Lecture 8
Wednesday January 29

class STACK[G]

imp: ~~ARRAY~~[G] -- ~~end~~ of ~~a~~ is the top

LL ~~front~~ LL

top : G

empty

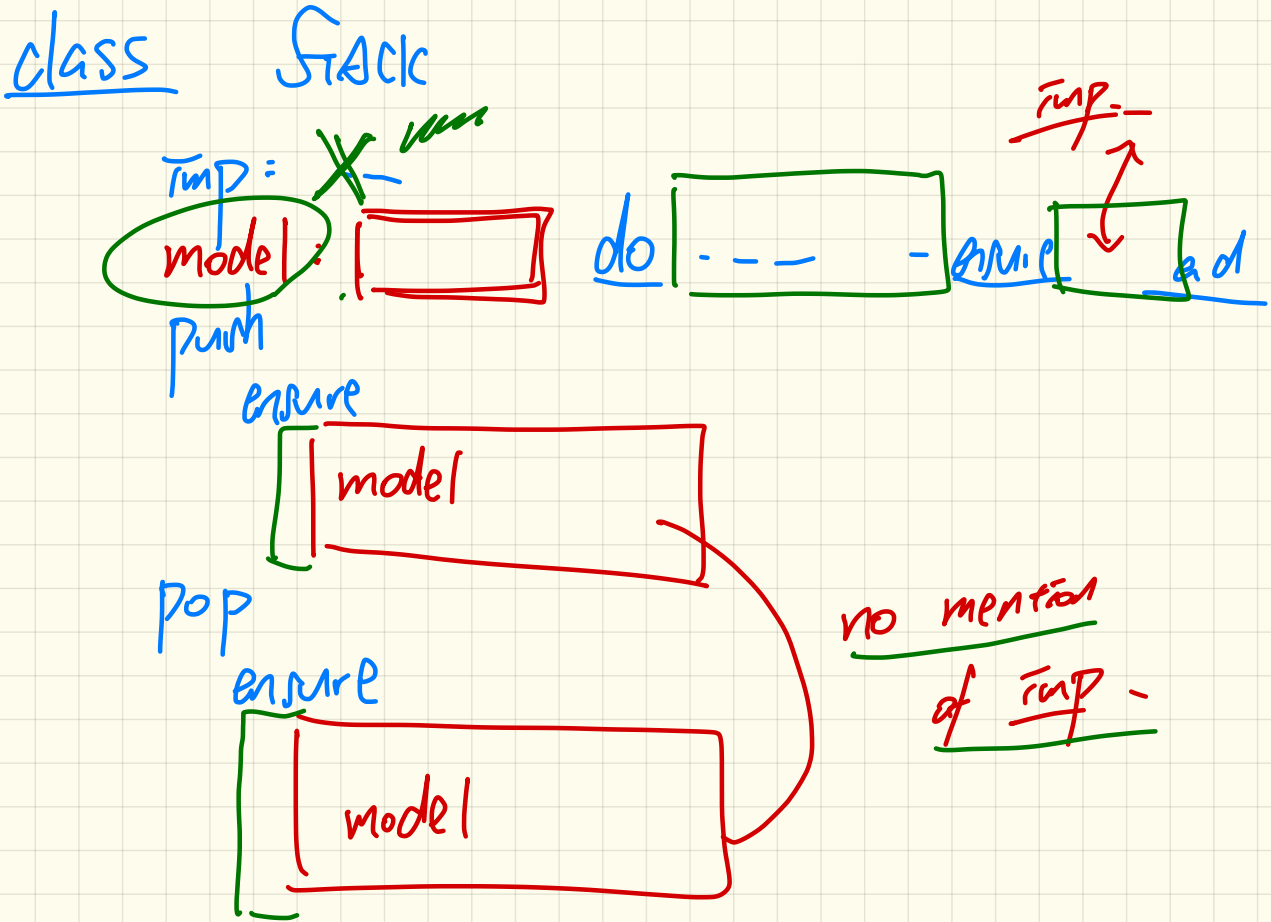
→ Result ~ imp[~~count~~]

