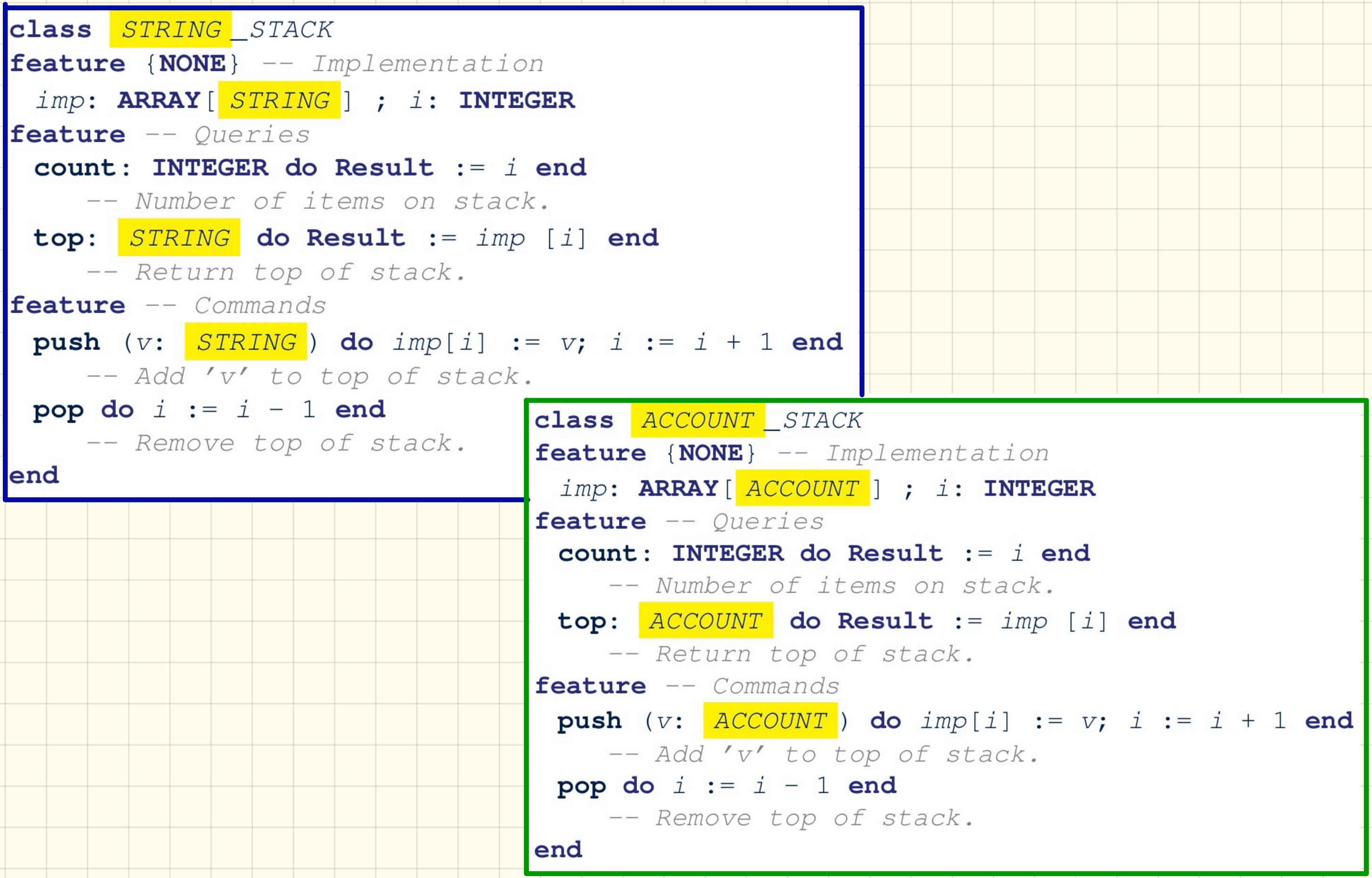
Lecture 3

Part 1

Writing & Using a Generic Class

Stack of Strings vs. Stack of Accounts



```
class STRING_STACK
feature {NONE} -- Implementation
  imp: ARRAY[ STRING ] ; i: INTEGER
feature -- Queries
  count: INTEGER do Result := i end
      -- Number of items on stack.
  top: STRING do Result := imp [i] end
      -- Return top of stack.
feature -- Commands
  push (v: STRING ) do imp[i] := v; i := i + 1 end
      -- Add 'v' to top of stack.
  pop do i := i - 1 end
      -- Remove top of stack.
end
```

```
class ACCOUNT_STACK
feature {NONE} -- Implementation
  imp: ARRAY[ ACCOUNT ] ; i: INTEGER
feature -- Queries
  count: INTEGER do Result := i end
      -- Number of items on stack.
  top: ACCOUNT do Result := imp [i] end
      -- Return top of stack.
feature -- Commands
  push (v: ACCOUNT ) do imp[i] := v; i := i + 1 end
      -- Add 'v' to top of stack.
  pop do i := i - 1 end
      -- Remove top of stack.
end
```

A Generic Stack

Supplier

```
class STACK [G]
feature {NONE} -- Implementation
  imp: ARRAY[ G ] ; i: INTEGER
feature -- Queries
  count: INTEGER do Result := i end
      -- Number of items on stack.
  top: G do Result := imp [i] end
      -- Return top of stack.
feature -- Commands
  push (v: G ) do imp[i] := v; i := i + 1 end
      -- Add 'v' to top of stack.
  pop do i := i - 1 end
      -- Remove top of stack.
end
```

Client

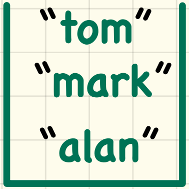
```
test_stacks: BOOLEAN
  local
    ss: STACK[STRING] ; sa: STACK[ACCOUNT]
    s: STRING ; a: ACCOUNT
  do
    ss.push("A")
    ss.push(create {ACCOUNT}.make ("Mark", 200))
    s := ss.top
    a := ss.top
    sa.push(create {ACCOUNT}.make ("Alan", 100))
    sa.push("B")
    a := sa.top
    s := sa.top
  end
```

