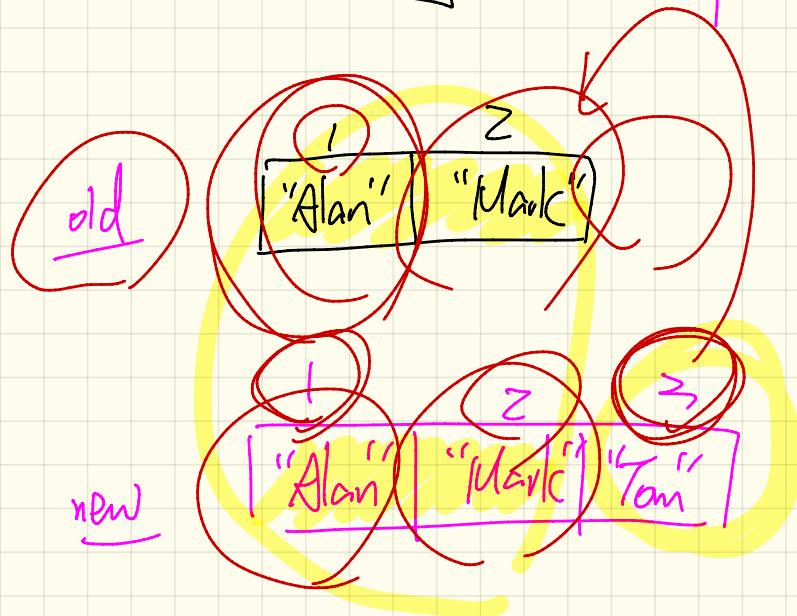


Monday February 11
Lecture 10

S1

`["Mark"]`
`["Alan"]`

push (`"Tom"`)



Q: Empty collection?

S2.

first. first

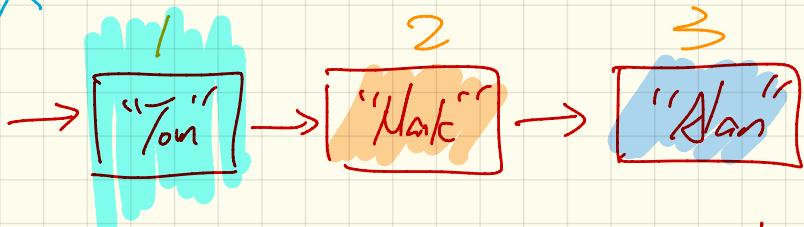
first[1]

old

["Tom"
"Mark"
"Alan"]



first. lower X
upper
new



across 1..1 count as \bar{c}
all

end
[Count - \bar{c} .item.]

push("Tom")

across 2..1 count as \bar{c}

all

imp [i.item] ~
end (old imp.d-t) [i.item-1]

across count 1..1 | as \bar{c} X

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
```

Annotations on the right side of the code:

- A pink circle highlights "Strategy 1: array".
- A blue circle highlights "imp.deep_twin[i.item]" in the "push" command.
- A blue circle highlights "imp.deep_twin[i.item]" in the "pop" command.
- A blue circle highlights "imp.deep_twin[i.item]" in the "push" command's ensure block.
- A blue circle highlights "imp.deep_twin[i.item]" in the "pop" command's ensure block.

Handwritten notes on the left side of the code:

- A pink arrow points from "imp" to "imp.deep_twin[i.item]" in the "push" command's ensure block.
- The handwritten notes say: "not only but also contracts we must modify around violates SCP".

```
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])
      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
```

Annotations on the left side of the code:

- A pink circle highlights "Strategy 2: linked-list first item as top".
- A pink circle highlights "LINKED_LIST[G]".
- A blue circle highlights "imp.deep_twin[i.item]" in the "push" command's ensure block.
- A blue circle highlights "imp.deep_twin[i.item]" in the "pop" command's ensure block.
- A blue circle highlights "imp.deep_twin[i.item + 1]" in the "pop" command's ensure block.

