YORK
UNIVERSITÉ
UNIVERSITY

EECS3311 Software Design

Why DbC?

Some Examples

Modelling

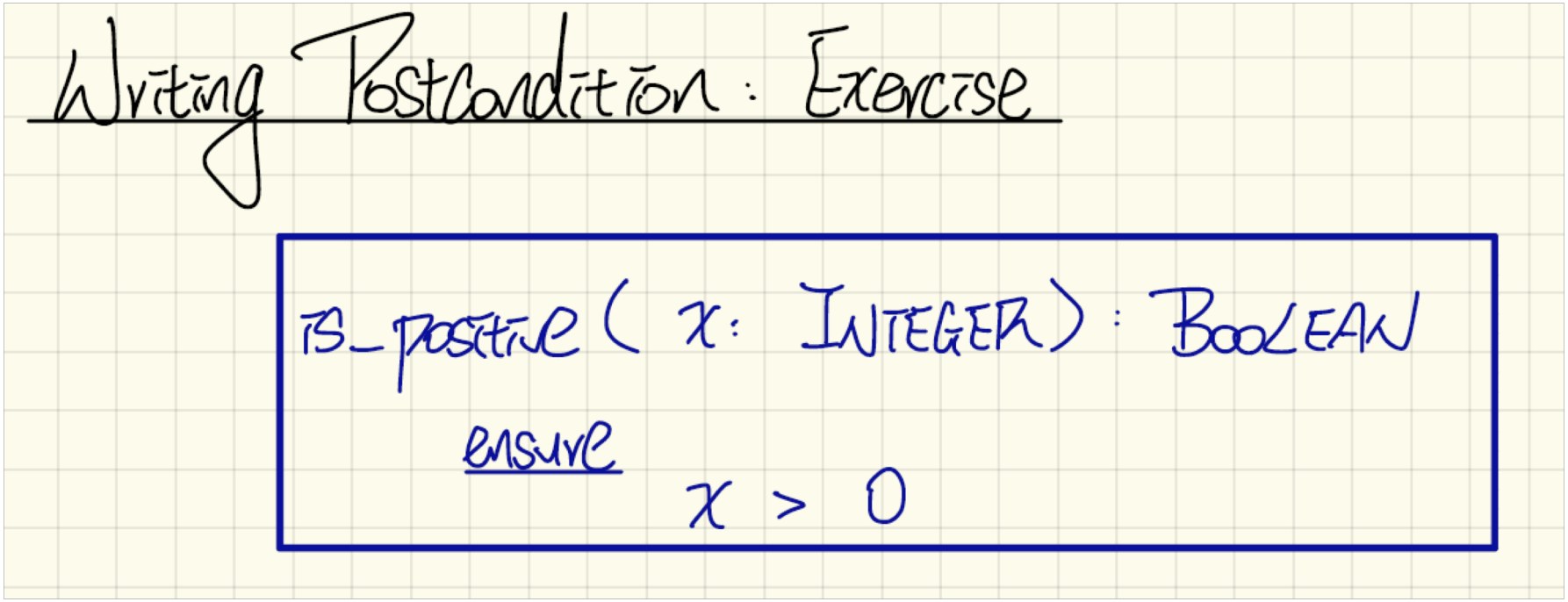
Hoare Logic

Required Reading:

OOSC2 – whole of chapter 11

See Touch of Class for introductory material on contracts

Is the contract correct?



Is the contract correct?

Writing Postcondition: Exercise

```
a: ARRAY [INTEGER]
change_at( i: INTEGER ; v: INTEGER )
        ensure
                across  a.lower |..| a.upper as  j
                 all
                    j.item = i  implies  a[j.item] = v
                   and
                    j.item /= i  implies  a[j.item] = (old a.twin)[j.item]
                 end
```

```
24 feature -- tests
25     a: ARRAY[INTEGER]
26         attribute
27             Result := <<10, 22>>
28             Result.compare_objects
29         end
30
31     change_at(i: INTEGER; v: INTEGER)
32         do
33             a[i] := v
34             a.force (33, 3)
35         ensure
36 --              post0: a.count = old a.count
37             post1: a[i] = v
38             post2:
39                 across a.lower |..| a.upper as j all
40                     j.item /= i implies a[j.item] = (old a.twin)[j.item]
41                 end
42         end
43
44     t0: BOOLEAN
45         local
46             a1, a2: like a
47         do
48             comment("t0: test array")
49             a1:=  <<10, 22>>; a1.compare_objects
50             a2:=  <<11, 22, 33>>; a2.compare_objects
51             Result := a ~ a1
52             check Result end
53             change_at (1, 11)
54             Result := a ~ a2
55             check Result end
56         end
```

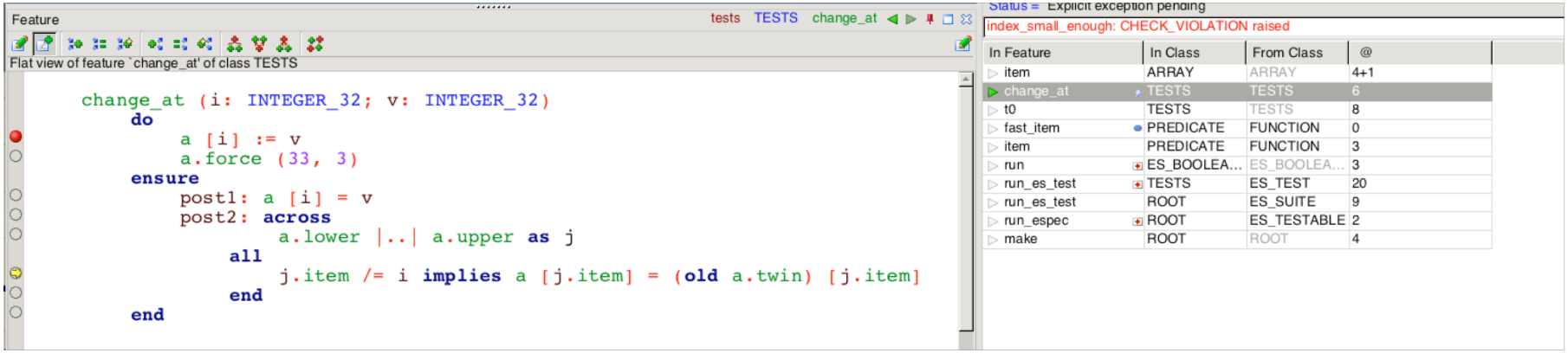
```
change_at(i: INTEGER; v: INTEGER)
    do
        a[i] := v
        a.force (33, 3) -- BAD
    ensure
        post0: a.count = old a.count
        post1: a[i] = v
        post2:
            across a.lower |..| a.upper as j all
                j.item /= i
                implies
                a[j.item] = (old a.twin)[j.item]
            end
    end
```

FAILED (1 failed & 0 passed out of 1)

Case Type	Passed	Total
Violation	0	0
Boolean	0	1
All Cases	0	1

State	Contract Violation	Test Name
Test1	TESTS	
FAILED	Postcondition violated.	t0: test array

Check violation instead of postcondition violation



FAILED (1 failed & 0 passed out of 1)

Case Type	Passed	Total
Violation	0	0
Boolean	0	1
All Cases	0	1

State	Contract Violation	Test Name
Test1	TESTS	
FAILED	Check assertion violated.	t0: test array

Is contract correct?

Writing Postcondition: Exercise

```
all_positive_values( a: ARRAY[INTEGER]): ARRAY[INTEGER]
    ensure
        across Result as x
        all
            x.item > 0
        end
```

BIRTHDAY_BOOK

model: FUN[STRING, BIRTHDAY]
-- set of tuples [name, birthday]

put (a_name: STRING; d: BIRTHDAY)
ensure
model ~ (**old** model) ↾ [a_name, d]
-- symbol ↾ used for function override

remind (d: BIRTHDAY): ARRAY[STRING]
ensure
Result.count = (model ▷ {d}).count
∀name ∈ (model ▷{d}).domain: name ∈ Result
-- symbol ▷ used for range restriction

Design of Birthday Book

BIRTHDAY_BOOK

model: FUN[NAME, BIRTHDAY]

put(n: NAME; d: BIRTHDAY)
ensure model = **old** model ↾ [n,d]
-- infix symbol for override operator: @<+

remind(d: BIRTHDAY): ARRAY[NAME]
ensure
Result.count = (model ▷ {d}).count
∀ name ∈ (model ▷ {d}).domain: name ∈ Result
-- infix symbol for range restriction: model @> (d)

count: INTEGER -- number of names

invariant:
count = model.count

model: FUN[NAME, ..]

remind: ARRAY[NAME]