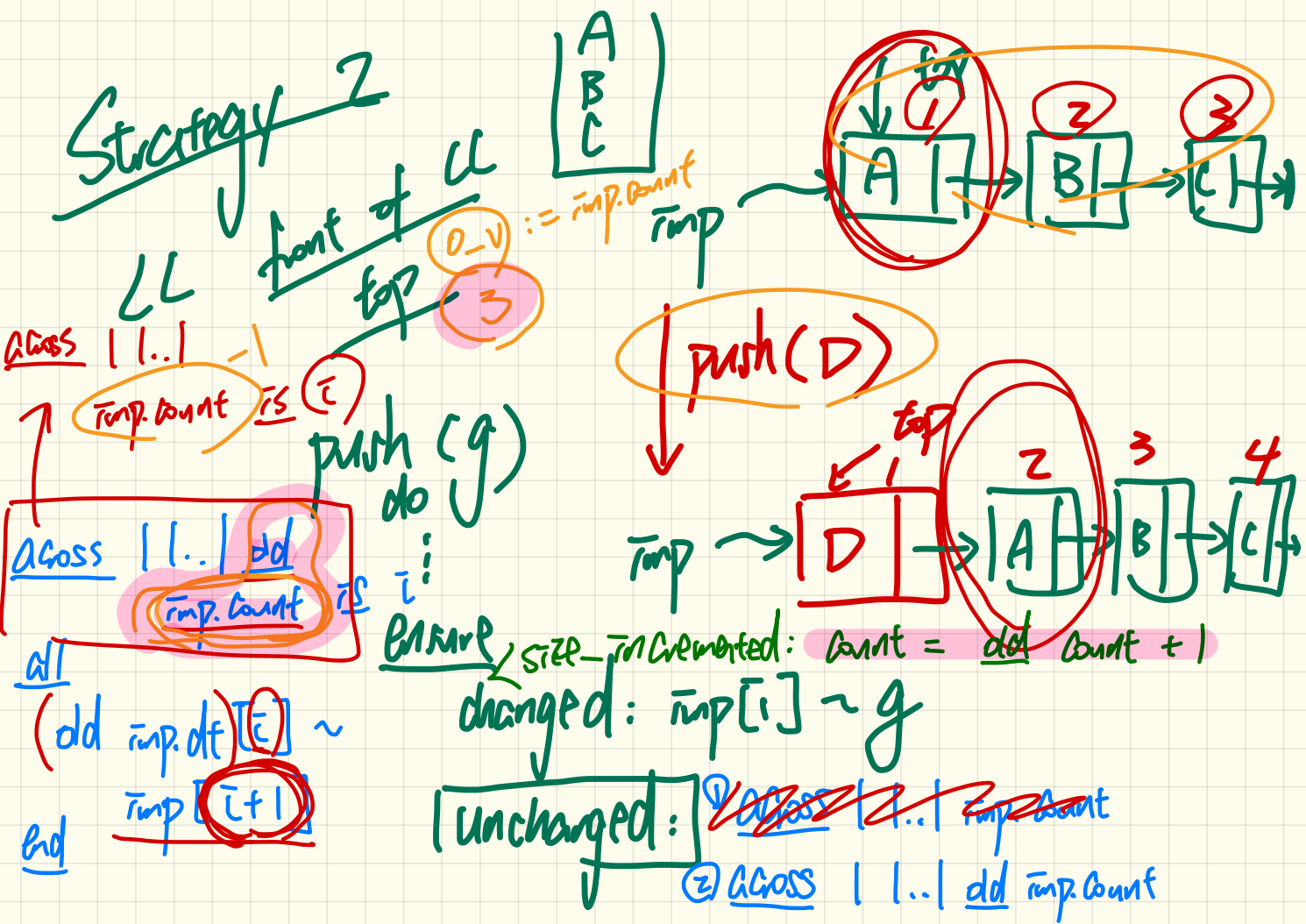
Developing a LIFO Stack

```
class LIFO_STACK[G] create make
feature {NONE} -- Strategy 1: array
  imp: ARRAY[G]
feature -- Initialization
  make do create imp.make_empty ensure imp.count = 0 end
feature -- Commands
  push(g: G)
    do imp.force(g, imp.count + 1)
    ensure
      changed: imp[count] ~ g
      unchanged: across 1 |..| count - 1 as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item] end
    end
  pop
    do imp.remove_tail(1)
    ensure
      changed: count = old count - 1
      unchanged: across 1 |..| count as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item] end
    end
```

```
class LIFO_STACK[G] create make
feature {NONE} -- Strategy 2: linked-list first item as top
  imp: LINKED_LIST[G]
feature -- Initialization
  make do create imp.make ensure imp.count = 0 end
feature -- Commands
  push(g: G)
    do imp.put_front(g)
    ensure
      changed: imp.first ~ g
      unchanged: across 2 |..| count as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item - 1] end
    end
  pop
    do imp.start ; imp.remove
    ensure
      changed: count = old count - 1
      unchanged: across 1 |..| count as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item + 1] end
    end
```

```
class LIFO_STACK[G] create make
feature {NONE} -- Strategy 3: linked-list last item as top
  imp: LINKED_LIST[G]
feature -- Initialization
  make do create imp.make ensure imp.count = 0 end
feature -- Commands
  push(g: G)
    do imp.extend(g)
    ensure
      changed: imp.last ~ g
      unchanged: across 1 |..| count - 1 as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item] end
    end
  pop
    do imp.finish ; imp.remove
    ensure
      changed: count = old count - 1
      unchanged: across 1 |..| count as i all
                   imp[i.item] ~ (old imp.deep_twin)[i.item] end
    end
```

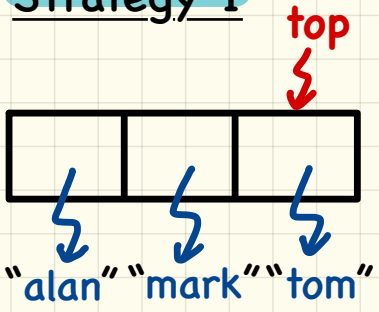

class Stack
imp:
model
push
ensure
model
pop
ensure
model
do
ensure
old
imp
imp
no mention of imp


Strategy 2
front of LL
LL
top
0_V
:= tmp.count
tmp
A
B
C
front
1
2
3
A
B
C
push(D)
top
D
A
B
C
4
across | |..|
tmp.count is i
push (g)
do
across | |..| dd
tmp.count is i
all
(dd tmp.df[i] ~
tmp[i+1])
end
ensure
size_incremented: count = old count + 1
changed: tmp[i] ~ g
unchanged:
① across | |..| tmp.count
② across | |..| dd tmp.count

