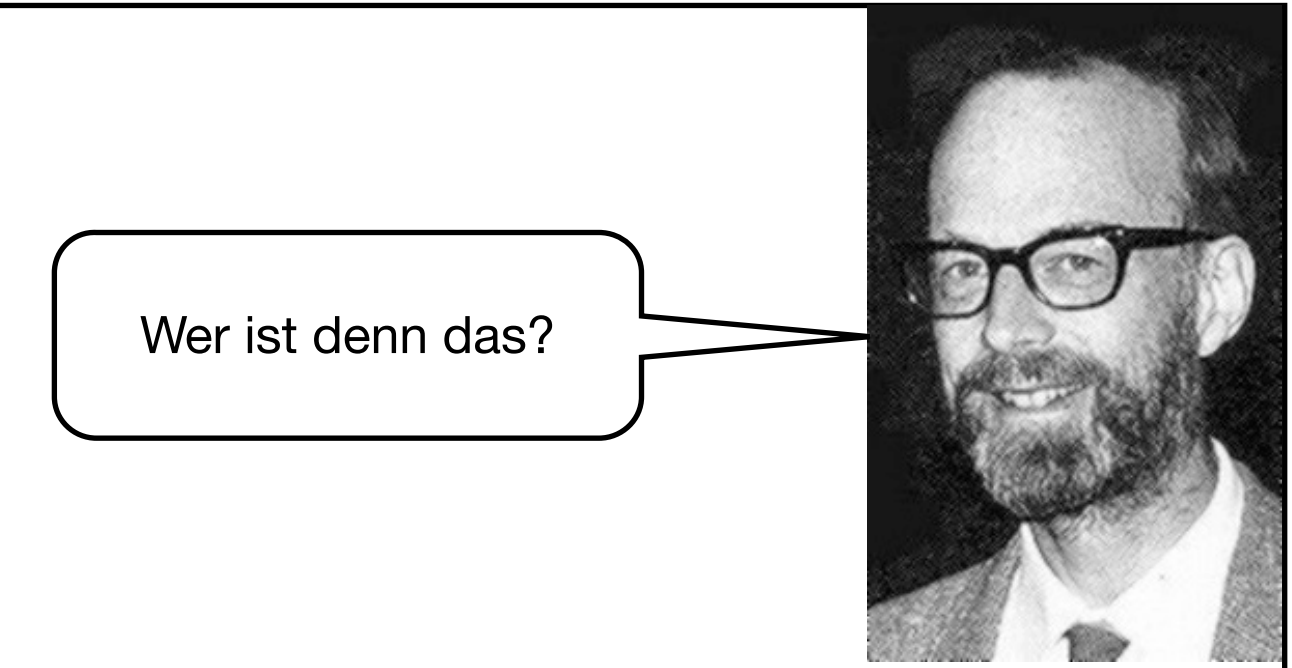


Spezifikation von Programmen

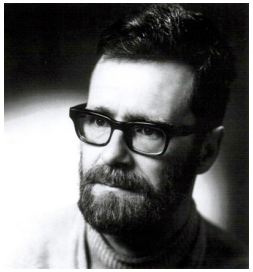
OOPM, Ralf Lämmel

Programmspezifikation mit Vor- und Nachbedingungen

Gesprächsprotokoll

“Joe the programmer”

Die beiden Leute führen ein Gespräch zur Korrektheit von Programmen.



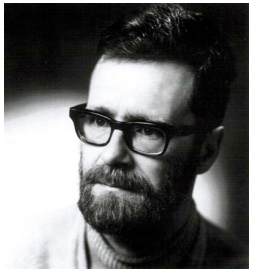
“Ed the computer scientist”

Die Frage nach der Korrektheit

Ich habe hier ein Programm für die Division mit Rest.

Ist dieses Programm korrekt?

```
q = 0;
r = x;
while  (r > y) {
  r = r - y;
  q = q + 1;
}
```



“Division mit Rest” ist zu informal.

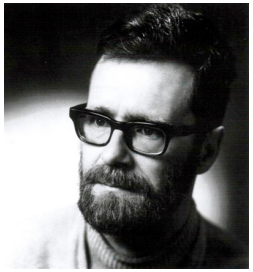
Können Sie das formaler ausdrücken?

Was ist also die **Spezifikation** für Ihr Programm?

