



Svigruppo

Monga

Asserzioni

Eiffel

What & How

Lezione XV: *Design by Contract* con Eiffel

Eiffel è esplicitamente progettato come linguaggio “di progetto”, non solo “di programmazione”:

“specify, design, implement and modify quality software” [Ecma standard 367]

“Programmazione in grande” con oggetti, derivati da **classi** organizzate in gerarchie di **ereditarietà** e raggruppate in **cluster**, che forniscono **feature** (**command** o **query**) ai loro **client**.



Svigruppo

Monga

Asserzioni

Eiffel

What & How

- pre/post-condizioni sulle feature (`require`, `ensure`)
- Invarianti di classe (`invariant`)
- asserzioni (`check`)
- loop invariant (`from .. invariant .. until .. variant .. loop .. end`)



Svigruppo

Monga

Asserzioni

Eiffel

What & How

- **invarianti di classe** sono condizioni che devono essere vere in ogni momento “critico”, ossia osservabile dall'esterno. In pratica e come se facessero parte di ogni pre- e post-condizione.
- è possibile avere un supporto run-time alle **violazioni**: se una condizione non vale viene sollevata un'eccezione
- L'eccezione porta il sistema nel precedente stato stabile ed è possibile
 - terminare con un fallimento
 - riprovare

Demo

```
class ROOT_TEST_STABLE_STATES
create make
feature {NONE}
  secret: BOOLEAN
feature {ANY}
  make -- root class cannot have preconditions
    -- require ok_pre("make")
    do
      print("Executing make%N")
      mycommand; secret := TRUE
      ensure ok_post("make")
    end
  mycommand
    require ok_pre("mycommand")
    do
      print("Executing mycommand%N")
      secret := FALSE; myother("1"); secret := TRUE
      -- But what happens if myother is a "client"?
      -- secret := FALSE; Current.myother("2"); secret := TRUE
      ensure ok_post("mycommand")
    end
  myother (s: STRING)
    require ok_pre("myother")
    do
      print("Executing myother " + s + "%N")
      ensure ok_post("myother")
    end
  ok_inv: BOOLEAN do print("Checking ok_inv!%N"); Result := True; end
  ok_pre (w: STRING): BOOLEAN do print("Checking ok_pre @ " + w + "%N"); Result := True; end
  ok_post (w: STRING): BOOLEAN do print("Checking ok_post @ " + w + "%N"); Result := True; end
```



Swigrosso

Monga

Asserzioni

Eiffel

What & How

```
class GCD
create make
feature gcd (x: INTEGER; y: INTEGER): INTEGER
  require
    positive_parms: x >= 0 and y >= 0
    not_zero: x /= 0 or y /= 0
  local t: INTEGER
  do
    if x = 0 or y = 0 then Result := x.max (y)
    else
      from Result := x; t := y
      invariant
        positive_result: Result > 0
        positive_t: t > 0
        gcd_inv: mathgcd(x, y) = mathgcd(Result, t)
      until Result = t
      loop
        if Result > t then Result := Result - t
        else t := t - Result
        end
      variant t.max(Result)
    end
  end
end
ensure
  positive_ris: Result > 0
  dividex: x = 0 or else x.integer_remainder(Result) = 0
  dividey: y = 0 or else y.integer_remainder(Result) = 0
  Result = x.min(y) or else across ((Result+1).to_integer |..| x.min(y)) as ic
    all (x.integer_remainder(ic.item) /= 0
    or y.integer_remainder(ic.item) /= 0) end
```



Svigruppo

Monga

Asserzioni

Eiffel

What & How

end

```
make do
  print("Ris: " + gcd(126,294).out + " %N")
  print("Ris: " + gcd(0,294).out + " %N")
end

mathgcd(x,y: INTEGER):INTEGER do
  from Result := x.min(y)
  until y.integer_remainder(Result) = 0
    and then x.integer_remainder(Result) = 0
  loop
    Result := Result - 1
  end
end
end
```




Svigruppo

Monga

Asserzioni

Eiffel

What & How

Spesso si scrivono le “stesse” cose due volte:

do

```
balance := balance - x
```

- Implementazione e specifica
- How & What

ensure

```
balance = old balance - x
```

Il client è responsabile delle precondizioni, il fornitore di postcondizioni e invarianti.