EECS3311 Software Design (Fall 2020)

Q&A - Lecture Series W2

Monday, September 21

Q: How to distinguish between which relation/arrow is client-supplier and which relation/arrow is inheritance?

Q: Does it purely depend on our design?

Software Architecture: Client Supplier vs. Inheritance

An architectural design diagram:
- Represents an abstraction of your implementation code
- Facilitates communication with co-workers or clients

version of code with details hidden

Code

```
class
  SAVINGS_ACCOUNT
inherit
  ACCOUNT
...
  owner: CLIENT
  ts: ARRAY[TRANSACTION]
end
```

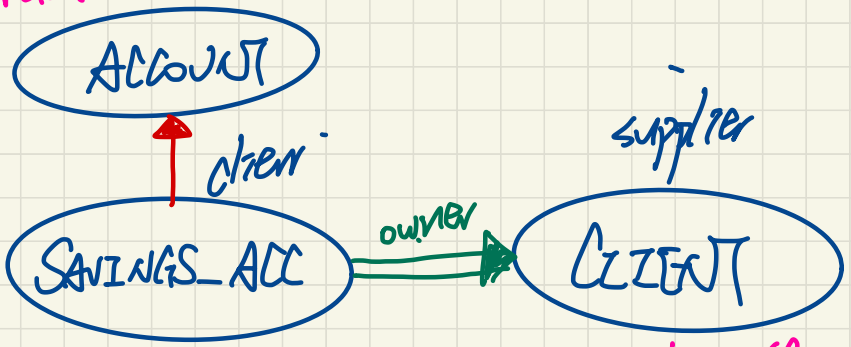
declared a CS relation

supplier

supplier

supplier

Diagram



owner.append("...")

make use of CS relation

Q. For the shallow copy, what happens
if the imp is an array storing primitive type?

Q. Lets say imp:ARRAY[INT], I can image from the graph there is no arrow pointing to the address because it is primitive type and directly store into the array (am I right?)

Q. In this case, if we imp[2] := some different integer, will imp[2] ~ old_imp[2] return True?

Collection Objects: Shallow Copy & Make 1st-Level Changes

imp, old_imp: ARRAY[INTEGER]

```
1  old_imp := imp.twin
2  Result := old_imp ~ imp   -- Result = [redacted]
3  imp[2] := [redacted]
4  Result :=
5    across 1 |..| imp.count is j
6    all imp [j] ~ old_imp [j]
7  end -- Result = [redacted]
```

primitive.

old_imp[1] := imp[1]

i, j: INT

i = j

i ~ j

imp[1] ~ old_imp[1] T

imp[2] ~ old_imp[2] F.

F.

F.

imp → ARRAY[INT]

1	2	3
46	~~89~~	75

23

old_imp → A[1]

46	89	75

tmp, old_tmp: A[S]

Q. For the deep copies in the Eiffel code example, after the initialization of `old_imp` and right before `imp[2]` is changed, how come `imp[1] ~ old_imp[1]`, `imp[2] ~ old_imp[2]`, and `imp[3] ~ old_imp[3]` are all true?

① tmp ~ old_tmp
from ARRAY is_equal.

② is_equal STRING

imp[1] ~ old_imp[1]	True
imp[2] ~ old_imp[2]	True
imp[3] ~ old_imp[3]	True

Q. The object_comparison is false for the arrays, so that should mean we're comparing addresses. And since we made a deep copy, all the objects should be distinct.

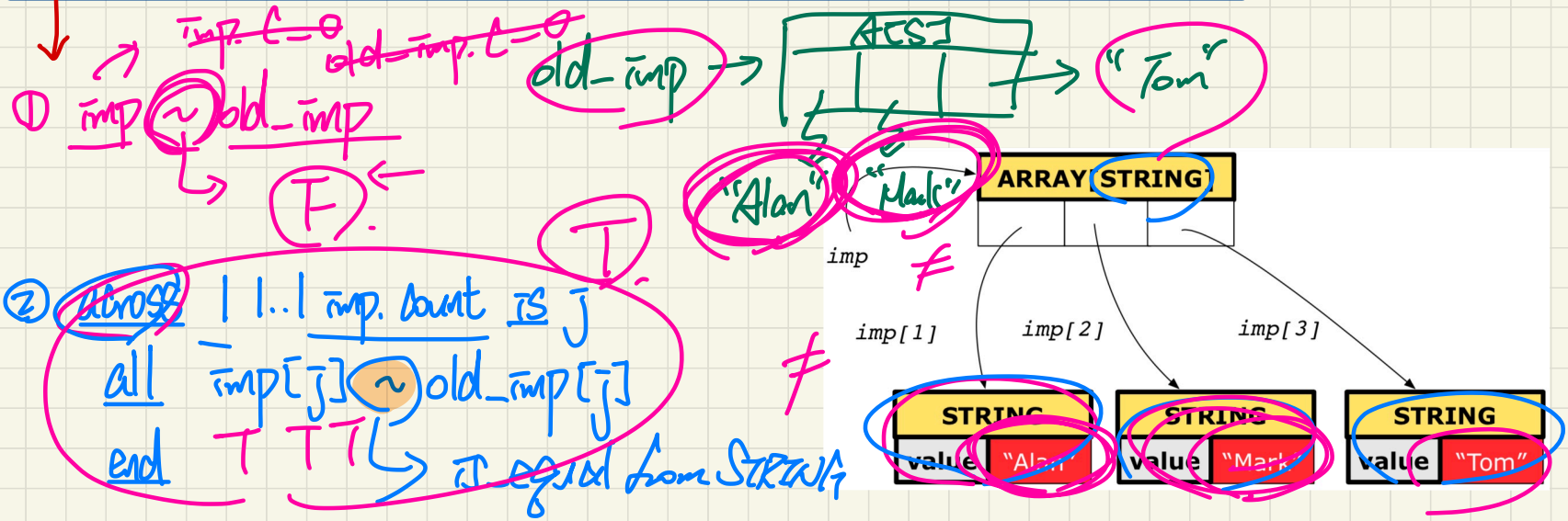
Q. Is STRING treated like a primitive type in Eiffel?

