

252-0027

**Einführung in die Programmierung
Übungen**

Woche 4: Types, Schleifen, Methoden

Jonas Wetzel

Departement Informatik

ETH Zürich

Organisatorisches

- Mein Name: Jonas Wetzel
- Bei Fragen: jwetzel@ethz.ch
- **Mails bitte mit «[EProg24]» 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

- **website: n.ethz.ch/~jwetz**
- **komm in die WhatsApp Gruppe**

Plan heute

- **Nachbesprechung Übung**
- **Theorie**
- **Tipps für die nächste Übung**
- **Kahoot**
- **Outro**

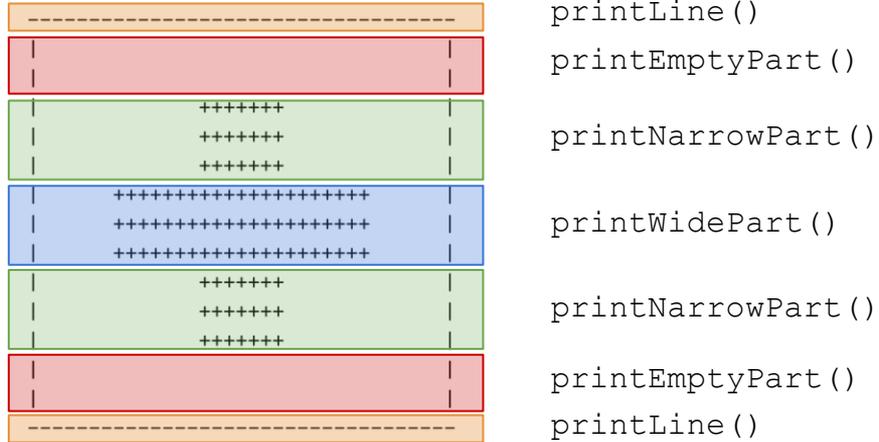
Nachbesprechung

Aufgabe 1: Fehlerbehebung

```
1 public class FolleVehler {  
2     public static main(String args) {  
3         System.out.println(Hello world);  
4         system.out.Pritnln("Gefällt Ihnen dieses Programm?");  
5         System.out.println()  
6  
7         System.println("Ich habe es selbst geschrieben.");  
8     }  
9 }
```

```
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?

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?

$x / 10$ gibt uns die erste Ziffer von x

$x \% 10$ gibt uns die letzte Ziffer

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. $a + c == b$ " ausgegeben (wobei die Werte für a, b, und c einzusetzen sind). Wenn das nicht der Fall ist, dann soll die Methode "Unmoeglich." ausgegeben.

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.)

```
<pyramid> <- [0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [9] [8] [7] [6] [5] [4] [3] [2] [1] [0]
```

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: $1 + 4$, $(1 + 4)$, $1 + (3 + 4)$, $(1 + 1) + x + 5$.

2.

- `<even> <- 0|2|4|6|8`
- `<odd> <- 1|3|5|7|9`
- `<digitSum> <- { <even> | <odd> { <digitSum> } <odd> }`

3.

- $\langle \text{rec} \rangle \leftarrow \{ X \mid Y \mid Z \}$
- $\langle \text{xyz} \rangle \leftarrow \{ Z \mid Y \} [Y \langle \text{rec} \rangle Y \{ Z \}]$
- Basically müssen wir nur forcen, dass jedes X immer von zwei Y's umschlossen ist

Z, YXY, YXXY, ZYXYY.

4.

- $\langle \text{digit} \rangle \quad \leftarrow 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
- $\langle \text{number} \rangle \quad \leftarrow \langle \text{digit} \rangle \{ \langle \text{digit} \rangle \mid 0 \} \mid 0$
- $\langle \text{var} \rangle \quad \leftarrow x \mid y \mid z$
- $\langle \text{bracketedTerm} \rangle \quad \leftarrow '(' \langle \text{term} \rangle + \langle \text{term} \rangle ')'$
- $\langle \text{term} \rangle \quad \leftarrow (\langle \text{term} \rangle + \langle \text{term} \rangle) \mid \langle \text{bracketedTerm} \rangle \mid \langle \text{number} \rangle \mid \langle \text{var} \rangle$

$1 + 4, (1 + 4), 1 + (3 + 4), (1 + 1) + x + 5.$

Theorie

data types in java

- **Es gibt primitive types und reference types**
 - Über reference types lernen wir später, wenn ihr Objekte seht

Primitive Typen in Java

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

<u>Name</u>	<u>Beschreibung</u>	<u>Beispiele</u>
<code>int</code>	ganze Zahlen	<code>-2147483648, -3, 0, 42, 2147483647</code>
<code>long</code>	grosse ganze Zahlen	<code>42, -3, 0, 9223372036854775807</code>
<code>double</code>	reelle Zahlen	<code>3.1, -0.25, 9.4e3</code>
<code>char</code>	(einzelne) Buchstaben	<code>'a', 'X', '?', '\n'</code>
<code>boolean</code>	Wahrheitswerte	<code>true, false</code>

- **Landen auf dem Stack (Memory)**
- **Grösse ist beschränkt**
 - Bsp. int hat 32 bits, long hat 64 bits, double hat 64 bits
- **Einfache Daten**

- **Strings sind keine primitive types, sondern sogenannte Reference Types! -> lernen wir noch später**
 - Arrays, Objects...

Was passiert hier?