BIRTHDAY

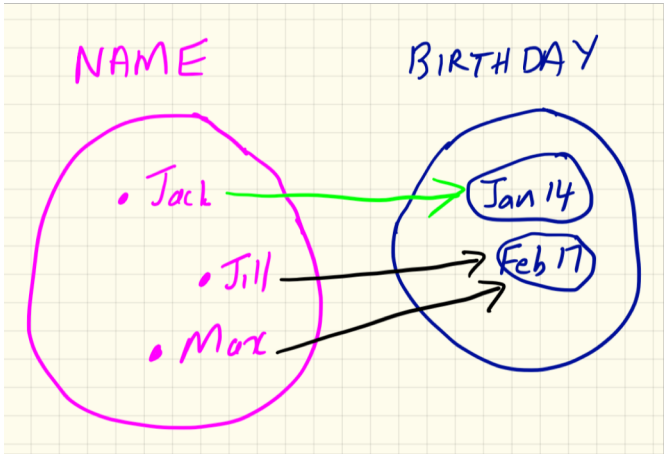
day: INTEGER
month: INTEGER

invariant
1 ≤ month ≤ 12
1 ≤ day ≤ 31

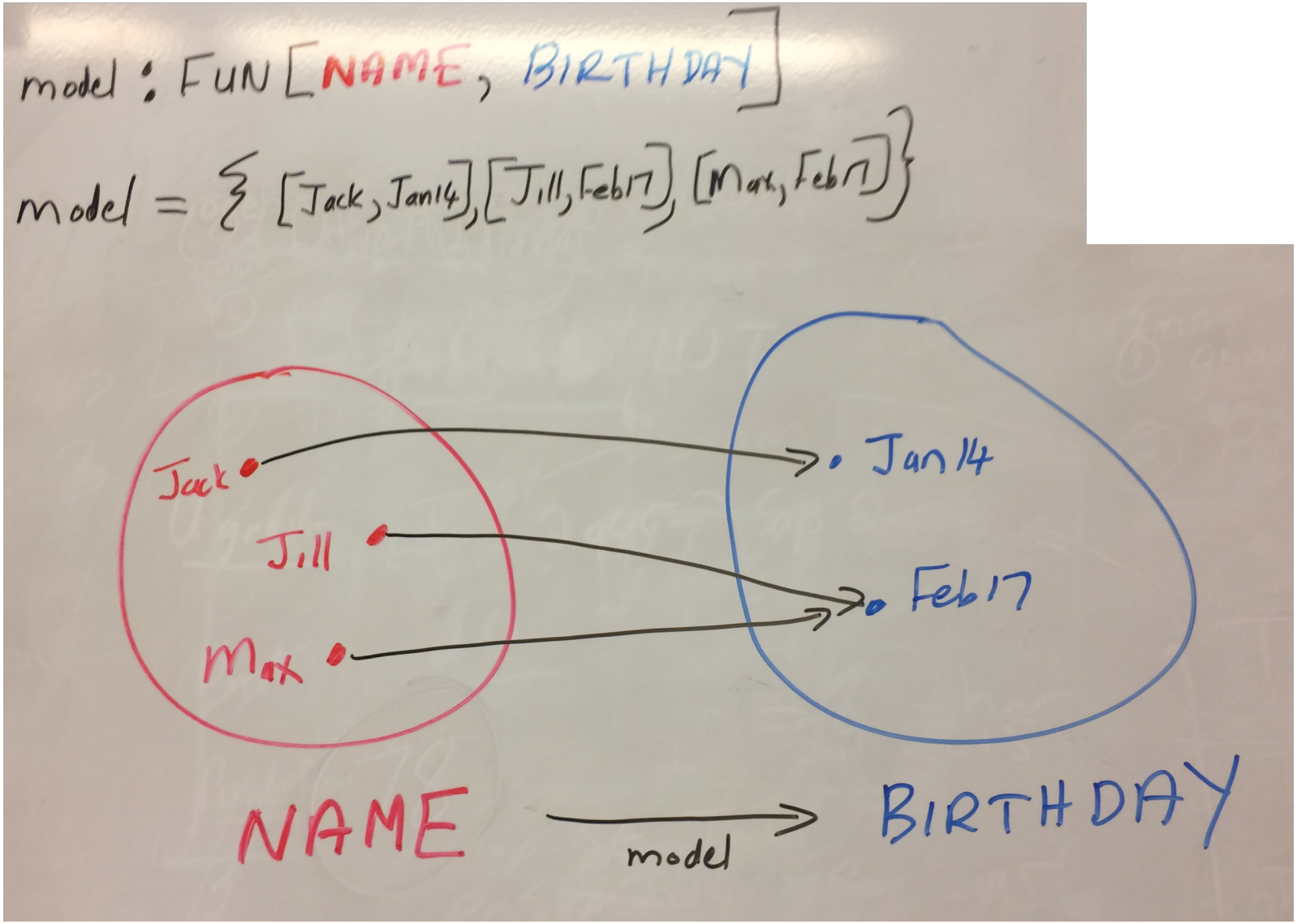
NAME

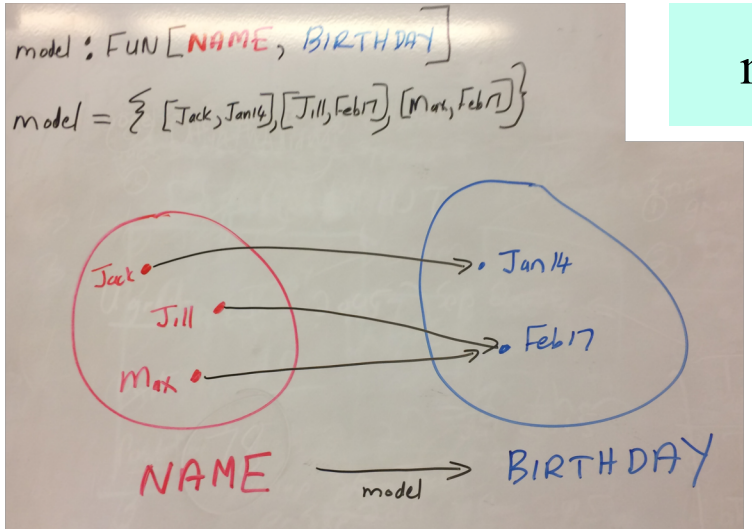
item: STRING

invariant
item[1] ∈ A..Z



model: FUN [NAME, BIRTHDAY]





model: FUN [NAME, BIRTHDAY]

- model = {[Jack, Jan14], [Jill, Feb17], [Max, Feb17]}
- model.domain = {Jack, Jill, Max}
- model.range = {Jan14, Feb17}
- Range restriction:
  model ▷ {Feb17} = {[Jill, Feb17], [Max, Feb17]}
- Override operator:
  model ↾ [Pam, Dec12] =
  {[Jack, Jan14], [Jill, Feb17], [Max, Feb17], [Pam, Dec12]}

Design of Birthday Book

BIRTHDAY_BOOK

model: FUN[NAME, BIRTHDAY]

put(n: NAME; d: BIRTHDAY)
ensure model = **old** model ↾ [n,d]
-- infix symbol for override operator: @<+

remind(d: BIRTHDAY): ARRAY[NAME]
ensure
Result.count = (model ▷ {d}).count
∀ name ∈ (model ▷ {d}).domain: name ∈ Result
-- infix symbol for range restriction: model @> (d)

count: INTEGER -- number of names

invariant:
count = model.count

model: FUN[NAME, ..]

remind: ARRAY[NAME]

BIRTHDAY

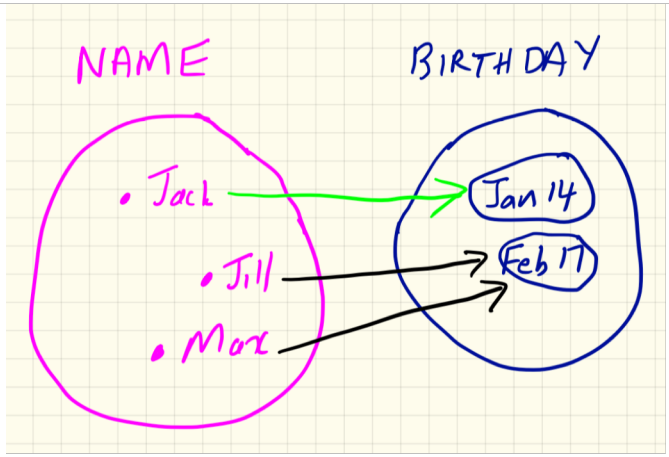
day: INTEGER
month: INTEGER

invariant
1 ≤ month ≤ 12
1 ≤ day ≤ 31

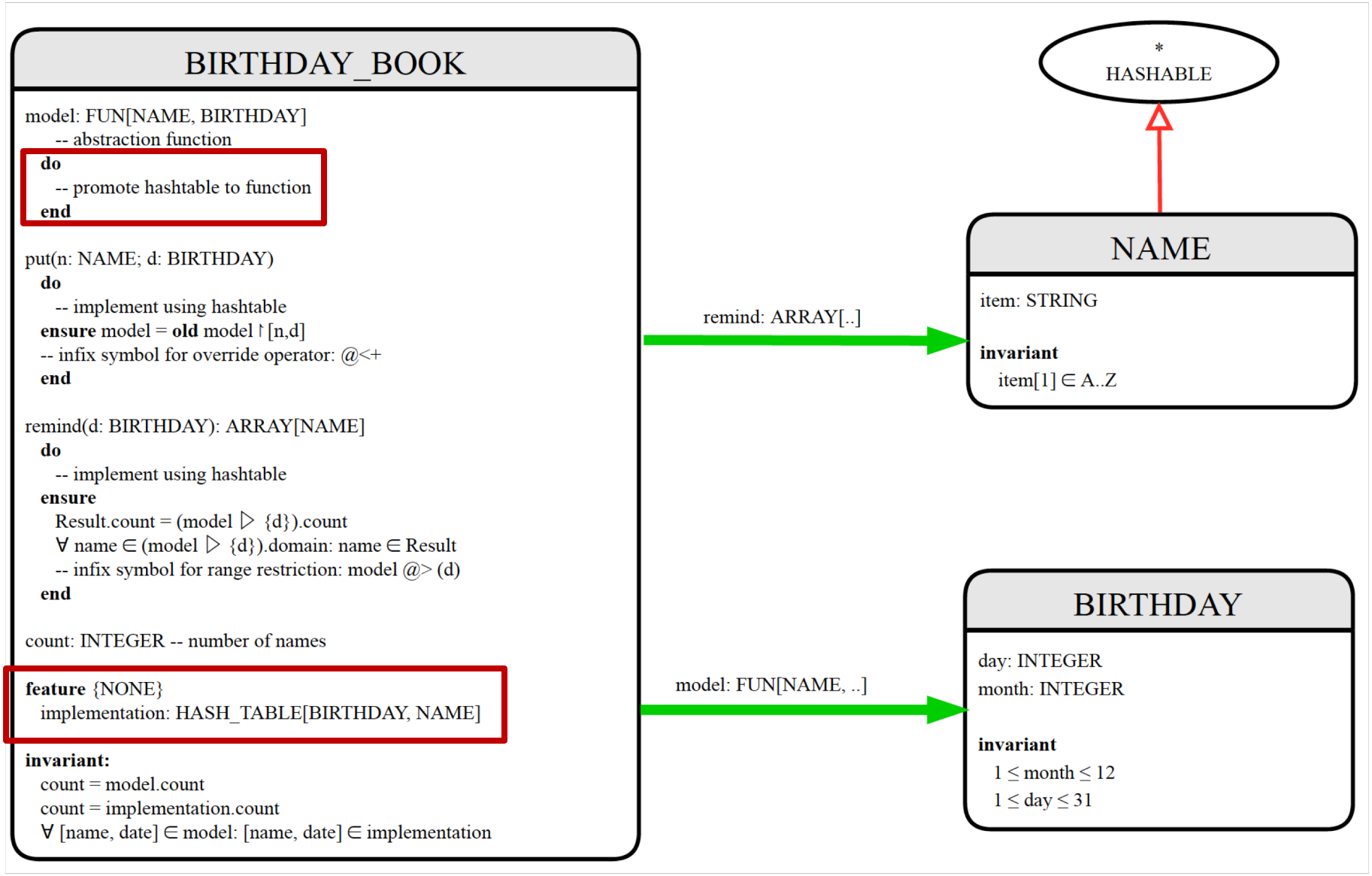
NAME

item: STRING

invariant
item[1] ∈ A..Z



Efficient hashtable implementation



```
class interface BIRTHDAY_BOOK feature

    model: FUN [NAME, BIRTHDAY]
            -- model is a function from NAME --> BIRTHDAY
            -- abstraction function

    put (a_name: NAME; d: BIRTHDAY)
            -- add date of birth for `a_name` at date `d`
            -- or overrride current birthday with new
        ensure
            model_override:
                model ~ (old model @<+ [a_name, d])

    remind (d: BIRTHDAY): ARRAY [NAME]
            -- returns an array of names with birthday `d`
        ensure
            remind_count: Result.count = (model @> (d)).count
            remind_model_range_restiction:
                across
                    (model @> (d)).domain as cr
                all
                    Result.has (cr.item)
                end

end
```

Efficient hash table implementation

```
feature {NONE, ES_TEST} -- implementation

    imp: HASH_TABLE [BIRTHDAY, NAME]
            -- implementation as an efficient hash table

    make
            -- create a birthday book
        do
            create imp.make (10)
        ensure
            model.is_empty
        end

feature -- model

    model: FUN [NAME, BIRTHDAY]
            -- model is a function from NAME --> BIRTHDAY
            -- abstraction function
            -- Abstraction Function from an object's concrete implementation
            -- to the abstract value it represent
        local
            l_name: NAME
            l_date: BIRTHDAY
        do
            create Result.make_empty
            from
                imp.start
            until
                imp.after
            loop
                l_name := imp.key_for_iteration
                l_date := imp [l_name]
                check attached l_date as l_date2 then
                    Result.extend ([l_name, l_date2])
                end
                imp.forth
            end
            imp.start
        end
```

Efficient hash table implementation

```
put (a_name: NAME; d: BIRTHDAY)
        -- add birthday for `a_name' at date `d'
        -- or overrride current birthday with new
    do
        if not imp.has_key (a_name) then
            imp.extend (d, a_name)
        else
            imp.replace (d, a_name)
        end
    ensure
        model_override:
            model ~ (old model @<+ [a_name, d])
    end
```

```
feature {NONE, ES_TEST} -- implementation

    imp: HASH_TABLE [BIRTHDAY, NAME]
            -- implementation as an efficient hash table
```