Reference type

↳ No.

Collection Objects: Deep Copy & Make 2nd-Level Changes

```
1  old_imp := imp.deep_twin
2  Result := old_imp = imp   -- Result = [redacted]
3  imp[2].append ("***")
4  Result :=
5    across 1 |..| imp.count is j
6    all imp [j] ~ old_imp [j] end   -- Result = [redacted]
```



s1 := "alan"

s2 := "mary"

s1 := "sarah"

s1 ~ s2 F

s1.count = s2.count F ?

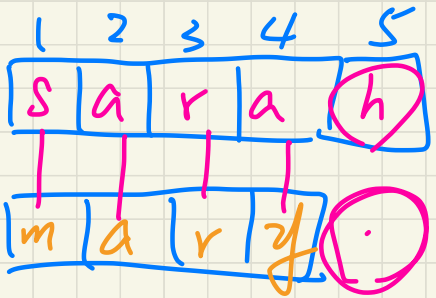
and then

across 1 |..| s1.count is i

all

s1.item(i) = s2[i]

end



s1[5] = s2[5]

h

Short circuit!

b1 b2

① F

② T

Reference Type objects.

① obj1 = obj2 address

② obj1 ~ obj2

↳ obj1.is_equal(obj2)

Primitive Type variables value

① v1 = v2 value

② v1 ~ v2

↳ v1 = v2.

Q. When I debug this, I knew the value of `j` would be 1, 2, and then 3. But it doesn't show the value of j. Even though I am kind of sure of the value, I do want to see it to make sure my code is correct.

Q. Also, when I added `imp [1]`, I could see "Alan", but when I tried `imp[j]` (j had the value of 1), there was an "error occurred" in the `Value` column. How can I see the value of `imp[j]`.

Q. How can I see the overall T/F value of the `across...` statement in the post condition? Should I copy the whole thing to the `Expression` when do the debugging?

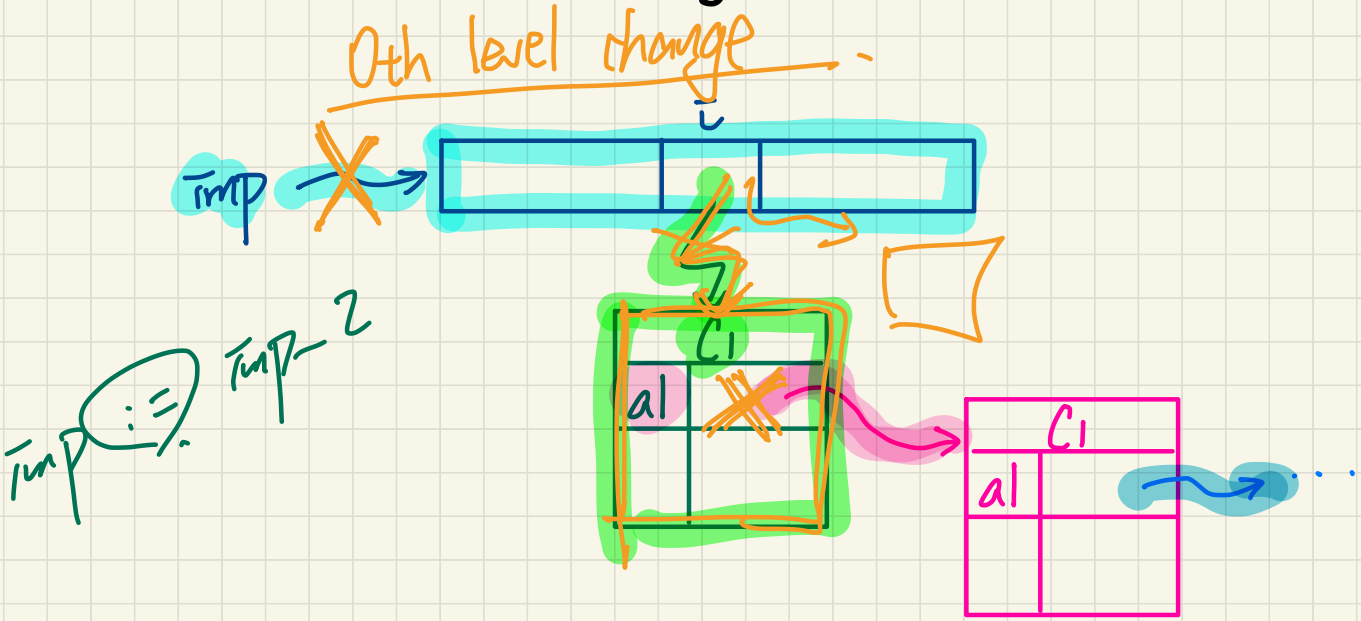
EStudio does not supprt debugging of across construct properly.

