252-0027

Einführung in die Programmierung
Übungen

Woche 4: Types, Schleifen, Methoden

Timo Stucki
Departement Informatik
ETH Zürich

Organisatorisches

- Mein Name: *Timo Stucki*
- Bei Fragen: *tistucki@student.ethz.ch*
  - Mails bitte mit «[EProg25]» im Betreff
- 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

Discord: timostucki

Types

Primitive Typen in Java

Acht primitive Typen («primitive types»): für Zahlen, Buchstaben und Wahrheitswerte. Beispiele:

Name	Beschreibung	Beispiele
`int`	ganze Zahlen	`-2147483648`, `-3`, `0`, `42`, `2147483647`
`long`	grosse ganze Zahlen	`42`, `-3`, `0`, `92233720368547758O7`
`double`	reelle Zahlen	`3.1`, `-0.25`, `9.4e3`
`char`	(einzelne) Buchstaben	`'a'`, `'X'`, `'?'`, `'\n'`
`boolean`	Wahrheitswerte	`true`, `false`

Wer es genau wissen möchte: Java Language Specs §4.2

Fragen: Wozu evaluieren diese Ausdrücke?

1. `(30 * 2 + 5) % 7 / 3` ergibt 0
2. `485 % (12 + 6 * 2) – 4 * 5 / 2` ergibt -5
3. `(8 + 15 * 3 % 7) / 2 + 9` ergibt 14
4. `10 * (6 + 5 % 4) – 20 / 4 + 8` ergibt 73
5. `100 % (50 / 2) + 75 % (12 * 2) - 2` ergibt 1
6. `(9 + 3) * 2 – 18 % (5 + 2 * 3)` ergibt 17
7. `(6 * 3 + 2) % 7 – (14 / 2 + 4)` ergibt -5

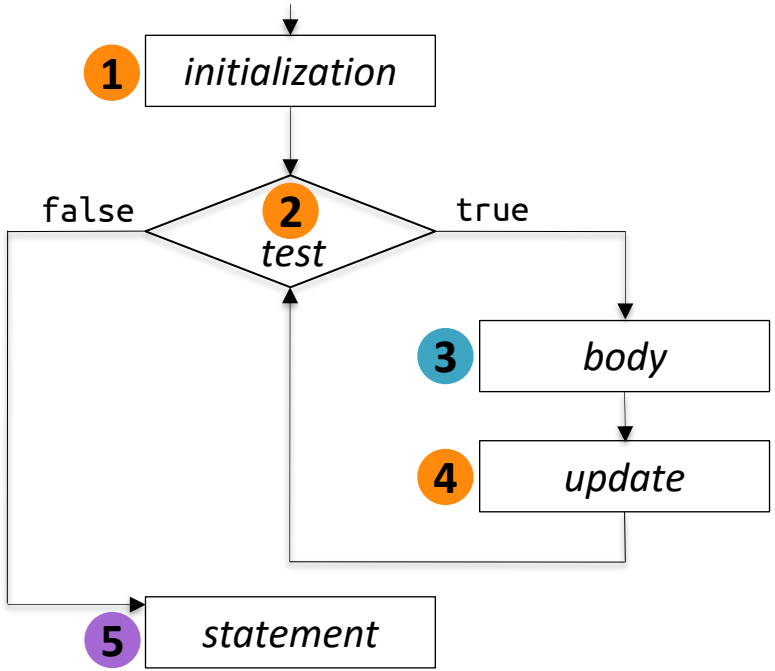
Schleifen

for-Schleife («for loop»): Kontrollfluss

```
for (initialization; test; update) {
    body
}
statement;
```

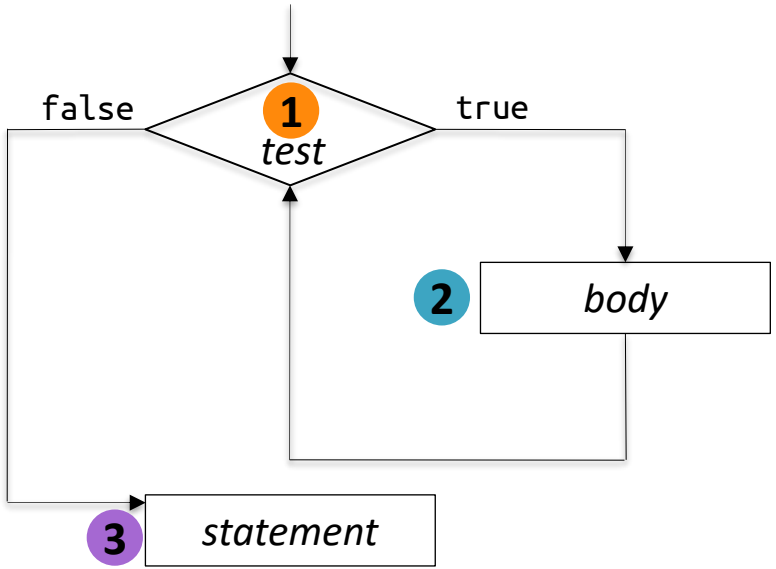
1 2 4 3 5

```
for (int i = 1; i <= 5; i = i + 1) {
 System.out.println(i); // 1 2 3 4 5
}
```



while-Schleife («while loop»)

```
while (test) {
    body
}
statement;
```



while-Schleife führt Rumpf so lange aus, wie *test* den Wert `true` ergibt.
In anderen Worten: die while-Schleife bricht ab, sobald *test* den Wert `false` ergibt.

Schleifen

Was gibt diese Methode aus?

```
public static void main (String[] args) {
    int f = 0; int g = 1;

    for (int i = 0; i < 15; i++ ) {
        System.out.print(" " + f);
        f = f + g;
        g = f - g;
    }
    System.out.println();
}
```

i	f	g
	0	1
0	0	1
0	**0**	1
0	1	1
0	1	0
1	**1**	0
…	…	…

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

Schleifen: While vs. For vs. Do-While

- **For-Loop:** Benutzen wir, wenn wir die Anzahl Iterationen bereits for der Ausführung des Loops kennen.
- **While:** Benutzen wir, wenn wir die Anzahl Iterationen nicht kennen und diese Abhängig von einer Bedingung ist.
- **Do-While:** Wird benutzt, wenn wir zuerst den Code (im Loop Body) ausführen und dann erst die Bedingung prüfen wollen.

Schleifen: For vs. While

- **While und Do-While können das gleiche.**
- **For und While aber auch!**

Schleifen: For vs. While – Beispiel 1

```java
public class MyClass {
    public static void main(String[] args){
        for(int i = 0; i < 5; i++){
            System.out.println("i = " + i);
        }
    }
}
```

Schleifen: For vs. While – Beispiel 1

```java
public class MyClass {
    public static void main(String[] args){
        int i = 0;          // Initialisierung
        while(i < 5){       // Bedingung
            System.out.println("i = " + i);
            i++;            // Aktualisierung
        }
    }
}
```

Schleifen: For vs. While – Beispiel 2

```java
public class MyClass {
    public static void main(String[] args){
        int i = 1;
        while(fancyCheck(i)){
            int result = complexOperation(i);
            i = i * 42 - 15;
            System.out.println(result);
        }
    }
}
```

Schleifen: For vs. While – Beispiel 2

```java
public class MyClass {
    public static void main(String[] args){
        int i = 1;
        for(; fancyCheck(i); ){
            int result = complexOperation(i);
            i = i * 42 - 15;
            System.out.println(result);
        }
    }
}
```

Schleifen: Do-While?

- **Selten genutzt aber kann sehr nützlich sein!**

```java
public class MyClass {
    public static void main(String[] args){
        Scanner console = new Scanner(System.in);
        String pin;
        do {
            pin = console.nextLine();
        } while(!isPassword(pin));
        System.out.println("You logged in successfully!");
        console.close();
    }
}
```

Inkrement und Dekrement

Inkrement und Dekrement

- **Präfix: ++i**

```
int a = 5;
int b = ++a;  // Zuerst wird a inkrementiert, dann wird a zugewiesen.
// a = 6, b = 6
```

- **Postfix i++**

```
int a = 5;
int b = a++;  // Zuerst wird a zugewiesen, dann wird a inkrementiert.
// a = 6, b = 5
```

Diese Operatoren sind besonders nützlich in Schleifen, um beispielsweise einen Zähler zu erhöhen oder zu verringern.

Weitere Kurzformen

- **Prä-Inkrement («pre-increment»)** und **Prä-Dekrement («pre-decrement»)**: zuerst ändern, dann verwenden
  - `++variable` für `variable = variable + 1`
  - `--variable` für `variable = variable - 1`
- Veränderung mit beliebigen Wert (statt nur ± 1)
  - `variable += value` für `variable = variable + value;`
  - `variable -= value` für `variable = variable - value;`
  - `variable *= value` für `variable = variable * value;`
  - `variable /= value` für `variable = variable / value;`
  - `variable %= value` für `variable = variable % value;`

Achtung:

```
x += 3
x =+ 3
```

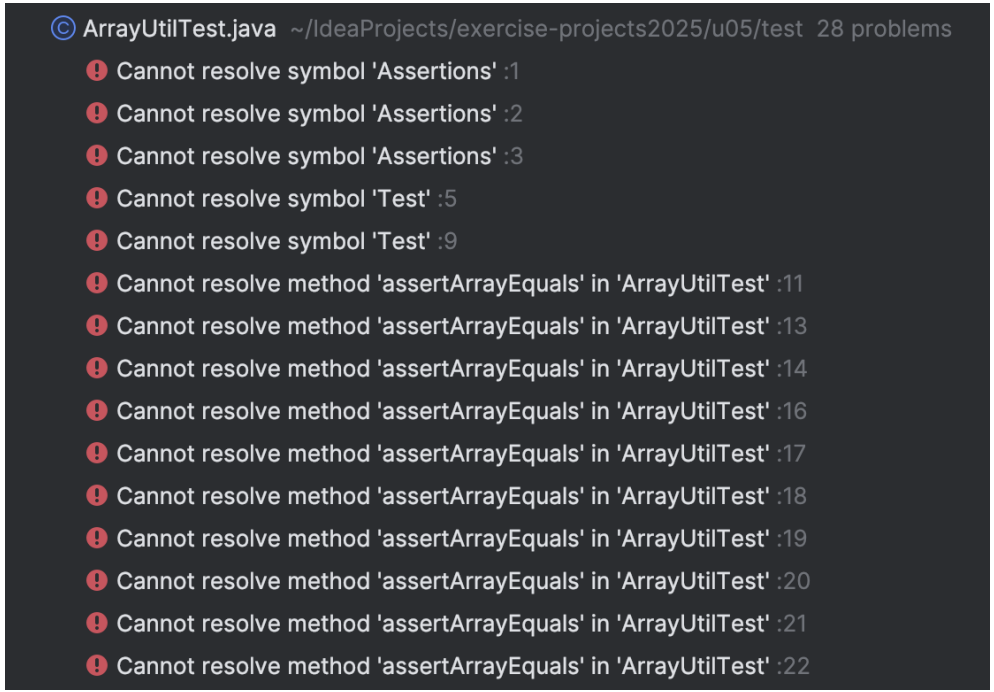
JUnit

JUnit Tests: Introduction

- **JUnit Test erlauben es Code zu testen.**
- **Jetzt: Tests benutzen**
- **Später: Tests selbst schreiben.**

