

Übungsstunde 4

Einführung in die Programmierung

Probleme bei Übung 2

Probleme bei Übung 2

Absolutes Maximum

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.

Postconditions

Aufgabenstellung genau durchlesen!

Nachbesprechung Übung 3

Aufgabe 1: Folgen und Reihen

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

```
import java.util.Scanner;

public class Reihe {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Geben Sie eine natürliche Zahl ein: ");
        int n = scanner.nextInt();
        if(n < 0) {
            System.out.println("Keine natürliche Zahl!");
        }
        else {
            double sum = 0;
            for(int i = 1; i <= n; i++) {
                sum += 1.0 / (i * i);
            }
            System.out.println(sum);
        }
    }
}
```

Wichtig: Überprüfung der
Benutzereingabe

Achtung: Double-Division
erzwingen

Aufgabe 2: Binärdarstellung

Hat jemand einen Vorschlag?

Aufgabe 2: Binärdarstellung

```
// Finde twoToTheK = 2^k <= z
```

```
int k = 0;
```

```
int twoToTheK = 1;
```

```
while(z >= twoToTheK) {
```

```
    k++;
```

```
    twoToTheK *= 2;
```

```
}
```

```
k--;
```

```
twoToTheK /= 2;
```

```
// Drucke einzelne Ziffern der Binaerdarstellung von z
```

```
while(k >= 0) {
```

```
    int digit = z / twoToTheK;
```

```
    System.out.print(digit);
```

```
    z -= digit * twoToTheK;
```

```
    k--;
```

```
    twoToTheK /= 2;
```

```
}
```

```
System.out.println();
```

ohne zusätzlichen
Speicher (Array)

Aufgabe 3: 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 als y ist, dann ist — sofern sich x durch y teilen lässt — 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 $x \% y$.

```
while(x <= y || x % y != 0) {  
    // Zwischenspeichern von y  
    int altY = y;  
    y = x % y;  
    x = altY;  
}  
System.out.println(y);
```

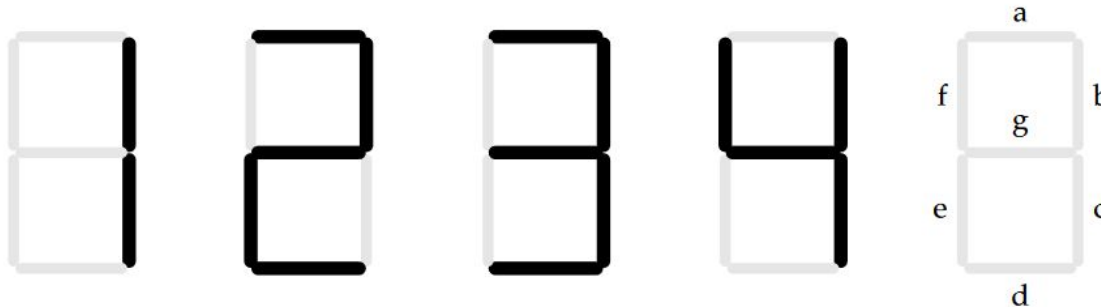
Negation der Bedingung in 1.

Aufgabe 4: Zahlenerkennung

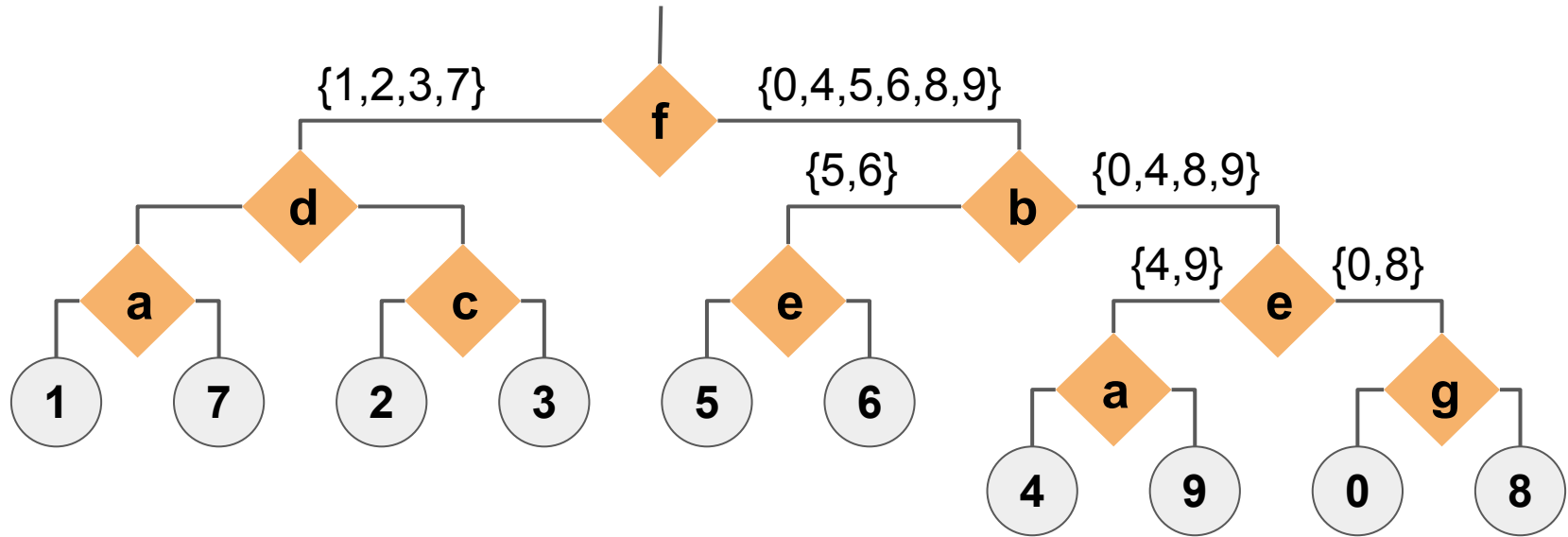
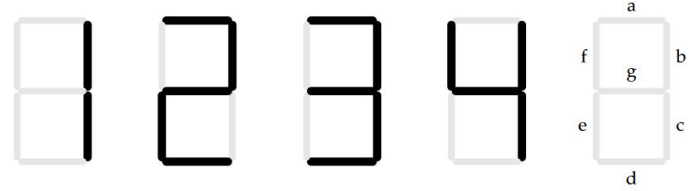
Für diese Aufgabe verwenden wir einen String um die erleuchtenden Segmente einer **Sieben-segmentanzeige** zu kodieren. Die Segmente sind, wie im Bild gezeigt, von a bis g nummeriert. Die Kodierung einer möglichen Anzeige ist ein String, in welchem der Buchstabe 'x' genau dann vorkommt, wenn das 'x'te 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: Zahlenerkennung



