Lecture 5

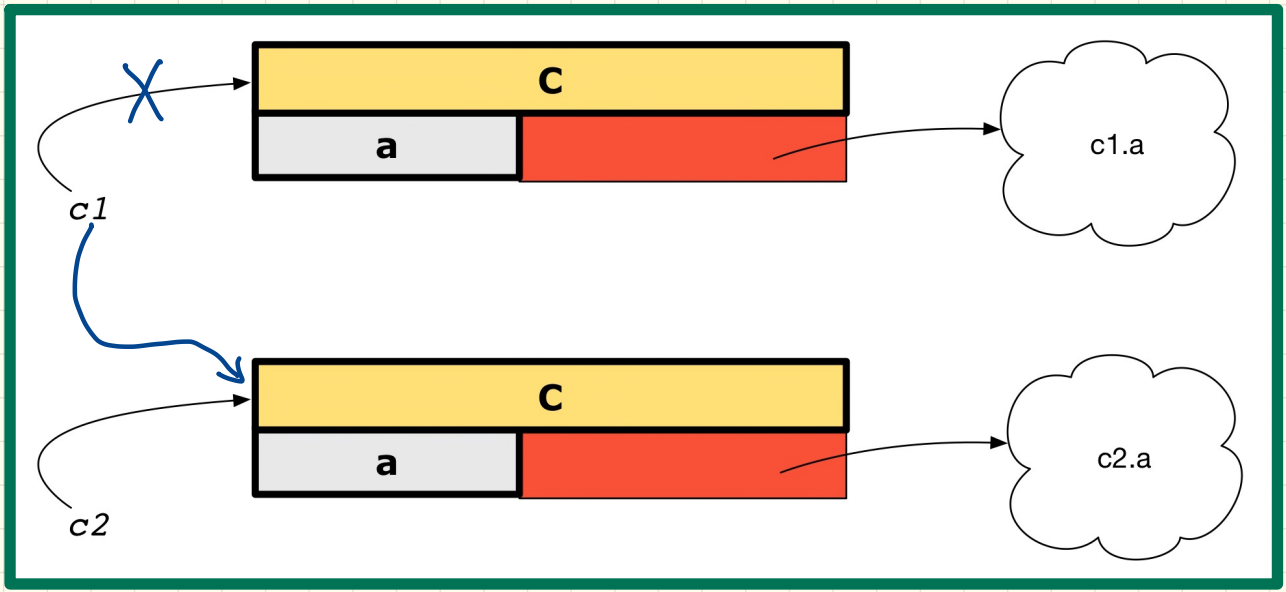
Monday January 20

In-Lab Demo last Friday:

- across
- comparator and sorter

Breakpoints and Debugger: See tutorial video

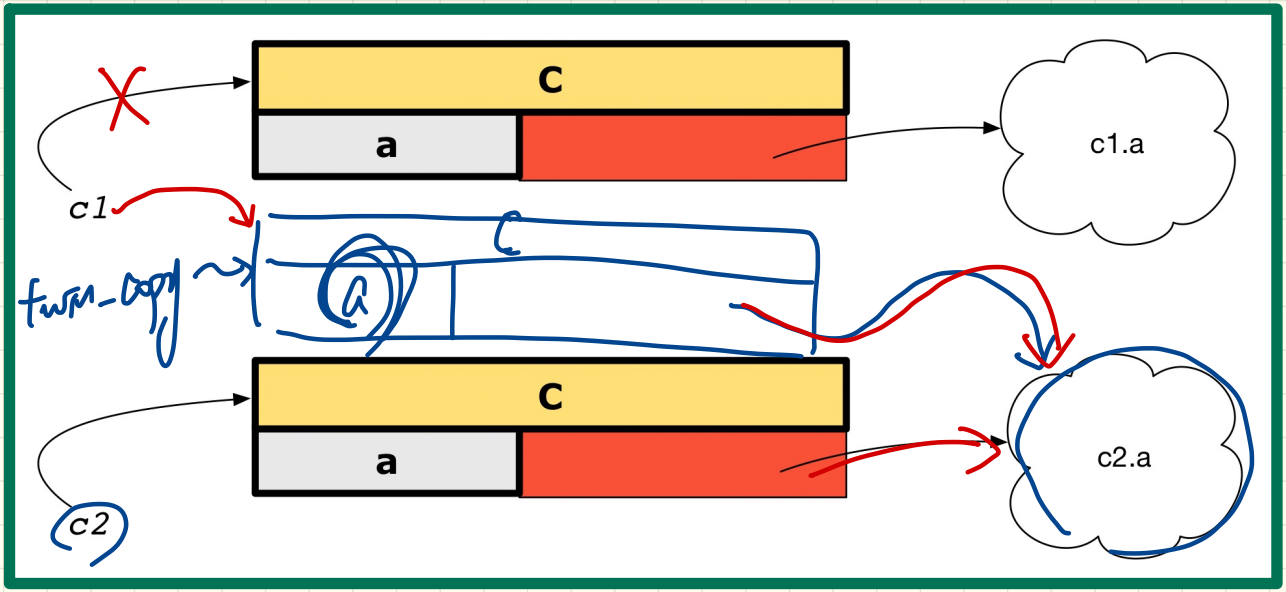
Reference Copy: c1 := c2



Shallow Copy: c1 := c2.twin

c2.a = c2.twin.a (T)