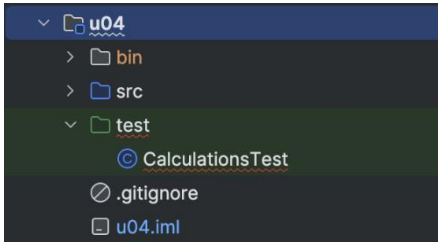
JUnit Tests: IntelliJ

- **JUnit kann nicht gefunden werden?**

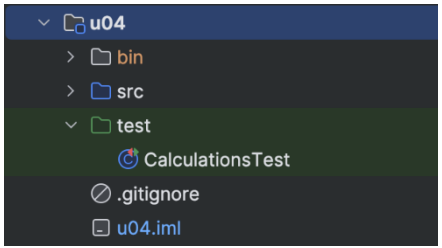


JUnit Tests: IntelliJ

- **JUnit kann nicht gefunden werden?**

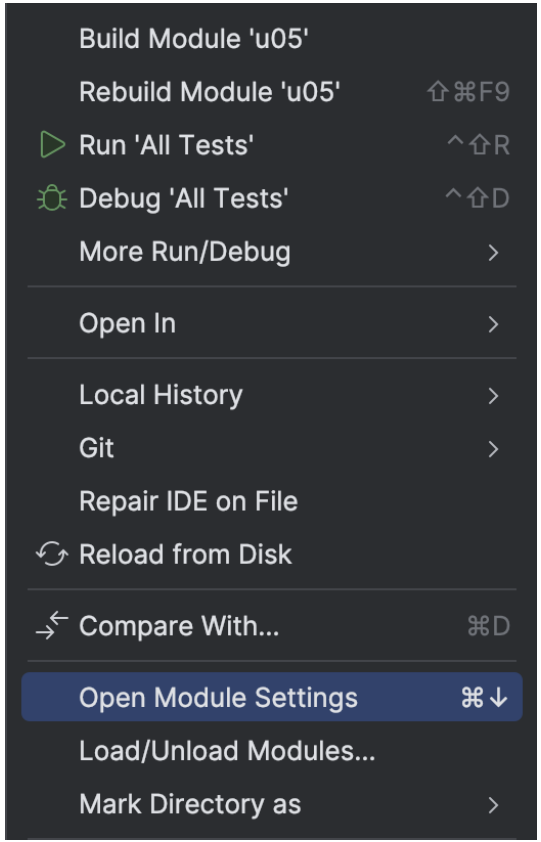


- **JUnit library muss importiert werden!**



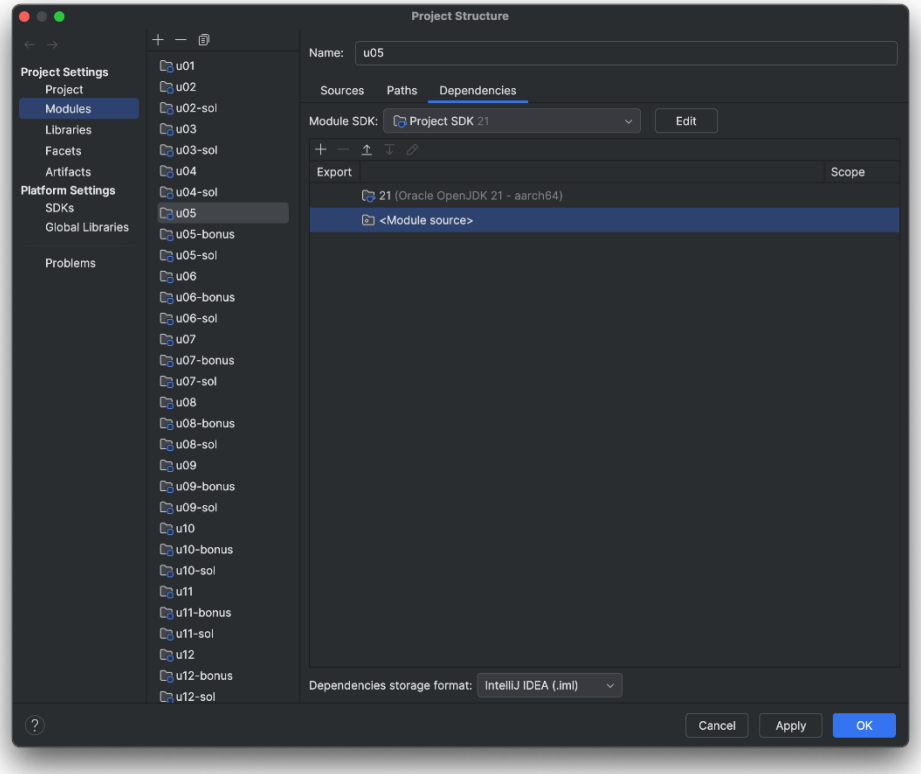
JUnit Tests: JUnit Importieren

- **Rechtklick auf Project Folder, z.B. u03.**
- **1. Open Module Settings**



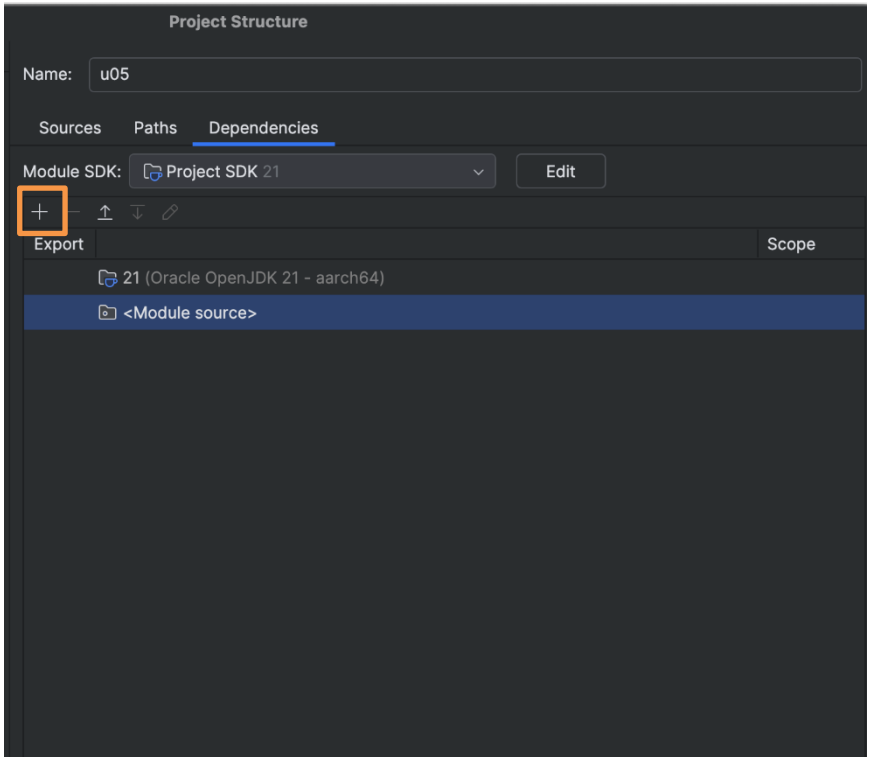
JUnit Tests: JUnit Importieren

- Rechtklick auf Project Folder, z.B. u03.
- 1. Open Module Settings
- 2. Click + in Project Structure



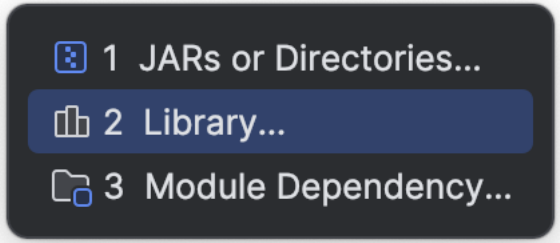
JUnit Tests: JUnit Importieren

- Rechtklick auf Project Folder, z.B. u03.
- 1. Open Module Settings
- 2. Wähle + in Project Structure



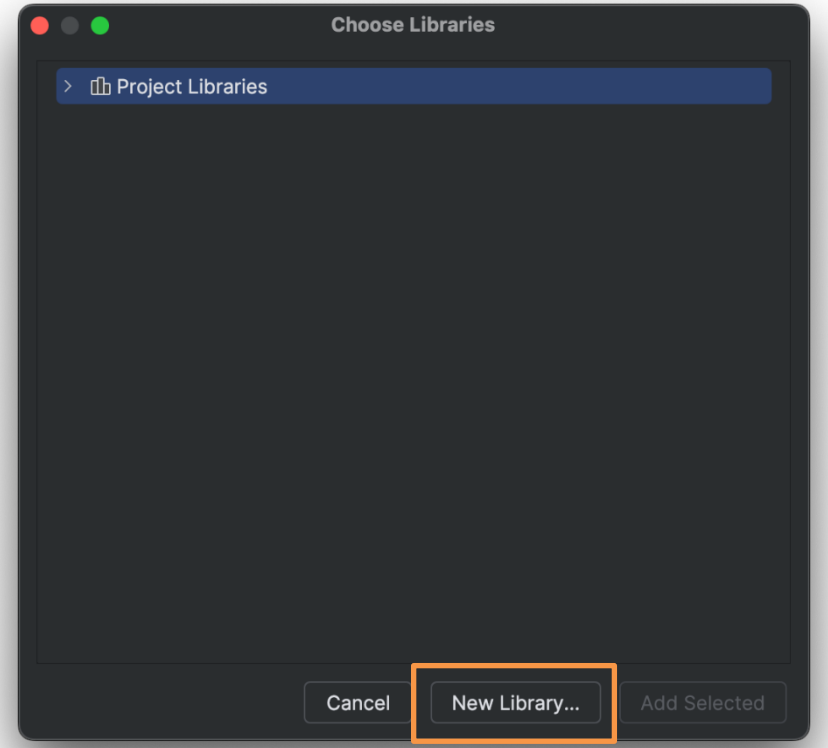
JUnit Tests: JUnit Importieren

- **Rechtklick auf Project Folder, z.B. u03.**
- **1. Open Module Settings**
- **2. Wähle + in Project Structure**
- **3. Library… -> New Library**