Try:

1. Inspec values in the Expressions panel.
2. Encode a from-until loop.

```
Result :=
  across 1 |..| imp.count is j
  all imp [j] ~ old_imp [j] end
```

Q. How to define the first level and 2nd level and so on? If we change the object address of `imp`, how do we name the change?

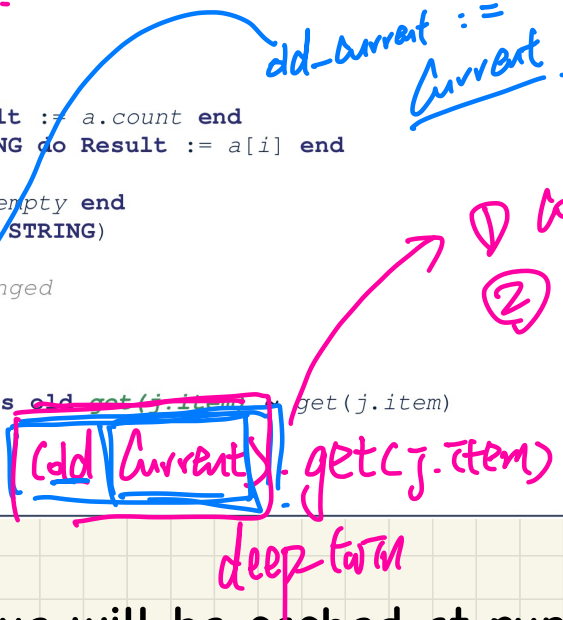


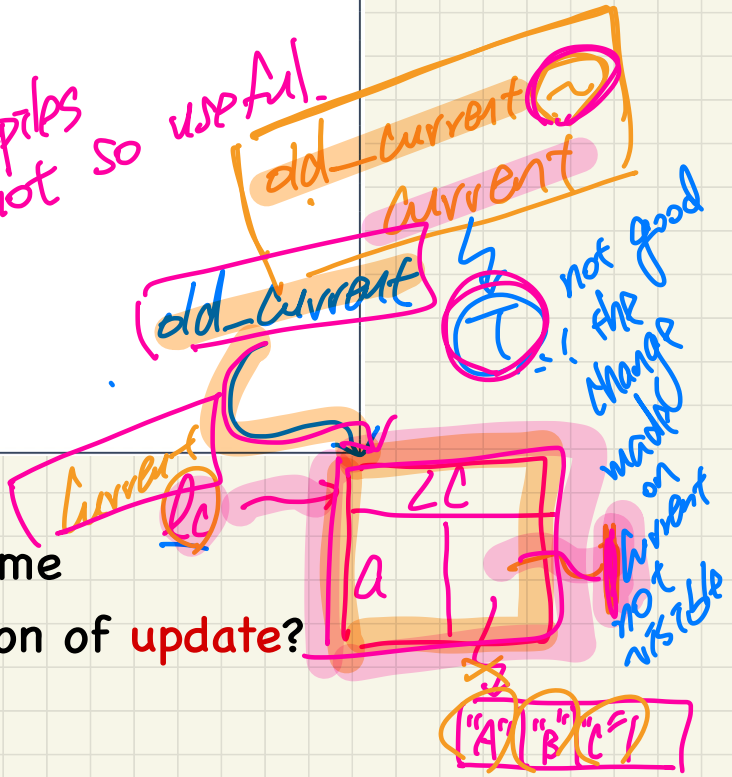
Q. For a possible fix, can we just
`(old Current).get(j.item)`
and not use deep_copy?

Q. Wouldn't the old keyword cache all the previous data before the actual implementation gets executed?

Use of old in across Expression in Postcondition

```
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 get(j.item) ~ get(j.item)
      end
  end
end
```





Hint: What value will be cached at runtime

before executing the implementation of update?

```
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 get(j.item) ~ get(j.item)
    end
  end
end
```

exp. to be cached in pre-state.

old_g_j_item := get(j.item)

Q. Why After the feature call, invariant is evaluated before the post-condition? Based on "Runtime Assertion Checking for Contracts - General Case" and your lectures, the post-condition is checked first. Also in Java, which should we check first?

Typo. Slide Updated.