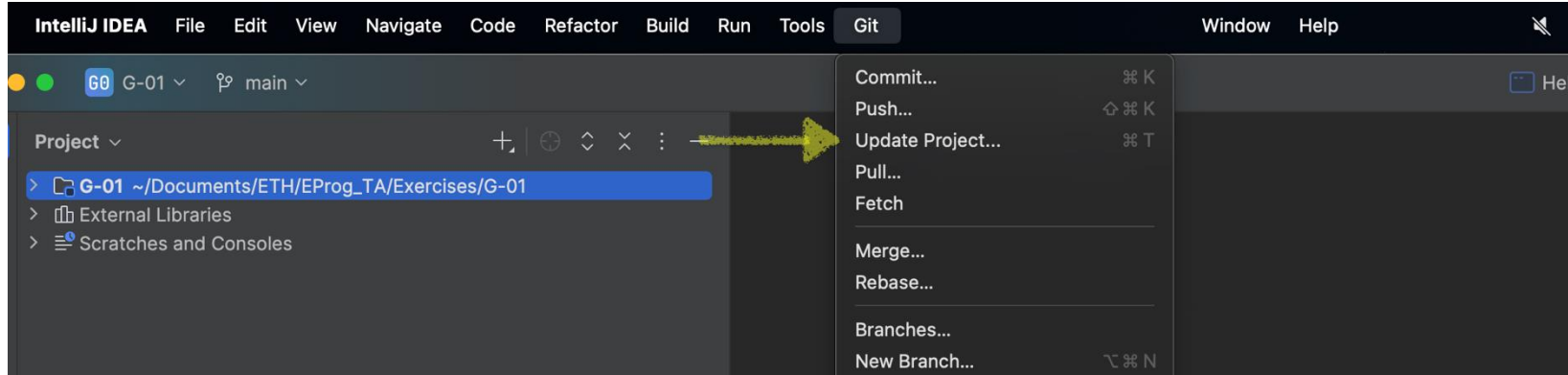
# Git Nachtrag

- Access Token haben ein Ablaufdatum
- Lifecycle einer Übung 
  - Git Pull
  - Übungsaufgaben lösen
  - Wenn alle Übungsaufgaben gelöst und zur Abgabe bereit sind:  
Git Pull, Commit und Push
  - Wieso Git Pull?
    - Divergierende Git Histories vermeiden
    - Fast-forward ist in 99% ausreichend
    - Fast-forward ist bei Divergenz **NICHT** möglich



# Git: Divergierende History

- Grundsätzlich kein Problem, kann aber einfach verhindert werden

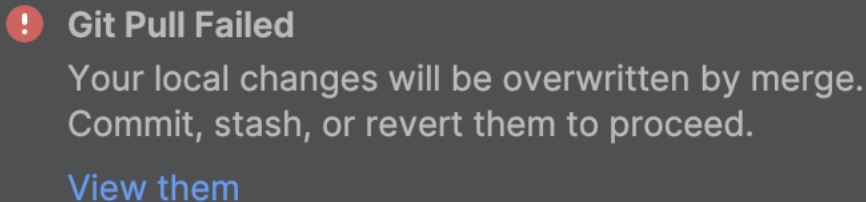


Nicht so cool, weil ihr jetzt zusätzlich einen Merge Commit in eurer Git History habt...

Rebase wäre die Lösung

# Wieso so kompliziert?

- Neue Übungsaufgabe: Git Pull
- Übungenaufgaben lösen
- Git Pull (bzw. Update Project mit Merge falls: )

A dark gray rectangular box containing a red circle with a white exclamation mark icon. To the right of the icon is the text "Git Pull Failed" in bold. Below it is the message "Your local changes will be overwritten by merge. Commit, stash, or revert them to proceed." in a lighter gray font. At the bottom left is a blue link "View them".

**! Git Pull Failed**  
Your local changes will be overwritten by merge.  
Commit, stash, or revert them to proceed.  
[View them](#)

Very confusing, not cool!!!!

- Verwendet Update Project (oder: `git pull --no-rebase`)
- Git Commit und Push

# Java



- Zur Entwicklung wird die Java JDK (Java Development Kit) verwendet
- Enthält Compiler, Libraries, wird zur Entwicklung von Programmen verwendet
- Java JRE (Java Runtime Environment) wird von “Java-Nutzern” verwendet
- Java JRE wird von 3 Milliarden Geräten verwendet
- Java Performance wird sehr oft falsch beurteilt

**Theorie**



# Auswertung – Arithmetische Operatoren

Operator	Beschreibung	Kurzbeispiel
+	Addition	<code>int</code> antwort = 40 + 2;
-	Subtraktion	<code>int</code> antwort = 48 - 6;
*	Multiplikation	<code>int</code> antwort = 2 * 21;
/	Division	<code>int</code> antwort = 84 / 2;
%	Teilerrest, Modulo-Operation, errechnet den Rest einer Division	<code>int</code> antwort = 99 % 57;
+	positives Vorzeichen	<code>int</code> j = +3;
-	negatives Vorzeichen	<code>int</code> minusJ = -j;

# Arithmetische Ausdrücke auswerten

- Was ist der “promoted type”?
- Numeric Types: byte, short, int, long, char, float, double
- Für uns ist normalerweise nur interessant: Integer oder Double?
- Type der Variablen und Konstanten int
- Welcher Type dominiert?