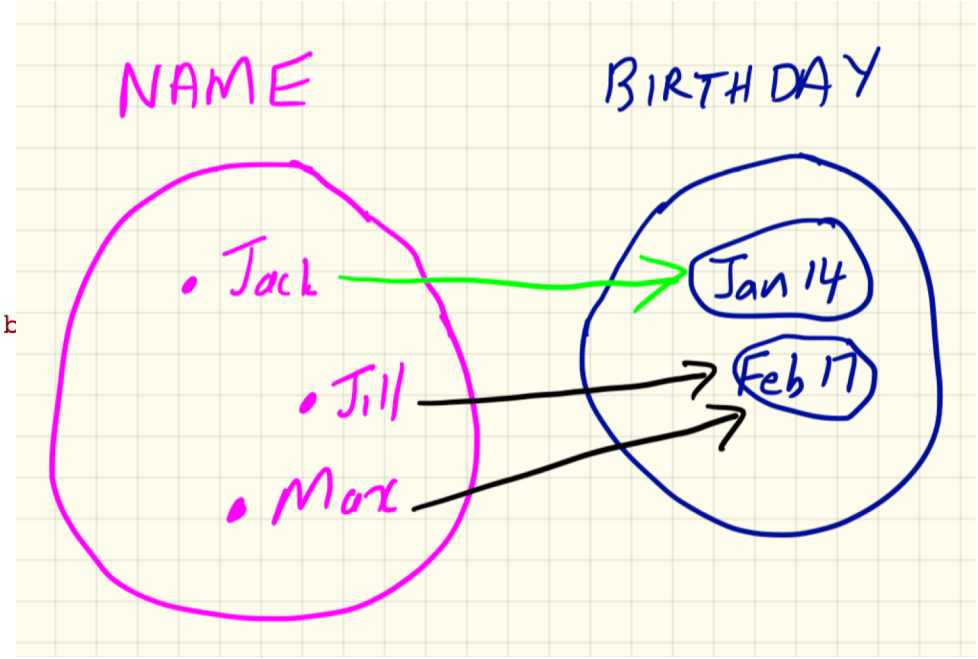
Efficient hash table implementation

```
remind (d: BIRTHDAY): ARRAY [NAME]
        -- returns an array of names with birthday `d'
    local
        l_name: NAME
        l_date: BIRTHDAY
        i: INTEGER
    do
        create Result.make_empty
        Result.compare_objects
        from
            imp.start
        until
            imp.after
        loop
            l_name := imp.key_for_iteration
            l_date := imp [l_name]
            if l_date ~ d then
                Result.force (l_name, i)
                i := i + 1
            end
            imp.forth
        end
    ensure
        remind_count:
            Result.count = (model @> (d)).count
        remind_model_range_restiction:
            across (model @> (d)).domain as cr all
                Result.has (cr.item)
            end
    end
```

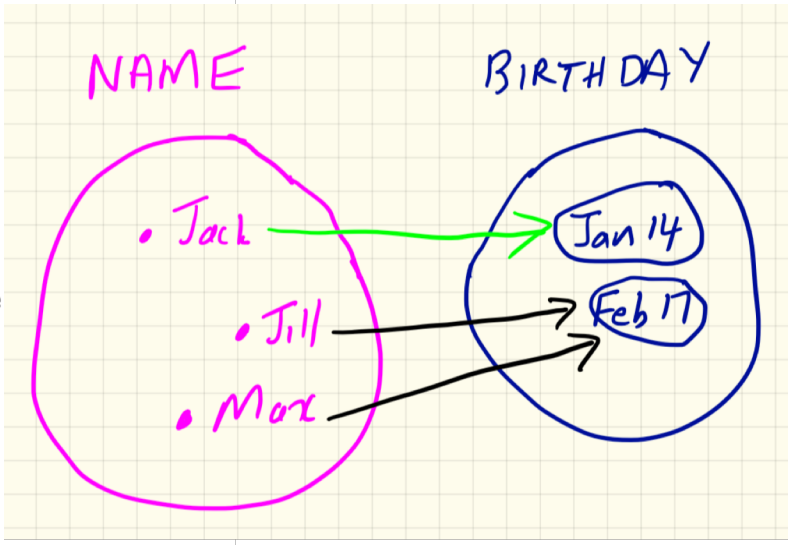
```
feature {NONE, ES_TEST} -- implementation

    imp: HASH_TABLE [BIRTHDAY, NAME]
            -- implementation as an efficient hash table
```

Efficient hash table implementation

```
remind (d: BIRTHDAY): ARRAY [NAME]
        -- returns an array of names with b
    local
        l_name: NAME
        l_date: BIRTHDAY
        i: INTEGER
    do
        create Result.make_empty
        Result.compare_objects
        from
            imp.start
        until
            imp.after
        loop
            l_name := imp.key_for_iteration
            l_date := imp [l_name]
            if l_date ~ d then
                Result.force (l_name, i)
                i := i + 1
            end
            imp.forth
        end
    ensure
        remind_count:
            Result.count = (model @> (d)).count
        remind_model_range_restiction:
            across (model @> (d)).domain as cr all
                Result.has (cr.item)
            end
    end
```



```
t1: BOOLEAN
    local
        feb17: BIRTHDAY
        bb: BIRTHDAY_BOOK
        a: ARRAY[NAME]
        l_name: NAME
        l_birthday: BIRTHDAY
        jack,jill,max: NAME
    do
        comment("t1: test birthday book")
        create jack.make ("Jack"); create jill.make
        create max.make ("Max")
        create feb17.make (02, 17) -- Feb 17
        create bb.make
        -- add birthdays for Jack and Jill
        bb.put (jack, [14,01])  -- Jan 14
        bb.put (jill, [17,02])  -- Feb 17
        Result := bb.model.count = 2
            and bb.model[jack] ~ [14,01]
            and bb.model[jill] ~ [17,02]
            and bb.out ~ "{ Jack -> (14,1), Jill -> (17,2) }"
        check Result end
        -- Add birthday for Max
        bb.put (max, [17,02])
        assert_equal ("Max's birthday", bb.model[max], feb17)
        -- Whose birthday is Feb 17th?
        a := bb.remind ([17,02])
        Result := a.count = 2
            and a.has (max)
            and a.has (jill)
            and not a.has (jack)
        check Result end
        -- check efficienthash table implementation
        across bb.model as cursor loop
            l_name := cursor.item.first
            l_birthday := cursor.item.second
            check
                bb.imp.has (l_name)
                and then bb.imp[l_name] ~ l_birthday
            end
        end
    end
```



```
class BIRTHDAY inherit
    ANY
        redefine
            out
        end
create
    make, make_from_tuple
convert
    make_from_tuple ({TUPLE [INTEGER, INTEGER]})

feature {NONE} -- Initialization

    make (a_month: INTEGER; a_day: INTEGER)
            -- Initialization for `Current'.
        do
            month := a_month
            day := a_day
        end

    make_from_tuple (t: TUPLE [day: INTEGER; month: INTEGER])
        require
            t.count = 2
            attached {INTEGER} t.day
            attached {INTEGER} t.month
        do
            make (t.month, t.day)
        end

feature
    day: INTEGER
    month: INTEGER
```

You'd rather discover the bug...

Fault could send Jaguars into reverse

By Graham Tibbetts

JAGUAR has recalled thousands of cars after discovering a gearbox fault that could send them into reverse at high speed.

The problem, traced to computer software, could cause drivers to lose control, safety experts said. But in most cases it would wreck the gearbox and engine.

The company has recalled 68,000 vehicles, including 14,000 in Britain.

The ranges affected are the top-of-the-range XJ model, which costs from £65,000, the S-Type, which costs up to £47,000, and the £59,000 XK coupe. Those at risk were all built before June 2003.

The recall, which is expected to cost the company millions of pounds, follows four cases of the gearboxes selecting reverse.

The Premiership footballers Michael Owen and Alan Shearer were believed to have been among those contacted, along with Gabby Logan, the television sports presenter.

A Jaguar spokesman said yesterday that if oil pressure in the transmission system reached a very low point a warning light would come on.

"In this condition it is possible that reverse could be selected by mistake," he said.

"There have been four instances of this problem, including one in the UK, but there were no crashes and no one was hurt."

The company said the repair work, involving the reprogramming of the transmission control module, would take about one hour.

… before your users discover, on your behalf, that in fact:

WHEN Oil Pressure Low

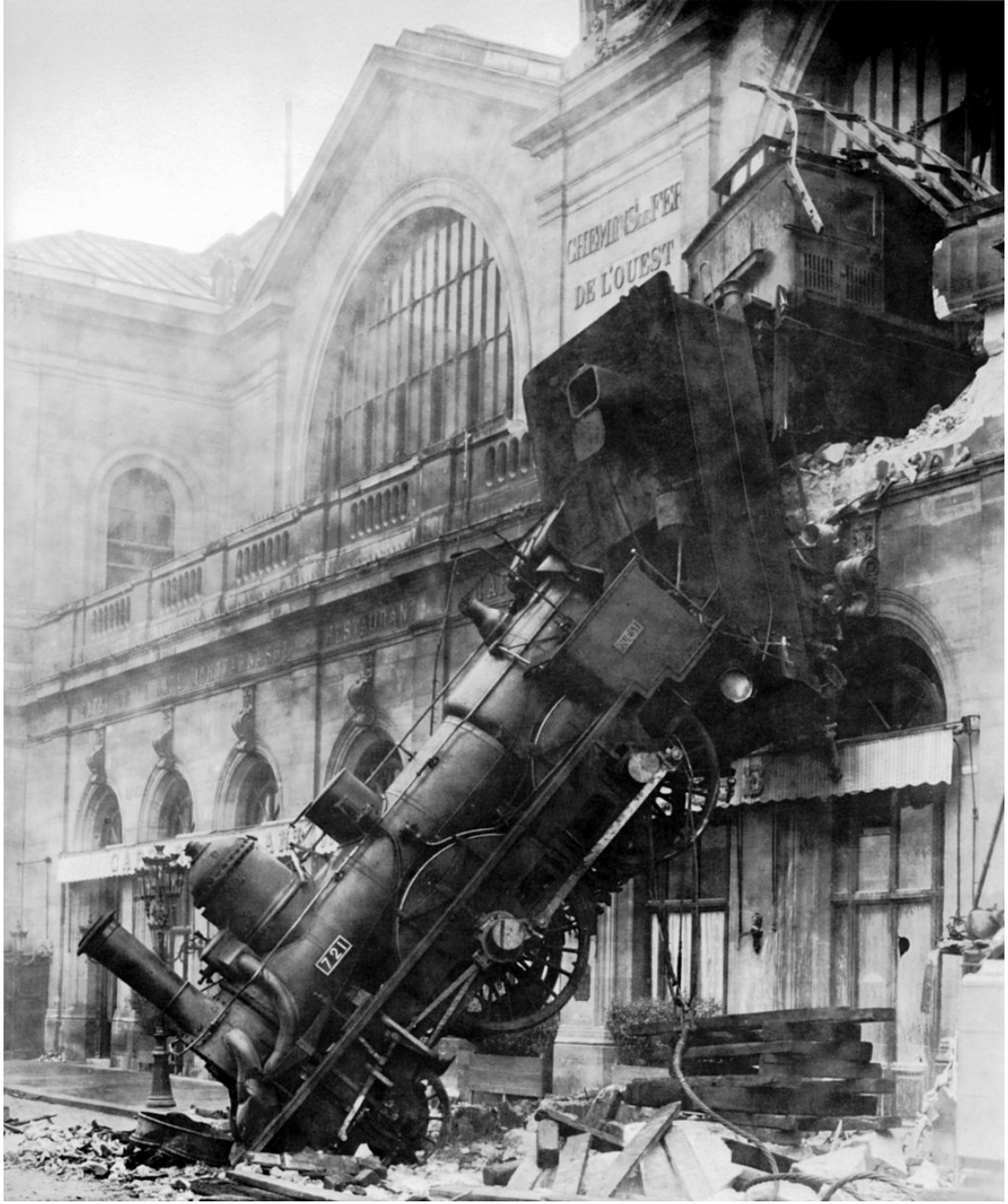
AND *unspecified conditions*

THEN Reverse Gear Engages Automatically on the Highway

AND 68,000 formerly loyal customers spend a day visiting their garage

Some recent software disasters

- Bay Area Rapid Transit new extension two years behind due to errors
- Uber and Tesla self-driving car crashes
- Court software is so awful it's getting people wrongly arrested
- FAA Requiring Airlines to Reboot Dreamliners Every Three Weeks
- Software Error crashes Mars Lander
- Stanford Mall robot runs over small child
- Nanaimo doctors say electronic health record system unsafe, should be shut down
- Google to cloud customers: Sorry, but here's how two bugs knocked us out worldwide
- A bug in fMRI software could invalidate 15 years of brain research
- Volvo recalls 59,000 cars over software fault
- Patch Linux now, Google, Red Hat warn, over critical glibc bug
- Billion-dollar mistake: How inferior IT killed Target Canada
- Centrelink blames computer glitch for wrongly billing 73,000 customers



SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2003; **33**:637–672 (DOI: 10.1002/spe.520)

Investigating the use of analysis contracts to improve the testability of object-oriented code

L. C. Briand*,†, Y. Labiche and H. Sun

SP&E

Contracting -- order of magnitude improvement in empirical tests

Another important result shown to be very strong and of high practical significance is that diagnosability, as we measure it, improves nearly an order of magnitude between mutant programs without contract assertions and those with contract assertions, regardless of the level of precision. This suggests significant savings during debugging, as faults will be much easier to locate if contracts are being instrumented and checked during execution.

In conclusion, our overall results suggest that instrumented contracts have a strong potential in terms of decreasing the cost of testing. Furthermore, they do not need to be defined and instrumented at a high level of precision to be useful. Those results are the outcome of a carefully planned and designed study and can be replicated on other system examples, using well-defined procedures and a commercial tool. Replication is essential in order to confirm the results we obtained here.

Contracting -- strong decrease in cost of testing

§They cannot be detected by inst… regarding where the exception was raised.

Unix string concatenation

Let's use Linux string concatenation (strtcat) to concatenate a string to itself, e.g.

```
# include <stdio.h> /* For printf() */
# include <string.h> /* For strcat() */
int main (void)
{
        char s[11] = "hello";
        strcat(s,s);
        printf("%s\n",s);
        return 0; //program exits successfully
}
```

```
/* What happens?
   Hopefully:
   "hellohello"
*/
```

Prism Linux

```
red% more append.c

# include <stdio.h> /* For printf() */
# include <string.h> /* For strcat() */
int main (void)
{
        char s[11] = "hello";
        strcat(s,s);
        printf("%s\n",s);
        return 0;
}

red% gcc append.c
red% ./a.out
Segmentation fault (core dumped) – centos6
red%
```

```
mac% ls
append.c
mac% more append.c
```

On my Mac/Mojave

```
# include <stdio.h> /* For printf() */
# include <string.h> /* For strcat() */

int main (void)
{
        char s[11] =
"hello";  strcat(s,s); printf("%s\n",s);

        return 0; //program exits successfully
}

mac% gcc append.c
mac% ls

a.out      append.c

mac% ./a.out

Abort trap: 6
```

Prism Linux

man strcat

STRCAT(3) Linux Programmer's Manual

NAME

strcat, strncat - concatenate two strings

SYNOPSIS

#include <string.h>

char *strcat(char *dest, const char *src);

char *strncat(char *dest, const char *src, size_t n);

DESCRIPTION

The strcat() function appends the src string to the dest string overwriting the `\0' character at the end of dest, and then adds a terminating `\0' character. The strings may not overlap, and the dest string must have enough space for the result.

Testing the informal documentation

- Testing string concatenation on several platforms yields
  - different results depending on the platform:
  - crashes due to segmentation error (Solaris 5.6).
  - crashes due to uncaught signal (macos Abort trap: 6).
  - crashes due to arithmetic error (!?).
  - hangs in an infinite loop.
  - success (yields “hellohello”).

Contracts for strings

```
append (s: STRING)
            -- append copy of 's' at end of Current.
        require
            argument_not_void:
                s /= Void
            argument_must_be_different:
                s /= Current
        external
            -- C code goes here
        ensure
            new_count:
                count = old count + old s.count
            argument_appended: -- try your skill ?
            old_prefix_unchanged: -- try your skill ?
        end
```

```
class
    EIFFEL_WITH_C
feature

    max3 (x, y, z: REAL_64): REAL_64
        external
            "[
                C use "eiffel_with_c.h"
            ]"
            alias "maxThree"
        end
```

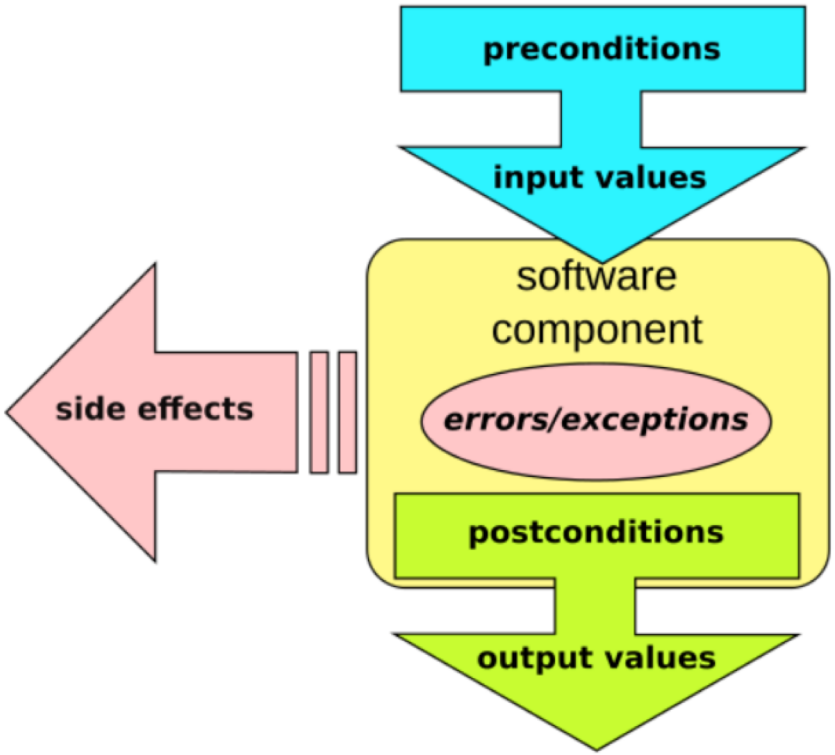
```
double maxThree(double x, double y, double z) {
    if (x>=y)
    {
        if(x>=z)
            return x;
        else
            return z;
    }
    else
    {
        if(y>=z)
            return y;
        else
            return z;
    }
}
```

C++ Core Guidelines: A Short Detour to Contracts in C++20

20 July 2018

Contents [Show]

My original plan was it to write in this post about the next rules to error handling. But I changed my plan to write about the future: contracts in C++20.



Ariane-5

- On June 4, 1996, the maiden flight of the European Ariane 5 launcher crashed about 40 seconds after takeoff.
- Media reports indicated that the amount lost was half a billion dollars -- uninsured.

Ariane-5 investigation

- The CNES (French National Center for Space Studies) and the European Space Agency immediately appointed an international inquiry board, made of respected experts from major European countries, who produced their report in hardly more than a month
- Its conclusion: the explosion was the result of a software error -- possibly the costliest in history up to that point (at least in dollar terms, since earlier cases have caused loss of life).

Ariane-5 investigation (2)

- The exception was due to a floating-point error in the Inertial Reference System
- a conversion from a 64-bit integer to a 16-bit signed integer, which should only have been applied to a number less than 2^15, was erroneously applied to a greater number, representing the "horizontal bias" of the flight.
- There was no explicit exception handler to catch the exception, so it followed the usual fate of uncaught exceptions and crashed the entire software, hence the on-board computers, hence the mission.

Ariane-5 investigation (3)

- The Inertial Reference subsystem in the Araine-5 was re-used from the Ariane-4 where the documented constraint was calculated to hold.
- That constraint never made it into the software code!

```
convert (horizontal_bias: INTEGER): INTEGER
    require
        horizontal_bias <= Maximum_bias   -- from Ariane-4
    do
      ...
    ensure
       ...
    end
```

Environmental constraint from the problem domain

How would the precondition have helped?

- Contracts are a prime component of the software and its documentation ("contract view" is produced automatically by tools).
- In an environment such as that of Ariane where there is much emphasis on thorough validation of everything, the contracts would be the QA team's primary focus of attention.

- Any team worth its salt would have checked systematically that every call satisfies the precondition.

Hoare Logic

The meaning of a program statement S is described by a triple

$$\{ P \} \quad S \quad \{ Q \}$$

where P is called the precondition

Q is called the postcondition

Informal meaning

"when program S is started in a state satisfying P, all executions of S terminate in a state satisfying Q"

Hoare Logic

{ P } S { Q }

{ ? } x := x-1 { (x+y < 0) and (y = old y) and (x = old x - 1)}

{ ? } x := x*x { x^4 = 10}

Hoare Logic

{ P } S { Q }

{ x + y < 1 } x := x-1 { (x+y < 0) and (y = old y) and (x = old x - 1)}

{ ? } x := x*x { x^4 = 10}

{ P } S { Q }

Weakest precondition
calculation of precondition from postcondition

wp("x := exp", Q) $\triangleq$ Q[x := exp]

In postcondition Q,

- replace (old x) by constant x_{-1}
- replace (old y) by constant y_{-1}
- { $x=x_{-1} \wedge y=y_{-1} \wedge$ ...?} S { Q }

Hoare Logic -- replace (old x) by x_1

{ P } S { Q }

{$x=x_{-1} \wedge y=y_{-1} \wedge ?$} x := x*x {Q: $x+y<0 \wedge (y=y_{-1}) \wedge (x=x_{-1}-1)$}

wp("x := x-1", Q)

= < replace by wp defn.>

Q [x := x-1]

= <replace Q by its defn>

$(x+y < 0 \wedge (y = y_{-1}) \wedge (x = x_{-1} - 1))$ [x := x-1]

= < do the substitution for x>

$(x-1)+y < 0 \wedge (y = y_{-1}) \wedge ((x-1) = x_{-1} -1)$

= < arithmetic and Leibniz>

$(x+y < 1) \wedge (y=y_{-1}) \wedge (x=x_{-1})$

Hoare Logic -- replace (old x) by x_1

{ P } S { Q }

{x=x_1 ∧ ?} x := x*x {Q: x^4 = 10}

wp("x := x*x1", Q)

= < replace by wp defn.>

Q [x := x*x]

= <replace Q by its defn>

(x^4 = 10) [x := x*x1]