```
public class PassByValueDemo {  
  
    public static void main(String[] args) {  
        int originalValue = 10;  
  
        System.out.println("Wert vor dem Aufruf der Methode: " + originalValue);  
  
        // Methode wird aufgerufen und der Wert wird kopiert  
        changeValue(originalValue);  
  
        // Der ursprüngliche Wert bleibt unverändert  
        System.out.println("Wert nach dem Aufruf der Methode: " + originalValue);  
    }  
  
    public static void changeValue(int value) {  
        // Die übergebene Kopie wird verändert  
        value = 20;  
        System.out.println("Wert innerhalb der Methode: " + value);  
    }  
}
```

Was passiert hier?

```
public class PassByValueDemo {  
  
    public static void main(String[] args) {  
        int originalValue = 10;  
  
        System.out.println("Wert vor dem Aufruf der Methode: " + originalValue);  
  
        // Methode wird aufgerufen und der Wert wird kopiert  
        changeValue(originalValue);  
  
        // Der ursprüngliche Wert bleibt unverändert  
        System.out.println("Wert nach dem Aufruf der Methode: " + originalValue);  
    }  
  
    public static void changeValue(int value) {  
        // Die übergebene Kopie wird verändert  
        value = 20;  
        System.out.println("Wert innerhalb der Methode: " + value);  
    }  
}
```



```
Wert vor dem Aufruf der Methode: 10  
Wert innerhalb der Methode: 20  
Wert nach dem Aufruf der Methode: 10
```

Was passiert hier?

```
public class PassByValueDemo {  
  
    public static void main(String[] args) {  
        int originalValue = 10;  
  
        System.out.println("Wert vor dem Aufruf der Methode: " + originalValue);  
  
        // Methode wird aufgerufen und der Wert wird kopiert  
        changeValue(originalValue);  
  
        // Der ursprüngliche Wert bleibt unverändert  
        System.out.println("Wert nach dem Aufruf der Methode: " + originalValue);  
    }  
  
    public static void changeValue(int value) {  
        // Die übergebene Kopie wird verändert  
        value = 20;  
        System.out.println("Wert innerhalb der Methode: " + value);  
    }  
}
```

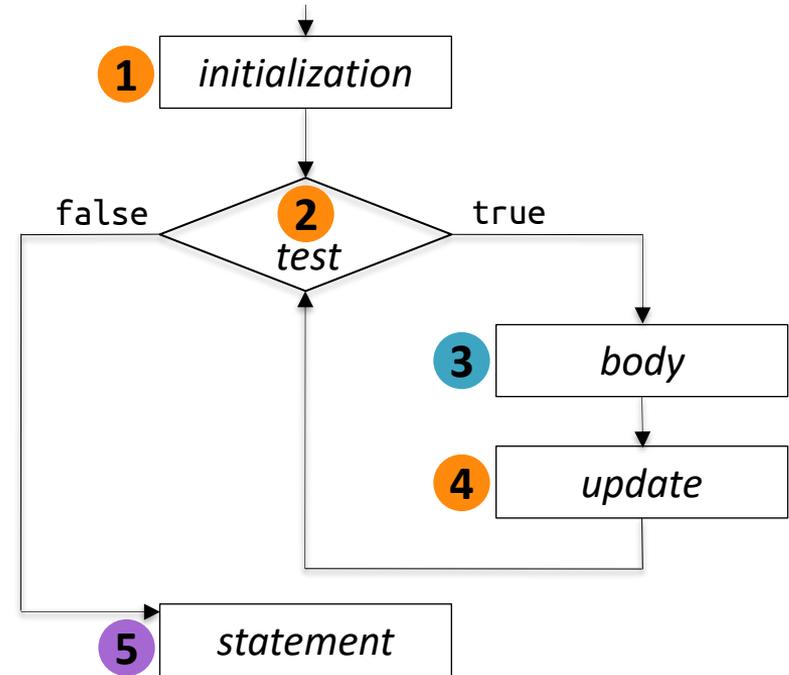
- Pass-by-value bedeutet, dass bei der Übergabe an eine Methode eine Kopie des Werts übergeben wird.
- Bei primitiven Datentypen (z.B. int, float, boolean):
 - Die Methode erhält nur eine Kopie des Werts, nicht die ursprüngliche Variable.
 - Änderungen in der Methode wirken sich nicht auf die ursprüngliche Variable außerhalb der Methode aus.
 - Nach dem Ende der Methode wird die Kopie verworfen und der ursprüngliche Wert bleibt unverändert.

Schleifen

for-Schleife («for loop»): Kontrollfluss

```
for (1 initialization; 2 test; 4 update) {  
    3 body  
}  
5 statement;
```

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



for-Schleife: Syntax, Semantik

```
for (initialization; test; update) {
```

```
statement1;  
statement2;  
statement3;  
...
```

Kopf («head»)

Rumpf/Körper
(«body»)