LONG



INTEGER

# Typen

- Integer: 100, -2, 40000
- Long: 100L, -2L, 4234L
- Double: 2.0, -4.0, 80001.5
- Float: 2.0F, 1F (nicht wichtig)

```
long java_youre_fine = 100_000_003L;  
float java_go_home_youre_drunk = java_youre_fine;
```

```
1.0E8  
100000003
```

double ><sub>1</sub> float

float ><sub>1</sub> long

???

long ><sub>1</sub> int

int ><sub>1</sub> char

int ><sub>1</sub> short

short ><sub>1</sub> byte

# Teilerrest / Modulo Operation

- Java garantiert, dass  $(a/b)*b+(a\%b) == a$  ist
- Somit gilt:  $a - (a/b)*b = a\%b$
- Bei Division von Ganzzahlen wird immer nach 0 gerundet

```
jshell> int r = -5 % 2  
r ==>
```

a?b:(c?d:(e?f:g))

oder

a=(b=c)

aber:

(a+b)+c

Level	Operator	Description	Associativity
16	()	parentheses	left-to-right
	[]	array access	
	new	object creation	
	.	member access	
	::	method reference	
15	++	unary post-increment	left-to-right
	--	unary post-decrement	
14	+	unary plus	right-to-left
	-	unary minus	
	!	unary logical NOT	
	~	unary bitwise NOT	
	++	unary pre-increment	
	--	unary pre-decrement	
13	()	cast	right-to-left
12	* / %	multiplicative	left-to-right
11	+ -	additive	left-to-right
	+	string concatenation	
10	<< >>	shift	left-to-right
	>>>		
9	< <=	relational	left-to-right
	> >=		
	instanceof		
8	==	equality	left-to-right
	!=		
7	&	bitwise AND	left-to-right
6	^	bitwise XOR	left-to-right
5		bitwise OR	left-to-right
4	&&	logical AND	left-to-right
3		logical OR	left-to-right
2	?:	ternary	right-to-left
1	=	assignment	right-to-left
	+= -=		
	*= /= %=		
	&= ^=  =		
	<<= >>= >>>=		
0	->	lambda expression	right-to-left
	->	switch expression	

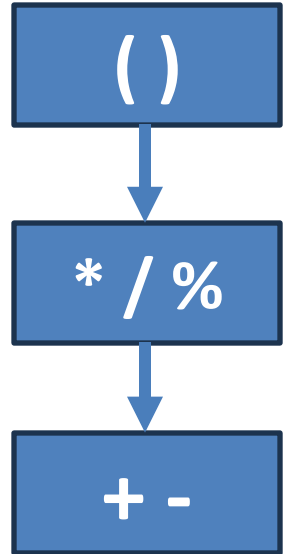


# Assoziativität

- Die **Assoziativität** («**associativity**») eines Operators bestimmt, wie ein Operand verwendet wird
- Ist ein Operator  $\otimes$ 
  - **Linksassoziativ** («**left-associative**»)  $\rightarrow X \otimes Y \otimes Z$  gleich  $(X \otimes Y) \otimes Z$
  - **Rechtsassoziativ** («**right-associative**»)  $\rightarrow X \otimes Y \otimes Z$  gleich  $X \otimes (Y \otimes Z)$

# Auswertung – Ordnung

- § Operand wird vom Operator mit höherer Rang Ordnung (“precedence”) verwendet
- § Wenn zwei Operatoren die selbe Rang Ordnung haben, dann entscheidet die Assoziativität
- § Wenn etwas anderes gewünscht wird: Klammern verwenden!



# Kurzschlussauswertung («short-circuit evaluation»)

Bei Berechnung von `p && q` und `p || q`

- Java wertet zuerst den linken Operanden (`p`) und dann den rechten (`q`) aus
- die Auswertung wird beendet, sobald das Ergebnis feststeht.
  - Bei `p && q`: Falls `p == false`, dann ist `p && q == false`
  - Bei `p || q`: Falls `p == true`, dann ist `p || q == true`

p	q	p && q
true	true	true
true	false	false
false	true	false
false	false	false

p	q	p    q
true	true	true
true	false	true
false	true	true
false	false	false

In diesen Fällen ist keine Auswertung von `q` nötig!



# Vorgehen

- **Gute Analogie: Anziehungskräfte (Chemie: Van-der-Waals-Kräfte und Wasserstoffbrückenbindungen)**
- **Zuerst da Klammern setzen, wo die die Bindungen / Kräfte am stärksten sind**
- **Von links nach rechts auswerten**

# Auswertung – Beispiele

()

\* / %

+ -

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println(2/3+4.5/3);  
4     }  
5 }
```

**Output:** 1.5

# Auswertung – Beispiele

( )

\* / %

+ -

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println(8%3+2.5+4/3);  
4     }  
5 }
```

**Output:** 5.5

# Auswertung – Casting

()

\* / %

+ -

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println((double)5/2);  
4     }  
5 }
```

**Output:** 2.5

# Auswertung – String

()

\* / %

+ -

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Wir schreiben das Jahr "+2023/2*2.0);  
4     }  
5 }  
6
```

**Output:** Wir schreiben das Jahr 2022.0

# Auswertung – Prüfungsbeispiel

( )