= < do the substitution for x>

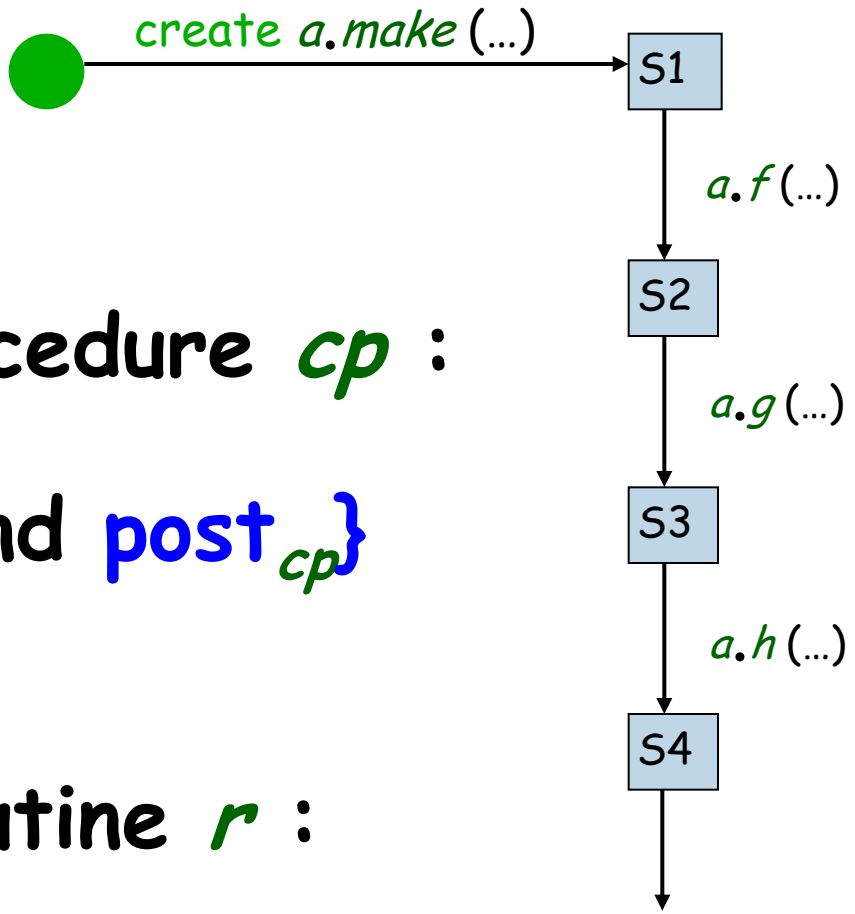
(x**x)^4 = 10

= < arithmetic and Leibniz>

x^8 = 10

Week	Date	Labs	Weights	Required Readings
1	Thu 03 Jan	Lab0		OOSC2 Chapters 7 and 8 (Classes vs. Objects, Reference vs. Value Types)
2	Mon 07 Jan	Lab1	3%	OOSC2 Chapter 11.1-11.9 (Specifications, Design by Contract)
3	Mon 14 Jan	Lab1		OOSC2 Chapter 6 (Abstract Data Types)
4	Mon 21 Jan	Lab2	3%	OOSC2 Chapter 10 (Genericity)
5	Mon 28 Jan	Lab3	4%	OOSC2 Chapter 14 (Inheritance)
6	Mon 04 Feb	Lab4	5%	OOSC2 Chapter 15 (Multiple Inheritamce)
7	Mon 11 Feb	Lab4		OOSC2 Chapter 11.10 -11.15 (DbC, ADTs and loop variants/inv)
Mon 18 Feb				OOSC2 Chapter 21 (Undo/redo Design pattern)
8	Mon 25 Feb	Labtest 1	10%	OOSC2 Chapter 12 (Broken contract, exceptions)
9	Mon 04 Mar	Project		OOSC2 Chapter 17 (Class as a merge of Typing/Modularity)
10	Mon 11 Mar	Labtest 2	10%	OOSC2 Chapters 20, 23.1, 23.4, 23.8 (Design of Classes)
11	Mon 18 Mar	Lab 5	5%	OOSC2 Chapters 1-3 (Software Quality/Modularity)
12	Mon 25 Mar	Project	15%	OOSC2 chapter 18 (Global constants)
13	Mon 01 April			OOSC2 Chapter 26 (A Sense of Style)
	5-20, April	Exam	45%	See also *Touch of Class*

The correctness of a class



For every creation procedure *cp* :

$\{pre_{cp}\}\ do_{cp}\ \{INV \text{ and } post_{cp}\}$

For every exported routine *r* :

$\{INV \text{ and } pre_{r}\}\ do_{r}\ \{INV \text{ and } post_{r}\}$

- Any team worth its salt would have checked systematically that every call satisfies the precondition, i.e.
- {INV & Pre} sri.convert(horizontal_bias) {INV & post}

```
{horizontal_bias <= Maximum_bias}
sri.convert(horizontal_bias)
{INV & post}
```

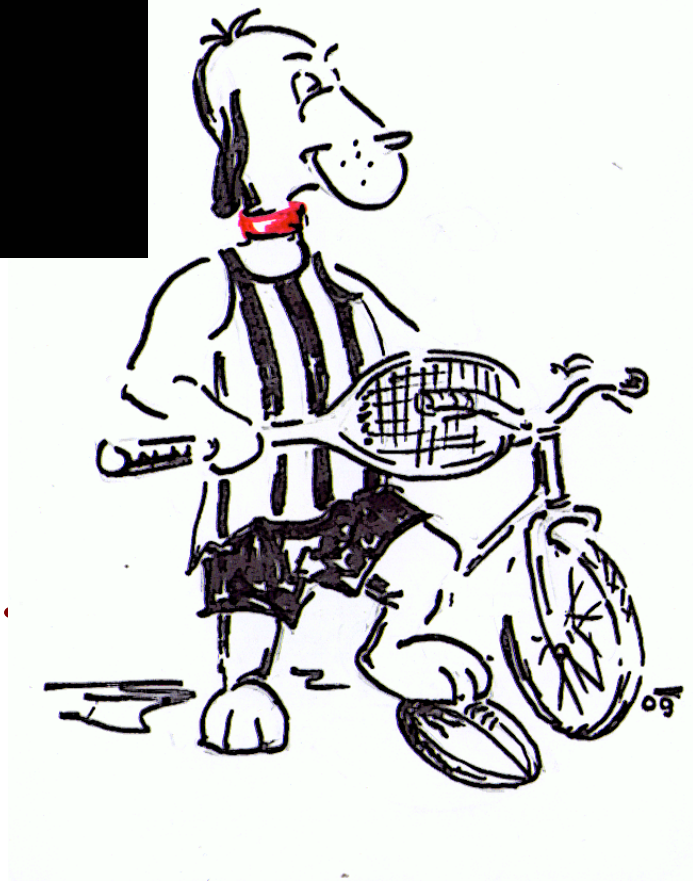
- Any team worth its salt would have checked systematically that every call satisfies the precondition, i.e.
- {INV & Pre} sri.convert(horizontal_bias) {INV & post}

```
{horizontal_bias <= Maximum_bias}
sri.convert(horizontal_bias)
{INV & post}
```

- Do the above check either via calculation or run-time assertion checking
- That would have immediately revealed that the Ariane 5 calling software did not meet the expectation of the Ariane 4 routines that it called.

- The Inquiry Board made a number of recommendations with respect to improving the software process of the European Space Agency. Many are justified; some may be overkill; some may be very expensive to put in place.
- There is a more simple lesson to be learned from this unfortunate event:
- REUSE WITHOUT A CONTRACT IS SHEER FOLLY

If they stop explosions ...
Contracts are cool!



a_other.pair is a qualified call

```
note
     description: "Two members make a team"
class
     MEMBER
feature
     team_mate: detachable like Current

     pair (a_other: like Current)
          do
               team_mate := a_other
               a_other.pair (Current) -- problematic!
          end
invariant
      team_mate.team_mate = Current -- problematic?
end
```

```
team_mate: detachable MEMBER

pair(other: like Current)
    do
        team_mate := other
    end
```

Call Stack

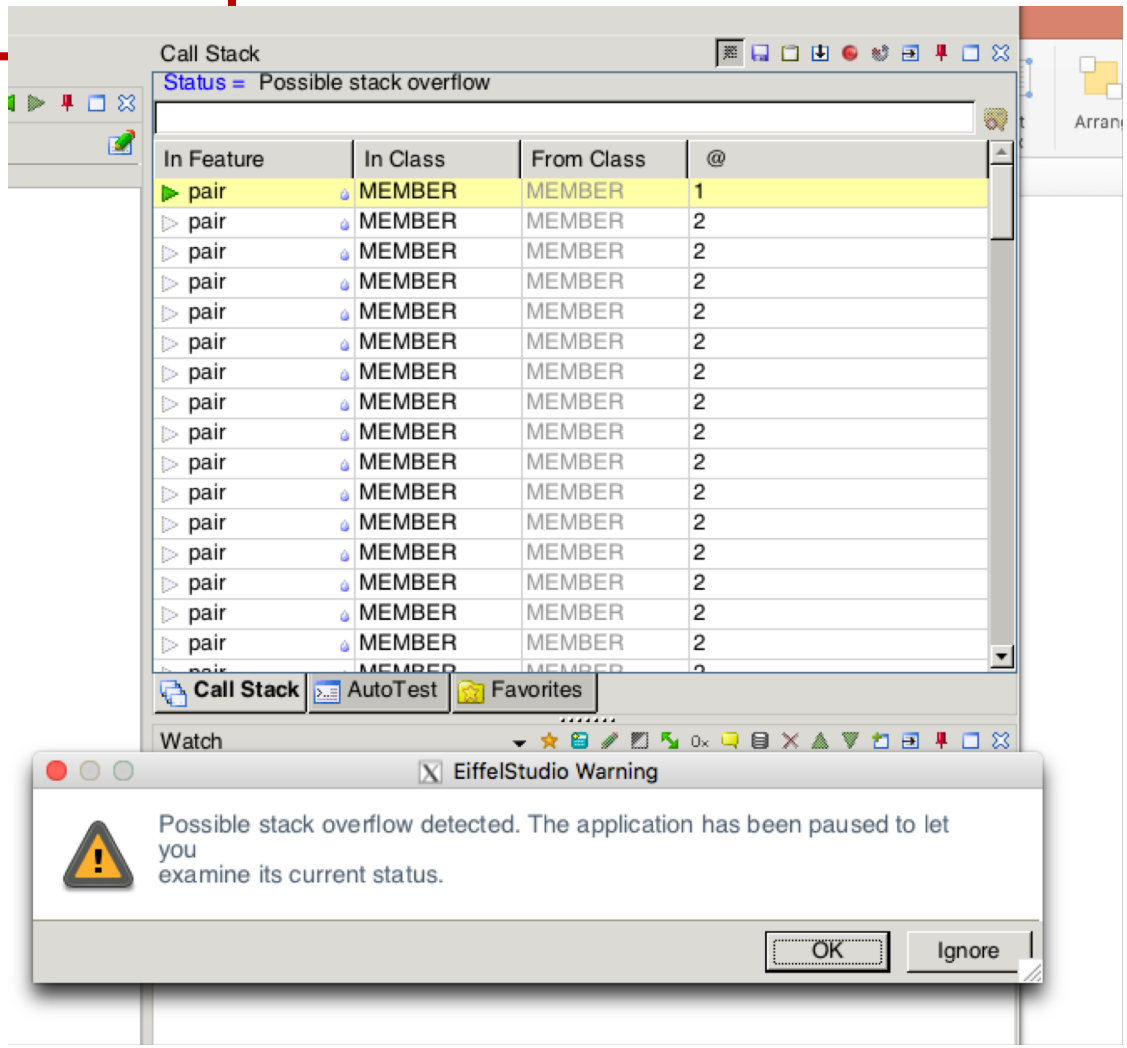
Status = Implicit exception pending

CHECK_VIOLATION raised

In Feature	In Class	From Class	@
t0	TESTS	TESTS	6
fast_item	PREDICATE	FUNCTION	0
item	PREDICATE	FUNCTION	5
run	ES_BOOLEA...	ES_BOOLEA...	3
run_es_test	TESTS	ES_TEST	20
run_es_test	ROOT	ES_SUITE	9
run_espec	ROOT	ES_TESTABLE	2
make	ROOT	ROOT	4

```
t0: BOOLEAN
    local
        m1, m2: MEMBER
    do
        comment ("t0: test symmetry pairing team mates")
        create m1
        create m2
        m1.pair (m2)
        Result := m1.team_mate = m2 and m2.team_mate = m1
        check
            Result
        end
    end
```

```
team_mate: detachable MEMBER

pair(other: like Current)
     do
          team_mate := other
          other.pair (Current)
     end
```



```
class
    MEMBER
feature
    team_mate: detachable MEMBER

    pair(other: like Current)
        do
            team_mate := other
--          other.pair (Current) -- stack overflow
            other.imp_pair (Current)
        end

feature{MEMBER}
    imp_pair(other: like Current)
        do
            team_mate := other
        end

invariant
    spouse_of_spouse:
    attached team_mate as mate
    and then attached mate.team_mate
    implies
    mate.team_mate = Current
end
```

