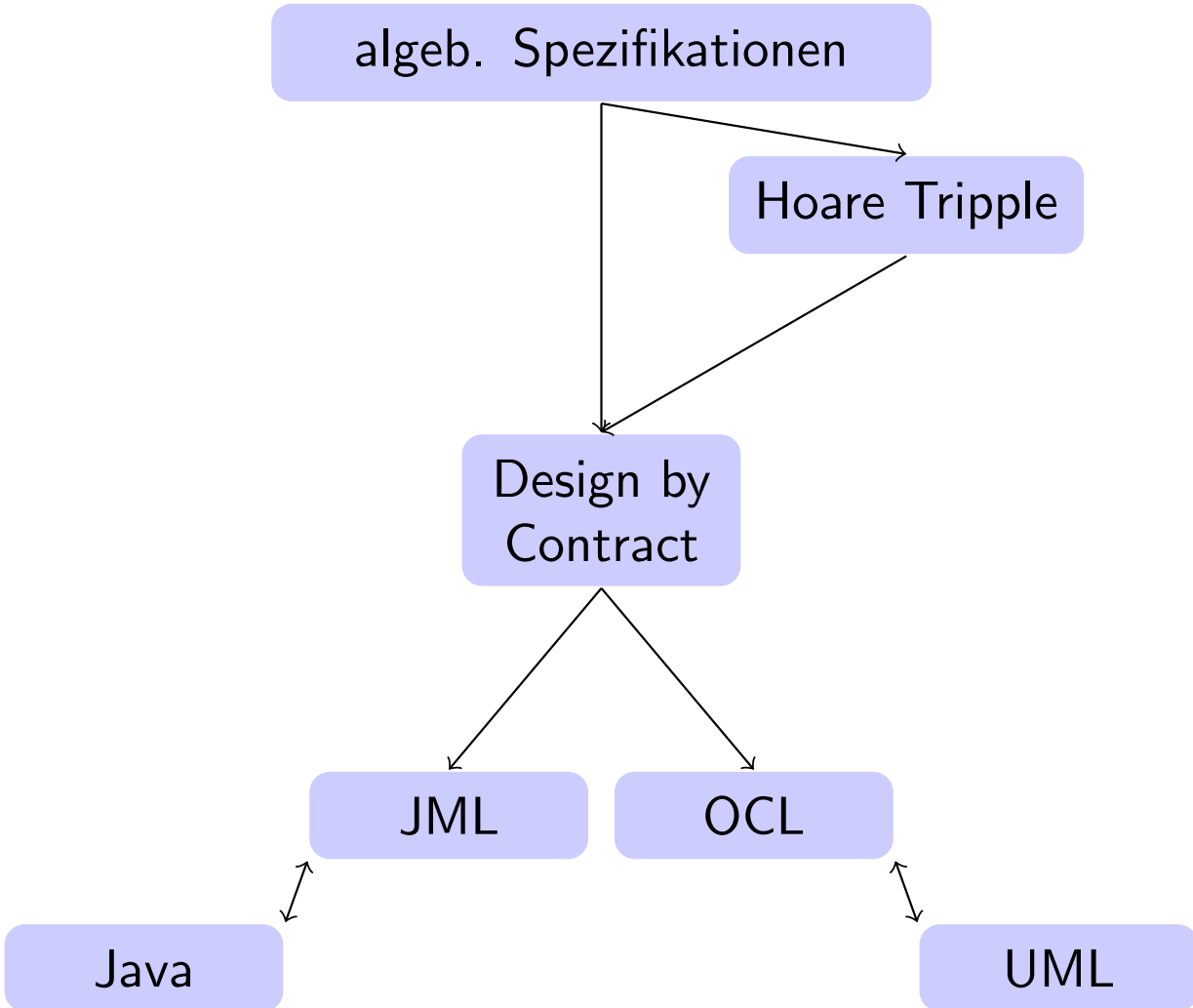
Java Modeling Language (JML) und Design by Contract (DbC)

Thomas Ruhroth

16. Juni 2011

Übersicht - Inhaltliche Abhängigkeiten



Von algb. Spezifikationen zu Design by Contract

Vorgehen

- Ausgang: algb. Spezifikationen
- + Nutzung von Prädikatenlogik 1. Ord.
- + Abstraktion
- Ziel: Design by Contract

Mein Beispiel

Test, ob in einem Integer-Array nur positive Zahlen enthalten sind.

Ihr Beispiel

Bestimmen des maximalen Wertes in einem Integer-Array.

Int-Array

Array

algebra *IntArray* introduces sorts $int, nat0, bool, IntArray$;
operations
$create : nat0 \to IntArray$
$set : IntArray, nat0, int \to IntArray$
$get : IntArray, nat0 \to int$
$size : IntArray \to nat0$
constraints $create, set, get, size$
so that for all st : $IntArray, m : int;\ n : nat0$
$size(create(n) = n$
$n < size(st) \wedge size(get(st, n)) = size(st)$
$n < size(st) \wedge get(set(st, n, m), n) = m$

isPositive

operations
...
$isPositive : IntArray \to bool$
$tailarray(st) : IntArray \to IntArray$
constraints ..., $isPositive$
...
$size(st) = 0 \Rightarrow isPositive = true$
$size(st) > 0 \Rightarrow isPositive = isPositive(tailarray(st))$ and $get(st, 0) > 0$

isPositive

operations

...