↳ c2 = c2.twin F

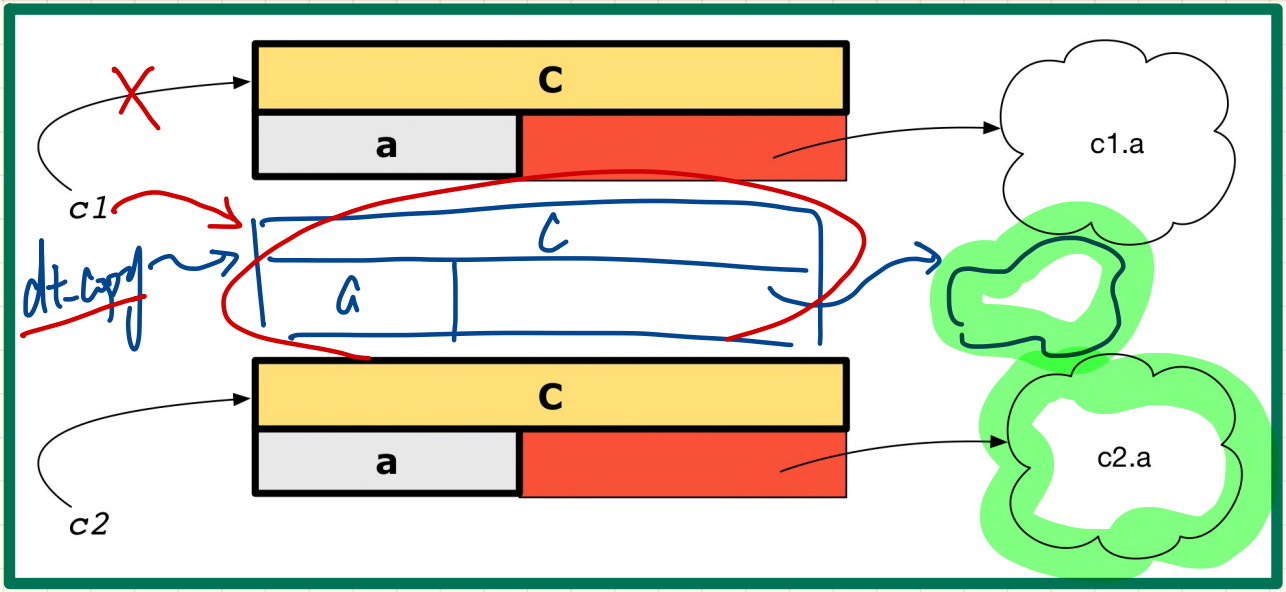


twin_copy.a := c2.a

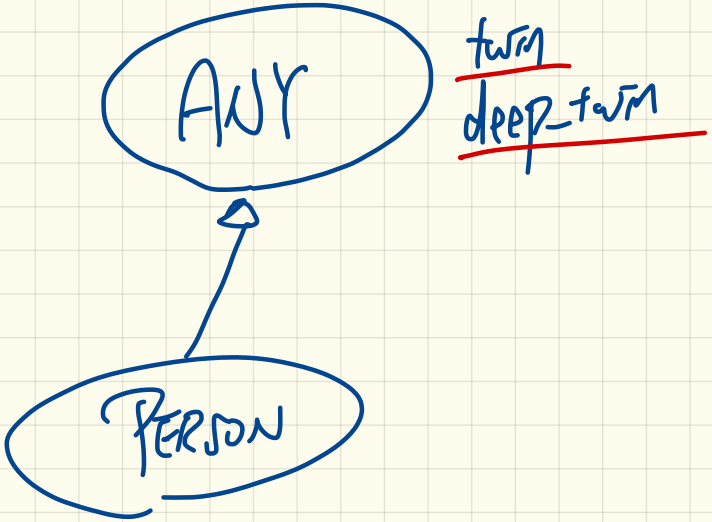
Deep Copy: c1 := c2.deep_twin

c2 = c2.dt F

c2.a = c2.dt.a T

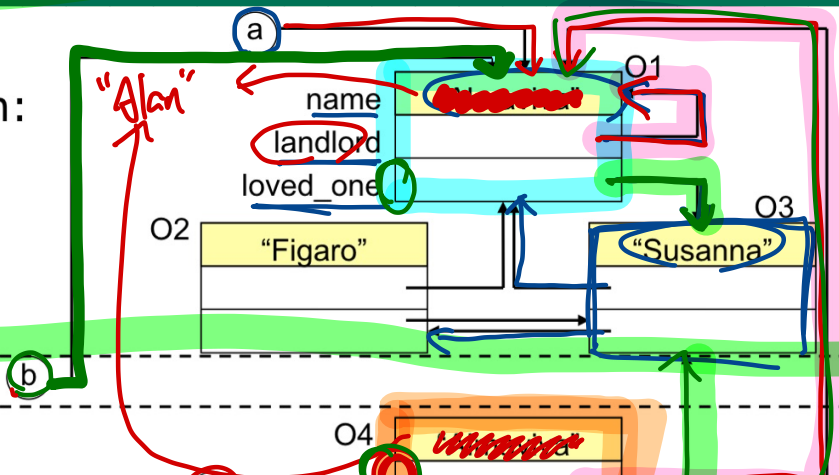


dt_copy.a := c2.a.deep_twin c1 := c2.deep_twin


ANY
PERSON
form
deep_form

Reference vs. Shallow vs. Deep Copies

- Initial situation:



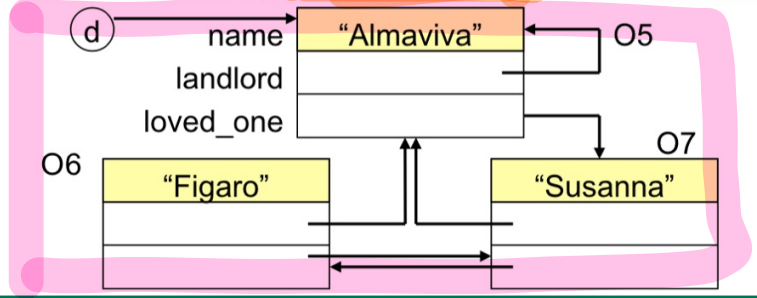
- Result of:

b := a

c := a.twin

c.landlord := a.landlord

d := a.deep_twin



c.landlord = a (T)

c.landlord = c (F)

f

do

⋮

ensure

old a

old a.twin

old a.deep_twin

triggers different kinds of copies in pre-state

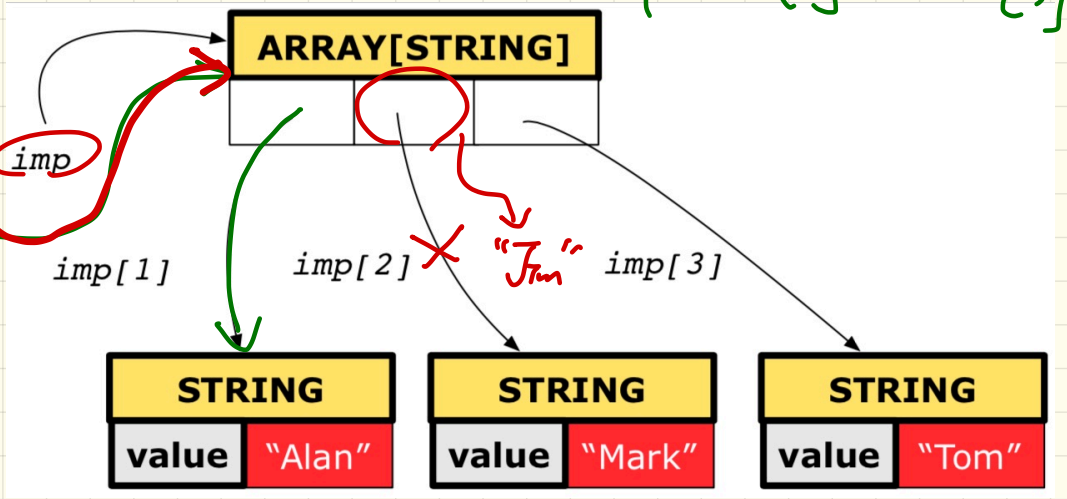
end

Collection Objects: Reference Copy & Make Changes

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

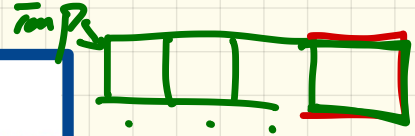
T
T
T

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

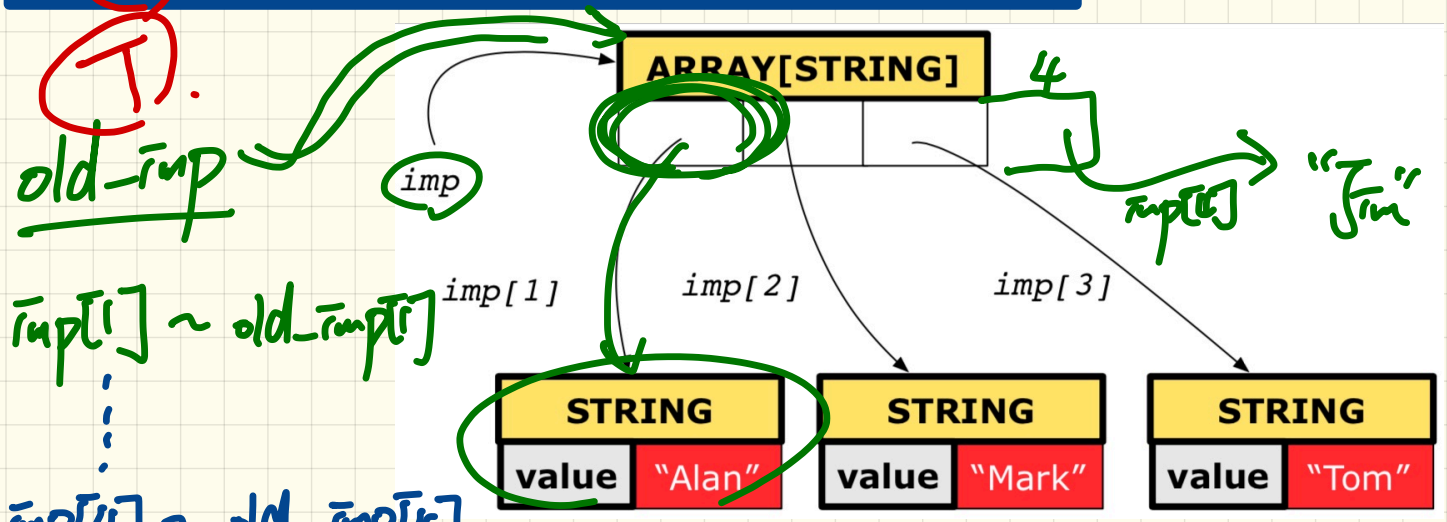


imp[2] ~ old_imp[2]