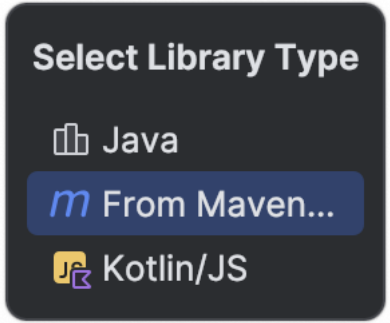
JUnit Tests: JUnit Importieren

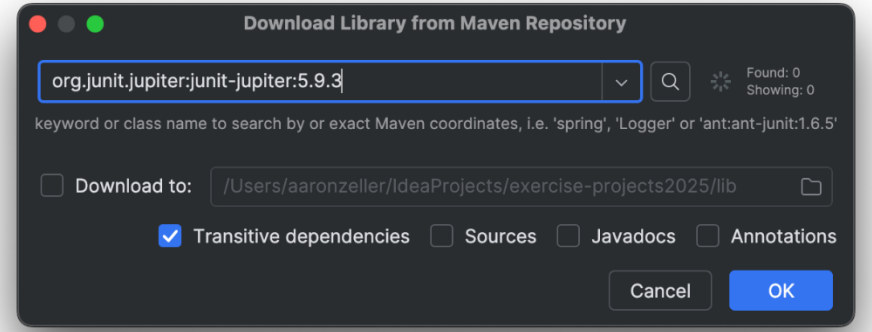
- Rechtklick auf Project Folder, z.B. u03.
- 1. Open Module Settings
- 2. Wähle + in Project Structure
- 3. Library… -> New Library



JUnit Tests: JUnit Importieren

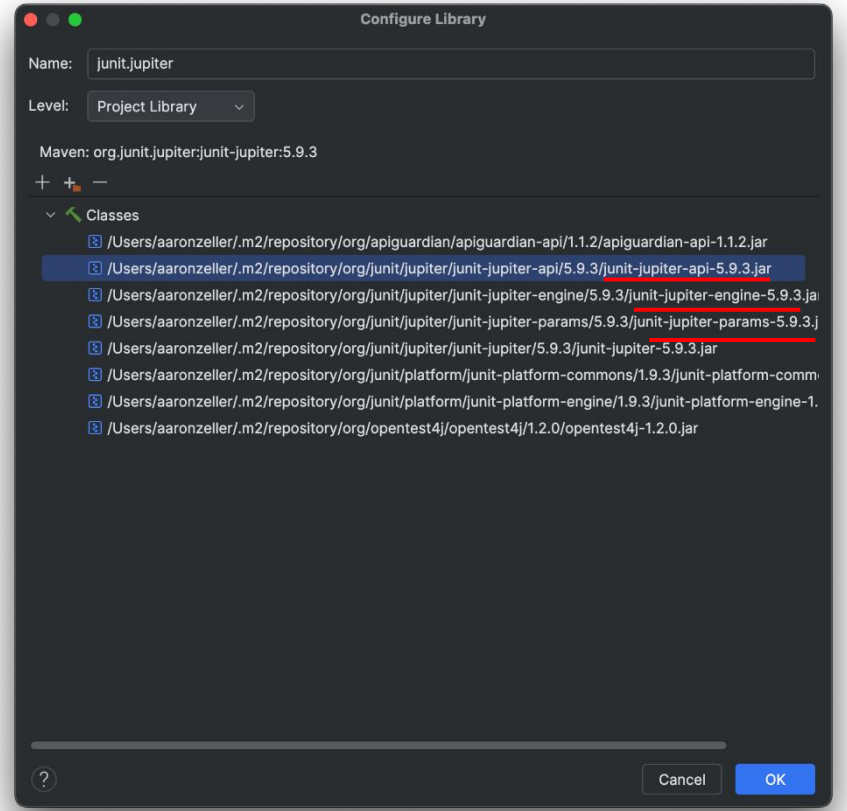
- Rechtklick auf Project Folder, z.B. u03.
- 1. Open Module Settings
- 2. Wähle + in Project Structure
- 3. Library… -> New Library
- 4. From Maven… -> Suche
  org.junit.jupiter:junit-jupiter:5.9.3





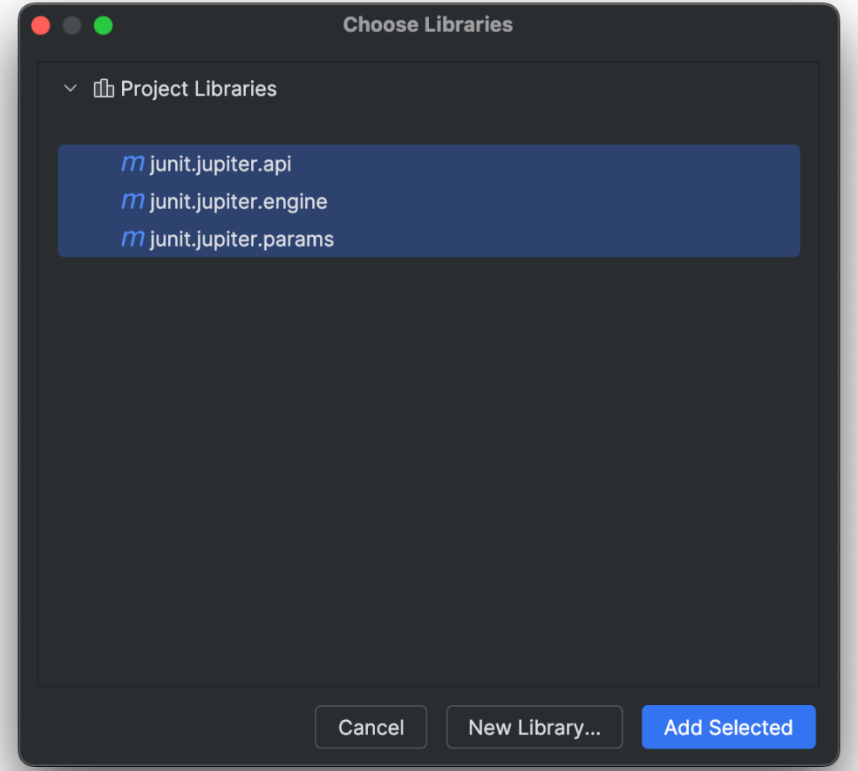
JUnit Tests: JUnit Importieren

- **Wir brauchen:**
  - junit-jupiter-api-5.9.3.jar
  - junit-jupiter-engine.5.9.3.jar
  - junit-jupiter-params.5.9.3.jar
- **Wähle die jar-Dateien und klicke *OK***



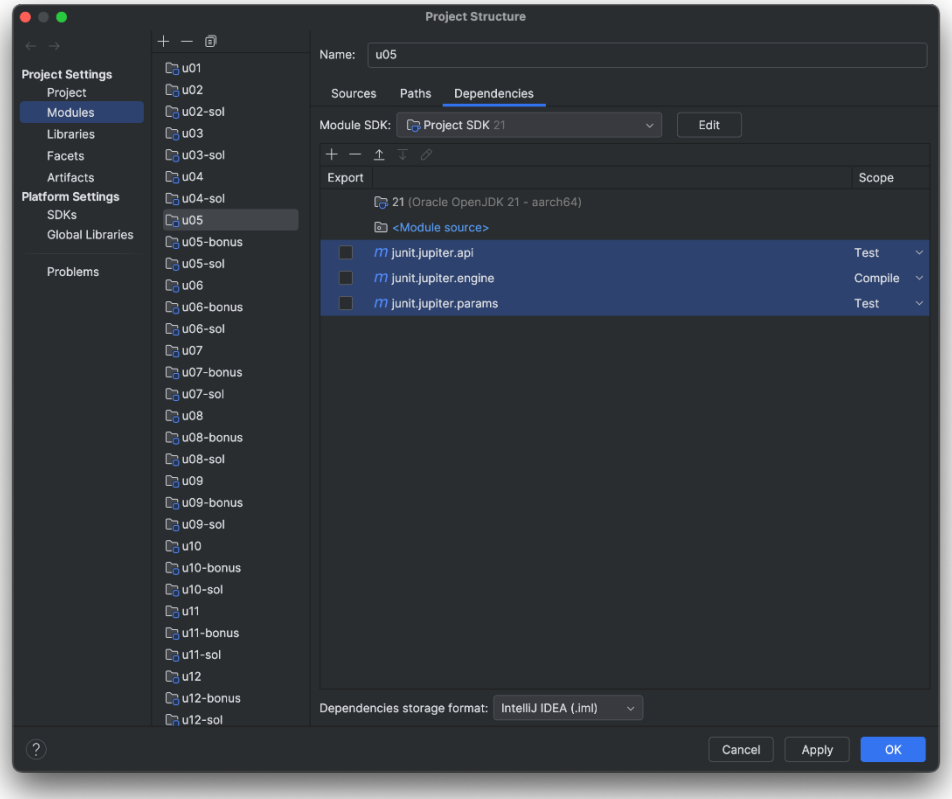
JUnit Tests: JUnit Importieren

- **Wir brauchen:**
  - junit-jupiter-api-5.9.3.jar
  - junit-jupiter-engine.5.9.3.jar
  - junit-jupiter-params.5.9.3.jar
- **Wähle *Add Selected***



