Informatik II - Übung 4

Pascal Schärli

Ctrl-Shift-F_lover96@pascscha.ch

16.10.2020

ETH zürich

Nachbesprechung

Programmverifikation

```
1  static int f(int i, int j) {
2     assert(i >= 0 && j >= 0);
3     int u = i;
4     int z = 0;
5     while (u > 0){
6        z = z + j;
7        u = u - 1;
8     }
9     return z;
10 }
```

- Schleifeninvarainte:
  - $z + u \cdot j = i \cdot j$ && $u \geq 0$
- Nach der Schleife:
  - $u \geq 0$ && $u \leq 0 \Rightarrow u = 0$
  - $\Rightarrow z = i \cdot j$

Schleifen Invariante

Sonst wäre die Schleife noch nicht fertig

String und StringBuffer

- String Buffer sind bei häufiger Veränderungen viel schneller als Strings
- Möglicher Output:

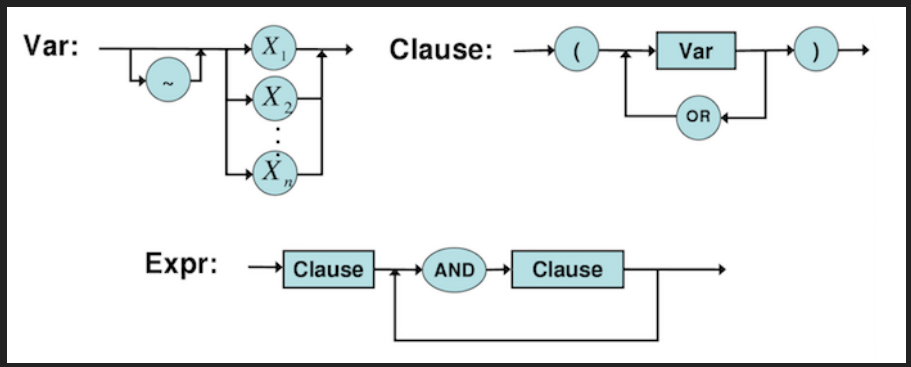
```
Starting encryption (using Strings)
Done - Duration: 3732 ms.
Starting decryption (using StringBuffers)
Done - Duration: 73 ms.
Decryption successful :-)
```

```java
public static String encrypt(String s) {
    String ret = "";
    for (int i = 0; i != s.length(); ++i) {
        ret = ret + (char) (s.charAt(i) + 3);
    }
    return ret;
}
```

```java
public static String decrypt(String s) {
    StringBuffer ret = new StringBuffer();
    for (int i = 0; i != s.length(); ++i) {
        ret.append((char) (s.charAt(i) - 3));
    }
    return ret.toString();
}
```

Syntaxdiagramm

- Sind die folgenden Strings korrekte Clauses?
  1. X_2
  2. $(\sim X_1)$
  3. $\sim (X_1\ OR \sim X_2)$
  4. $(X_2)\ OR\ (\sim X_1\ OR\ X_2)$



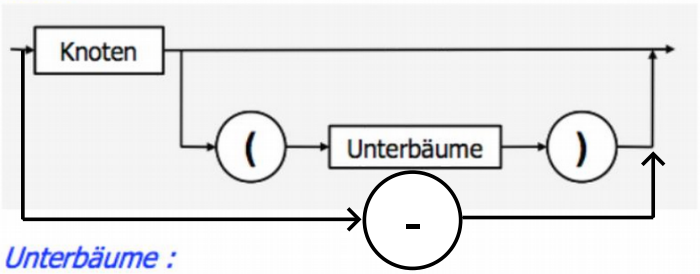
- Sind die folgenden Strings korrekte Expressions?
  1. $(X_1\ OR\ X_2)\ AND\ (\sim X_2)$
  2. $(X_1)\ AND\ (\sim X_1\ OR\ \sim X_2)\ AND\ (X_2)$

Syntaxdiagramm für Bäume

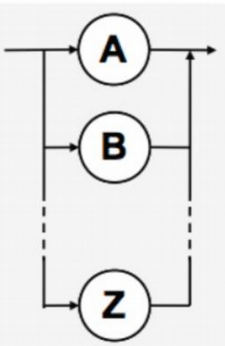
Ein Syntaxdiagramm-Beispiel: Klammerdarstellung eines Baums

Beispiel: A (B (D) , C (E , F))

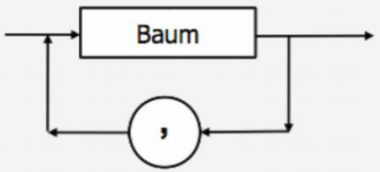
Baum:



Knoten:



Unterbäume :

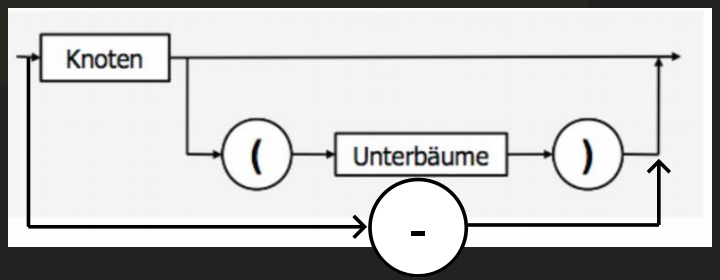


Wie könnte man Binärbäume durch ein Syntaxdiagramm darstellen?

