Software Architecture

Contracting Patterns
Introduction

Plan for today

- The Iterator Pattern
- Contracting patterns: why and how?

Traversing without Iterator

```
-- Printing all elements of an arrayed list
from
    i := 1
until
    i > arrayed_list.count
loop
    print (arrayed_list.i_th (i))
    i :=i + 1
end

-- Printing all elements of a linked list
from
    cell := linked_list.first_element
until
    cell = Void
loop
    print (cell.item)
    cell := cell.right
end
```

Iterators

Allow traversing different containers uniformly

Classification:

By traversing mechanism:

- Linear
- Hierarchical
- ...

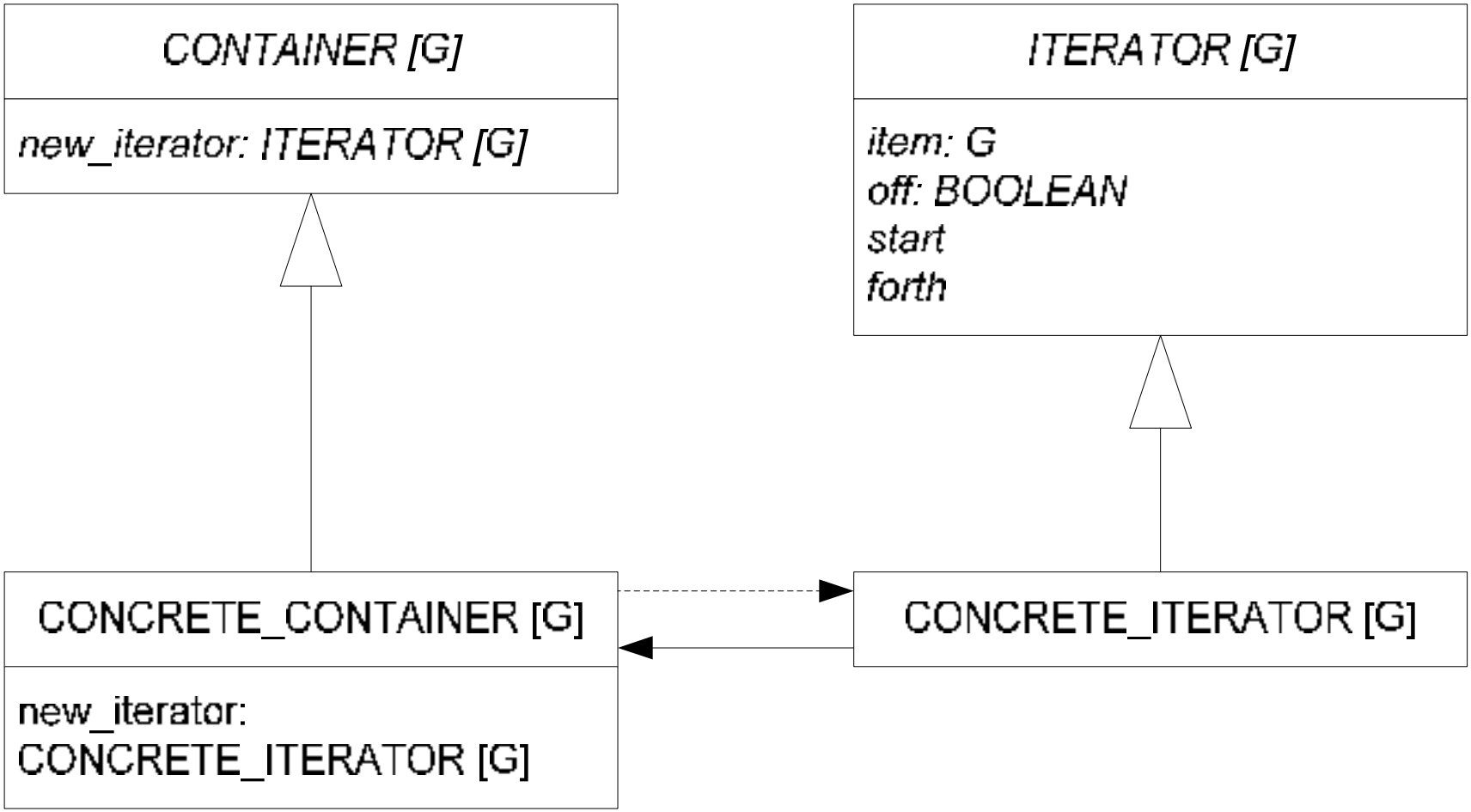
By initiator of iteration steps:

- Active (iterator)
- Passive (client)

By embedding into the container:

- External (separate object)
- Internal (iterator features are provided by the container)

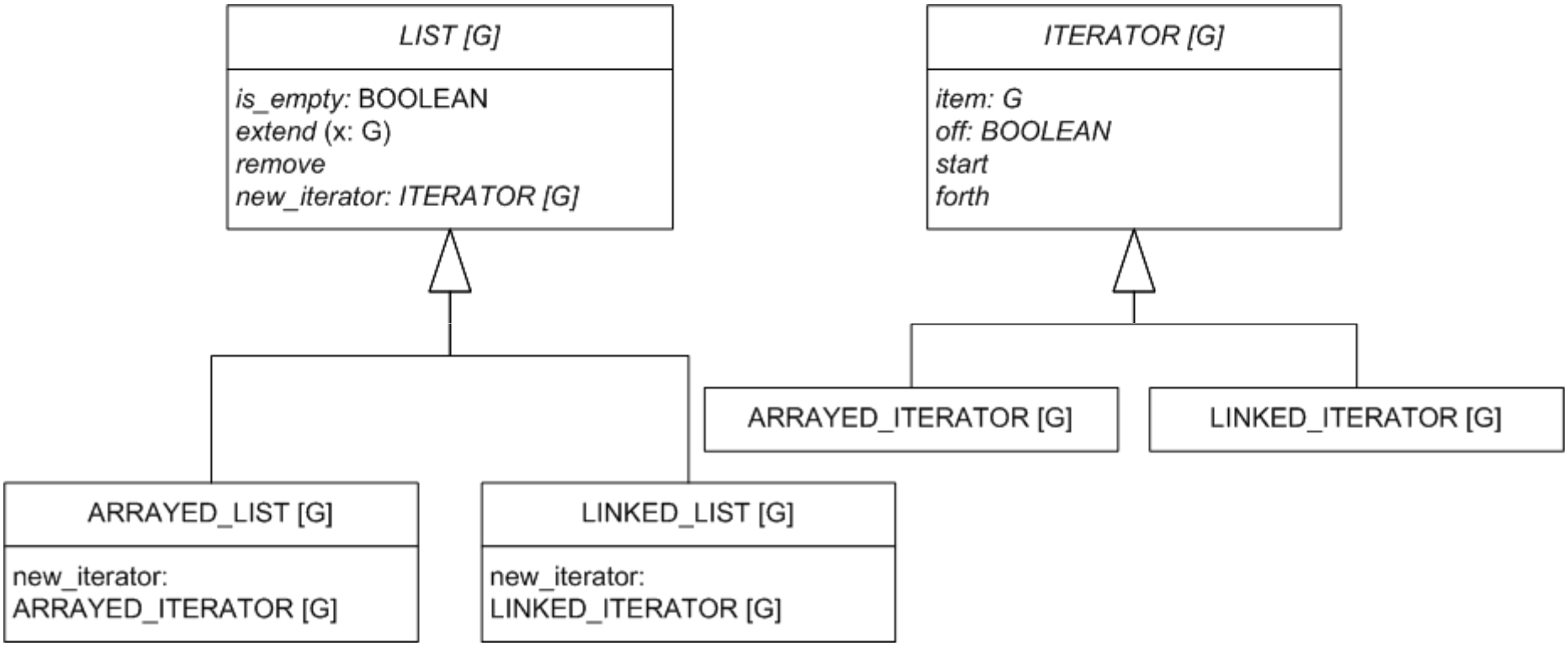
Passive External Linear Iterator



Traversing with Iterator

```
-- `list' can be any implementation of lists
from
    i := list.new_iterator
    i.start
until
    i.off
loop
    print (i.item)
    i.forth
end
```

Special case



Problems

- How would you implement the ARRAYED_LIST_ITERATOR?
- How would you implement the LINKED_LIST_ITERATOR?
- What happens with both of them if an element is inserted or removed at the front of the list while iteration is in progress?

