DIE LIEBE ZUR REKURSION

Florian Sihler

Der Java-Stack, Funktionsaufrufe und Rekursion.
Ein wiederkehrendes Dilemma.

10. November 2022
SP, Universität Ulm

All rights reserved

DIE LIEBE ZUR REKURSION

Der Java-Stack, Funktionsaufrufe und Rekursion.
Ein wiederkehrendes Dilemma.

Florian Sihler

Officially supported by the Pingu-Foundation for Emotional Support. There will be light and there will be more peeps to come!

10. November 2022
SP, Universität Ulm

All rights reserved

2.1

Javas’ Speicherverwaltung

[5]: *The Java® Virtual Machine Specification [Heap & Stack]*
Lindholm, 2020

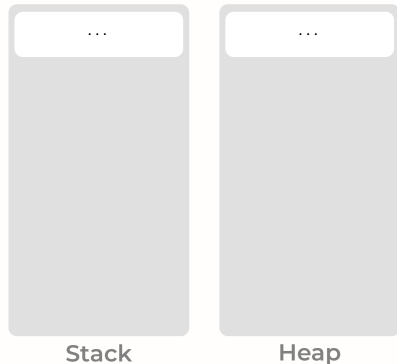
Javas' Speicherverwaltung

```java
int x = 0;
byte y = 12;
String f = "Hallo␣Welt";
char[] c = {2, 3, 4};
```

[5]: *The Java® Virtual Machine Specification [Heap & Stack]*
Lindholm, 2020

2.3 Javas' Speicherverwaltung

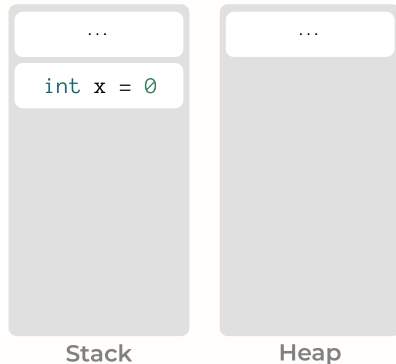
```java
int x = 0;
byte y = 12;
String f = "Hallo Welt";
char[] c = {2, 3, 4};
```



[5]: *The Java® Virtual Machine Specification [Heap & Stack]*
Lindholm, 2020

Javas' Speicherverwaltung

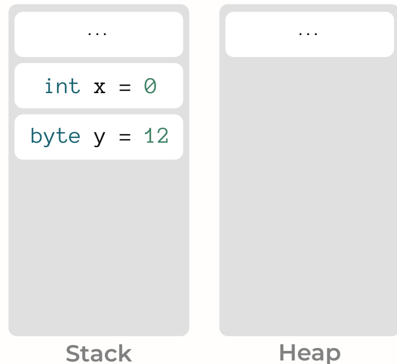
- ```java
  int x = 0;
  byte y = 12;
  String f = "Hallo Welt";
  char[] c = {2, 3, 4};
  ```



[5]: *The Java® Virtual Machine Specification [Heap & Stack]*
Lindholm, 2020
  ```

2.5 Javas' Speicherverwaltung

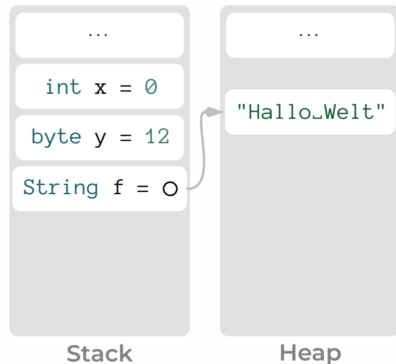
```
int x = 0;
byte y = 12;
String f = "Hallo␣Welt";
char[] c = {2, 3, 4};
```



[5]: *The Java® Virtual Machine Specification [Heap & Stack]*
Lindholm, 2020

Javas' Speicherverwaltung

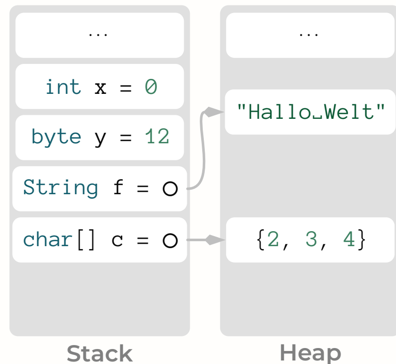
```
int x = 0;
byte y = 12;
String f = "Hallo␣Welt";
char[] c = {2, 3, 4};
```



[5]: *The Java® Virtual Machine Specification [Heap & Stack]*
Lindholm, 2020

Javas' Speicherverwaltung

```
int x = 0;
byte y = 12;
String f = "Hallo␣Welt";
char[] c = {2, 3, 4};
```



[5]: *The Java® Virtual Machine Specification [Heap & Stack]* Lindholm, 2020

3.1

Komplexere Konstrukte

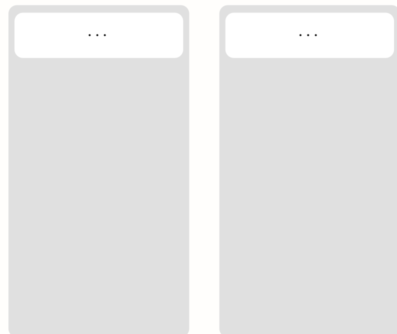
[5]: *The Java® Virtual Machine Specification [Heap & Stack]*
Lindholm, 2020

[8]: *LaTeX-Package, tikzpingus*
Sihler, 2021

Komplexere Konstrukte

```
class Mega {
    public int i;
    public String s;
}

Mega mon = new Mega();
mon.s = "Hallo";
mon.s = "Welt";
```

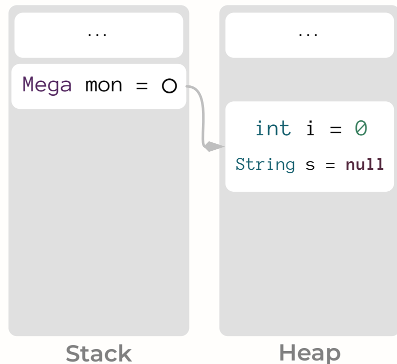


[5]: *The Java® Virtual Machine Specification [Heap & Stack]*
Lindholm, 2020

[8]: *LaTeX-Package, tikzpingus*
Sihler, 2021

Komplexere Konstrukte

```java
class Mega {
    public int i;
    public String s;
}

Mega mon = new Mega();
mon.s = "Hallo";
mon.s = "Welt";
```



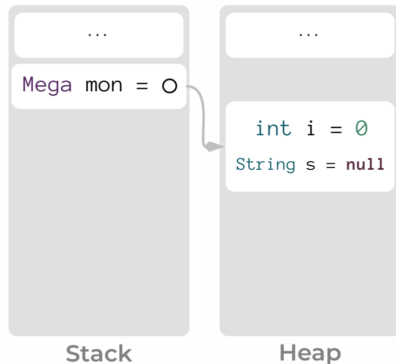
[5]: *The Java® Virtual Machine Specification [Heap & Stack]*
Lindholm, 2020

[8]: *LaTeX-Package, tikzpingus*
Sihler, 2021

Komplexere Konstrukte

Beachte: Alle Variablen erhalten in Java „Default-Werte". Für komplexe Datentypen ist dies `null`.

```java
class Mega {
    public int i;
    public String s;
}

Mega mon = new Mega();
mon.s = "Hallo";
mon.s = "Welt";
```



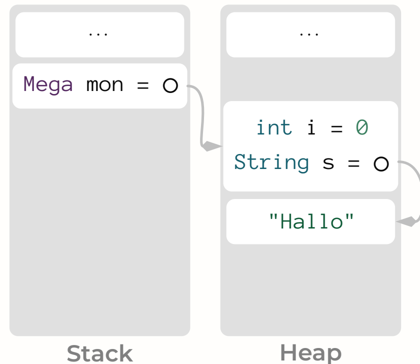
[5]: *The Java® Virtual Machine Specification [Heap & Stack]* Lindholm, 2020

[8]: *LaTeX-Package, tikzpingus* Sihler, 2021

Komplexere Konstrukte

Beachte: Alle Variablen erhalten in Java „Default-Werte“. Für komplexe Datentypen ist dies **null**.

```
class Mega {
    public int i;
    public String s;
}

Mega mon = new Mega();
mon.s = "Hallo";
mon.s = "Welt";
```



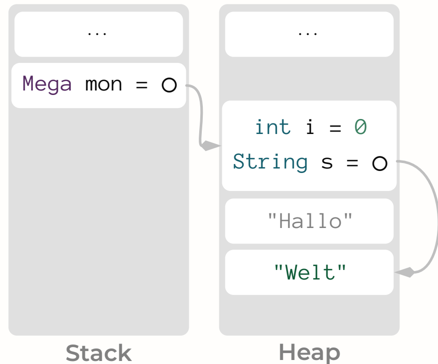
[5]: *The Java® Virtual Machine Specification [Heap & Stack]* Lindholm, 2020

[8]: *LaTeX-Package, tikzpingus* Sihler, 2021

Komplexere Konstrukte

Beachte: Alle Variablen erhalten in Java „Default-Werte“. Für komplexe Datentypen ist dies **null**.

```java
class Mega {
    public int i;
    public String s;
}

Mega mon = new Mega();
mon.s = "Hallo";
mon.s = "Welt";
```



[5]: *The Java® Virtual Machine Specification [Heap & Stack]*
Lindholm, 2020

[8]: *LaTeX-Package, tikzpingus*
Sihler, 2021

4.1 Stack Frames

[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

4.2 Stack Frames

- Handhaben (lokale) Gültigkeitsbereiche.

[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021

Stack Frames

- Handhaben (lokale) Gültigkeitsbereiche.
- Werden beim Betreten auf dem Stack abgelegt.

[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

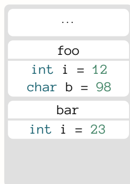
Stack Frames

- Handhaben (lokale) Gültigkeitsbereiche.
- Werden beim Betreten auf dem Stack abgelegt.
- Werden beim Verlassen wieder vom Stack entnommen.

[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021

Stack Frames

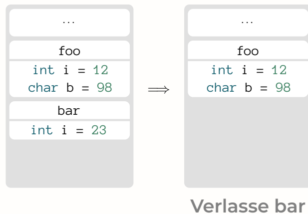
- Handhaben (lokale) Gültigkeitsbereiche.
- Werden beim Betreten auf dem Stack abgelegt.
- Werden beim Verlassen wieder vom Stack entnommen.



[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

Stack Frames

- Handhaben (lokale) Gültigkeitsbereiche.
- Werden beim Betreten auf dem Stack abgelegt.
- Werden beim Verlassen wieder vom Stack entnommen.

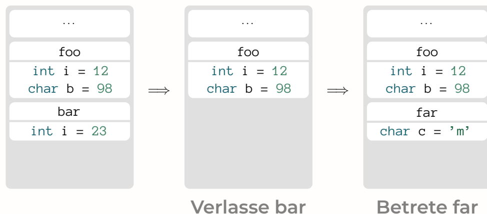


Verlasse bar

[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

Stack Frames

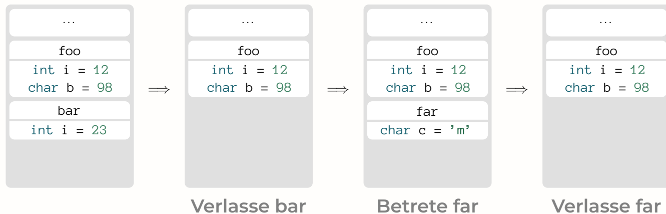
- Handhaben (lokale) Gültigkeitsbereiche.
- Werden beim Betreten auf dem Stack abgelegt.
- Werden beim Verlassen wieder vom Stack entnommen.



[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

Stack Frames

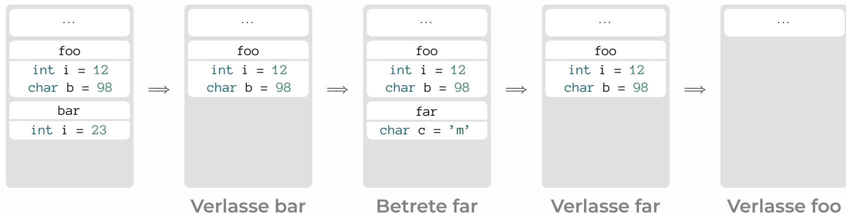
- Handhaben (lokale) Gültigkeitsbereiche.
- Werden beim Betreten auf dem Stack abgelegt.
- Werden beim Verlassen wieder vom Stack entnommen.



[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

Stack Frames

- Handhaben (lokale) Gültigkeitsbereiche.
- Werden beim Betreten auf dem Stack abgelegt.
- Werden beim Verlassen wieder vom Stack entnommen.



[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021

5.1 Stack Frames – Beispiel

[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021

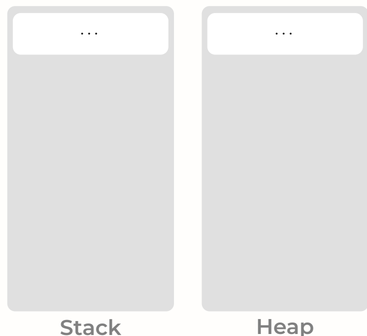
Stack Frames – Beispiel