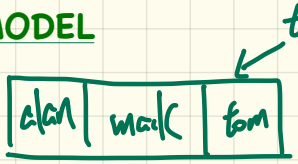
Implementing a LIFO Stack

"tom"
"mark"
"alan"

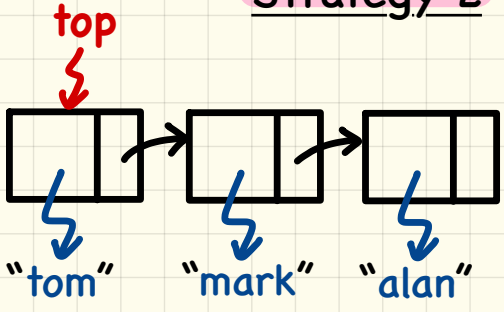
Strategy 1



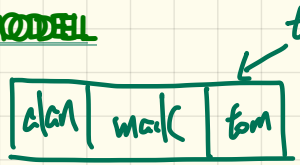
MODEL



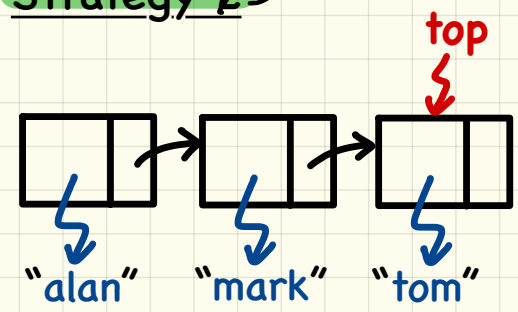
Strategy 2



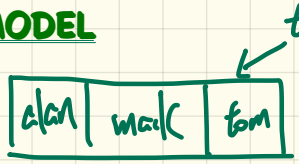
MODEL



Strategy 3



MODEL



Using MATHMODELS Library

Implementing an Abstraction Function

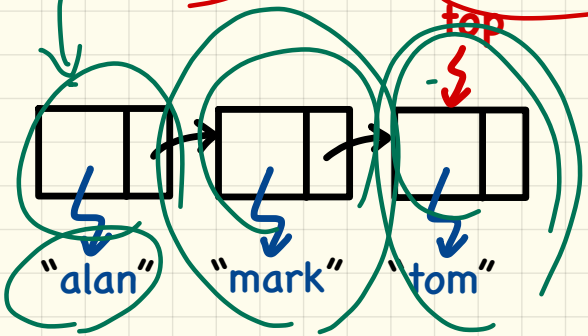
```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty
      across imp as cursor loop Result.append(cursor.item) end
    end
```

