Java Modeling Language

Marta Maciejewska

22 lutego 2012

JML

Java Modeling Language (JML)

- programowanie kontraktowe (ang. *design by contract*, DbC)
- warunki wstępne, warunki końcowe i niezmienniki
- komentarze w kodzie źródłowym
- składnia przypominająca wyrażenia logiczne w Javie

Przykład

```
1  public class TickTockClock {
2    //@ public model JMLDataGroup _time_state;
3
4    //@ protected invariant 0 <= hour && hour <= 23;
5    protected int hour; //@ in _time_state;
6    //@ protected invariant 0 <= minute && minute <= 59;
7    protected int minute; //@ in _time_state;
8    //@ protected invariant 0 <= second && second <= 59;
9    protected int second; //@ in _time_state;
10
11   //@ ensures getHour() == 12 && getMinute() == 0 && getSecond() == 0;
12   public /*@ pure @*/ TickTockClock() {
13       hour = 12; minute = 0; second = 0;
14   }
15
16   //@ requires true;
17   //@ ensures 0 <= \result && \result <= 23;
18   public /*@ pure @*/ int getHour() { return hour; }
19
20   //@ ensures 0 <= \result && \result <= 59;
21   public /*@ pure @*/ int getMinute() { return minute; }
```

Słowa kluczowe

- requires
- ensures

Warunki wstępne i końcowe

1. metoda:
   - może zakładać, że zachodzi warunek wstępny
   - musi zapewnić warunek końcowy
2. korzystający z metody:
   - musi zapewnić warunek wstępny
   - może zakładać, że zachodzi warunek końcowy

Prosty przykład

```
/* @ requires 0 <= hour && hour <= 23;
   @ requires 0 <= minute && minute <= 59;
   @ */
public void setAlarmTime(int hour, int minute) {
        alarmHour = hour;
        alarmMinute = minute;
}

//@ ensures 0 <= \result && \result <= 59;
public int getMinute() {
        return minute;
}
```

Niezmienniki

- zapewniane przez każdy konstruktor
- zachowywane przez wszystkie metody

Przykład

```
public class TickTockClock {

  //@ invariant 0 <= hour && hour <= 23;
  protected int hour;

  //@ invariant 0 <= minute && minute <= 59;
  protected int minute;

  //@ invariant 0 <= second && second <= 59;
  protected int second;

  ...
}
```

Przykład 2

```
public void tick() {
   second++;
   /* tutaj niezmiennik
      nie musi zachodzi
   */
   canvas.paint();
   (...)
}
```

```
public void paint() {

   (...)

   timer.getSecond();

   (...)
}
```

Niezmienniki

- zapewniane przez każdy konstruktor
- zachowywane przez wszystkie metody

Niezmienniki

- zapewniane przez każdy konstruktor
- zachowywane przez wszystkie metody
- zachowane przy każdym wywołaniu metody (niepomocniczej)

Symbole

- && || ! ==
- ==> <== <==>
- \result \old( )

Kwantyfikacja

\forall \exists \min \max...

```
(\forall Student s;
        juniors.contains(s);
                s.getAdvisor() != null)
```

Wywoływanie metod

- tylko metody bez efektów ubocznych

Wywoływanie metod

- tylko metody bez efektów ubocznych

```
/*@ ensures getHour() == 12
      && getMinute() == 0
      && getSecond() == 0; @*/
public TickTockClock() {
  hour = 12;
  minute = 0;
  second = 0;
}
```

assignable

- zmieniane mogą być **tylko** wymienione pola
- domyślnie *assignable* *everything*

assignable

- zmieniane mogą być **tylko** wymienione pola
- domyślnie *assignable* *everything*

```
//@ assignable hour, minute, second;
public void tick() {
  second++;
  if (second == 60){
    second = 0; minute++;
  }
  if (minute == 60) {
    minute = 0; hour++;
  }
  if (hour == 24) {
    hour = 0;
  }
}
```

pure

- *pure* = *assignable* *nothing*
- znaczenie:
  - metoda - żadnych efektów ubocznych
  - kostruktor - przypisania tylko na pola tworzonego obiektu