```
}  
statement;
```

Schleifen-/Laufvariable («loop variable»)

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

Syntax (vereinfacht)

- **initialization**: Variablendeklaration oder Variablenzuweisung
- **test**: Boolescher Ausdruck
- **update**: Ausdruck mit Seiteneffekt

Semantik (siehe auch vorherige Folie)

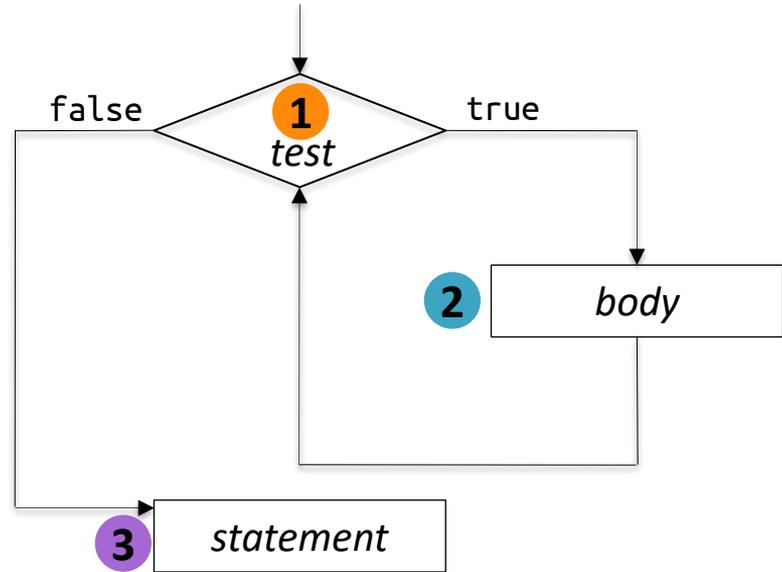
1. Einmalige Ausführung von **initialization**
2. Wiederhole bis Abbruch:
 1. Prüfe, ob **test** zu true evaluiert. Falls ja, Abbruch: springe zu **statement**.
 - Führe Rumpf aus
 - Führe **update** aus

Our favorite function

- See `factorial.java`

while-Schleife («while loop»)

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



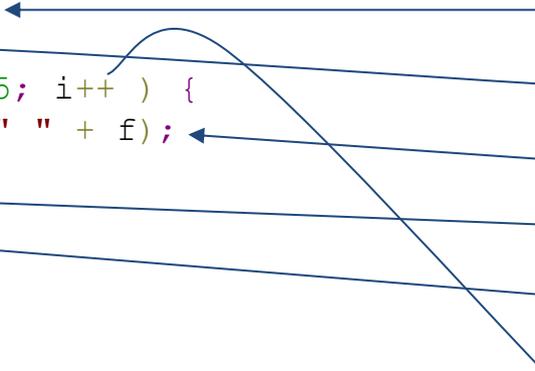
while-Schleife führt Rumpf so lange aus, wie *test* den Wert *true* ergibt.
In anderen Worten: die while-Schleife bricht ab, sobald *test* den Wert *false* ergibt.

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
...

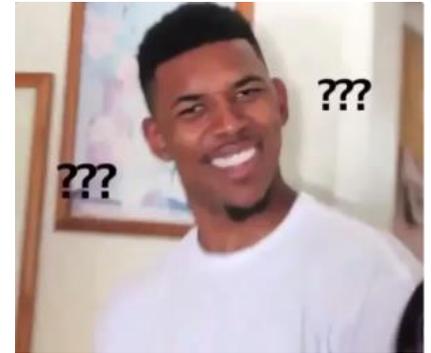


Schleifen

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
...

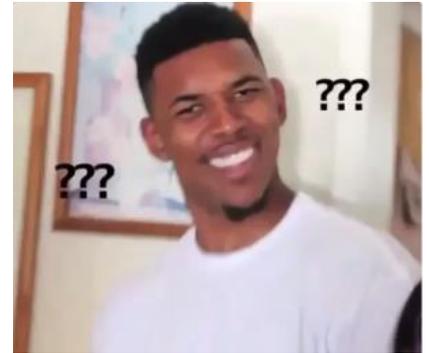


Schleifen

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 vor 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



```
1 public class MyClass {  
2     public static void main(String[] args){  
3         for(i = 0; i < 5; i++){  
4             System.out.println("i = " + i);  
5         }  
6     }  
7 }
```

Schleifen: For vs. While – Beispiel 1

```
1 public class MyClass {  
2     public static void main(String[] args){  
3         int i = 0;           // Initialisierung  
4         while(i < 5){       // Bedingung  
5             System.out.println("i = " + i);  
6             i++;           // Aktualisierung  
7         }  
8     }  
9 }
```

Schleifen: For vs. While – Beispiel 2

```
1 public class MyClass {  
2     public static void main(String[] args){  
3         int i = 1;  
4         while(fancyCheck(i)){  
5             int result = complexOperation(i);  
6             i = i * 42 - 15;  
7             System.out.println(result);  
8         }  
9     }  
10 }
```

Schleifen: For vs. While – Beispiel 2

```
1 public class MyClass {  
2     public static void main(String[] args){  
3         int i = 1;  
4         for(; fancyCheck(i); ){  
5             int result = complexOperation(i);  
6             i = i * 42 - 15;  
7             System.out.println(result);  
8         }  
9     }  
10 }
```

Schleifen: Do-While?

- Selten genutzt aber kann sehr nützlich sein!