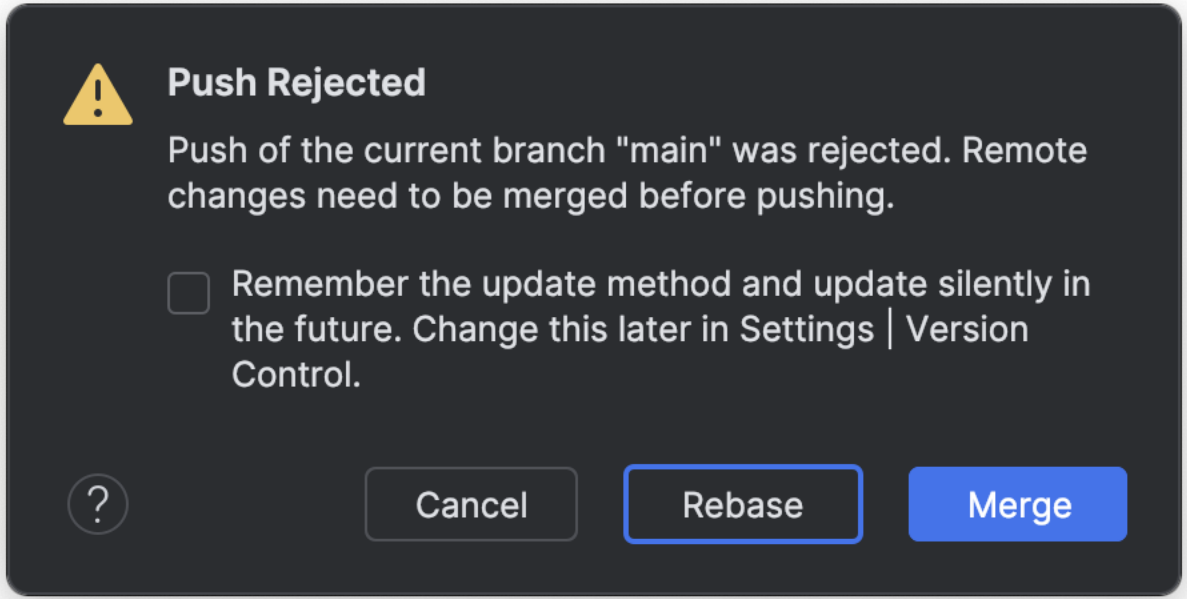
JUnit Tests: JUnit Importieren

- **Wir brauchen:**
  - junit-jupiter-api-5.9.3.jar
  - junit-jupiter-engine.5.9.3.jar
  - junit-jupiter-params.5.9.3.jar



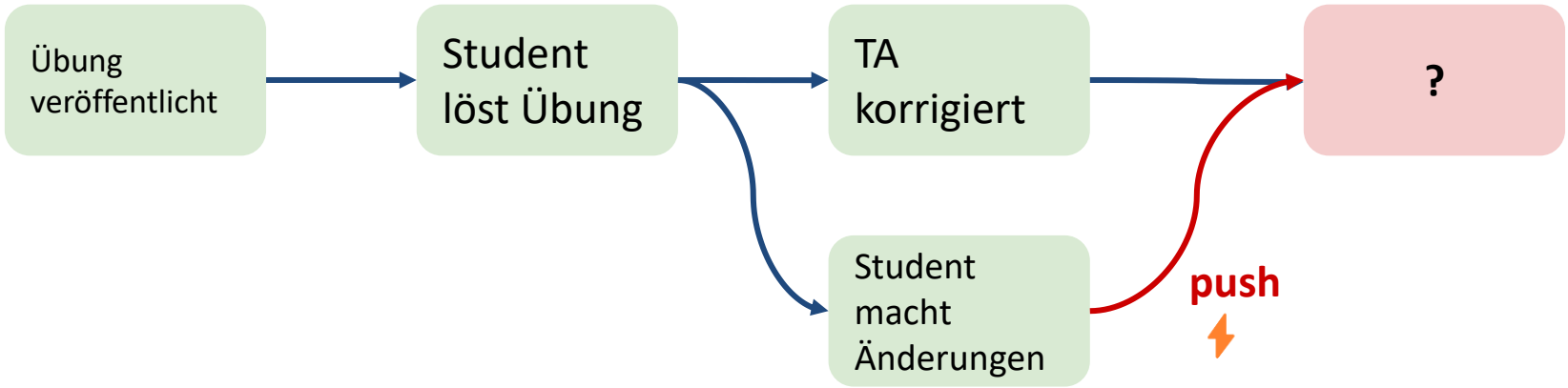
Git Merge Conflicts

Git sagt “Push Rejected”



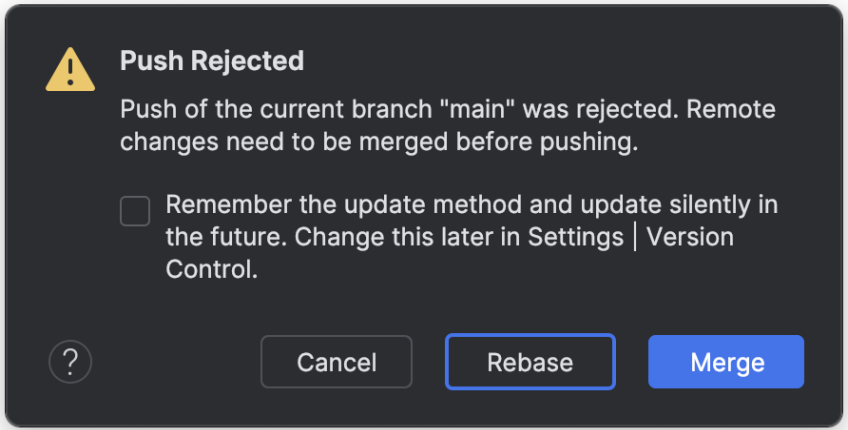
Git - Merge Conflicts

Student löst Übung gleichzeitig während TA korrigiert
→ Beim Versuch zu pushen, Konflikt!



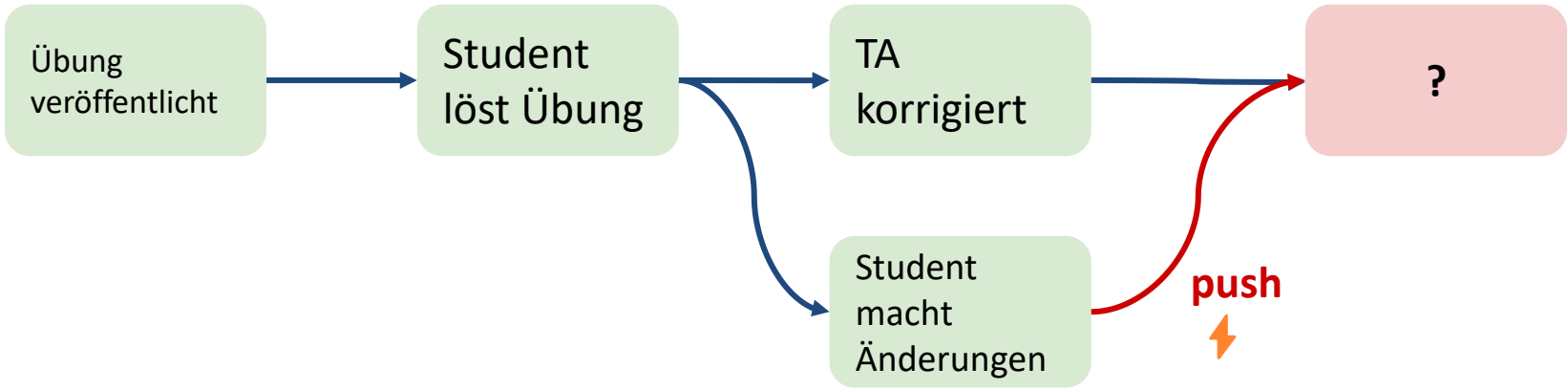
Zwei Optionen

- **Merge:** Kombiniert die zwei unterschiedlichen Versionen der Datei in eine kombinierte Datei.
- **Rebase:** Übernimmt die Änderungen auf dem Server und versucht deine Änderungen anzuwenden.

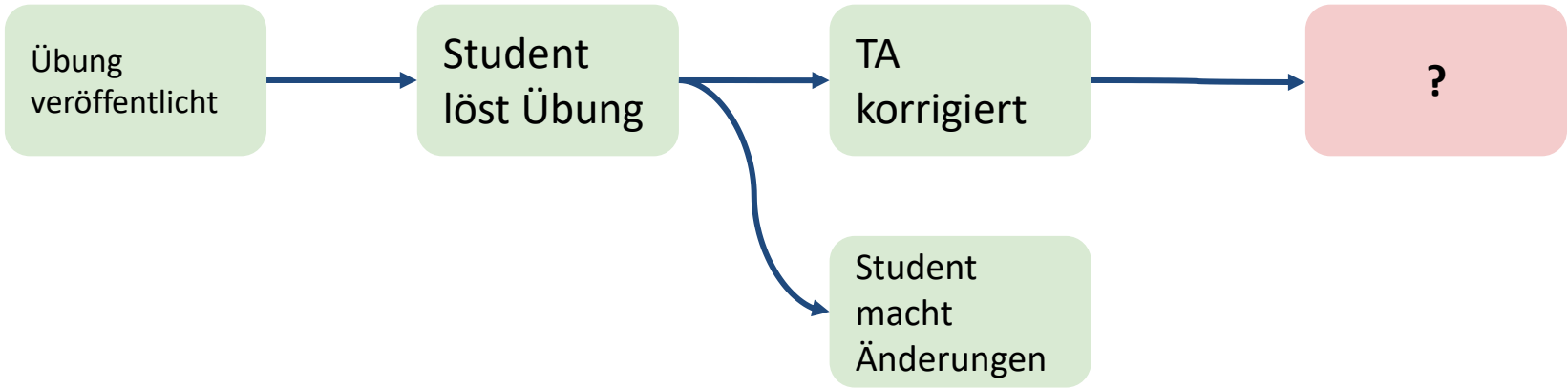


Git Merge

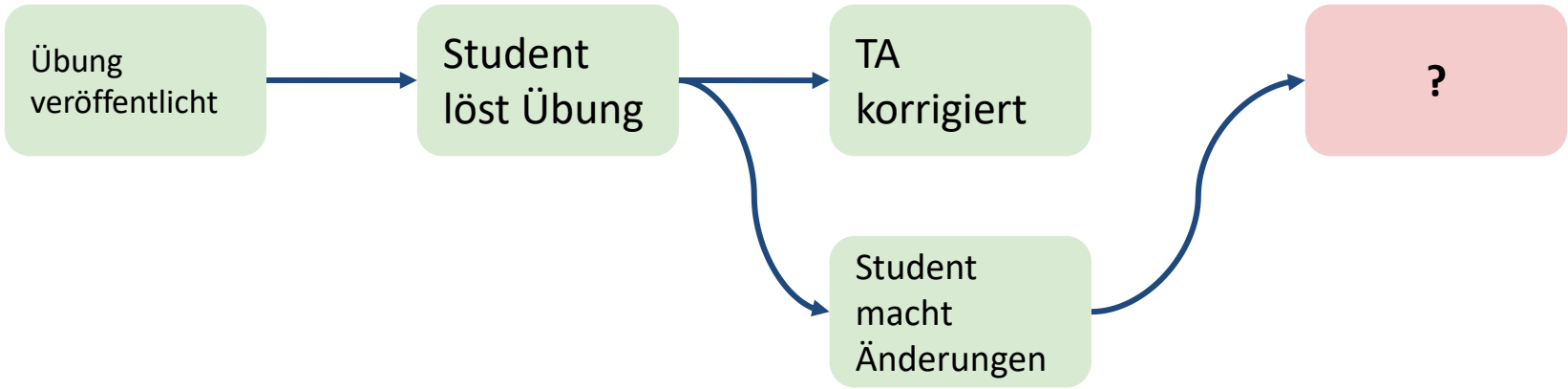
Git – Merge


Übung veröffentlicht
Student löst Übung
TA korrigiert
?
Student macht Änderungen
push