```java
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```

[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

Stack Frames – Beispiel

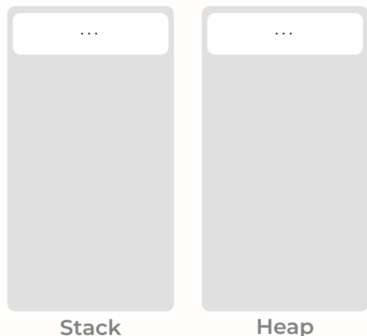
```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

Stack Frames – Beispiel

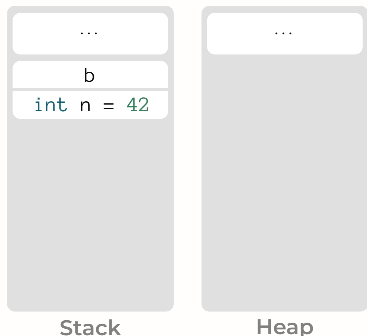
```java
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

5.5 Stack Frames – Beispiel

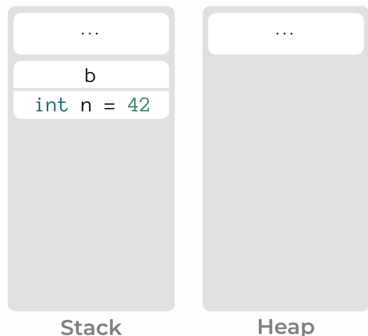
```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021

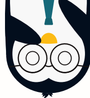
5.6 Stack Frames – Beispiel

```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



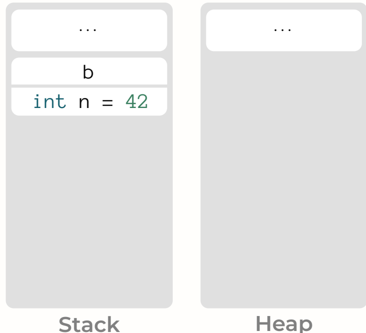
[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

Wir werden Rückgabemechanismen sträflich vereinfachen!



5.7 Stack Frames – Beispiel

```java
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```

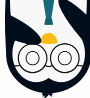


[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021

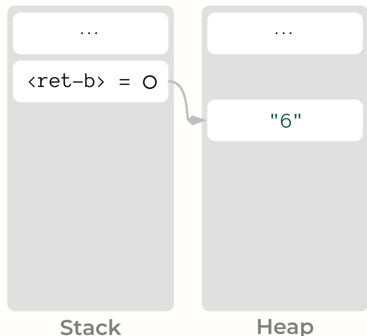
5.8

Stack Frames – Beispiel

Wir werden Rückgabemechanismen sträflich vereinfachen!



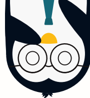
```java
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



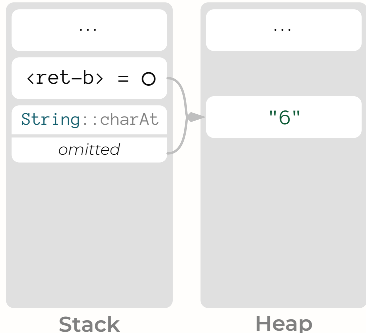
[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021

Stack Frames – Beispiel

Wir werden Rückgabemecha-
nismen sträflich vereinfachen!

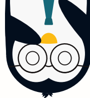


```java
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021

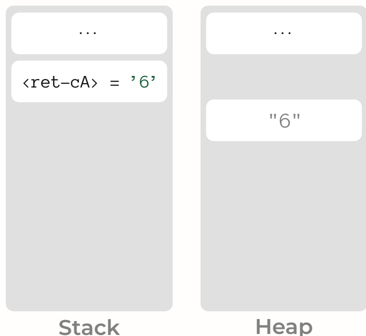
Stack Frames – Beispiel



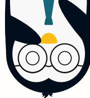
Wir werden Rückgabemecha-nismen sträflich vereinfachen!

```java
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}
```

- `f(b(42).charAt(0));`



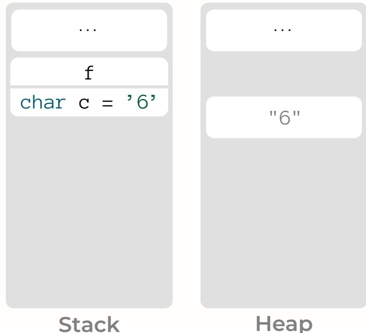
[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021



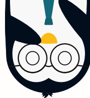
Wir werden Rückgabemecha-
nismen sträflich vereinfachen!

5.11 Stack Frames – Beispiel

```java
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



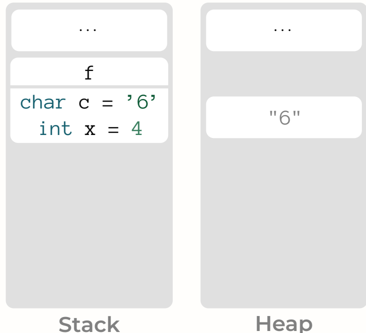
[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021



Wir werden Rückgabemecha-
nismen sträflich vereinfachen!

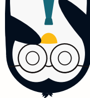
5.12 Stack Frames – Beispiel

```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021

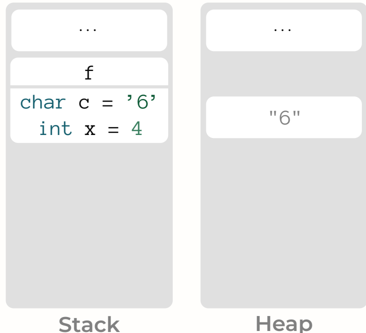
Wir werden Rückgabemechanismen sträflich vereinfachen!



5.13

Stack Frames – Beispiel

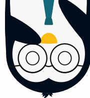
```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



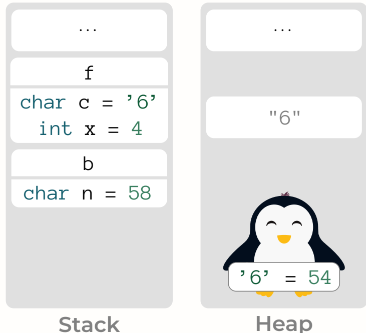
[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

Stack Frames – Beispiel

Wir werden Rückgabemechanismen sträflich vereinfachen!



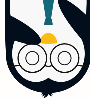
```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



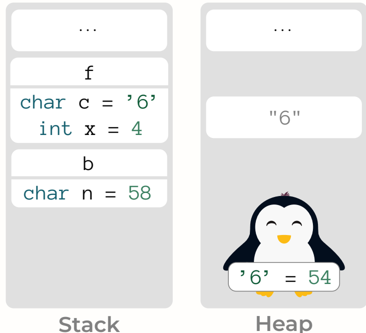
[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

Stack Frames – Beispiel

Wir werden Rückgabemechanismen sträflich vereinfachen!



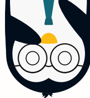
```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



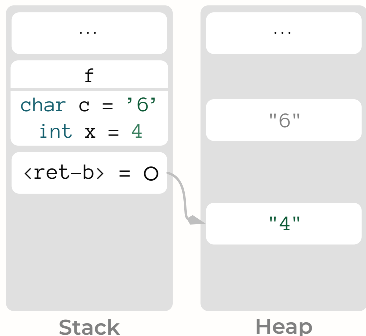
[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021

Stack Frames – Beispiel

Wir werden Rückgabemechanismen sträflich vereinfachen!



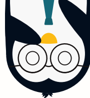
```java
void f(char c) {
  int x = 4;
  System.out.print(b(c+x)•);
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



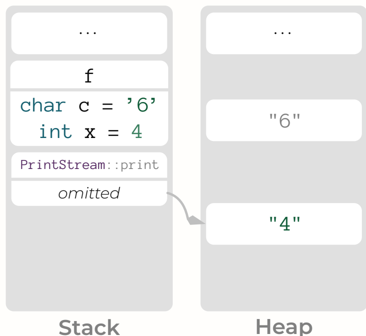
[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021

Stack Frames – Beispiel

Wir werden Rückgabemecha-nismen sträflich vereinfachen!



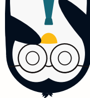
```java
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

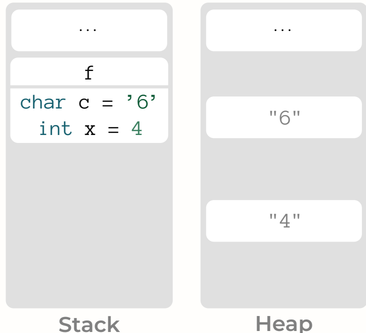
Stack Frames – Beispiel

Wir werden Rückgabemecha-
nismen sträflich vereinfachen!



```java
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



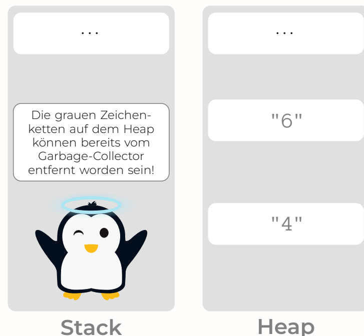
[3]: *Java Debug Interface – StackFrame (JDK 11)* , 2021

Stack Frames – Beispiel

Wir werden Rückgabemecha-
nismen sträflich vereinfachen!

```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



[3]: *Java Debug Interface – StackFrame (JDK 11)*, 2021

6.1 Anatomie einer Rekursion

6.2 Anatomie einer Rekursion

```
1  func recursive(problem arguments) is

9  end
```

Anatomie einer Rekursion

```
1 func recursive(problem arguments) is
2     if at elemental problem then

4     else




8     end
9 end
```

Anatomie einer Rekursion

```
1 func recursive(problem arguments) is
2     if at elemental problem then
3         solve elemental problem;
4     else



8     end
9 end
```

6.5 Anatomie einer Rekursion

```
1 func recursive(problem arguments) is
2     if at elemental problem then
3         solve elemental problem;
4     else
5         Split problem into smaller ones;



8     end
9 end
```

Anatomie einer Rekursion

```
1 func recursive(problem arguments) is
2     if at elemental problem then
3         solve elemental problem;
4     else
5         Split problem into smaller ones;
6         Try to solve them recursively;

8     end
9 end
```

Anatomie einer Rekursion

```
1 func recursive(problem arguments) is
2     if at elemental problem then
3         solve elemental problem;
4     else
5         Split problem into smaller ones;
6         Try to solve them recursively;
7         Combine step-solutions to one;
8     end
9 end
```

Anatomie einer Rekursion

```
1 func recursive(problem arguments) is
2     if at elemental problem then
3         solve elemental problem;
4     else
5         Split problem into smaller ones;
6         Try to solve them recursively;
7         Combine step-solutions to one;
8     end
9 end
```

Nicht alle müssen genau dieser Struktur folgen.
Be special. Be wild!

7.1 Stack Frames und Aufrufgraphen

7.2

Stack Frames und Aufrufgraphen

- Stack Frames werden schnell unübersichtlich.

7.3 Stack Frames und Aufrufgraphen

- Stack Frames werden schnell unübersichtlich.
- Wir vereinfachen dies durch Aufrufgraphen.

Stack Frames und Aufrufgraphen

- Stack Frames werden schnell unübersichtlich.
- Wir vereinfachen dies durch Aufrufgraphen.
- Knoten repräsentieren Stack Frames.

7.5

Stack Frames und Aufrufgraphen

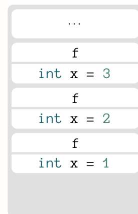
- Stack Frames werden schnell unübersichtlich.
- Wir vereinfachen dies durch Aufrufgraphen.
- Knoten repräsentieren Stack Frames.

```
int f(int x) {
   return f(x - 1);
}
```

Stack Frames und Aufrufgraphen

- Stack Frames werden schnell unübersichtlich.
- Wir vereinfachen dies durch Aufrufgraphen.
- Knoten repräsentieren Stack Frames.