\* / %

+ -

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println(12/4*3+("X"+7%3)+18%5*2);  
4     }  
5 }
```

Output: 9X16

# Auswertung – Prüfungsbeispiel

()

\* / %

+ -

```
1
2 public class Main {
3     public static void main(String[] args) {
4         System.out.println("Wir schreiben das Jahr " + 2000 + 24);
5     }
6 }
```

**Output:** Wir schreiben das Jahr 200024

# Aufgabe 1

```
1  class MyClass{  
2      public static void main(String[] args){  
3          System.out.println(9 / 3 * 2 + 1 + "" + 2 * 3 + (4 * 3) % 3);  
4      }  
5  }
```

760



# Aufgabe 2



```
1  class MyClass{
2      public static void main(String[] args){
3          System.out.println(8 / 5 + 0.5 + 5 / 2 + (8 % 3) * 1.0);
4      }
5  }
```

5.5

# Aufgabe 3



```
class MyClass {  
  
    public static void main(String[] args) {  
        System.out.println(23 == 23 || (5/0 == 0));  
    }  
}
```

Ist die Ausgabe true oder false? Ist 5/0 ein Problem?

# Evaluation Order

- **Unabhängig von Assoziativität und Operator Precedence**
- **Operator Precedence und Assoziativität setzen Klammern**
- **Evaluation Order ist immer von links nach rechts**
- **Operator Precedence bestimmt nicht den Zeitpunkt der Evaluation**

# Aufgabe



```
class MyClass {  
    public static void main(String[] args) {  
        System.out.println(23 == 23 && 1 == 0 || 8*3 == 24%25);  
    }  
}
```

Was ist der Output?

# Zusammenfassung

- **Operator Precedence & Assoziativität zu Beginn identifizieren**
- **Von links nach rechts auswerten**
- **Kann gut mittels JShell geübt werden**

# Rechnung - Prüfungsaufgabe