Contracting patterns: Composite

```
class GRAPHIC

feature -- Basic operations
    draw is deferred end

feature -- Composite
    extend (g: GRAPHIC) is deferred end
            -- Add `g' as a child of `Current'

    prune (g: GRAPHIC) is deferred end
            -- Remove `g' from children of `Current'

    child (i: INTEGER): GRAPHIC is deferred end
            -- I-th child of `Current'
end
```

Concrete componenets

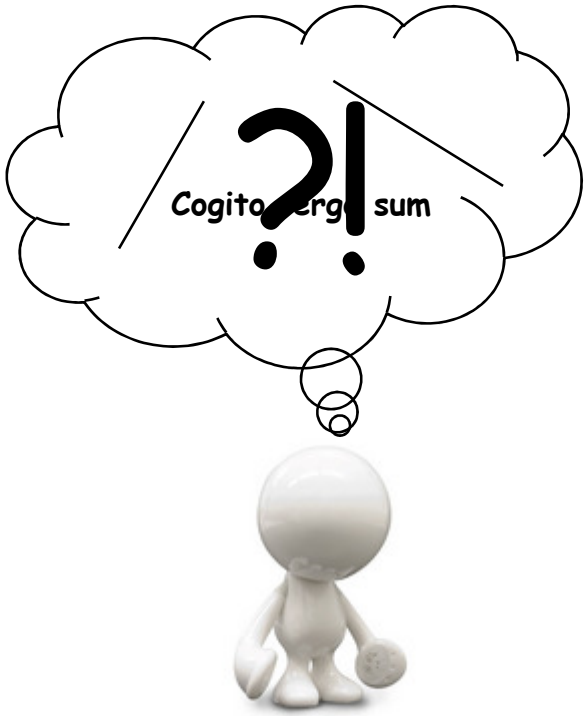
```
class LINE
feature -- Basic operations
    draw is do ... end
feature -- Composite
    extend (g: GRAPHIC) is do end
            -- Add `g' as a child of `Current'
...
end

class TEXT
feature -- Basic operations
    draw is do ... end
feature -- Composite
    extend (g: GRAPHIC) is do end
            -- Add `g' as a child of `Current'
...
end
```

Composite

```
class COMPOSITE

feature -- Basic operations
    draw is do children.do_all (agent {GRAPHIC}.draw) end
feature -- Composite
    extend (g: GRAPHIC) is do children.extend (g) end
            -- Add `g' as a child of `Current'
                                                    i
    prune (g: GRAPHIC) is do children.prune (g) end
            -- Remove `g' from children of `Current'

    child (i: INTEGER): GRAPHIC is do Result := children.i_th (i) end
            -- I-th child of `Current'
feature -- Implementation
     children: LIST [GRAPHIC]
end
```

Client

```
work_with_graphic (g: GRAPHIC) is
        -- Do some work with `g'
    do
        g.extend (create {LINE}.make (10))
        g.extend (create {TEXT}.make ("Cogito, ergo sum"))
        g.extend (create {LINE}.make (10))

        g.draw

        g.child (3).draw
    end
```



Welcome to Hell!

Call on Void target

Adding preconditions

```
class GRAPHIC
feature -- Status report
    extendible: BOOLEAN is deferred end
            -- Can new graphics be added?
    child_count: INTEGERis deferred end
            -- Number of children
feature -- Composite
    extend (g: GRAPHIC) is
            -- Add `g' as a child of `Current'
        deferred
        end
    prune (g: GRAPHIC) is
            -- Remove `g' from children of `Current'
        deferred
        end
    child (i: INTEGER): GRAPHIC is
            -- I-th child of `Current'
        deferred
        end
end
```

```
require extendible
```

```
require child_count > 0
```

```
require i >= 1 and i <= child_count
```

Adding postconditions and invariants

```
class GRAPHIC
feature -- Status report
    extendible: BOOLEAN is deferred end
            -- Can new graphics be added?
    child_count: INTEGERis deferred end
            -- Number of children
feature -- Composite
    extend (g: GRAPHIC) is
            -- Add `g' as a child of `Current'
        deferred
        end
    prune (g: GRAPHIC) is
            -- Remove `g' from children of `Current'
        deferred
        end
    child (i: INTEGER): GRAPHIC is
            -- I-th child of `Current'
        deferred
        end
end
```

```
invariant not extendible implies child_count = 0
```

```
require extendible
```

```
ensure child_count = old child_count + 1
```

```
require child_count > 0
```

```
ensure child_count = old child_count - 1
```

```
require i >= 1 and i <= child_count
```

```
ensure Result.parent = Current          ?
```