An introduction to formal specifications and JML

1. Why use (formal) specifications?

Yves Ledru
Université Grenoble-1
Laboratoire d'Informatique de Grenoble
Yves.Ledru@imag.fr
2013

Université Joseph Fourier GRENOBLE

Why do we need specifications?

- To make a distinction between:
  – What a program does. (specification)
  – How the program does it. (implementation)
- To provide a basis for the validation/verification (V&V) of the program:
  you need a reference to decide on the conformance of the program to this reference!
- V&V can proceed in different ways:
  – Informal reasoning
  – Testing
  – Formal proof (using proof tools)

Why do we need specifications? (2)

- To provide a concise description of the program (not always the case)
- Example : concise specification of sort(l)

Forall i,j in 0...len(Result)-1. i<j => l(i) <= l(j)
and
Result is a permutation of l

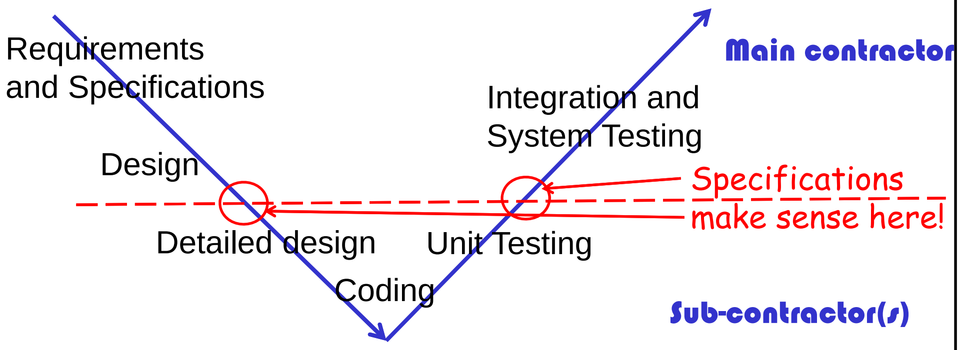
- This specification:
  - Is concise, mixing formal and informal languages
  - Applies to all variants of sort (bubble sort, quicksort,…)

Specifications and Components

- When you buy a component (black box), you only need to read the specification to understand:
  - what it does
  - Under which assumptions
- You don’t need to test it to understand what it does, …
- …, but you may test it to evaluate its quality.

Sub-contractors

- In a global world, it may the case that the V cycle is shared between a main contractor and its sub-contractor(s).



Sub-contractors and specification

- The specification is the reference document between the main and the sub contractors!
- The main contractor is responsible to provide a correct specification!
- The sub-contractor is responsible to provide code which conforms to the specification!
- When a problem arises, the specification helps to clarify the responsibilities of each stake-holder.

Implementation freedom and evolution

- The specification tells « what » but not « how »: it leaves implementation freedom!
- Implementation freedom allows
  - To quickly release a first unefficient implementation
  - To optimize implementation in release 2
  - To allow evolution of every unspecified feature
- Unveiling the code may block evolution: the code does not distinguish what is fixed and what may evolve!

Limits of natural language

- Formal specifications can be written in natural language…
- … but natural language is often ambiguous!

Ambiguity 1 : « sorted »

- *« the list of names is sorted »*
  looks like a precise specification.
- Can you use the Unix sort command?
- How do you sort uppercases and lowercases?
- E.g. « Dupont » and « du Pont »
- What about special characters?
  « é » vs « e », « ç » vs « c »
- *« the list of names is sorted in alphabetical order »*
  is more precise!
- Have a look at the Wikipedia page on « Alphabetical Order » for more examples…

Ambiguity 2: Opening hours

- Taken from a real example:
  - *The super-market is opened from 8:30 till 20:30 during summer*
  - *Otherwise it is opened every day from 9:00 to 19:30*
  - *It is opened on Sunday morning, during the whole year*

- When does it close on Sunday (incomplete and incorrect specification)?

Ambiguity 3: Airport Security

article 4.1, 4th chapter of Annex 17 of ICAO

Each Contracting State shall establish measures to prevent weapons, explosives or any dangerous devices which may be used to commit an act of unlawful interference, the carriage or bearing of which is not authorized, from being introduced, by any means whatsoever, on board an aircraft engaged in international civil aviation.