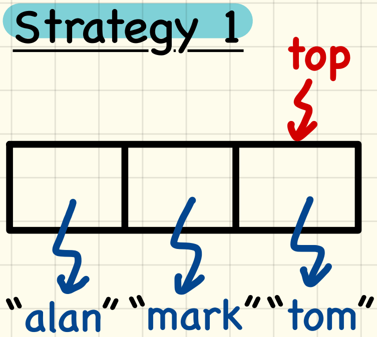
Lecture 3

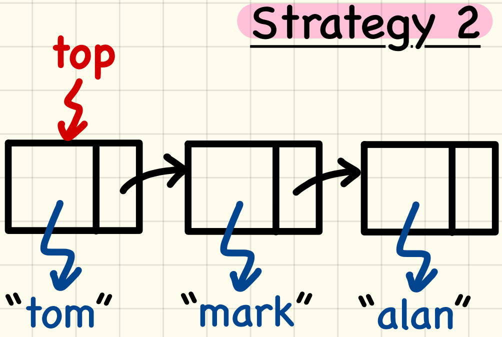
Part 2

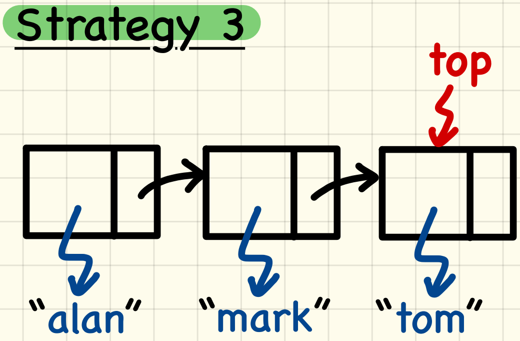
Abstractions via Mathematical Models

Implementing a LIFO Stack









Developing a LIFO Stack

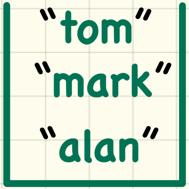
```
class LIFO_STACK[G] create make
feature {NONE} -- Strategy 1: array
  imp: ARRAY[G]
feature -- Initialization
  make do create imp.make_empty ensure imp.count = 0 end
feature -- Commands
  push(g: G)
    do imp.force(g, imp.count + 1)
    ensure
      changed: imp[count] ~ g
      unchanged: across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

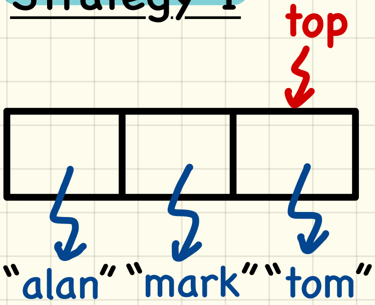
```
