Design and Programming by Contract

Anne Haxthausen

ah@imm.dtu.dk

Informatics and Mathematical Modelling

Technical University of Denmark

Oversigt

Brug af assertions ved design og programmering.

- Det generelle begreb “Design by contract”.
- “Design by contract” for Java.

Design by Contract

Design by Contract

- er en teknik til at specificere software
- bruger prædikater (assertions) til dette :
  - Hoare style pre-betingelser og post-betingelser for metoder/operationer
  - klasse-invarianter

Prædikater

- Et *prædikat (assertion)* er en formel, hvis sandhedsværdi afhænger af tilstanden (variablernes værdier).

Eks.: Prædikatet "x delelig med 3", hvor $x \in \mathbb{N}$, er sand for de tilstande hvor $x = 0, x = 3, x = 6, \ldots$.

Prædikaterne kan udtrykkes:

- i naturligt sprog,
- i sædvanlig matematisk notation,
- som boolske udtryk i et programmeringssprog, eller
- sætninger i et specielt dedikeret sprog (f.eks. OCL for UML, og JML for Java)

Design by Contract: pre- og postbetingelser

For hver operation/metode skrives en *kontrakt* bestående af to prædikater: en *pre-betingelse* og en *post-betingelse*

Eksempel: Square root operation `sqrt` med input x og output y.

- Pre-betingelse: $x \geq 0$.
- Post-betingelse: $y * y = x$.

Ideelt set skrives kontrakten *før* implementeringen.

Kan efter implementeringen evt. dokumenteres i programmet, checkes eller bevises overholdt.

Design by Contract: pre- og postbetingelser

Pre-betingelsen

- angiver, hvornår metoden må benyttes, dvs. noget, der skal gælde lige *før* metode-kaldet
- kan referere til input og tilstanden lige før metode-kaldet
- skal overholdes af "klient-programmer"

Post-betingelsen

- angiver noget, der skal gælde lige *efter* metode-kaldet
- kan referere til input, output og tilstanden lige før og efter metode-kaldet
- skal garanteres af "leverandør-programmet" (metode-implementeringen), *hvis* klienten holder sin del

Bemærk: Post-betingelsen beskriver hvad metode-kaldet resulterer i, ikke hvordan det algoritmisk opnås.

Eksempel

```
public class Person {
  private int weight;
  ...
  public void addKgs(int kgs){
    ...
  }
}
```

Kontrakt for `addKgs`:

- Pre-betingelse: `kgs >= 0`.
- Post-betingelse: `(weight == \old(weight) + kgs)`.

`\old(weight)` angiver `weight` i "pre-tilstanden".

Dokumentation af pre- og postbetingelser

Muligheder for dokumentation af pre- og postbetingelser i leverandør-programmer:

- Som *kommentarer*.
- Som *assertions* for sprog (f.eks. Java), der tilbyder det.
- Nogle programmeringssprog (f.eks. Eiffel og Spec#) har *"native support"*, dvs. specielt dedikerede sprogkonstruktioner til pre- og postbetingelser.
- Nogle programmeringssprog (f.eks. Java) har *"third-party support"* f.eks. i form af syntaksudvidelser, der preprocesseres til assertions i sproget selv (f.eks. JML for Java).

De tre sidste former giver mulighed for ved kørsel af et klient-program at *checke* om kontrakten overholdes af de to parter.

Hoare-logik (se separat foil-set) kan bruges til at *bevise*, at leverandør-programmet *altid* overholder sin del af kontrakten.

Java Assert Statements

Syntaks: `assert` <*boolean-expression*>`;`
Eksempel: `assert (x >= 0);`
Effekt:

- Ved normal programudførelse med `java`: Ingen.
- Ved programudførelse med `java -ea`:
  boolean-expression vil blive evalueret.
  Hvis resultatet er `true`, fortsætter programmet sin udførelse.
  Hvis resultatet er `false`, vil en `AssertionError` exception blive kastet.

Brug:

- F.eks. til at checke invarianter, pre- og post-betingelser under udviklingsfasen.
- Assertions kan evt. undertrykkes af effektivitetsgrunde i det færdige produkt.