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



imp.force("Jim", imp.count + 1)



old_imp

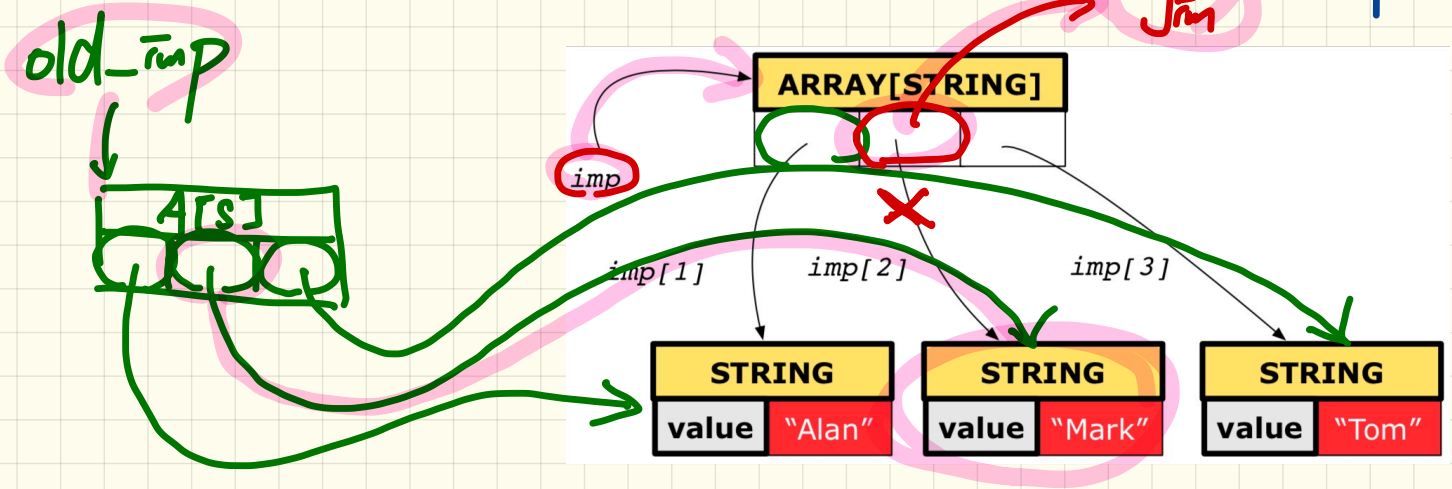
imp[1] ~ old_imp[1]

imp[4] ~ old_imp[4]

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

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

imp[1] ~ old_imp[1]
imp[2] ~ old_imp[2]

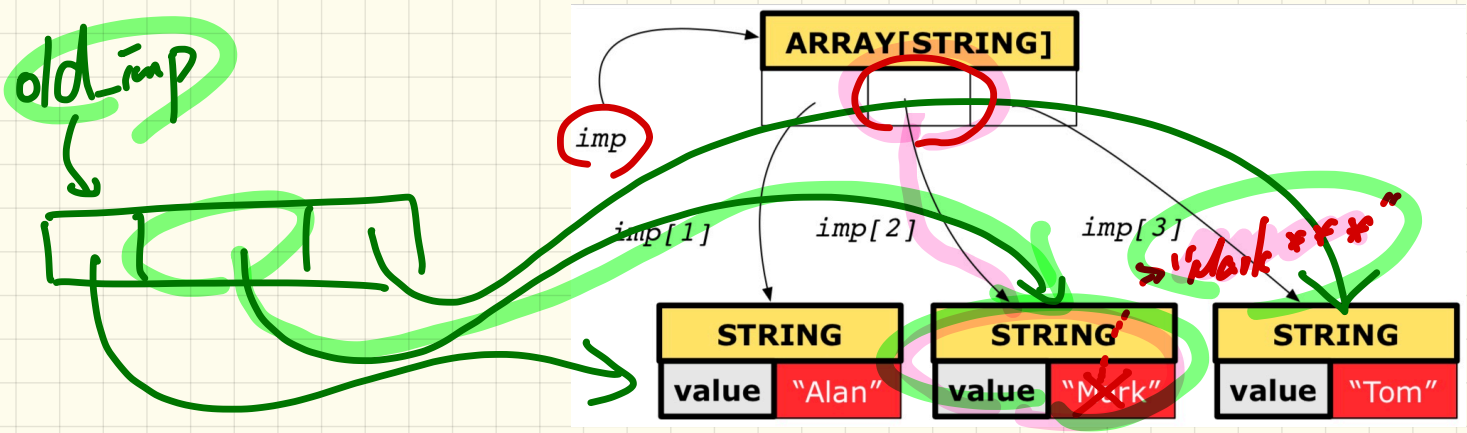


Collection Objects: Shallow Copy & Make 2nd-Level Changes

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

imp[1] ~ old_imp[1]

imp[2] ~ old_imp[2]