Git – Merge (Accept Theirs)

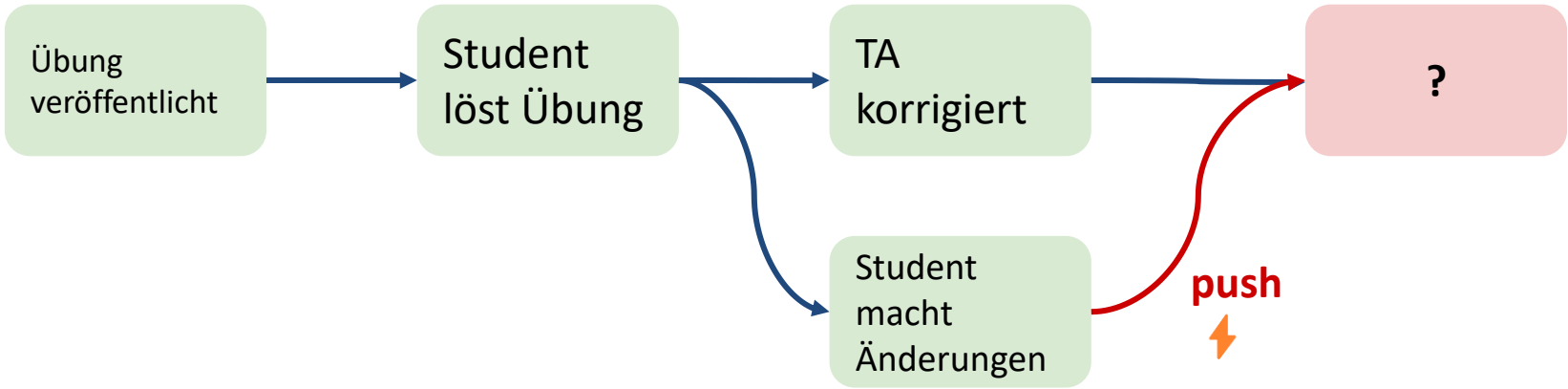


Git – Merge (Accept Yours)

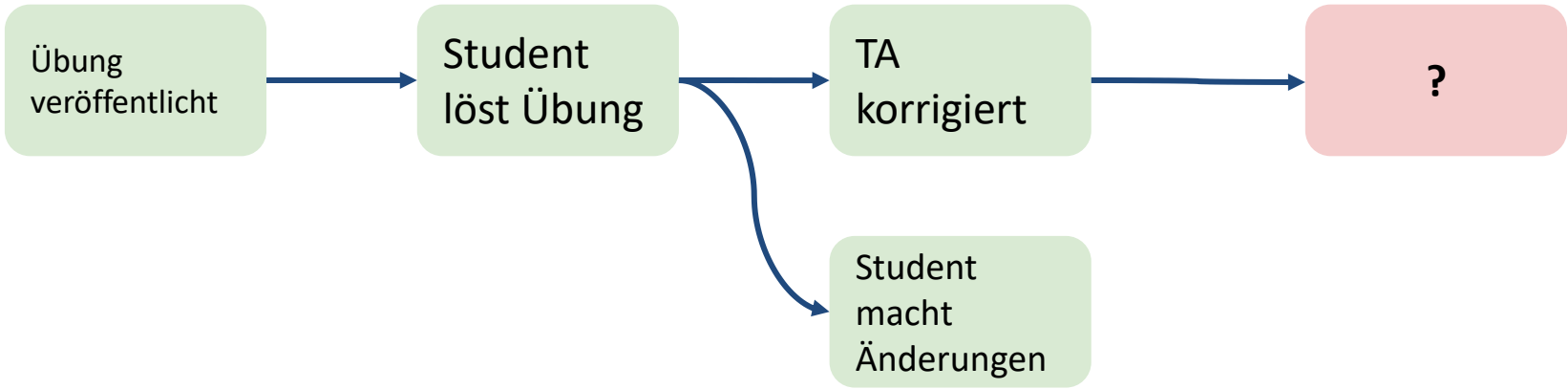


Git Rebase

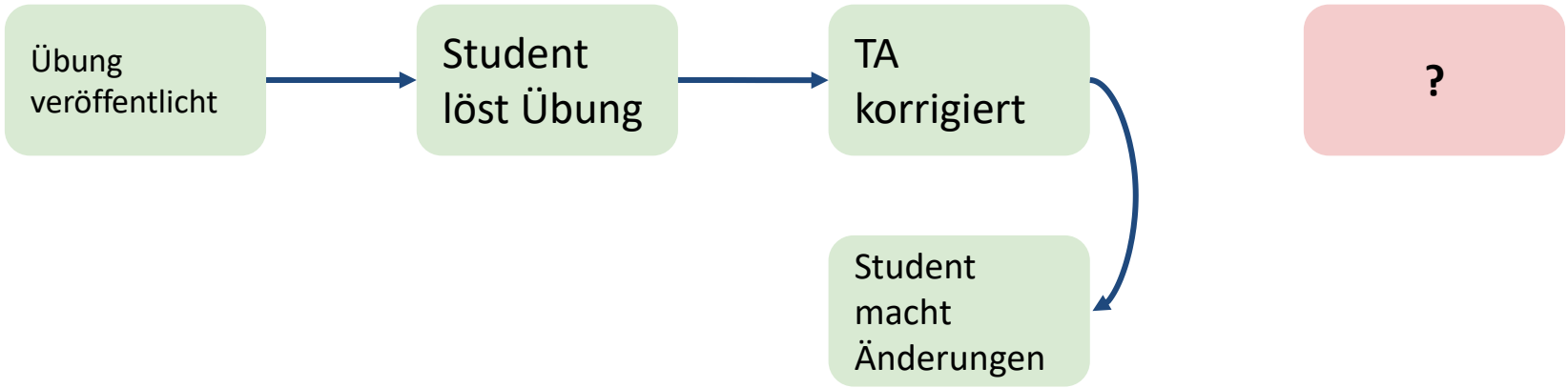
Git – Rebase


Übung veröffentlicht
Student löst Übung
TA korrigiert
?
Student macht Änderungen
push

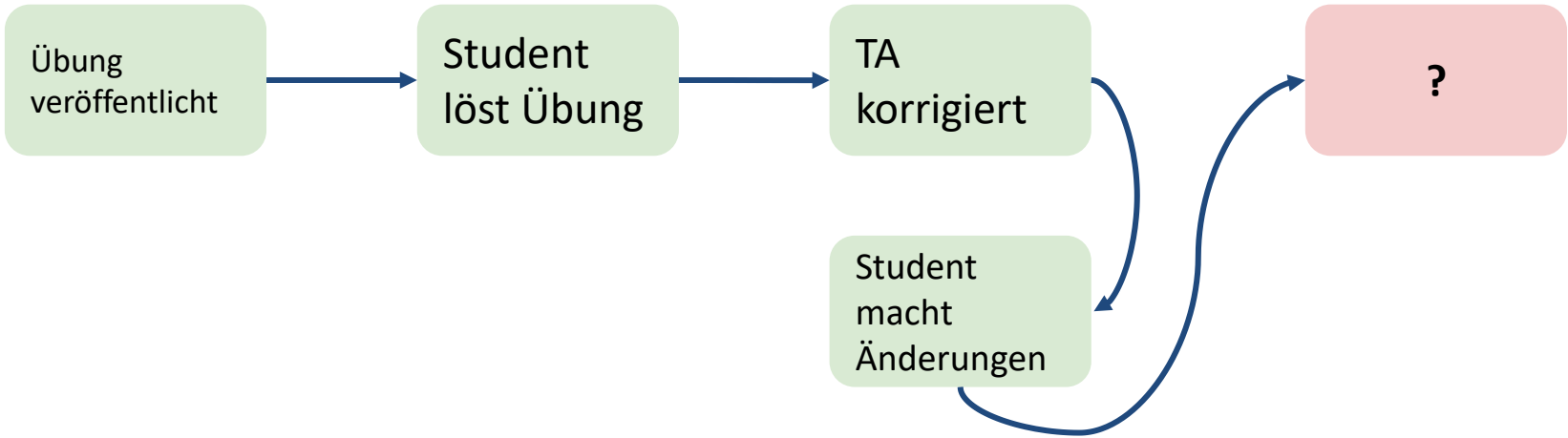
Git – Rebase


Übung veröffentlicht
Student löst Übung
TA korrigiert
?
Student macht Änderungen

Git – Rebase


Übung veröffentlicht
Student löst Übung
TA korrigiert
Student macht Änderungen
?

