

Tutorium 12: Design by Contract

Paul Brinkmeier

01. Februar 2022

Tutorium Programmierparadigmen am KIT

Cheatsheet: Design by Contract

ProPa-Stoff zu Design by Contract:

- Grundlagen: Pre-/Postconditions, Caller, Callee
 - A.K.A.: *Vor-/Nachbedingungen, Aufrufer, Aufgerufener*
- JML (Java Modeling Language):
 - @ requires
 - @ ensures (mit \old und \result)
 - @ invariant
 - /*@ pure @*/, /*@ nullable @*/, /*@ spec_public @/
 - Quantoren: \forall, \exists
 - ...
- Liskovsches Substitutionsprinzip

JML-Klausuraufgabe

Klausur 19SS, Aufgabe 6d (3P.)

```
class MaxAbsCombinator {
    //@ requires left != Integer.MIN_VALUE;
    //@ requires right != Integer.MIN_VALUE;
    //@ ensures \result <= left || \result <= right;
    //@ ensures \result >= left && \result >= right;
    int combine(int left, int right) {
        return Math.max(Math.abs(left), Math.abs(right));
    }
}
```

(d) Der Vertrag der Methode `combine` wird *vom Aufrufenen* verletzt. Begründen Sie dies und geben Sie an, wie die verletzte Nachbedingung angepasst werden könnte.

Klausur 19SS, Aufgabe 6e (2P.)

```
class MaxAbsCombinator {
    //@ requires left != Integer.MIN_VALUE;
    //@ requires right != Integer.MIN_VALUE;
    //@ ensures \result <= left || \result <= right;
    //@ ensures \result >= left && \result >= right;
    int combine(int left, int right) {
        return Math.max(Math.abs(left), Math.abs(right));
    }
}

new MaxAbsCombinator().combine(
    random.nextInt(),
    random.nextInt());
```

(d) Wird der Vertrag hier *vom Aufrufer* erfüllt? Begründen Sie kurz.

JML

@ requires

```
//@ requires b != 0;  
int divide(int a, int b) {  
    return a / b;  
}
```

- @ requires definiert eine Vorbedingung für eine Methode.
- Vorbedingungen müssen vom Aufrufer erfüllt werden.

```
//@ ensures \result.length() == s.length();  
String[] reverse(String [] s) { ... }  
  
//@ requires amount > 0;  
//@ ensures balance > \old(balance);  
void deposit(int amount) {  
    this.balance += amount;  
}
```

- @ ensures definiert eine Nachbedingung für eine Methode.
- Nachbedingungen müssen vom Aufrufenen erfüllt werden.
- Mit \old und \result werden Beziehungen zwischen Ursprungszustand, Rückgabewert und neuem Zustand eingeführt.


```
class FixedSizeList<A> {  
    //@ invariant elementCount <= elements.length;  
    A[] elements;  
    int elementCount;  
}
```

- @ invariant definiert Invarianten für eine Klasse.
- Diese können bspw. wiederverwendet werden, um Vorbedingungen für Methoden zu erfüllen.

```
/*@ pure @*/
```

```
class ResizingArray<A> {  
    private A[] elements;  
    private int elementCount;  
    /*@ pure @*/ public int getElementCount();  
  
    /*@ ensures getElementCount() ==  
    /*@          \old(getElementCount()) + 1;  
    public void add(A element) { ... }  
}
```

- Verträge sind implizit public.
 \leadsto private-Attribute nicht verwendbar
- Um Getter-Funktionen in Verträgen nutzen zu können,
 müssen diese frei von Seiteneffekten und mit `/*@ pure @*/`
 markiert sein.

```
/*@ spec_public @*/
```

```
class ResizingArray<A> {  
    private A[] elements;  
    private /*@ spec_public @*/ int elementCount;  
  
    /*@ ensures elementCount ==  
    /*@          \old(elementCount) + 1;  
    public void add(A element) { ... }  
}
```

- Alternative: private-Attribute als /*@ spec_public @*/ markieren.
- Immer noch private, können vom Checker aber trotzdem gesehen werden.

```
/*@ requires \forall int i;  
    0 <= i && i < xs.length;  
    xs[i] != null;  
    ensures xs.length == 0 ==> \result == 0;  
    @*/  
int totalLength(String[] xs) {  
    ...  
}
```

- Für das Arbeiten mit Aussagen in Verträgen gibt es ein paar Helferchen:
 - `\forall decl; <cond>; <expr>`
 - `\exists decl; <cond>; <expr>`
 - `<cond> ==> <expr>`

Übungsaufgabe 1 — Set

- `demos/java/jml/Set.java`
- Behebt alle Compiler- und Laufzeitfehler in der Klasse `Set`.

Übungsaufgabe 1 — Set

- `demos/java/jml/Set.java`
- Behebt alle Compiler- und Laufzeitfehler in der Klasse `Set`.
- Achtet darauf, dass die gegebenen JML-Verträge erfüllt sind.

Übungsaufgabe 1 — Set

- `demos/java/jml/Set.java`
- Behebt alle Compiler- und Laufzeitfehler in der Klasse `Set`.
- Achtet darauf, dass die gegebenen JML-Verträge erfüllt sind.
- Fügt je mind. eine (sinnvolle) Vor- oder Nachbedingung zu folgenden Methoden hinzu:
 - `size()`
 - `isEmpty()`
 - `add()`
 - `contains()`
 - `getElements()`
- OpenJML bekommt ihr auf openjml.org
- Zum automatischen Prüfen der Verträge:

```
./openjml -esc Set.java
```

Übungsaufgabe 2 — Gefilterte Summe (18SS A7)

```
xs: List<Integer>
foreach x of xs:
  threads.add(new Thread () -> {
    if x < 10 then x else 0
  })
foreach t of threads:
  result += t.join()
```

- Wie lässt sich ein „Speedup“ von 0.005 gegenüber einer Lösung ohne Threads erklären, wenn $|xs| \approx 10^7$ (3P.)?
- Ersetzen Sie [die entsprechenden Zeilen] durch eine Implementierung mit möglichst hohem Speedup (7,5P.).
- Geben Sie eine Implementierung mittels parallelen Streams an (2,5P.).
- Zusätzlich: DbC-Fragen für 5P.

Ende

- Nächste Woche: Letztes Tut!
- Thema: Compiler
- Wieder in Präsenz