181

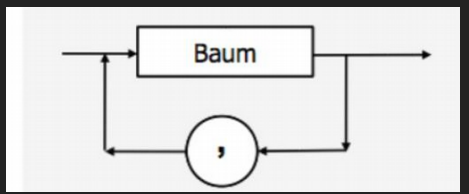
Syntaxchecker für Bäume

```java
private static int parseTree(String kd, int offset) throws ParseException {
    if (checkNext('-', kd, offset)) {
        return offset + 1;
    }
    offset = parseNode(kd, offset);
    if (checkNext('(', kd, offset)) {
        offset += 1;
        offset = parseSubtree(kd, offset);
        if (!checkNext(')', kd, offset)) {
            throw new ParseException("expected ')'", offset);
        }
        offset += 1;
    }
    return offset;
}
```



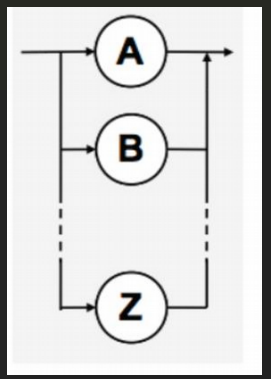
Syntaxchecker für Bäume

```java
private static int parseSubtree(String kd, int offset) throws ParseException {
    offset = parseTree(kd, offset);
    while (checkNext(',', kd, offset)) {
        offset += 1;
        offset = parseTree(kd, offset);
    }
    return offset;
}
```



Syntaxchecker für Bäume

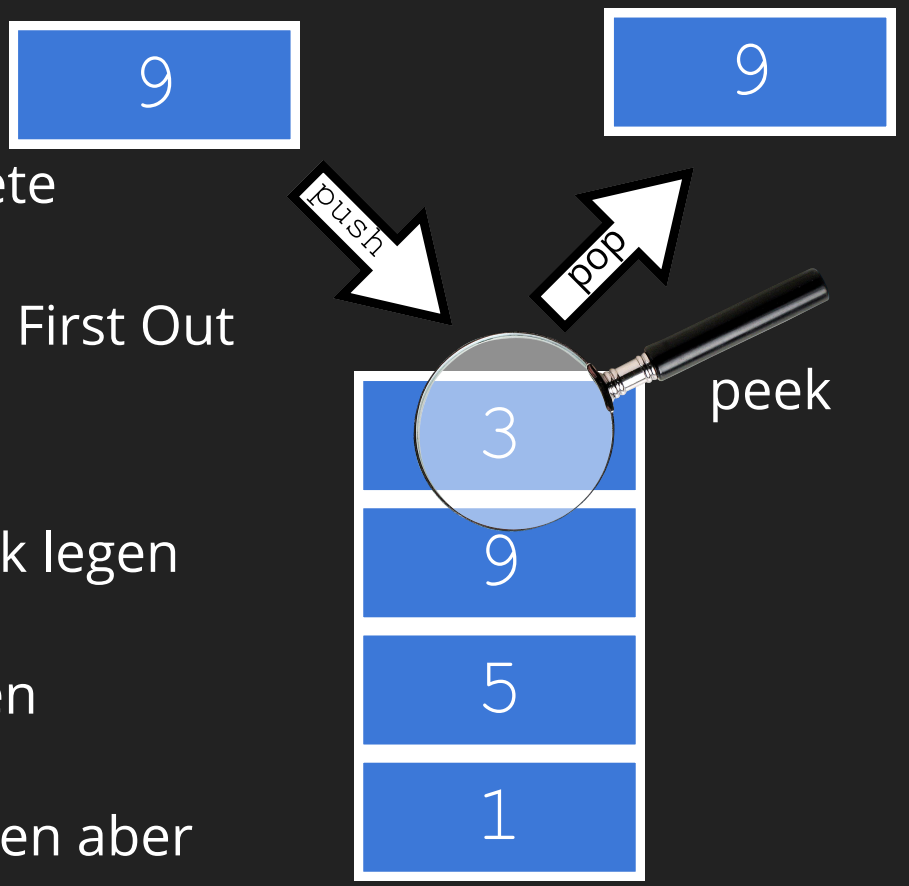
```java
private static int parseNode(String kd, int offset) throws ParseException {
    if (offset >= kd.length()) {
        throw new ParseException("Expected a node", offset);
    }

    char c = kd.charAt(offset);
    if (Character.isUpperCase(c)) {
        return offset + 1;
    } else {
        throw new ParseException(String.format("'%c' is not a valid node name", c), offset);
    }
}
```



Vorlesung

Stacks, LIFO (Last In First Out)

- Stacks sind sehr weit verbreitete Datenstrukturen
- Sie benutzen LIFO, also Last In First Out
- Es gibt drei Hauptfunktionen:
  - push
    Element oben auf den Stack legen
  - pop
    Oberstes Element entfernen
  - peek
    Oberstes Element anschauen aber nicht entfernen



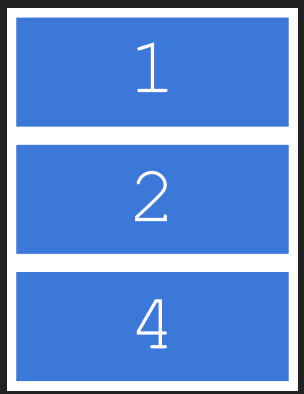
Stacks, LIFO (Last In First Out)

Program

```
1 Stack myStack = new Stack();
2
3 myStack.push(4);
4 System.out.println(myStack.peek());
5 myStack.push(2);
6 myStack.push(3);
7 System.out.println(myStack.pop());
8 myStack.push(1);
```

Output

```
4
3
```



Ackermann Funktion

- Die Ackermannfunktion ist eine extrem schnell wachsende Funktion, mit deren Hilfe in der theoretischen Informatik Grenzen von Modellen aufgezeigt werden können.

