Class Invariants

Pre-conditions and post-conditions describe the properties of *individual methods.*

A *class invariant* is a *global* property of the *instances* of a *class*, which must be preserved by *all methods*.

A class invariant is an assertion in the class definition.

E.g. a stack class might have the following class invariant:

```
count >= 0   and
count <= capacity   and
stack_is_empty = (count = 0)
```

Class Invariants

An invariant for a class C must be satisfied by every instance of C at all “stable” times.

“Stable” times are those in which the instance is in an observable state

Example

A (mutable) class representing a range of real numbers:

```
public class RealRange {

        private RealNumber min, max;
        public RealRange(RealNumber min,
                         RealNumber max) {
              this.min = min; this.max = max;
        }
         public void setRange(RealNumber newMin,
                              RealNumber newMax) {
              this.min = newMin; this.max = newMax;
        }
}
```

invariant: **min <= max.**

The Invariant Rule

An assertion ***I*** is a correct invariant for a class *C* if and only if:

Every constructor of *C*, when applied to arguments satisfying its precondition in a state where the attributes have their default values, yields a state satisfying ***I***.

Every method of the class, when applied to arguments and a state satisfying both ***I*** and the method's precondition, yields a state satisfying ***I***.

DbC and Inheritance

What happens to assertions when classes are inherited?

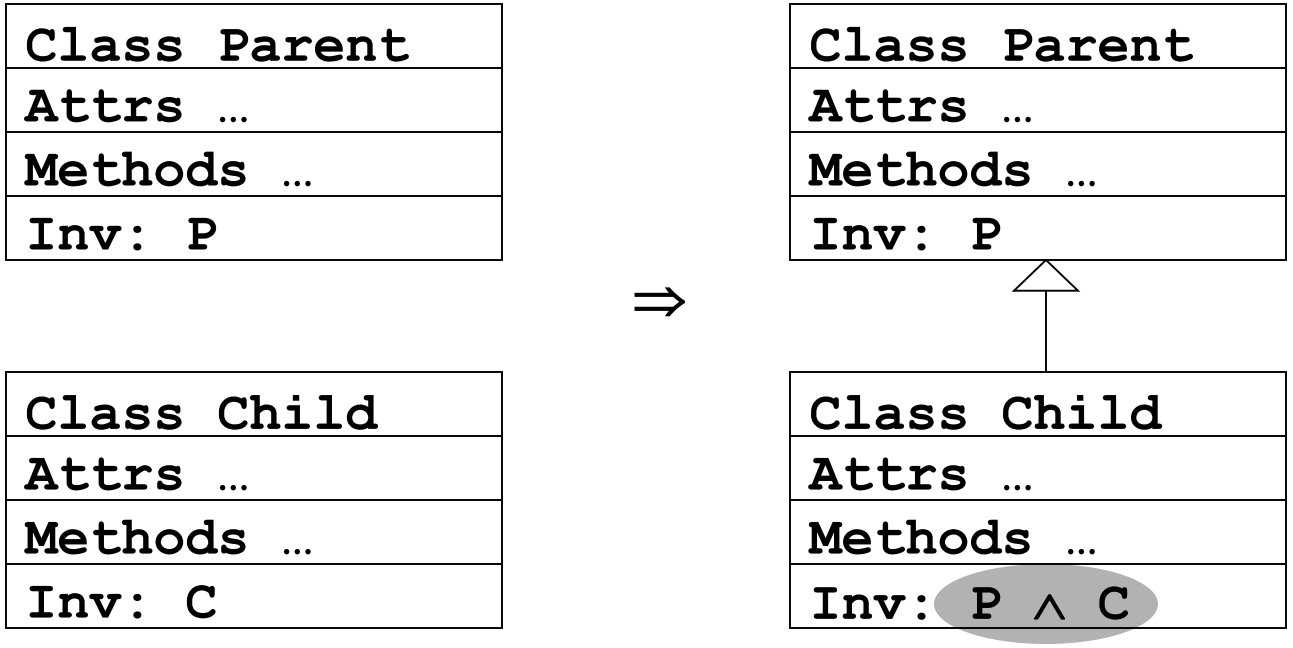
How can assertions be “preserved” in the face of redeclaration (overriding) and dynamic binding?

Actually assertions help maintain the semantics of classes and methods when they are inherited.

Invariants

The invariants of all the parents of a class apply to the class itself.

The parents' invariants are added (logically "and"ed) to the class's own invariants.



The parents' invariants need not be repeated in the class.

Pre and Postconditions

A method redeclaration may only do the following:

Pre-condition

replace the original *precondition* by one *equal* or *weaker*

The new version must accept all calls that were acceptable to the original.

It may, but does not have to, accept more cases.

e.g. replace **`pre: x<10`** by **`pre: x<=10`**

Post-condition

replace the original *postcondition* by one *equal* or *stronger*

The new version must guarantee at least as much as the original.

It may, but does not have to, guarantee more.

e.g. replace **`post: x<=10`** by **`post: x=10`**

Summary

Software reliability requires precise specifications which are honoured by both the supplier and the client.

DbC uses assertions (pre and postconditions, invariants) as a contract between supplier and client.

DbC works equally well under inheritance.

Languages with third-party support:

- C and C++: DBC for C preprocessor, GNU Nana
- C#: eXtensible C# (XC#).
- Java: iContract2, Contract4J, jContractor, Jcontract, C4J, CodePro Analytix, STclass, Jass preprocessor, OVal with AspectJ, Java Modeling Language (JML), SpringContracts for the Spring framework, or Modern Jass, Custos using AspectJ.
- JavaScript: Cerny.js or ecmaDebug.
- Common Lisp: the macro facility or the CLOS metaobject protocol.
- Scheme: the PLT Scheme extension
- Perl: the CPAN modules Class::Contract , Carp::Datum
- Python, PyDBC , Contracts for Python.
- Ruby: Ruby DBC , ruby-contract.