What is JML?

- A Design by Contract (DBC) tool for Java
- Specifies agreement between a class and client code
  - Obligations/Rights of the class and the client

Contracts in Software

```
/*@ requires x >= 0.0;
  @ ensures JMLDouble.approximatelyEqualTo(x,
  @                 \result * \result, eps);
  @*/
public static double sqrt(double x) { ... }
```

	Obligations	Rights
Client	Passes non-negative number	Gets square root approximation
Implementor	Computes and returns square root	Assumes argument is non-negative

JML Syntax: comments

- Specifications written in *annotation comments*
- Single-line:

```
//@ assert x >= 0;
```

- Multi-line:

```
/*@ ensures kgs >= 0
  @     && weight == kgs + 10;
  @*/
```

- Comments:

```
//@ requires x > 0;  (* x is positive *)
```

JML Syntax: Assertions

- Assertions are Java expressions that evaluate to a boolean value, but:
  - Cannot have side effects
    - No use of =, ++, --, etc., and
    - Can only call *pure* methods.

```
public /*@ pure @*/ int getWeight();
```

JML – Types of Assertions

- Class Invariants
- Loop Invariants
- Method Pre and Postconditions
  - Normal and exceptional postconditions

Class Invariants

- *invariant* keyword used
- Checked at the start and end of each method call to the class

```
public class Person{
  private String name;
  //@ public invariant !name.equals(“”);
  …
```

Loop Invariants

- *assert* keyword used
- Checked at each iteration at the designated point in a loop

```
for(i=0;i<n;i++){
  //@ assert !list.isEmpty();
  list.remove(i);
}
```

Method Pre and Postconditions

- *requires* keyword used for preconditions
  - Checked immediately before method invocation
- *ensures* keyword used for normal postconditions
  - Checked immediately following method invocation

```
/*@ requires n != null && !n.equals(“”);
  @ ensures name.equals(n)
  @*/
public setName(String n);
```

Exceptional Postconditions

- *signals* keyword used
- Checked when method throws an exception
  - multiple exceptional postconditions possible

```
/*@ signals (IllegalArgumentException e)
  @          e.getMessage() != null
  @          && !(x > 0.0);
  @*/
public static double sqrt(double x) throws
   IllegalArgumentException
```

JML: Additional Syntax

- JML has some extensions to Java syntax

Syntax	Meaning
\result	result of method call
a ==> b	a implies b
a <== b	b implies a
a <==> b	a iff b
a <=!=> b	!(a <==> b)
\old(E)	value of E in pre-state

```
/*@ ensures kgs >= 0
  @    && \result == \old(weight + kgs);
  @*/
public int addWeight(int weight);
```

JML: Quantification

- JML also provides for quantification

```
/*@ requires a != null
  @     && (\forall int i;
  @              0 < i && i < a.length;
  @              a[i-1] <= a[i];
  @*/
int binarySearch(int[] a, int x) {...}
```

JML Tools

- jmlc
  - parses annotation comments and creates Java bytecode
  - calls javac
- jmlrac
  - executes code with assertions, throws exception if assertion violated
  - calls java

JML: Exercising Assertions

- Java program with “main” method required by jmlrac
- Test cases needed to exercise assertions
    - A method that is never called in a program can’t cause an assertion violation!

For more information

- www.jmlspecs.org