

**JML**

Feb-10

# Design by Contract (DbC)

- Design by Defining a contract between a client and a supplier
- Client
  - User of code
- Supplier
  - Author of code

# Obligations and Benefits

- Pre-condition
  - *requires*
  - Obligation on client to meet pre-condition
  - Benefits supplier
- Post-condition
  - *ensures*
  - Obligation on supplier to meet post-condition
  - Benefits 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
Supplier	Computes and returns square root	Assumes argument is non-negative

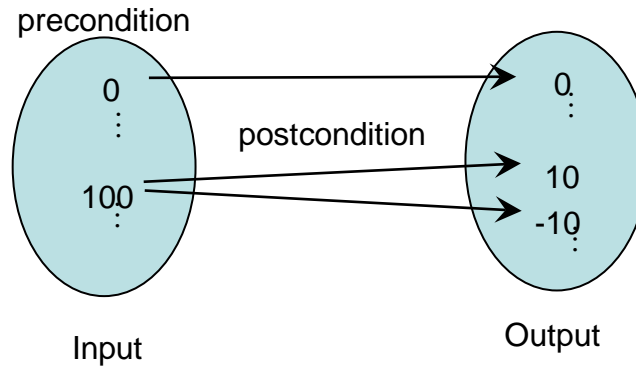
# Pre and Post Conditions

- Definition
  - A method's *precondition* says what must be true to call it.
  - A method's *normal postcondition* says what is true when it returns normally (i.e., without throwing an exception).
  - A method's *exceptional postcondition* says what is true when a method throws an exception.

```
/* @ signals (IllegalArgumentException e) x < 0;  
  @*/
```

# DbC and Testing

- Can think of a method as a relation:  
Inputs  $\leftrightarrow$  Outputs



- Pre and post conditions can limit the input/output space

# Contracts as Documentation

- For each method say:
  - What it requires (if anything), and
  - What it ensures.
- Contracts are:
  - More abstract than code,
  - Not necessarily constructive,
  - Often machine checkable, so can help with debugging, and
  - Machine checkable contracts can always be up-to-date.

# Abstraction by Specification

- A contract can be satisfied in many ways:
  - E.g., for square root:
    - Linear search
    - Binary search
    - Newton's method
    - ...
- These will have varying non-functional properties
  - Efficiency
  - Memory usage
- So, a contract abstracts all these implementations, thus you can change implementations later.

# More on Contracts

- Blame assignment
  - Who is to blame if:
    - Precondition doesn't hold?
    - Postcondition doesn't hold?
- Avoid inefficient defensive checks

```
//@ requires a != null && (* a is sorted *);  
public static int binarySearch(Thing[] a, Thing x) { ... }
```

# Contracts and Intent

- Code makes a poor contract, because can't separate:
  - What is intended (contract)
  - What is an implementation decision
    - E.g., if the square root gives an approximation good to 3 decimal places, can that be changed in the next release?
- By contrast, contracts:
  - Allow vendors to specify intent,
  - Allow vendors freedom to change details, and
  - Tell clients what they can count on.
- Question
  - What kinds of changes might vendors want to make that don't break existing contracts?

# JML

- What is it?
  - Stands for “Java Modeling Language”
    - A formal behavioral interface specification language for Java
  - Design by contract for Java
  - Uses Java 1.4 or later
  - Available from [www.jmlspecs.org](http://www.jmlspecs.org)

# Annotations

- JML specifications are contained in annotations, which are comments like:

`//@ ...`

or

`/*@ ...`

`@ ...`

`@*/`

At-signs (@) on the beginning of lines are ignored within annotations.

- Question
  - What's the advantage of using annotations?

# Attribute Keywords

- invariant
  - A Predicate over the attributes
    - `//@ invariant 0 < i;`
- non\_null
  - Attributes that are NEVER null
    - `private /*@ non_null @*/ Object a;`
- spec\_public
  - Private attributes that are used in Spec
    - `private /*@ spec_public @*/ int size;`

# DbC Keywords

- requires
  - Predicate assumed true by the method
    - `//@ requires size < maxSize;`
- ensures
  - Predicated guaranteed by the method
    - `//@ ensures a !=null;`
- also
  - Inherits specifications from its superclasses and interfaces that it implements

# Valuation Keywords

- old
  - The value of an attribute before the method call
    - `//@ ensures size > \old(size);`
- result
  - The value returned by the method
    - `//@ ensures \result = size-1;`

# Quantifier Keywords

- exists
  - Predicate is true of at least one value
    - `//@ \exists i; 0<i && i<50; i=10;`
- forall
  - Predicate is true for all values
    - `//@ \forall i; 0<i && i<10; i*i<100;`