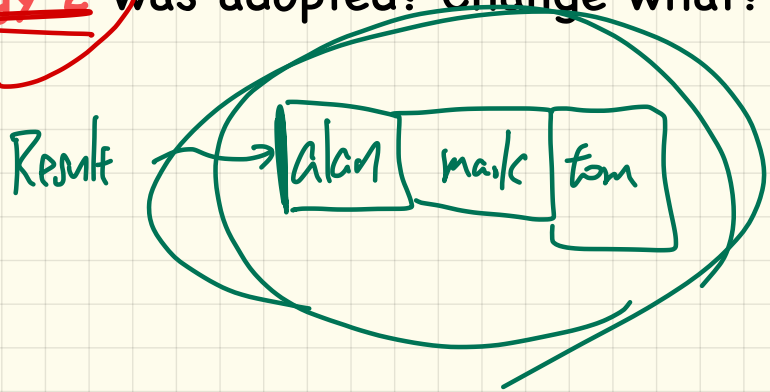
Strategy 3

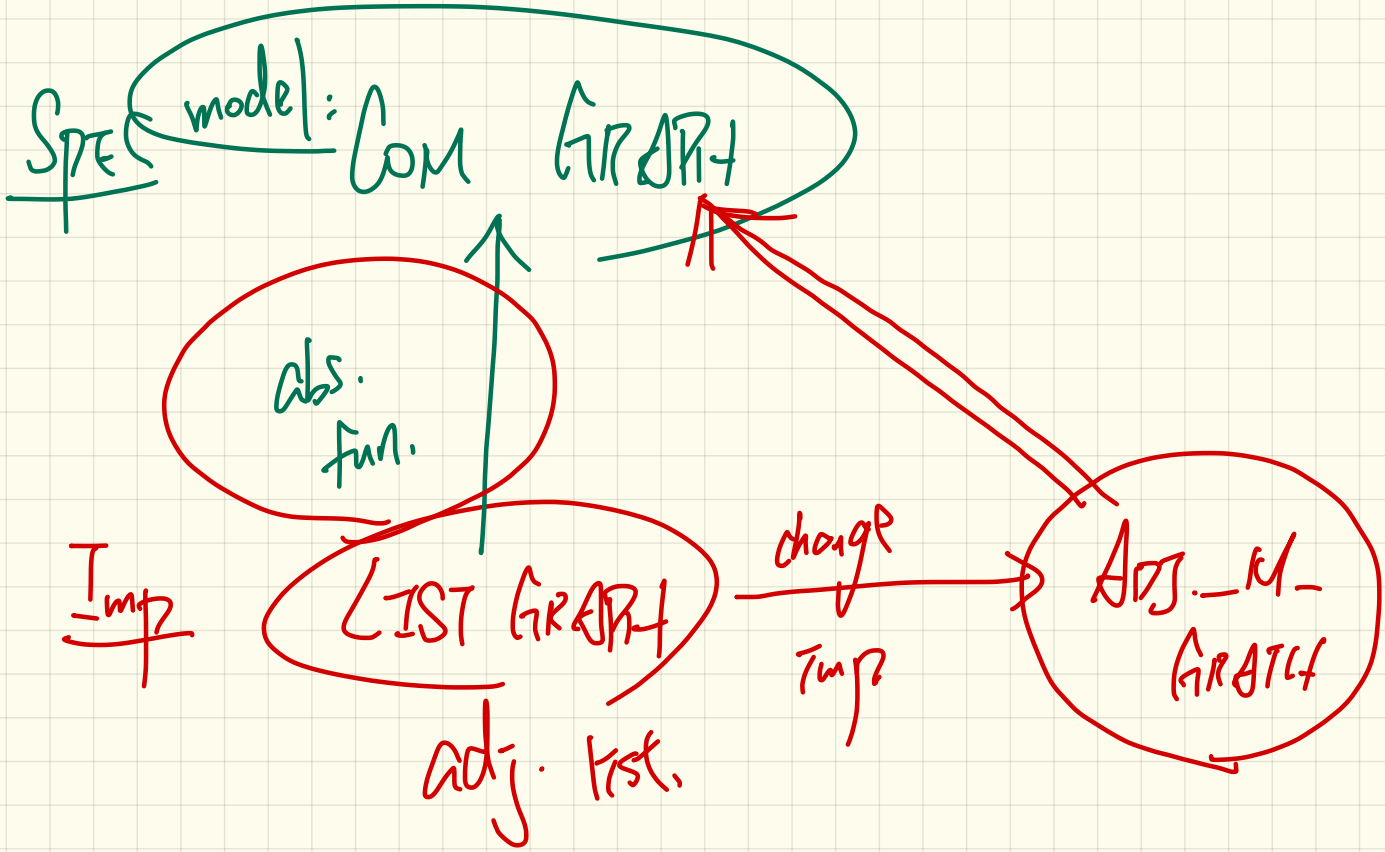
end of CC.

Exercise 1: Write postcondition of model.

Exercise 2: What if Strategy 2 was adopted? Change what?





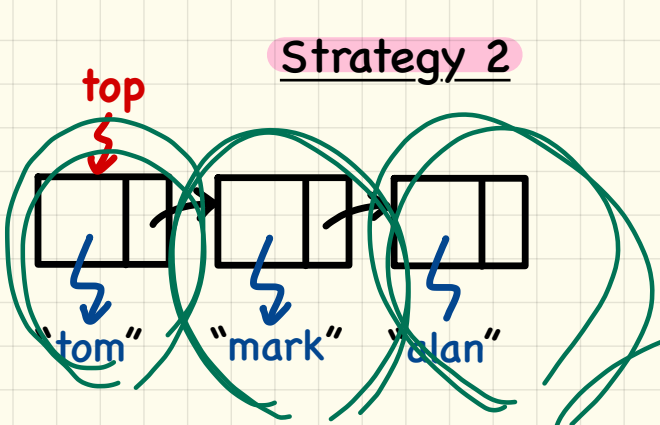

Spec
model: COM GRAPH
abs. fun.
Imp
LIST GRAPH
adj. list.
change
Imp
ADJ_M_GRAPH

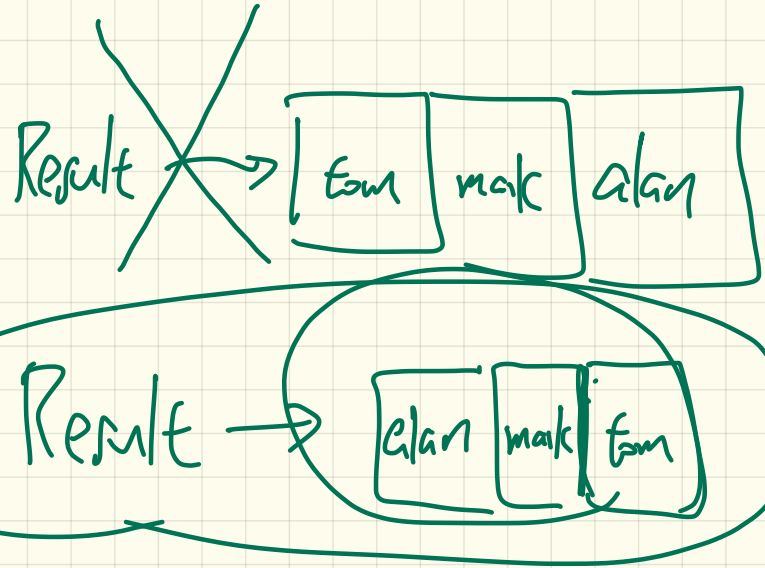
```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation
 imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
 model: SEQ[G]
   do create Result.make_empty
      across imp as cursor loop Result.prepend(cursor.item) end
   end
```

Strategy 2

front is the top.

prepend





Using MATHMODELS Library

Writing Contracts using the Abstraction Function

```
class LIFO_STACK[G -> attached ANY] create make
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
feature -- Commands
  push (g: G)
    ensure model ~ (old model.deep_twin).appended(g) end
```

a seperate call to model in the pre state.