Complementarity of Tests and Contracts

Specification tests vs Unit Tests

Square root test – precision?

```
epsilon: REAL = .001

test_square_root_of_ten: BOOLEAN
   local
      m: MATH
      d: REAL
   do
      create m
      d := 10 - (m.sqrt (10) * m.sqrt (10))
      Result :=  d.abs <= epsilon
   end
```

$$\left|\sqrt{10} * \sqrt{10} - 10\right| \leq .001$$

Tests vs. Contracts

- Test

  Result :=

  (m.sqrt (10)*m.sqrt (10) - 10).abs <= m.epsilon

- Contract

  (Result * Result – x).abs <= epsilon

- The contract is a more general specification of the correct behaviour than the test
- The unit test can be used to exercise and verify the contract.

  i.e. the test "amplifies the contract"

```
epsilon: REAL

sqrt (x:REAL): REAL
        -- return square root of `x'
    require
        non_zero: x >= 0
    local
        guess: REAL
    do
        from
            guess := 1
        until
            (guess*guess - x).abs <= epsilon
        loop
            guess := .5*(guess + x/guess)
        end
            Result := guess
    ensure
        result_accurate: (Result * Result - x).abs <= epsilon
        epsilon_unchanged: epsilon = old epsilon
    end
```

Unit Test

```
t0: BOOLEAN
    local
        s: SQUARE_ROOT
        sqrt10: REAL_64
    do
        comment("t0: Unit Test for sqrt")
        create s.make
        sqrt10 := s.sqrt (10)
        Result := sqrt10-3.1623 <= s.epsilon
    end
```

Specification Test

```
t1: BOOLEAN
    local
        s: SQUARE_ROOT
        x: REAL_64
        r: RANDOM
        low, hi: REAL_64
    do
        comment("t0: Specification Test for sqrt")
        create r.make
        -- range of random numbers
        low := 0.01; hi := 20000.00
        create s.make
        across 1 |..| 100000 as i loop
            x := r.real_item
            x := x * ( hi - low )
            r.forth
            Result := (x - s.sqrt (x)*s.sqrt (x)).abs
                      <= s.epsilon
            check Result end
        end
    end
```

Specification Tests

- Once you start thinking specifications (contracts) you can write specification tests in any language with a unit testing framework

Loop Variants and Invariants

```
from
    init statements
invariant
    assertions for invariant
until
    exit condition
loop
    body statements
variant
  integer expression
end
```

- always non negative
- body decreases value on every iteration

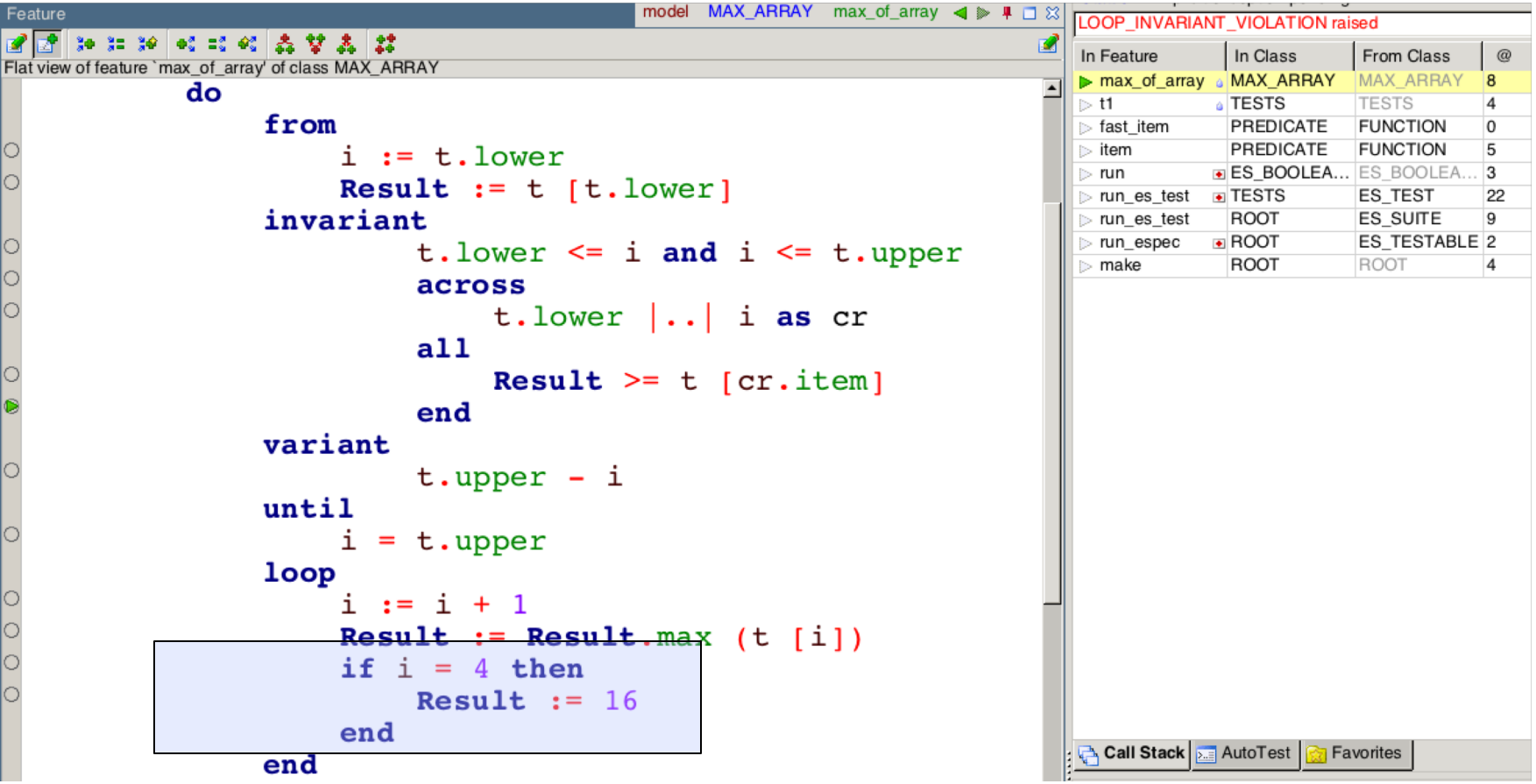
```
sum: INTEGER
sum_of(a: ARRAY[INTEGER])
        -- calculate the sum of all elements in array a
    local i: INTEGER;
    do
        from
                i := a.lower; sum := 0
        invariant
                a.lower <= i
                i <= a.upper + 1
        until
                i > a.upper
        loop
                sum := sum + a [i]
                i := i + 1
        variant
                a.upper + 1 - i
        end
    end
```

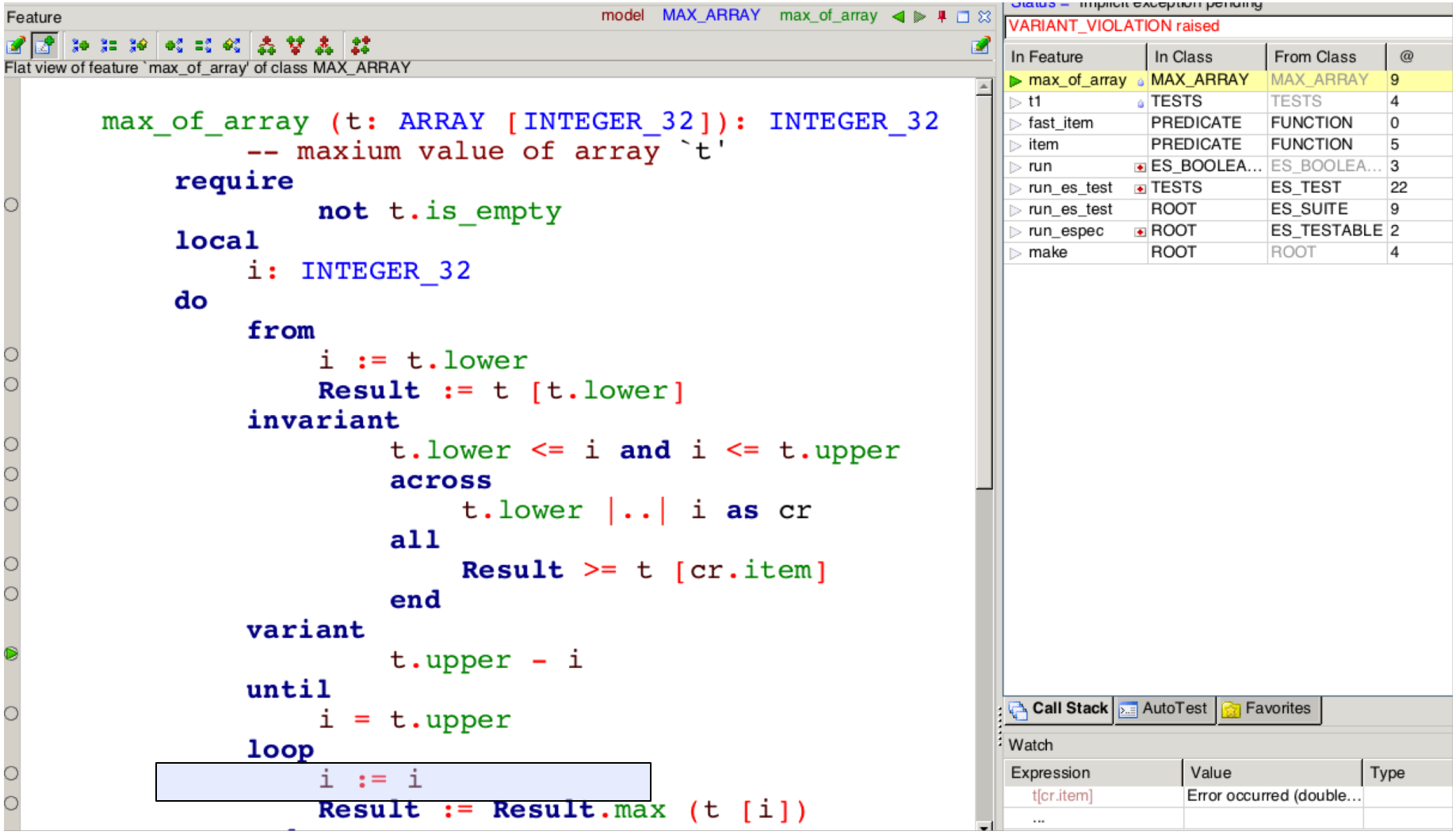
Guarantees loop termination

```
max_of_array(t: ARRAY[INTEGER]):INTEGER
        -- maxium value of array `t'
    require
        not t.is_empty
    local
        i: INTEGER
    do
        from
            i := t.lower
            Result := t[t.lower]
        invariant
            t.lower <= i and i <= t.upper
            across t.lower |..| i as cr all
                Result >= t[cr.item]
            end
        until
            i = t.upper
        loop
            i := i + 1
            Result := Result.max (t[i])
--          if i = 4 then Result := 16 end  -- coding error
        variant
            t.upper - i
        end
    ensure
        across t.lower |..| t.upper as cr all
                Result >= t[cr.item]
        end
    end
```

```
max_of_array(t: AR
        -- maxium
    require
        not t.is_
    local
        i: INTEGE
    do
        from
            i :=
            Resul
        invariant
            t.lower <= i and i <= t.upper
            across t.lower |..| i as cr all
                Result >= t[cr.item]
            end
        until
            i = t.upper
        loop
            i := i + 1
            Result := Result.max (t[i])
--          if i = 4 then Result := 16 end  -- coding error
        variant
            t.upper - i
        end
    ensure
        across t.lower |..| t.upper as cr all
                Result >= t[cr.item]
        end
    end
```

```
t1: BOOLEAN
        local
            a: ARRAY[INTEGER]
            am: MAX_ARRAY
        do
            comment("test_max_of_array")
            create am
            a := <<191, 32, 423, 16>>
            Result := am.max_of_array (a) = 423
        end
```