# Exception Keywords

- signals
  - If the method generates an Exception, this predicate will hold
    - `//@ signals (Exception e)`  
`//@ size=\old(size);`
- normal\_behaviour
  - The method never generates an exception
    - `//@ normal_behaviour`  
`//@ (*contract goes here*)`

# Informal Description

- An informal description looks like:

(\* some text describing a property \*)

- It is treated as a boolean value by JML, and
- Allows
  - Escape from formality, and
  - Organize English as contracts.

```
public class IMath {  
    /*@ requires (* x is positive *);  
    @ ensures \result >= 0 &&  
    @ (* \result is an int approximation to square root of x *)  
    @*/  
    public static int isqrt(int x) { ... }  
}
```

# Formal Specifications

- Formal assertions are written as Java expressions, but:
  - Cannot have side effects
    - No use of =, ++, --, etc., and
    - Can only call *pure* methods.
  - Can use some extensions to Java:

Syntax	Meaning
$\backslash\text{result}$	result of method call
$a \implies b$	$a$ implies $b$
$a \impliedby b$	$b$ implies $a$
$a \iff b$	$a$ iff $b$
$a \not\iff b$	$\neg(a \iff b)$
$\backslash\text{old}(E)$	value of $E$ in pre-state

# Example

```
// File: Person.refines-java
/*@ refine "Person.java"

public class Person {
    private /*@ spec_public non_null @*/ String name;
    private /*@ spec_public @*/ int weight;

    /*@ public invariant !name.equals( "" ) && weight >= 0;

    /*@ also
       @ ensures Wresult != null;
       @*/
    public String toString();

    /*@ also ensures Wresult == weight;
    public int getWeight();

    /*@ also
       @ ensures kgs >= 0 && weight == Wold(kgs + weight);
       @ signals (Exception e) kgs < 0 &&
       @                                     (e instanceof IllegalArgumentException);
       @*/
    public void addKgs(int kgs);

    /*@ also
       @ requires !n.equals( "" );
       @ ensures n.equals(name) && weight == 0;
       @*/
    public Person(/*@ non_null @*/ String n);
}
```

# Invariants

- Definition
  - An *invariant* is a property that is always true of an object's state (when control is not inside the object's methods).
- Invariants allow you to define:
  - Acceptable states of an object, and
  - Consistency of an object's state.

```
//@ public invariant !name.equals( "" ) && weight >= 0;
```

# Quantifiers

- JML supports several forms of quantifiers
  - Universal and existential (`\forall` and `\exists`)
  - General quantifiers (`\sum`, `\product`, `\min`, `\max`)
  - Numeric quantifier (`\num_of`)

```
(\forall Student s; juniors.contains(s); s.getAdvisor() != null)
```

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

# Model Declarations

- What if you want to change a `spec_public` field's name?

```
private /*@ spec_public non_null @*/ String name;
```

to

```
private /*@ non_null @*/ String fullName;
```

- For specification:
  - need to keep the old name public
  - but don't want two strings.

- So, use a model field:

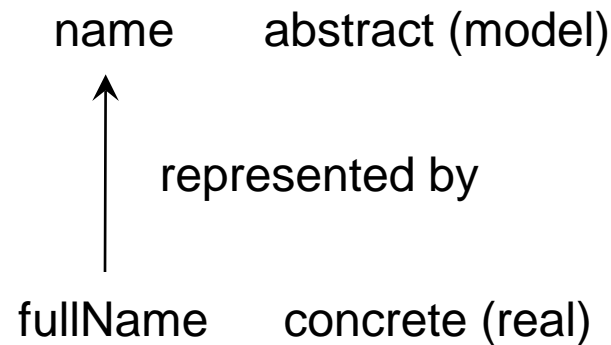
```
//@ public model non_null String name;
```

and a represents clause

```
//@ private represents name <- fullName;
```

# Model Variables

- Are specification-only variables
  - Like domain-level constructs
  - Given value only by represents clauses:



# Tools for JML

- **jmlc** – JML compiler (.java → .class (JML enhanced))
  - Must run using JDK 1.5 (JDK 1.4 – missing Iterator bug)
- **jmlrac** – JML Runtime Assertion Checker, failed assertions will cause runtime exception with offending (or occasionally the nearby) line & column
  - Requires a class w/ main method
  - Main method codifies all tests, or
- **jmlunit** – JML annotations → JUnit 3.x test case generator
  - ~95% complete: you must supply all data values for test cases
  - Basic generated Java class contains a template
- **jml-junit** – Runs JUnit against generated test cases & annotated classes
- **jmldoc** - HTML generator



# Tools

- ESC/Java2
  - Extended Static Checker – static code analysis of annotations
  - CLI
- JMLEclipse
  - Integrated compiler & other tools
  - Syntax highlighting
  - Compiler errors & RAC failures appear in Problems view
- Mobius Program Verification Environment
  - Includes several JML tools (ESC/Java2, JML2, and a variety of other static checkers and verification tools),
  - Works on Linux, Mac OS X, and Windows. (Many of the tools only work with Java 1.4 source files, but some others work with Java 1.5 and later.)