```
1. System.out.println(36 / (3 * 12    4));
```

```
//Output: 4
```

```
2. System.out.println(64 % (4    8 + 1) / 4);
```

```
//Output: 7
```

```
3. System.out.println(32    (2    6));
```

```
//Output -8
```

**Bestimmen Sie für jede Anweisung die fehlenden Operatoren so, das die Anweisung die gezeigte Ausgabe erzeugt. Mögliche Operatoren sind +, -, \*, / und %.**

# Rechnung - Prüfungsaufgabe

1. `System.out.println(36 / (3 * 12 / 4));`

`//Output: 4`

2. `System.out.println(64 % (4 * 8 + 1) / 4);`

`//Output: 7`

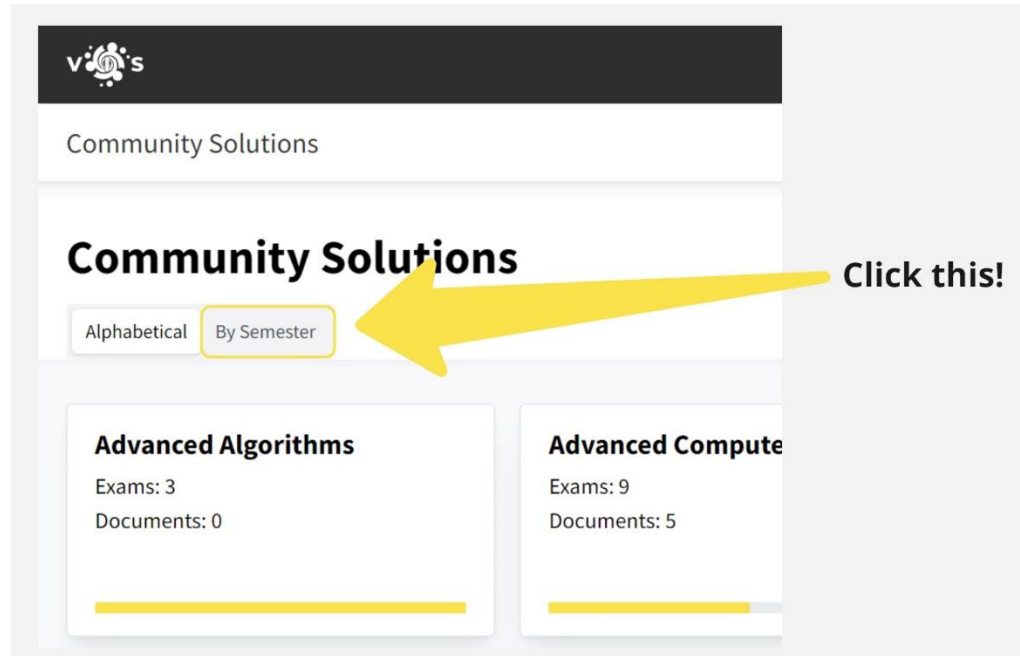
3. `System.out.println(32 / (2 - 6));`

`//Output -8`

**Bestimmen Sie für jede Anweisung die fehlenden Operatoren so, das die Anweisung die gezeigte Ausgabe erzeugt. Mögliche Operatoren sind +, -, \*, / und %.**

# Rechnung - Prüfungsaufgabe

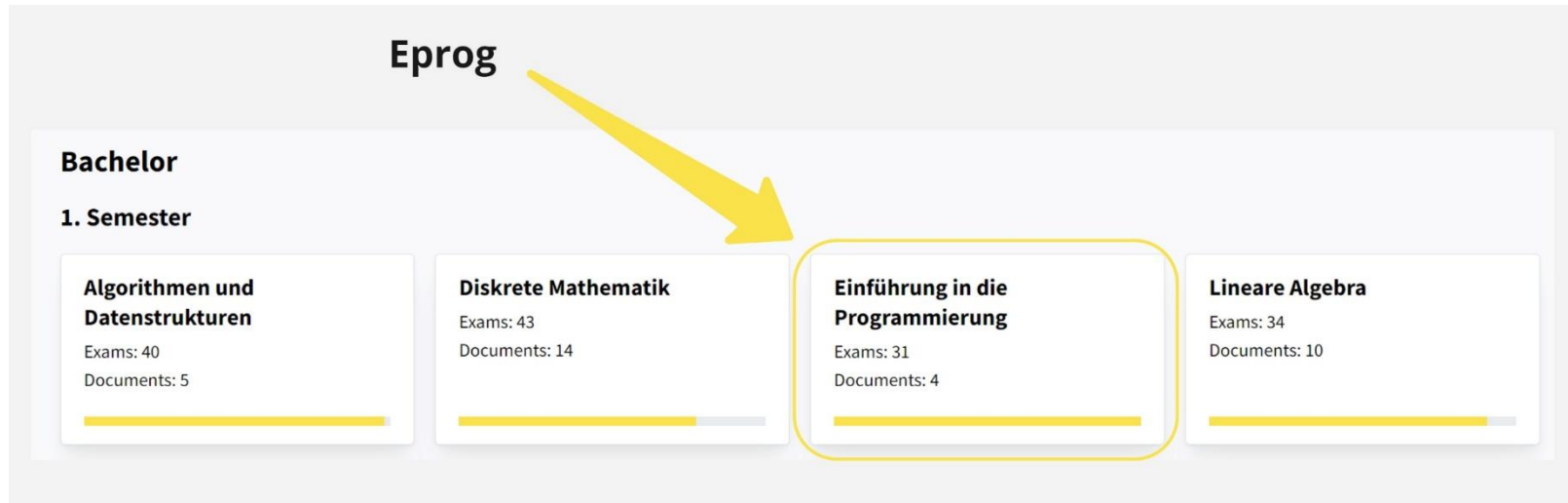
- Prüfungen auf: <https://exams.vis.ethz.ch>





# Rechnung - Prüfungsaufgabe

- Prüfungen auf: <https://exams.vis.ethz.ch>



# Rechnung - Prüfungsaufgabe

- Alte Prüfungen auf: <https://exams.vis.ethz.ch>
- Alte Prüfungen lösen ist nicht Teil des Semesters.
  - Dafür bleibt genügend Zeit in der Lernphase in der vorlesungsfreien Zeit vor den Prüfungen.
- Alte Prüfungsaufgaben werden von Zeit zu Zeit in den Übungen kommen, aber es verbleiben genügend alte Prüfungen, welche ihr selbstständig lösen könnt.

# Nützliche Methoden / Klassen

- **Math.random()** oder die Klasse **Random** kann verwendet werden um pseudorandom Nummern zu generieren.
- **Achtung:** **Math.random()**  $\in [0, 1)$
- **Achtung:** **random.nextInt(i)**  $\in [0, i) = [0, i - 1]$
- **Scanner** kann verwendet werden um Daten von einer Quelle zu lesen
- **Scanner scanner = new Scanner(System.in);**

# Kahoot

<https://create.kahoot.it/details/6fa5d268-3c90-45a1-9c5f-66f68029280f>

# **Nachbesprechung Übung 2**

**Weitere Fragen zu IntelliJ oder Git?**

**(Interactive)**

# Lösung Übung 2, Aufgabe 2

```
/*  
 * Author: Maximiliana Muster  
 * für Einführung in die Programmierung  
 */  
public class HelloProgrammer {  
  
    public static void main(String[] args) {  
        System.out.println("Hello, my name is Max!");  
    }  
}
```

# Lösung Übung 2, Aufgabe 4

1. Erstellen Sie eine Beschreibung `<geradezahl>`, die als legale Symbole alle geraden Zahlen (d.h. Zahlen, die ohne Rest durch 2 teilbar sind) zulässt. Beispiele sind +02, 4, 10, -20.



# Lösung Übung 2, Aufgabe 4

2. Zeigen Sie in einer Tabelle, dass Ihre Beschreibung das Symbol “28” als gerade Zahl erkennt.

# Lösung Übung 2, Aufgabe 4

3. Erstellen Sie eine Beschreibung `<x2ygemischt>`, die als legale Symbole genau jene Wörter zulässt, in denen für jedes "X" zwei "Y" als Paar auftreten. Beispiele sind `XYX`, `YYX`, `XYYYXX`, `XXYYYY`. Die Beschreibung schließt den leeren String nicht aus.

# Lösung Übung 2, Aufgabe 4

Die folgenden EBNF-Beschreibungen sind nicht äquivalent. Finden Sie ein *kürzestmögliches* Symbol, das von der einen Beschreibung als legal erkannt wird, aber nicht von der anderen. (Fangen Sie mit einfachen Kombinationen von A und B an.)

**EBNF-Beschreibung:** <beispiel1>

<beispiel1>  $\Leftarrow$  [A][B]

**EBNF-Beschreibung:** <beispiel2>

<beispiel2>  $\Leftarrow$  [A[B]]

# Lösung Übung 2, Aufgabe 4

5. Erstellen Sie eine EBNF Beschreibung `<doppelt>`, die als legale Symbole genau jene Wörter zulässt, in denen die doppelte Anzahl "Y" nach einer Folge von "X" auftritt. Beispiele sind `XYX`, `XXYYYY`, usw. Die Beschreibung schliesst den leeren String nicht aus.

# **Vorbesprechung Übung 3**

# Aufgabe 1: Fehlerbehebung

Finden und beheben Sie alle Fehler im Programm "FollerVehler.java". Eclipse hilft Ihnen dabei, indem es anzeigt, wo die Fehler sind (und eine mehr oder weniger hilfreiche Fehlermeldung dazu ausgibt), aber Sie müssen selber herausfinden, was das Problem ist und wie Sie es beheben können. Wenn Sie alle Fehler behoben haben, sollte das Programm folgendes ausgeben:

```
Hello world
```

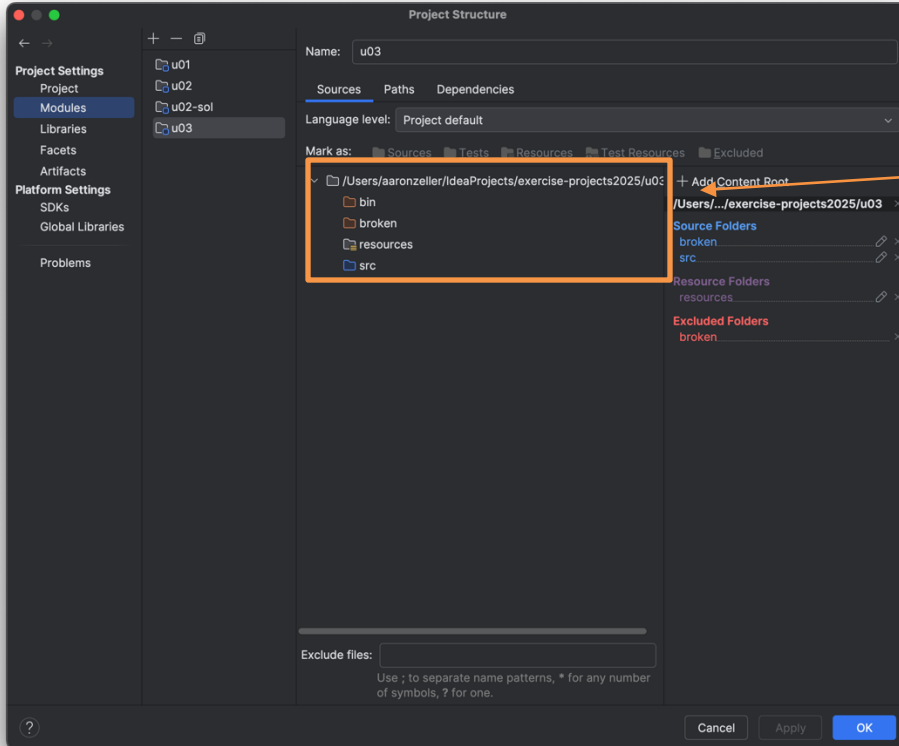
```
Gefällt Ihnen dieses Programm?
```