Feature
model MAX_ARRAY max_of_array
Flat view of feature `max_of_array' of class MAX_ARRAY
do
from
i := t.lower
Result := t [t.lower]
invariant
t.lower <= i and i <= t.upper
across
t.lower |..| i as cr
all
Result >= t [cr.item]
end
variant
t.upper - i
until
i = t.upper
loop
i := i + 1
Result := Result.max (t [i])
if i = 4 then
Result := 16
end
end
LOOP_INVARIANT_VIOLATION raised
In Feature
In Class
From Class
@
max_of_array MAX_ARRAY MAX_ARRAY 8
t1 TESTS TESTS 4
fast_item PREDICATE FUNCTION 0
item PREDICATE FUNCTION 5
run ES_BOOLEA... ES_BOOLEA... 3
run_es_test TESTS ES_TEST 22
run_es_test ROOT ES_SUITE 9
run_espec ROOT ES_TESTABLE 2
make ROOT ROOT 4
Call Stack
AutoTest
Favorites


Feature
model MAX_ARRAY max_of_array
Flat view of feature `max_of_array' of class MAX_ARRAY
max_of_array (t: ARRAY [INTEGER_32]): INTEGER_32
-- maxium value of array `t'
require
not t.is_empty
local
i: INTEGER_32
do
from
i := t.lower
Result := t [t.lower]
invariant
t.lower <= i and i <= t.upper
across
t.lower |..| i as cr
all
Result >= t [cr.item]
end
variant
t.upper - i
until
i = t.upper
loop
i := i
Result := Result.max (t [i])
VARIANT_VIOLATION raised
In Feature
In Class
From Class
@
max_of_array MAX_ARRAY MAX_ARRAY 9
t1 TESTS TESTS 4
fast_item PREDICATE FUNCTION 0
item PREDICATE FUNCTION 5
run ES_BOOLEA... ES_BOOLEA... 3
run_es_test TESTS ES_TEST 22
run_es_test ROOT ES_SUITE 9
run_espec ROOT ES_TESTABLE 2
make ROOT ROOT 4
Call Stack
AutoTest
Favorites
Watch
Expression
Value
Type
t[cr.item]
Error occurred (double...
...

Abstract Data Types

This opened my mind, I started to grasp what it means to use the tool known as algebra. I'll be damned if anyone had ever told me before: over and again Mr. Dupuy [the mathematics teacher] was making pompous sentences on the subject, but not once would he say this simple word: it is a ***division of labor****, which like any division of labor produces miracles, and allows the mind to concentrate all of its forces on just one side of objects, on just one of their qualities.*

Stendhal, *The Life of Henry Brulard*, 1836.

$$cos^2\,(a-b) + sin^2\,(a+b-2\times b)$$

- $STACK\ [G]$

FUNCTIONS

- put: $STACK\ [G] \times G \rightarrow STACK\ [G]$
- $remove$: $STACK\ [G] \nrightarrow STACK\ [G]$
- $item$: $STACK\ [G] \nrightarrow G$
- $empty$: $STACK\ [G] \rightarrow BOOLEAN$
- new: $STACK\ [G]$

AXIOMS

For any x: G, s: $STACK\ [G]$

A1 • $item\ (put\ (s, x)) = x$

A2 • $remove\ (put\ (s, x)) = s$

A3 • $empty\ (new)$

A4 • **not** $empty\ (put\ (s, x))$

PRECONDITIONS

- $remove\ (s$: $STACK\ [G])$ **require not** $empty\ (s)$
- $item\ (s$: $STACK\ [G])$ **require not** $empty\ (s)$

Use Axioms to calculate

$$item\ (remove\ (put\ (remove\ (put\ (put\ (\\ remove\ (put\ (put\ (put\ (new, x1), x2), x3)),\\ item\ (remove\ (put\ (put\ (new, x4), x5)))), x6)), x7)))$$

$= x4$

- *STACK* [*G*]

FUNCTIONS

- *put*: *STACK* [*G*] × *G* → *STACK* [*G*]
- *remove*: *STACK* [*G*] ↛ *STACK* [*G*]
- *item*: *STACK* [*G*] ↛ *G*
- *empty*: *STACK* [*G*] → *BOOLEAN*
- *new*: *STACK* [*G*]

AXIOMS

For any *x*: *G*, *s*: *STACK* [*G*]

A1 • *item* (*put* (*s*, *x*)) = *x*

A2 • *remove* (*put* (*s*, *x*)) = *s*

A3 • *empty* (*new*)

A4 • **not** *empty* (*put* (*s*, *x*))

PRECONDITIONS

- *remove* (*s*: *STACK* [*G*]) **require not** *empty* (*s*)
- *item* (*s*: *STACK* [*G*]) **require not** *empty* (*s*)

Abstract State Machine

```
class ABSTRACT_STACK [G -> attached ANY] inherit
    ANY
        redefine is_equal end
create
    make
feature {NONE}
    make
        do
            create model.make_empty
        end

feature -- model
    model: SEQ [G]

feature -- Queries
    is_equal (other: like Current): BOOLEAN
        do
            Result := model ~ other.model
        end

    count: INTEGER
            -- number of items in stack
        do
            Result := model.count
        ensure
            Result = model.count
        end
```

Immutable Queries

```
    top: G
            -- top of stack
        require
            not model.is_empty
        do
            Result := model[1]
        ensure
            Result ~ model.first
        end

feature -- Commands
    push (x: G)
            -- push `x' on to the stack
        do
            model.prepend (x)
        ensure
            pushed_othewise_unchanged:
                model ~ ((old model.deep_twin) |<- x)
        end

    pop
            -- pop top of stack
        require
            not model.is_empty
        do
            model.remove (1)
        ensure
            model ~ old model.deep_twin.tail
        end
end
```

```
feature -- model
    model: SEQ [G]
```

```
    top: G
            -- top of stack
        require
            not model.is_empty
        do
            Result := model[1]
        ensure
            Result ~ model.first
        end

feature -- Commands
    push (x: G)
            -- push `x' on to the stack
        do
            model.prepend (x)
        ensure
            pushed_othewise_unchanged:
                model ~ ((old model.deep_twin) |<- x)
        end

    pop
            -- pop top of stack
        require
            not model.is_empty
        do
            model.remove (1)
        ensure
            model ~ old model.deep_twin.tail
        end
end
```

```
feature -- model
    model: SEQ [G]
```

Command
prepend

Immutable Query
prepended

ADT vs. ASM

- *STACK* [*G*]

FUNCTIONS

- *put*: *STACK* [*G*] × *G* → *STACK* [*G*]
- *remove*: *STACK* [*G*] ↛ *STACK* [*G*]
- *item*: *STACK* [*G*] ↛ *G*
- *empty*: *STACK* [*G*] → *BOOLEAN*
- *new*: *STACK* [*G*]

AXIOMS

For any *x*: *G*, *s*: *STACK* [*G*]

A1 • *item* (*put* (*s*, *x*)) = *x*

A2 • *remove* (*put* (*s*, *x*)) = *s*

A3 • *empty* (*new*)

A4 • **not** *empty* (*put* (*s*, *x*))

PRECONDITIONS

- *remove* (*s*: *STACK* [*G*]) **require not** *empty* (*s*)
- *item* (*s*: *STACK* [*G*]) **require not** *empty* (*s*)

```
    top: G
            -- top of stack
        require
            not model.is_empty
        do
            Result := model[1]
        ensure
            Result ~ model.first
        end

feature -- Commands
    push (x: G)
            -- push `x' on to the stack
        do
            model.prepend (x)
        ensure
            pushed_othewise_unchanged:
                model ~ ((old model.deep_twin) |<- x)
        end

    pop
            -- pop top of stack
        require
            not model.is_empty
        do
            model.remove (1)
        ensure
            model ~ old model.deep_twin.tail
        end
end
```

For Descendants (inheritance = reuse)

```
note
    description: "[
        Infinite stack API,
        fully contracted using Mathmodels SEQ[G]
    ]"

class ABSTRACT_STACK [G -> attached ANY] inherit
    ANY
        redefine is_equal end
create
    make
feature {NONE}
    make
        do
            create imp.make_empty
        end

    imp: SEQ[G]

feature -- model

    model: SEQ [G]
        do
            Result := imp
        end
```

```
class
    MY_STACK [G -> attached ANY]
inherit

    ANY
        undefine is_equal end

    ABSTRACT_STACK [G]
        redefine
            count,
            make,
            model,
            top,
            push,
            pop
        end

create
    make
```

```
feature {NONE} -- creation

    implementation: LINKED_LIST [G]
            -- implementation of stack as array
```

```
feature -- model

    model: SEQ [G]
            -- abstraction function
        do
            create Result.make_empty
            from
                implementation.start
            until
                implementation.after
            loop
                Result.append (implementation.item)
                implementation.forth
            end
        end
```

```
feature {NONE} -- creation

    implementation: LINKED_LIST [G]
            -- implementation of stack as array
```

```
feature -- model

    model: SEQ [G]
            -- abstraction function
        do
            create Result.make_empty
            from
                implementation.start
            until
                implementation.after
            loop
                Result.append (implementation.item)
                implementation.forth
            end
        end
```

```
invariant
    same_count:
        model.count = implementation.count
    equality: across 1 |..| count as i all
        model [i.item] ~ implementation [i.item] end
    -- top of stack is model[1] and implementation[1])
end
```

```
pop
    -- pop top of stack
  require -- from ABSTRACT_STACK
    not model.is_empty
  do
    implementation.start
    implementation.remove
  ensure -- from ABSTRACT_STACK
    model ~ old model.deep_twin.tail
  end
```

```
push (x: G)
  -- push x on to the stack
  do
    implementation.put_front (x)
  ensure -- from ABSTRACT_STACK
    pushed_othewise_unchanged:
      model ~ ((old   model.deep_twin) |<- x)
end
```

Readings

- OOSC2 Chapter 11
- For introductory material, see Touch of Class (available online via Steacie)
  - For loop variants and invariants see Chapter 7.5

What's wrong with `break'

| | Subject: AT& T Bug
| | Date: Fri Jan 19 12: 18: 33 1990
| |
| | This is the bug that caused the AT& T breakdown the other day:
| | In the switching software (written in C),
| | there was a long do . . . while construct,
| | which contained a switch statement,
| | which contained an if clause,
| | which contained a break,
| | which was intended for the if clause,
| | but instead broke from the switch.