# Install/Configuration

- Install
  - DL and extract JML.5.6\_rc4.tar.gz for CLI tools, or
  - DL JML4Eclipse Plugin
- CLI Configuration
  - Define environment variable for JMLDIR
  - Add JMLDIR/bin to search path
  - Update jmlenv.bat – essential
    - Requires JUnit 3.x
    - JDK 1.4+
  - Update jmle.bat
    - Assemble your classpath as necessary
- Integrate JML tools into Ant Build process



# Install/Configuration

In JML5.6 they divide it into  
jmlenv.bat and jmle.bat

```
@ECHO OFF
rem @(#)Date: 2004/01/27 03:18:38 $
rem Configuration parameters for JML batch files
rem AUTHOR: Johan Stuyts

rem If needed, change the following configuration parameters for your system

set JUNITDIR=D:\Developer\Tools\Java\JUnit
set JMLDIR=D:\Developer\Tools\Java\JML
set JMLJAR=%JMLDIR%\bin\jml-release.jar

rem The following is needed because jml doc only works with the
rem JDK 1.4.1 or 1.4.2 tools.jar.
rem Edit the path so that it points to tools.jar in your JDK 1.4 install.
set JDK=D:\Developer\JavaKit150
set JAVA_HOME=%JDK%
set JDKTOOLS=%JDK%\lib\tools.jar
set PROJECT_CLASSPATH=<set this to what your project requires>
set JAVA_RUNTIME_JAR=%JDK%\jre\lib\rt.jar
set CLASSPATH=%JDKTOOLS%;%JMLDIR%\specs;%JUNITDIR%\junit.jar;%JMLJAR%;%PROJECT_CLASSPATH%
rem set CLASSPATH=

set PATH=%JDK%\bin;%JMLDIR%\bin
```

jmlenv.bat

# JML Compiler (jmlc)

- Basic usage

```
$ jmlc Person.java  
    produces Person.class
```

```
$ jmlc -Q *.java  
    produces *.class, quietly
```

```
$ jmlc -d ../bin Person.java  
    produces ../bin/Person.class
```

# Example

```
// File: Person.refines-java
/*@ refine "Person.java"

public class Person {
    private /*@ spec_public non_null @*/ String name;
    private /*@ spec_public @*/ int weight;

    /*@ public invariant !name.equals( "" ) && weight >= 0;

    /*@ also
       @ ensures Wresult != null;
       @*/
    public String toString();

    /*@ also ensures Wresult == weight;
    public int getWeight();

    /*@ also
       @ ensures kgs >= 0 && weight == Wold(kgs + weight);
       @ signals (Exception e) kgs < 0 &&
       @                                     (e instanceof IllegalArgumentException);
       @*/
    public void addKgs(int kgs);

    /*@ also
       @ requires !n.equals( "" );
       @ ensures n.equals(name) && weight == 0;
       @*/
    public Person(/*@ non_null @*/ String n);
}
```

# A Main Program

```
public class PersonMain {  
    public static void main(String[] args) {  
        System.out.println(new Person(null));  
        System.out.println(new Person(""));  
    }  
}
```

# Example (Formatted)

```
$ jmlc -Q Person.java
```

```
$ javac PersonMain.java
```

```
$ jmlrac PersonMain
```

```
Exception in thread "main" org.jmlspecs.jmlrac.runtime.JMLEntryPreconditionError
```

```
: by method Person.Person regarding specifications at
```

```
File "Person.refines-java", line 52, character 20 when
```

```
  'n' is null
```

```
  at org.jmlspecs.samples.jmltutorial.Person.checkPre$$init$$Person(
```

```
    Person.refines-java:1060)
```

```
  at org.jmlspecs.samples.jmltutorial.Person.<init>(Person.refines-java:51)
```

```
  at org.jmlspecs.samples.jmltutorial.PersonMain.main(PersonMain.java:27)
```

# Summary

- JML is a powerful DBC tool for Java.
- For details, refer to the JML web page at

[www.jmlspecs.org](http://www.jmlspecs.org)

# Exercise

- Formally specify the missing part, i.e., the fact that `a` is sorted in ascending order.

```
/*@ old boolean hasx = (\exists int i; i >= 0 && i < a.length; a[i] == x);  
  @ requires  
  @  
  @ ensures (hasx ==> a[\result] == x) && (!hasx ==> \result == -1);  
  @ requires_redundantly (* a is sorted in ascending order *);  
  @*/  
public static int binarySearch(/*@ non_null @*/ int[] a, int x) { ... }
```

Hint: use a nested quantification!