Keine Korrektheit ohne Spezifikation

Das Programm soll den Quotient *q* und den Rest *r* für die Division der natürlichen Zahlen *x* und *y* berechnen.

```
q = 0;
r = x;
while  (r > y) {
  r = r - y;
  q = q + 1;
}
```



Haben Sie diese Spezifikation (Nachbedingung) im Sinn?

$\{ x == y * q + r \}$

Programm mit Spezifikation

```
q = 0;
r = x;
while  (r > y) {
  r = r - y;
  q = q + 1;
}
```

Es ist dies zu zeigen: $\{ x == y * q + r \}$

Nach Programmausführung gilt, dass ...

Nachbedingung

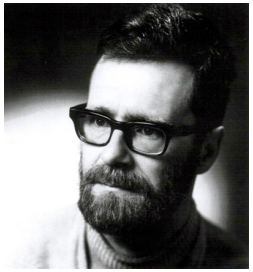
Zu schwache Vorbedingung

Das Programm ist nicht korrekt!

Es terminiert nicht für $y = 0$.

```
q = 0;
r = x;
while  (r > y) {
  r = r - y;
  q = q + 1;
}
```

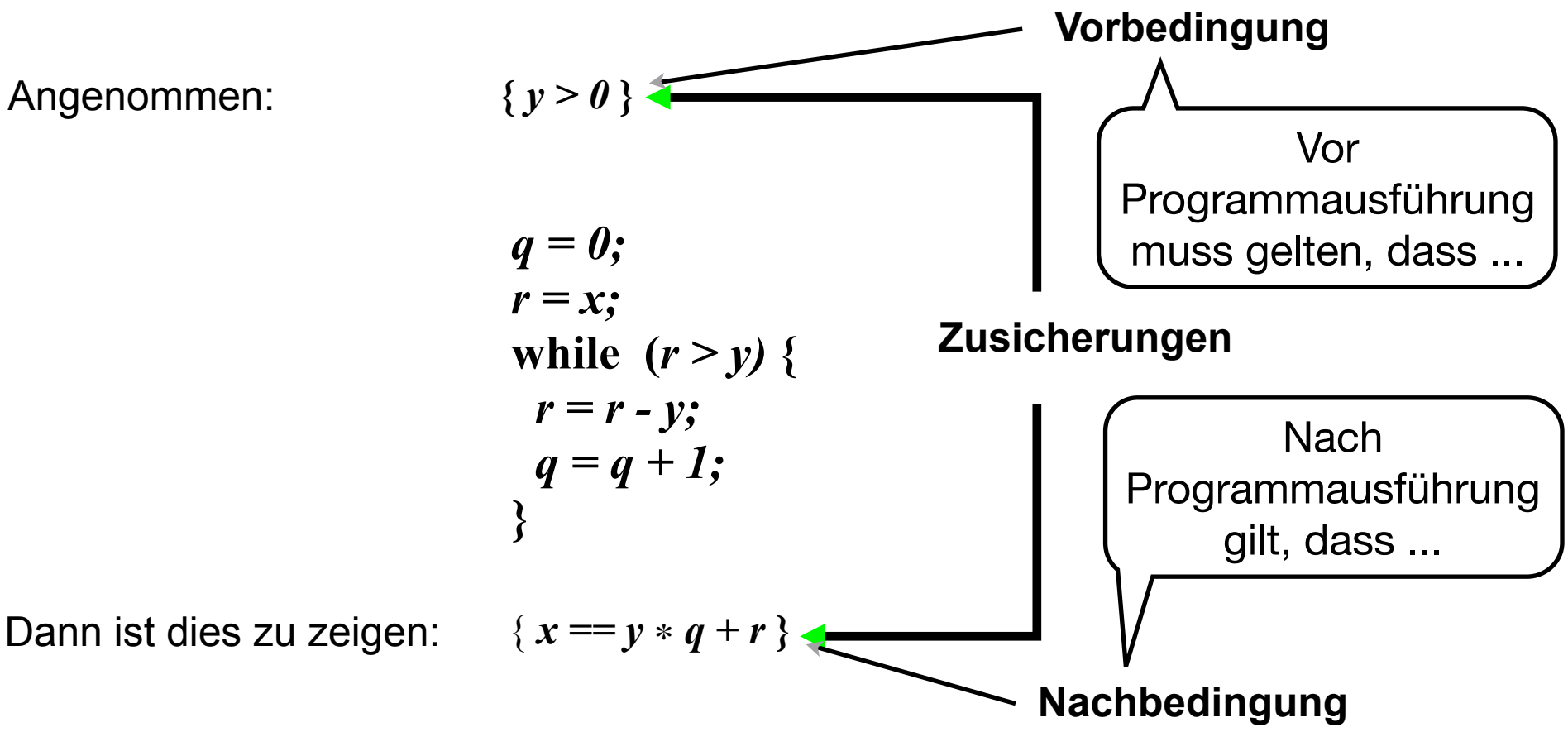
$\{ x == y * q + r \}$



Terminieren soll es also auch?