Git – Rebase


Übung veröffentlicht
Student löst Übung
TA korrigiert
?
Student macht Änderungen

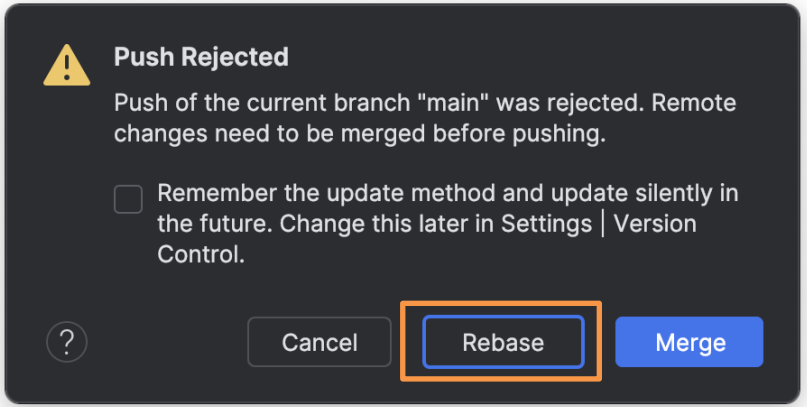
Zwei Situationen

- **TA und Student haben nicht dieselbe(n) Datei(en) bearbeitet.** 🥳

- **TA und Student haben dieselbe(n) Datei(en) bearbeitet.** 😞

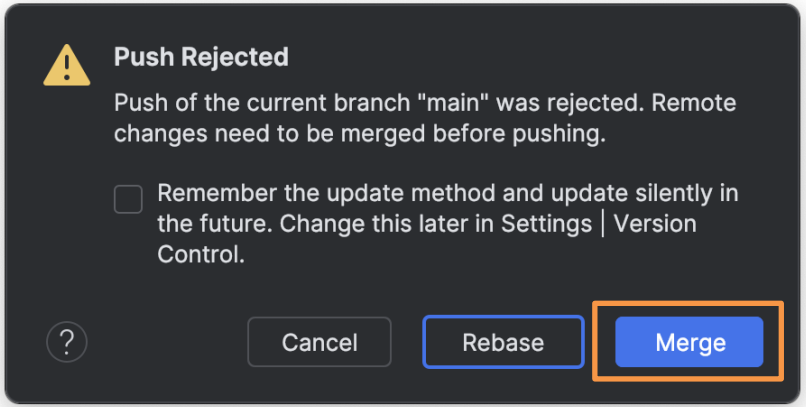
Zwei Situationen

- **TA und Student haben nicht dieselbe(n) Datei(en) bearbeitet.** 🥳
  - ***Rebase*** wählen.
  - Änderungen sind gepushed.



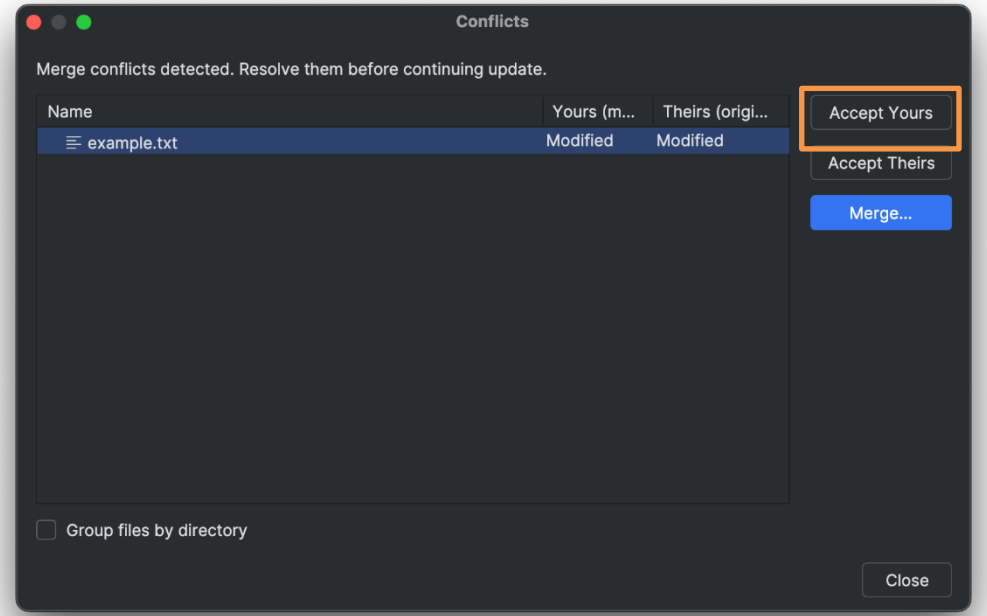
Zwei Situationen

- **TA und Student haben dieselbe(n) Datei(en) bearbeitet.** 😓
  - ***Merge*** wählen.



Zwei Situationen

- **TA und Student haben dieselbe(n) Datei(en) bearbeitet.** 😓
  - ***Merge*** wählen.
  - ***Accept Yours*** wählen.
  - Änderungen gepushed.



Methoden

Methoden: Folgen und Reihen

Schreiben Sie ein Programm "Reihe.java", das eine Zahl N von der Konsole einliest und dann die folgende Summe berechnet:

$$\frac{1}{1^2} + \frac{1}{2^2} + \cdots + \frac{1}{N^2}$$

Beispiel: Für $N = 4$ sollte Ihr Programm ca. `1.42` ausgeben. Wie verhält sich diese Summe für grosse N?

Potenzieren

```
public static void main(String[] args) {
          int n, k;

          Scanner console = new Scanner(System.in);

          System.out.print("Geben Sie Zahl 1 und 2 ein: ");
          n = console.nextInt();
          k = console.nextInt();

          int pot = 1;
          for (int i = 1; i <= k; i++) {
                    pot = pot * n;
          }

          System.out.println(n + " hoch " + k + " = " + pot);
}
```

String

String-Methoden, die String liefern

Name der Methode	Beschreibung der Methode

`substring(i1, i2)`
or
`substring(i1)`
	Ein neuer String:
	Der Substring von `i1` (inklusive) bis `i2` (exklusive);
	falls kein `i2` übergeben wird: bis Ende des Strings

`toLowerCase()`	Ein neuer String mit nur Kleinbuchstaben
`toUpperCase()`	Ein neuer String mit nur Grossbuchstaben
`stripLeading()`	Ein neuer String ohne Leerzeichen am Anfang
`stripTrailing()`	Ein neuer String ohne Leerzeichen am Ende

Substrings

```
String name = "Bob Dylan";
String firstName = name.substring(0, 3); // "Bob"
String lastName = name.substring(4); // "Dylan"
String firstCharacter = name.substring(0, 1); // "B"
name = name.toLowerCase(); // "bob dylan"
```

Indizes 0, 1, 2

Von Index 4 bis Ende

name:	B	o	b		D	y	l	a	n
Index	0	1	2	3	4	5	6	7	8

Achtung: `name.substring(0, 1)` ist nicht das gleiche wie `name.charAt(0)`

- `name.substring(0, 1)` hat Wert `"B"` und ist vom Typ `String`
- `name.charAt(0)` hat Wert `'B'` und ist vom Typ `char`

String-Methoden, die `int` liefern

Name der Methode	Beschreibung der Methode
`length()`	Länge des Strings (Anzahl Zeichen)

`indexOf(s, fromIndex)`
`indexOf(c, fromIndex)`
`indexOf(s)`
`indexOf(c)`
	Index wo String `s` oder Zeichen `c` zum ersten Mal nach Index `fromIndex` im String auftaucht (-1 falls nicht gefunden);
	falls `fromIndex` nicht übergeben wird, von Anfang des Strings

```
String s = "S. Beckett";
int indexBeck = s.indexOf("Beck");          // 3
int eFirstOccurrence = s.indexOf('e');      // 4
int eSecondOccurrence = s.indexOf('e', 5); // 7
```

String-Methoden, die boolean liefern

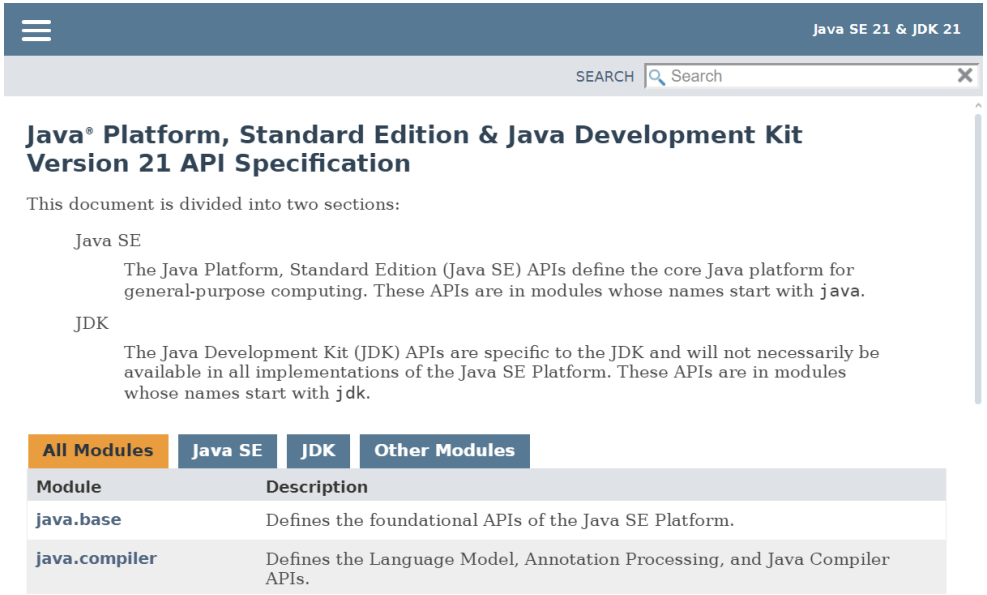
Name der Methode	**Beschreibung der Methode** bei Aufruf `s1.method(s2)`
`equals(s2)`	ob `s1` und `s2` die gleichen Buchstaben enthalten
`equalsIgnoreCase(s2)`	ob `s1` und `s2` die gleichen Buchstaben enthalten, ohne Berücksichtigung von Gross- und Kleinschreibung
`startsWith(s2)`	ob `s1` mit den Buchstaben von `s2` anfängt
`endsWith(s2)`	ob `s1` mit den Buchstaben von `s2` endet
`contains(s2)`	ob `s1` irgendwo `s2` enthält

Hinweis zu == und `equals`

- `==` nur für Basistypen (z.B. `int` oder `char`), nicht für Strings
- `==` für Strings wird später erklärt!

Wo findet man diese Informationen?