class LIFO_STACK[G] create make
feature {NONE} -- Strategy 3: linked-list last item as top
  imp: LINKED_LIST[G]
feature -- Initialization
  make do create imp.make ensure imp.count = 0 end
feature -- Commands
  push(g: G)
    do imp.extend(g)
    ensure
      changed: imp.last ~ g
      unchanged: 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
```

Abstracting a LIFO Stack

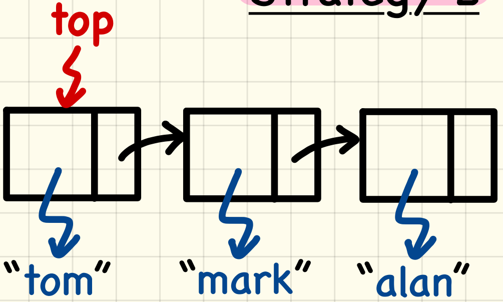
MODEL



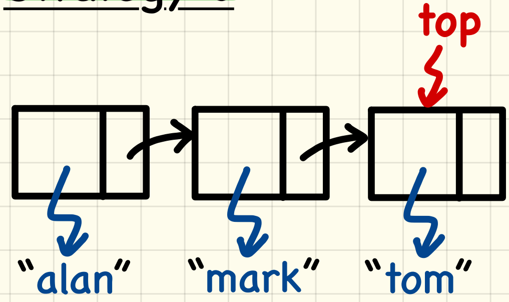
Strategy 1



Strategy 2

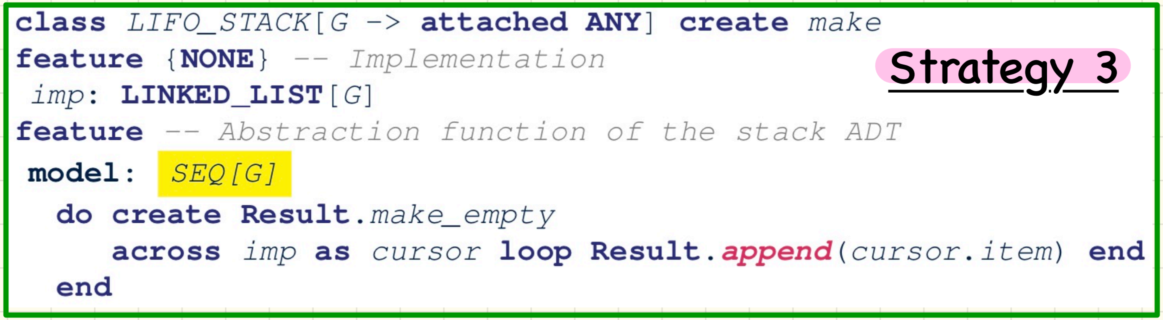


Strategy 3



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
```

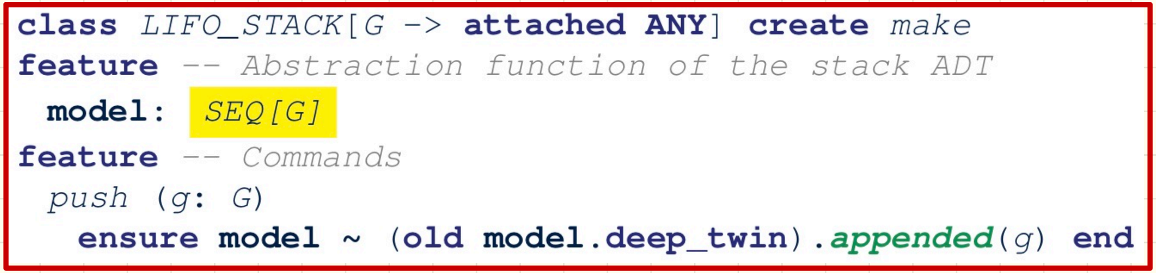
Strategy 3

Exercise 1: Write postcondition of model.

