Thursday Nov. 8

Lecture 17

Void Safe in Java? (3)

```
1  class Point {
2    double x;
3    double y;
4    Point(double x, double y) {
5      this.x = x;
6      this.y = y;
7  }
```

```
1  class PointCollector {
2    ArrayList<Point> points;
3    PointCollector() {
4      points = new ArrayList<>(); }
5    void addPoint(Point p) {
6      points.add(p); }
7    Point getPointAt(int i) {
8      return points.get(i); } }
```

```
1  public void test3() {
2    PointCollector pc = new PointCollector();
3    Scanner input = new Scanner(System.in);
4    System.out.println("Ener an integer:");
5    int i = input.nextInt();
6    if (x < 0) { pc = null; }
7    pc.addPoint(new Point(3, 4));
8    assertTrue(pc.getPointAt(0).x == 3 && pc.getPointAt(0).y == 4);
9  }
```

Non-detachable vs. Detachable

nullable

attached

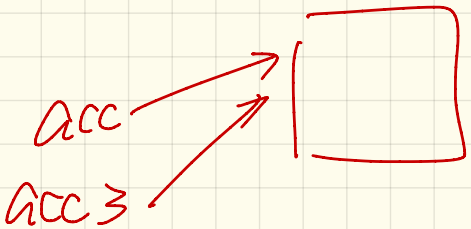
acc → ACC

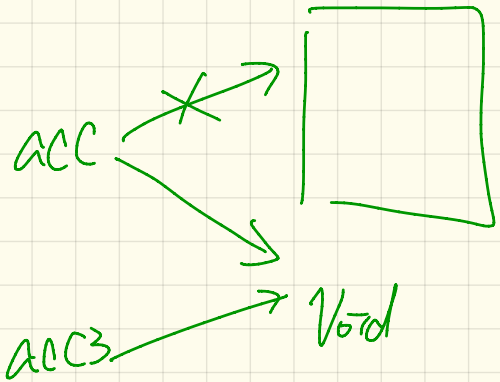
null = void

acc	ACCOUNT	
acc2	ACCOUNT	
acc3	detachable	ACCOUNT

!!! Account acc3; Java

1. acc := acc2 ✓
2. acc := acc3 ✗
3. acc3 := acc





Void Safe in Eiffel ! (1)

```
1  class
2    POINT
3  create
4    make
5  feature
6    x: REAL
7    y: REAL
8  feature
9    make (nx: REAL; ny: REAL)
10     do x := nx
11        y := ny
12     end
13 end
```

```
1  class
2    POINT_COLLECTOR_1
3  create
4    make
5  feature
6    points: LINKED_LIST[POINT]
7  feature
8    make do end
9  add_point (p: POINT)
10   do points.extend (p) end
11 get_point_at (i: INTEGER): POINT
12   do Result := points [i] end
13 end
```

Void Safe in Eiffel ! (2)

```
1  class
2    POINT
3  create
4    make
5  feature
6    x: REAL
7    y: REAL
8  feature
9    make (nx: REAL; ny: REAL)
10     do x := nx
11        y := ny
12     end
13 end
```

```
1  class
2    POINT_COLLECTOR_2
3  create
4    make
5  feature
6    points: LINKED_LIST[POINT]
7  feature
8    make do create points.make end
9  add_point (p: POINT)
10   do points.extend (p) end
11 get_point_at (i: INTEGER): POINT
12   do Result := points [i] end
13 end
```

```
1  test_2: BOOLEAN
2    local
3      pc: POINT_COLLECTOR_2 ; p: POINT
4    do
5      create pc.make
6      p := Void
7      pc.add_point (p)
8      p := pc.get_point_at (0)
9      Result := p.x = 3 and p.y = 4
10   end
```

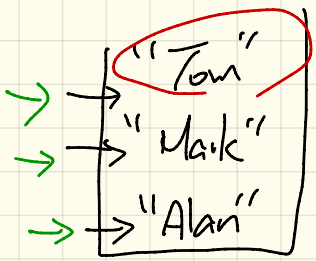
the 1st usage of pc that might be void

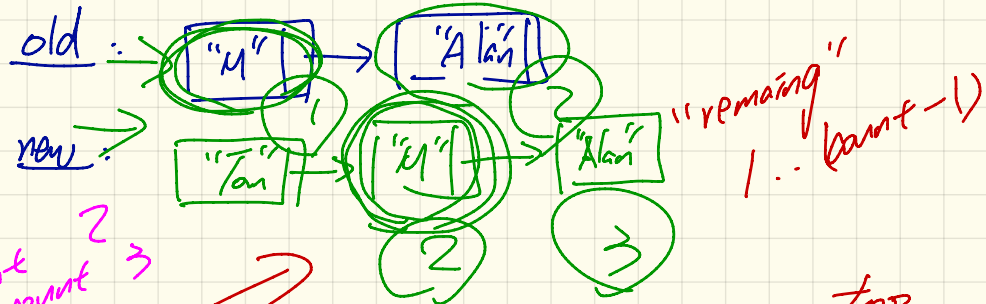
Void Safe in Eiffel ! (3)