- This sentence is ambiguous ! Are all weapons not authorized? Are there weapons that are authorized?

Limits of natural language

- Ambiguities lead to diverging interpretations of the reference document!
- Difficult to establish who is right/wrong? Both stakeholders (main and subcontractor) may be right but not agree!
- Even more difficult when the document is written in a foreign language (english) for both stakeholders!

The need for a precise specification language

- Natural language is unsufficient!
- Formal languages, whose semantics is defined unambiguously by mathematical notions are a solution!
- Several languages exist. This course uses JML…

Why JML (Java Modeling Language)?

- Based on the syntax of Java and close to code (Evolution and not Revolution)
- Because the RAC (Run-time Assertion Checking) can be used in a testing approach
- Because a lot of work has been dedicated to its definition, and the identification of useful constructs.

Similar languages

- JML is based on « Design By Contract »TM
- Other languages use similar concepts:
  - OCL (Object Constraint Language), part of UML2
  - Spec# and Code Contracts (.NET)
  - Modern Jass for Java
  - Overture/VDM

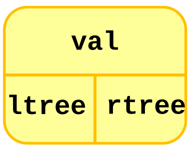
Variants of JML

- This course uses JML5.6, an older version of JML which only supports the concepts of Java 1.4
- OpenJML is a more recent version, which supports more recent constructs of Java.
- There exist other variants, with varying levels of tool support, documentation, and maintenance!