- Rekursive Definition:
  - $A(0, m) = m + 1$
  - $A(n, 0) = A(n - 1, 1)$
  - $A(n, m) = A(n - 1, A(n, m - 1))$

Ackermann Funktion

Beispiel: $A(1,1)$

```
1 A(1,1)          // A(1,1)
2   A(1,0)        // A(1,1) = A(0,A(1,0))
3     A(0, 1)     //              A(1,0) = A(0,1)
4     <- 2        //                       A(0,1) = 2
5   <- 2          //              A(1,0) = 2
6   A(0, 2)       // A(1,1) = A(0,2)
7   <- 3          //          A(0,2) = 3
8 <- 3            // A(1,1) = 3
```

$\Rightarrow A(1,1) = 3$

Weitere Beispiele:

- $A(4,0) = {}_{13}$
- $A(4,1) = 65533$
- $A(4,2) = 2^{65536} - 3$

$$A(0,m) = m + 1$$

$$A(n,0) = A(n-1,1)$$

$$A(n,m) = A(n-1, A(n,m-1))$$

Java Bytecode

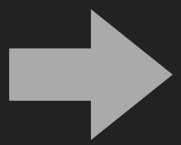
- Java funktioniert cross-plattform auf unterschiedlichsten Gerätetypen und Architekturen.
- Um die selbe Funktionalität für all diese Geräte zu ermöglichen, gibt es beim Kompilieren eine Zwischenstufe, der Bytecode.
- Dieser kann dann für die jeweilige Architektur in Maschinensprache übersetzt werden.
- Auf Wikipedia gibt es eine Liste von allen Bytecode Instruktionen.

Java Bytecode

- Der Bytecode arbeitet mit einem "Operand Stack"
- Die Werte mit welchen gerechnet wird, befinden sich auf einem Stack und wir verarbeiten immer die obersten Elemente auf dem Stack.

Bytecode

```
1 iconst_4
2 iconst_2
3 iconst_3
4 imul
5 iconst_1
6 isub
7 iadd
```



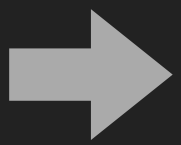
Stack

Java Bytecode

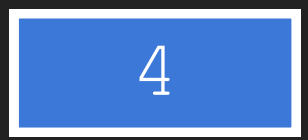
- Der Bytecode arbeitet mit einem "Operand Stack"
- Die Werte mit welchen gerechnet wird, befinden sich auf einem Stack und wir verarbeiten immer die obersten Elemente auf dem Stack.

Bytecode

```
1 iconst_4
2 iconst_2
3 iconst_3
4 imul
5 iconst_1
6 isub
7 iadd
```



Stack

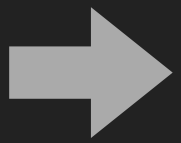


Java Bytecode

- Der Bytecode arbeitet mit einem "Operand Stack"
- Die Werte mit welchen gerechnet wird, befinden sich auf einem Stack und wir verarbeiten immer die obersten Elemente auf dem Stack.

Bytecode

```
1 iconst_4
2 iconst_2
3 iconst_3
4 imul
5 iconst_1
6 isub
7 iadd
```



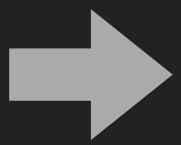
Stack

Java Bytecode

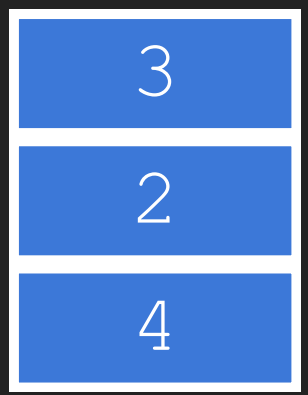
- Der Bytecode arbeitet mit einem "Operand Stack"
- Die Werte mit welchen gerechnet wird, befinden sich auf einem Stack und wir verarbeiten immer die obersten Elemente auf dem Stack.

Bytecode

```
1 iconst_4
2 iconst_2
3 iconst_3
4 imul
5 iconst_1
6 isub
7 iadd
```



Stack

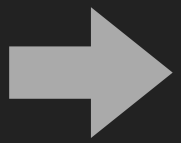


Java Bytecode

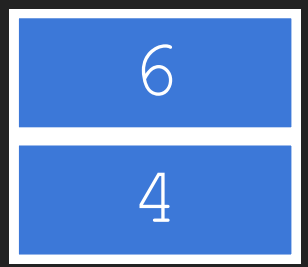
- Der Bytecode arbeitet mit einem "Operand Stack"
- Die Werte mit welchen gerechnet wird, befinden sich auf einem Stack und wir verarbeiten immer die obersten Elemente auf dem Stack.

Bytecode

```
1 iconst_4
2 iconst_2
3 iconst_3
4 imul
5 iconst_1
6 isub
7 iadd
```



Stack

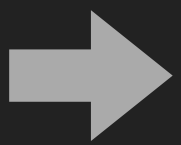


Java Bytecode

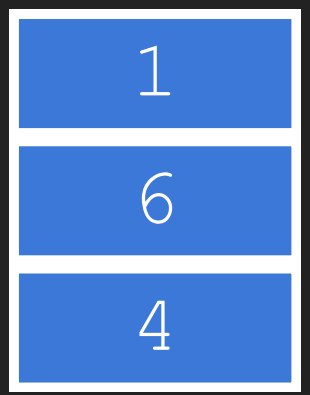
- Der Bytecode arbeitet mit einem "Operand Stack"
- Die Werte mit welchen gerechnet wird, befinden sich auf einem Stack und wir verarbeiten immer die obersten Elemente auf dem Stack.

