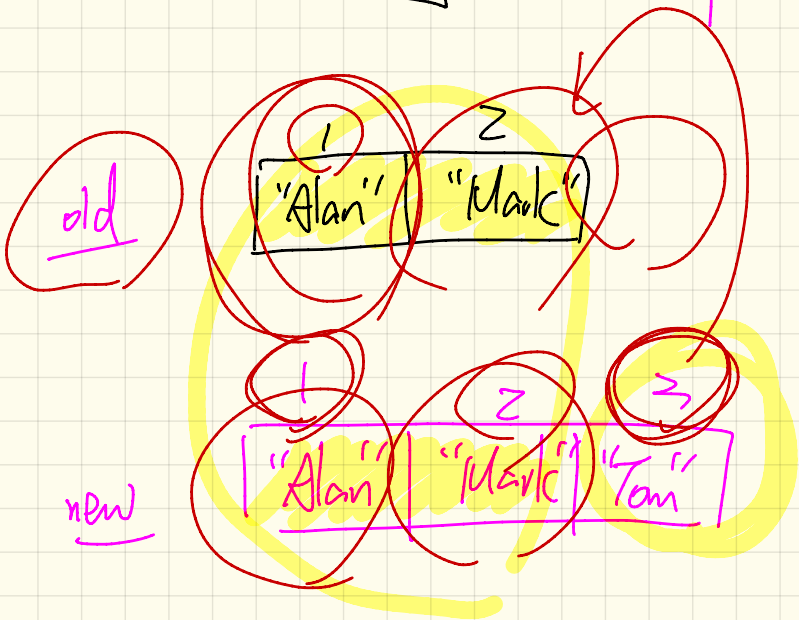
Monday February 11

Lecture 10

S1

push ("Tom")



S2.

"Tom"
"Mark"
"Alan"

Q: empty collection?

push("Tom")

across (2)..|count| as i

all

[imp[i.item] ~ (old imp.d_t)[i.item-1]]

end

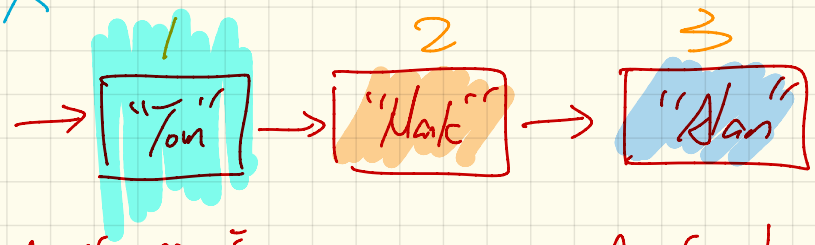
lst.first

:=

lst[1]

old

lst.lower upper new X



across 1..|count| as i

all

[count - i.item.

