Testen van Java code met JML

Engelbert Hubbers
Martijn Oostdijk
Erik Poll

University of Nijmegen

Overzicht

- **De specificatietaal JML voor Java**

  Wat voorbeelden van JML specificaties

- **Runtime assertion checking en testen met JML**

- **Andere JML tools**

  voor statische analyse, programmaverificatie, . . .

De specificatietaal JML

JML (Java Modeling Language)

Formele specificatietaal voor Java, waarmee je

- **het gedrag van Java classes kunt beschrijven**
- **ontwerp/implementatie-beslissingen kunt documenteren**

door middel van asserties, voor bijv.

- **precondities**
- **postcondities**
- **invarianten**

(zoals ‘Design-By-Contract’ in Eiffel, maar expressiever)

JML begonnen door Gary Leavens, maar uitgegroeid tot breder initiatief.

Doel: JML te begrijpen voor iedere Java programmeur.

JML (Java Modeling Language)

Om JML begrijpelijk & bruikbaar te maken:

- **JML gebruikt gewone Java syntax,**
  uitgebreid met een paar keywords

- **JML specs staan gewoon in de Java source file,**
  als speciaal soort commentaar, na `//@` , of tussen `/*@ ... @*/`

JML voorbeeld: invarianten

```
private int balance;
final static int MAX_BALANCE;

/*@ invariant 0 <= balance &&
              balance < MAX_BALANCE;
  @*/
```

Invariant moet gelden ná de constructor-aanroep, en zowel vóór als ná elke methode-aanroep.

JML voorbeeld: precondities

```
//@ requires amount >= 0;
public int debit(int amount) {
   ...
}
```

Preconditie is een verplichting voor klanten van deze methode.

JML voorbeeld: postconditities

```
/*@ requires ...;
    ensures balance == \old(balance)-amount;
  @*/
public int debit(int amount) {
  ...
}
```

Postconditie is garantie aan klanten van deze methode.

Een pre-/postconditie paar wordt wel een contract genoemd.

JML voorbeeld: exceptions

```
/*@ requires ...;
    ensures  ...;
    signals (DebitException) amount > balance;
  @*/
public int debit(int amount) {
  ...
}
```

‘exceptional’ postconditie specifeert wanneer een bepaalde exceptie gegooid kan worden.

JML voorbeeld: assert

```
private Object[] obs;
...
for (k=0; k<obs.length & obs[k]!=null; k++);
/*@ assert
      (\forall int i; 0 <= i && i < k;
                      obs[i] != null);
  @*/
...
```

`assert` specificeert eigenschap op een willekeurig punt in code.

Zit inmiddels in Java 1.4, maar JML is expressiever (door bijv. `\old` of `\forall`).

Voordelen van JML

Betere documentatie van code,

- **door het expliciet maken van aannames en ontwerpbeslissingen (met name als invarianten)**
- **in formele ipv. natuurlijke taal**

Een formele taal is preciezer, en maakt tool support mogelijk, bijvoorbeeld bij testen . . .

Testen met JML

Runtime assertion checking

Speciale compiler [Gary Leavens & Yoonsik Cheon, Iowa]
vertaalt JML annotaties naar runtime checks:

- **tijdens executie worden alle asserties getest (dwz alle pre- en postcondities, invarianten, asserts, . . .)**
- **schendingen van asserties resulteren in een runtime error**

Testen met JML

Runtime assertion checking is handig tijdens het testen.
De JML specs dienen dan als test oracle.

- **JML specs geven eigenschappen die we kunnen testen.**
- **De testcode hiervoor krijgen we gratis.**
- **Deze testcode doet veel tests, op veel tijdstippen tijdens de executie, en geeft dus goede feedback.**

Voordelen van JML bij testen (1)

Betere feedback bij testen, over de locatie en de reden van fouten.

Bijv. na 10 seconden
"**`Invariant violated in line 20000`**"
ipv. na 100 seconden
"**`NullPointerException in line 30000`**"

Informatie over oorzaak ipv. symptomen van het probleem.

Voordelen van JML bij testen (2)

Ingewikkelde testcode wordt automatisch gegenereerd.

Bijv. voor

```
/*@ requires amount >= 0;
    ensures  balance == \old(balance)-amount;
    signals (DebitException) amount > balance;
  @*/
public int debit(int amount) { ...
```

wordt de `\old(balance)` gelogd, de exception afgevangen en opnieuw gegooid, en eventuele invarianten worden getest.

jmlunit

JML kan gecombineerd worden met JUnit, een populair regressie-testing framework voor Java (`www.junit.org`)

Enige invoer nodig voor testen: voor elke type een lijst met waarden/objecten van dat type

Bijv., gegeven

```
Purse[] purses = {new Purse(), new Purse(10)};
int[]   ints =   {-9, -1, 0, 1, 2, 10,
                  100, Integer.MAX_VALUE};
```

wordt de `debit` methode aangeroepen op alle `purses` met alle mogelijke `int`'s, en worden alle schendingen van asserties (muv. de eerste preconditie!) gerapporteerd.

Andere JML tools

Statische analyse

Ambitieuzer dan run-time testen:

probeer (mogelijke) schendingen van asserties te detecteren bij compile-time.

Voordelen

- **kan éérder in de ontwikkel-cyclus;**
- **kan meer zekerheid bieden: analyse is niet afhankelijk van test-sets, maar beschouwt willekeurige inputs.**

Maar:

- **in hoeverre is dit mogelijk ?**

JML tools voor statische analyse

- **static checking met ESC/Java** [Compaq, Leino et al]

  automatische verificatie van eenvoudige JML annotaties
  Bijv, verificatie dat er nooit een IndexOutOfBounds- of NullPointerException kan optreden.

  Niet sound, niet volledig, maar vindt snel veel bugs.

- **programmaverificatie met LOOP tool** [Nijmegen]

  interactieve verificatie van willekeurig ingewikkelde JML annotaties mbv. stellingbewijzer

  Wel sound en volledig, maar vereist veel gebruikerinteractie

Feasible voor JavaCard smartcard programma's.

Andere JML tools

Niet voor checken, maar voor lezen/schrijven van JML annotaties:

- **jmldoc** [David Cok, Kodak]

  Integreert javadoc en JML

- **Daikon** [Michael Ernst, MIT]

  Observeert executies en detecteert (waarschijnlijke) invarianten, pre- en postcondities, en produceert JML annotaties hiervoor.

  Handig voor legacy code, of om test coverage te bepalen.

Related Work

(Commerciële) tools voor runtime assertion checking:

- **i-Contract**
- **Jcontract en Jtest**

Gebruikte assertietalen minder expressief dan JML.

Tools voor statische analyse:

- **PREfix en PREfast**

voor C(++) code, gebruikt bij MicroSoft.
Focus op memory management, buffer overflows, etc.

Conclusies

- **JML is een eenvoudig te leren, formele specificatietaal voor Java**
- **Bij gebruik van JML ga je in vroeg stadium al nadenken over testbaarheid van code**
- **Het gebruik van JML runtime assertion checking bij testen: beter testen voor minder werk**
- **Het kan nuttiger zijn te investeren in asserties dan in test-code:**
  - **dit kan in eerder stadium,**
  - **asserties zijn beter onderhoudbaar, en**
  - **ook nuttig voor andere doeleinden (mn. documentatie)**

Voor meer informatie, en alle tools: `www.jmlspecs.org`