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 |
+---+---+---+---+---+
```

Aufgabe 6: Weakest Precondition

Bitte geben Sie für die folgenden Programmsegmente die schwächste Vorbedingung (weakest precondition) an. Bitte verwenden Sie Java Syntax (die Klammern { und } können Sie weglassen). Alle Anweisungen sind Teil einer Java Methode. Alle Variablen sind vom Typ int und es gibt keinen Overflow.

1.
P: { $n > 7/4$ } oder { $n \geq 2$ } oder { $n > 1.75$ }

S: $m = n * 4;$ $k = m - 2;$

Q: { $n > 0 \ \&\& \ k > 5$ }

2.
P: { $n \neq 0$ }

S: $m = n * n;$ $k = m * 2;$

Q: { $k > 0$ }

3.
P: { $y > x \ || \ x > y$ } oder { $x \neq y$ }

S: if ($x > y$) {

$z = x - y;$

 } else {

$z = y - x;$

 }

Q: { $z > 0$ }

Aufgabe 7: Berechnungen

In dieser Aufgabe implementieren Sie verschiedene Berechnungen.

1. Implementieren Sie die Methode `Calculations.checksum(int x)`, das heisst die Methode `checksum` in der Klasse `Calculations`. Die Methode nimmt einen Integer `x` als Argument, welcher einen nicht-negativen Wert hat. Die Methode soll die Quersumme von `x` zurückgeben. Sie sollen für diese Aufgabe **keine** Schleife verwenden.

Beispiele

- `checksum(258)` gibt 15 zurück.
- `checksum(49)` gibt 13 zurück.
- `checksum(12)` gibt 3 zurück.

```
public static int checksum(int x) {  
    if (x < 10) {  
        return x;  
    } else {  
        return x%10 + checksum(x/10);  
    }  
}
```

Hinweis: Für einen Integer `a` ist `a % 10` die letzte Ziffer und `a / 10` entfernt die letzte Ziffer. Zum Beispiel `258 % 10` ist 8 und `258 / 10` ist 25.

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.

```
public static boolean magic7(int a, int b) {  
    return a == 7 || b == 7 || a+b == 7 || a-b == 7 || b-a == 7;  
}
```

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

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 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 Methode `false` zurückgeben.

Beispiele

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

```
public static boolean fast12(int z) {  
    return (z + 2)%12 <= 4;  
}
```

Theorie Recap

Rekursion

Invariante

Was wollen wir?

Gegeben ein Loop

Wir wollen eine Aussage über den Zustand unseres Programms machen

Zwischen Pre- und Postcondition ist ein Loop

Aussage sollte ganze Zeit dazwischen gültig sein

Hoare Logik

- Gegeben sei ein Tripel für einen while-loop

$$\{P\} \text{ while}(B) \text{ S}; \{Q\}$$

Ein solches Tripel ist gültig wenn es eine Invariante I gibt so dass:

- $P \Rightarrow I$ Invariante gilt zu Beginn
- $\{I \wedge B\} S \{I\}$ Nach Ausführen des Rumpfes gilt die Invariante wieder
- $(I \wedge \neg B) \Rightarrow Q$ Invariante (und Verlassen der Schleife, d.h. test B ist false) impliziert Postcondition Q .

- Für ein gültiges Hoare Tripel einer Schleife

$$\{P\} \text{ while}(B) \ S; \{Q\}$$

sind Schleifentest B , Schleifenrumpf S und die Schleifeninvariante I aufeinander abgestimmt

- S kann eine Folge von Anweisungen sein (auch geschachtelte Loops)
- Für eine Postcondition Q gibt es (oft, manchmal) verschiedene Schleifen, die für die Precondition P das selbe Result berechnen
 - Diese Schleifen haben dann andere Invarianten und Statements im Rumpf
- Definition allgemein genug, deckt auch den Fall ab dass der Rumpf keinmal durchlaufen wird

Arrays

Array Initialisierung

type[] name = new type[length];

- **Beispiel:**

`int[] numbers = new int[10];`

oder

`int[] numbers = {1,2,3};`

und

`int[][] matrix = new int[10][];`

- Jedes Element wird auf einen Wert der Null «entspricht» gesetzt
 - Voreinstellung («default»)

Type	Default Wert
int	0
boolean	false
String	null

Objects

null

<i>Index</i>	0	1	2	3	4	5	6	7	8	9
<i>Wert</i>	0	0	0	0	0	0	0	0	0	0

numbers



