Formal Methods for Java

Lecture 6: Introduction to JML

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

May 15, 2017

Semantics of a Specification (formally)

A function satisfies the specification

requires e_1

ensures e_2

iff for all executions

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

with $(Norm, heap, lcl) \xrightarrow{e_1 \triangleright v_1} q_1$, $v_1 \neq 0$, the post-condition holds, i. e., there exists v_2, q_2, such that

$$(Norm, heap', lcl') \xrightarrow{e_2 \triangleright v_2} q_2, \text{ where } v_2 \neq 0$$

However we need a new rule for evaluating $\backslash old$:

$$\frac{(Norm, heap, lcl) \xrightarrow{e \triangleright v} q}{(Norm, heap', lcl') \xrightarrow{\backslash old(e) \triangleright v} q},$$

where $heap, lcl$ is the state of the program before $body$ was executed

Side-Effects in Specification

In JML side-effects in specifications are forbidden:
If e is an expression in a specification and

$$(Norm, heap, lcl) \xrightarrow{e \triangleright v} (flow, heap', lcl')$$

then $heap \subseteq heap'$ and $lcl = lcl'$.
Here, $heap \subseteq heap'$ indicates that the new heap may contain new (unreachable) objects.
Also $flow \neq Norm$ is possible. In that case the expression is considered to be false.
A tool should warn the user if $flow \neq Norm$ is possible.

Exceptions in Specification

There were some discussions on exceptions in JML specifications.

- `next == null || next.prev == this` is okay. It never throws a null-pointer exception.
- `next.prev == this || next == null` is not equivalent. It is not valid if `next` is null.

Specifications that can throw an exception should be avoided.

Lightweight vs. Heavyweight Specifications

A lightweigth specification

```
/*@ requires P;
  @ assignable X;
  @ ensures Q;
  @*/
public void foo() throws IOException;
```

is an abbreviation for the heavyweight specification

```
/*@ public behavior
  @   requires P;
  @   diverges false;
  @   assignable X;
  @   ensures Q;
  @   signals_only IOException
  @*/
public void foo() throws IOException;
```

With the `behavior`-keyword there are no default values for `diverges`, `signals_only`, and `assignable`.

Making Exceptions Explicit

```
/*@ 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
}
```

Making Exceptions Explicit (2)

- If several specification are given with also, the method must fulfill all specifications.
- Specifications with normal_behavior implicitly have the clause

```
signals (java.lang.Exception) false
```

  so the method must not throw an exception.
- Specifications with exceptional_behavior implicitly have the clause

```
ensures false
```

  so the method must not terminate normally.

The Java Modelling Language (JML)

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

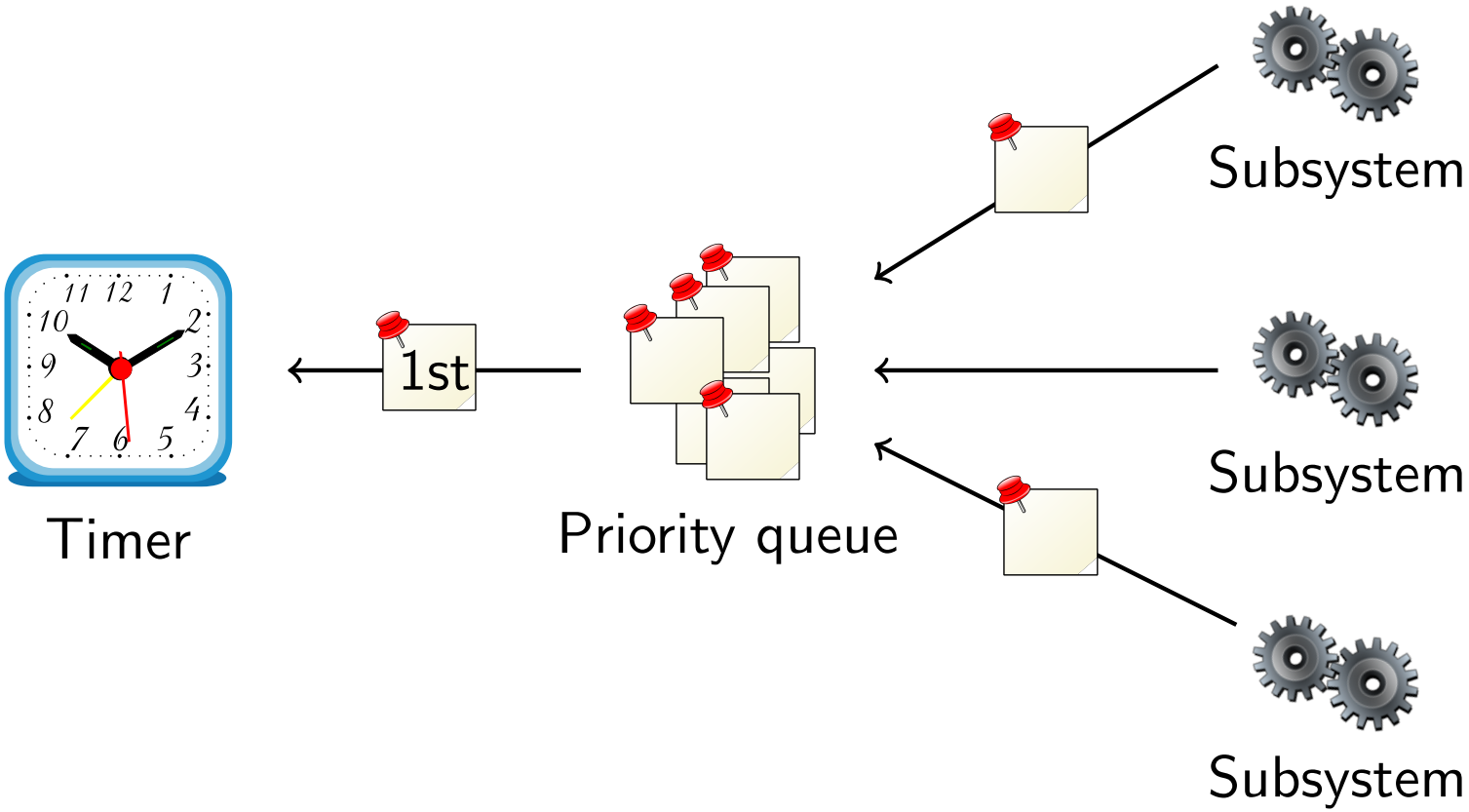
- Proposed by G. Leavens, A. Baker, C. Ruby:
  JML: A Notation for Detailed Design, 1999
- It combines ideas from two approaches:
  - Eiffel with it's built-in language for Design by Contract (DBC)
  - Larch/C++ a BISL for C++

The Roots of JML

- Ideas from Eiffel:
  - Executable pre- and post-condition (for runtime checking)
  - Uses Java syntax (with a few extensions).
  - Operator `\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)

JML and Abstract Data Types

Running Example: A priority queue



- Subsystems request timer events and queue them.
- First timer event is passed to the timer.
- Priority queue maintains events in its internal data structure.

Interface for Priority Queue

```
public interface PriorityQueue {

  public void enqueue(Comparable o);

  public Comparable removeFirst();

  public boolean isEmpty();

}
```

Adding (Incomplete) Specification

```
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();

}
```