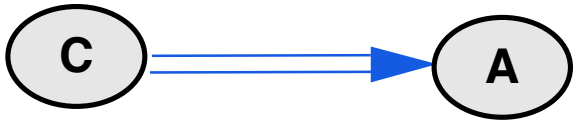
Inheritance
and
Design by Contract

Parents Invariant Rule

- The invariants of all the parents of a class apply to the class itself
  - » **The parent's invariants are AND'ed together, along with the invariants of this class**
  - » **If no invariants are given then TRUE is assumed**
- Flat and flat short forms provide a convenient way to see the whole story
  - » **Flat is used by the supplier**
  - » **Flat short is used by the client**
    - > **Does not have class history – redefine, rename, etc.**

Meaning of Design by Contract



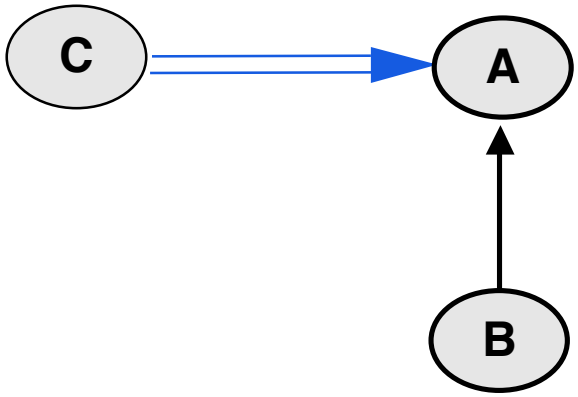

```
r is require  α
...
ensure  β
end
```

```
-- In C
a1 : A
if  a1. α then
    a1.r
    check a1. β
    ... assume a1. β  is true
end
```

Verify preconditions
if not clear they are satisfied

Verify postconditions.
Not needed with exception handling

Enter Dynamic Binding




```
r is require  α
...
ensure  β
end
```

```
r++ is require  γ
...
ensure  δ
end
```

```
-- In C
a1 : A
a1 := instance of type B
if  a1. ?pre? then
    a1.r
    check a1. ?post?
    ... assume a1. ?post? is true
end
```

What are ?pre? and ?post?

What restrictions are on γ and δ ?

How to cheat

```
-- In C
a1 : A
a1 := instance of type B
if  a1. ?pre? then
    a1.r
    check a1. ?post?
    ... assume a1. ?post?
end
```

- Two ways
  - » C expects α is sufficient but B has stronger preconditions
    - > don't accept all inputs
    - > demand more from client
    - > client is wrong
  - » C expects β is delivered but B has weaker postcondition
    - > deliver outside the range
    - > effectively deliver less

Be Honest

- Replace precondition with a weaker precondition
  - » **Expect less from the client than they are prepared to do**
    - > **require clause becomes weaker**
- Replace postcondition with a stronger postcondition
  - » **Deliver more to the client than they expect to get**
    - > **ensure clause becomes stronger**
- Willing to do the job as good as or better

Design by Contract with Dynamic Binding

- Contracts cannot be broken by redefinition
- Assertions require and ensure are inherited
  - » **Every behaviour of the redefined method satisfies the original contract**
  - » **But can do more**
    - > **Accept more input cases**
    - > **Deliver more specific outputs**

Subcontracting

- Redefinition is like subcontracting
- To validate a subcontract requires a theorem prover for the general case
- This is inefficient so we provide an approximation

$\alpha \rightarrow (\alpha$ **or** $\gamma)$

> **Weaker precondition is to accept** α **or** γ

$(\beta$ **and** $\delta) \rightarrow \beta$

> **Stronger postcondition is to accept** β **and** δ

Subcontracting – 2

- Language support
  - » When redefining do not use require and ensure
  - » Use require else γ
    - γ is or'ed with α – the inherited precondition
  - » Use ensure then δ
    - δ is and'ed with β – the inherited postcondition

Subcontracting example

Original definition

```
invert (epsilon : REAL ) is -- Invert matrix with precision epsilon
    require   epsilon >= 10^(– 6)
    ...
   ensure abs ((Current * inverse ) – Identity )  <=  epsilon
end
```

Redefinition

```
invert (epsilon : REAL ) is -- Invert matrix with precision epsilon
   require else   epsilon >= 10^(– 20)
   ...
  ensure then abs ((Current * inverse ) – Identity )  <= ( epsilon / 2 )
end
```

Assertion Redeclaration Rule

- In the redeclared version of a routine it is not permitted to use a **require** or an **ensure** clause. Instead you may:
  - **Use a clause introduced by require else to be or'ed with the original precondition**
  - **Use a clause introduced by ensure then to be and'ed with the original postcondition**
- In the absence of such a clause the original is retained
- The lazy evaluation (non-strict) form of **or else** and **and then** are used

Apparent Precondition Strengthening

- Consider the case of general containers that have no bounds on capacity

  List implementation

- Inherit from List but have a bounded capacity container

  Array implementation

- It looks like original has no restrictions when using **add** but refinement has restrictions

  > **cannot add when full**

Apparent Precondition Strengthening – 2

- Actually have the following in the unbounded container

  require not full

  > **With full defined as returning false**

- In child define

  full : BOOLEAN is Result := (count = Capacity) end

- In client have

  » **if not container.full then container.add(...) end**

- No changes **and no surprises** in the client
- Use **abstract** preconditions

Redefining a function into an attribute

- Small problem here
  - » **Precondition becomes the weaker True as the value can be accessed at any time**
  - » **But attributes do not have a postcondition**
    - > **The postcondition is added to the class invariant**
    - > **Thereby ensuring the contract still holds**

```
foo : INTEGER is
    require xyz > 0
    ...
     ensure Result = k + 1
end
```

```
foo : INTEGER
    ...
     invariant
         foo = k + 1
end
```

On Style

- Functions without arguments could be attributes
- Could have postcondition or use class invariants
  - class invariants are the preferred style