Java Modeling Language (JML)

Assertions

JML Overview

- Supports Design by Contract for Java
  - JML specifications defined in JML annotation comments
  - JML toolset that compiles and runs JML specifications
- Can be used for runtime assertion checking, invariant discovery, specification browsing, formal verification
- http://www.eecs.ucf.edu/~leavens/JML/
- http://www.eecs.ucf.edu/~leavens/JML//jmldbc.pdf -- Short paper, great introduction to JML

Types of Assertions Supported

- Class Invariants
- Pre-conditions
- Post-conditions
- Loop Invariants
- Local assertions

JML Assertions Syntax

- Informal specifications
  - Syntax: Basically a comment
  - Example: //@ **requires** (* x is non-negative *);
  - Not checked at runtime
  - Can be used before developing formal specifications or for constructs that are not supported formally by JML (e.g. input by reading from a file)
- Formal specifications
  - Syntax: Extended Java logical expressions (i.e. more expressive expressions, can use quantifiers in JML). But also some restrictions (i.e. no side effects)
  - Example: //@ **requires** x >= 0;
  - Checked at runtime
  - Note: no space between // and @

Example: Account class

```
public class Account {
  private /*@ spec_public @*/ int bal;
  //@ public invariant bal >= 0;

  /*@ requires amt >= 0;
    @ assignable bal;
    @ ensures bal == amt; @*/
  public Account(int amt) {
    bal = amt;
  }

  /*@ assignable bal;
    @ ensures bal == acc.bal; @*/
  public Account(Account acc) {
    bal = acc.balance();
  }

  /*@ requires amt > 0 && amt <= acc.balance();
    @ assignable bal, acc.bal;
    @ ensures bal == \old(bal) + amt
    @   && acc.bal == \old(acc.bal - amt); @*/
  public void transfer(int amt, Account acc) {
    acc.withdraw(amt);
    deposit(amt);
  }

  /*@ requires amt > 0 && amt <= bal;
    @ assignable bal;
    @ ensures bal == \old(bal) - amt; @*/
  public void withdraw(int amt) {
    bal -= amt;
  }

  /*@ requires amt > 0;
    @ assignable bal;
    @ ensures bal == \old(bal) + amt; @*/
  public void deposit(int amt) {
    bal += amt;
  }

  //@ ensures \result == bal;
  public /*@ pure @*/ int balance() {
    return bal;
  }

    public static void main(String[] args) {
        Account acc = new Account(100);
        acc.withdraw(200);
        System.out.println("Balance after withdrawal: " + acc.balance());
    }
}
```

Expressions Used in Assertions

- Must be side-effect free
  - Class fields or method parameters should not be modified
    - Example: Don't use =, ++, --, …
  - Only “pure” methods (i.e. methods that have no side effects on the program state) can be called in assertions.
    - Pure methods must be declared as such
    - Example: public **/*@ pure @*/** int balance()

Commonly Used Extensions

- Expressions:
  - **\old(E)** defined to be value of E in pre-state
  - **\result** defined to be result of method call

- Logical expressions: **==>, <==, <==>, <=!=>**

- Quantifiers: **\forall, \exists**
- Other: **\sum, \product, \min, \max**, **\num_of**

Information Hiding[1]

- JML uses same privacy levels for specifications as Java uses for its language constructs
- The privacy of a JML specification (assertions) is determined by the privacy of the method it specifies
- In JML, public specifications should mention only publicly-visible names
- If a public specification needs to mention non-pubilc field, the **spec_public** annotation should be used where the non-public field is declared.
  - Example: private /*@ **spec_public** @*/ int weight;

[1] JML information hiding does not seem to be fully supported by jml4c, i.e. it sometimes lets you refer to private fields even without spec_public annotation

Class Invariants

- Property that should be true in all client-visible states -- must be true at the end of each constructor's execution, and at the beginning and end of all methods.
- May access fields
- Keyword: **invariant**
- Example: `//@ public invariant bal >= 0;`

Pre-conditions

- Must be true before/when a method is called
- May access fields and method parameters
- Keyword: **requires**, **pre**
- Example:

```
/*@ requires amt >= 0;
  @ assignable bal;
  @ ensures bal == amt; @*/
 public Account(int amt) {
   bal = amt;
 }
```