$isPositive : IntArray \rightarrow bool$

constraints ..., $isPositive$

...

$$\forall\, i \in int \mid 0 \leq i < size(st) \bullet get(st, i) > 0$$
$$\wedge\ size(st) > 0 \Leftrightarrow isPositive(st)$$

isPositive

operations ...
$isPositive : IntArray \to bool$
constraints ..., *isPositive* ...
$(\forall\, i \in int \mid 0 \leq i < size(st) \bullet get(st, i) >= 0)$
$\Leftrightarrow isPositive(st)$

Maximum

operations ...
$max : IntArray \to int$
constraints ..., *max* ...
$\exists\, i \in int \mid 0 \leq i < size(st) \bullet max(st) = get(st, i)$
$\forall\, j \in int \mid 0 \leq i < size(st) \bullet get(st, i) <= max(st)$

Achtung!

Das sind keine algb. Spezifikationen im engeren Sinn mehr!
Aufpassen in der Klausur!

Maximum

operations ...
$max : IntArray \to int$
constraints ..., max ...
$\exists\, i \in int \mid 0 \leq i < size(st) \bullet max(st) = get(st, i)$
$\forall\, j \in int \mid 0 \leq i < size(st) \bullet get(st, i) <= max(st)$

Die Aussage

- für alle!
- Vorbedingung: $size(st) > 0$
- Nachbedingung:

$\exists\, i \in int \mid 0 \leq i < size(st) \bullet max(st) = get(st, i)$
$\forall\, j \in int \mid 0 \leq i < size(st) \bullet get(st, i) <= max(st)$

Design by Contract

Für jede Methode/Operation/Funktion

- Die Vorbedingung beschreibt die Bedingungen, damit die Methode/Operation/Funktion richtig funktioniert
- Die Nachbedingung beschreibt den Zustand, falls die Funktion richtig funktioniert hat

Wenn ein Operationsaufruf die Vorbedingung beachtet, kann der Aufrufer sich darauf verlassen, dass die Nachbedingung erfüllt wird!

Weitere Angaben

- Invariante: Bedingungen, die von einer Einheit (Klasse, Modul, Block...) immer erfüllt werden müssen.
- Angabe von Dingen, die nicht passieren dürfen:
  - Schreibverbot für Variablen
  - ...

Sprachen und Erweiterungen

Sprachen

- Eiffel: Sprache baut direkt auf Design by Contract auf
- D: Design by Contract Unterstützung

Sprachen

- Java: JML - Java Modeling Language
- .NET: Code Contracts (MS)
- ...

DbC und Object-Orientierung

- Klassen können Invarianten besitzen
- Unterklassen sind behavioural Subtyps

Behavioural Subtypes

Eine Klasse bzw. deren Operationen können sich nur konform zur Oberklasse verhalten.

- Jede Invariante der Oberklasse ist eine Invariante der Unterklasse.
- Die Vorbedingung der Operationen der Oberklasse impliziert die Vorbedingung der Unterklasse.
- Die Nachbedingung der Operationen der Unterklasse implizieren die Nachbedingungen der Oberklasse.

Achtung!

Unterklassen in Programmiersprachen sind im Allgemeinen keine Behavioural Subtypes!

Behavioural Subtypes: Beispiel

Oberklasse

- var int i
  Invariant : $abs(i) < 10$
- op inc
  pre: $0 <= a \wedge a < 2 \wedge$
  $\texttt{old}(i) < 6$
  post: $\texttt{old}(i) + a == \texttt{new}(i)$

Unterklasse

- var int i
  Invariant : $abs(i) < 20$
  Achtung hier gilt die Inv der Oberklasse!
- op inc
  pre: $0 <= a \wedge a < 2 \wedge$
  $\texttt{old}(i) < 7$
  post: $\texttt{old}(i) + a ==$
  $\texttt{new}(i) \wedge \texttt{new}(i) < 8$

JML

Java Modeling Language

JML ist eine DbC-Erweiterung für Java

Tools

Es gibt eine reiche Toolunterstützung:

- Laufzeitprüfung (jmlrc, jml2tools, rac)
- Statische Analyse
- Beweissysteme (Key, Daphne)

JML: Eine Einführung

JML-Kommentare

```
/*@ JML-Code
  @ JML-Code
  @*/
```

- Code bleibt mit Standard-Tools kompilierbar
- JML-Tools können JML-Code auswerten
- Ähnlich wie JavaDoc

Einfaches Beispiel

```
//@ requires x >= 0.0;
/*@ ensures JMLDouble
    .approximatelyEqualTo(x, \result * \result, eps);
@*/
public static double sqrt(double x) {
return Math.sqrt(x);
}
```

Ausdrücke

Java Ausdrücke

- Prädikate: `x >= 0.0`
- Boolesche Verknüpfungen: `a != null && a == b+7`
- Achtung: Nur seiteneffektfreie Methoden und Operationen: Kein $=$, $+=$, $++$, ...