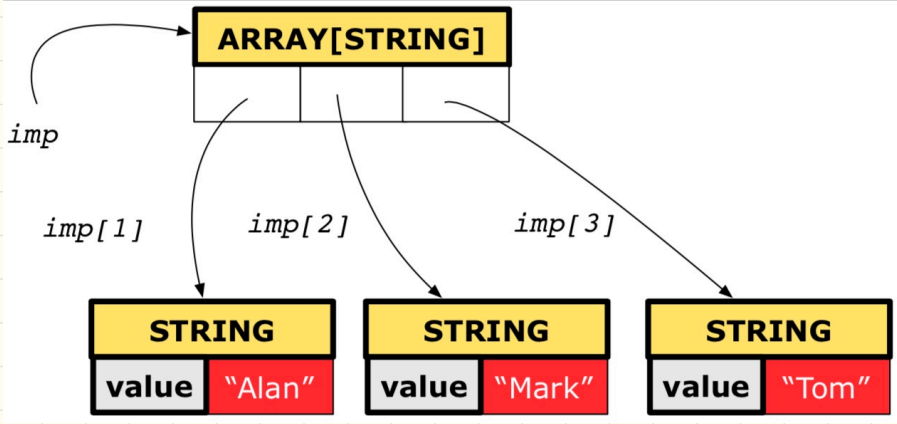
```
1  old_imp := imp.twin
2  Result := old_imp = imp   -- Result = false
3  imp.force ("Jim", imp.count + 1)
4  Result :=
5   across 1 |..| imp.count is j
6   all imp [j] ~ old_imp [j]
7   end -- Result = true
```

T ?

F ?

? ?

Execute

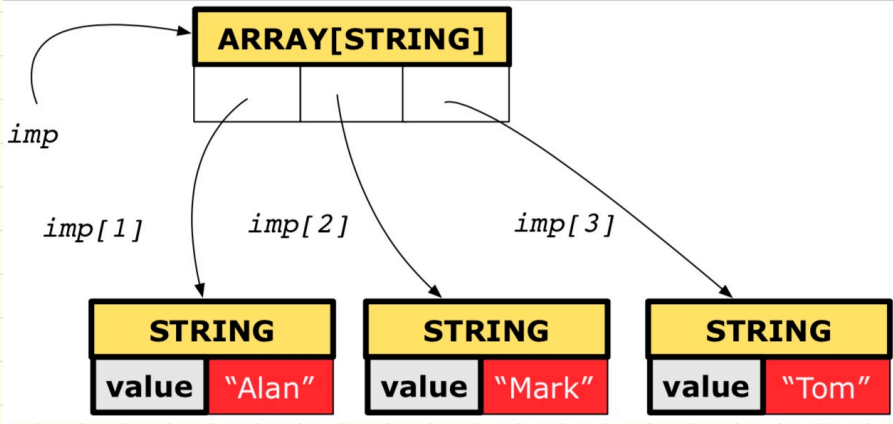


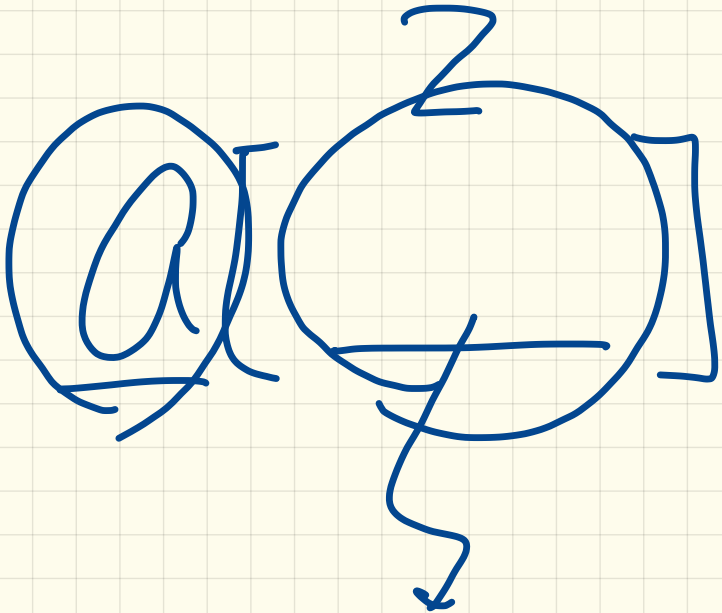
```
1  old_imp := imp.twin
2  Result := old_imp = imp   -- Result = false
3  imp[2].append ("**")
4  Result :=
5    across 1 |..| imp.count is j
6    all imp [j] ~ old_imp [j]
7    end -- Result = true
```

imp.force("Jim", 2)