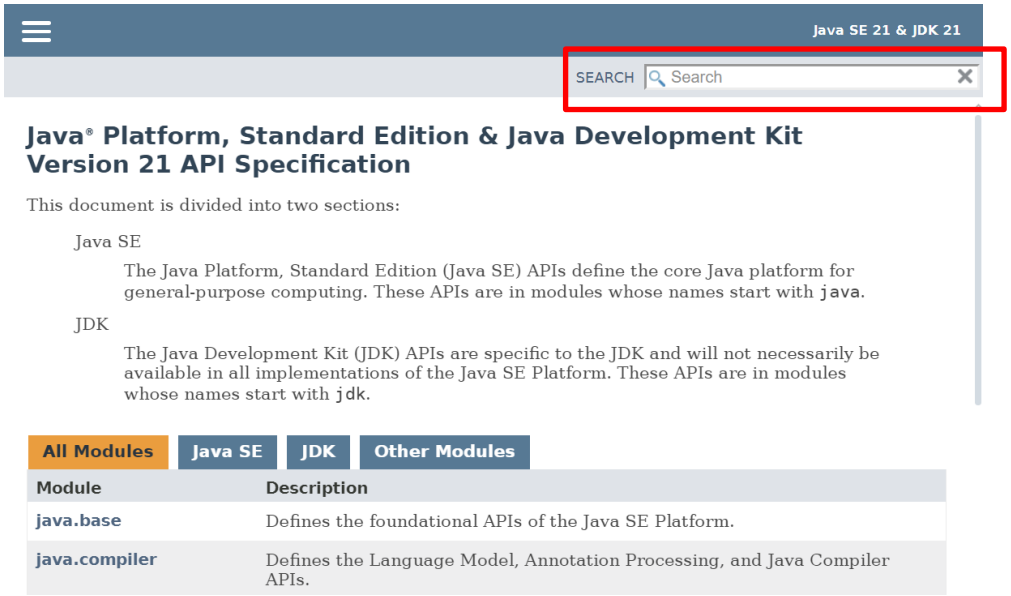
- **In der Java Documentation 21**



https://docs.oracle.com/en/java/javase/21/docs/api/

Wo findet man diese Informationen?

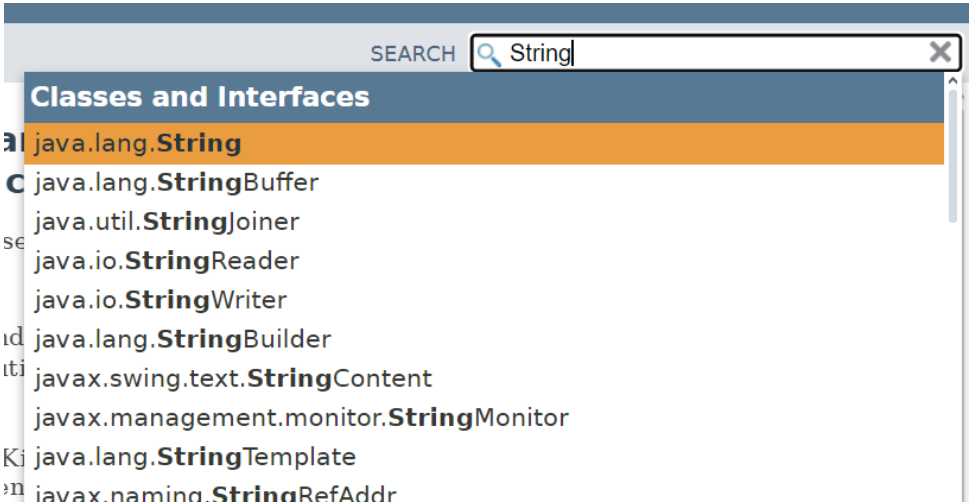
- **In der Java Documentation 21**



https://docs.oracle.com/en/java/javase/21/docs/api/

Wo findet man diese Informationen?

- **In der Java Documentation 21**

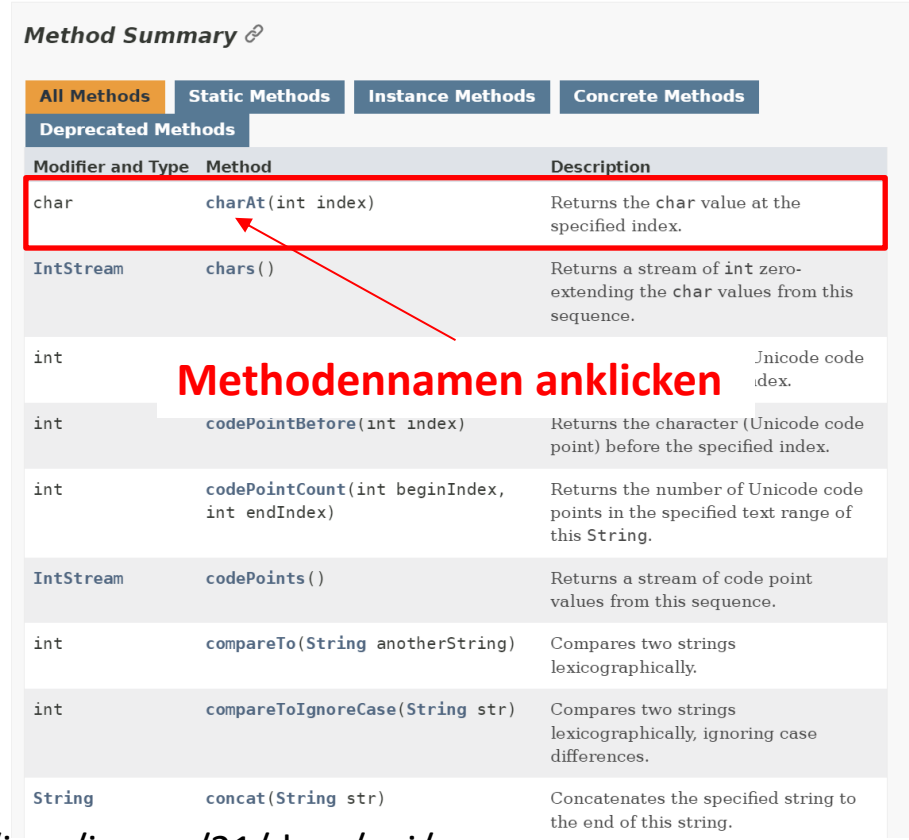


https://docs.oracle.com/en/java/javase/21/docs/api/

Wo findet man diese Informationen?

- In der Java Documentation 21

https://docs.oracle.com/en/java/javase/21/docs/api/



Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods | Deprecated Methods

Modifier and Type	Method	Description
char	charAt(int index)	Returns the char value at the specified index.
IntStream	chars()	Returns a stream of int zero-extending the char values from this sequence.
int		Unicode code index.
int	codePointBefore(int index)	Returns the character (Unicode code point) before the specified index.
int	codePointCount(int beginIndex, int endIndex)	Returns the number of Unicode code points in the specified text range of this String.
IntStream	codePoints()	Returns a stream of code point values from this sequence.
int	compareTo(String anotherString)	Compares two strings lexicographically.
int	compareToIgnoreCase(String str)	Compares two strings lexicographically, ignoring case differences.
String	concat(String str)	Concatenates the specified string to the end of this string.

Methodennamen anklicken

Wo findet man diese Informationen?

- **In der Java Documentation 21**

charAt

```
public char charAt(int index)
```

Returns the `char` value at the specified index. An index ranges from `0` to `length() - 1`. The first `char` value of the sequence is at index `0`, the next at index `1`, and so on, as for array indexing.

If the `char` value specified by the index is a surrogate, the surrogate value is returned.

Specified by:

`charAt` in interface `CharSequence`

Parameters:

`index` - the index of the `char` value.

Returns:

the `char` value at the specified index of this string. The first `char` value is at index `0`.

Throws:

`IndexOutOfBoundsException` - if the `index` argument is negative or not less than the length of this string.

https://docs.oracle.com/en/java/javase/21/docs/api/

Escaping in Strings

- Strings in Java sind gekennzeichnet durch “” (reservierte Symbole)
- Was machen wir, wenn wir ein reserviertes Symbol im String verwenden wollen?
- Wir können es mit \ (ebenfalls ein reserviertes Symbol) “escapen”

"\"\\\\\"\""

```
"\"\\\\\"\""
```

Aussengrenzen des Strings
Escape-Charakter
Im Resultat sichtbar

Aussengrenzen des Strings
Escape-Charakter
Im Resultat sichtbar

Aussengrenzen des Strings
Escape-Charakter
Im Resultat sichtbar

Aussengrenzen des Strings
Escape-Charakter
Im Resultat sichtbar

Aussengrenzen des Strings
Escape-Charakter
Im Resultat sichtbar

"\\""

Aussengrenzen des Strings
Escape-Charakter
Im Resultat sichtbar

String Beispiele

```java
public class MyClass{
    public static void main(String[] args){
        String str = "EPROG2024";
        int result = str.indexOf('2') + str.length() * 2 - 10;
    }
}
```

Output: 13

String Beispiele

```java
public class MyClass{
    public static void main(String[] args){
        String str = "Einfuehrung";
        String result = str.substring(3, 8).toUpperCase() + str.charAt(str.length() - 1);
    }
}
```

Output: FUEHRg

String Beispiele

```java
public class MyClass{
    public static void main(String[] args){
        String str = "Programmierung";
        String result = str.replace('r', '*').substring(5, 12) + str.charAt(0);
    }
}
```

Output: ammie*uP

String Beispiele

```java
public class MyClass{
    public static void main(String[] args){
        String str1 = "Java21";
        String str2 = "EPROG2024";
        String result = str1.substring(0, 2) + str2.substring(str2.length() - 4);
    }
}
```

Output: Ja2024

String Beispiele

```java
public class MyClass{
    public static void main(String[] args){
        String str = "Substring";
        int result = str.indexOf('S') + str.lastIndexOf('g') - str.length();
    }
}
```

Output: -1

String Beispiele

```java
public class MyClass{
    public static void main(String[] args){
        String str = "Einfuehrung";
        String result = str.substring(2, 6).replace('f', 'X').concat(str.substring(6, 9));
    }
}
```