```
boolean[] results = new boolean[5];  
results[2] = true;  
results[4] = true;
```

<i>Index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>
<i>Wert</i>	false	false	true	false	true

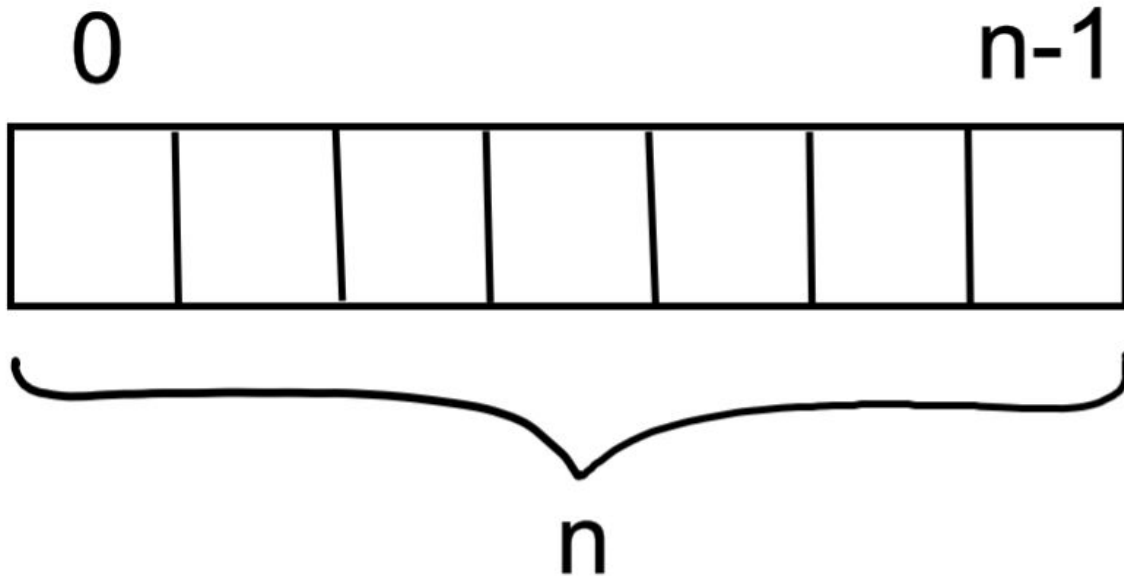
```
int[][] matrix = new int[10][10];  
matrix[0][0] = 1;
```


- Lesen oder Schreiben (Zugriff, «access») mit einem Index ausserhalb dieses Bereichs resultiert in einer `ArrayIndexOutOfBoundsException`
- «Out-of-bounds» Fehler

...	-3	-2	-1	Index	0	1	2	3	4	5	6	7	8	9	10	11	12	...
				Wert	0	0	0	0	0	0	0	0	0	0				

Array length

Länge eines Arrays: `array.length`; (ohne Klammer `()` !)

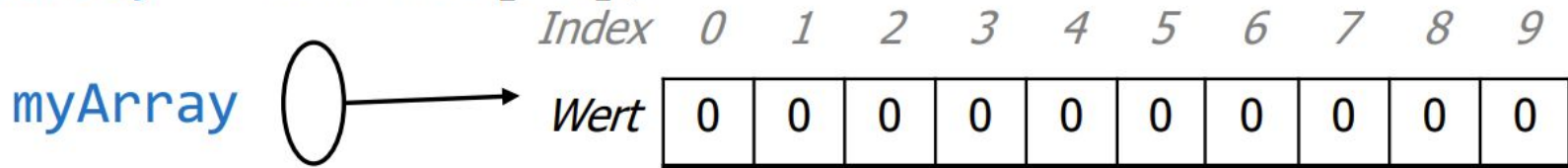


primitive variables vs reference type variables

Eine Referenzvariable *erlaubt den Zugriff* auf einen Array (ein Objekt)

- Entweder neu erstellt (mit Deklaration)

```
int[] myArray = new int[10];
```



Array Aufgaben

Parameter: Basistyp

```
public static void main (String[] args) {  
    int j = 3;  
    int k = plusOne(j);  
    System.out.println(j);  
    System.out.println(k);  
}
```

```
public static int plusOne(int k) {  
    k = k + 1;  
    return k;  
}
```

Parameter: Basistyp

```
public static void main (String[] args) {  
    int j = 3;  
    int k = plusOne(j);  
    System.out.println(j);  
    System.out.println(k);  
}
```

```
public static int plusOne(int k) {  
    k = k + 1;  
    return k;  
}
```

Output:

3
4

Parameter: Element eines Basistyps

```
public static void main (String[] args) {  
    int[] x = {2, 4, 6, 8};  
    int m = plusOne(x[0]);  
    System.out.println(Arrays.toString(x));  
    System.out.println(m);  
}
```

```
public static int plusOne(int k) {  
    k = k + 1;  
    return k;  
}
```

Parameter: Element eines Basistyps

```
public static void main (String[] args) {  
    int[] x = {2, 4, 6, 8};  
    int m = plusOne(x[0]);  
    System.out.println(Arrays.toString(x));  
    System.out.println(m);  
}
```

```
public static int plusOne(int k) {  
    k = k + 1;  
    return k;  
}
```

Output:

```
[2, 4, 6, 8]  
3
```


Parameter: Array

```
public static void main (String[] args) {  
    int[] x = {2, 4, 6, 8};  
    plusOneA(x);  
    System.out.println(Arrays.toString(x));  
}
```

```
public static void plusOneA(int[] y) {  
    for (int k=0; k<y.length; k++) {  
        y[k]++;  
    }  
}
```

Parameter: Array

```
public static void main (String[] args) {  
    int[] x = {2, 4, 6, 8};  
    plusOneA(x);  
    System.out.println(Arrays.toString(x));  
}
```

```
public static void plusOneA(int[] y) {  
    for (int k=0; k<y.length; k++) {  
        y[k]++;  
    }  
}
```

Output:

[3, 5, 7, 9]

Parameter: Array mit alias

```
public static void main (String[] args) {  
    int[] a = {2, 4, 6, 8};  
    int[] b = a;  
    plusOneA(a);  
    System.out.println(Arrays.toString(a));  
    System.out.println(Arrays.toString(b));  
}  
  
public static void plusOneA(int[] y) {  
    for (int k=0; k<y.length; k++) {  
        y[k]++;  
    }  
}
```

Parameter: Array mit alias

```
public static void main (String[] args) {  
    int[] a = {2, 4, 6, 8};  
    int[] b = a;  
    plusOneA(a);  
    System.out.println(Arrays.toString(a));  
    System.out.println(Arrays.toString(b));  
}  
  
public static void plusOneA(int[] y) {  
    for (int k=0; k<y.length; k++) {  
        y[k]++;  
    }  
}
```

Output:

```
[3, 5, 7, 9]  
[3, 5, 7, 9]
```

Parameter: Array (und es gibt Array mit gleichen Elementen)

```
public static void main (String[] args) {  
    int[] c = {2, 4, 6, 8};  
    int[] d = {2, 4, 6, 8};  
    plusOneA(c);  
    System.out.println(Arrays.toString(c));  
    System.out.println(Arrays.toString(d));  
}  
  
public static void plusOneA(int[] y) {  
    for (int k=0; k<y.length; k++) {  
        y[k]++;  
    }  
}
```

Parameter: Array (und es gibt Array mit gleichen Elementen)