```
Ich habe es selbst geschrieben.
```

# Aufgabe 1: Fehlerbehebung

- In IntelliJ werden **alle** Java-Dateien in einem Projekt kompiliert.
- Wenn auch **nur eine** Java-Datei nicht kompiliert, dann kann **kein** Java-Code ausgeführt werden in dem Projekt.

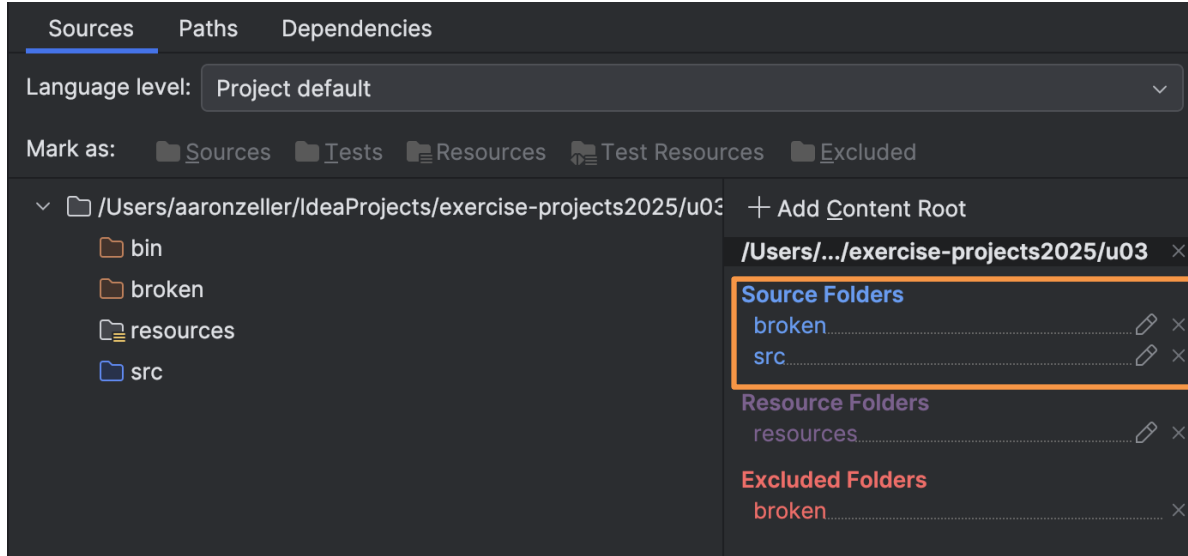
# Aufgabe 1: Fehlerbehebung



Hier sind die Unterordner



# Aufgabe 1: Fehlerbehebung



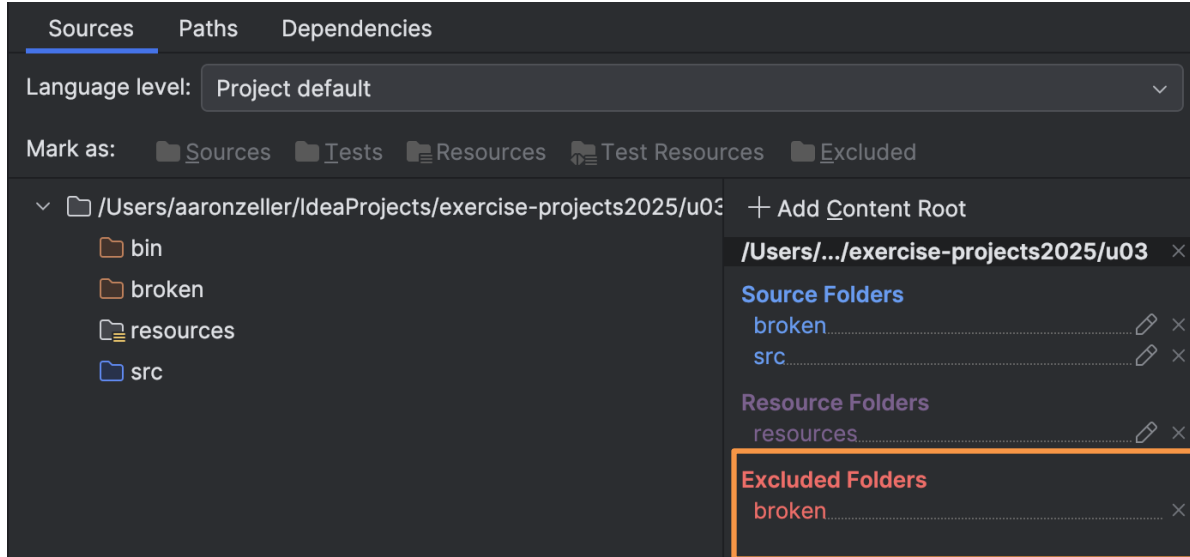
The screenshot shows the 'Sources' tab in IntelliJ IDEA. The 'Language level' is set to 'Project default'. The 'Mark as' section shows icons for Sources, Tests, Resources, Test Resources, and Excluded. The project structure on the left includes folders: bin, broken, resources, and src. The right pane shows the configuration for the project root, with 'Source Folders' (broken and src) highlighted by an orange box. Below this are 'Resource Folders' (resources) and 'Excluded Folders' (broken).

Category	Folder	Action
Source Folders	broken	[Edit] [X]
	src	[Edit] [X]
Resource Folders	resources	[Edit] [X]
Excluded Folders	broken	[X]

Alle "Sources" werden kompiliert



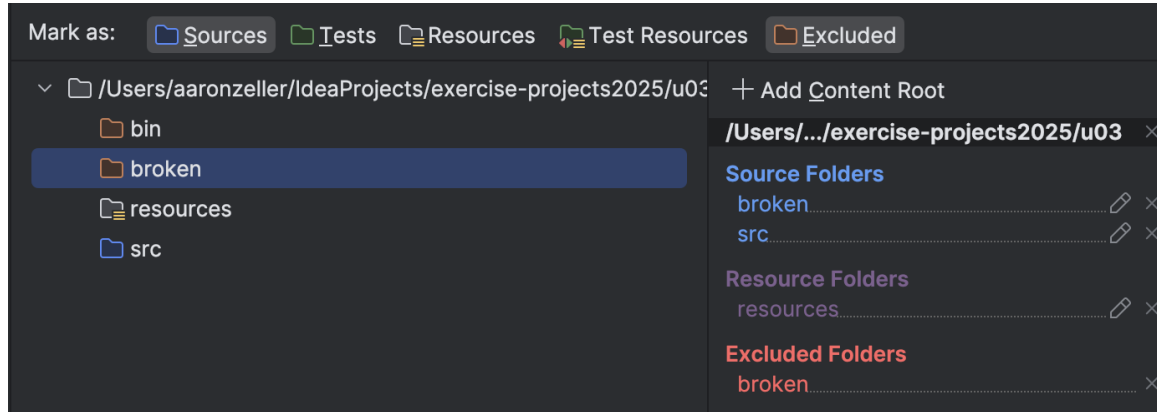
# Aufgabe 1: Fehlerbehebung



Ausser sie sind "Excluded"

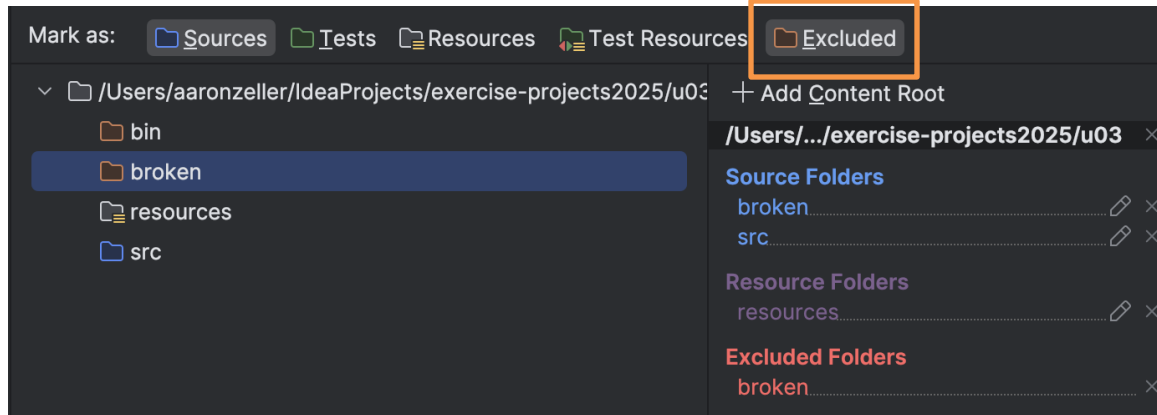


# Aufgabe 1: Fehlerbehebung



Wenn man auf einen Unterordner klickt, dann kann man ihn als “Sources”, “Excluded”, “Resources”, etc. markieren.


# Aufgabe 1: Fehlerbehebung

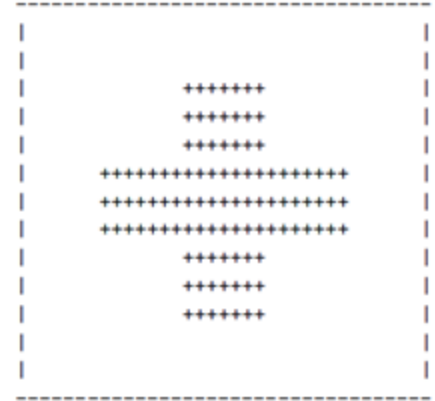


Wenn man auf “Excluded” klickt, dann wird der broken-Ordner wieder kompiliert!

# Aufgabe 2: Schweizerfahne

Statt eine Vorlage zu benützen, schreiben Sie in dieser Aufgabe ein Programm von Grund auf selbst. Eclipse wird Ihnen allerdings etwas Schreibarbeit abnehmen.

1. Erstellen Sie eine neue Java-Datei "SwissFlag.java". Wählen Sie dafür im Menü *File* → *New* → *Class* oder klicken Sie auf das  Symbol in der Symbolleiste. Geben Sie im Dialog bei *Name* "SwissFlag" ein und drücken Sie "Finish".
2. Erstellen Sie zuerst eine leere `main`-Methode, so wie Sie es bei anderen Programmen gesehen haben. (Tipp: Sie können in Zukunft auch die Option *public static void main(String[] args)* im Dialog für neue Java-Klassen auswählen, um sich ein wenig Arbeit zu sparen.)
3. Erweitern Sie das Programm so, dass es die Schweizerfahne in der Konsole ausgibt. Die Fahne könnte ungefähr wie folgt aussehen, Sie dürfen aber auch eine grössere oder schönere Version entwerfen:



Teilen Sie das Programm in mehrere Methoden auf, welche von der `main`-Methode aufgerufen werden. Damit sorgen Sie dafür, dass weniger Wiederholungen von Code-Stücken vorkommen, was das Ändern des Programms deutlich einfacher macht.