Bytecode

```
1 iconst_4
2 iconst_2
3 iconst_3
4 imul
5 iconst_1
6 isub
7 iadd
```



Stack

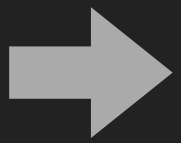


Java Bytecode

- Der Bytecode arbeitet mit einem "Operand Stack"
- Die Werte mit welchen gerechnet wird, befinden sich auf einem Stack und wir verarbeiten immer die obersten Elemente auf dem Stack.

Bytecode

```
1 iconst_4
2 iconst_2
3 iconst_3
4 imul
5 iconst_1
6 isub
7 iadd
```



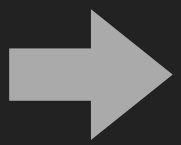
Stack

Java Bytecode

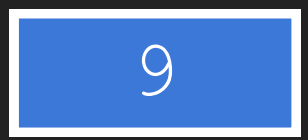
- Der Bytecode arbeitet mit einem "Operand Stack"
- Die Werte mit welchen gerechnet wird, befinden sich auf einem Stack und wir verarbeiten immer die obersten Elemente auf dem Stack.

Bytecode

```
1 iconst_4
2 iconst_2
3 iconst_3
4 imul
5 iconst_1
6 isub
7 iadd
```



Stack



Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```

Stack

Übersetzung

```
1  static int f(int a) {
2
3
4
5
6  }
```

Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```

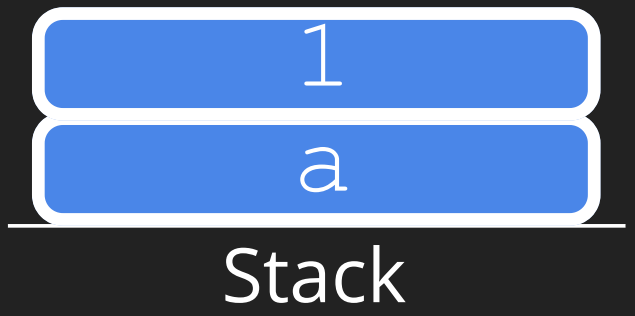
Übersetzung

```
1  static int f(int a) {
2          a
3
4
5
6  }
```

Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```



Übersetzung

```
1  static int f(int a) {
2          a     1
3
4
5
6  }
```

Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```

Stack

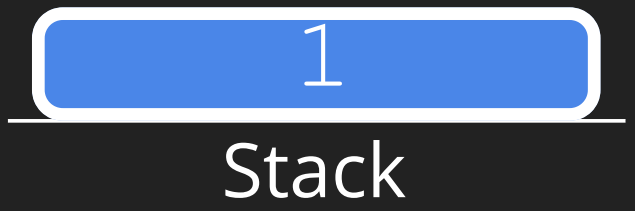
Übersetzung

```
1  static int f(int a) {
2      if(a <= 1){
3
4      }
5
6  }
```

Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```



Übersetzung

```
1  static int f(int a) {
2      if(a <= 1){
3                  1
4      }
5
6  }
```

Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```

Stack

Übersetzung

```
1 static int f(int a) {
2     if(a <= 1){
3         return 1;
4     }
5
6 }
```

Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```

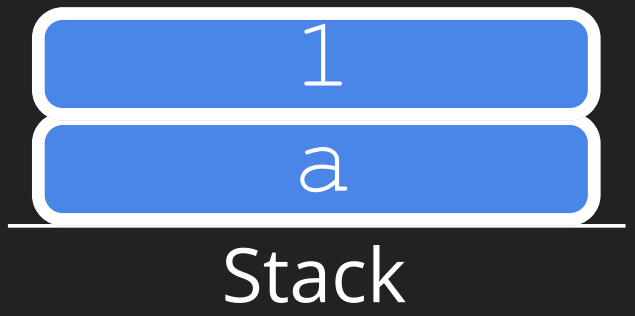
Übersetzung

```
1  static int f(int a) {
2      if(a <= 1){
3          return 1;
4      }
5                  a
6  }
```

Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```



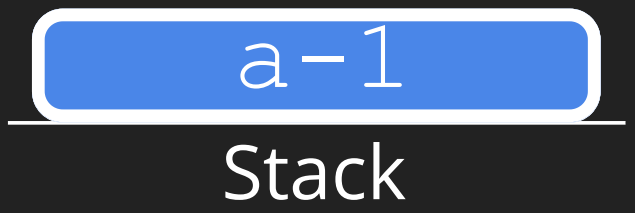
Übersetzung

```
1  static int f(int a) {
2      if(a <= 1){
3          return 1;
4      }
5                  a 1
6  }
```

Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```



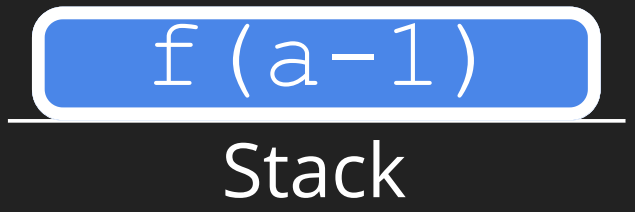
Übersetzung

```
1  static int f(int a) {
2      if(a <= 1){
3          return 1;
4      }
5                  a-1
6  }
```

Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```



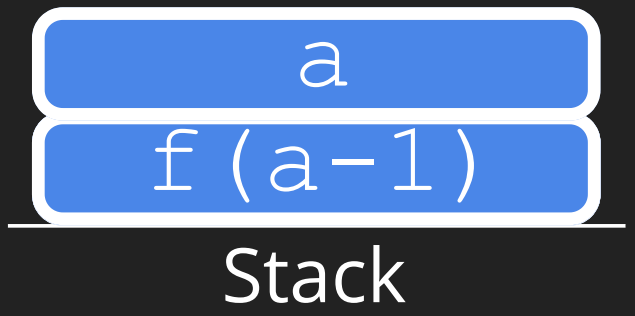
Übersetzung

```
1  static int f(int a) {
2      if(a <= 1){
3          return 1;
4      }
5              f(a-1)
6  }
```

Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```



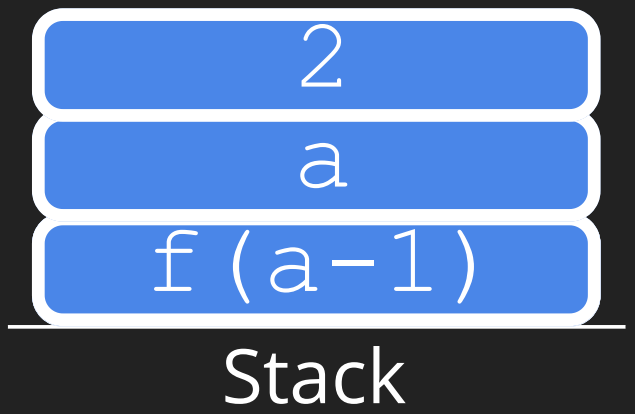
Übersetzung

```
1 static int f(int a) {
2     if(a <= 1){
3         return 1;
4     }
5             f(a-1)     a
6 }
```

Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```



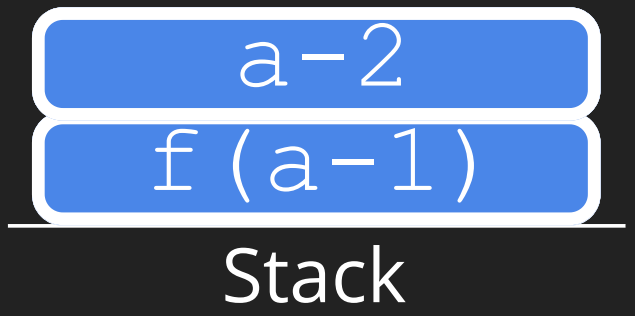
Übersetzung

```
1 static int f(int a) {
2     if(a <= 1){
3         return 1;
4     }
5             f(a-1)     a 2
6 }
```

Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```



Übersetzung

```
1 static int f(int a) {
2     if(a <= 1){
3         return 1;
4     }
5             f(a-1)     a-2
6 }
```

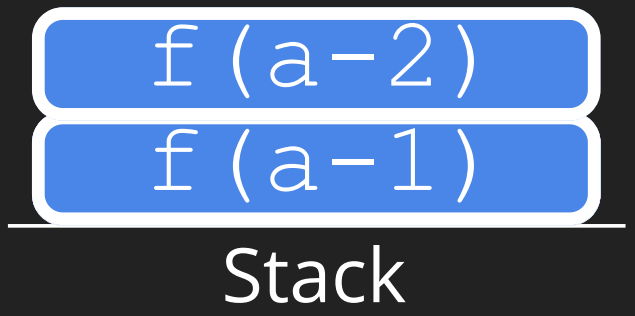
Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```

Übersetzung

```
1 static int f(int a) {
2     if(a <= 1){
3         return 1;
4     }
5             f(a-1)   f(a-2)
6 }
```



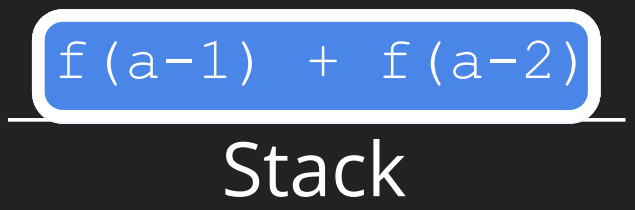
Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```

Übersetzung

```
1 static int f(int a) {
2     if(a <= 1){
3         return 1;
4     }
5             f(a-1) + f(a-2)
6 }
```



Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```

Stack

Übersetzung

```
1 static int f(int a) {
2     if(a <= 1){
3         return 1;
4     }
5     return f(a-1) + f(a-2);
6 }
```

Java Bytecode

Bytecode

```
1  static int f(int a);
2     0  iload_0 [a]
3     1  iconst_1
4     2  if_icmpgt 7
5     5  iconst_1
6     6  ireturn
7     7  iload_0 [a]
8     8  iconst_1
9     9  isub
10   10  invokestatic test.Main.f(
11   13  iload_0 [a]
12   14  iconst_2
13   15  isub
14   16  invokestatic test.Main.f(
15   19  iadd
16   20  ireturn
```