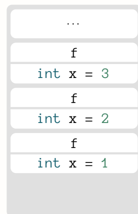
```
int f(int x) {
    return f(x - 1);
}
```

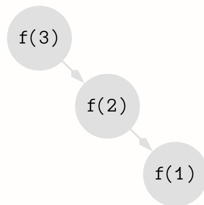


Stack Frames und Aufrufgraphen

- Stack Frames werden schnell unübersichtlich.
- Wir vereinfachen dies durch Aufrufgraphen.
- Knoten repräsentieren Stack Frames.

```
int f(int x) {
    return f(x - 1);
}
```





Lineare Rekursion

[2]: *About primitive recursive algorithms* Colson, 1991

8.2 Lineare Rekursion

- ```
  int f(int n) {
    if (n <= 0) return n;
    else return n + f(n - 1);
  }
  ```

[2]: *About primitive recursive algorithms* Colson, 1991
  ```

Lineare Rekursion

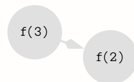
```
int f(int n) {
  if (n <= 0) return n;
  else return n + f(n - 1);
}
```

f(3)

[2]: *About primitive recursive algorithms* Colson, 1991

8.4 Lineare Rekursion

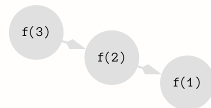
```
int f(int n) {
  if (n <= 0) return n;
  else return n + f(n - 1);
}
```



[2]: *About primitive recursive algorithms* Colson, 1991

Lineare Rekursion

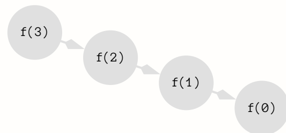
```
int f(int n) {
  if (n <= 0) return n;
  else return n + f(n - 1);
}
```



[2]: *About primitive recursive algorithms* Colson, 1991

8.6 Lineare Rekursion

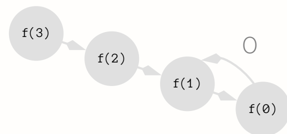
```java
int f(int n) {
  if (n <= 0) return n;
  else return n + f(n - 1);
}
```



[2]: *About primitive recursive algorithms* Colson, 1991

8.7 Lineare Rekursion

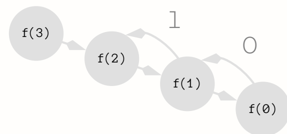
```
int f(int n) {
  if (n <= 0) return n;
  else return n + f(n - 1);
}
```



[2]: *About primitive recursive algorithms* Colson, 1991

Lineare Rekursion

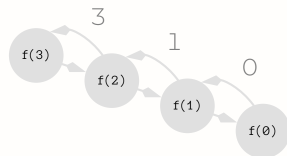
```java
int f(int n) {
  if (n <= 0) return n;
  else return n + f(n - 1);
}
```



[2]: *About primitive recursive algorithms* Colson, 1991

Lineare Rekursion

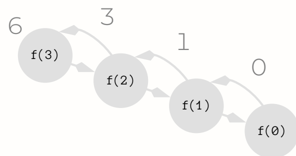
```
int f(int n) {
  if (n <= 0) return n;
  else return n + f(n - 1);
}
```



[2]: *About primitive recursive algorithms* Colson, 1991

Lineare Rekursion

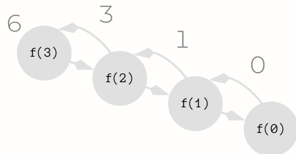
```
int f(int n) {
  if (n <= 0) return n;
  else return n + f(n - 1);
}
```



[2]: *About primitive recursive algorithms* Colson, 1991

Lineare Rekursion

```
int f(int n) {
  if (n <= 0) return n;
  else return n + f(n - 1);
}
```



Die einfachste und geläufigste Form der Rekursion.

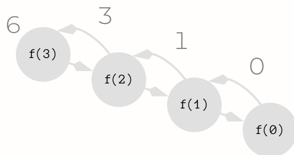
Hier handelt es sich sogar um eine *primitiv-rekursive* Funktion![2]

[2]: *About primitive recursive algorithms* Colson, 1991

Lineare Rekursion

Lineare Rekursion wird weiter unterteilt! So beispielweise in Head- und Tailrekursiv.

```
int f(int n) {
  if (n <= 0) return n;
  else return n + f(n - 1);
}
```



Die einfachste und geläufigste Form der Rekursion.

Hier handelt es sich sogar um eine *primitiv-rekursive* Funktion![2]

[2]: *About primitive recursive algorithms* Colson, 1991

Verschachtelte Rekursion

[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study*
Sundblad, 1971

Verschachtelte Rekursion

Dies ist die Acker-mann Funktion!

```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```

[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study* Sundblad, 1971

Verschachtelte Rekursion

`a(1, 1)`

```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```

[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study*
Sundblad, 1971

Verschachtelte Rekursion

```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```

a(1, 1)

a(1, 0)

[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study*
Sundblad, 1971

9.5 Verschachtelte Rekursion

```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```

a(1, 1)

a(1, 0)

[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study*
Sundblad, 1971

Verschachtelte Rekursion

Dies ist die Ackermann Funktion!

```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```

a(1, 1)

a(1, 0)

a(0, 1)

[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study* Sundblad, 1971

Verschachtelte Rekursion

```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```

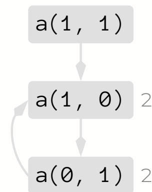
a(1, 1)
a(1, 0)
a(0, 1) 2

[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study* Sundblad, 1971

Verschachtelte Rekursion

Dies ist die Ackermann Funktion!

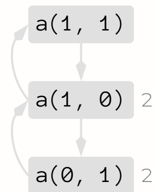
```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```



[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study* Sundblad, 1971

Verschachtelte Rekursion

```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```

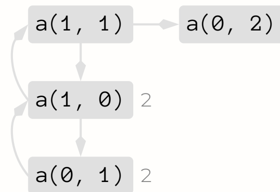


[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study*
Sundblad, 1971

Verschachtelte Rekursion

Dies ist die Acker-mann Funktion!

```java
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```

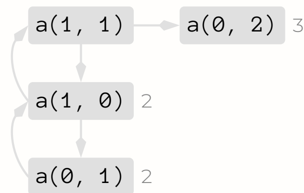


[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study* Sundblad, 1971

Verschachtelte Rekursion

Dies ist die Ackermann Funktion!

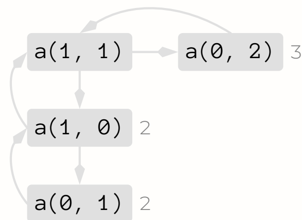
```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```



[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study*
Sundblad, 1971

Verschachtelte Rekursion

```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```

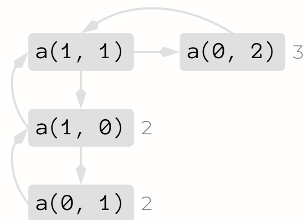


[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study* Sundblad, 1971

Verschachtelte Rekursion

Dies ist die Ackermann Funktion!

```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```

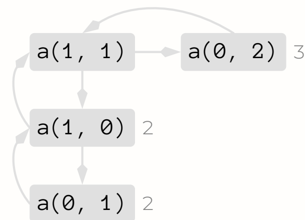


Damit ist $a(1, 1) = 3$.

[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study*
Sundblad, 1971

Verschachtelte Rekursion

```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```



Damit ist $a(1, 1) = 3$.

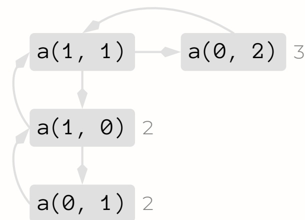
Was ist dann beispielsweise $a(4, 2)$?

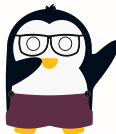
[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study* Sundblad, 1971

Verschachtelte Rekursion

Dies ist die Ackermann Funktion!

```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```





Damit ist $a(1, 1) = 3$.

Was ist dann beispielsweise $a(4, 2)$?

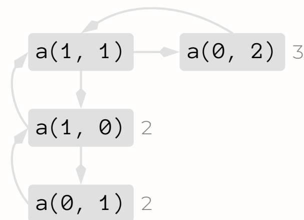
Nun... $a(4, 2) \approx 2 \cdot 10^{19\,729}$.

[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study* Sundblad, 1971

Verschachtelte Rekursion

Dies ist die Ackermann Funktion!

```
int a(int m, int n) {
  if(m == 0) return n + 1;
  else if (n == 0) return a(m - 1, 1);
  else return a(m - 1, a(m, n - 1));
}
```



Verschachtelt-Rekursive Funktionen sind meist sehr schwer nachzuvollziehen.

Damit ist $a(1,1) = 3$.

Was ist dann beispielsweise $a(4,2)$?

Nun... $a(4,2) \approx 2 \cdot 10^{19\,729}$.

[9]: *The Ackermann function. a theoretical, computational, and formula manipulative study* Sundblad, 1971

Kaskadenförmige Rekursion

[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

10.2 Kaskadenförmige Rekursion

```
int f(int n) {
   if(n <= 2) return 1;
   else return f(n - 1) + f(n - 2);
}
```

[7]: *Exploring Recursion with Fibonacci Numbers* Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

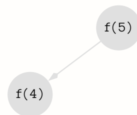
f(5)

- ```
  int f(int n) {
      if(n <= 2) return 1;
      else return f(n - 1) + f(n - 2);
  }
  ```

[7]: *Exploring Recursion with Fibonacci Numbers* Rubio-Sánchez, 2007
  ```

Kaskadenförmige Rekursion

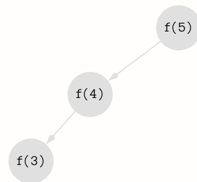
```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

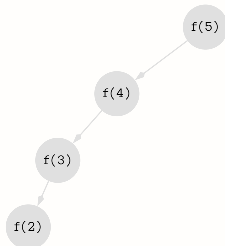
```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

10.6 Kaskadenförmige Rekursion

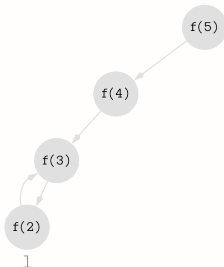
```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

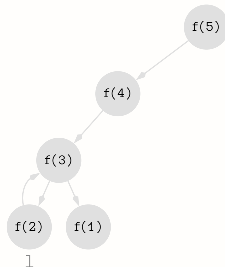
```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

10.8 Kaskadenförmige Rekursion

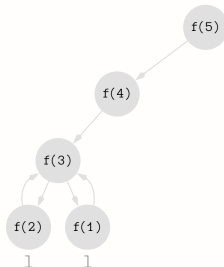
```
int f(int n) {
   if(n <= 2) return 1;
   else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

10.9 Kaskadenförmige Rekursion

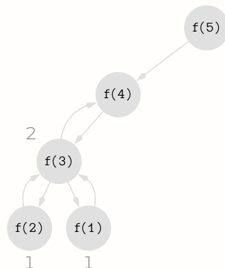
```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers* Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

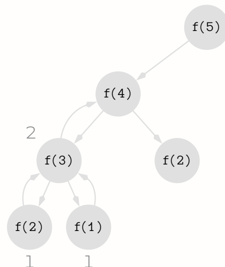
```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

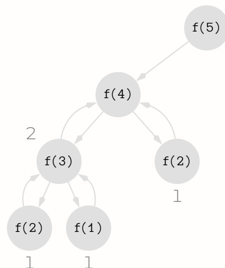
```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers* Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

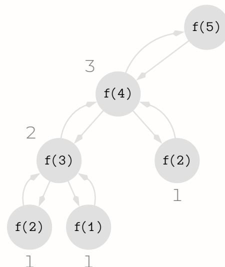
```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

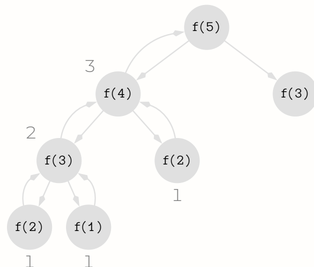
```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

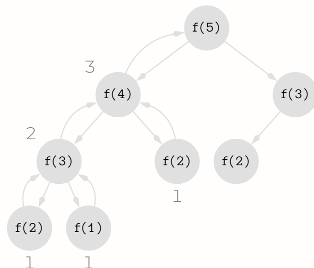
```java
int f(int n) {
   if(n <= 2) return 1;
   else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

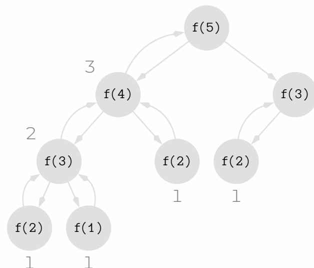
```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

