Design and Programming by Contract

Michael R. Hansen & Anne Haxthausen

mrh@imm.dtu.dk

Informatics and Mathematical Modelling

Technical University of Denmark

Oversigt

Brug af assertions ved design og programmering.

- Design by contract
- Java Modelling Language (JML)

Design by Contract Bertrand Meyer 1986

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.

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

Design by Contract: pre- og postbetingelser

For en operation/metode beskrives *kontrakten* ved to prædikater:

- en *pre-betingelse*, der angiver forudsætningen for at benytte operationen, og
- en *post-betingelse*, der karakteriserer resultatet.

Eksempel: Square root, med input $x \in \mathbb{R}$ og output $y \in \mathbb{R}$.

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

Bemærk: Post-betingelsen beskriver hvad ikke hvordan.

Design by Contract: pre- og postbetingelser

- Pre-betingelser og post-betingelser for operationer bør dokumenteres, f.eks. som kommentarer i koden (for Java: JML annotations).
- En pre-betingelse er den del af kontrakten som "klient-programmer" skal overholde.
- En post-betingelse er den del af kontrakten som "leverandør-programmet" garanterer, *hvis* klienten holder sin del.
- Den egenskabsbaserede prædikat-form støtter et højt abstraktionsniveau (beskriver hvad ikke hvordan).

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 enhver operation på instanser af klassen.

Eksempler:

- En mængde kan repræsenteres som en liste *uden gentagelser*. Operationer, som *indsæt* og *union*, skal implementeres så denne invariant overholdes.
- Elementer i resultatliste skal være *sorteret mht. tiderne* med hurtigste tid først. Operationer på resultatlister skal respektere denne invariant.

Progr. by contract — defensiv programmering

En leverandør *skal* implementere en operation så post-betingelsen er opfyldt, når pre-betingelsen er opfyldt.

- Leverandøren kan ignorere brud på pre-betingelsen,
- men kan også ved *defensiv programmering* tage højde for brud.

I Java kan dette med fordel gøres ved brug af `assert`-sætningen.

Eks.: operation der giver 1'te element i ikke-tomt array A:

```
public static int first(int[] A){
   assert(A.length > 0);       // pre-betingelse
   return A[0];}
```

- Brug assertions under udviklingsfasen.
- Assertions kan evt. undertrykkes af effektivitetsgrunde i det færdige produkt.

Progr. by contract — beskyt klasseinvariant

- En leverandør skal tilbyde operationer der bevarer invarianten.
- En klient må ikke kunne bryde invarianten.

Eksempel: Stak med konkret repræsentation

- Array A[1..N] og et (synligt) index top, til første toppen af stakken

Operationer:

- push(x) : x tilføres stakken
- pop : topelementet returneres og fjernes fra (ikke tom) stakken
- empty : den tomme stak
- tom? : test på om stakken er tom

Klasseinvariant: top er antal elementer i stakken.

Problem: Datastrukturen bryder sammen ved misbrug af top.

Løsning: Indkapsling, f.eks. ved brug af abstrakte datatyper

Konkrete og abstrakte datatyper

En *datatype* er en mængde af værdier og en samling operationer på disse. F.eks. den konkrete stak.

En *abstrakt datatype* er en mængde af værdier og en samling operationer på disse. Værdiernes repræsentation er ikke synlig for klienter. (I Java: gør felter `private`.)

Ofte beskrives en abstrakt datatype ved en *signatur* (interface) og operationernes egenskaber.

Eksempel: Stak med elementer af type A:

```
empty: () -> Stak
push: A * Stak -> Stak
pop : Stak -> A
isEmpty: Stak -> Boolean
```

og et eksempel på sådan en egenskab kunne være:

```
pop(push(x,s)) = x
```


Abstrakte datatyper og programdesign

- Datastrukturer skal indkapsles i klasserne så invarianter ikke brydes.
- Vælg datatyper hensigtsmæssigt, så brugen af ekstra invarianter begrænses så meget som muligt. Letter arbejder for både klient og leverandør.
- Vælg type så højt oppe i typehierarkiet som muligt. Dette letter arbejdet ved ændring i valg af konkrete datatyper.

Abstrakte datatyper understøtter opdeling af systemet i afgrænsede velforståede komponenter, og et et redskab til at opnå et modulært programdesign.

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
- ...

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.

Example of invariant in JML

```
public class Person {

  private /*@ spec_public @*/ int weight;

  //@ invariant weight >= 0;
  ...
}
```


Example of pre and post conditions in JML

```
public class Person {

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

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

  //@ ensures \result == weight;
  public /*@ pure @*/int getWeight(){
     return weight;
  }
}
```


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) {...}
```

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`)`	$\exists x : T \mid P \wedge Q$
...	...

JML Syntax (III)

New modifiers in JML:

- `spec_public`: to indicate that it field may be referred to in JML assertions
- `pure`: to indicate that method does not change the state
- `assignable` x: to indicate that a method is allowed to assign to the field x.