Formal Methods for Java
Lecture 4: JML and Abstract Data Types

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

November 4, 2011
The Java Modelling Language (JML)

JML is a behavioral interface specification language (BISL) for Java

- Proposed by G. Leavens, A. Baker, C. Ruby:
  JML: A Notation for Detailed Design, 1999

- It combines ideas from two approaches:
  - Eiffel with it’s built-in language for Design by Contract (DBC)
  - Larch/C++ a BISL for C++
Semantics of Specification

/*@ requires x >= 0;
   @ ensures result <= Math.sqrt(x) && Math.sqrt(x) < result + 1;
   @*/

public static int isqrt(int x) {
    body
}

Whenever the method is called with values that satisfy the requires-formula and the method terminates normally then the ensures-formula holds. For all executions of the method,

\[(\text{Norm}, \text{heap}, lcl) \xrightarrow{\text{body}} (\text{Ret}, \text{heap}', lcl'),\]

if \(lcl(x) \geq 0\) then the formula

\[lcl'(\text{result}) \leq Math.sqrt(lcl(x)) < lcl'(\text{result}) + 1\]

holds.
What About Exceptions?

```java
/*@ requires true;
   @ ensures \result <= Math.sqrt(x) && Math.sqrt(x) < \result + 1;
   @ signals (IllegalArgumentException) x < 0;
   @ signals_only IllegalArgumentException;
   @*/
public static int isqrt(int x) {
    body
}
```

For all transitions

\[
(Norm, heap, lcl) \xrightarrow{body} (Exc(v), heap', lcl')
\]

where \(lcl\) satisfies the precondition and \(v\) is an Exception, \(v\) must be of type IllegalArgumentException. Furthermore, \(lcl\) must satisfy \(x < 0\).

The code is still allowed to throw an Error like a OutOfMemoryError or a ClassNotFoundsError.

If no signals_only clause is specified, JML assumes a sane default value: The method may throw only exceptions it declares with the throws keyword (in this case none).
Side-Effects

A method can change the heap in an unpredictable way.

The assignable clause restricts changes:

```java
/*@ requires x >= 0;
 assignable \nothing;
 ensures \result <= Math.sqrt(x) && Math.sqrt(x) < \result + 1;
 @*/
public static int isqrt(int x) {
    body
}
```

For all executions of the method,

\[
(Norm, heap, lcl) \xrightarrow{body} (Ret, heap', lcl'),
\]

if \(lcl(x) >= 0\) then the formula

\[
lcl'(<result>) <= Math.sqrt(lcl(x)) < lcl'(\result + 1)
\]

holds and \(heap = heap'\).
Lightweight vs. Heavyweight Specifications

A lightweight specification
```java
/*@ requires P;
  @ assignable X;
  @ ensures Q;
  @*/
public void foo() throws IOException;
```
is an abbreviation for the heavyweight specification
```java
/*@ public behavior
  @ requires P;
  @ diverges false;
  @ assignable X;
  @ ensures Q;
  @ signals_only IOException
  @*/
public void foo() throws IOException;
```
/*@ public normal_behavior
@ requires x >= 0;
@ assignable \nothing;
@ ensures \result <= Math.sqrt(x) && Math.sqrt(x) < \result + 1;
@ also
@ public exceptional_behavior
@ requires x < 0;
@ assignable \nothing;
@ signals (IllegalArgumentException) true;
@*/

public static int isqrt(int x) throws IllegalArgumentException {
    if (x < 0)
        throw new IllegalArgumentException();
body}
If several specification are given with also, the method must fulfill all specifications.

A specification with normal_behavior implicitly has the clause
\begin{verbatim}
 signals (java.lang.Exception) false
\end{verbatim}
so the method may not throw an exception.

A specification with exceptional_behavior implicitly has the clause
\begin{verbatim}
 ensures false
\end{verbatim}
so the method may not terminate normally.
The Roots of JML

- Ideas from Eiffel:
  - Executable pre- and post-condition (for runtime checking)
  - Uses Java syntax (with a few extensions).
  - Operator \( \texttt{old} \) to refer to the pre-state in the post-condition.