A first example : implementing a set as a tree

SetAsTree data structure

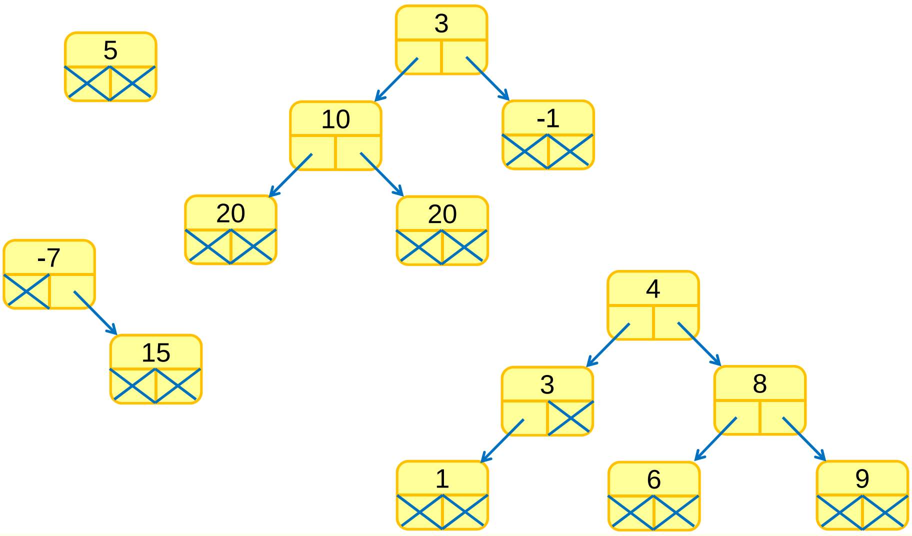
- Consider the following Java Class



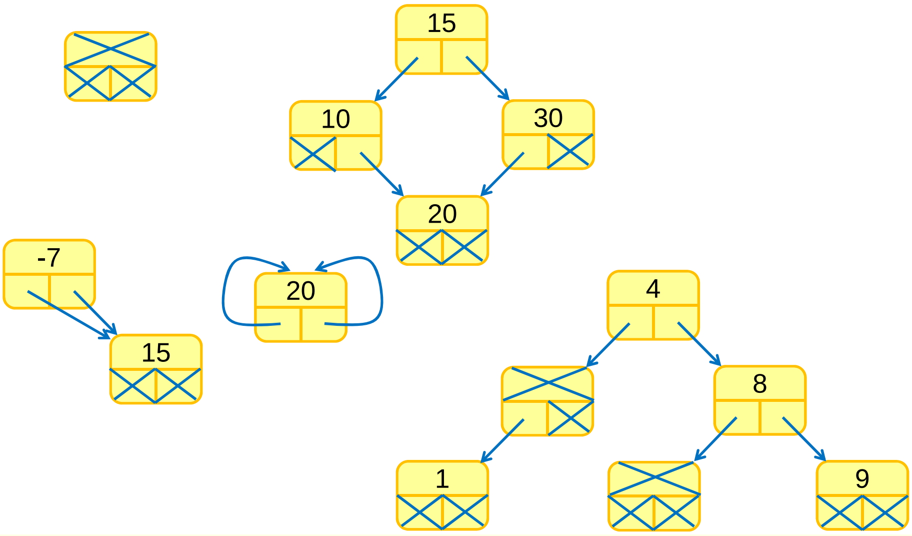

```
public class SetAsTree{
    public Integer val;
    public SetAsTree ltree;
    public SetAsTree rtree;
…
```

- Give several instantiations of objects of this class.

Here are some examples…



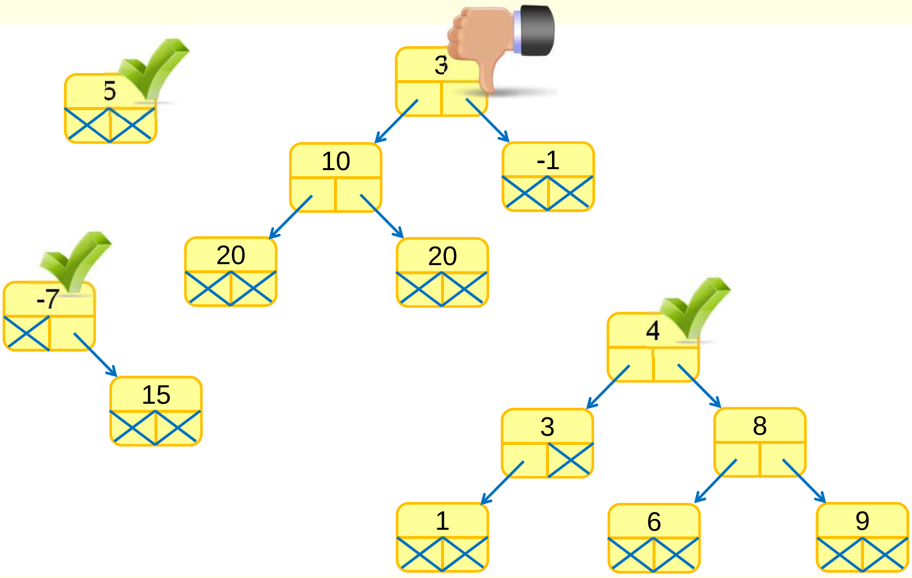
… and some other examples



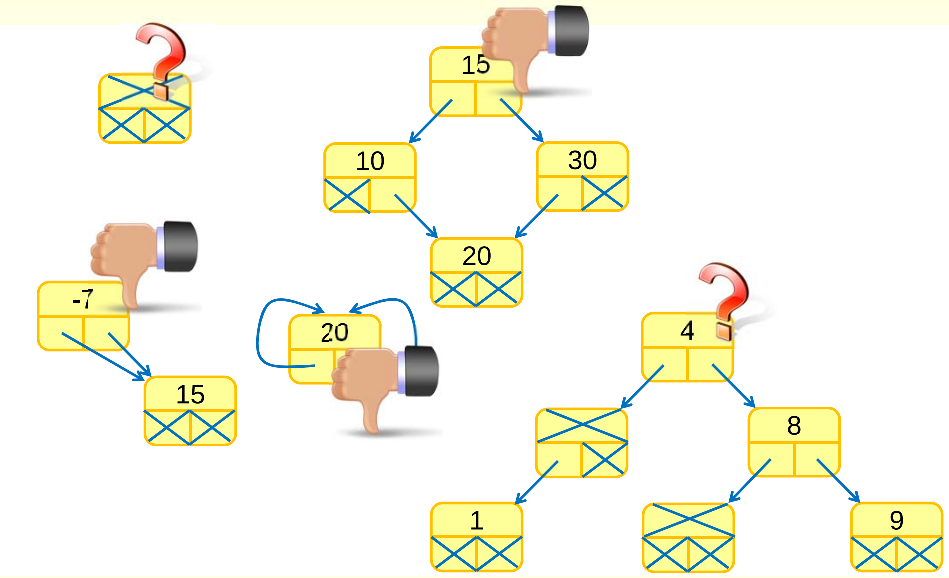
Intended use of the class…

- The class is intended to represent a set of Integers, stored in a sorted tree.
- Consequences:
  - Integer values may only appear once in the tree
  - Elements in the left tree are less than the value
  - Elements in the right tree are greater than the value