```
public static void main (String[] args) {  
    int[] c = {2, 4, 6, 8};  
    int[] d = {2, 4, 6, 8};  
    plusOneA(c);  
    System.out.println(Arrays.toString(c));  
    System.out.println(Arrays.toString(d));  
}  
  
public static void plusOneA(int[] y) {  
    for (int k=0; k<y.length; k++) {  
        y[k]++;  
    }  
}
```

Output:

```
[3, 5, 7, 9]  
[2, 4, 6, 8]
```

Parameter: Array (umständlich)

```
public static void main (String[] args) {  
    int[] f = {2, 4, 6, 8};  
    plusOneArrTemp(f);  
    System.out.println(Arrays.toString(f));  
}  
  
public static void plusOneArrTemp(int[] y) {  
    int[] t = y;  
    for (int k=0; k<t.length; k++) {  
        t[k]++;  
    }  
}
```

Parameter: Array (umständlich)

```
public static void main (String[] args) {  
    int[] f = {2, 4, 6, 8};  
    plusOneArrTemp(f);  
    System.out.println(Arrays.toString(f));  
}
```

```
public static void plusOneArrTemp(int[] y) {  
    int[] t = y;  
    for (int k=0; k<t.length; k++) {  
        t[k]++;  
    }  
}
```

Output:

[3, 5, 7, 9]

Parameter: Array (umständlich, mit Rückgabe)

```
public static void main (String[] args) {  
    int[] g = {2, 4, 6, 8};  
    int[] h = plusOneMitRueck(g);  
    System.out.println(Arrays.toString(g));  
    System.out.println(Arrays.toString(h));  
}  
public static int[] plusOneMitRueck(int[] y) {  
    int[] t = new int[y.length];  
    for (int k=0; k<t.length; k++) {  
        t[k] = y[k]+1;  
    }  
    return t;  
}
```

Parameter: Array (umständlich, mit Rückgabe)

```
public static void main (String[] args) {  
    int[] g = {2, 4, 6, 8};  
    int[] h = plusOneMitRueck(g);  
    System.out.println(Arrays.toString(g));  
    System.out.println(Arrays.toString(h));  
}  
  
public static int[] plusOneMitRueck(int[] y) {  
    int[] t = new int[y.length];  
    for (int k=0; k<t.length; k++) {  
        t[k] = y[k]+1;  
    }  
    return t;  
}
```

Output:

[2, 4, 6, 8]

[3, 5, 7, 9]

Parameter: Array – was wird ausgegeben?

```
public static void main (String[] args) {  
    int[] g = {2, 4, 6, 8};  
    plusOhneEffekt(g);  
    System.out.println(Arrays.toString(g));  
}  
public static void plusOhneEffekt(int[] y) {  
    int[] t = new int[y.length];  
    for (int k=0; k<t.length; k++) {  
        t[k] = y[k]+1;  
    }  
    y = t;  
}
```

Parameter: Array – was wird ausgegeben?

```
public static void main (String[] args) {  
    int[] g = {2, 4, 6, 8};  
    plusOhneEffekt(g);  
    System.out.println(Arrays.toString(g));  
}  
public static void plusOhneEffekt(int[] y) {  
    int[] t = new int[y.length];  
    for (int k=0; k<t.length; k++) {  
        t[k] = y[k]+1;  
    }  
    y = t;  
}
```

Output:
[2, 4, 6, 8]

Parameter: Array – aber ohne Wirkung

```
public static void main (String[] args) {  
    int[] g = {2, 4, 6, 8};  
    plusOhneEffekt(g);  
    System.out.println(Arrays.toString(g));  
}  
  
public static void plusOhneEffekt(int[] y) {  
    int[] t = new int[y.length];  
    for (int k=0; k<t.length; k++) {  
        t[k] = y[k]+1;  
    }  
    y = t;  
}
```

Output:
[2, 4, 6, 8]

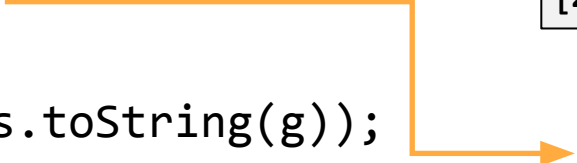
Index	0	1	2	3
Wert	2	4	6	8

Index	0	1	2	3
Wert	0	0	0	0

Parameter: Array – aber ohne Wirkung

```
public static void main (String[] args) {  
    int[] g = {2, 4, 6, 8};  
    plusOhneEffekt(g);  
    System.out.println(Arrays.toString(g));  
}  
  