Output: nXuehru

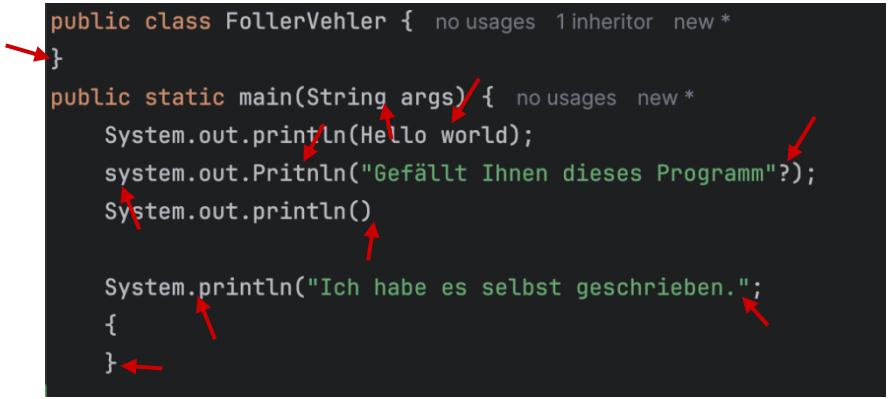
String Beispiele

```java
public class MyClass{
    public static void main(String[] args){
        String str = "Programmierung";
        String result = str.charAt(0) + str.substring(2, 5).toLowerCase() + str.charAt(str.length() - 1);
    }
}
```

Output: Pogrg

Nachbesprechung

Aufgabe 1: Fehlerbehebung



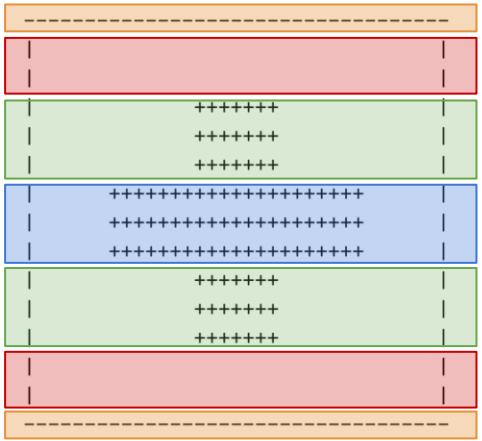
```
public class FollerVehler {  no usages  1 inheritor  new *
}
public static main(String args) {  no usages  new *
    System.out.println(Hello world);
    system.out.Pritnln("Gefällt Ihnen dieses Programm"?);
    System.out.println()

    System.println("Ich habe es selbst geschrieben.";
    {
    }
```

```java
public class OhneFehler {
    public static void main(String[] args) {
        System.out.println("Hello world");
        System.out.println("Gefällt Ihnen dieses Programm?");
        System.out.println();

        System.out.println("Ich habe es selbst geschrieben.");
    }
}
```

Aufgabe 2: Schweizerfahne (Konsole)



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!
   ```

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 `<digitsum>`, welche als legale Symbole genau jene natürlichen Zahlen zulässt, deren Quersumme eine gerade Zahl ist.

3. Erstellen Sie eine Beschreibung `<xyz>`, 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 `<term>`, 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.

Vorbesprechung

Meine Aufgaben Ratings:

- **Best Case: Ihr macht alle Aufgaben (, wenn die Zeit reicht)**
- **Falls nicht: Ich mache Ratings zur Wichtigkeit der einzelnen Aufgaben**

- **(Keine offizielle Empfehlung, meine subjektive Meinung auf Basis meiner eigenen Erfahrung als Student in diesem Kurs)**
- **! Sagen nichts über die Schwierigkeit der Aufgaben aus !**

Wichtig für die Prüfung:
(Prüfungs- oder prüfungsähnliche Aufgaben)

Wichtig für euer Verständnis:
(Aber nicht in Prüfungsform)

Wichtig für tiefes Verständnis/ Zusatzaufgaben:
(Bspw. viele vom gleichen Typ)

Webseite

- **timostucki.com**

Aufgabe 1: Binärdarstellung

Schreiben Sie ein Programm "Binaer.java", das eine positive Zahl Z einliest und dann die Binärdarstellung druckt. (Hinweis: Finden Sie zuerst die grösste Zahl k, so dass die Zweierpotenz $K = 2^k$ kleiner als Z ist.)

Beispiel: Für $Z = 14$ sollte Ihr Programm `1110` ausgeben.

Aufgabe 2: Grösster gemeinsamer Teiler

Schreiben Sie ein Programm "GGT.java", das den grössten gemeinsamen Teiler (ggT) zweier ganzer Zahlen mithilfe des Euklidischen Algorithmus berechnet. Hierbei handelt es sich um eine iterative Berechnung, die auf folgender Beobachtung basiert:

1. Wenn x grösser oder gleich y ist und sich x durch y teilen lässt, dann ist der ggT von x und y gleich y;
2. andernfalls ist der ggT von x und y der gleiche wie der ggT von y und $x \,\%\, y$.

Üben Sie diesen Algorithmus zuerst von Hand an ein paar Beispielen und schreiben Sie dann das Java Programm.

Beispiel: Für $x = 36$ und $y = 44$:

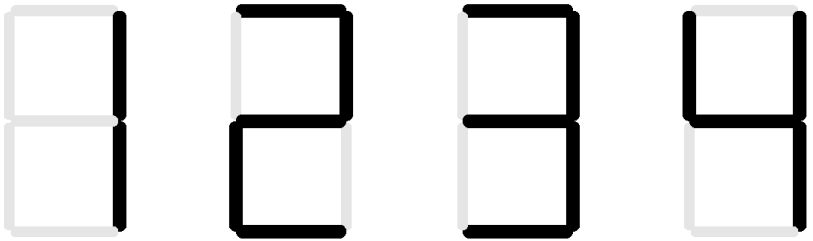
$$ggT(\underset{x}{36}, \underset{y}{44}) \overset{2.}{=} ggT(44, \underbrace{36 \,\%\, 44}_{36}) \overset{2.}{=} ggT(36, \underbrace{44 \,\%\, 36}_{8}) \overset{2.}{=} ggT(8, \underbrace{36 \,\%\, 8}_{4}) \overset{1.}{=} 4$$

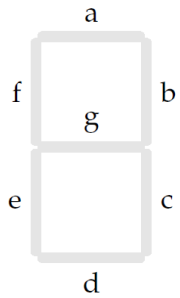
Aufgabe 3: Zahlenerkennung

Für diese Aufgabe verwenden wir einen String um die erleuchtenden Segmente einer Sieben-segmentanzeige zu kodieren. Die Segmente sind, wie im Bild gezeigt, von a bis g nummeriert. Die Kodierung einer möglichen Anzeige ist ein String, in welchem der Buchstabe 'x' genau dann vorkommt, wenn das 'x'te Segment der Anzeige erleuchtet ist. Zum Beispiel wird die Zahl 2 kodiert durch 'abged'. Zur Einfachheit darf angenommen werden, dass kein Buchstabe mehr als einmal in der Kodierung vorkommt und dass nur die Zahlen 0 bis 9 kodiert werden.

Schreiben Sie ein Programm "Zahlen.java", das einen String, der eine Anzeige kodiert, einliest und die kodierte Zahl als Integer ausgibt. Überlegen Sie wie viele IF Blöcke benötigt werden um jede Zahl zu erkennen.

Tipp: Sie können `str.contains("a")` verwenden, um zu überprüfen, ob ein String `str` den Buchstaben 'a' enthält.





Aufgabe 4: Berechnungen

2. Implementieren sie die Methode `Calculations.magic7(int a, int b)`. Die Methode gibt einen Boolean zurück. Die Methode soll `true` zurückgeben, wenn einer der Parameter 7 ist oder wenn die Summe oder Differenz der Parameter 7 ist. Ansonsten soll die Methode `false` zurückgeben.

   Beispiele

   - `magic7(2,5)` gibt `true` zurück.
   - `magic7(7,9)` gibt `true` zurück.
   - `magic7(5,6)` gibt `false` zurück.

   Hinweis: Mit der Funktion `Math.abs(num)` können Sie den absoluten Wert einer Zahl `num` erhalten.

Aufgabe 4: Berechnungen

3. Implementieren Sie die Methode `Calculations.fast12(int z)`. Das Argument `z` ist nicht negativ. Die Methode gibt einen Boolean zurück. Die Methode soll `true` zurückgeben, wenn `z` nahe an einem Vielfachen von `12` ist. Eine Zahl `x` ist nahe an einer Zahl `y`, wenn eine der Zahlen um maximal `2` grösser oder kleiner ist als die andere Zahl. Ansonsten soll die Method `false` zurückgeben.

   - `fast12(12)` gibt `true` zurück.
   - `fast12(14)` gibt `true` zurück.
   - `fast12(10)` gibt `true` zurück.
   - `fast12(15)` gibt `false` zurück.

Aufgabe 5: Scrabble

In dieser Aufgabe sollen Sie Scrabble-Steine legen, mittels ASCII-Art auf der Konsole. Vervollständigen Sie die Methode `drawNameSquare` in der Klasse `Scrabble`. Diese Methode nimmt einen Namen als String-Parameter und soll den Namen als in einem Quadrat angeordnete Scrabble-Steine auf der Konsole (`System.out`) ausgeben. Wenn z.B. der String `Alfred` übergeben wird, sollte folgendes Bild ausgegeben werden:

```
+---+---+---+---+---+---+
| A | L | F | R | E | D |
+---+---+---+---+---+---+
| L |                   | E |
+---+                   +---+
| F |                   | R |
+---+                   +---+
| R |                   | F |
+---+                   +---+
| E |                   | L |
+---+---+---+---+---+---+---+
| D | E | R | F | L | A |
+---+---+---+---+---+---+
```

(Alte Prüfungsaufgabe: https://exams.vis.ethz.ch/exams/w0cikcqb.pdf)

(Achtung: Auch 1 Sterne Prüfungsaufgaben sind mittlerweile schwieriger geworden)

Aufgabe 5: Scrabble

Ihr Code braucht nur Namen der Länge 3 oder länger unterstützen. Für einen Namen mit Länge 3, z.B. `Jim`, sollte die Ausgabe so aussehen:

```
+---+---+---+
| J | I | M |
+---+---+---+
| I |   | I |
+---+---+---+
| M | I | J |
+---+---+---+
```

Beachten Sie, dass Ihr Programm keinerlei andere Ausgabe als das Scrabble-Quadrat machen darf.