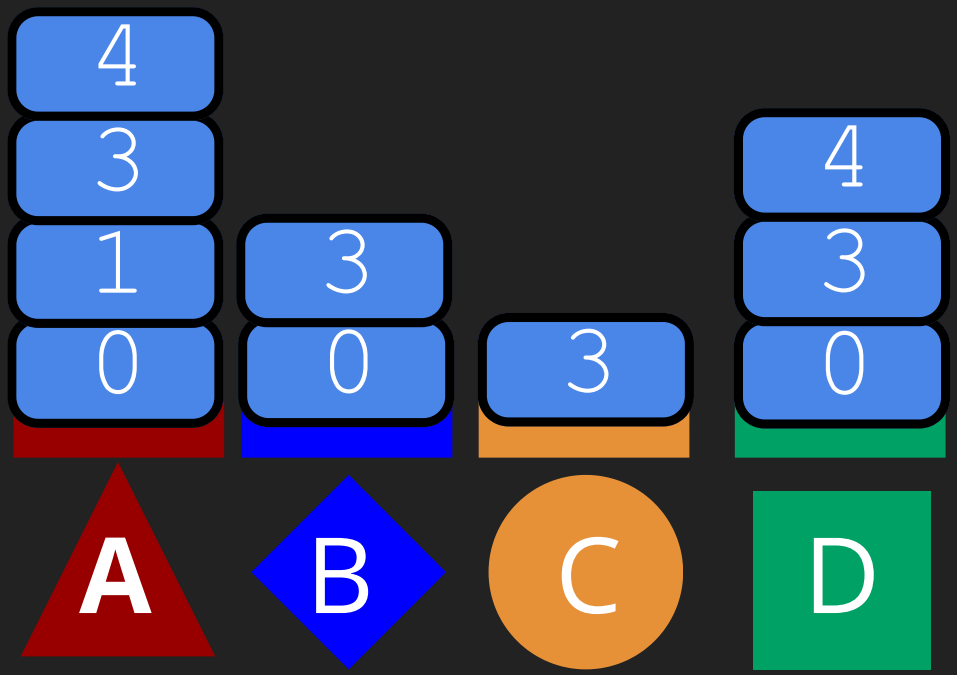
Stack

Übersetzung

```
1  static int f(int a) {
2      if(a <= 1){
3          return 1;
4      }
5      return f(a-1) + f(a-2);
6  }
```

Wie sieht der Stack nach den folgenden Operationen aus?

```
1  push(0);
2  push(1);
3  push(2);
4  pop();
5  peek();
6  pop();
7  push(3);
8  push(4);
9  peek();
10 pop();
```



Was macht iload_1?

- **Load an integer value from local variable 1**
- Load the integer value 1 onto the stack
- Add 1 to the top value of the stack
- Subtract 1 from the top value of the stack

Welcher ByteCode passt zu welchem Java Code?

```
0  iload_0
1  iconst_1
2  imul
3  ireturn
```

```
0  iload_0
1  iconst_1
2  if_icmple 7
5  iconst_1
6  ireturn
7  iconst_0
8  ireturn
```

```
0  iload_0
1  iconst_1
2  if_icmpge 7
5  iconst_1
6  ireturn
7  iconst_0
8  ireturn
```

```
return x > 1;
```

```
return x * 1;
```

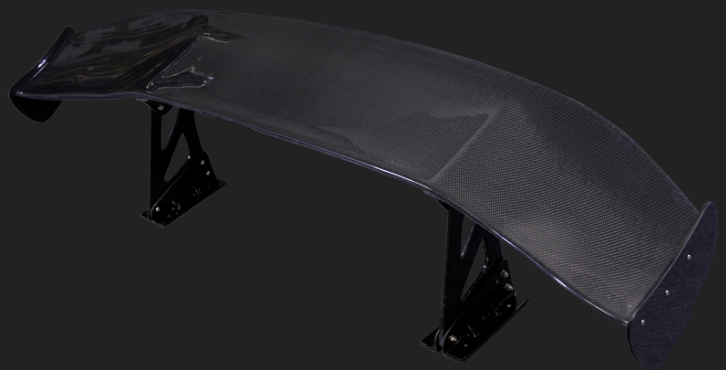
```
return x < 1;
```

Welcher ByteCode passt zu welchem Java Code?

```java
static int whichAnswerIsCorrect(int a){
        int i = a%2;
        int j = a*2;
        int k = a/2;
        return (j-2*i)/k;
}
```

```
 0  iload_0
 1  iconst_2
 2  irem
 3  istore_1
 4  iload_0
 5  iconst_2
 6  imul
 7  istore_2
 8  iload_0
 9  iconst_2
10  idiv
11  istore_3
12  iload_2
13  iconst_2
14  iload_1
15  imul
16  isub
17  iload_3
18  idiv
19  ireturn
```

Vorbesprechung

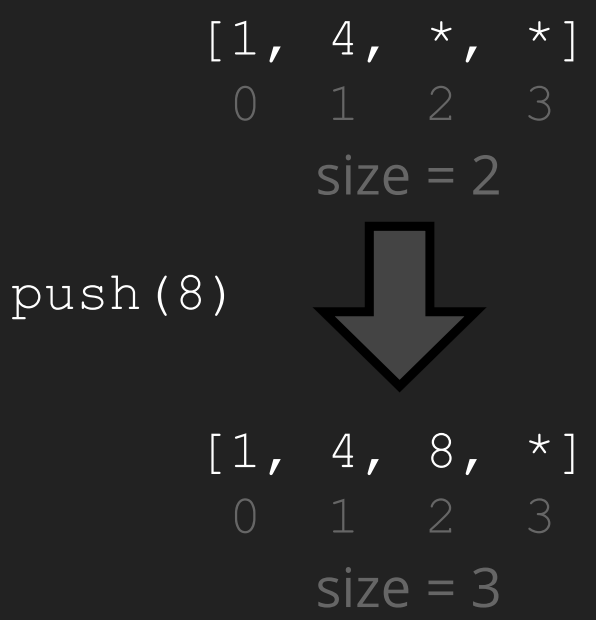


Ein wachsender Stack

push

Neue Daten oben auf den Stack legen

```
1 Stack myStack = Stack(4);
2
3  // […]
4
5  myStack.push(8)
```