- A missing "break" statement brought down the entire long-distance telephone network in the North Eastern US, resulting in millions of dollars of damage!

Breaks/Go-Tos

- 1970, Edsger Dijkstra: don't use control structures other than one-entry/one-exit blocks ("structured programming").
- With one-entry/one-exit, you can read/reason about the code sequentially
  - You don't have to turn yourself into a computer and apply operational reasoning: "if when executing this instruction I'm coming from here it will satisfy this property, but if I am coming from there it will satisfy that other property, so in both cases ... it will be true that ... maybe ... wait a minute... etc." Humans are not computers!

Breaks/Go-Tos

- Understanding a loop:
  - loop invariant: property ensured by initialization and maintained by every iteration. Combined with the loop exit condition it allows the program reader to check, through a simple inspection of the loop text, that the loop is doing its job.
  - loop variant: reason about termination by telling us which decreasing quantity guarantees that the loop will terminate.

Structured Programming

- The key point is to be able to reason about a program statically, i.e. by associating properties with its text as given, rather than operationally, i.e. by trying to mimic its execution within your head.
- Computers are there to execute programs; humans can't really do it effectively for any non-trivial examples.
- The one-entry, one-exit structure helps us do something at which we are much better: static reasoning based on the simple rules of logic.

Use Check Statements as well

- To know if all the strings in my_list have lengths greater than three characters, we could code:

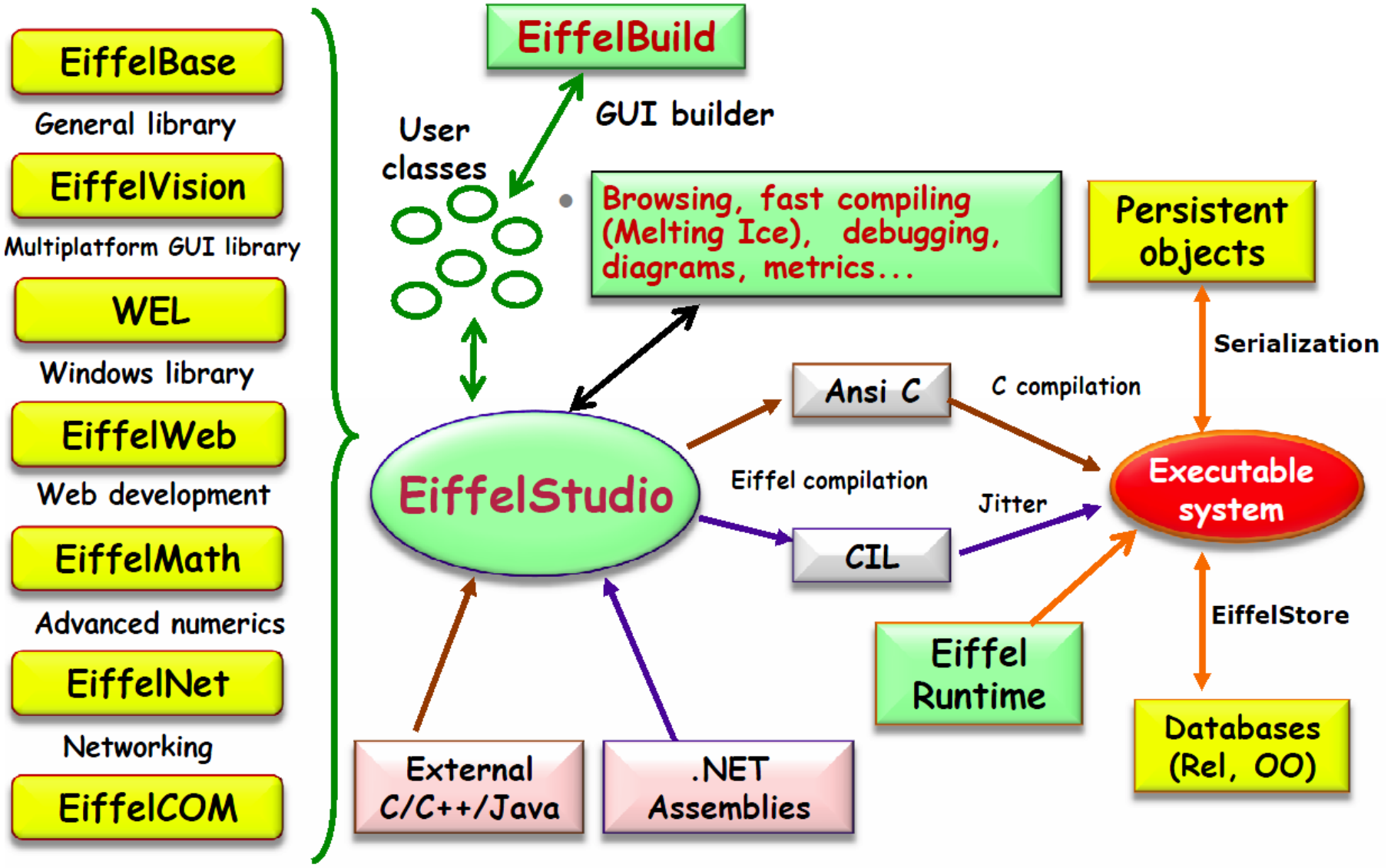
```
my_list: LIST[STRING]
check
    across my_list as ic all ic.item.count > 3 end
end
```

- Class LIST inherits from ITERABLE

- From: Thomas Beale (Chief Technology Officer, Ocean Informatics)
- Date: Monday, May 04, 2009
- To: eiffel_software@yahoogroups.com

- I had also meant to say that another strength of Eiffel is that its semantics are very close to the theoretically ideal OO meta-model, so you don't have to continually perform workarounds when formalism any design concept - all the basics are there - nested generics (that work; by contrast, see the horror that is UML generics);
  - multiple inheritance (that works, no need for 'aspect-oriented programming');
  - agents (a real source of power when performing recursive processing and event-handling), DbC and other features (let's not forget basics like covariant redefinition, uniform reference etc).
  - There are some things I would really like, that we do have to work around: the ability to distinguish between an association reference to another object and an aggregation reference (i.e. uses vs. part-of), and to mark which attributes should be persistable. But just try doing any clear design thinking in Java, and you soon see the difference.

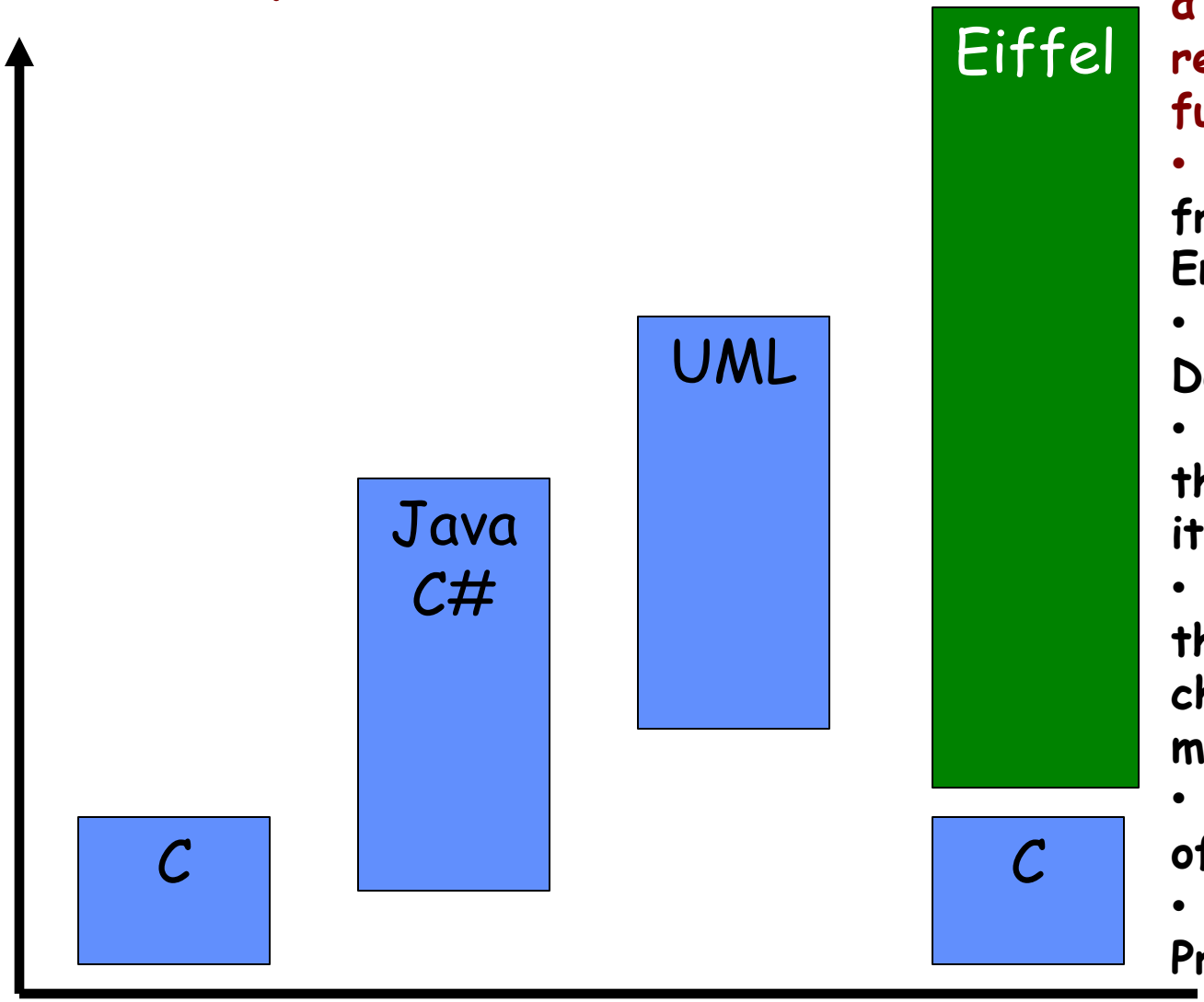
- In the old days, 'design' was a largely non-computational exercise, sometimes using UML or other graphical tools to help understand things;
- today (and for at least 10 years) it has been possible to do serious 'design' using less documentary effort, and more formal tools.
- For me, this is in fact why I can't leave Eiffel - it is a killer design lab. Our little application built for openEHR (ADL workbench, a kind of compiler tool) is what enabled me to do the design of the 'archetype' language ADL (now an ISO standard 13606-2). While the design is not perfect, it is pretty coherent, and the defects are details rather than theoretical problems.


EiffelBase
General library
EiffelVision
Multiplatform GUI library
WEL
Windows library
EiffelWeb
Web development
EiffelMath
Advanced numerics
EiffelNet
Networking
EiffelCOM
.....
EiffelBuild
GUI builder
User classes
Browsing, fast compiling (Melting Ice), debugging, diagrams, metrics...
Persistent objects
Serialization
Ansi C
C compilation
EiffelStudio
Eiffel compilation
Jitter
CIL
Executable system
EiffelStore
Eiffel Runtime
Databases (Rel, OO)
External C/C++/Java
.NET Assemblies

Design Notation

- Abstraction
- Information hiding
- Seamlessness
- Reversibility
- Design by Contract
- Open-Closed principle
- Single choice principle
- Single model principle
- Uniform access principle
- Command-query separation principle
- Option-operand separation principle
- Style matters

Abstract Expressive Power



Modelling = a mental representation of a future system that

- includes constraints from a possibly hostile Environment
- describes the Design Decisions/Patterns
- we can reason about the model and predict its behaviour
- The correctness of the program can be checked against the model
- it is included as part of the program text
- Single Model Principle

Take home lesson from EECS3311

- **The 3311-Ostroff Rule**
  - *Always work one level of abstraction higher than where you started*