Question: Can clients tell which strategy is being adopted?

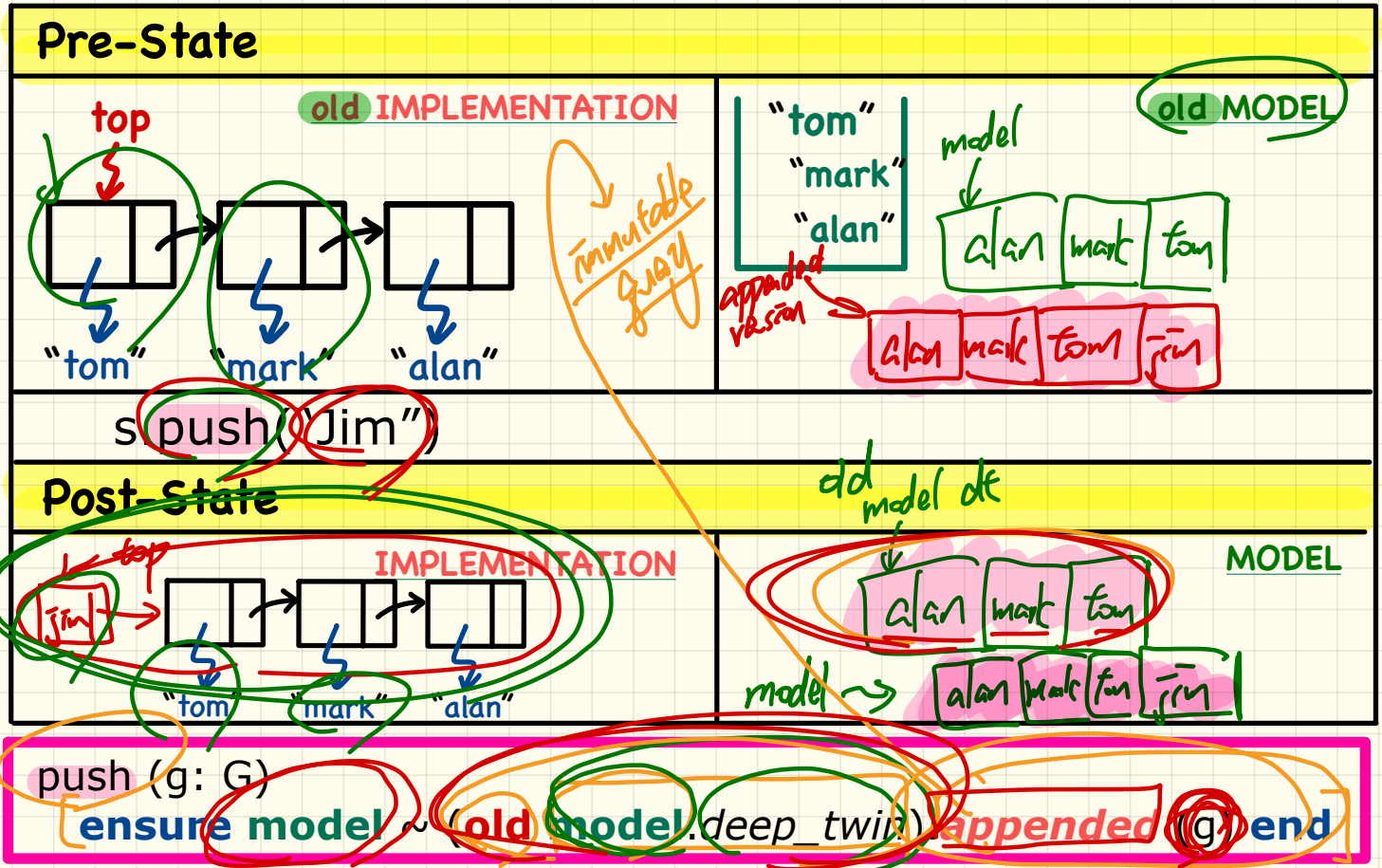
No : no mention of fun?

Exercise: What if strategy was changed? Change what?