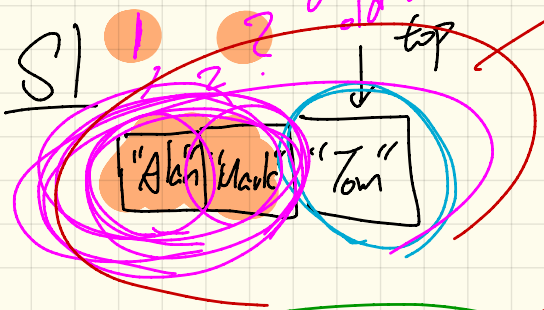
```
1  class
2    POINT
3  create
4    make
5  feature
6    x: REAL
7    y: REAL
8  feature
9    make (nx: REAL; ny: REAL)
10     do x := nx
11        y := ny
12     end
13 end
```

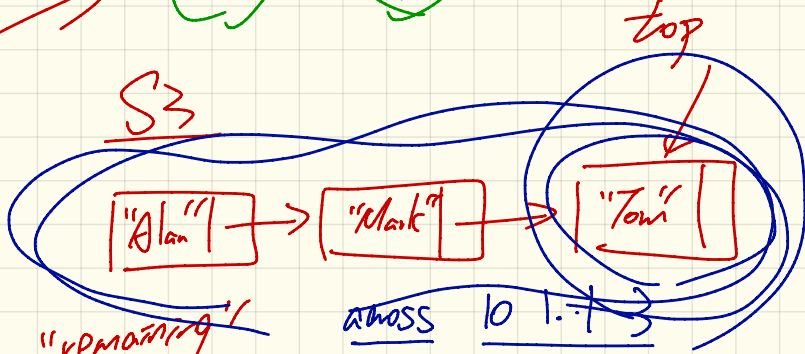
```
1  class
2    POINT_COLLECTOR_2
3  create
4    make
5  feature
6    points: LINKED_LIST[POINT]
7  feature
8    make do create points.make end
9  add_point (p: POINT)
10   do points.extend (p) end
11 get_point_at (i: INTEGER): POINT
12   do Result := points [i] end
13 end
```

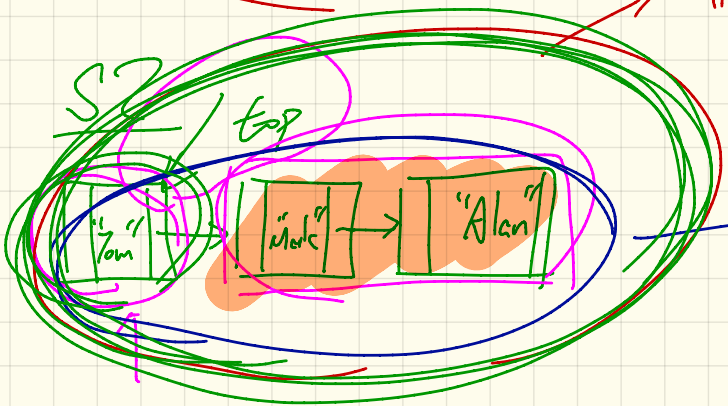
```
1  test_3: BOOLEAN
2    local pc: POINT_COLLECTOR_2 ; p: POINT ; i: INTEGER
3    do create pc.make
4       io.print ("Enter an integer:%N")
5       io.read_integer
6       if io.last_integer < 0 then pc := Void end
7       pc.add_point (create {POINT}.make (3, 4))
8       p := pc.get_point_at (0)
9       Result := p.x = 3 and p.y = 4
10   end
```











push (g)

remaining_unchanged :

across (2) 1..1 count as c

all

tmp[c.item] ~ (old tmp.d_t).[c.item]

end

Developing a LIFO STACK

- information hiding
- single-choice principle

when changing strategy, flaps to change stacks

client

change of strategy

1. imp is private but mentioned in postcondition
2. when changing the type of imp, client view affected.

```
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] 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
```

solution: have a common aspect to specify the contract with.

```
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
```

Using MATHMODELS Library

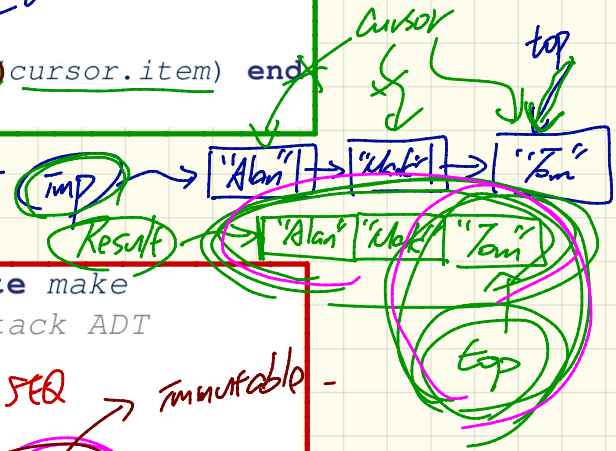
Implementing Abstraction Function

model query: convert from imp to SEQ.

```
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
```

prepend command

SEQ



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
```

conversion of the old imp to SEQ

immutable -

conversion of the new imp to SEQ

model ~ (old model).append(g) X not compile