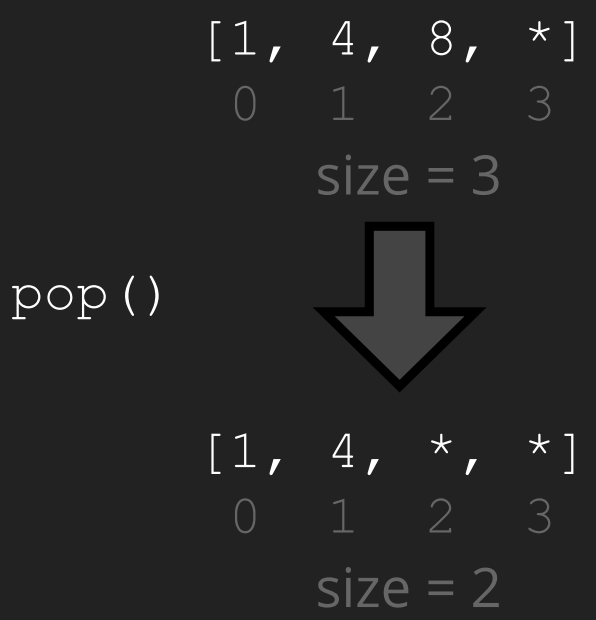


Ein wachsender Stack

pop

Daten von zuoberst im Stack entfernen

```
1 Stack myStack = Stack(4);
2
3  // […]
4
5  int myInt = myStack.pop();
```

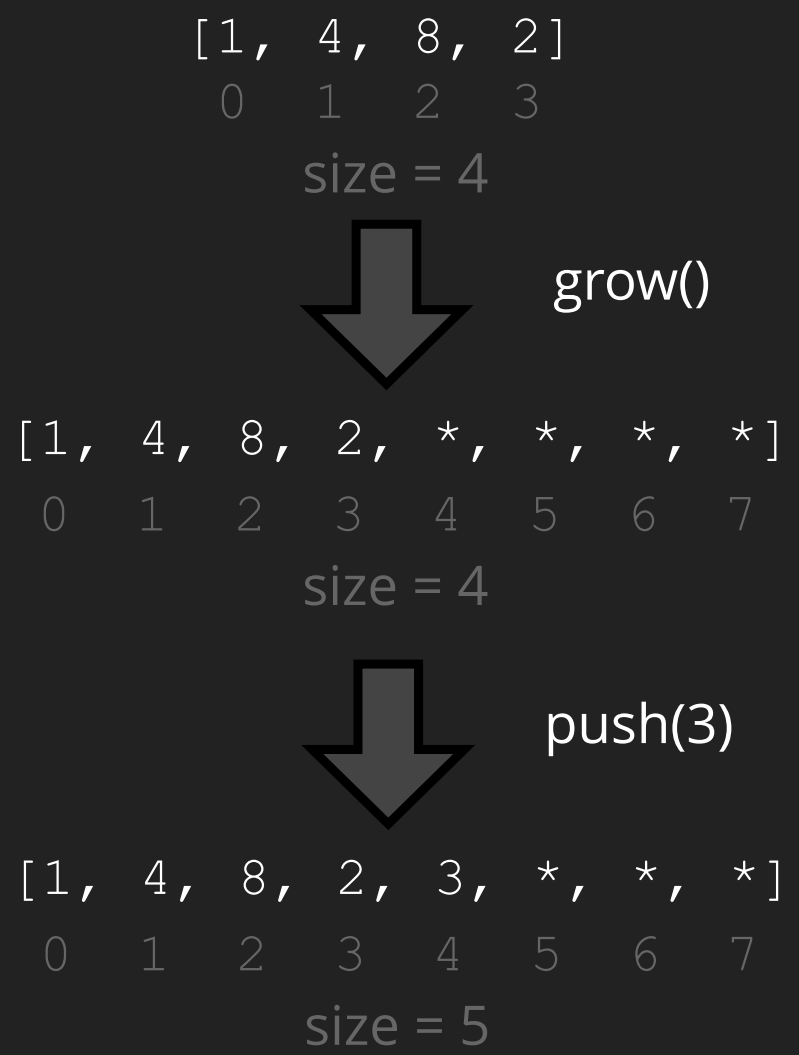


Ein wachsender Stack

grow

Falls der Array voll ist, und ein weiteres push aufgerufen wird, müssen wir den Platz vergrössern

```
1 Stack myStack = Stack(4);
2
3  // […]
4
5  int myInt = myStack.push(3);
```



Ackermann Funktion

- Gebt die Berechnungen für $A(2,1)$ in der eingerückten Form an
- Selbes Vorgehen wie im BestOf

view Slide

Ackermann als Stack

- Ackermann-Formel benötigt immer (genau) zwei Werte
- Die gerade benötigten Werte sollten also zuoberst auf dem Stack liegen
- Benutzt While Schleife
  - Abbruch wenn nur noch ein Element im Stack liegt → Resultat

```
1  Stack stack = new Stack(10);
2
3  // Anfangswerte auf Stack schieben
4  stack.push(n);
5  stack.push(m);
6
7  while(stack.size() > 1) {
8
9      // Neues n, m aus Stack auslesen
10
11     if(n == 0){
12         // A(0,m) = m + 1
13     }
14     else if(m == 0){
15         // A(n,0) = A(n-1,1)
16     }
17     else{
18         // A(n,m) = A(n-1,A(n,m-1));
19     }
20 }
```