public static void plusOhneEffekt(int[] y) {  
    int[] t = new int[y.length];  
    for (int k=0; k<t.length; k++) {  
        t[k] = y[k]+1;  
    }  
    y = t;  
}
```

Output:
[2, 4, 6, 8]




<i>Index</i>	0	1	2	3
<i>Wert</i>	2	4	6	8

<i>Index</i>	0	1	2	3
<i>Wert</i>	3	5	7	9

Merge Conflicts

Pushed to aroesti-test - origin



 master → master [rejected - non-fast-forward]

[rejected - non-fast-forward]

Message Details

Repository

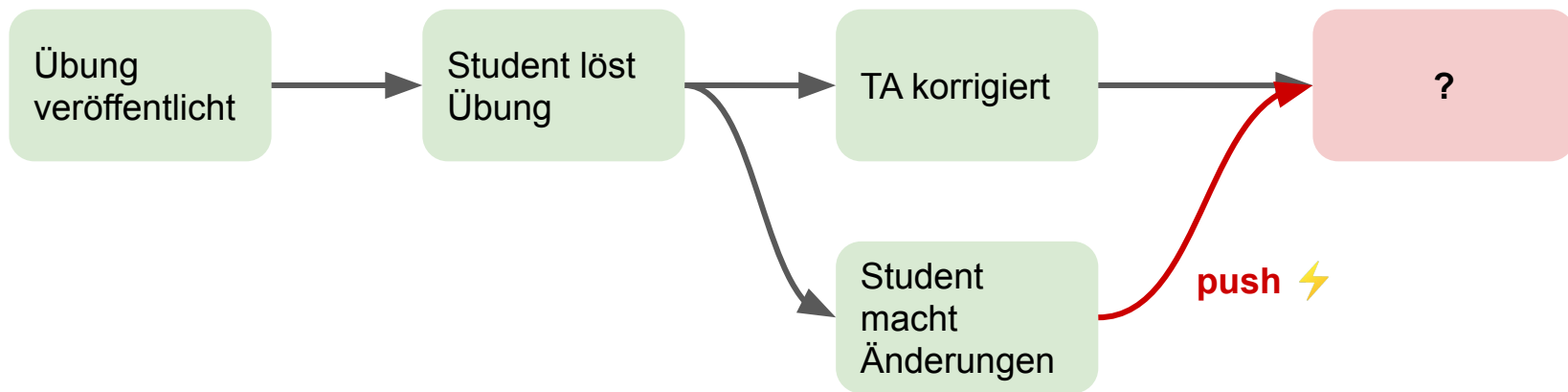
<https://gitlab.inf.ethz.ch/COURSE-EPROG2018/playground/aroesti-test.git>

Configure...

Close

Git - Merge Conflicts

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



Einfache Lösung

1. Im Fenster **Git Repositories**, Rechtsklick auf das Repository, dann **“Pull”**.
2. Zwei mögliche Ergebnisse:
 - a. Result: **“Merged”**. Das Problem ist gelöst. Mit Rechtsklick auf Repo, dann **“Push to Upstream”** können die eigenen Änderungen hochgeladen werden.
 - b. Result: **“Conflict”**. TA und Student haben dieselbe(n) Datei(en) bearbeitet. Die rot markierten Dateien müssen manuell durchgegangen werden. Konflikt-Zeilen sind mit **“<<<<”**, **“====”** und **“>>>>”** markiert. Nach dem Bearbeiten erneut committen. Dann ist ein Push möglich.

Zusatzübungen

Die Klasse Arrays

Java Doc!


- Die Klasse `Arrays` in der Bibliothek `java.util` enthält einige Methoden, die wir in einer `static` Methode aufrufen können

Method name	Description
<code>binarySearch(array, value)</code>	returns the index of the given value in a <i>sorted</i> array (or < 0 if not found)
<code>copyOf(array, length)</code>	returns a new copy of an array
<code>equals(array1, array2)</code>	returns true if the two arrays contain same elements in the same order
<code>fill(array, value)</code>	sets every element to the given value
<code>sort(array)</code>	arranges the elements into sorted order
<code>toString(array)</code>	returns a string representing the array, such as "[42, -7, 1, 15]"


Arrays und Schleifen

Welche Werte hat `vector` am Ende dieser Anweisungen?

```
int limit = 10;  
int [] vector = new int[limit];  
vector[0] = 1;  
vector[1] = 1;  
  
for (int i = 2; i < limit; i++) {  
    vector[i] = vector[i-1] +  
        vector[i-2];  
}
```



0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---



1	1	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

i-2	i-1	i							
1	1	0	0	0	0	0	0	0	0
1	1	2	0	0	0	0	0	0	0
1	1	2	3	0	0	0	0	0	0

...



1	1	2	3	5	8	13	21	34	55
---	---	---	---	---	---	----	----	----	----

Boolsche Ausdrücke

Was ergibt die Auswertung dieser boolschen Ausdrücke? (a, b vom Typ **boolean**; x, y, z vom Typ **int**)

```
x = 12; y = 7; z = 5;
```

```
x % y == z || x < y
```

```
y < x && y <= z
```

```
x % 2 == y % 2 || x % 2 == z % 2
```

```
x <= y + z && x >= y + z
```

```
!(x < y && x < z)
```

```
(x + y) % 2 == 0 || !((z - y) % 2 == 0)
```

```
(!(a&&b) && (a||b)) || ((a&&b) || !(a||b))
```

Die Mitte

Schreiben Sie eine Methode `middleElements` die für einen Array ganzer Zahlen (also `int[]` input) folgendes zurückgibt:

- a. Wenn die Anzahl der Elemente ungerade ist, dann wird das Element zurückgegeben, welches genau in der Mitte ist.
- b. Wenn die Anzahl der Elemente gerade ist, dann werden die beiden Elemente zurückgegeben, welche in der Mitte sind.

Vorbesprechung Übung 4

Aufgabe 1: Sieb des Eratosthenes

Schreiben Sie ein Programm "Sieb.java", das eine Zahl *limit* einliest und die Anzahl der Primzahlen, die grösser als 1 und kleiner oder gleich dem *limit* sind, ausgibt. Dazu ermitteln Sie in einem ersten Schritt alle Primzahlen, die kleiner oder gleich *limit* sind, und merken sich diese in einem Array. Dieses Teilproblem können Sie mit dem [Sieb des Eratosthenes](#) lösen. Danach können Sie die Anzahl der gefundenen Primzahlen anhand dieses Arrays bestimmen.

Beispiel: Für $limit = 13$ sollte Ihr Programm 6 ausgeben (Primzahlen: 2, 3, 5, 7, 11, 13).

Sieb des Eratosthenes

1. Erstelle array `sieb` von `boolean`, Länge `limit+1`
2. Setze alle Elemente auf `true`
3. Für $i > 2$, setze `sieb[i]` auf `false` wenn i ein ganzzahliges Vielfaches einer Zahl $z \geq 2$ ist

index i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
sieb[i]			t	t	t	t	t	t	t	t	t	t	t	t	t	t
			t	t	f	t	f	t	f	f	f	t	f	t	f	f

Aufgabe 2: Newton-Raphson

Schreiben Sie ein Programm "Newton.java", das eine ganze positive Zahl einliest und eine Approximation der Quadratwurzel nach dem Newton-Raphson Verfahren auf 12 Stellen nach dem Komma berechnet. (Sie sollen selbst entscheiden, welche Methoden Sie definieren.)

Hinweis:

t Approximation der Wurzel von c (t und c sind vom Typ double)

eps maximal erlaubter Fehler

$abs(t \cdot t - c) < eps$ falls true ist unsere Approximation gut genug, sonst ...

$t' = \frac{c/t + t}{2.0}$ neue Approximation t' ist der Mittelwert von c/t und t

siehe auch: <https://de.wikipedia.org/wiki/Newton-Verfahren>,
vor allem *Berechnung der Quadratwurzel*

Aufgabe 3: Loop Invariante

Gegeben sind die Precondition und Postcondition für das folgende Programm

```
public int compute(int a, int b) {  
    // Precondition:  a >= 0  
    int x;  
    int res;  
  
    x = 0;  
    res = b;  
  