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

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
```

Question: Can clients tell which strategy is being adopted?

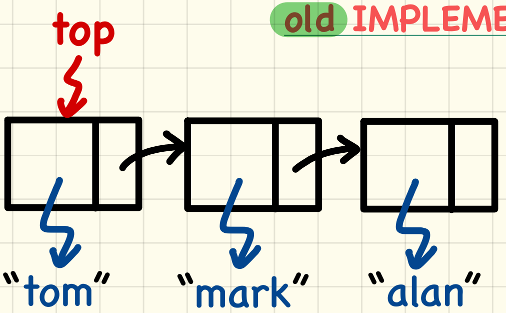
Exercise: What if strategy was changed? Change what?

Checking MATHMODELS Contracts at Runtime

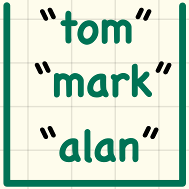
Strategy 2

Pre-State

old IMPLEMENTATION



old MODEL



s.push("Jim")

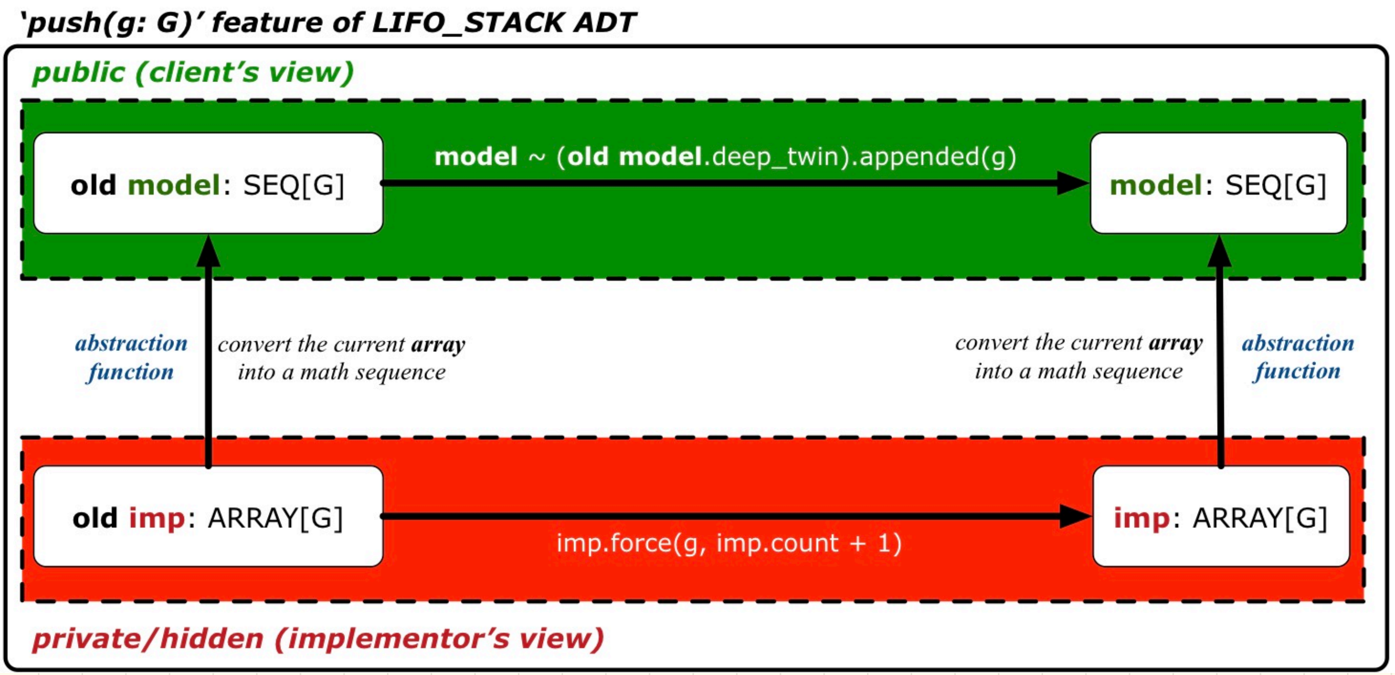
Post-State

IMPLEMENTATION

MODEL

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

Strategy 1: Mathematical Abstraction



'push(g: G)' feature of LIFO_STACK ADT

public (client's view)

old model: SEQ[G]

model ~ (**old model**.deep_twin).appended(g)