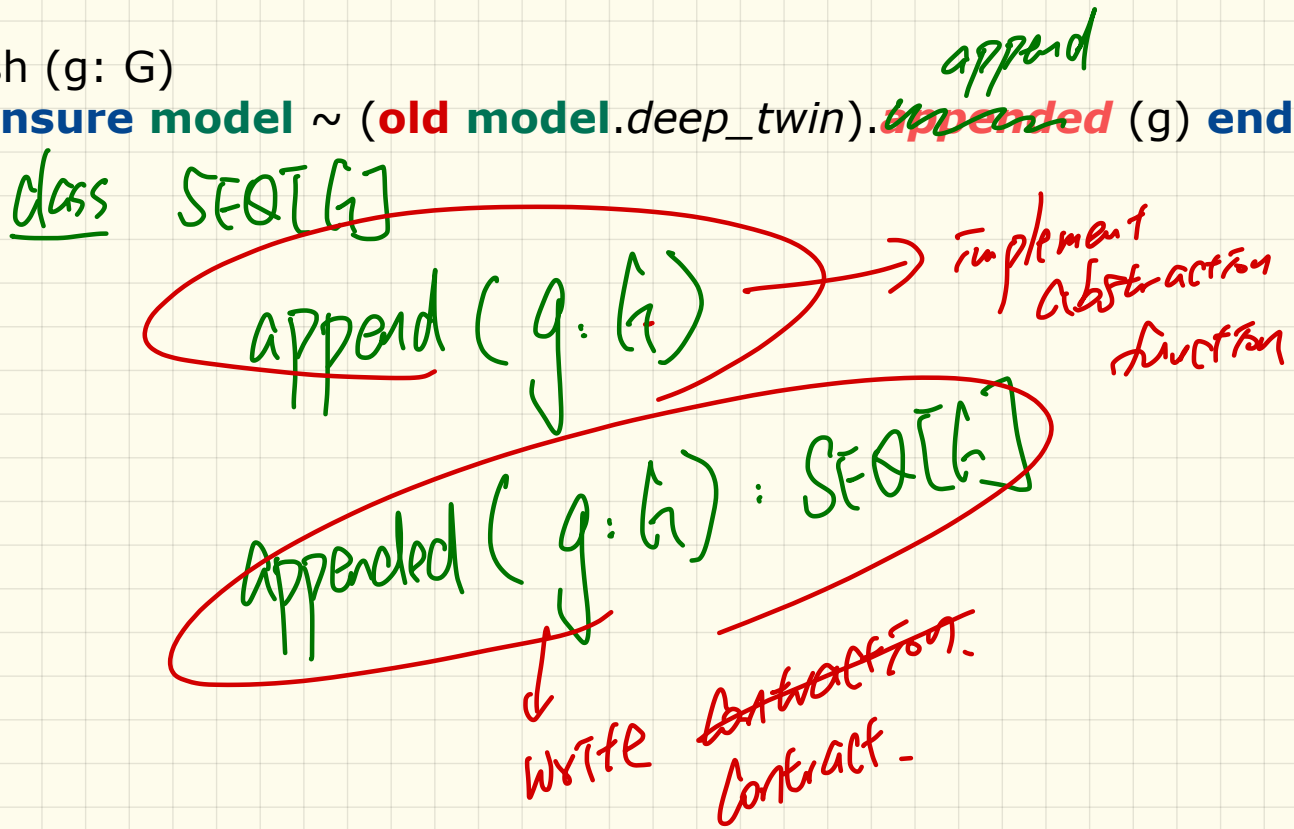
one call to model in the post-state

Checking MATHMODELS Contracts at Runtime

Strategy 2



```
push (g: G)
  ensure model ~ (old model.deep_twin).appended (g) end
```

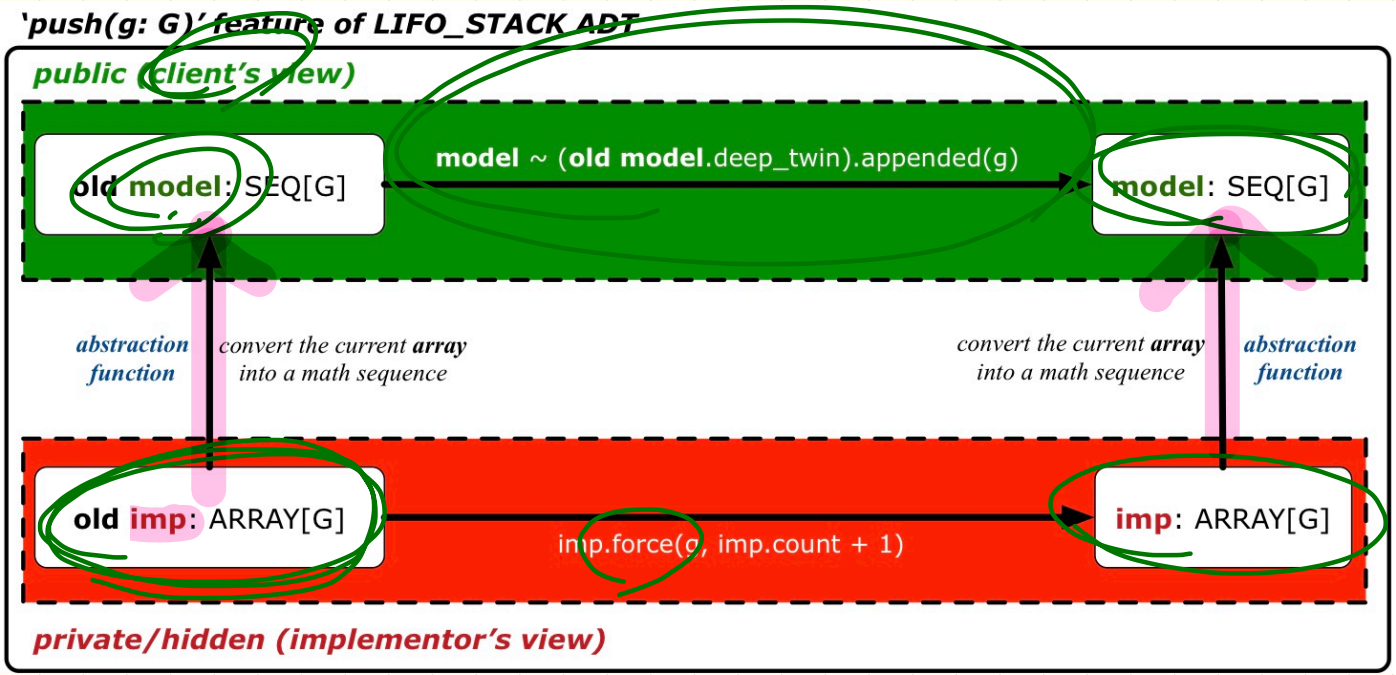


String s = . . .