Valid and invalid examples (1)

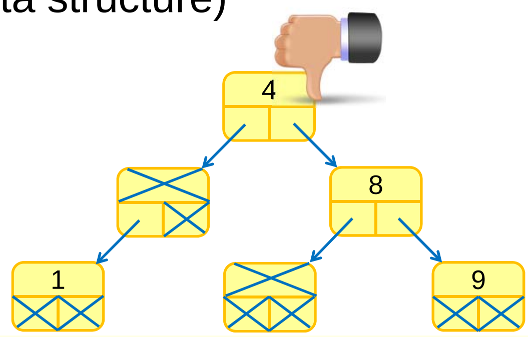


Valid and invalid examples (2)



The case of the empty set

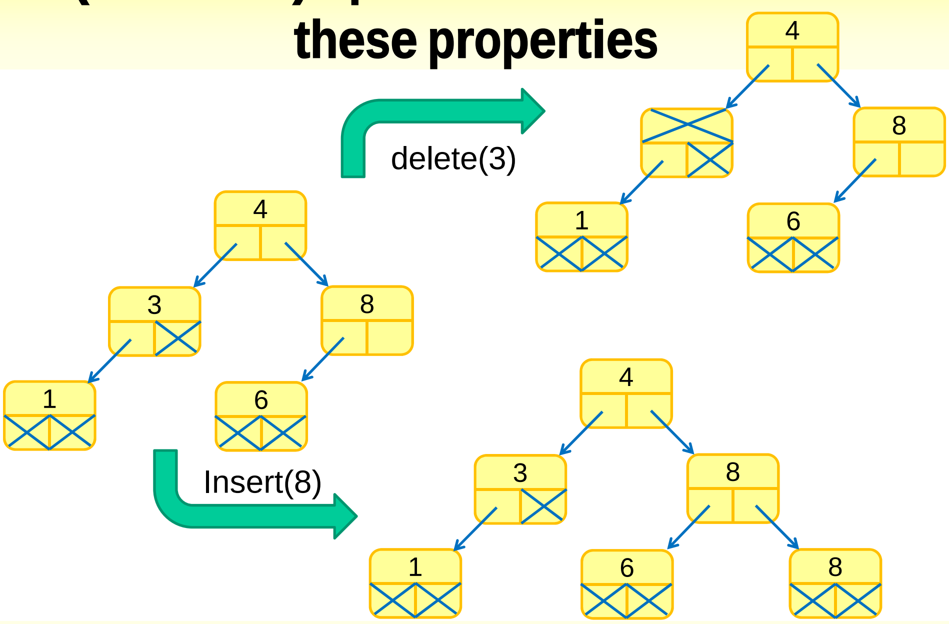
- May **`val`** take value null?
- Yes, because the set may be empty!
- But only at the root of the tree (to minimize the data structure)



The properties of SetAsTree

1. Integer values may only appear once in the tree
   (this property can be deduced from properties 2 and 3)
2. Elements in the left tree are less than **`val`**
3. Elements in the right tree are greater than **`val`**
4. **`val`** may only be null at the root of an empty set
5. There are no loops in the structure and only one way from the root to a given element
   (this property can be deduced from properties 2 and 3)

(incorrect) Operations can invalidate these properties



How can we make sure that these 5 properties will always be fulfilled?

- It should be documented somewhere (otherwise, it will be rapidly forgotten and maintenance operations might introduce defects in operations).
- It should be covered by regression tests:
  - But you need to express an appropriate « assert » for each test
- **It may be expressed in JML and checked at run-time!**