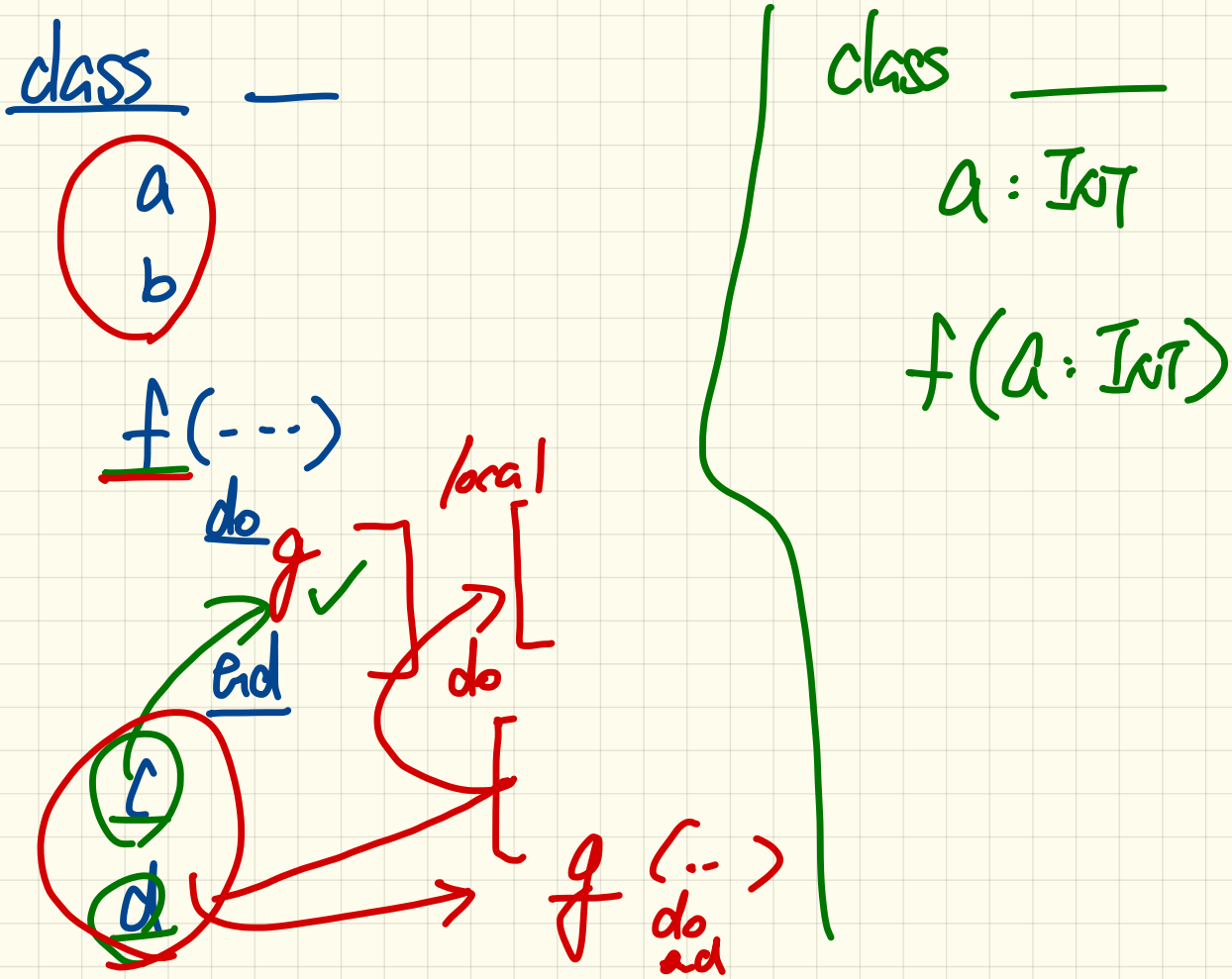
imp[2] := "Jim"

Exercise



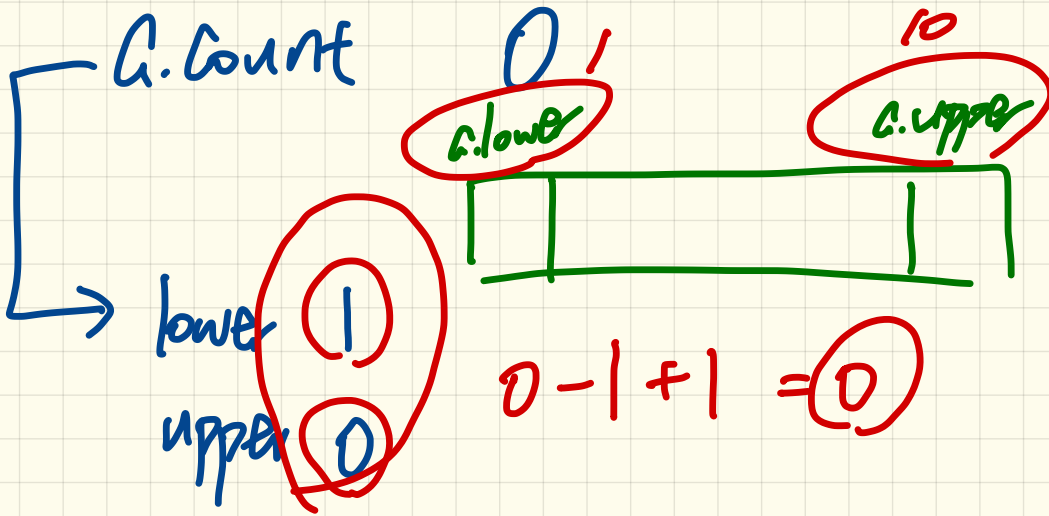


Invalid index is a runtime error, not Compile time.


class
a
b
f(...)
do
g
end
local
do
c
d
g(...)
do
end
class
a : INT
f(a : INT)