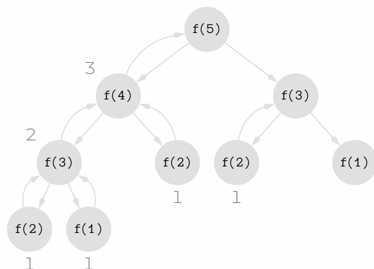
```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

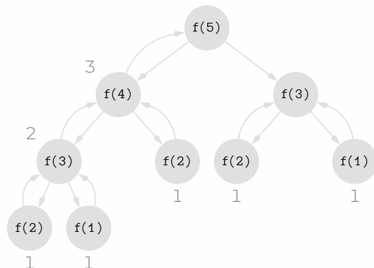
```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

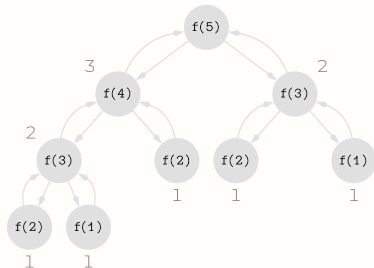
```java
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

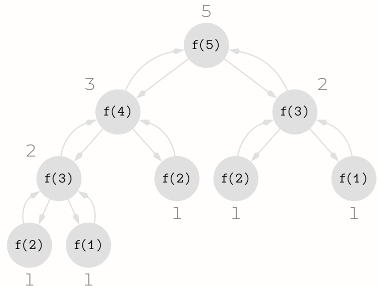
```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

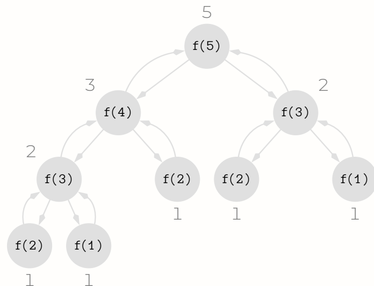
```java
int f(int n) {
   if(n <= 2) return 1;
   else return f(n - 1) + f(n - 2);
}
```



[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

```
int f(int n) {
    if(n <= 2) return 1;
    else return f(n - 1) + f(n - 2);
}
```



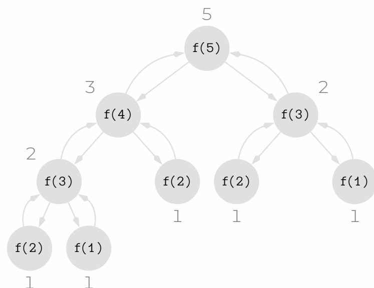
Elegant, aber meist
nicht sonderlich effizient.

[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

Kaskadenförmige Rekursion

Sie wird manchmal auch „ver-zweigte Rekursion“ genannt.

```
int f(int n) {
   if(n <= 2) return 1;
   else return f(n - 1) + f(n - 2);
}
```



Elegant, aber meist
nicht sonderlich effizient.

[7]: *Exploring Recursion with Fibonacci Numbers*
Rubio-Sánchez, 2007

11.1

Wechselseitige Rekursion

Wechselseitige Rekursion

```
int a(int x) {
   if(x <= 0) return x + 1;
   return b(x - 1);
}

int b(int x) {
   if(x <= 0) return x;
   return a(x % 3);
}
```

Wechselseitige Rekursion

Sie kann durch Funktionstupel stets in „normale“ Rekursionen transformiert werden.

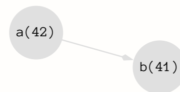
```
int a(int x) {
    if(x <= 0) return x + 1;
    return b(x - 1);
}

int b(int x) {
    if(x <= 0) return x;
    return a(x % 3);
}
```

a(42)

Wechselseitige Rekursion

Sie kann durch Funktionstupel stets in „normale“ Rekursionen transformiert werden.

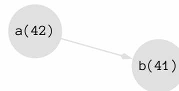
```
int a(int x) {
    if(x <= 0) return x + 1;
    return b(x - 1);
}

int b(int x) {
    if(x <= 0) return x;
    return a(x % 3);
}
```



11.5 Wechselseitige Rekursion

Sie kann durch Funktionstupel stets in „normale“ Rekursionen transformiert werden.

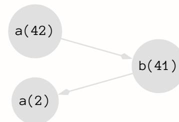
```
int a(int x) {
    if(x <= 0) return x + 1;
    return b(x - 1);
}

int b(int x) {
    if(x <= 0) return x;
    return a(x % 3);
}
```



Wechselseitige Rekursion

Sie kann durch Funktionstupel stets in „normale“ Rekursionen transformiert werden.

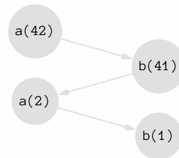
```
int a(int x) {
    if(x <= 0) return x + 1;
    return b(x - 1);
}

int b(int x) {
    if(x <= 0) return x;
    return a(x % 3);
}
```



Wechselseitige Rekursion

Sie kann durch Funktionstupel stets in „normale“ Rekursionen transformiert werden.

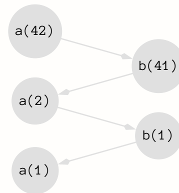
```
int a(int x) {
    if(x <= 0) return x + 1;
    return b(x - 1);
}

int b(int x) {
    if(x <= 0) return x;
    return a(x % 3);
}
```



Wechselseitige Rekursion

Sie kann durch Funktionstupel stets in „normale“ Rekursionen transformiert werden.

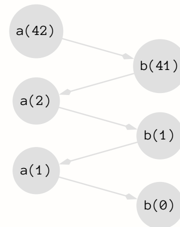
```
int a(int x) {
    if(x <= 0) return x + 1;
    return b(x - 1);
}

int b(int x) {
    if(x <= 0) return x;
    return a(x % 3);
}
```



Wechselseitige Rekursion

Sie kann durch Funktionstupel stets in „normale“ Rekursionen transformiert werden.

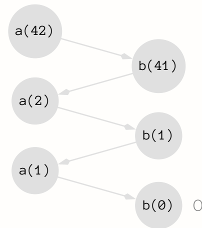
```
int a(int x) {
    if(x <= 0) return x + 1;
    return b(x - 1);
}

int b(int x) {
    if(x <= 0) return x;
    return a(x % 3);
}
```



Wechselseitige Rekursion

Sie kann durch Funktionstupel stets in „normale“ Rekursionen transformiert werden.

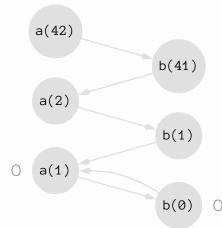
```
int a(int x) {
    if(x <= 0) return x + 1;
    return b(x - 1);
}

int b(int x) {
    if(x <= 0) return x;
    return a(x % 3);
}
```



Wechselseitige Rekursion

Sie kann durch Funktionstupel stets in „normale“ Rekursionen transformiert werden.

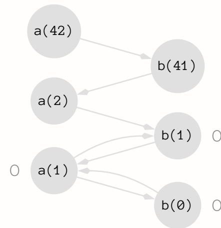
```
int a(int x) {
    if(x <= 0) return x + 1;
    return b(x - 1);
}

int b(int x) {
    if(x <= 0) return x;
    return a(x % 3);
}
```



Wechselseitige Rekursion

Sie kann durch Funktionstupel stets in „normale“ Rekursionen transformiert werden.

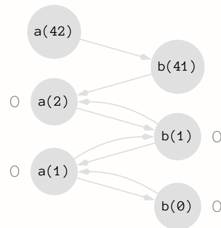
```
int a(int x) {
    if(x <= 0) return x + 1;
    return b(x - 1);
}

int b(int x) {
    if(x <= 0) return x;
    return a(x % 3);
}
```



Wechselseitige Rekursion

Sie kann durch Funktionstupel stets in „normale“ Rekursionen transformiert werden.

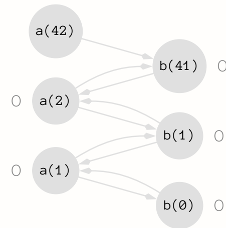
```
int a(int x) {
    if(x <= 0) return x + 1;
    return b(x - 1);
}

int b(int x) {
    if(x <= 0) return x;
    return a(x % 3);
}
```



Wechselseitige Rekursion

Sie kann durch Funktionstupel stets in „normale“ Rekursionen transformiert werden.

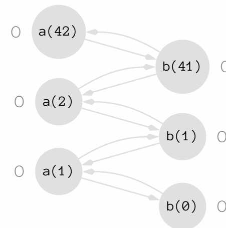
```
int a(int x) {
    if(x <= 0) return x + 1;
    return b(x - 1);
}

int b(int x) {
    if(x <= 0) return x;
    return a(x % 3);
}
```



Wechselseitige Rekursion

Sie kann durch Funktionstupel stets in „normale“ Rekursionen transformiert werden.

```
int a(int x) {
    if(x <= 0) return x + 1;
    return b(x - 1);
}

int b(int x) {
    if(x <= 0) return x;
    return a(x % 3);
}
```



12.1 Ein paar Notizen

[6]: *From Recursion to Iteration: What Are the Optimizations?*
Liu, 1999

Ein paar Notizen

- Die gezeigten Beispiele waren möglichst minimal

[6]: *From Recursion to Iteration: What Are the Optimizations?*
Liu, 1999

Ein paar Notizen

- Die gezeigten Beispiele waren möglichst minimal
- Die Varianten können kombiniert werden!

[6]: *From Recursion to Iteration: What Are the Optimizations?* Liu, 1999

12.4

Ein paar Notizen

- Die gezeigten Beispiele waren möglichst minimal
- Die Varianten können kombiniert werden!
  - z.B. eine kaskadenförmige, wechselseitige Rekursion.

[6]: *From Recursion to Iteration: What Are the Optimizations?* Liu, 1999

Ein paar Notizen

- Die gezeigten Beispiele waren möglichst minimal
- Die Varianten können kombiniert werden!
  - z.B. eine kaskadenförmige, wechselseitige Rekursion.
- Rekursion und Iteration sind meist gleich mächtig.

[6]: *From Recursion to Iteration: What Are the Optimizations?* Liu, 1999

13.1 Head- vs. Tail-Rekursion

13.2

Head- vs. Tail-Rekursion

```java
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```

```java
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```

13.3

Head- vs. Tail-Rekursion

```java
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```

h(3)

```java
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```

t(3)

13.4 Head- vs. Tail-Rekursion

```java
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```

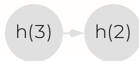
```java
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```

h(3)

t(3)

13.5 Head- vs. Tail-Rekursion

```java
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```

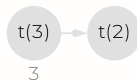



```java
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```

Head- vs. Tail-Rekursion

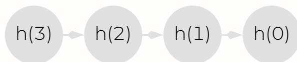
```java
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```

```java
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```

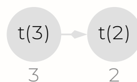


13.7 Head- vs. Tail-Rekursion

```java
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```



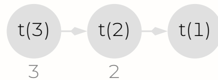

```java
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```



13.8 Head- vs. Tail-Rekursion

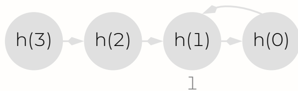
```java
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```

```java
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```

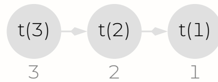


Head- vs. Tail-Rekursion

```
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```

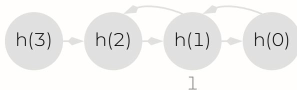



```
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```

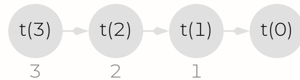


Head- vs. Tail-Rekursion

```java
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```



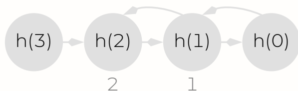

```java
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```

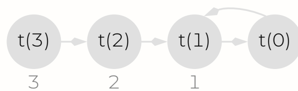


Head- vs. Tail-Rekursion

```java
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```

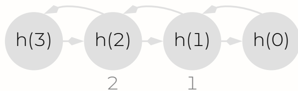
```java
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```



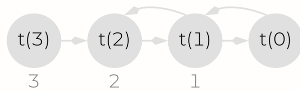


Head- vs. Tail-Rekursion

```java
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```



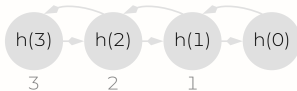

```java
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```

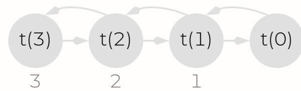


Head- vs. Tail-Rekursion

```java
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```

```java
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```

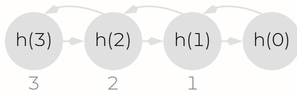


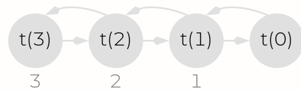


Head- vs. Tail-Rekursion

```
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```

```
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```





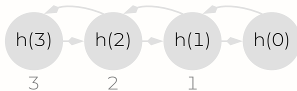
Ausgabe: $1 \rightarrow 2 \rightarrow 3$

Ausgabe: $3 \rightarrow 2 \rightarrow 1$

Head- vs. Tail-Rekursion

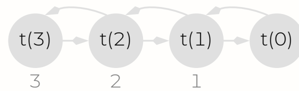
Kurzgesagt: Passiert noch etwas im rekursiven Auf-stieg oder nicht (tail).

```java
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```



Ausgabe: $1 \to 2 \to 3$

```java
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```

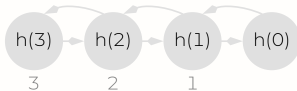


Ausgabe: $3 \to 2 \to 1$

Head- vs. Tail-Rekursion

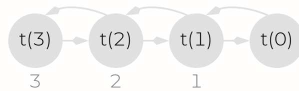
Kurzgesagt: Passiert noch etwas im rekursiven Aufstieg oder nicht (tail).

```
void h(int n) {
    if(n <= 0) return;
    h(n - 1);
    System.out.println(n);
}
```



Ausgabe: $1 \to 2 \to 3$

```
void t(int n) {
    if(n <= 0) return;
    System.out.println(n);
    t(n - 1);
}
```



Ausgabe: $3 \to 2 \to 1$

Disclaimer: Die Feinheiten einer Head-Rekursion werden hier vernachlässigt. Tail-Rekursionen sind interessanter, da sie sich problemlos in Iterationen transformieren lassen.