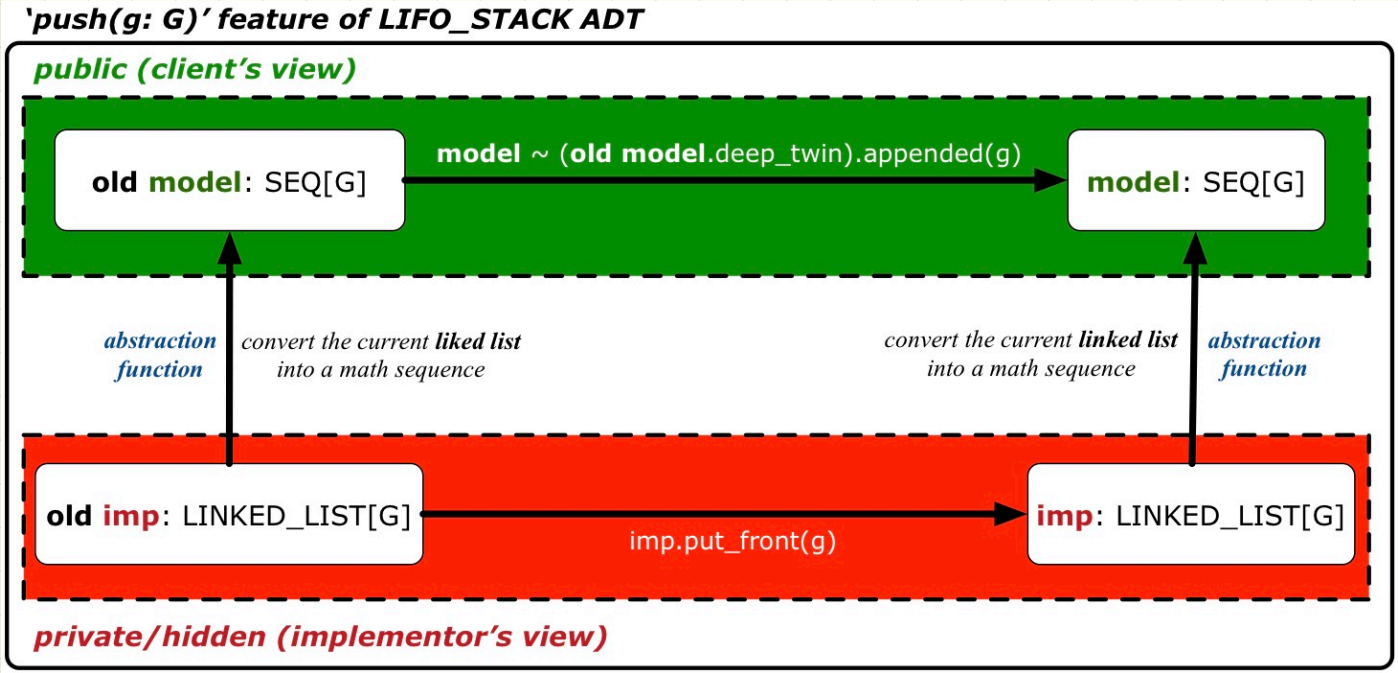
s.substring(____ , ____)

Jim.substring(. .) → "Jim"

Strategy 1: Mathematical Abstraction



Strategy 2: Mathematical Abstraction



Use of MATHMODELS:

Single-Choice Principle

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 1
  imp: ARRAY[G] end
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_from_array (imp)
    ensure
      counts: imp.count = Result.count
      contents: across 1 |..| Result.count as i all
                  Result[i.item] ~ imp[i.item]
    end
feature -- Commands
  make do create imp.make_empty ensure model.count = 0 end
  push (g: G) do imp.force(g, imp.count + 1)
    ensure pushed: model ~ (old model.deep_twin).appended(g) end
  pop do imp.remove_tail(1)
    ensure popped: model ~ (old model.deep_twin).front end
end
```

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 2 (first as top)
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty
      across imp as cursor loop Result.prepend(cursor.item) end
    ensure
      counts: imp.count = Result.count
      contents: across 1 |..| Result.count as i all
                  Result[i.item] ~ imp[count - i.item + 1]
    end
feature -- Commands
  make do create imp.make ensure model.count = 0 end
  push (g: G) do imp.put_front(g)
    ensure pushed: model ~ (old model.deep_twin).appended(g) end
  pop do imp.start ; imp.remove
    ensure popped: model ~ (old model.deep_twin).front end
end
```

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 3 (last as top)
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty
      across imp as cursor loop Result.append(cursor.item) end
    ensure
      counts: imp.count = Result.count
      contents: across 1 |..| Result.count as i all
                  Result[i.item] ~ imp[i.item]
    end
feature -- Commands
  make do create imp.make ensure model.count = 0 end
  push (g: G) do imp.extend(g)
    ensure pushed: model ~ (old model.deep_twin).appended(g) end
  pop do imp.finish ; imp.remove
    ensure popped: model ~ (old model.deep_twin).front end
end
```

Testing REL in MATHMODELS

$$
\begin{aligned}
& r.\textbf{overridden}(\underbrace{\{(a,3),(c,4)\}}_{t}) \\
= \; & \underbrace{\{(a,3),(c,4)\}}_{t} \cup \underbrace{\{(b,2),(b,5),(d,1),(e,2),(f,3)\}}_{r.\textbf{domain\_subtracted}(\underbrace{t.\textbf{domain}}_{\{a,c\}})} \\
= \; & \{(a,3),(c,4),(b,2),(b,5),(d,1),(e,2),(f,3)\}
\end{aligned}
$$

