

Design by Contract in Java

Vladimir Bacvanski and Petter Graff



www.inferdata.com

vladimir@inferdata.com

Phone: (512) 306-8225

Overview

- ◆ Goal: overview of the Design by Contract in Java
- ◆ Design by Contract and Java: A good fit?
- ◆ Benefits and problems
- ◆ Implementation approaches and systems
- ◆ Design by Contract and Java: The road towards Trusted Components

What is Design By Contract (DbC)?

- ◆ Objects in a system communicate as clients and servers
- ◆ Communication is regulated through clear responsibilities for involved parties
- ◆ Both clients and servers in communication have defined benefits and obligations
- ◆ UML defines placeholders to specify contracts and a language for specifications: OCL

Elements of Contracts

- ◆ Java with a DbC extension, for an interface *Square*:

```
interface Square {  
  
    public init( Point location, int width );  
        pre: ( width > 0 )  
        post: ( getLocation() == location )  
  
    public void resize(int times);  
        pre:  
        post: ( getWidth() = getWidth()@old * times )  
  
    ...  
  
    public Point getLocation();  
    public int getWidth();  
  
    invariant: ( getWidth() > 0 )  
}
```

Contracts and Interfaces

- ◆ If we specify contracts at the interface level, we define what are the correct implementations
- ◆ Interface actually defines a model for the component, that can be implemented in several ways
- ◆ Model serves for understanding – similar to JavaBeans properties

DbC and OO Development Process

- ◆ Use of pre and postconditions starts in domain modeling, for description of events in problem domain
- ◆ In analysis, contracts define the system operations
- ◆ In design, contracts define benefits and responsibilities for every component
- ◆ In testing, contracts define the correctness criteria for every component

The Need for DbC in Java

- ◆ The most frequently required extension of Java at Sun's Java Developer Connection "bug parade"
- ◆ Contracts enable us to enforce correct use of a component
- ◆ Contracts should be checked during development
 - We may want to outcompile them for deployment
- ◆ Java originally contained pre and postconditions as a part of the language
 - This feature was later removed because of a deadline

When Contracts Fail

- ◆ In correct program, no contracts should fail
- ◆ Checking of contracts is enforced during development and testing
- ◆ Alternatives when a contract fails:
 - Abort execution
 - Notify user
 - Throw an exception
 - Try to recover, then retry the failed method
- ◆ Preserving contract checking in delivered system enables quick problem identification

Benefits of the Design by Contract

- ◆ The semantics of methods and components known from their interfaces – without analyzing implementation
- ◆ The semantics of components is enforced in implementation
- ◆ Debugging is easier, since most problems will be caught by the contracts
- ◆ Contracts represent description of valid results for testing

Problems with Design by Contract

- ◆ Checking of contracts imposes performance penalty
 - Contracts can be removed from delivery version
 - Checking can be turned on/off for a method or a class
- ◆ Writing contracts requires time
 - The time will be saved in testing and debugging
 - Users of the component don't need to waste time reading all the implementation
 - Contracts are already known in analysis and design
- ◆ Accepting and spreading the DbC idea

Implementing Design by Contract in Java

- ◆ Ideal: Java includes pre and postconditions
- ◆ Workarounds:
 - Manual encoding
 - Conventions (if-then statements)
 - Class library
 - Preprocessor
 - Loader instrumenting bytecode
 - Intercepting VMs file operations
- ◆ Higher-level tool support:
 - CASE-Tools
 - IDEs

Implementation Issues

- ◆ Public methods need to check contracts
- ◆ Contract execution needs to be synchronized
- ◆ No contracts in private methods and `finalize()` method
- ◆ No checking in methods invoked from another method of the same class
- ◆ Checking of contracts in constructor should occur before the call to the superclass constructor

Systems and Implementations

- ◆ AssertMate
- ◆ iContract
- ◆ Jass
- ◆ jContractor
- ◆ Handshake
- ◆ AspectJ

Trusted Components and Java

- ◆ Building reusable components that we can trust
- ◆ Component should carry their own checkable description:
 - Model of the component
 - Contracts describing the behavior in terms of the model
- ◆ Contracts are essential for the reuse

Summary

- ◆ Design by contract is a key technique for reliable components
- ◆ Contracts are present in the whole OO development process:
 - Domain modeling
 - Analysis
 - Design
 - Implementation
- ◆ Several implementation techniques available: still no satisfactory solution
- ◆ Providing DbC in the language is the ideal

References

- ◆ Bartetzko, A: Jass, *semantik.Informatik.Uni-Oldenburg.DE/~jass/*
- ◆ Duncan, A., Hölzle, U.: Adding Contracts to Java with Handshake, *www.cs.ucsb.edu/oocsb/papers/handshake98.html*
- ◆ InferData Corporation: Advanced Object-Oriented Analysis and Design, Course Notes, *www.inferdata.com*
- ◆ InferData Corporation: Object-Oriented Testing, Course Notes, *www.inferdata.com*
- ◆ Karaorman, M., Hölzle, U., Bruno, J.: jContractor: A Reflective Java Library to Support, *www.cs.ucsb.edu/TRs/TRCS98-31.html*
- ◆ Kramer, R.: iContract, *www.reliable-systems.com*
- ◆ Meyer, B.: *Object-Oriented Software Construction, 2nd Edition*, Prentice-Hall 1997
- ◆ Reliable Software Technologies: AssertMate, *www.rstcorp.com*
- ◆ Xerox: *Aspect-Oriented Programming*, *www.parc.xerox.com/spl/projects/aop*