    // Loop Invariante:  
    while (x < a) {  
        res = res - 1;  
        x = x + 1;  
    }  
    // Postcondition:  res == b - a  
    return res;  
}
```

Was ist die Loop Invariante?

Loop Invariante

Beispiel

```
public int compute(int a, int b) {  
  
    // Precondition:  a >= 0  
  
    int x;  
    int res;  
  
    x = a;  
    res = b;  
  
    // Loop Invariante: ??  
  
    while (x > 0) {  
        x = x - 1;  
        res = res + 1;  
    }  
  
    // Postcondition:  res = a + b  
    return res;  
}
```

Aufgabe 4: Testen mit JUnit

Zweck des Programms:

- Wochentag eines Datums (nach 01.01.1900) ausgeben
Beispiel: 13.10.2017 → Friday
Gibt fälschlicherweise aber *“The 13.10.2017 is a Sunday”* aus.
- Berücksichtigt Schaltjahre (“Leap year”)

Funktionsweise:

1. Überprüft, ob Datum OK ist
2. Zählt die Tage ab 1.1.1900 bis zum eingegebenen Datum
3. Wochentag = Tage % 7

Aufgabe 4: Testen mit JUnit

Tests in *PerpetualCalendarTest.java*

- Einzelne Tests prüfen Rückgabewerte von einzelnen Methoden des Programms *PerpetualCalendar.java*
- Tests sollten *interessante* Parameter für die Methoden testen
- Beispiel `testCountDaysInYear()`:
`assertEquals(366, PerpetualCalendar.countDaysInYear(1904));`
1904 ist ein Schaltjahr, also sollte `countDaysInYear()` 366 Tage zurückgeben

DEMO

Aufgabe 5: Arrays

In dieser Aufgabe implementieren Sie Methoden, welche Arrays verwenden.

1. Implementieren Sie die Methode `ArrayUtil.zeroInsert(int[] x)`, das heisst die Methode `zeroInsert` in der Klasse `ArrayUtil`. Die Methode nimmt einen Array `x` als Argument und gibt einen Array zurück. Der zurückgegebene Array soll die gleichen Werte wie `x` haben, ausser: Wenn eine positive Zahl direkt auf eine negative Zahl folgt oder wenn eine negative Zahl direkt auf eine positive Zahl folgt, dann wird dazwischen eine 0 eingefügt.

Beispiele:

- Wenn `x` gleich `[3, 4, 5]` ist, dann wird `[3, 4, 5]` zurückgegeben.
- Wenn `x` gleich `[3, 0, -5]` ist, dann wird `[3, 0, -5]` zurückgegeben.
- Wenn `x` gleich `[-3, 4, 6, 9, -8]` ist, dann wird `[-3, 0, 4, 6, 9, 0, -8]` zurückgegeben.

Versuchen Sie die Methode rekursiv zu implementieren.

Aufgabe 5: Arrays

2. Implementieren Sie die Methode `ArrayUtil.tenFollows(int[] x, int index)`. Die Methode gibt einen Boolean zurück. Die Methode soll `true` zurückgeben, wenn im Array `x` ab Index `index` der zehnfache Wert einer Zahl `n` direkt der Zahl `n` folgt. Ansonsten soll die Methode `false` zurückgeben.

Beispiele:

- `tenFollows([1, 2, 20], 0)` gibt `true` zurück.
- `tenFollows([1, 2, 7, 20], 0)` gibt `false` zurück.
- `tenFollows([3, 30], 0)` gibt `true` zurück.
- `tenFollows([3], 0)` gibt `false` zurück.
- `tenFollows([1, 2, 20, 5], 1)` gibt `true` zurück.
- `tenFollows([1, 2, 20, 5], 2)` gibt `false` zurück.

Erste Bonusaufgabe

Achtung: eigenes Eclipse-Projekt

Aufgabe 6: Close Neighbors (Bonus)

Achtung: Diese Aufgabe gibt Bonuspunkte (siehe “Leistungskontrolle” im www.vvz.ethz.ch). Die Aufgabe muss eigenhändig und alleine gelöst werden. Die Abgabe erfolgt wie gewohnt per Push in Ihr Git-Repository auf dem ETH-Server. Verbindlich ist der letzte Push vor dem Abgabetermin. Auch wenn Sie vor der Deadline committen, aber nach der Deadline pushen, gilt dies als eine zu späte Abgabe. Bitte lesen Sie zusätzlich [die allgemeinen Regeln](#).

Aufgabe 6: Close Neighbors (Bonus)

Achtung: Diese Aufgabe gibt Bonuspunkte (siehe “Leistungskontrolle” im www.vvz.ethz.ch). Die Aufgabe muss eigenhändig und alleine gelöst werden. Die Abgabe erfolgt wie gewohnt per Push in Ihr Git-Repository auf dem ETH-Server. Verbindlich ist der letzte Push vor dem Abgabetermin. Auch wenn Sie vor der Deadline committen, aber nach der Deadline pushen, gilt dies als eine zu späte Abgabe. Bitte lesen Sie zusätzlich [die allgemeinen Regeln](#).

Aufgabe 6: Close Neighbors (Bonus)

Schreiben Sie ein Programm, welches für eine sortierte Folge X von `int`-Werten (x_1, x_2, \dots, x_n) und einen `int`-Wert `key` die drei unterschiedlichen Elemente x_a , x_b und x_c aus X zurückgibt, die dem Wert `key` am nächsten sind. Für x_a , x_b und x_c muss gelten, dass $|\text{key} - x_a| \leq |\text{key} - x_b| \leq |\text{key} - x_c| \leq |\text{key} - x_i|$ für alle $i \neq a, b, c$ und dass $x_a \neq x_b \neq x_c \neq x_a$. Wenn die drei Werte nicht eindeutig bestimmt sind, dann ist jede Lösung zugelassen, die die obige Bedingung erfüllt.

Beispiele:

Die nächsten Nachbarn für `key == 5` in $(1, 4, 5, 7, 9, 10)$ sind 5, 4, 7.

Die nächsten Nachbarn für `key == 5` in $(1, 4, 5, 6, 9, 10)$ sind 5, 4, 6 oder 5, 6, 4.

Die nächsten Nachbarn für `key == 10` in $(1, 4, 5, 6, 9, 10)$ sind 10, 9, 6.

Implementieren Sie die Berechnung in der Methode `int[] neighbors(int[] sequence, int key)`, welche sich in der Klasse `Neighbor` befindet. Die Deklaration der Methode ist bereits vorgegeben. Sie können davon ausgehen, dass das Argument `sequence` nicht `null` ist, sortiert ist, nur unterschiedliche Elemente enthält, und mindestens drei Elemente enthält. Denken Sie daran, dass der Wert `key` nicht unbedingt in der Folge X auftritt. Sie dürfen den Eingabearray `input` nicht ändern.

Aufgabe 6: Close Neighbors (Bonus)

In der `main` Methode der Klasse `Neighbor` finden Sie die oberen Beispiele als kleine Tests, welche Beispiel-Aufrufe zur `neighbors`-Methode machen und welche Sie als Grundlage für weitere Tests verwenden können. In der Datei `NeighborTest.java` geben wir die gleichen Tests zusätzlich auch als JUnit Test zur Verfügung. Sie können diese ebenfalls nach belieben ändern. Es wird *nicht* erwartet, dass Sie für diese Aufgabe den JUnit Test verwenden.

Pushzeitpunkt


Kahoot

Zusatzübungen


Arrays und Schleifen

Welche Werte hat `vector` am Ende dieser Anweisungen?