```
test_rel: BOOLEAN
 local
   r, t: REL[STRING, INTEGER]
   ds: SET[STRING]
 do
   create r.make_from_tuple_array (
     <<["a", 1], ["b", 2], ["c", 3],
       ["a", 4], ["b", 5], ["c", 6],
       ["d", 1], ["e", 2], ["f", 3]>>)
   create ds.make_from_array (<<"a">>)
   -- r is not changed by the query 'domain_subtracted'
   t := r.domain_subtracted (ds)
   Result :=
     t /~ r and not t.domain.has ("a") and r.domain.has ("a")
   check Result end
   -- r is changed by the command 'domain_subtract'
   r.domain_subtract (ds)
   Result :=
     t ~ r and not t.domain.has ("a") and not r.domain.has ("a")
 end
```

Say $r = \{(a,1),(b,2),(c,3),(a,4),(b,5),(c,6),(d,1),(e,2),(f,3)\}$

- r.***domain***: set of first-elements from *r*
  - r.**domain** = $\{ d \mid (d,r) \in r \}$
  - e.g., r.**domain** = $\{a,b,c,d,e,f\}$
- r.***range***: set of second-elements from *r*
  - r.**range** = $\{ r \mid (d,r) \in r \}$
  - e.g., r.**range** = $\{1,2,3,4,5,6\}$
- r.***inverse***: a relation like *r* except elements are in reverse order
  - r.**inverse** = $\{ (r,d) \mid (d,r) \in r \}$
  - e.g., r.**inverse** = $\{(1,a),(2,b),(3,c),(4,a),(5,b),(6,c),(1,d),(2,e),(3,f)\}$
- r.***domain_restricted***(ds): sub-relation of *r* with domain *ds*.
  - r.**domain_restricted**(ds) = $\{ (d,r) \mid (d,r) \in r \wedge d \in ds \}$
  - e.g., r.**domain_restricted**({a, b}) = $\{(\mathbf{a},1),(\mathbf{b},2),(\mathbf{a},4),(\mathbf{b},5)\}$
- r.***domain_subtracted***(ds): sub-relation of *r* with domain not *ds*.
  - r.**domain_subtracted**(ds) = $\{ (d,r) \mid (d,r) \in r \wedge d \notin ds \}$
  - e.g., r.**domain_subtracted**({a, b}) = $\{(\mathbf{c},6),(\mathbf{d},1),(\mathbf{e},2),(\mathbf{f},3)\}$
- r.***range_restricted***(rs): sub-relation of *r* with range *rs*.
  - r.**range_restricted**(rs) = $\{ (d,r) \mid (d,r) \in r \wedge r \in rs \}$
  - e.g., r.**range_restricted**({1, 2}) = $\{(a,\mathbf{1}),(b,\mathbf{2}),(d,\mathbf{1}),(e,\mathbf{2})\}$
- r.***range_subtracted***(ds): sub-relation of *r* with range not *ds*.
  - r.**range_subtracted**(rs) = $\{ (d,r) \mid (d,r) \in r \wedge r \notin rs \}$
  - e.g., r.**range_subtracted**({1, 2}) = $\{(c,\mathbf{3}),(a,\mathbf{4}),(b,\mathbf{5}),(c,\mathbf{6})\}$

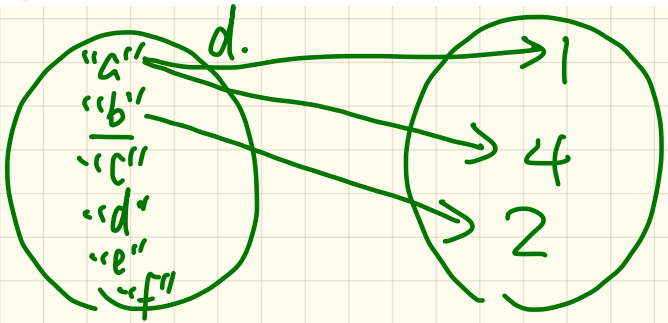
```
test_rel: BOOLEAN
  local
    r, t: REL[STRING, INTEGER]
    ds: SET[STRING]
  do
    create r.make_from_tuple_array (
      <<["a", 1], ["b", 2], ["c", 3],
        ["a", 4], ["b", 5], ["c", 6],
        ["d", 1], ["e", 2], ["f", 3]>>)
    create ds.make_from_array (<<"a">>)
    -- r is not changed by the query 'domain_subtracted'
    t := r.domain_subtracted (ds)
    Result :=
      t /~ r and not t.domain.has ("a") and r.domain.has ("a")
    check Result end
    -- r is changed by the command 'domain_subtract'
    r.domain_subtract (ds)
    Result :=
      t ~ r and not t.domain.has ("a") and not r.domain.has ("a")
  end
```

r →

t → << a 1, b 2, c 3
a 4, b 5, c 6
d 1, e 2, f 3 >>

"a"

domain_restricted .



Model of an Example Birthday Book