```
1 public class MyClass {
2     public static void main(String[] args){
3         Scanner console = new Scanner(System.in);
4         String pin = "0000";
5         do {
6             pin = console.next();
7         } while(isPassword(pin));
8         System.out.println("You logged in successfully!")
9         console.close();
10    }
11 }
```

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

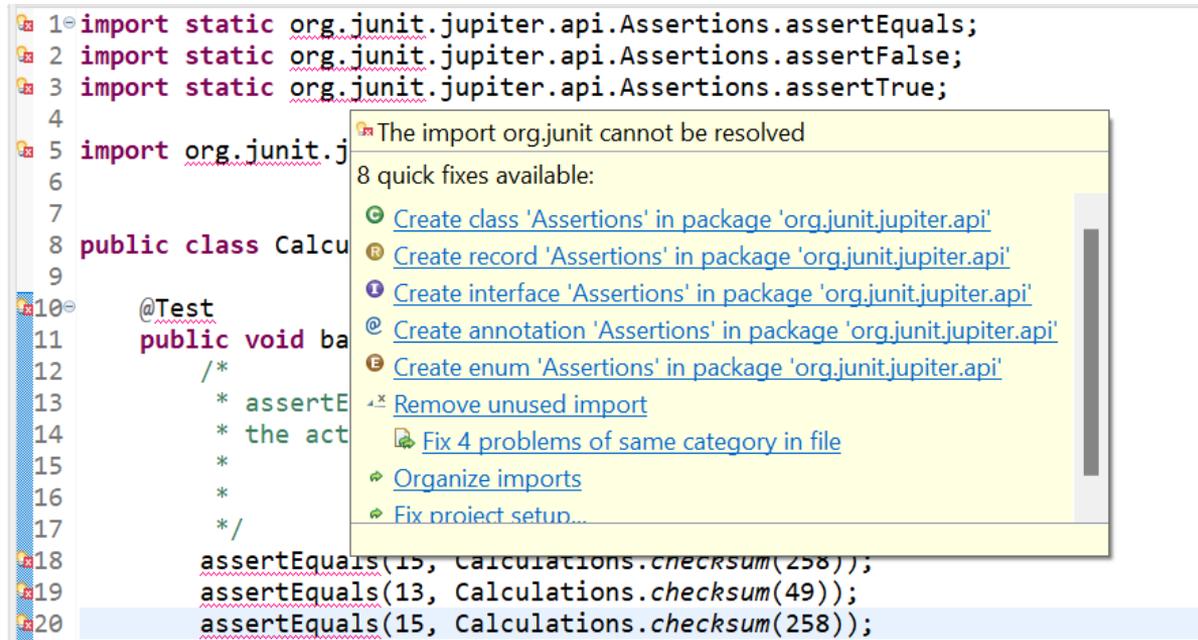
JUnit Tests: Introduction

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

JUnit Tests: Eclipse

- Junit kann nicht gefunden werden?