```
int limit = 10;  
int [] vector = new int[limit];  
vector[0] = 1;  
vector[1] = 1;  
  
for (int i = 2; i < limit; i++) {  
    vector[i] = vector[i-1] +  
        vector[i-2];  
}
```



0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---



1	1	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

i-2	i-1	i							
1	1	0	0	0	0	0	0	0	0
1	1	2	0	0	0	0	0	0	0
1	1	2	3	0	0	0	0	0	0

...



1	1	2	3	5	8	13	21	34	55
---	---	---	---	---	---	----	----	----	----

Boolsche Ausdrücke

Was ergibt die Auswertung dieser boolschen Ausdrücke? (a, b vom Typ **boolean**; x, y, z vom Typ **int**)

```
x = 12; y = 7; z = 5;
```

```
x % y == z || x < y
```

```
y < x && y <= z
```

```
x % 2 == y % 2 || x % 2 == z % 2
```

```
x <= y + z && x >= y + z
```

```
!(x < y && x < z)
```

```
(x + y) % 2 == 0 || !((z - y) % 2 == 0)
```

```
(!(a&&b) && (a||b)) || ((a&&b) || !(a||b))
```

Die Mitte

Schreiben Sie eine Methode `middleElements` die für einen Array ganzer Zahlen (also `int[]` input) folgendes zurückgibt:

- a. Wenn die Anzahl der Elemente ungerade ist, dann wird das Element zurückgegeben, welches genau in der Mitte ist.
- b. Wenn die Anzahl der Elemente gerade ist, dann werden die beiden Elemente zurückgegeben, welche in der Mitte sind.

Boolsche Ausdrücke

```
boolean a = true;
```

```
boolean b = false;
```

```
boolean c = ((a && b) || (a || b));
```

```
System.out.println(c);
```

a	b	
false	false	false
false	true	true
true	false	true
true	true	true

a	b	&&
false	false	false
false	true	false
true	false	false
true	true	true

Boolsche Ausdrücke

Vereinfachen Sie den Ausdruck (a und b sind vom Typ `int`):

$$(\neg (a < b) \ \&\& \ \neg (a > b))$$

Schleifen

Wie oft wird diese while-Schleife durchlaufen? Ausgabe des Programms?

```
int x = 1;
System.out.print(x);
while (x < 100) {
    x = x + x;
    System.out.print(", " + x);
}
```

Binäre Darstellung

Gegeben sei ein `int` array `'bits'` der Länge 8 (`int [] bits = new int[8]`). Jedes Element von `bits` ist entweder 0 oder 1. `bits` ist die Binärdarstellung einer Zahl `z`, die wie folgt definiert ist:

$$z = \sum_{j=0}^7 \text{bits}[j] \cdot 2^j \quad \longrightarrow \quad \text{for } (\text{int } j = 0; j \leq 7; j++) \{ \dots \}$$

Schreiben Sie ein Programm, das 8 Ziffern (0 oder 1) von der Input Console liest und die entsprechende Zahl `z` ausgibt. (Achtung: Sie müssen die 8 Binärziffern einzeln eingeben, also z.B. `0 0 0 1 0 0 0 1` (17) und nicht `00010001`)

Schleifen

Was gibt die Methode
aus für $n = 2, 5, 24, 28$?

Was “bedeutet” die
Ausgabe?

```
public static void methodeA(int n) {  
    int x = 1;  
    int y = 2;  
    while (y < n) {  
        if (n % y == 0) {  
            n = n / y;  
            x++;  
        } else {  
            y++;  
        }  
    }  
    System.out.println(x + " " + n);  
}
```

Schleifen

Schreiben Sie eine Methode `quersumme`, welche die Quersumme einer ganzen Zahl berechnet.

Beispiele:

`quersumme (315)` gibt 9 zurück

`quersumme (-903)` gibt 12 zurück

`quersumme (0)` gibt 0 zurück

Methoden

Finden Sie die Fehler...

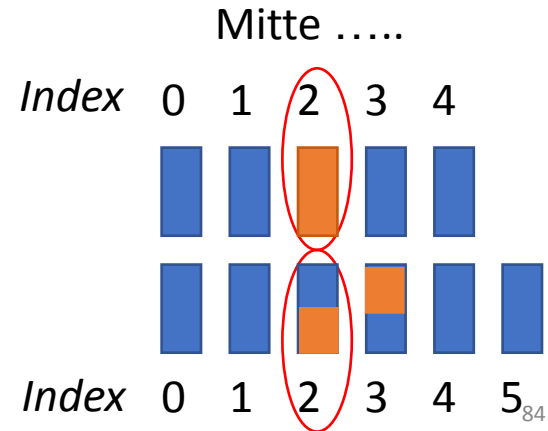
```
public class Parameters {  
    public static void main() {  
        double bubble = 867.5309;  
        double x = 10.01;  
        printer(double x, double y);  
        printer(x);  
        printer("hello", "world");  
        System.out.println("z = " + z);  
    }  
  