Post-conditions

- Must be true when a method returns/throws an exception
- May access fields and method parameters
  - Often use \old and \result
- Behavior
  - Normal: Return terminates method call
  - Exceptional: Exception thrown terminates method call

Post-conditions: Normal Behavior

- Applies when a method reaches “normal post-state”, i.e. it returns normally,without throwing any exceptions.
- Keyword: **ensures, post**
- Example:

```
/*@ ensures bal == \old(bal) - amt; @*/
 public void withdraw(int amt) {
   bal -= amt;
 }
```

Post-conditions: Exceptional Behavior

- Applies when a method reaches an “exceptional post-state”, i.e. it throws an exception.
- Keyword:
  - *signals_only* (specifies what exceptions a method can throw)
  - *signals* (allows to specify other information, e.g. postconditions that need to be true if an exception of a given type is thrown)

Post-conditions: Exceptional Behavior

- Example:

```
/*@
  @ public normal_behavior
  @ requires ! isEmpty();
  @ ensures elementsInQueue.has(\result);
  @ also
  @ public exceptional_behavior
  @ requires isEmpty();
  @ signals (Exception e) e instanceof NoSuchElementException;
  @*/
/*@ pure @*/ Object peek() throws NoSuchElementException;
```

If an exception of type Exception is thrown, JML checks the following expression, namely that the exception must be an instance of NoSuchElementException.

Loop Invariants

- Must be true at every iteration of a loop
- May access fields and local variables within loop statement
- Keyword: **loop_invariant**
- **Example:**

```
 int Factorial (int n) {
          int f = 1; int i = 1;
/*@ loop_invariant
  @    i<=n && f==(\product int j; 1<=j && j<=i;  j);
  @*/
    while (i < n) { i = i + 1; f = f * i; }
    return f;
}
```

Local Assertions

- May access fields and local variables within a statement
- Keyword: **assert**
- Example:

```
double posInput = 9.0;
//@ assert posInput >= 0.0;
Math.sqrt(posInput);
```

More Advanced JML Features: Quantifiers

- **\forall, \exists**
- Example:

```
//@ ensures (\forall Student s;
juniors.contains(s) ==> s.getAdvisor() != null)
```

- Note: Quantifiers can declare and modify local variables
  - `//@ ensures (\exists int j; 0 <= j && j <= \result; f(j));`

Logical Expressions Tips

- The JML specifications will generate Java code. So be careful about the order of evaluation.

Example:

```
private /*@ spec_public @*/ Collection c_;

// This evaluates successfully
//@ assert (c_ != null) && (! c_.isEmpty());

// This may throw a NullPointerException
//@ assert (! c_.isEmpty()) && (c_ != null);
```

Assignable Clause Tips

- You can use the **assignable** clause to indicate what fields a method can modify. Other fields, not specified in an assignable clause, cannot be modified by a method.
- Note: not all JML compilers check that methods modify only fields in their assignable clause.

Overall JML Assertion Tips

- The JML assertions should be as general as possible.

Example 1:

```
private /*@ spec_public @*/ String s_;

// This is general. The implementation may be either
// "s_ = s;" or "s_ = new String(s);"
//@ requires s != null;
//@ ensures s_.equals(s);
public void setS(String s) { ... }

// This is more specific. The implementation must be
// "s_ = s;"
//@ requires s != null;
//@ ensures s_ == s;
public void setS(String s) { ... }
```

Overall JML Assertion Tips (cont.)

Example 2:

```
public class MyInt {
  private /*@ spec_public @*/ int value_;

 // This is general.     //@ ensures \result != null && (! \result.length() == 0);
 public String toString() { ... }

 // This is more specific.
 //@ ensures \result != null && \result.equals("" + value_);
 public String toString() { ... }
}
```

JML Toolset

- **jml4c**:   A compiler for JML source files (like javac, but also translates the JML specifications into executable code)
- **jmlrt**: A runtime JML library used by the Java Virtual Machine when executing code compiled by jml4c

Recommended JML Toolset Installation

- Download and installation instructions are available here: http://www.cs.utep.edu/cheon/download/jml4c/download.php
- This is a command line tool and it requires a java installation (Java 1.5 or Java 1.6 recommended)