```
1 import static org.junit.jupiter.api.Assertions.assertEquals;
2 import static org.junit.jupiter.api.Assertions.assertFalse;
3 import static org.junit.jupiter.api.Assertions.assertTrue;
4
5 import org.junit.j
6
7
8 public class Calcula
9
10 @Test
11 public void ba
12     /*
13      * assertEquals
14      * the act
15      *
16      *
17      */
18 assertEquals(15, Calculations.checksum(258));
19 assertEquals(13, Calculations.checksum(49));
20 assertEquals(15, Calculations.checksum(258));
```



JUnit Tests: Eclipse

- **JUnit kann nicht gefunden werden?**

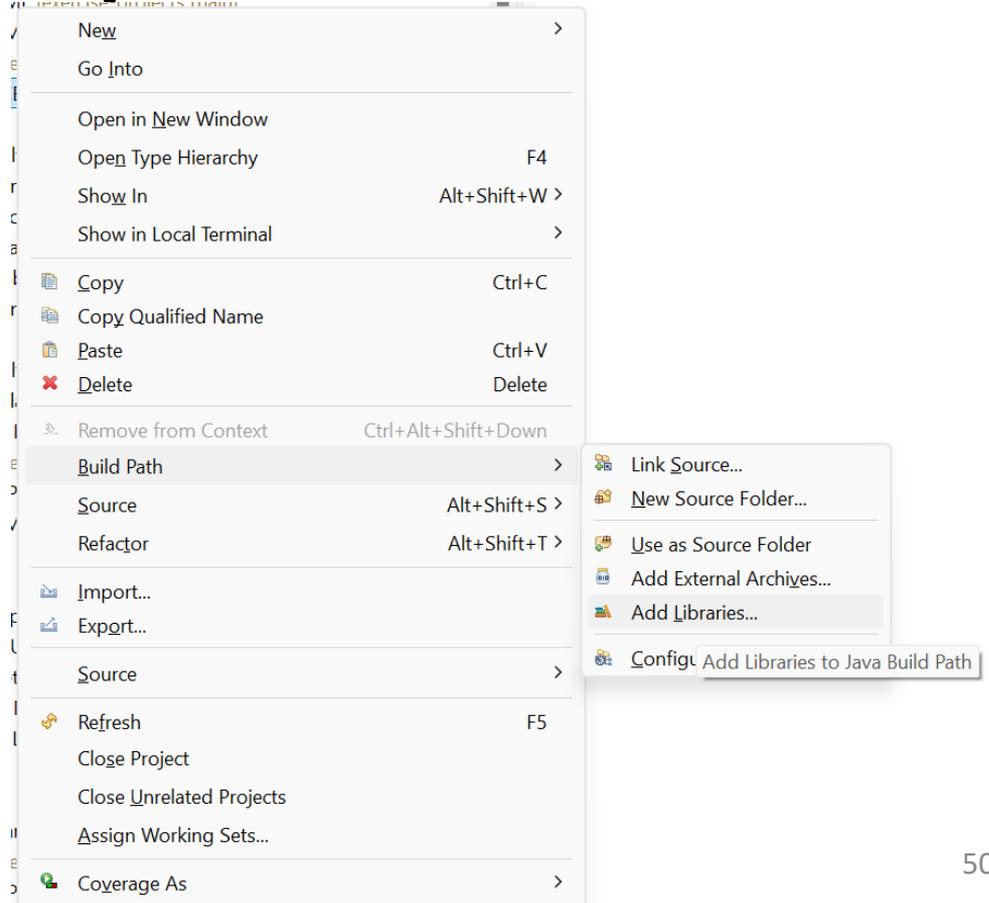
- ∨  u03
 - >  src
 - >  test
 - >  JRE System Library [jre]

- **JUnit library muss importiert werden!**

- ∨  u03
 - >  src
 - >  test
 - >  JRE System Library [jre]
 - >  JUnit 5

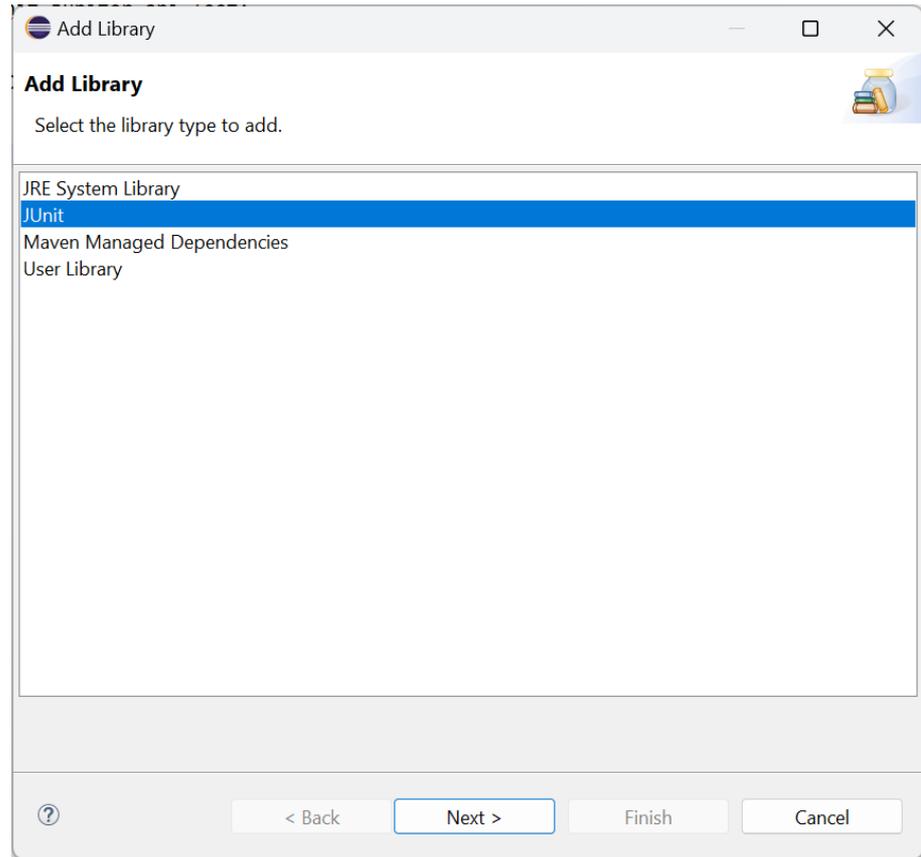
JUnit Tests: JUnit Importieren

- **Rechtclick auf Project Folder, z.B. u03.**
- **Build Path -> Add Libraries**



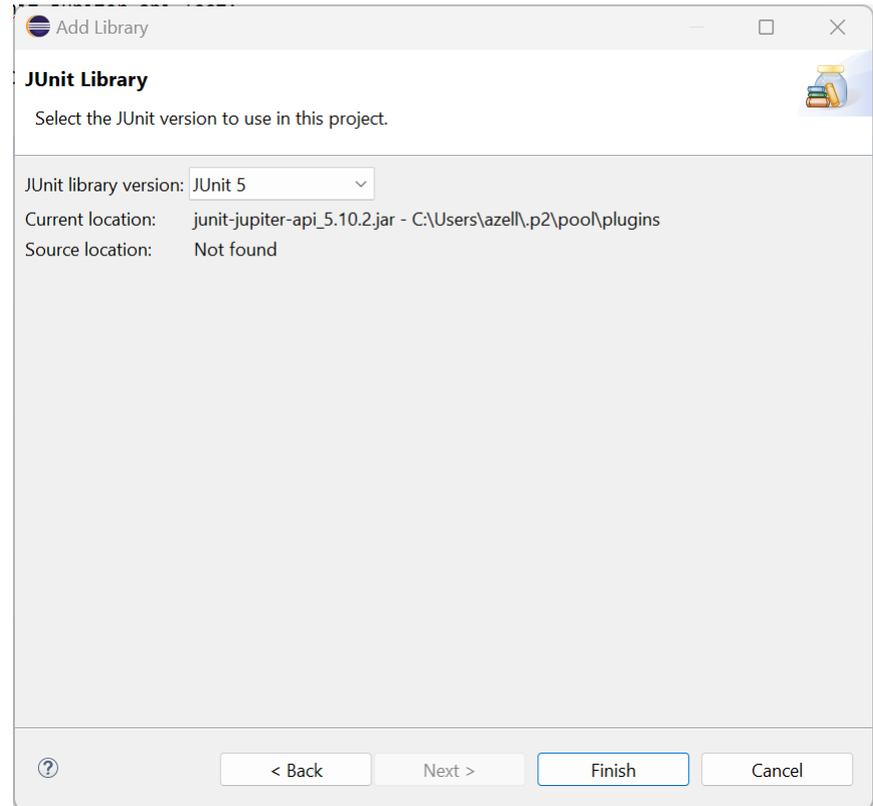
JUnit Tests: JUnit Importieren

- **Rechtsklick auf Project Folder, z.B. u03.**
- **Build Path -> Add Libraries**
- **Add Library: JUnit + next**



JUnit Tests: JUnit Importieren

- **Rechtsklick auf Project Folder, z.B. u03.**
- **Build Path -> Add Libraries**
- **Add Library: Junit und next klicken.**
- **JUnit 5 wählen und finish klicken.**



- **See FactorialTest.java**

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:

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

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 = 0; i <= k; i++) {
        pot = pot * n;
    }

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

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);
}
```