14.1 Mathematische Funktionen

Mathematische Funktionen

- Einige mathematischen Funktionen sind rekursiv definiert.

Mathematische Funktionen

- Einige mathematischen Funktionen sind rekursiv definiert.
- Die gegebene Definition lässt sich meist `1:1` übersetzen.

Mathematische Funktionen

- Einige mathematischen Funktionen sind rekursiv definiert.
- Die gegebene Definition lässt sich meist `1:1` übersetzen.
- Sie nimmt bereits einiges des Denkaufwands ab.

14.5 Mathematische Funktionen

- Einige mathematischen Funktionen sind rekursiv definiert.
- Die gegebene Definition lässt sich meist `1:1` übersetzen.
- Sie nimmt bereits einiges des Denkaufwands ab.

$$f(n) = \begin{cases} 1, & n \leq 1 \\ n \cdot f(n-1), & n > 1 \end{cases}$$

$$f_n : \mathbb{N} \to \mathbb{N}, n \mapsto n \cdot f_{n-1}$$

$$f_1 = 1$$

Mathematische Funktionen

- Einige mathematischen Funktionen sind rekursiv definiert.
- Die gegebene Definition lässt sich meist 1:1 übersetzen.
- Sie nimmt bereits einiges des Denkaufwands ab.

$$f(n) = \begin{cases} 1, & n \leq 1 \\ n \cdot f(n-1), & n > 1 \end{cases}$$

$$f_n : \mathbb{N} \to \mathbb{N}, n \mapsto n \cdot f_{n-1}$$

$$f_1 = 1$$

```java
public int f(int n) {

}
```

Mathematische Funktionen

- Einige mathematischen Funktionen sind rekursiv definiert.
- Die gegebene Definition lässt sich meist 1:1 übersetzen.
- Sie nimmt bereits einiges des Denkaufwands ab.

$$f(n) = \begin{cases} 1, & n \leq 1 \\ n \cdot f(n-1), & n > 1 \end{cases}$$

$$f_n : \mathbb{N} \to \mathbb{N}, n \mapsto n \cdot f_{n-1}$$

$$f_1 = 1$$

```java
public int f(int n) {
  if(n <= 1) return 1;

}
```

Mathematische Funktionen

- Einige mathematischen Funktionen sind rekursiv definiert.
- Die gegebene Definition lässt sich meist 1:1 übersetzen.
- Sie nimmt bereits einiges des Denkaufwands ab.

$$f(n) = \begin{cases} 1, & n \leq 1 \\ n \cdot f(n-1), & n > 1 \end{cases}$$

$$f_n : \mathbb{N} \to \mathbb{N}, n \mapsto n \cdot f_{n-1}$$

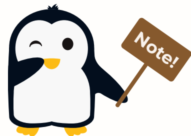
$$f_1 = 1$$

```java
public int f(int n) {
  if(n <= 1) return 1;
  else return n * f(n - 1);
}
```

Mathematische Funktionen

- Einige mathematischen Funktionen sind rekursiv definiert.
- Die gegebene Definition lässt sich meist 1:1 übersetzen.
- Sie nimmt bereits einiges des Denkaufwands ab.

„Elementarprobleme“ sind hier die nicht-rekursiven Fälle: Teile der Folge, welche oft axiomatisch gegeben sind.



$$f(n) = \begin{cases} 1, & n \leq 1 \\ n \cdot f(n-1), & n > 1 \end{cases}$$

$$f_n : \mathbb{N} \to \mathbb{N}, n \mapsto n \cdot f_{n-1}$$
$$f_1 = 1$$

```
public int f(int n) {
  if(n <= 1) return 1;
  else return n * f(n - 1);
}
```

15.1 Die Ackermann Funktion, Alternativ

15.2 Die Ackermann Funktion, Alternativ

$$f(n) = \varphi(n, n, n)$$

Die Ackermann Funktion, Alternativ

$$f(n) = \varphi(n, n, n)$$

$$\varphi(a, b, 0) = a + b$$
$$\varphi(a, 0, n) = \alpha(a, n - 1)$$
$$\varphi(a, b, n) = \varphi(a, \varphi(a, b - 1, n), n - 1)$$

Die Ackermann Funktion, Alternativ

$$f(n) = \varphi(n, n, n)$$

$$\varphi(a, b, 0) = a + b$$
$$\varphi(a, 0, n) = \alpha(a, n - 1)$$
$$\varphi(a, b, n) = \varphi(a, \varphi(a, b - 1, n), n - 1)$$

$$\alpha(a, n) = \begin{cases} 0, & \text{wenn } n = 0 \\ 1, & \text{wenn } n = 1 \\ a, & \text{wenn } n > 1 \end{cases}$$

Die Ackermann Funktion, Alternativ

```
int f(int n) {
  return phi(n, n, n);
}

int alpha(int a, int n) {
  return n <= 1 ? n : a;
}
```

$$f(n) = \varphi(n, n, n)$$

$$\varphi(a, b, 0) = a + b$$
$$\varphi(a, 0, n) = \alpha(a, n - 1)$$
$$\varphi(a, b, n) = \varphi(a, \varphi(a, b - 1, n), n - 1)$$

$$\alpha(a, n) = \begin{cases} 0, & \text{wenn } n = 0 \\ 1, & \text{wenn } n = 1 \\ a, & \text{wenn } n > 1 \end{cases}$$

Die Ackermann Funktion, Alternativ

Die Ackermann Funktion, Alternativ

```java
int phi(int a, int b, int n) {

}
```

Die Ackermann Funktion, Alternativ

```
int phi(int a, int b, int n) {


}
```

$$\varphi(a,b,0) = a + b$$
$$\varphi(a,0,n) = \alpha(a,n-1)$$
$$\varphi(a,b,n) = \varphi(a,\varphi(a,b-1,n),n-1)$$

Die Ackermann Funktion, Alternativ

```
int phi(int a, int b, int n) {
    if(n == 0) return a + b;


}
```

$$\varphi(a,b,0) = a + b$$
$$\varphi(a,0,n) = \alpha(a,n-1)$$
$$\varphi(a,b,n) = \varphi(a,\varphi(a,b-1,n),n-1)$$

Die Ackermann Funktion, Alternativ

```
int phi(int a, int b, int n) {
    if(n == 0) return a + b;
    else if (b == 0) return alpha(a, n - 1);

}
```

$$\varphi(a, b, 0) = a + b$$
$$\varphi(a, 0, n) = \alpha(a, n - 1)$$
$$\varphi(a, b, n) = \varphi(a, \varphi(a, b - 1, n), n - 1)$$

Die Ackermann Funktion, Alternativ

```
int phi(int a, int b, int n) {
    if(n == 0) return a + b;
    else if (b == 0) return alpha(a, n - 1);
    else return phi(a, phi(a, b - 1, n), n - 1);
}
```

$$\varphi(a,b,0) = a + b$$
$$\varphi(a,0,n) = \alpha(a,n-1)$$
$$\varphi(a,b,n) = \varphi(a,\varphi(a,b-1,n),n-1)$$

Die Ackermann Funktion, Alternativ

```
int phi(int a, int b, int n) {
    if(n == 0) return a + b;
    else if (b == 0) return alpha(a, n - 1);
    else return phi(a, phi(a, b - 1, n), n - 1);
}
```

```
f(1) = 1
```

$$\varphi(a,b,0) = a + b$$
$$\varphi(a,0,n) = \alpha(a,n-1)$$
$$\varphi(a,b,n) = \varphi(a,\varphi(a,b-1,n),n-1)$$

Die Ackermann Funktion, Alternativ

```
int phi(int a, int b, int n) {
    if(n == 0) return a + b;
    else if (b == 0) return alpha(a, n - 1);
    else return phi(a, phi(a, b - 1, n), n - 1);
}
```

```
f(1) = 1, f(2) = 4
```

$$\varphi(a, b, 0) = a + b$$
$$\varphi(a, 0, n) = \alpha(a, n - 1)$$
$$\varphi(a, b, n) = \varphi(a, \varphi(a, b - 1, n), n - 1)$$

Die Ackermann Funktion, Alternativ

```
int phi(int a, int b, int n) {
    if(n == 0) return a + b;
    else if (b == 0) return alpha(a, n - 1);
    else return phi(a, phi(a, b - 1, n), n - 1);
}
```

f(1) = 1, f(2) = 4, f(3) = ϟ

$$\varphi(a,b,0) = a + b$$
$$\varphi(a,0,n) = \alpha(a,n-1)$$
$$\varphi(a,b,n) = \varphi(a,\varphi(a,b-1,n),n-1)$$

Die Ackermann Funktion, Alternativ

```
int phi(int a, int b, int n) {
    if(n == 0) return a + b;
    else if (b == 0) return alpha(a, n - 1);
    else return phi(a, phi(a, b - 1, n), n - 1);
}
```

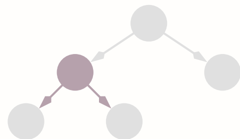
f(1) = 1, f(2) = 4, f(3) = ϟ

$\gtrsim 2 \uparrow^2 7 \approx 10^{10^{10^{19\,000}}}$

$$\varphi(a, b, 0) = a + b$$
$$\varphi(a, 0, n) = \alpha(a, n - 1)$$
$$\varphi(a, b, n) = \varphi(a, \varphi(a, b - 1, n), n - 1)$$

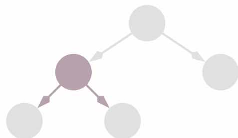
17.1 Rekursive Strukturen

Rekursive Strukturen



Jeder Knoten ist die Wurzel eines Teilbaums

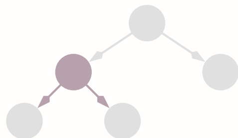
Rekursive Strukturen



Jeder Knoten ist die Wurzel eines Teilbaums

```
class Node {
   Node left;
   Node right;
   int data;
}
```

Rekursive Strukturen



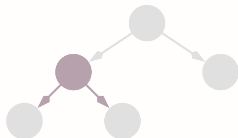
Jeder Knoten ist die Wurzel eines Teilbaums

```
class Node {
    Node left;
    Node right;
    int data;
}
```

```
class Node<T> {
  Node left;
  Node right;
  T data;
}
```

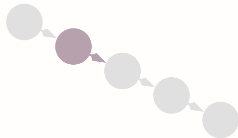
Cooler mit Generics![4]

17.5 Rekursive Strukturen



Jeder Knoten ist die Wurzel eines Teilbaums

```
class Node {
   Node left;
   Node right;
   int data;
}
```

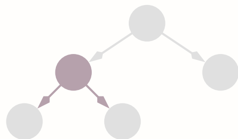


Jedes Element ist Kopf einer Teilliste

```
class Node<T> {
  Node left;
  Node right;
  T data;
}
```

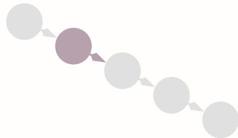
Cooler mit Generics![4]

17.6 Rekursive Strukturen



Jeder Knoten ist die Wurzel eines Teilbaums

```
class Node {
   Node left;
   Node right;
   int data;
}
```



Jedes Element ist Kopf einer Teilliste

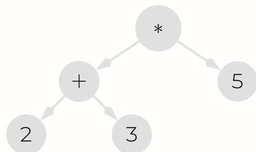
```
class Element {
   Element next;
   int data;
}
```

```
class Node<T> {
  Node left;
  Node right;
  T data;
}
```

Cooler mit Generics![4]

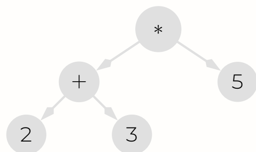
18.1 Einen Baum evaluieren

18.2 Einen Baum evaluieren



Ein Arithmetik-Baum!