end

across

count 1..| | as i 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
```

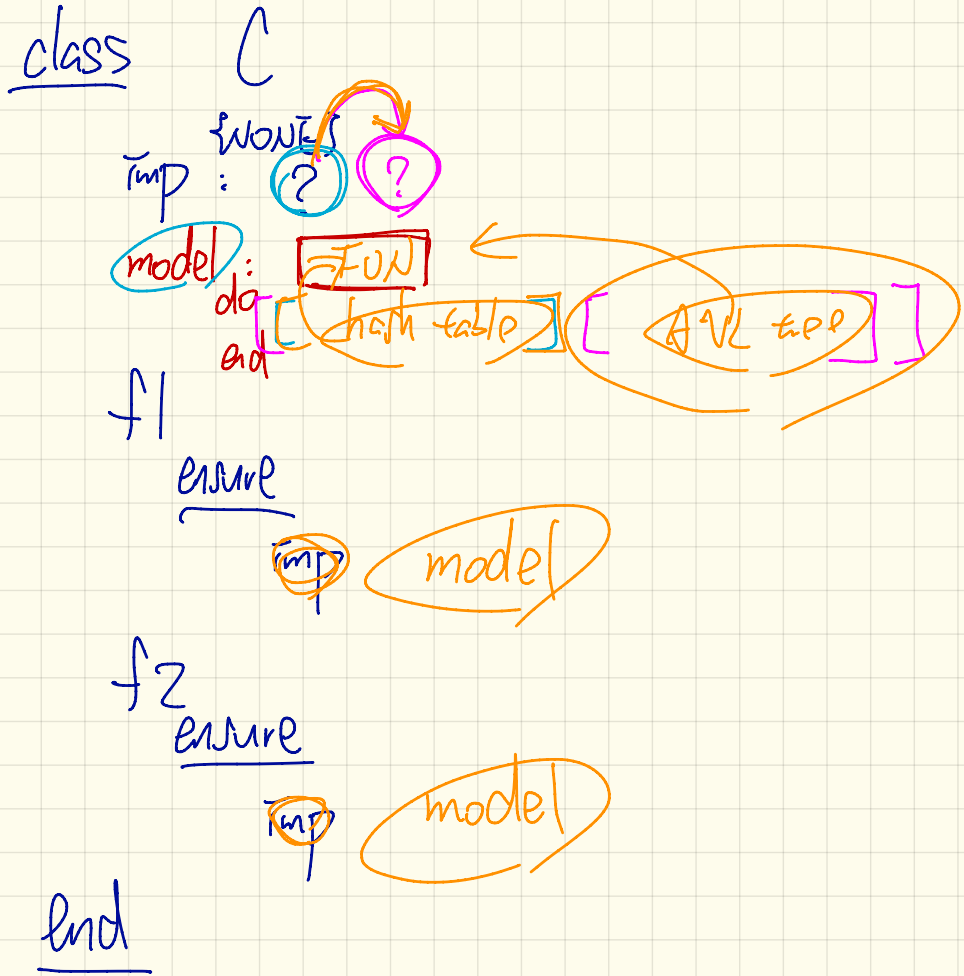
not only imp. but also contracts are must modify accord. ⇒ violates SCP.

```
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
```

-1

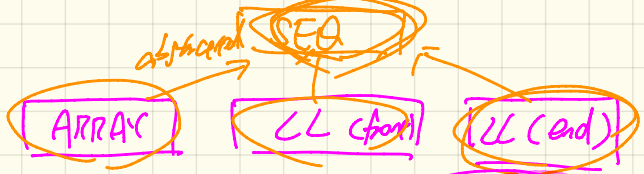
```
class LIFO_STACK[G] create make
feature {NONE} -- Strategy 3: linked-list last item as top
  imp: LINKED_LIST[G]
feature -- Initialization
  make do create imp.make ensure imp.count = 0 end
feature -- Commands
  push(g: G)
    do imp.extend(g)
    ensure
      changed: imp.last ~ g
      unchanged: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