# Aufgabe 3: Eingabe und Zufall

## Aufgabe 3: Eingabe und Zufall

In dieser Aufgabe arbeiten Sie mit der Eingabe und Ausgabe von Java und lernen die Klassen `Scanner` und `Random` näher kennen.

1. Schreiben Sie ein Programm "Adder.java", welches zwei ganze Zahlen einliest und die Summe davon ausgibt. Sie sollen dafür die `Scanner`-Klasse verwenden, wie in der Vorlesung gezeigt. Das Programm soll nach der ersten Zahl fragen:

Geben Sie Zahl 1 ein:

dann nach der zweiten Zahl:

Geben Sie Zahl 2 ein:

und schliesslich, wenn Sie zum Beispiel "4" und "1999" eingeben, folgendes ausgeben:

4 + 1999 = 2003

Sie können davon ausgehen, dass nur ganze Zahlen eingegeben werden. Testen Sie das Programm mit verschiedenen Zahlen.

2. Schreiben Sie ein Programm `Wuerfel.java`, das einen Würfel simuliert. Hierbei soll eine positive ganze Zahl `N` eingelesen werden, welche die Anzahl der Seiten des Würfels repräsentiert. Der übliche Würfel hat 6 Seiten, jedoch existieren auch Würfel mit 12, 16, 20 (siehe Abbildung 1) und mehr Seiten. Jede Seite trägt eine unterschiedliche Zahl, die von 1 bis `N` (inklusive `N`) reicht.

Das Programm soll den Wurf simulieren, indem es die Klasse `Random` verwendet. Ein möglicher Ablauf des Programmes könnte folgendermassen aussehen:

Wie viele Seiten hat Ihr Würfel?

Der Benutzer gibt eine Zahl ein, z.B. 20, danach wird gewürfelt:

Es wurde eine 17 gewürfelt!



3. Schreiben Sie ein Programm `ChatGTP.java`, welches die Interaktion mit einem AI-Chatbot simuliert. Das Programm soll den Benutzer begrüßen, Sie nach Ihrem Namen und Alter fragen, und dem Benutzer Ihre Glückszahl verraten. Die Glückszahl soll zufällig gewählt werden, in dem Sie die `Random` Klasse benutzen. Ein möglicher Ablauf des Programmes könnte folgendermassen aussehen:

Guten Tag! Ich bin ChatGTP, der beste Chatbot, denn es gibt! Wie heissen Sie?

Ein Namen wird eingegeben, z.B. Alice, danach antwortet der Chatbot auf diesen Input:

Sehr erfreut Alice! Wie alt sind Sie?

Ein Alter wird eingegeben, z.B. 23, danach antwortet der Chatbot auf diesen Input:

Mittels dieser Informationen habe ich Ihre Glückszahl gefunden! Die Glückszahl lautet 42.



# Aufgabe 4: Berechnung

2. Vervollständigen Sie "SharedDigit.java". In der Main-Methode sind zwei int Variablen a und b deklariert und mit einem Wert zwischen 10 und 99 (einschliesslich) initialisiert. Das Programm soll einer int Variablen r einen bestimmten Wert zuweisen. Wenn a und b eine Ziffer gemeinsam haben, dann wird r die gemeinsame Ziffer zugewiesen (wenn a und b beide Ziffern gemeinsam haben, dann kann eine beliebige Ziffer zugewiesen werden). Wenn es keine gemeinsame Ziffer gibt, dann soll -1 zugewiesen. Sie brauchen für dieses Programm keine Schleife.