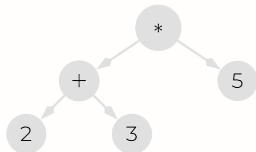
Einen Baum evaluieren



Ein Arithmetik-Baum!

```
class Node {
   Node left, right;
   int data; String op;
}
```

18.4 Einen Baum evaluieren

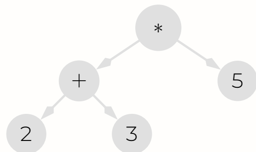


Ein Arithmetik-Baum!

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

```
class Node {
    Node left, right;
    int data; String op;
}
```

18.5 Einen Baum evaluieren



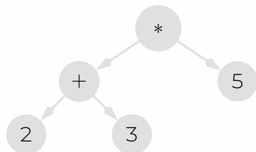
Ein Arithmetik-Baum!

```java
long postOrderEval(Node node) {

}
```

```java
class Node {
    Node left, right;
    int data; String op;
}
```

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

Einen Baum evaluieren



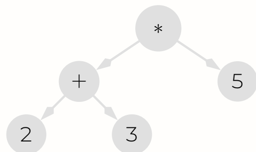
Ein Arithmetik-Baum!

```
class Node {
    Node left, right;
    int data; String op;
}
```

```
long postOrderEval(Node node) {
    if(node == null) throw new /* ... */;

}
```

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

18.7 Einen Baum evaluieren



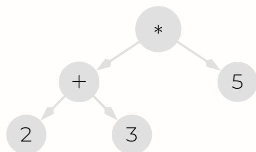
Ein Arithmetik-Baum!

```
class Node {
    Node left, right;
    int data; String op;
}
```

```
long postOrderEval(Node node) {
    if(node == null) throw new /* ... */;
    if(node.op == null) // no operation
        return node.data;



}
```

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

18.8 Einen Baum evaluieren



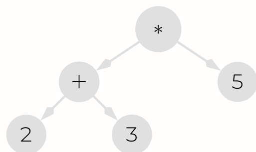
Ein Arithmetik-Baum!

```
class Node {
    Node left, right;
    int data; String op;
}
```

```
long postOrderEval(Node node) {
    if(node == null) throw new /* ... */;
    if(node.op == null) // no operation
        return node.data;
    long l = postOrderEval(node.left);
    long r = postOrderEval(node.right);


}
```

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

18.9 Einen Baum evaluieren



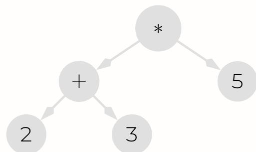
Ein Arithmetik-Baum!

```
class Node {
    Node left, right;
    int data; String op;
}
```

```
long postOrderEval(Node node) {
    if(node == null) throw new /* ... */;
    if(node.op == null) // no operation
        return node.data;
    long l = postOrderEval(node.left);
    long r = postOrderEval(node.right);
    switch(node.op) {
        case "+":
        case "*":

    }
}
```

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

18.10 Einen Baum evaluieren



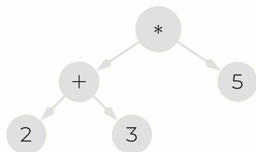
Ein Arithmetik-Baum!

```
class Node {
    Node left, right;
    int data; String op;
}
```

```
long postOrderEval(Node node) {
    if(node == null) throw new /* ... */;
    if(node.op == null) // no operation
        return node.data;
    long l = postOrderEval(node.left);
    long r = postOrderEval(node.right);
    switch(node.op) {
        case "+": return l + r;
        case "*":

    }
}
```

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

Einen Baum evaluieren



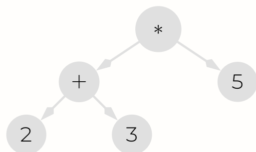
Ein Arithmetik-Baum!

```
long postOrderEval(Node node) {
    if(node == null) throw new /* ... */;
    if(node.op == null) // no operation
        return node.data;
    long l = postOrderEval(node.left);
    long r = postOrderEval(node.right);
    switch(node.op) {
        case "+": return l + r;
        case "*": return l * r;

    }
}
```

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

```
class Node {
    Node left, right;
    int data; String op;
}
```

Einen Baum evaluieren



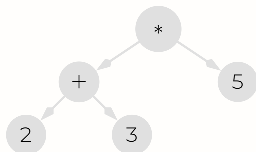
Ein Arithmetik-Baum!

```
class Node {
    Node left, right;
    int data; String op;
}
```

```
long postOrderEval(Node node) {
    if(node == null) throw new /* ... */;
    if(node.op == null) // no operation
        return node.data;
    long l = postOrderEval(node.left);
    long r = postOrderEval(node.right);
    switch(node.op) {
        case "+": return l + r;
        case "*": return l * r;
        // ... other cases, errors, ...
    }
}
```

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

Einen Baum evaluieren



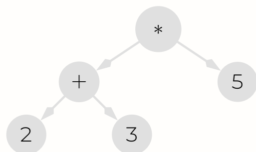
Ein Arithmetik-Baum!

```java
class Node {
    Node left, right;
    int data; String op;
}
```

```java
long postOrderEval(Node node) {
    if(node == null) throw new /* ... */;
    if(node.op == null) // no operation
        return node.data;
    long l = postOrderEval(node.left);
    long r = postOrderEval(node.right);
    switch(node.op) {
        case "+": return l + r;
        case "*": return l * r;
        // ... other cases, errors, ...
    }
}
```

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

Einen Baum evaluieren



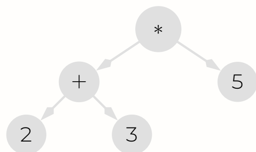
Ein Arithmetik-Baum!

```
class Node {
    Node left, right;
    int data; String op;
}
```

```
long postOrderEval(Node node) {
    if(node == null) throw new /* ... */;
    if(node.op == null) // no operation
        return node.data;
    long l = postOrderEval(node.left);
    long r = postOrderEval(node.right);
    switch(node.op) {
        case "+": return l + r;
        case "*": return l * r;
        // ... other cases, errors, ...
    }
}
```

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

Einen Baum evaluieren



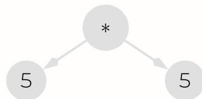
Ein Arithmetik-Baum!

```
class Node {
    Node left, right;
    int data; String op;
}
```

```
long postOrderEval(Node node) {
    if(node == null) throw new /* ... */;
    if(node.op == null) // no operation
        return node.data;
    long l = postOrderEval(node.left);
    long r = postOrderEval(node.right);
    switch(node.op) {
        case "+": return l + r;
        case "*": return l * r;
        // ... other cases, errors, ...
    }
}
```

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

18.16 Einen Baum evaluieren



Ein Arithmetik-Baum!

```
class Node {
    Node left, right;
    int data; String op;
}
```

```
long postOrderEval(Node node) {
    if(node == null) throw new /* ... */;
    if(node.op == null) // no operation
        return node.data;
    long l = postOrderEval(node.left);
    long r = postOrderEval(node.right);
    switch(node.op) {
        case "+": return l + r;
        case "*": return l * r;
        // ... other cases, errors, ...
    }
}
```

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

Einen Baum evaluieren

25

Ein Arithmetik-Traum!

```
class Node {
    Node left, right;
    int data; String op;
}
```

```
long postOrderEval(Node node) {
    if(node == null) throw new /* ... */;
    if(node.op == null) // no operation
        return node.data;
    long l = postOrderEval(node.left);
    long r = postOrderEval(node.right);
    switch(node.op) {
        case "+": return l + r;
        case "*": return l * r;
        // ... other cases, errors, ...
    }
}
```

An sich würde man die Knoten anders realisieren. Z.B. durch Vererbung.

19.1 Die Welt rekursiver Probleme

19.2 Die Welt rekursiver Probleme

Die Welt rekursiver Probleme

1. Rekursives Sortieren

Die Welt rekursiver Probleme

1. Rekursives Sortieren

19.5

Die Welt rekursiver Probleme

1. Rekursives Sortieren

19.6

Die Welt rekursiver Probleme

1. Rekursives Sortieren

Die Welt rekursiver Probleme

1. Rekursives Sortieren

19.8 Die Welt rekursiver Probleme

1. Rekursives Sortieren

2. Die Türme von Hanoi

Die Welt rekursiver Probleme

1. Rekursives Sortieren

2. Die Türme von Hanoi

Die Welt rekursiver Probleme

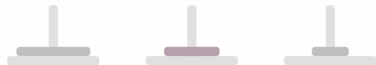
1. Rekursives Sortieren

2. Die Türme von Hanoi

Die Welt rekursiver Probleme

1. Rekursives Sortieren

2. Die Türme von Hanoi



Die Welt rekursiver Probleme

1. Rekursives Sortieren

2. Die Türme von Hanoi

Die Welt rekursiver Probleme

1. Rekursives Sortieren

2. Die Türme von Hanoi

Die Welt rekursiver Probleme

1. Rekursives Sortieren

2. Die Türme von Hanoi

Die Welt rekursiver Probleme

1. Rekursives Sortieren

2. Die Türme von Hanoi

Die Welt rekursiver Probleme

1. Rekursives Sortieren

2. Die Türme von Hanoi

Die Welt rekursiver Probleme

1. Rekursives Sortieren

2. Die Türme von Hanoi

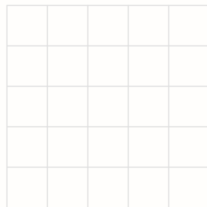
3. Das N-Damen Problem

Die Welt rekursiver Probleme

1. Rekursives Sortieren

2. Die Türme von Hanoi

3. Das N-Damen Problem

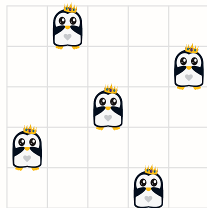


Die Welt rekursiver Probleme

1. Rekursives Sortieren

2. Die Türme von Hanoi

3. Das N-Damen Problem

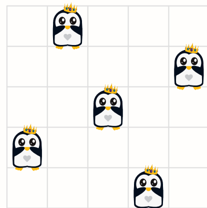


Die Welt rekursiver Probleme

1. Rekursives Sortieren

2. Die Türme von Hanoi

3. Das N-Damen Problem



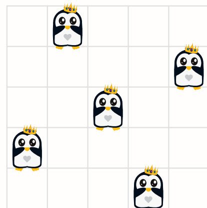
- Sudokus
- Wegfindung
- Hilbert-Kurven
- ...

Die Welt rekursiver Probleme

1. Rekursives Sortieren

2. Die Türme von Hanoi

3. Das N-Damen Problem



- Sudokus
- Wegfindung
- Hilbert-Kurven
- …

Alles was sich in (eventuell explorative) Einzelschritte zerteilen lässt.

Rekursives Sortieren

Rekursives Sortieren


```
void quicksort(int[] arr) {

}
```

Rekursives Sortieren


```
void quicksort(int[] arr) {
    quicksort(arr, 0, arr.length - 1);
}
```

Rekursives Sortieren


```
void quicksort(int[] arr) {
   quicksort(arr, 0, arr.length - 1);
}

void quicksort(int[] arr, int left, int right) {




}
```

Rekursives Sortieren


```
void quicksort(int[] arr) {
   quicksort(arr, 0, arr.length - 1);
}

void quicksort(int[] arr, int left, int right) {
   if(low >= right) return;



}
```

Rekursives Sortieren


```
void quicksort(int[] arr) {
   quicksort(arr, 0, arr.length - 1);
}

void quicksort(int[] arr, int left, int right) {
   if(low >= right) return;
   int partIndex = partition(arr, left, right);


}
```

Rekursives Sortieren


```
void quicksort(int[] arr) {
    quicksort(arr, 0, arr.length - 1);
}

void quicksort(int[] arr, int left, int right) {
    if(low >= right) return;
    int partIndex = partition(arr, left, right);
    quicksort(arr, left, partIndex - 1);
    quicksort(arr, partIndex + 1, right);
}
```

Rekursives Sortieren


```
void quicksort(int[] arr) {
   quicksort(arr, 0, arr.length - 1);
}

void quicksort(int[] arr, int left, int right) {
   if(low >= right) return;
   int partIndex = partition(arr, left, right);
   quicksort(arr, left, partIndex - 1);
   quicksort(arr, partIndex + 1, right);
}
```

Wählt Pivot und sortiert kleiner Pivotelement links und größer gleich rechts.