pure

- *pure* = *assignable* *nothing*
- znaczenie:
  - metoda - żadnych efektów ubocznych
  - kostruktor - przypisania tylko na pola tworzonego obiektu

```
public class Clock {

  public /*@ pure @*/ Clock() {
    hour = 12; minute = 0; second = 0;
  }

  public /*@ pure @*/ int getHour() {
    return hour;
  }
  //...
}
```

non-null

```
public class Directory {
  private File[] files;
  //@ invariant files != null;

  //@ requires name != null;
  void createSubdir(String name){...}

  //@ ensures \result != null;
  Directory getParent(){...}
```

non-null

```
public class Directory {
  private /*@ non null @*/ File[] files;

  void createSubdir(/*@ non null @*/ String name)

  Directory /*@ non null @*/ getParent(){...}
```

also

```
/*@  requires   getSecond() < 59;
  @  assignable hour, minute, second;
  @  assignable _time_state;
  @  ensures    getSecond() == \old(getSecond() + 1) &&
  @             getMinute() == \old(getMinute()) &&
  @             getHour() == \old(getHour());
  @ also
  @  requires   getSecond() == 59;
  @  assignable _time_state;
  @  ensures    getSecond() == 0;
  @  ensures    (* hours and minutes are updated appropriately *);
  @*/
public void tick() {
  second++;
  if (second == 60) { second = 0; minute++; }
  if (minute == 60) { minute = 0; hour++; }
  if (hour == 24) { hour = 0; }
}
```

Wyjątki

- normal_behavior
- exceptional_behavior

Wyjątki

- normal_behavior
- exceptional_behavior

- signals
- signals_only

Wyjątki

```
/*@ public normal_behavior
@ requires 0 <= hour && hour <= 23 &&
@          0 <= minute && minute <= 59;
@ assignable _time_state;
@ ensures getHour() == hour &&
@         getMinute() == minute && getSecond() == 0;
@ also
@ public exceptional_behavior
@ requires !(0 <= hour && hour <= 23 &&
@            0 <= minute && minute <= 59);
@ assignable \nothing;
@ signals (IllegalArgumentException e) true;
@ signals_only IllegalArgumentException;
@*/
public void setTime(int hour, int minute) {
  if (!(0 <= hour & hour <= 23 & 0 <= minute & minute <= 59)) {
    throw new IllegalArgumentException();
  }
  this.hour = hour;
  this.minute = minute;
  this.second = 0;
}
```

JML a dziedziczenie

- musi być spełniona specyfikacja z nadklas:
  - niezmienniki
  - przedefiniowywane metody
- also

datagroup

```
//@ public model JMLDataGroup _time_state;

protected int hour; //@ in _time_state;
protected int minute; //@ in _time_state;
protected int second; //@ in _time_state;
```

datagroup

```
//@ public model JMLDataGroup _time_state;

protected int hour; //@ in _time_state;
protected int minute; //@ in _time_state;
protected int second; //@ in _time_state;

//@ assignable _time_state;
public void tick() {
  second++;
  if (second == 60) { second = 0; minute++; }
  if (minute == 60) { minute = 0; hour++; }
  if (hour == 24) { hour = 0; }
}
```

model fields

- używane tylko w specyfikacji
- reprezentacja części stanu obiektu
- może być użyte tak jak datagroup
- szczególnie użyteczne przy interfejsach

model fields

```
public class Clock {
  //@ public model long _time;
  //@ private represents _time = second + minute*60 + hour*60*60;

  private int hour; //@ in _time;
  private int second; //@ in _time;
  private int minute; //@ in _time;

  //@ assignable _time;
  //@ ensures _time == \old(_time + 1) % 24*60*60;
  public void tick() {
    second++;
    if (second == 60) { second = 0; minute++; }
    if (minute == 60) { minute = 0; hour++; }
    if (hour == 24) { hour = 0; }
  }
}
```

Narzędzia

- test w czasie wykonania
- dowód w czasie kompilacji

ESC/Java2

- ESC = Extended Static Checker

ESC/Java2

- ESC = Extended Static Checker
- może dawać ostrzeżenia dla poprawnych programów
- może nie dawać ostrzeżenia dla błędnego programu