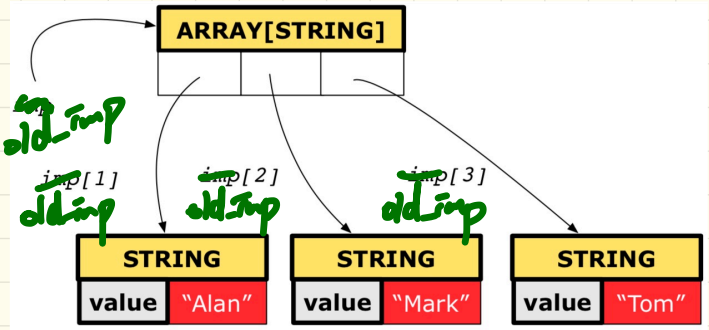
a.count = a.upper - a.lower + 1

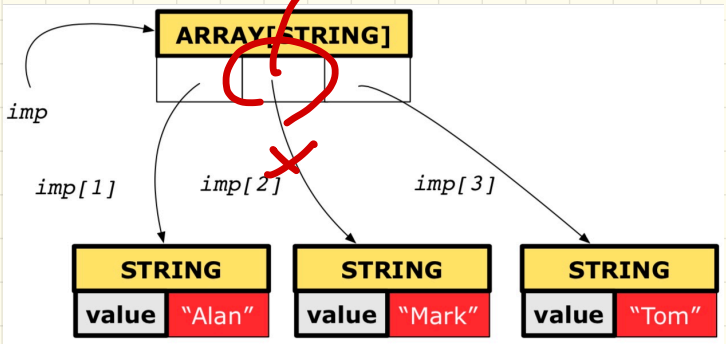
create a.make_empty



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

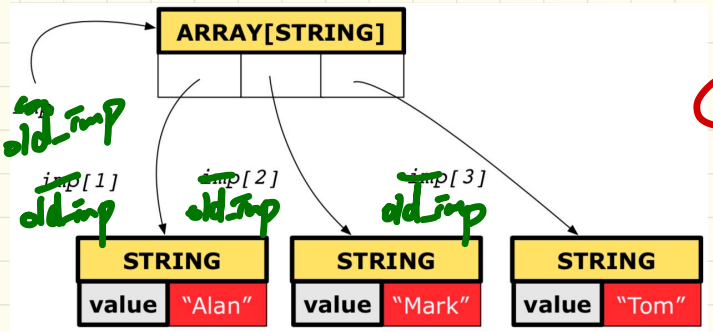
```
1  old_imp := imp.deep_twin
2  Result := old_imp = imp   -- Result = [redacted]
3  imp[2] := "Jim"
4  Result :=
5    across 1 |..| imp.count is j
6    all imp [j] ~ old_imp [j] end -- Result = [redacted]
```