21.1 Die Türme von Hanoi

Die Türme von Hanoi

Die Türme von Hanoi

- Trivial für eine Scheibe: $A \to C$.

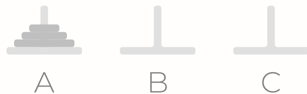
Die Türme von Hanoi

- Trivial für eine Scheibe: $A \to C$.
- Sonst:
  - Bewege $n - 1$ Scheiben auf Hilfsstapel: $A \to B$.

Die Türme von Hanoi

- Trivial für eine Scheibe: $A \to C$.
- Sonst:
  - Bewege $n - 1$ Scheiben auf Hilfsstapel: $A \to B$.
  - Bewege letzte Scheibe (größte) von $A \to C$.

Die Türme von Hanoi



- Trivial für eine Scheibe: $A \to C$.
- Sonst:
  - Bewege $n-1$ Scheiben auf Hilfsstapel: $A \to B$.
  - Bewege letzte Scheibe (größte) von $A \to C$.
  - Bewege $n-1$ Scheiben vom Hilfsstapel zum Ziel: $B \to C$.

Die Türme von Hanoi

- Trivial für eine Scheibe: $A \to C$.
- Sonst:
  - Bewege $n - 1$ Scheiben auf Hilfsstapel: $A \to B$.
  - Bewege letzte Scheibe (größte) von $A \to C$.
  - Bewege $n - 1$ Scheiben vom Hilfsstapel zum Ziel: $B \to C$.

```java
enum Pole {
  SOURCE,
  HELPER,
  TARGET
}
```

Die Türme von Hanoi

- Trivial für eine Scheibe: $A \to C$.
- Sonst:
  - Bewege $n - 1$ Scheiben auf Hilfsstapel: $A \to B$.
  - Bewege letzte Scheibe (größte) von $A \to C$.
  - Bewege $n - 1$ Scheiben vom Hilfsstapel zum Ziel: $B \to C$.

```
enum Pole {
  SOURCE,
  HELPER,
  TARGET
}
```

```
void hanoi(int disks, Pole a, Pole b, Pole c) {



}
```

Die Türme von Hanoi

- Trivial für eine Scheibe: $A \to C$.
- Sonst:
  - Bewege $n - 1$ Scheiben auf Hilfsstapel: $A \to B$.
  - Bewege letzte Scheibe (größte) von $A \to C$.
  - Bewege $n - 1$ Scheiben vom Hilfsstapel zum Ziel: $B \to C$.

```
enum Pole {
  SOURCE,
  HELPER,
  TARGET
}
```

```
void hanoi(int disks, Pole a, Pole b, Pole c) {
    if(disks == 1) { move(a, c); }
    else {



    }
}
```

Die Türme von Hanoi

- Trivial für eine Scheibe: $A \to C$.
- Sonst:
  - Bewege $n - 1$ Scheiben auf Hilfsstapel: $A \to B$.
  - Bewege letzte Scheibe (größte) von $A \to C$.
  - Bewege $n - 1$ Scheiben vom Hilfsstapel zum Ziel: $B \to C$.

```
enum Pole {
  SOURCE,
  HELPER,
  TARGET
}
```

```
void hanoi(int disks, Pole a, Pole b, Pole c) {
    if(disks == 1) { move(a, c); }
    else {
        hanoi(disks - 1, a, c, b); // move all disks above to help pole


    }
}
```

Die Türme von Hanoi

- Trivial für eine Scheibe: $A \rightarrow C$.
- Sonst:
  - Bewege $n - 1$ Scheiben auf Hilfsstapel: $A \rightarrow B$.
  - Bewege letzte Scheibe (größte) von $A \rightarrow C$.
  - Bewege $n - 1$ Scheiben vom Hilfsstapel zum Ziel: $B \rightarrow C$.

```
enum Pole {
  SOURCE,
  HELPER,
  TARGET
}
```

```
void hanoi(int disks, Pole a, Pole b, Pole c) {
    if(disks == 1) { move(a, c); }
    else {
        hanoi(disks - 1, a, c, b); // move all disks above to help pole
        move(a, c); // move last disk to target

    }
}
```

Die Türme von Hanoi

- Trivial für eine Scheibe: $A \rightarrow C$.
- Sonst:
  - Bewege $n - 1$ Scheiben auf Hilfsstapel: $A \rightarrow B$.
  - Bewege letzte Scheibe (größte) von $A \rightarrow C$.
  - Bewege $n - 1$ Scheiben vom Hilfsstapel zum Ziel: $B \rightarrow C$.

```
enum Pole {
  SOURCE,
  HELPER,
  TARGET
}
```

```
void hanoi(int disks, Pole a, Pole b, Pole c) {
    if(disks == 1) { move(a, c); }
    else {
        hanoi(disks - 1, a, c, b); // move all disks above to help pole
        move(a, c); // move last disk to target
        hanoi(disks - 1, b, a, c); // move disks from help to target pole
    }
}
```

Die Türme von Hanoi

- Trivial für eine Scheibe: $A \rightarrow C$.
- Sonst:
  - Bewege $n - 1$ Scheiben auf Hilfsstapel: $A \rightarrow B$.
  - Bewege letzte Scheibe (größte) von $A \rightarrow C$.
  - Bewege $n - 1$ Scheiben vom Hilfsstapel zum Ziel: $B \rightarrow C$.

```
enum Pole {
  SOURCE,
  HELPER,
  TARGET
}
```

```
void hanoi(int disks, Pole a, Pole b, Pole c) {
   if(disks == 1) { move(a, c); }
   else {
      hanoi(disks - 1, a, c, b); // move all disks above to help pole
      move(a, c); // move last disk to target
      hanoi(disks - 1, b, a, c); // move disks from help to target pole
   }
}
```

Gibt im einfachsten Fall einfach „$a \rightarrow c$“ aus.

22.1 Das N-Damen Problem

22.2 Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

22.3 Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {




}
```

Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;







}
```

Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {



    }

}
```

22.6 Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;



    }

}
```

Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here



    }

}
```

22.8 Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;

    }

}
```

Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }

}
```

Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

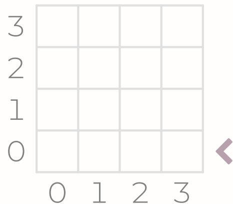
Wie könnte man `queenCanBePlacedHere` umsetzen?

22.12 Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

- ```java
  boolean nQueens(boolean[][] board, int row) {
      if(row == board.length) return true;
      for (int column = 0; column < board.length; column++) {
          if(!queenCanBePlacedHere(board, column, row)) continue;
          board[row][column] = true; // Try to place queen here
          if(nQueens(board, row + 1))
              return true;
          board[row][column] = false;
      }
      return false;
  }
  ```

Wie könnte man `queenCanBePlacedHere` umsetzen?

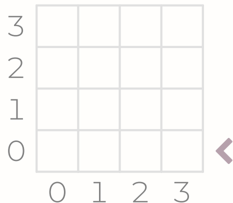

  ```

Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

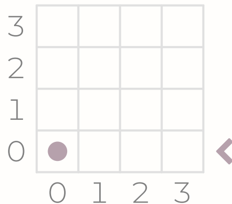


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

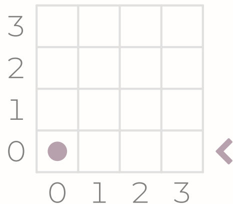


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

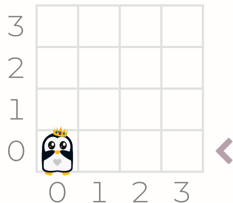


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

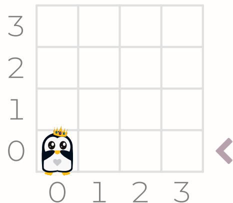


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if( nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

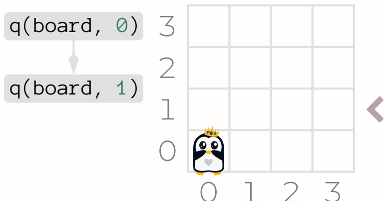


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

- ```
  boolean nQueens(boolean[][] board, int row) {
      if(row == board.length) return true;
      for (int column = 0; column < board.length; column++) {
          if(!queenCanBePlacedHere(board, column, row)) continue;
          board[row][column] = true; // Try to place queen here
          if(nQueens(board, row + 1))
              return true;
          board[row][column] = false;
      }
      return false;
  }
  ```

Wie könnte man queenCanBePlacedHere umsetzen?