Java-Bytecode

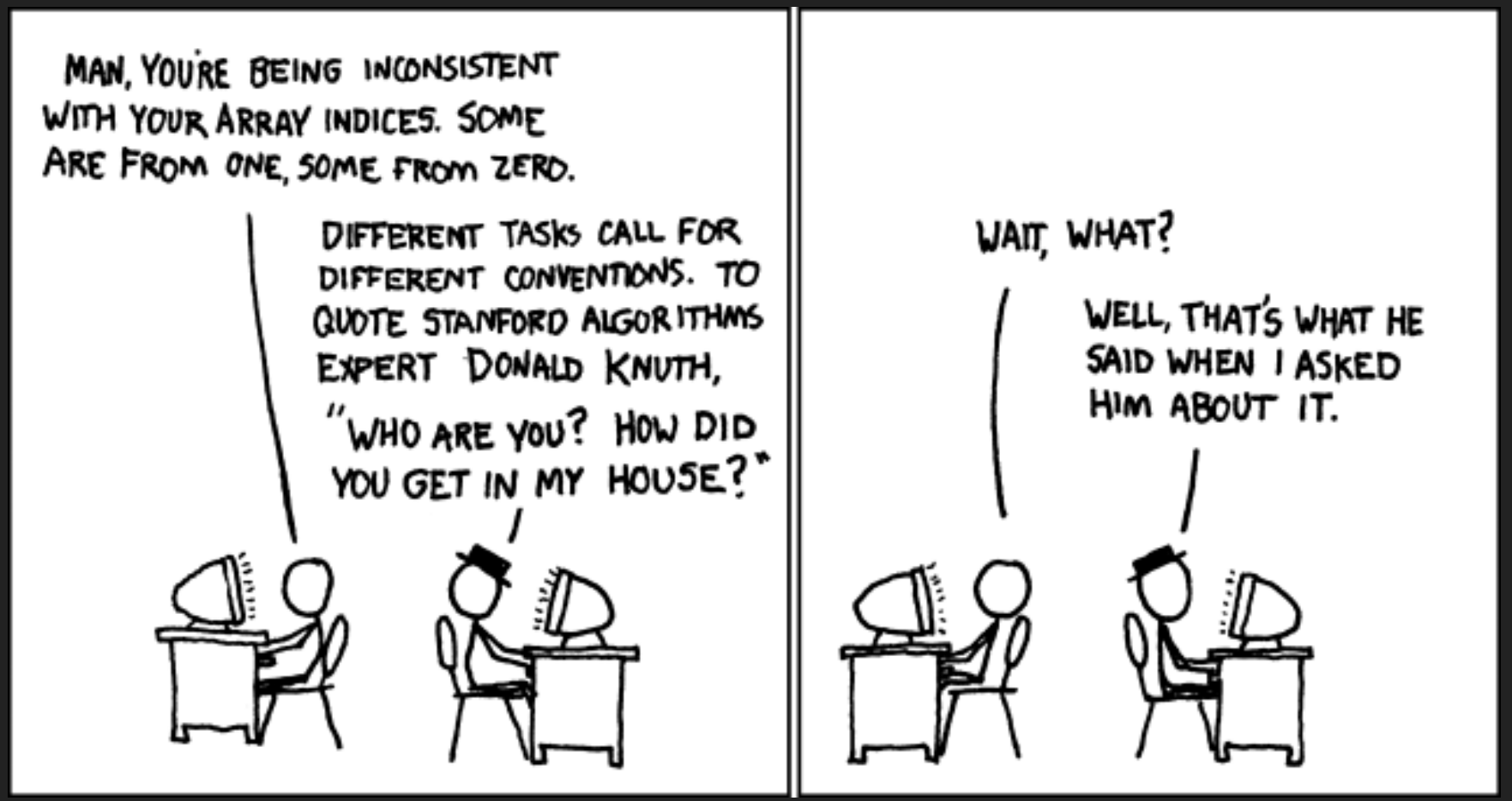
Wie kann man Java Code zu Bytecode umwandeln?

1. Konsole (Terminal):
   - `javap -c yourFile.java`
2. Eclipse:
   - Window → Show View → Navigator
   - Links oben Tab Navigator auswählen
   - Das .class File der Klasse suchen

```
1 Static int f(int a0,int a1,int a2){
2         return(a0+a1)/a2;
3 }
```

```
1   static int f(int a0, int a1, int a2);
2      0  iload_0
3      1  iload_1
4      2  iadd
5      3  iload_2
6      4  idiv
7      5  ireturn
```

Viel Spass!


MAN, YOU'RE BEING INCONSISTENT WITH YOUR ARRAY INDICES. SOME ARE FROM ONE, SOME FROM ZERO.
DIFFERENT TASKS CALL FOR DIFFERENT CONVENTIONS. TO QUOTE STANFORD ALGORITHMS EXPERT DONALD KNUTH, "WHO ARE YOU? HOW DID YOU GET IN MY HOUSE?"
WAIT, WHAT?
WELL, THAT'S WHAT HE SAID WHEN I ASKED HIM ABOUT IT.