Beispiele:

- Wenn a: 34 und b: 53, dann ist r: 3
- Wenn a: 10 und b: 22, dann ist r: -1
- Wenn a: 66 und b: 66, dann ist r: 6
- Wenn a: 34 und b: 34, dann ist r: 3 oder 4

Testen Sie Ihre Loesung mit a gleich 34 und b gleich 43. Was liefert Ihr Programm?

2. Vervollständigen Sie "SumPattern.java". In der Main-Methode sind drei int Variablen a, b, und c deklariert und mit irgendwelchen Werten initialisiert. Wenn die Summe von zwei der Variablen die dritte ergibt, nehmen wir an dass  $a + c == b$ , so soll die Methode "Moeglich.  $a + c == b$ " ausgeben (wobei die Werte für a, b, und c einzusetzen sind). Wenn das nicht der Fall ist, dann soll die Methode "Unmoeglich." ausgeben.

Beispiele:

- Wenn a: 4, b: 10, c: 6, dann wird "Moeglich.  $4 + 6 == 10$ " oder "Moeglich.  $6 + 4 == 10$ " ausgegeben.
- Wenn a: 2, b: 12, c: 0, dann wird "Unmoeglich." ausgegeben.

3. Vervollständigen Sie "AbsoluteMax.java". In der Main-Methode sind drei int Variablen a, b, und c deklariert und mit irgendwelchen Werten initialisiert. Das Programm soll einer int Variable r den grössten absoluten Wert von a, b, und c zuweisen.

# Aufgabe 5: EBNF

## Aufgabe 5: EBNF

In dieser Aufgabe erstellen Sie erneut verschiedene EBNF-Beschreibungen. Speichern Sie diese wie gewohnt in der Text-Datei "EBNF.txt", welche sich in Ihrem "u02"-Ordner bzw. im "U02 <N-ETHZ-Account>"-Projekt befindet. Sie können die Datei direkt in Eclipse bearbeiten.

1. Erstellen Sie eine Beschreibung <pyramid>, welche als legale Symbole genau jene Wörter zulässt, welche aus einer Folge von strikt aufsteigenden, gefolgt von einer Folge von strikt absteigenden Ziffern bestehen. Beispiele sind: 14, 121, 1221, 1341.  
Sie dürfen annehmen, dass das leere Wort auch zugelassen wird. (Als Challenge können Sie probieren, das leere Wort auszuschliessen.)

# Aufgabe 5: EBNF

2. Erstellen Sie eine Beschreibung  $\langle \text{digitsum} \rangle$ , welche als legale Symbole genau jene natürlichen Zahlen zulässt, deren Quersumme eine gerade Zahl ist.
3. Erstellen Sie eine Beschreibung  $\langle \text{xyz} \rangle$ , welche genau alle Wörter zulässt, die aus X, Y und Z bestehen und bei welchen jedes X mindestens ein Y im Teilwort links und rechts von sich hat. Beispiele sind: Z, YXY, YXXY, ZYXYY.
4. Erstellen Sie eine Beschreibung  $\langle \text{term} \rangle$ , welche als legale Symbole genau alle wohlgeformten arithmetischen Terme, bestehend aus positiven ganzen Zahlen, Variablen (x, y, z), Addition und Klammern zulässt. Geklammerte Terme müssen mindestens eine Addition enthalten. Beispiele sind:  $1 + 4$ ,  $(1 + 4)$ ,  $1 + (3 + 4)$ ,  $(1 + 1) + x + 5$ .