```
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
```

Annotations on the right side of the code:

- A red border surrounds the entire code block.
- A blue circle highlights "LINKED_LIST[G]".
- A blue circle highlights "imp.deep_twin[i.item]" in the "push" command's ensure block.
- A blue circle highlights "imp.deep_twin[i.item]" in the "pop" command's ensure block.

class C

imp : ? ?

f1

ensure

imp \rightarrow imp

f2

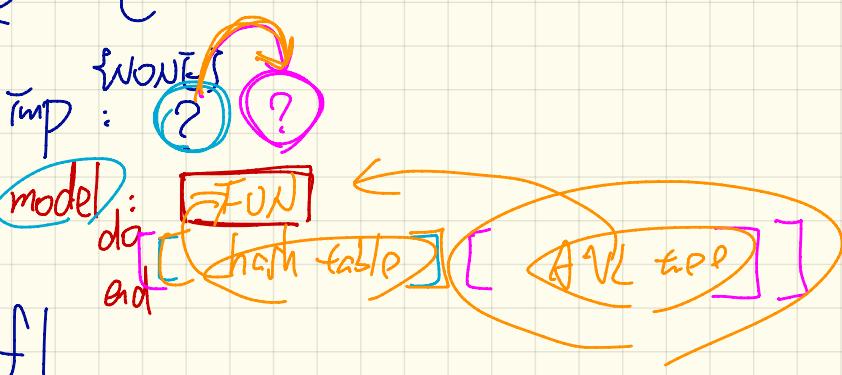
ensure

imp \rightarrow imp

end

class

C



ensure



f2

ensure



end

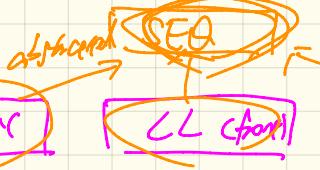
Using MATHMODELS Library

Implementing 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

```



Seq Model

end of seq
is the top

Writing Contracts using 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

```

call to query

a later call to same query.

dt of the return value