Sie sollten die Vorbedingung aufnehmen dass $y > 0$.

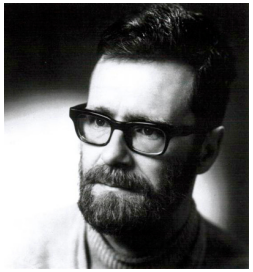
Programm mit Spezifikation V2



Zu schwache Nachbedingung + Bug

Für	$x = 6$ und $y = 3$
berechnet es	$q = 1$ und $r = 3$
anstatt	$q = 2$ und $r = 0$.

```
{ y > 0 }
q = 0;
r = x;
while (r > y) {
  r = r - y;
  q = q + 1;
}
{ x == y * q + r }
```



Sie wollen vielleicht, dass $r < y$ am Programmende gilt.

Ich kann Korrektheit dafür nicht beweisen.

Ihre Schleifenbedingung schaut komisch aus.

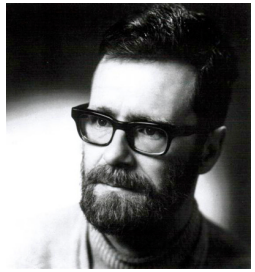
Programm mit Spezifikation V3

```
{ y > 0 }

q = 0;
r = x;
while (r >= y) {
  r = r - y;
  q = q + 1;
}

{ x == y * q + r && r < y }
```

Unterspezifikation

Ich möchte das Programm mit Datentyp int implementieren.
Muss ich dazu Anpassungen vornehmen?

```
{ y > 0 }
q = 0;
r = x;
while (r >= y) {
  r = r - y;
  q = q + 1;
}
{ x == y * q + r && r < y }
```



Sie solten wenigstens negative Werte ausschließen.
Eventuell wollen Sie das Programm verallgemeinern.

Programm mit Spezifikation V4

```
{ x >= 0 && y > 0 }

q = 0;
r = x;
while (r >= y) {
  r = r - y;
  q = q + 1;
}

{       x == y * q + r
 &&   r < y
 &&   r >= 0
 &&   q >= 0 }
```

Was tun?
(mit den Spezifikationen)

* Dokumentation
* **Überprüfen der Bedingungen zur Laufzeit**
* Verifizieren der Bedingungen vor Laufzeit
* Ableitung einer Implementation aus Spezifikation
* Verwendung der Spezifikation zur Testdatengenerierung

Programm ***ohne*** Spezifikation

```
// Variable declarations and test data
int x = 6;
int y = 3;
int q;
int r;

// The actual program (algorithm)
q = 0;
r = x;
while (r >= y) {
   r = r - y;
   q = q + 1;
}

// Print the result
System.out.println(
  x + " = " + y + " * " + q + " + " + r);
```