Assertion checking af pre- og postbetingelser

Kontrakt for `addKgs`:

- Pre-betingelse: `kgs >= 0`.
- Post-betingelse: `(weight == \old(weight) + kgs)`.

```
public class Person {
  private int weight;
  ...

  public void addKgs(int kgs){
    int oldweight = weight;
    assert (kgs >= 0); //pre condition check
    weight = weight + kgs;
    assert (weight == oldweight + kgs); //post cond check
  }
}
```


Design by Contract: Klasse-invarianter

En *klasse-invariant* er et prædikat på tilstanden for instanser af klassen. Denne invariant skal være opfyldt før og efter enhvert metode-kald på instanser af klassen.

Eksempel:

```
public class Person {
  private int weight;
  ...
}
```

Klasse-invariant: `weight >= 0`

Dokumentation: samme muligheder som for pre- og postbetingelser.

Progr. by contract — beskyt klasseinvariant

- En leverandør skal tilbyde metoder, der bevarer invarianten.
  Dvs. hvis pre-betingelsen og invarianten er sande før et metode-kald, så skal invarianten også være sand efter metode-kaldet.
- En klient må ikke kunne bryde invarianten.

Eksempel:

```
public class Person {
  private int weight;  //invariant weigth >= 0
  ...
  public void addKgs(int kgs){
    weight = weight + kgs;
  }
}
```

Hvordan kan vi checke, at `addKgs` bevarer invarianten `weigth >= 0`?

Assertion checking af klasseinvariant

```
public class Person {
  private int weight; //invariant weigth >= 0
  ...

  public void addKgs(int kgs){
    assert (kgs >= 0); //pre condition check
    assert (weight >= 0); //invariant check
    weight = weight + kgs;
    assert (weight >= 0); //invariant check
  }
}
```


Java Modeling Language (JML)

JML

- is a specification language for Java programs
- follows the design by contract paradigm
  using Hoare style pre- and postconditions and invariants
- the specifications are added as Java annotation comments to the final Java program
  which hence can be compiled with any Java compiler

JML verification tools:

- `jmlc`: an assertion checking compiler which converts JML annotations into runtime assertions
- ...

The released tools only work with the Java 1.4 subset, but a new release for Java 1.6 is on its way.

Java development using JML

- In the design: make assertions as JML annotations:
  - pre conditions and post conditions for methods
  - state invariants for classes
- Implement the classes.
- Compile them with `jmlc`.
- Run test cases. If an assertion is broaken, it results in a runtime error.
- Checks can be turned off for production use of the software.

Invariants, pre and post conditions in JML

```
public class Person {

  private /*@ spec_public @*/ int weight;
  //@ invariant weight >= 0;

  //@ requires kgs >= 0;
  //@ ensures weight == \old(weight) + kgs;
  public void addKgs(int kgs){
     weight = weight + kgs;
  }

}
```

Example of use of quantifiers in JML

```
/*@ requires a != null
  @          &&  (\forall int i; 0 < i && i < a.length;
  @                  a[i-1] <= a[i]);
@*/
int binarySearch(int[] a, int x) {...}
```

Exercise: formulate a post condition

JML Syntax (I)

JML specifications are added to Java code as annotations in comments:

- `//@` <JML specification>

<JML specifications>:

- `requires` <JML assertion> % pre condition
- `ensures` <JML assertion> % post condition
- `invariant` <JML assertion>
- ...

JML Syntax (II)

<JML assertion> = <Java expression>, but:

- must not have side effects (not using `=` neither directly nor indirectly, only call `pure` methods)
- can use extensions to Java:

Syntax	meaning
`\result`	result of method call
a `==>` b	a implies b
a `<==` b	a implied by b
a `<==>` b	a if and only if b
`\old`(E)	value of E in pre-state
`(\forall` T x`;` P`;` Q`)`	$\forall x : T \mid P \implies Q$
`(\exists` T x`;` P`;` Q`)`	$\exists x : T \mid P \wedge Q$
...	...

Design by Contract – Summary

Begreberne *klasse invariant*, *pre-betingelse* og *post-betingelse.*

- er prædikater
- "klient-programmer" må kun kalde en metode, når metodens *pre-betingelser* er opfyldt
- "leverandør-klasser" skal for hver af sine metoder garantere, at klasse-invarianter og post-betingelser er opfyldt efter hvert metode-kald, *hvis* pre-betingelser og invarianter var opfyldt lige før metodekaldet.

For Java:

- Kan *dokumenteres* i programmer som kommentarer (evt. i JML syntaks).
- Kan *checkes* med JML tools, hvis de skrives i JML syntaks.
- Kan checkes med `java -ea`, hvis de skrives i assertions.