```

class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 1
  imp: ARRAY[G]
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

```

Implementing a LIFO STACK

class LIFO_STACK[...]

imp: LCG[G] -> S2

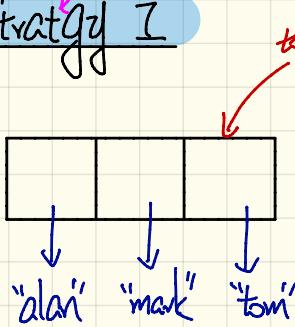
model: SEQ[G]

do
 create Result. map - empty
 -> across imp as cursor
end
 end

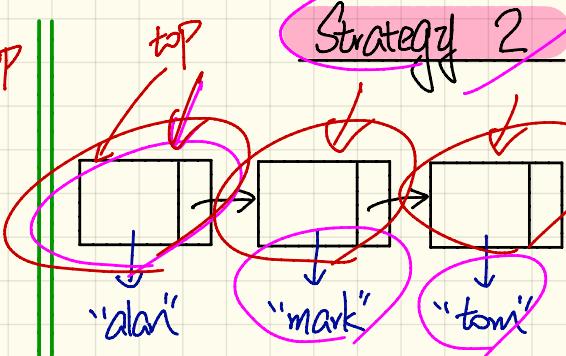
prepend



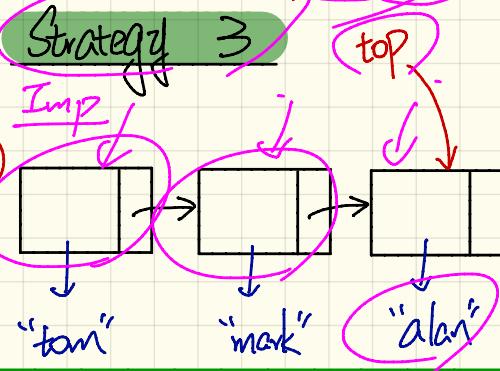
Strategy 1



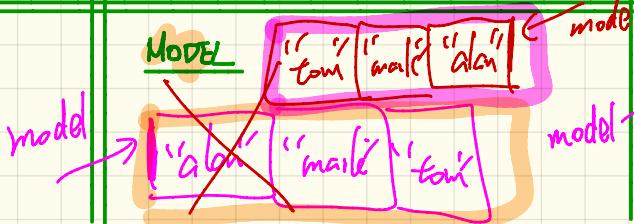
Strategy 2



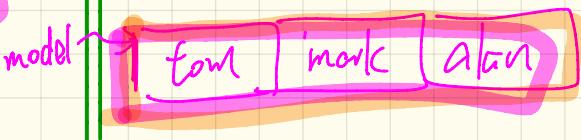
Strategy 3



MODEL



MODEL



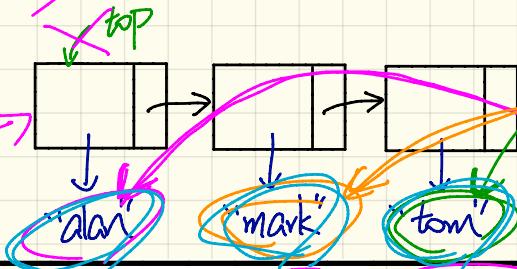
Checking MATH MODELS Contracts at Runtime

Strategy 2

Alan
mark
tom

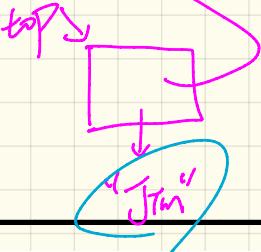
Pre-State

Implementation



s.push("Jim")

Post-State



Immutable
every

push (g: G)

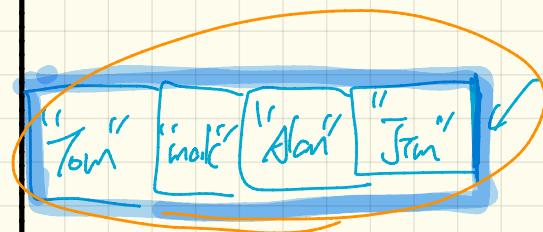
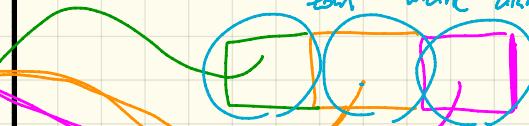
ensure model ~ (old model.deep_twin).appended(g) end

model.dt

Model

model.dt

"Jim"
model



model

gory
of

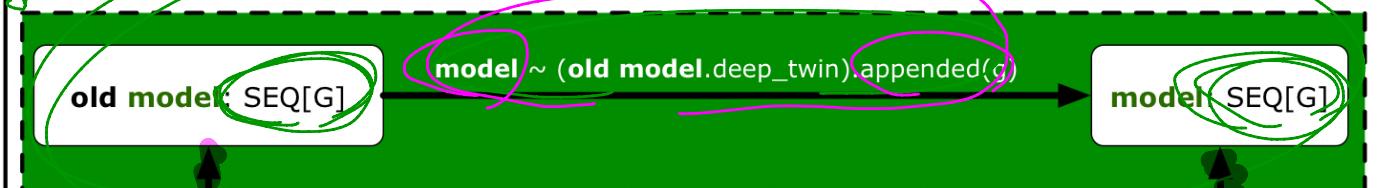
old model.deep_twin).appended(g)

end

Strategy 1 : Mathematical Abstraction

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

public (client's view)



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]

abstraction function
convert the current *linked list*
into a math sequence

// *zdarroj!*

model \sim (old model.deep_twin).appended(g)

model: SEQ[G]

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)