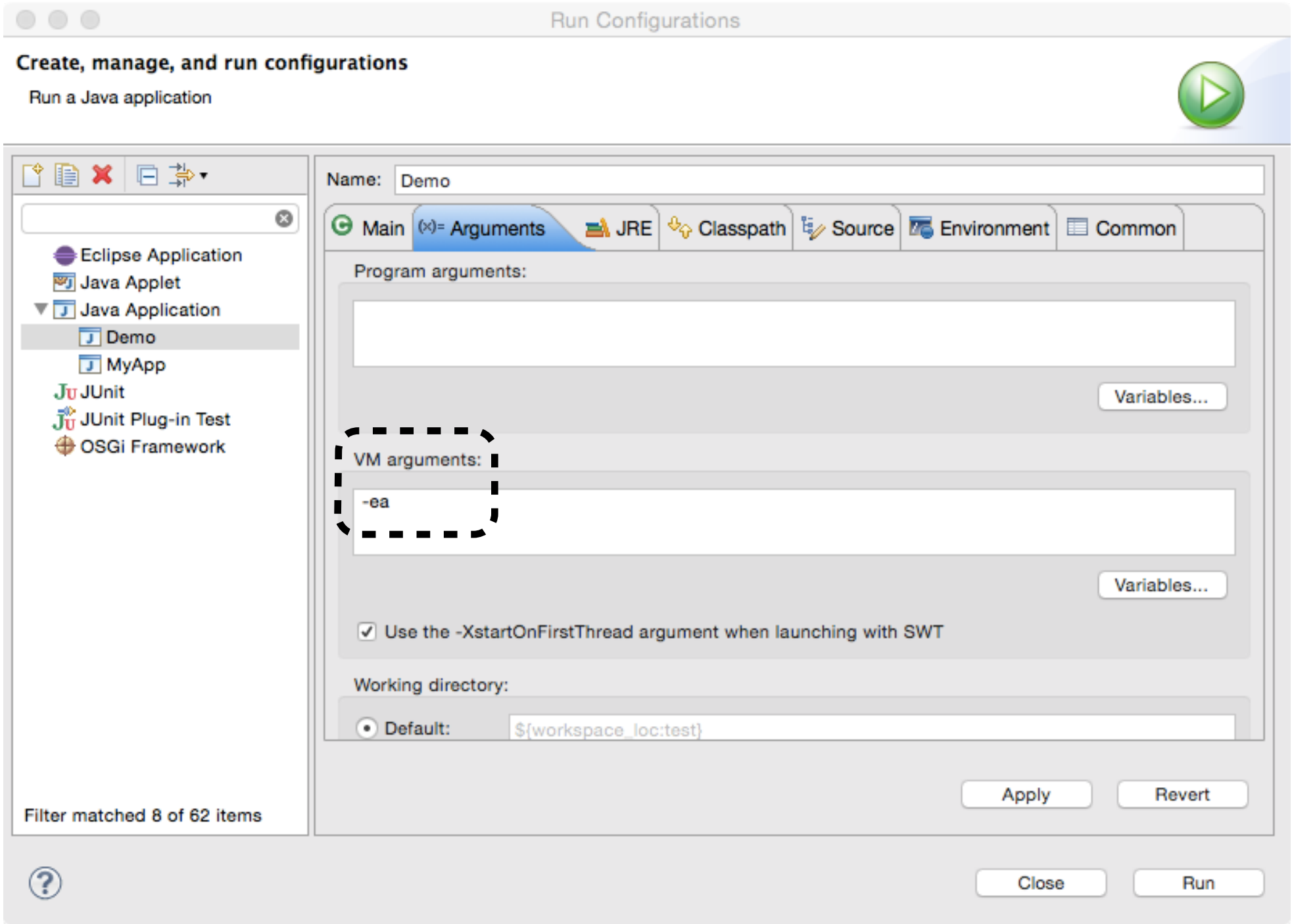
Programm ***mit*** Zusicherungen

Java's Zusicherungen ("Assertions") werden zur Laufzeit geprüft.

```
...
// Precondition
assert x >= 0 && y > 0;

// The actual program (algorithm)
q = 0;
r = x;
while (r >= y) {
   r = r - y;
   q = q + 1;
}

// Postcondition
assert x == y * q + r
    && r < y
    && r >= 0 && q >= 0;
...
```

Java's assert

- Anweisungsform:
  - **assert** *BoolescherAusdruck*;
- Zur Laufzeit:
  - Berechne Ausdruck.
  - Wirf Ausnahme wenn das Resultat false ist.
- Beachte:
  - Überprüfung verlangt VM Parameter:
    - -enableassertions


Run Configurations
Create, manage, and run configurations
Run a Java application
Name: Demo
Main
Arguments
JRE
Classpath
Source
Environment
Common
Program arguments:
Variables...
VM arguments:
-ea
Variables...
Use the -XstartOnFirstThread argument when launching with SWT
Working directory:
Default:
${workspace_loc:test}
Eclipse Application
Java Applet
Java Application
Demo
MyApp
JUnit
JUnit Plug-in Test
OSGi Framework
Filter matched 8 of 62 items
Apply
Revert
Close
Run

Verwendung von Spezifikationen für Laufzeitüberprüfungen

Beispiel: Korrektheit von BubbleSort

- Gegeben ist die Sortiermethode:
  - `static void bubbleSort(int[] a) { ... }`
- Ist diese Methode ***korrekt***?
- Was ist das Korrektheitskriterium?