- Ideas from Larch:
  - Describe the state transformation behavior of a method
  - Model Abstract Data Types (ADT)
Subsystems request timer events and queue them.
First timer event is passed to the timer.
Priority queue maintains events in its internal data structure.
public interface PriorityQueue {
    public void enqueue(Comparable o);
    public Comparable removeFirst();
    public boolean isEmpty();
}
Adding Incomplete Specification

```java
public interface PriorityQueue {

    /*@ public normal_behavior
        @ ensures !isEmpty();
        @*/
    public void enqueue(Comparable o);

    /*@ public normal_behavior
        @ requires !isEmpty();
        @*/
    public Comparable removeFirst();

    public /*@pure@*/ boolean isEmpty();

}
```
Why is Specification Incomplete?

The specification allows undesired things.

- After `removeFirst()` new value of `isEmpty()` is undefined.
- In a correct implementation, after two `enqueue()` and one `removeFirst()` list is not empty.
  Specification does not say so.
- Problem: the `internal state` is not visible in spec.
- There is not even internal state in an interface!
Adding Model Variables

Solution: add a model variable that records the size.

```java
public interface PriorityQueue {
    //@ public instance model int size;

    //@ public invariant size >= 0;

    /*@ public normal_behavior
     * @ ensures size == \old(size) + 1;
     * @*/
    public void enqueue(Comparable o);

    /*@ public normal_behavior
     * @ requires !isEmpty();
     * @ ensures size == \old(size) - 1;
     * @*/
    public Comparable removeFirst();

    /*@ public normal_behavior
     * @ ensures \result == (size == 0);
     * @*/
    public /*@pure@*/ boolean isEmpty();
}
```
Model Variables

//@ public instance model int size;

- Model variables only exist in the specification.
- Public model variables can be accessed by other classes.
- Only specification can access model variables (read-only).
- If a model variable is accessed in code, the compiler complains.
Visibility in JML

```java
//@ public instance model int size;
...
/*@ public normal_behavior
  @ ensures \result == (size > 0);
@*/
public /*@pure@*/ boolean isEmpty();
```

Why is size public?

- The external interface must be public.
- The specification is part of the interface.
- To understand the specification one needs to know about `size`.
- Therefore, `size` is public.
public class Heap implements PriorityQueue {
    private Comparable[] elems;
    private int numElems;
    
    // @ private represents size <- numElems;
    public void enqueue(Comparable o) {
        elems[numElems++] = o;
        ...
    }

    public Comparable removeFirst() {
        ...
        return elems[--numElems];
    }

    public isEmpty() {
        return numElems == 0;
    }
}

Every model variable in a **concrete** class must be represented:

```java
//@ private represents size <- numElems;
```

The expression can also call pure functions:

```java
//@ private represents size <- computeSize();
```
How to Model Internal Structure?

- Specification is still incomplete.
- Which values are returned by \texttt{removeFirst()}?
- We need a model variable representing the \texttt{queue}.
- JML defines useful types to model complex data structures.
Example: Model for Internal Structure

```java
//@ model import org.jmlspecs.models.JMLObjectBag;
public interface PriorityQueue {
    //@ public instance model JMLObjectBag queue;

    //@ public normal_behavior
    @ ensures queue.equals(old(queue).insert(o));
    @ modifies queue;
    @*/
    public void enqueue(Comparable o);

    //@ public normal_behavior
    @ requires !isEmpty();
    @ ensures old(queue).has(result)
    @ ensures queue.equals(old(queue).remove(result))
    @ ensures (forall java.lang.Comparable o;
               queue.has(o); result.compareTo(o) <= 0);
    @ modifies queue;
    @*/
    public Comparable removeFirst();

    //@ public normal_behavior
    @ ensures result == (queue.isEmpty());
    @*/
    public/*@pure@*/ boolean isEmpty();
}
```
What is JMLObjectBag?

- `org.jmlspecs.models.JMLObjectBag` is a pure class. It has pure function and no references to non-pure classes.
- Therefore, it can be used in specifications.
- There are lot of other classes:
  