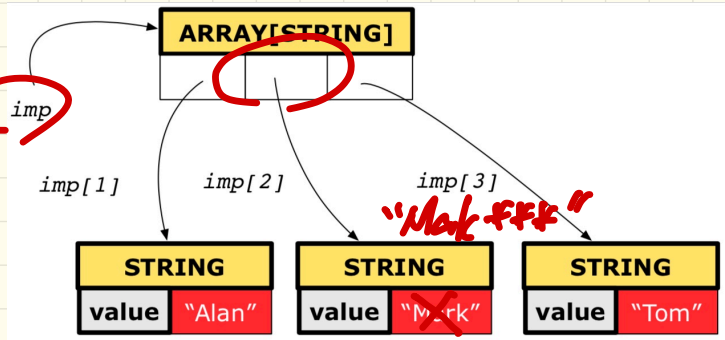


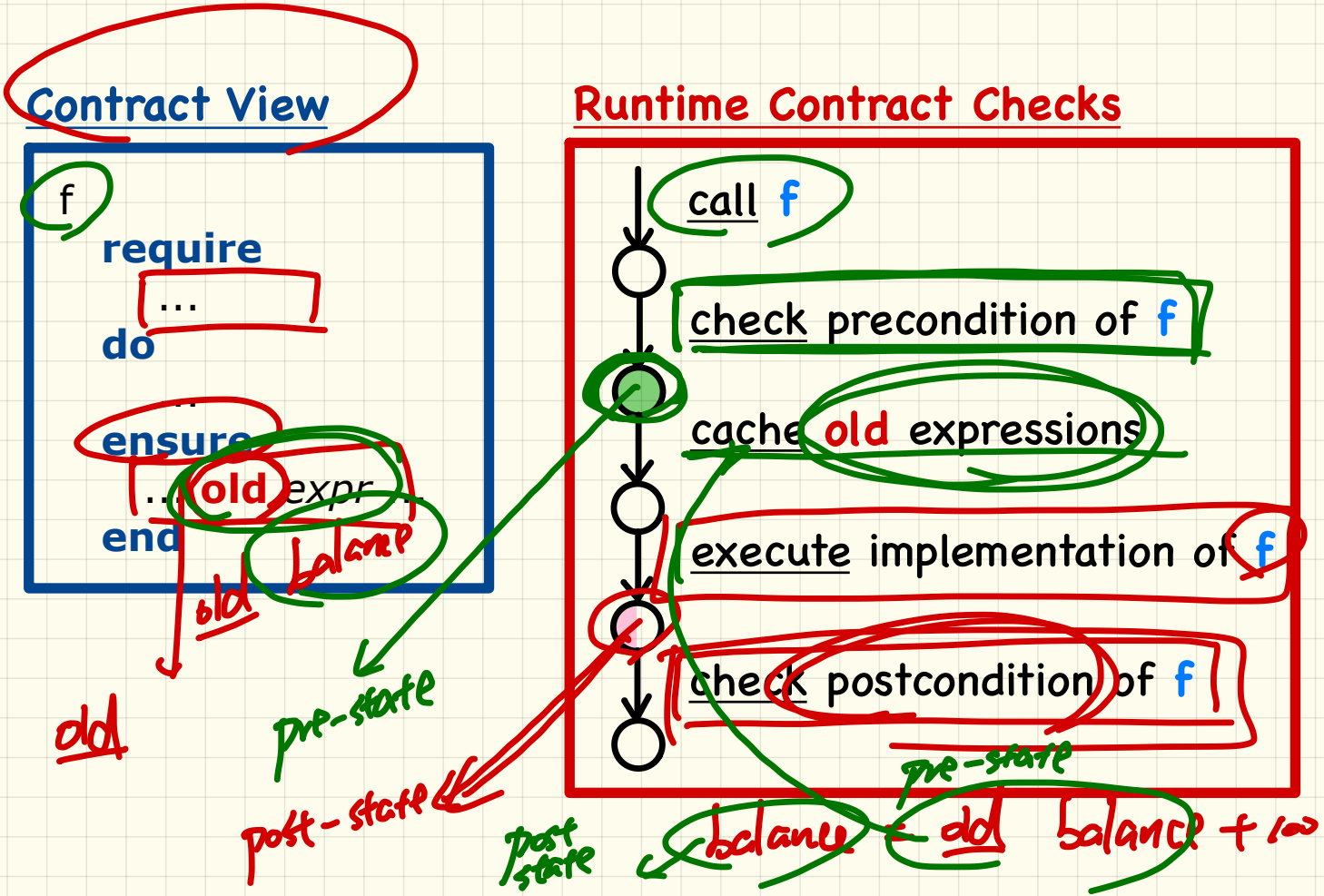


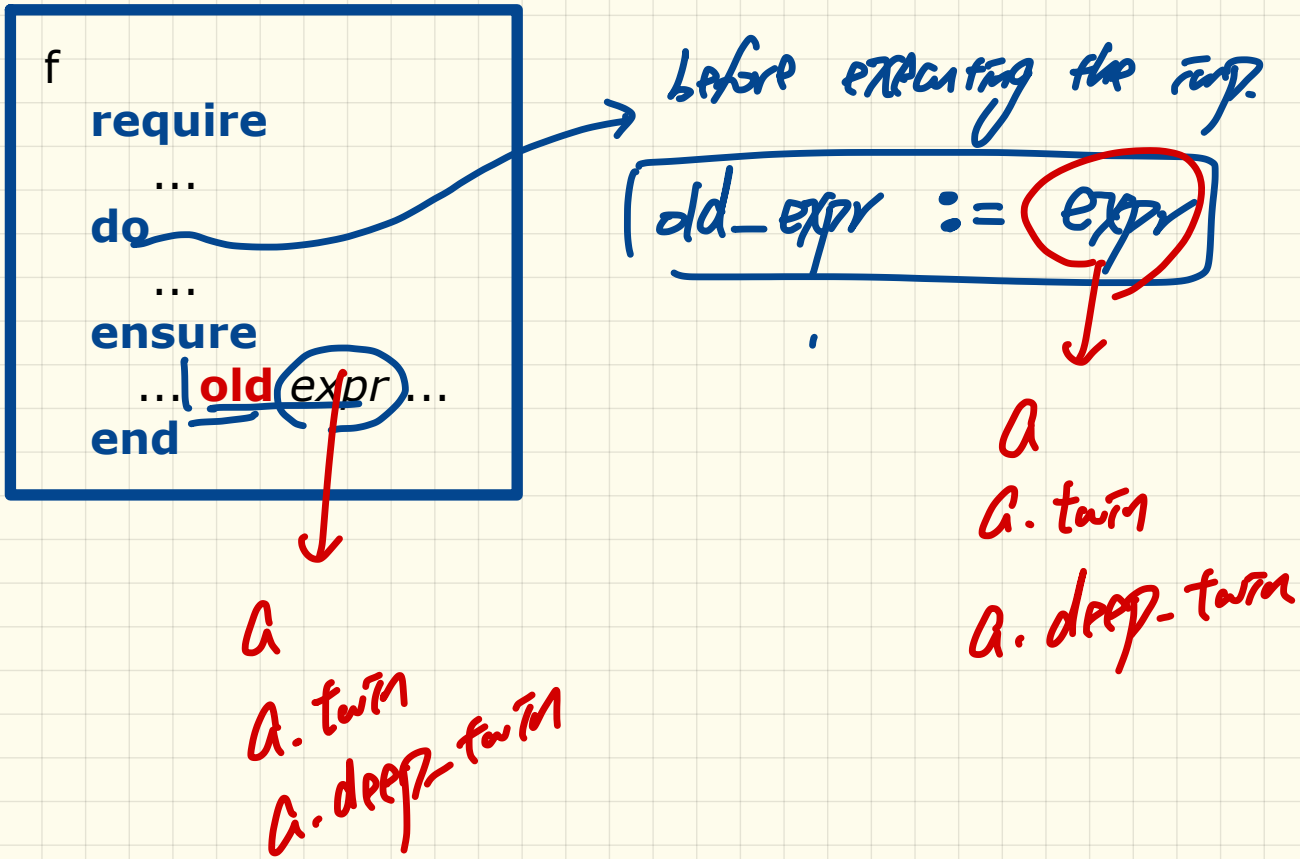
Collection Objects: Deep Copy & Make 2nd-Level Changes

```
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]
```






Contract View
f
require
...
do
...
ensure
... old expr ...
end
balance
old
old
pre-state
post-state
Runtime Contract Checks
call f
check precondition of f
cache old expressions
execute implementation of f
check postcondition of f
pre-state
balance = old balance + 100
post state


f
require
...
do
...
ensure
... old expr ...
end
Before executing the body
old_expr := expr
a
a.twin
a.deep_twin
a
a.twin
a.deep_twin