    public static void printer(x, y double) {  
        int z = 5;  
        System.out.println("x = " + double x + ", y = " + y);  
        System.out.println("The value from main: " + bubble);  
    }  
}
```

Das length Attribut

- Das length Attribut eines Arrays ***name*** liefert die Anzahl der Elemente.

name.length

- Was für ein Ausdruck erlaubt den Zugriff auf:
 - Das letzte Element des Arrays (***name***)?
 - Das Element in der Mitte?

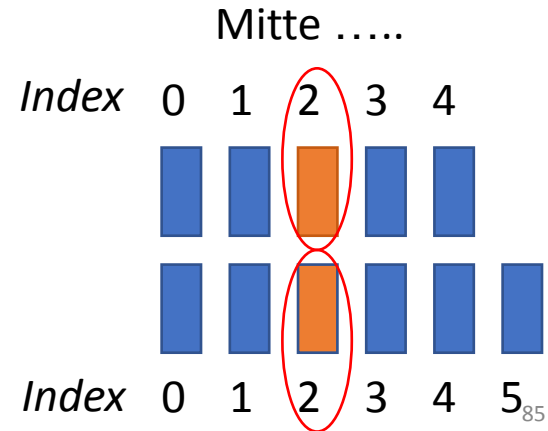


Das length Attribut

- Das length Attribut eines Arrays ***name*** liefert die Anzahl der Elemente.

name.length

- Was für ein Ausdruck erlaubt den Zugriff auf:
 - Das letzte Element des Arrays (***name***)?
 - Das Element in der Mitte?



Das length Attribut

- Das length Attribut eines Arrays ***name*** liefert die Anzahl der Elemente.

name.length

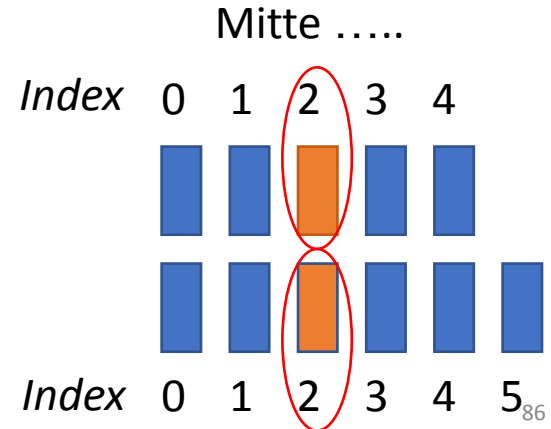
- Was für ein Ausdruck erlaubt den Zugriff auf:

- Das letzte Element des Arrays (*name*)?

`name[name.length-1]`

- Das Element in der Mitte?

`name[(name.length-1)/2]`




Fragen?

Merge Conflicts

Pushed to aroesti-test - origin



 master → master [rejected - non-fast-forward]

[rejected - non-fast-forward]

Message Details

Repository

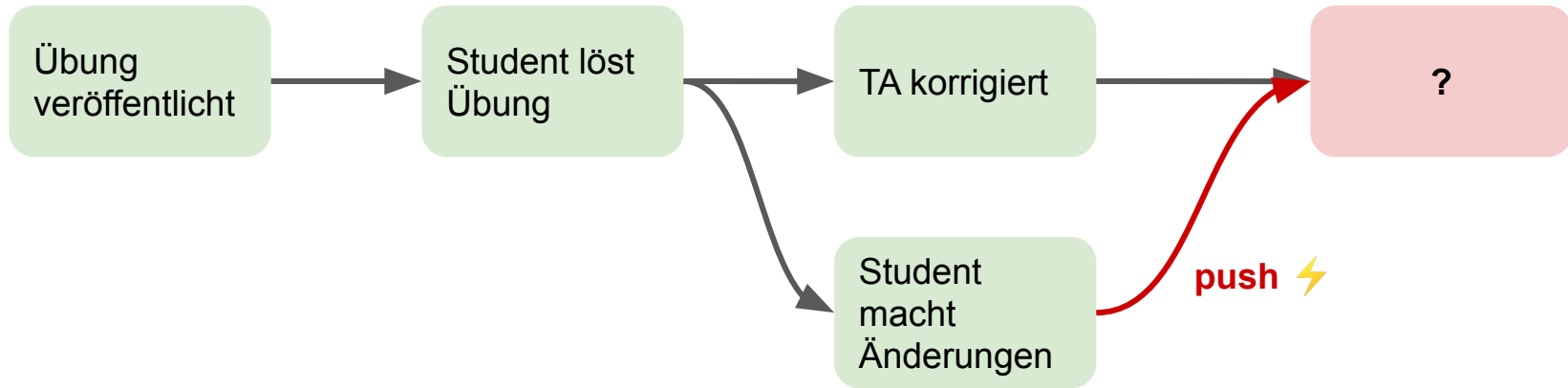
<https://gitlab.inf.ethz.ch/COURSE-EPROG2018/playground/aroesti-test.git>

Configure...

Close

Git - Merge Conflicts

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



Einfache Lösung

1. Im Fenster **Git Repositories**, Rechtsklick auf das Repository, dann **“Pull”**.
2. Zwei mögliche Ergebnisse:
 - a. Result: **“Merged”**. Das Problem ist gelöst. Mit Rechtsklick auf Repo, dann **“Push to Upstream”** können die eigenen Änderungen hochgeladen werden.
 - b. Result: **“Conflict”**. TA und Student haben dieselbe(n) Datei(en) bearbeitet. Die rot markierten Dateien müssen manuell durchgegangen werden. Konflikt-Zeilen sind mit **“<<<<”**, **“====”** und **“>>>>”** markiert. Nach dem Bearbeiten erneut committen. Dann ist ein Push möglich.

Pull Result for aroesti-test

Fetch Result

- ▼  master : origin/master [5661974..55e203a] (1)
 - ▶  5661974c: Korrektur (Andre Roesti on 2018-10-13 20:00:53)



Update Result

Result **Merged**

New HEAD Merge branch 'master' of <https://gitlab.inf.ethz.ch/COURSE-EPROG2018/playground/aroesti-test.git> [8d16702]

Merge input

-  ef91996e: Übung gelöst (Andre Roesti on 2018-10-13 20:01:29)
-  5661974c: Korrektur (Andre Roesti on 2018-10-13 20:00:53)

Close