Immer achten auf
Initialisierung von i,
Abbruchbedingung

String

String-Methoden, die String liefern

Name der Methode	Beschreibung der Methode
<code>substring(i1, i2)</code> or <code>substring(i1)</code>	Ein neuer String: Der Substring von <code>i1</code> (inklusive) bis <code>i2</code> (<u>exklusive</u>); falls kein <code>i2</code> übergeben wird: bis Ende des Strings
<code>toLowerCase()</code>	Ein neuer String mit nur Kleinbuchstaben
<code>toUpperCase()</code>	Ein neuer String mit nur Grossbuchstaben
<code>stripLeading()</code>	Ein neuer String ohne Leerzeichen am Anfang
<code>stripTrailing()</code>	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
<code>length()</code>	Länge des Strings (Anzahl Zeichen)
<code>indexOf(s, fromIndex)</code> <code>indexOf(c, fromIndex)</code> <code>indexOf(s)</code> <code>indexOf(c)</code>	Index wo String <code>s</code> oder Zeichen <code>c</code> zum ersten Mal nach Index <code>fromIndex</code> im String auftaucht (-1 falls nicht gefunden); falls <code>fromIndex</code> 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 <code>s1.method(s2)</code>
<code>equals(s2)</code>	ob s1 und s2 die gleichen Buchstaben enthalten
<code>equalsIgnoreCase(s2)</code>	ob s1 und s2 die gleichen Buchstaben enthalten, ohne Berücksichtigung von Gross- und Kleinschreibung
<code>startsWith(s2)</code>	ob s1 mit den Buchstaben von s2 anfängt
<code>endsWith(s2)</code>	ob s1 mit den Buchstaben von s2 endet
<code>contains(s2)</code>	ob s1 irgendwo s2 enthält

Hinweis zu `==` und `equals`

- `==` nur für Basistypen (z.B. `int` oder `char`), nicht für Strings
- `==` geht auch für Strings, ist aber nicht erwünschenswert

Methoden in String Klasse

- `str.substring(begin, end)`: Gibt den Teilstring des Strings `str` zurück von Index `begin` (**inklusive**) und bis Index `end` (**exklusive**)
- `str.charAt(index)`: Gibt das Zeichen am Index `index` des Strings `str` zurück.

Methoden in String Klasse

- `str.toLowerCase()`: Konvertiert den String `str` in Kleinbuchstaben.
- `str.toUpperCase()`: Konvertiert den String `str` in Grossbuchstaben.
- `str.length()`: Gibt die Länge von dem String `str` zurück.

Methoden in String Klasse

- `str.replace(oldChar, newChar)`: Ersetzt **alle** Vorkommen des Zeichens `oldChar` im String `str` mit dem Zeichen `newChar`.
- `str.concat(str2)`: Äquivalent zu `str + str2`.
- `str.contains(someChar)`: Gibt zurück, ob `someChar` in `str` enthalten ist.

Methoden in String Klasse

- `str.indexOf(someChar)`: Gibt den Index des **ersten** Vorkommen des Zeichens `someChar` im String `str` zurück.
- `str.lastIndexOf(someChar)`: Gibt den Index des **letzten** Vorkommen des Zeichens `someChar` im String `str` zurück.