model: SEQ[G]

abstraction function

*convert the current **array** into a math sequence*

*convert the current **array** into a math sequence*

abstraction function

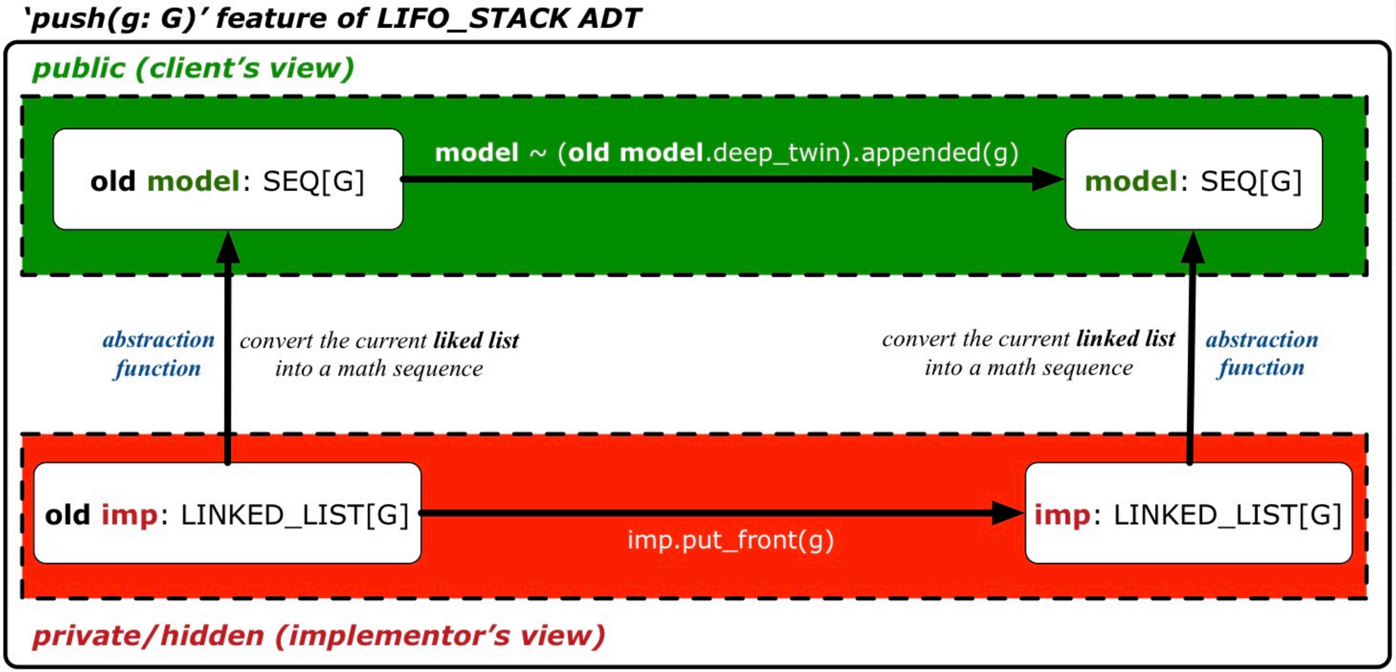
old imp: ARRAY[G]

imp.force(g, imp.count + 1)

imp: ARRAY[G]

private/hidden (implementor's view)

Strategy 2: Mathematical Abstraction



'push(g: G)' feature of LIFO_STACK ADT
public (client's view)
old model: SEQ[G]
model ~ (old model.deep_twin).appended(g)
model: SEQ[G]
abstraction function
convert the current liked list into a math sequence
convert the current linked list into a math sequence
abstraction function
old imp: LINKED_LIST[G]
imp.put_front(g)
imp: LINKED_LIST[G]
private/hidden (implementor's view)