Idea: define transition system for Java

Definition (Transition System)

A transition system ($TS$) is a structure $TS = (Q, Act, \rightarrow)$, where

- $Q$ is a set of states,
- $Act$ a set of actions,
- $\rightarrow \subseteq Q \times Act \times Q$ the transition relation.

- $Q$ reflects the current dynamic state (heap and local variables).
- $Act$ is the executed code or expressions.
- $q \xrightarrow{e \triangleright v} q'$ means that in state $q$ the expression $e$ is evaluated to $v$ and the side-effects change the state to $q'$.
- $q \xrightarrow{st} q'$ means that in state $q$ the statement $st$ is executable and changes the state to $q'$. 
Semantics of Specification

```java
/*@ 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 heap, heap', lcl, lcl' if lcl(x) ≥ 0 and (Norm, heap, lcl) body (Ret, heap', lcl'), then lcl'(\result) ≤ Math.sqrt(lcl(x)) < lcl'(\result) + 1 holds.
The JML code above states partial correctness of the Hoare triple

\[ x \geq 0 \]

body

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

It also states total correctness, as we will see later.
Is the following implementation correct?

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

No, because JML always evaluates input parameters always in the pre-state!

For all heap, heap', lcl, lcl' if lcl(x) ≥ 0
and (Norm, heap, lcl) \xrightarrow{body} (Ret, heap', lcl'),
then lcl'(\result) ≤ Math.sqrt(lcl(x)) < lcl'(\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
}
```

The `signals_only` specification denotes that for all transitions

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

where \(lcl\) satisfies the precondition and \(v\) is an Exception, \(v\) must be of type `IllegalArgumentException`.

The `signals` specification denotes that in that case \(lcl\) must satisfy \(x < 0\).

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

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,

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

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

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

holds and \(heap \subseteq heap'\).
What is the meaning of a formula

A formula like $x \geq 0$ is a Boolean Java expression. It can be evaluated with the operational semantics.

$x \geq 0$ holds in state $(heap, lcl)$, iff

$$(Norm, heap, lcl) \xrightarrow{x \geq 0 \uparrow 1} (Norm, heap', lcl')$$

An assertion may not have side-effects; it may create new objects, though, i.e., $heap \subseteq heap'$ and $lcl = lcl'$.

For the ensures formula both the pre-state and the post-state are necessary to evaluate the formula.
Semantics of a Specification (formally)

A function satisfies the specification

\[ \text{requires } e_1 \]
\[ \text{ensures } e_2 \]

iff for all executions

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

with \( (\text{Norm}, \text{heap}, \text{lcl}) \xrightarrow{e_1>\nu_1} q_1, \nu_1 \neq 0 \), the post-condition holds, i.e., there exists \( \nu_2, q_2 \), such that

\[ (\text{Norm}, \text{heap}', \text{lcl}') \xrightarrow{e_2>\nu_2} q_2, \text{ where } \nu_2 \neq 0 \]

However we need a new rule for evaluating \( \text{old} \):

\[ (\text{Norm}, \text{heap}, \text{lcl}) \xrightarrow{e>\nu} q \]
\[ (\text{Norm}, \text{heap}', \text{lcl}') \xrightarrow{\text{old}(e)>\nu} q \]

where \( \text{heap}, \text{lcl} \) is the state of the program before \( \text{body} \) was executed.
In JML side-effects in specifications are forbidden:
If $e$ is an expression in a specification and

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

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

With the `behavior`-keyword there are no default values for `diverges`, `signals_only`, and `assignable`. 
/**
 * @public normal_behavior
 * @requires \textit{x} \geq 0;
 * @assignable \textit{nothing};
 * @ensures \textit{result} \leq \textit{Math.sqrt}(\textit{x}) \land \textit{Math.sqrt}(\textit{x}) < \textit{result} + 1;
 * @also
 * @public exceptional_behavior
 * @requires \textit{x} < 0;
 * @assignable \textit{nothing};
 * @signals (IllegalArgumentException) \text{true};
 */

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

Specifications with normal_behavior implicitly have the clause

\texttt{signals (java.lang.Exception) false}

so the method must not throw an exception.

Specifications with exceptional_behavior implicitly have the clause

\texttt{ensures false}

so the method must not terminate normally.