

LASSONDE

Common Eiffel Errors: Contracts vs. Implementations



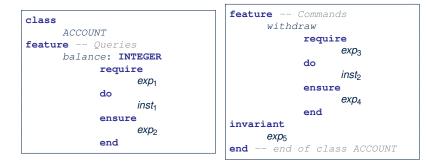
EECS3311 A: Software Design Winter 2020

CHEN-WEI WANG

• Instructions for Implementations: inst₁, inst₂

Contracts vs. Implementations: Where?

• Boolean expressions for Contracts: exp₁, exp₂, exp₃, exp₄, exp₅



3 of 23

Contracts vs. Implementations: Definitions

In Eiffel, there are two categories of constructs:

• Implementations

• are step-by-step instructions that have side-effects



- change attribute values
- do not return values
- ≈ commands
- Contracts
 - are Boolean expressions that have no side-effects

e.g., ... = ..., across ... as ... all ... end

- use attribute and parameter values to specify a condition
- return a Boolean value (i.e., True or False)
- ~ queries

2 of 23

Implementations: Instructions with No Return Values

• Assignments

balance := balance + a

• Selections with branching instructions:

if a > 0 then acc.deposit (a) else acc.withdraw (-a) end

Loops





Contracts:

Expressions with Boolean Return Values

• Relational Expressions (using =, /=, ~, /~, >, <, >=, <=)

a > 0

Binary Logical Expressions (using and, and then, or, or else, implies)

(a.lower <= index) **and** (index <= a.upper)

• Logical Quantification Expressions (using **all**, **some**)

```
across
  a.lower |..| a.upper as cursor
all
  a [cursor.item] >= 0
end
```

• **old** keyword can only appear in postconditions (i.e., **ensure**).

```
balance = old balance + a
```





LASSONDE

class	
ACCOUNT	
feature	
withdraw (a:	INTEGER)
do	
ensure	
balance =	old balance - a
end	



8 of 23

Contracts: Common Mistake (1)



class ACCOUNT feature withdraw (a: INTEGER) do	
ensure balance := old balance - a end 	

Colon-Equal sign (:=) is used to write assignment instructions.

Contracts: Common Mistake (2)

ass ACCOUNT
ature
withdraw (a: INTEGER)
do
ensure
across
a as cursor
loop
end

across...loop...end is used to create loop instructions.

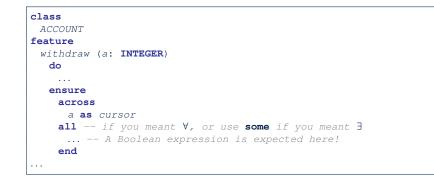
Contracts: Common Mistake (2) Fixed



LASSONDE

Contracts: Common Mistake (3) Fixed





class	
ACCOUNT	
feature	
withdraw (a: INTEGER)	
do	
ensure	
<pre>postcond_1: balance = old balance - a</pre>	
<pre>postcond_2: old balance > 0</pre>	
end	

9 of 23





class ACCOUNT	
feature	
withdraw (a: INTEGER)	
do	
ensure	
old balance - a	
end	

Contracts can only be specified as Boolean expressions.



- Only postconditions may use the old keyword to specify the relationship between pre-state values (before the execution of withdraw) and post-state values (after the execution of withdraw).
- *Pre-state values* (right before the feature is executed) are

Contracts: Common Mistake (4) Fixed



LASSONDE

Lass ACCOUNT
eature
withdraw (a: INTEGER)
require
balance > 0
do
ensure
end

13 of 23

Contracts: Common Mistake (5) Fixed

class linear container
create make
feature Attributes
a: ARRAY[STRING]
feature Queries
count: INTEGER do Result := a.count end
get (i: INTEGER): STRING do Result := a[i] end
feature Commands
make do create a.make_empty end
update (i: INTEGER; v: STRING)
do
ensure Others Unchanged
across
1 count as j
all
j.item /= i implies (old Current).get(j.item) ~ get(j.item)
end
end
end

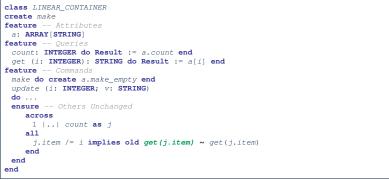
LASSONDE

LASSONDE

- The idea is that the **old** expression should not involve the local cursor variable j that is introduced in the postcondition.
- Whether to put (old *Current.twin*) or (old *Current.deep_twin*) is up to your need.

15 of 23

Contracts: Common Mistake (5)



Compilation Error

- Expression value to be cached before executing update?
 - [Current.get(j.item)]
- But, in the *pre-state*, integer cursor j does not exist!

Implementations: Common Mistake (1)



- Equal sign (=) is used to write Boolean expressions.
- In the context of implementations, Boolean expression values must appear:
 - on the RHS of an *assignment*;
 - as one of the branching conditions of an if-then-else statement; or
 - as the *exit condition* of a loop instruction.

Implementations: Common Mistake (1) Fixed

class	
ACCOUNT	
feature	
withdraw (a: INTEGER)	
do	
balance := balance + 1	
end	

Implementations: Common Mistake (2) Fixed

1	class
2	BANK
3	feature
4	min_credit: REAL
5	accounts: LIST[ACCOUNT]
6	
7	no_warning_accounts: BOOLEAN
8	do
9	Result :=
10	across
11	accounts as cursor
12	all
13	cursor.item.balance > min_credit
14	end
15	end
16	

Rewrite L10 – L14 using across ... as ... some ... end.

17 of 23

Implementations: Common Mistake (2)

class
BANK
feature
min_credit: REAL
accounts: LIST[ACCOUNT]
no_warning_accounts: BOOLEAN
do
across
accounts as cursor
all
cursor.item.balance > min_credit
end
end

Again, in implementations, Boolean expressions cannot appear alone without their values being "captured".

Implementations: Common Mistake (3)

Hint: $\forall x \bullet P(x) \equiv \neg(\exists x \bullet \neg P(x))$

19 of 23

LASSONDE

class
BANK
feature
accounts: LIST[ACCOUNT]
total_balance: REAL
do
Result :=
across
accounts as cursor
loop
Result := Result + cursor.item.balance
end
end

LASSONDE

In implementations, since instructions do not return values, they cannot be used on the RHS of assignments.

Implementations: Common Mistake (3) Fixed



Contracts: Common Mistake (5) Fixed

Implementations: Common Mistake (1)

Implementations: Common Mistake (1) Fixed

LASSONDE

Implementations: Common Mistake (2)

Implementations: Common Mistake (2) Fixed

Implementations: Common Mistake (3)

Implementations: Common Mistake (3) Fixed

23 of 23

Index (1)	
Contracts vs. Implementations: Definitions	
Contracts vs. Implementations: Where?	
Implementations:	
Instructions with No Return Values	-
Contracts:	
Expressions with Boolean Return Values	
Contracts: Common Mistake (1)	
Contracts: Common Mistake (1) Fixed	
Contracts: Common Mistake (2)	
Contracts: Common Mistake (2) Fixed	
Contracts: Common Mistake (3)	
Contracts: Common Mistake (3) Fixed	
Contracts: Common Mistake (4)	
Contracts: Common Mistake (4) Fixed	
Contracts: Common Mistake (5)	