Erweiterungen

- Variablen: `\result`
- Boolesche Operatoren: `<==>`,
- Quantoren: `\forall`
- Modifikatoren für Variablen: `\old(.)`

JML- Erweiterte Ausdrücke

Operatoren

`\result`	Ergebnis eines Methodenaufrufs
`a ==> b`	a impliziert b
`a <== b`	b impliziert a
`a <==> b`	a genau dann wenn b
`a <=!=> b`	not (a genau dann wenn b)
`\old(E)`	Wert von E im Vorzustand

JML- Quantoren

Operatoren

`\forall`	Allquantor ($\forall$)
`\exists`	Existenz-Quantor ($\exists$)
`\sum`	Summe ($\sum$)
`\product`	Produkt ($\prod$)
`\no_of`	Anzahl

Achtung, müssen immer in Klammern stehen und haben drei Teile.
Ähnlich wie eine For-Anweisung in Java.

```
(\forall int i; i < 10 && i > 0; p(i))
```

Beispiel

Aufgabe:

Bestimme die Häufigkeit eines Zahlenwertes in einem Integer-Array.

- Normales Verhalten: `normal_behaviour`

Beispiel

Aufgabe:

Bestimme die Häufigkeit eines Zahlenwertes in einem Integer-Array.

Eine Lösung

```
/*@ normal_behaviour
  @ requires a != null;
  @ ensures \result == (\no_of int j; j >= 0 && j < a.size(
                                      a[j] == i );
  @ exceptional_behaviour
  @ requires a == null;
  @ signal_only  Exception;
  @*/
public int count(int[] a, i) throws Exception
```

- Normales Verhalten: `normal_behaviour`

Aufgaben

Spezifizieren sie die Methode ggT:

```
public int ggT (int a, int b) { ...
```

Aufgaben

Spezifizieren sie die Methode ggT:

```
public int ggT (int a, int b) { ...
```

Eine Lösung:

```
\*@ normal_behaviour requires a > 0 && b > 0;
  @ ensures a % \result == 0
                  && b % \result == 0
                  && (\forall int i;
                        i <= a && i <= b && i > \result;
                        a % i != 0 || b % i != 0)
```

Aufgabe

Bestimme das erste Auftreten eines Zahlenwertes in einem Integer-Array.

```
public int first(int[] a, i) throws Exception
```

Eine Lösung:

```
/*@ normal_behaviour
  @ requires a != null && (\exists int v; j < a.size()
                                    && j >= 0  ; a[j] == i);
  @ ensures (/forall int j; j >= 0 && j < \result;
                                 a[j] != i) && a[\result] != i;
  @ normal_behaviour
  @ requires a != null && (\forall int j; j < a.size()
                                       && j >= 0 ;a[j] != i);
  @ ensures \result == -1;
  @ exceptional_behaviour
  @ requires a == null;
  @ signal_only  Exception;
  @*/
public int first(int[] a, i) throws Exception
```

Pure Methoden

Seiteneffekte

Ein Seiteneffekt ist eine Änderung des inneren Zustandes einer Klasse oder eines Systems

JML-Ausdrücke und Seiteneffekt

Für die Auswertung eines JML-Ausdruckes muss die Operation ausgeführt werden können. Dabei ist es nicht gewollt, dass sich der Zustand des Systems ändert.

Schlüsselwort `pure`

`pure` vor einer Methodendefinition verbietet Seiteneffekte. Dies Methode kann in JML-Ausdrücken verwendet werden.

Beispiel

```
/*@ requires x >= 0.0;
  @ ensures JMLDouble
     .approximatelyEqualTo(x, \result * \result, eps);
@*/
public static /*@ pure */ double sqrt(double x) {
return Math.sqrt(x);
}
```

Aufgaben

Spezifizieren Sie die Methode kgV:

```
public int kgV (int a, int b) { ...
```

Aufgaben

Spezifizieren Sie die Methode kgV:
`public int kgV (int a, int b) { ...`

Eine Lösung:

```
\*@ normal_behaviour requires a > 0 && b > 0;
  @ ensures \result == a * b / ggT (a,b);
```

Achtung: ggT muss als pure definiert sein.

Aufgaben

Bestimme das Maximum eines Integer-Arrays:

```
public int max(int [] a);
```

Bestimme die Häufigkeit des Maximum eines Integer-Arrays:

```
public int no_max(int [] a);
```

Aufgaben

Eine Lösung:

```
/*@ requires a != null && (array.size() > 0)
  @ ensures (/forall  int j; j < array.size();
                                a[j] <= \result)
    && /\exists int i| 0 <= i && i < a.size();
                                \result == a[i];  @*/
public /*@ pure */ int max(int[] a, i) { ...

/*@ requires a != null && (array.size() > 0)
  @ ensures \result == count(a,max(a)) \\
  @*/
  // Achtung count muss mit pure gekennzeichnet sein...
  public int no_max(int [] a) { ...
```