Wo findet man diese Informationen?

- In der Java Documentation 21



Java SE 21 & JDK 21

SEARCH

Java® Platform, Standard Edition & Java Development Kit Version 21 API Specification

This document is divided into two sections:

- Java SE
The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with `java`.
- JDK
The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with `jdk`.

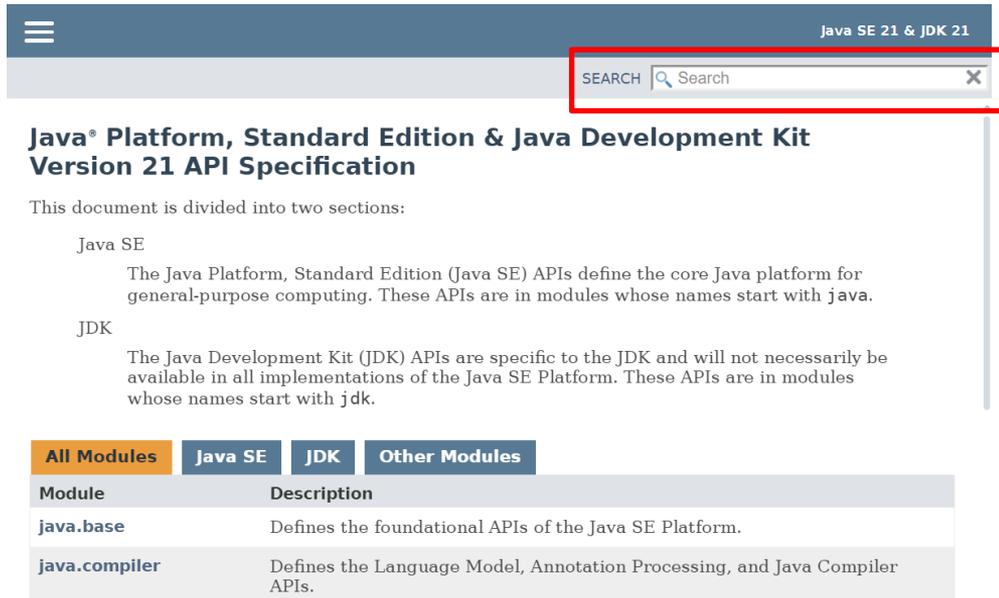
All Modules	Java SE	JDK	Other Modules
Module	Description		
<code>java.base</code>	Defines the foundational APIs of the Java SE Platform.		
<code>java.compiler</code>	Defines the Language Model, Annotation Processing, and Java Compiler APIs.		



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

Wo findet man diese Informationen?

- In der Java Documentation 21



Java SE 21 & JDK 21

SEARCH

Java® Platform, Standard Edition & Java Development Kit Version 21 API Specification

This document is divided into two sections:

Java SE

The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with `java`.

JDK

The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with `jdk`.

All Modules | **Java SE** | **JDK** | **Other Modules**

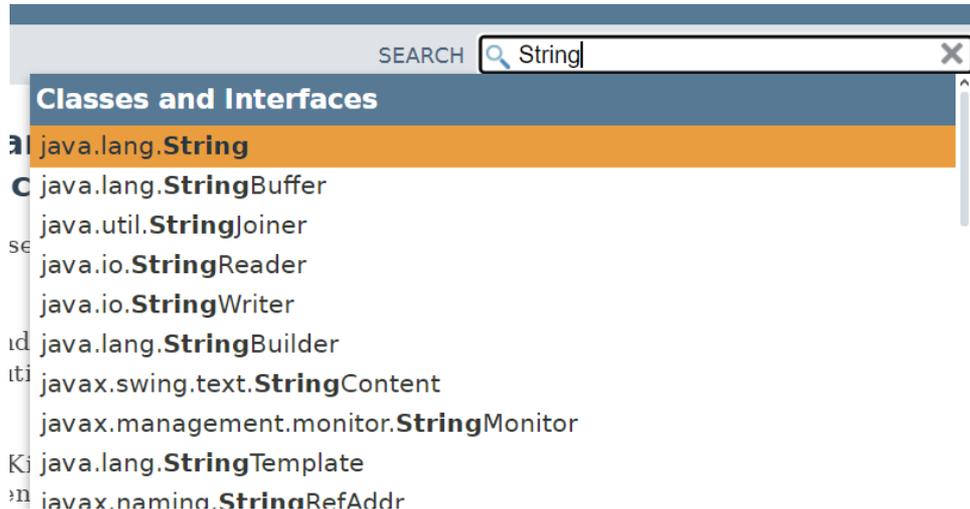
Module	Description
<code>java.base</code>	Defines the foundational APIs of the Java SE Platform.
<code>java.compiler</code>	Defines the Language Model, Annotation Processing, and Java Compiler APIs.



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

Wo findet man diese Informationen?

- In der Java Documentation 21



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.