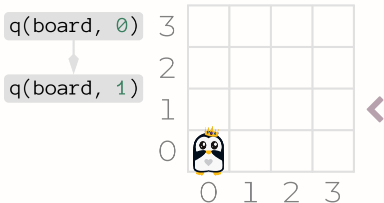

  ```

Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

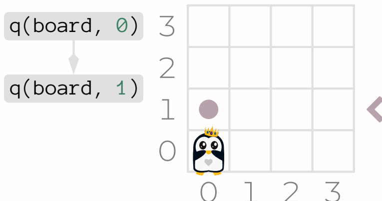


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

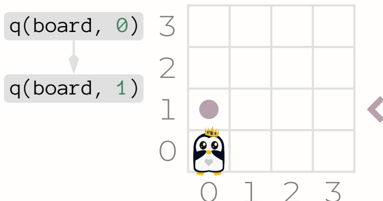


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

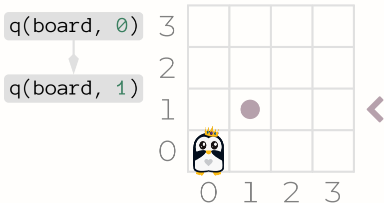


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

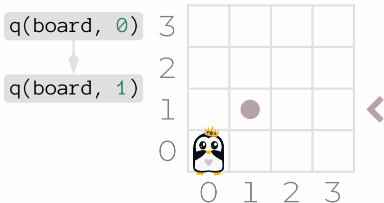


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

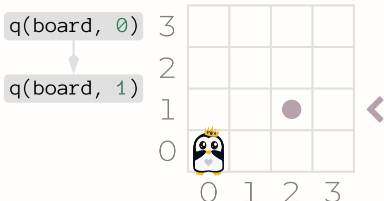


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

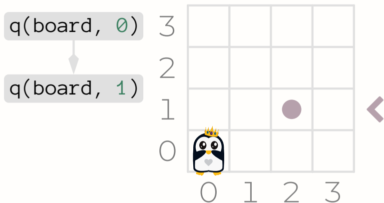


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

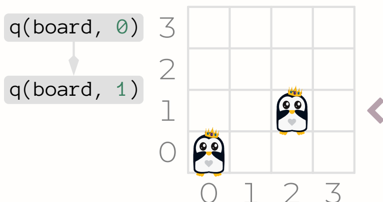


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

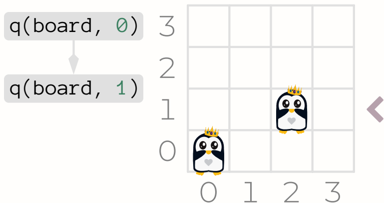


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if( nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

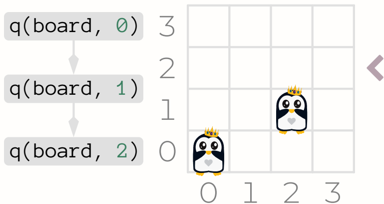


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

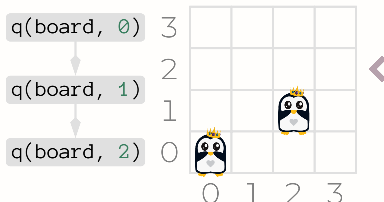


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

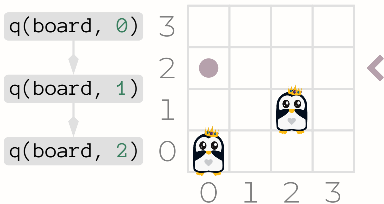


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

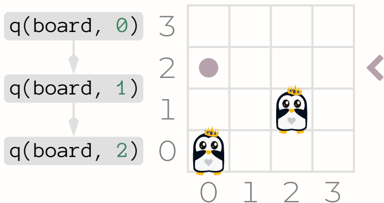


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

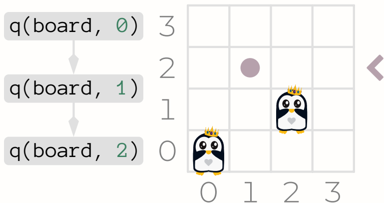


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

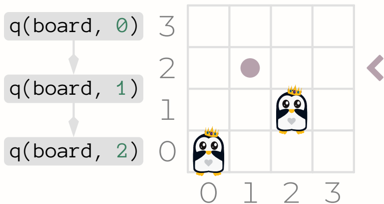


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

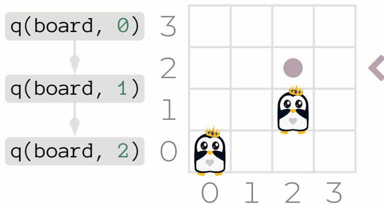


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

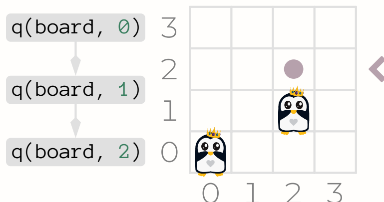


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

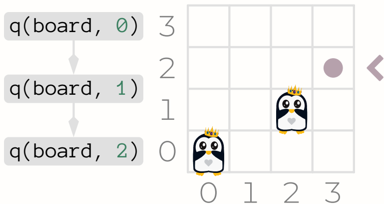


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

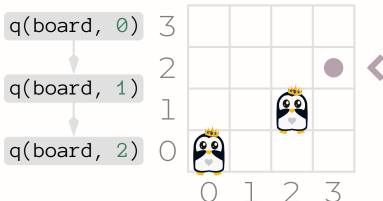


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

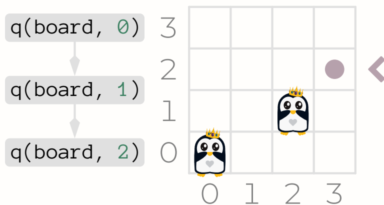


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

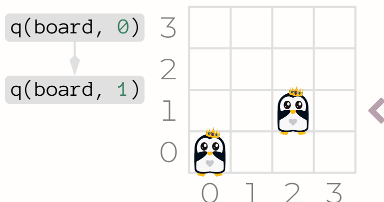


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

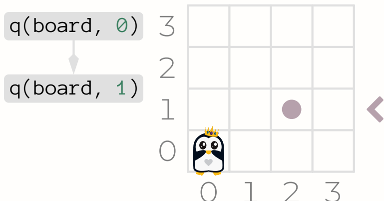


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

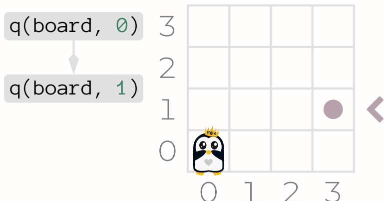


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

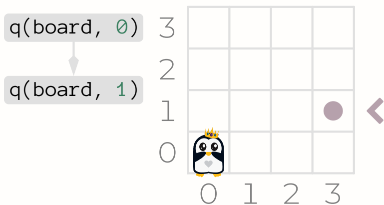


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

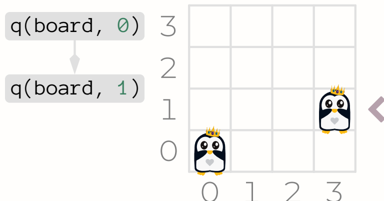


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

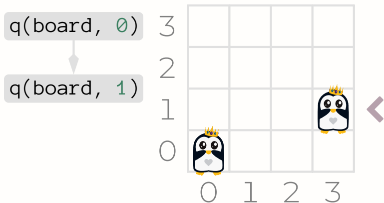


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if( nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

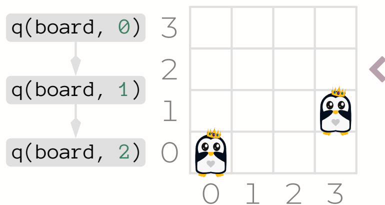


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?

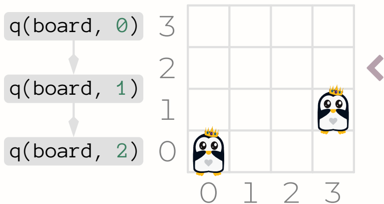


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

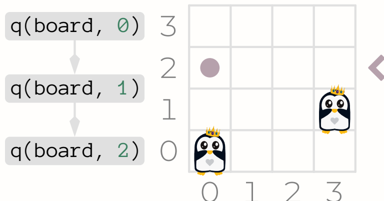


Das N-Damen Problem

```java
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```java
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

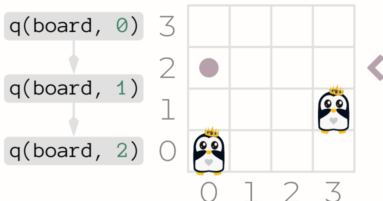


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

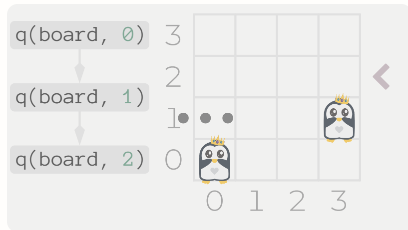


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man `queenCanBePlacedHere` umsetzen?

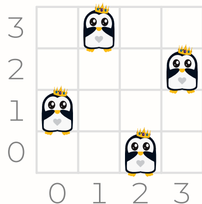


Das N-Damen Problem

```
boolean nQueens(boolean[][] board) {
  return nQueens(board, 0);
}
```

```
boolean nQueens(boolean[][] board, int row) {
    if(row == board.length) return true;
    for (int column = 0; column < board.length; column++) {
        if(!queenCanBePlacedHere(board, column, row)) continue;
        board[row][column] = true; // Try to place queen here
        if(nQueens(board, row + 1))
            return true;
        board[row][column] = false;
    }
    return false;
}
```

Wie könnte man queenCanBePlacedHere umsetzen?



23.1 Das obligatorische „Übrigens…“

[1]: *Proper Tail Recursion and Space Efficiency* Clinger, 1998

Das obligatorische „Übrigens…“

- Viele Varianten:

[1]: *Proper Tail Recursion and Space Efficiency* Clinger, 1998

23.3 Das obligatorische „Übrigens…“

- Viele Varianten:
  - Tail-Rekursionen

[1]: *Proper Tail Recursion and Space Efficiency* Clinger, 1998

Das obligatorische „Übrigens…“

- Viele Varianten:
  - Tail-Rekursionen
  - Primitiv-Rekursive Funktionen

[1]: *Proper Tail Recursion and Space Efficiency* Clinger, 1998

Das obligatorische „Übrigens…“

- Viele Varianten:
  - Tail-Rekursionen
  - Primitiv-Rekursive Funktionen
- Stack-Frames sind spannender und enthalten noch mehr!

[1]: *Proper Tail Recursion and Space Efficiency* Clinger, 1998

Das obligatorische „Übrigens…“

- Viele Varianten:
  - Tail-Rekursionen
  - Primitiv-Rekursive Funktionen
- Stack-Frames sind spannender und enthalten noch mehr!
- Das riesige Thema des Compilerbaus liefert mehr:

[1]: *Proper Tail Recursion and Space Efficiency* Clinger, 1998

Das obligatorische „Übrigens…“

- Viele Varianten:
  - Tail-Rekursionen
  - Primitiv-Rekursive Funktionen
- Stack-Frames sind spannender und enthalten noch mehr!
- Das riesige Thema des Compilerbaus liefert mehr:
  - Optimierungen & Transformationen

[1]: *Proper Tail Recursion and Space Efficiency* Clinger, 1998

Das obligatorische „Übrigens…“

- Viele Varianten:
  - Tail-Rekursionen
  - Primitiv-Rekursive Funktionen
- Stack-Frames sind spannender und enthalten noch mehr!
- Das riesige Thema des Compilerbaus liefert mehr:
  - Optimierungen & Transformationen
  - Recursive Descent Parser

[1]: *Proper Tail Recursion and Space Efficiency* Clinger, 1998

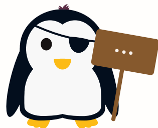
Das obligatorische „Übrigens…“

- Viele Varianten:
  - Tail-Rekursionen
  - Primitiv-Rekursive Funktionen
- Stack-Frames sind spannender und enthalten noch mehr!
- Das riesige Thema des Compilerbaus liefert mehr:
  - Optimierungen & Transformationen
  - Recursive Descent Parser
- Dynamische Programmie-rung!

[1]: *Proper Tail Recursion and Space Efficiency* Clinger, 1998

Das obligatorische „Übrigens…“

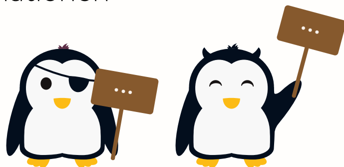
- Viele Varianten:
  - Tail-Rekursionen
  - Primitiv-Rekursive Funktionen
- Stack-Frames sind spannender und enthalten noch mehr!
- Das riesige Thema des Compilerbaus liefert mehr:
  - Optimierungen & Transformationen
  - Recursive Descent Parser
- Dynamische Programmierung!



[1]: *Proper Tail Recursion and Space Efficiency* Clinger, 1998

Das obligatorische „Übrigens...“

- Viele Varianten:
  - Tail-Rekursionen
  - Primitiv-Rekursive Funktionen
- Stack-Frames sind spannender und enthalten noch mehr!
- Das riesige Thema des Compilerbaus liefert mehr:
  - Optimierungen & Transformationen
  - Recursive Descent Parser
- Dynamische Programmie-rung!

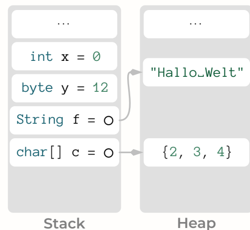


[1]: *Proper Tail Recursion and Space Efficiency* Clinger, 1998

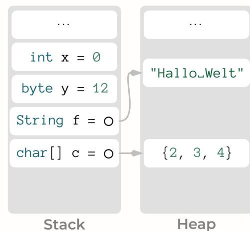
Das obligatorische „Übrigens…“

- Viele Varianten:
  - Tail-Rekursionen
  - Primitiv-Rekursive Funktionen
- Stack-Frames sind spannender und enthalten noch mehr!
- Das riesige Thema des Compilerbaus liefert mehr:
  - Optimierungen & Transformationen
  - Recursive Descent Parser
- Dynamische Programmie-rung!

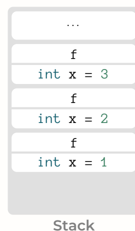
[1]: *Proper Tail Recursion and Space Efficiency* Clinger, 1998



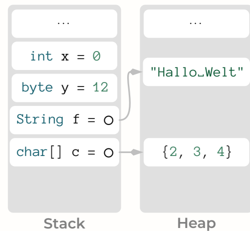
Speicherverwaltung



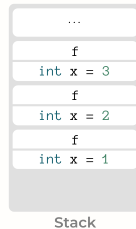
Speicherverwaltung



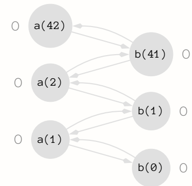
Stack Frames



Speicherverwaltung



Stack Frames



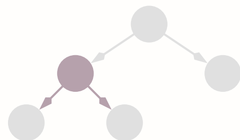
Rekursionstypen

$$f(n) = \begin{cases} 1, & n \leq 1 \\ n \cdot f(n-1), & n > 1 \end{cases}$$

Mathematisch

$$f(n) = \begin{cases} 1, & n \leq 1 \\ n \cdot f(n-1), & n > 1 \end{cases}$$

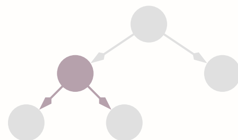
Mathematisch



Datenstrukturen

$$f(n) = \begin{cases} 1, & n \leq 1 \\ n \cdot f(n-1), & n > 1 \end{cases}$$

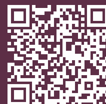
Mathematisch



Datenstrukturen

Divide- & Conquer

[1] William D. Clinger. „Proper Tail Recursion and Space Efficiency“. 1998
[2] Loïc Colson. „About primitive recursive algorithms“. 1991
[3] *Java Debug Interface – StackFrame (JDK 11)*. 2021
[4] *Lesson: Generics*. 2021
[5] Tim Lindholm u. a. *The Java® Virtual Machine Specification [Heap & Stack]*. 2020
[6] Yanhong A. Liu und Scott D. Stoller. „From Recursion to Iteration: What Are the Optimizations?“ Nov. 1999
[7] Manuel Rubio-Sánchez und Isidoro Hernán-Losada. „Exploring Recursion with Fibonacci Numbers“. 2007
[8] Florian Sihler. *LaTeX-Package, tikzpingus*. 2021
[9] Yngve Sundblad. „The Ackermann function. a theoretical, computational, and formula manipulative study“. 1971



Florian Sihler
Ulm, 10. November 2022