    f1
        ensure
            imp -> imp

    f2
        ensure
            imp -> imp

end
```


class C
{NONE}
tmp : ?
?
model : FUN
do
end
hash table
AVL tree
f1
ensure
tmp
model
f2
ensure
tmp
model
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 some 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[G]

imp : LL[G] -- S2

model : SEQ[G]

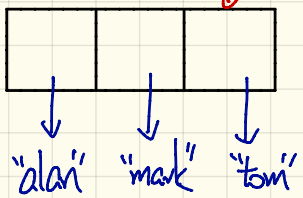
do
create Result.make_empty
across imp as cursor
loop Result.append (cursor.item)
end

end

prepend

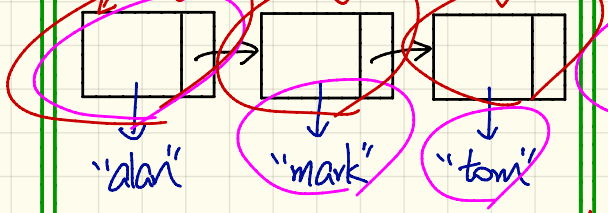
"tom"
"mark"
"alan"

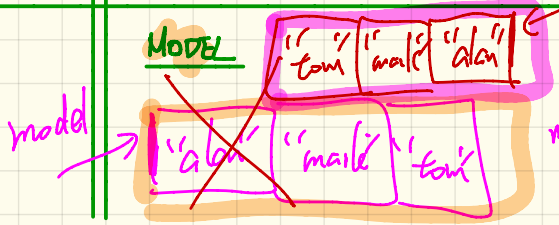
Strategy 1



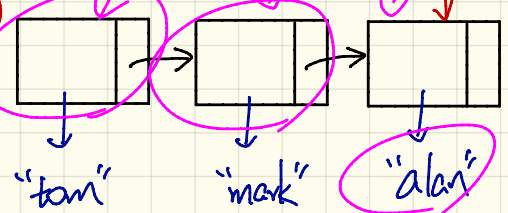
MODEL

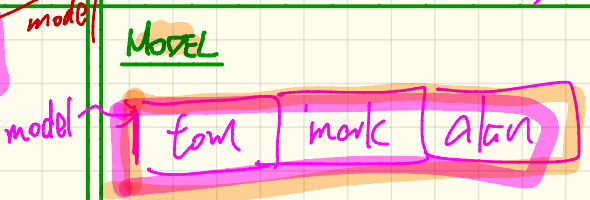
Strategy 2





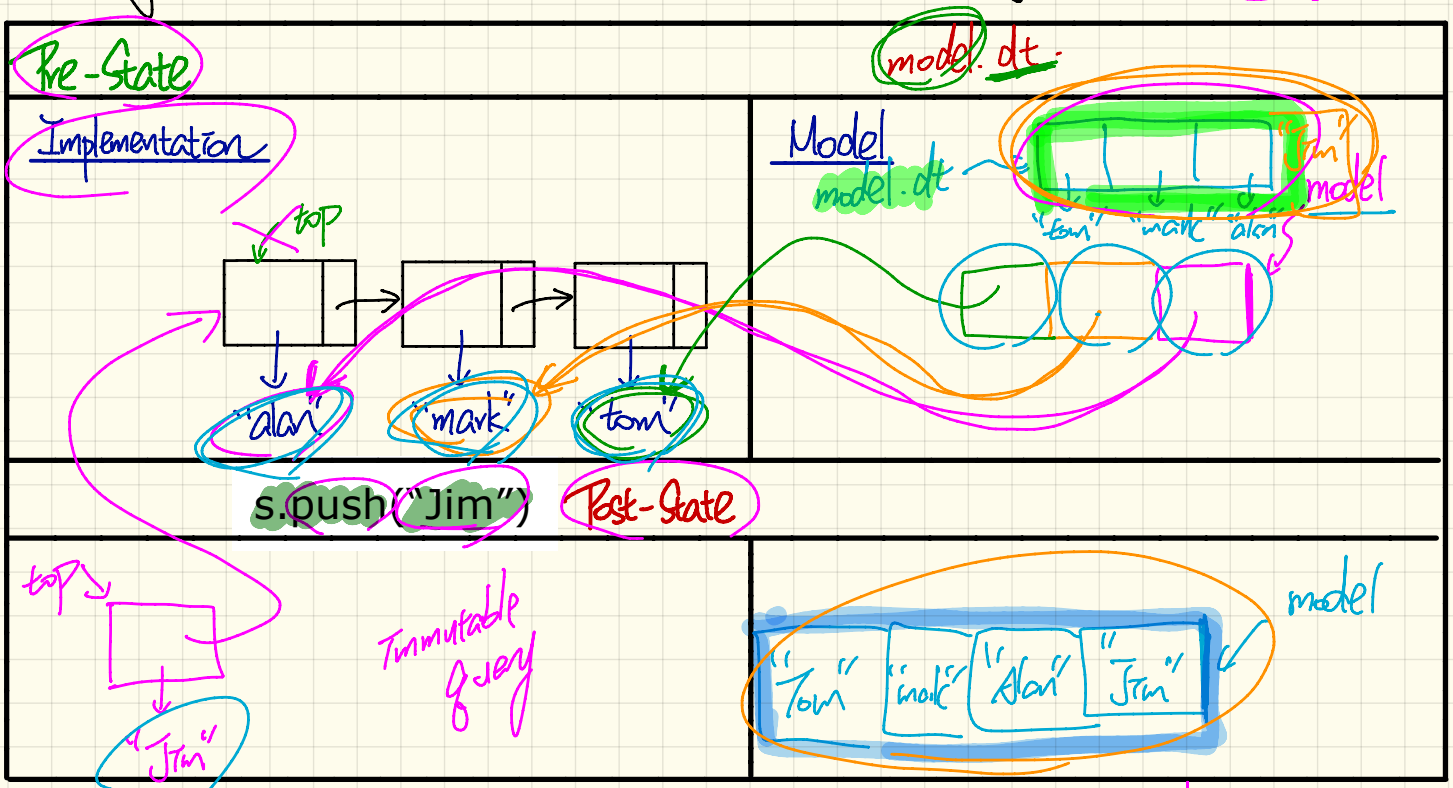
Strategy 3





Checking MATHMODELS Contracts at Runtime — Strategy 2

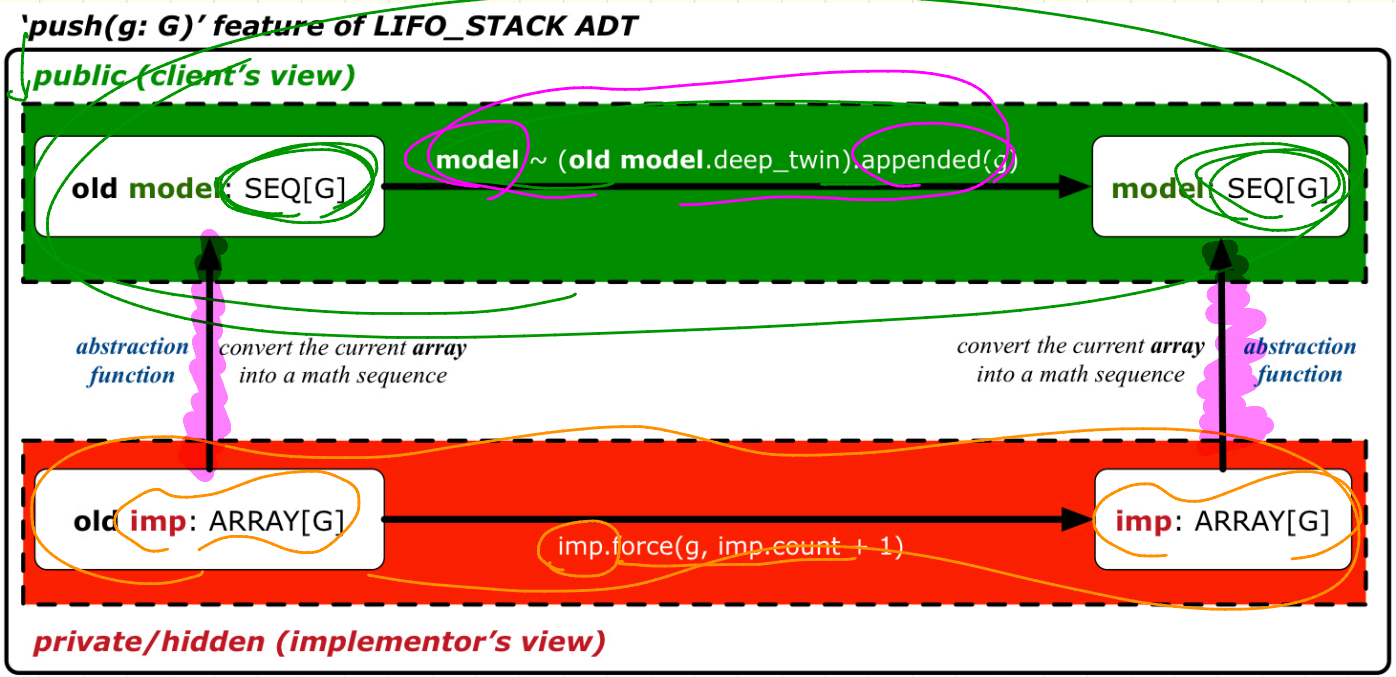
Alan
mark
tom




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

query

Strategy 1 : Mathematical Abstraction



Strategy 2. Mathematical Abstraction