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

Output:

""\\""

Aussengrenzen des Strings
Escape-Charakter
Im Resultat sichtbar

String Beispiele



```
1 public class MyClass{
2     public static void main(String[] args){
3         String str = "EPROG2024";
4         int result = str.indexOf('2') + str.length() * 2 - 10;
5     }
6 }
```

Output: 13

String Beispiele

```
1 public class MyClass{
2     public static void main(String[] args){
3         String str = "Einfuehrung";
4         String result = str.substring(3, 8).toUpperCase() + str.charAt(str.length() - 1);
5     }
6 }
```

Output: FUEHRg

String Beispiele



```
1 public class MyClass{
2     public static void main(String[] args){
3         String str = "Programmierung";
4         String result = str.replace('r', '*').substring(5, 12) + str.charAt(0);
5     }
6 }
```

Output: ammie*uP

String Beispiele

```
1 public class MyClass{
2     public static void main(String[] args){
3         String str1 = "Java21";
4         String str2 = "EPROG2024";
5         String result = str1.substring(0, 2) + str2.substring(str2.length() - 4);
6     }
7 }
```

Output: Ja2024

String Beispiele



```
1 public class MyClass{
2     public static void main(String[] args){
3         String str = "Substring";
4         int result = str.indexOf('S') + str.lastIndexOf('g') - str.length();
5     }
6 }
```

Output: -1

String Beispiele

```
1 public class MyClass{
2     public static void main(String[] args){
3         String str = "Einfuehrung";
4         String result = str.substring(2, 6).replace('f', 'X').concat(str.substring(6, 9));
5     }
6 }
```

Output: nXuehru

String Beispiele

```
1 public class MyClass{  
2     public static void main(String[] args){  
3         String str = "Programmierung";  
4         String result = str.charAt(0) + str.substring(2, 5).toLowerCase() + str.charAt(str.length() - 1);  
5     }  
6 }
```

Output: Pogrg

Tipps für die nächste Übung

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.

- **7 = 0111**
 - Weil $7 = 2^2 + 2^1 + 2^0 = 4 + 2 + 1 = 7$
- **13 = 8 + 4 + 1 = 2^3 + 2^2 + 2^0 = 1101**
- **16 = 2^4 > 13 > 8 = 2^3. -> Also reichen 4 bits**

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 $y = 44$:

$$\text{ggT}(36, 44) \stackrel{2.}{=} \text{ggT}(44, \underbrace{36 \% 44}_{36}) \stackrel{2.}{=} \text{ggT}(36, \underbrace{44 \% 36}_{8}) \stackrel{2.}{=} \text{ggT}(8, \underbrace{36 \% 8}_{4}) \stackrel{1.}{=} 4$$

DiskMat...

Iterative Variante [Bearbeiten | Quelltext bearbeiten]

EUCLID(a,b)

```
1 solange b ≠ 0
2   h ← Divisionsrest(a durch b) // Siehe Modulo-Funktion
3   a ← b
4   b ← h
5 //
6 Ergebnis = a
```

$$\bullet \text{ ggT} \begin{matrix} a & b \\ (16, 12) \end{matrix}$$

$$h = a \% b = 16 \% 12 = 4$$

$$a = 12$$

$$b = 4$$

nächste Iteration

$$h = a \% b = 12 \% 4 = 0$$

$$a = 4$$

$$b = 0$$

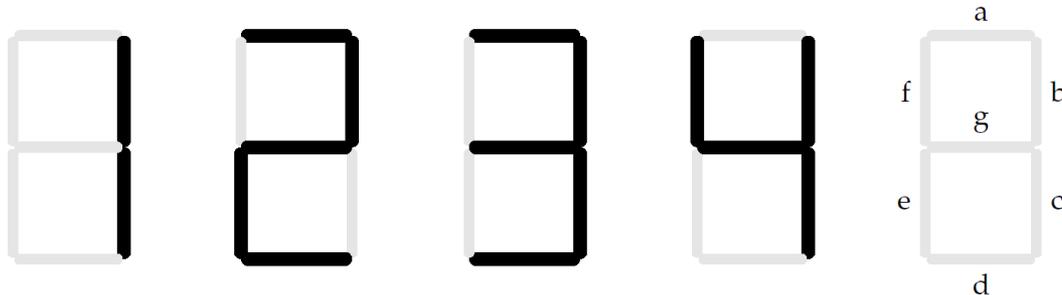
output : 4

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.



```
public static int dekodiereAnzeige(String encoding) {  
    //0,1,2,3,4,5,6,7,8,9  
  
    if (encoding.contains("f")) {  
        //0,4,5,6,8,9  
        return dekodiereFGruppe(encoding);  
    } else {  
        //1,2,3,7  
        return dekodiereKeineFGruppe(encoding);  
    }  
}
```

z.B. bc == 1

Geht durch alle Fälle durch und guckt
immer welche Zahl es sein kann

Aufgabe 4: 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 4: 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.

- **Teilt euch das ganze wieder in verschiedene Methoden auf**
- **Ihr müsst immer in Abhängigkeit von der Länge des Namens arbeiten**
 - `int n = name.length();`

Kahoot

Outro

- **Nutzt das Studycenter für Eprog**
- **Wenn ihr möchtet das ich bestimmte Aufgaben nochmal bespreche, dann schreibt mir gerne eure Wünsche**
- **Gerade kann alles noch etwas überwältigend sein, aber bleibt dran, ihr schafft das**

Extra

- HS22