Korrektheit des Sortieralgorithmus

- Für alle **`int[] a, b`**,
- wobei **`a`** und **`b`** Klone voneinander sind,
- gilt nach der Ausführung von

  `bubbleSort(a)`,
- dass:
  - **`isSorted(a)`** und
  - **`isPermutation(a,b)`**

Die Formulierung würde sich ändern, wenn **`bubbleSort`** ohne Seiteneffekte arbeitete.

Kontrollierende Anwendung von BubbleSort

```
int[] b = a.clone();
bubbleSort(a);
assert isSorted(a)
    && isPermutation(a,b);
```

Verschieben der Zusicherungen in die Methode zum Sortieren

```
public static void bubbleSort(int[] a) {
  int[] b = a.clone();
    boolean swapped; // to notice swaps during a pass
    do {
       swapped = false;
       for (int i = 1; i < a.length; i++)
          if (a[i - 1] > a[i]) {
             // Swap!
             int swap = a[i];
             a[i] = a[i - 1];
             a[i - 1] = swap;
             swapped = true;
          }
    } while (swapped); // another pass if swaps happened
  assert isSorted(a) && isPermutation(a,b);
}
```

Das Ausschalten von „assert“ vermeidet nicht das Klonen. Wir kann man dies ändern?

Beispiel: Korrektheit von Suchverfahren

- Relevante Algorithmen
  - Lineare Suche (mit Indexrückgabe)
  - Binäre Suche (mit Indexrückgabe)
- ***Erfassung des Fortschritts der Suche.***
- Verwendung einer einfachen Hilfseigenschaft.
  - public static boolean **member**(

    int[] a, int x, -- Reguläre Eingabe

    int from, int to) -- Einschränkung der Indizes

Hier erfassen wir keine allgemeinen Vor- und Nachbedingungen sondern andere Eigenschaften an verschiedenen Stellen des Programms.

```
/**
 * @param a an array to search for x
 * @param x the element to search in a
 * @param from the start index of a for the window of searching
 * @param to the end index of a for the window of searching
 * @return Boolean to say whether x is a member of a
 */
public static boolean member(int[] a, int x, int from, int to) {
   for (int i=from; i<=to; i++)
      if (a[i]==x)
         return true;
   return false;
}
```

Hilfsmethode zur Benutzung in Spezifiaktionen. (Was ist wenn diese Methode “falsch” ist?)

Lineare Suche ***ohne*** Zusicherungen

```
public static int linear(int[] a, int x) {
   for (int i=0; i<a.length; i++)
      if (a[i] == x)
         return i;
   return -1;
}
```

Lineare Suche ***mit*** Zusicherungen

```
public static int linear(int[] a, int x) {
  for (int i=0; i<a.length; i++) {
    assert !member(a,x,0,i-1);
    if (a[i] == x) {
      assert member(a,x,i,i);
      assert member(a,x,0,a.length-1);
      return i;
    }
    assert !member(a,x,0,i);
  }
  assert !member(a,x,0,a.length-1);
  return -1;
}
```

Binäre Suche ***ohne*** Zusicherungen

```
public static int binary(int[] a, int x) {
   int first = 0;
   int last = a.length - 1;
   while (first <= last) {
      int middle = first + ((last - first) / 2);
      if (a[middle] < x) {
         first = middle + 1;
      } else if (a[middle] > x) {
         last = middle - 1;
      } else
         return middle;
   }
   return -1;
}
```

Binäre Suche ***mit*** Zusicherungen

```
public static int binary(int[] a, int x) {
   int first = 0;
   int last = a.length - 1;
   while (first <= last) {
      int middle = first + ((last - first) / 2);
      if (a[middle] < x) {
        assert !member(a,x,first,middle);
         first = middle + 1;
      } else if (a[middle] > x) {
        assert !member(a,x,middle,last);
         last = middle - 1;
      } else
         return middle;
   }
   assert !member(a,x,0,a.length-1);
   return -1;
}
```

- **Zusammenfassung**
  - Programme mit Spezifikationen
    - Vor-, Nach- und „Zwischen-“ Bedingungen
  - Laufzeitüberprüfung von Spezifikationen
    - Auch Assertion-basiertes Debugging
- **Ausblick**
  - Tests als Spezifikationen
  - Statische Verifikation von Spezifikationen