Iterator Pattern – Behavioural

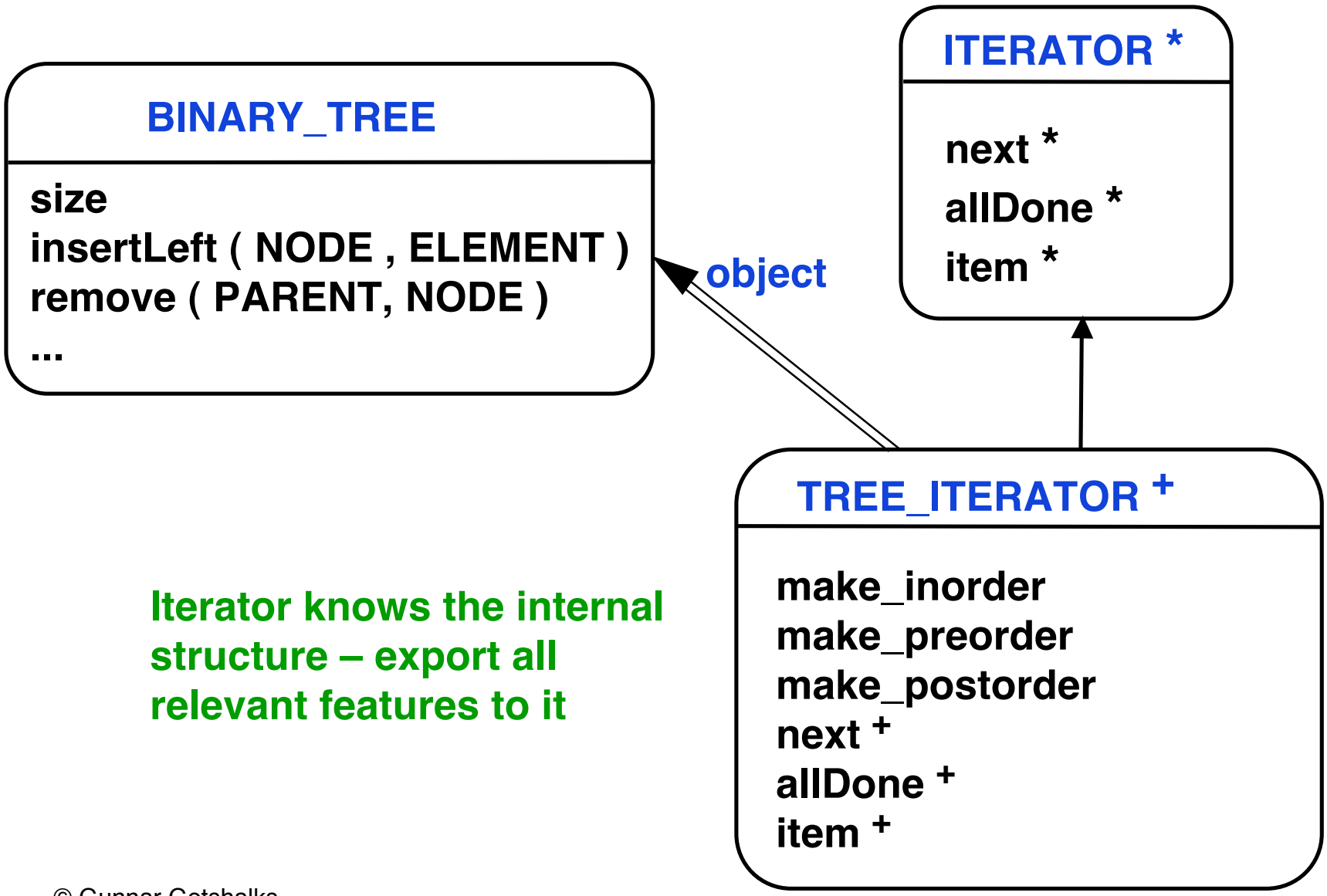
- Intent
  - » **Access elements of a container sequentially without exposing the underlying representation**

Motivation

- Be able to process all the elements in a container

- Different iterators can give different sequential ordering
  - » **Binary tree**
    - > **preorder, inorder, postorder**
  - » **Do not need to extend container interface**

Example Architecture

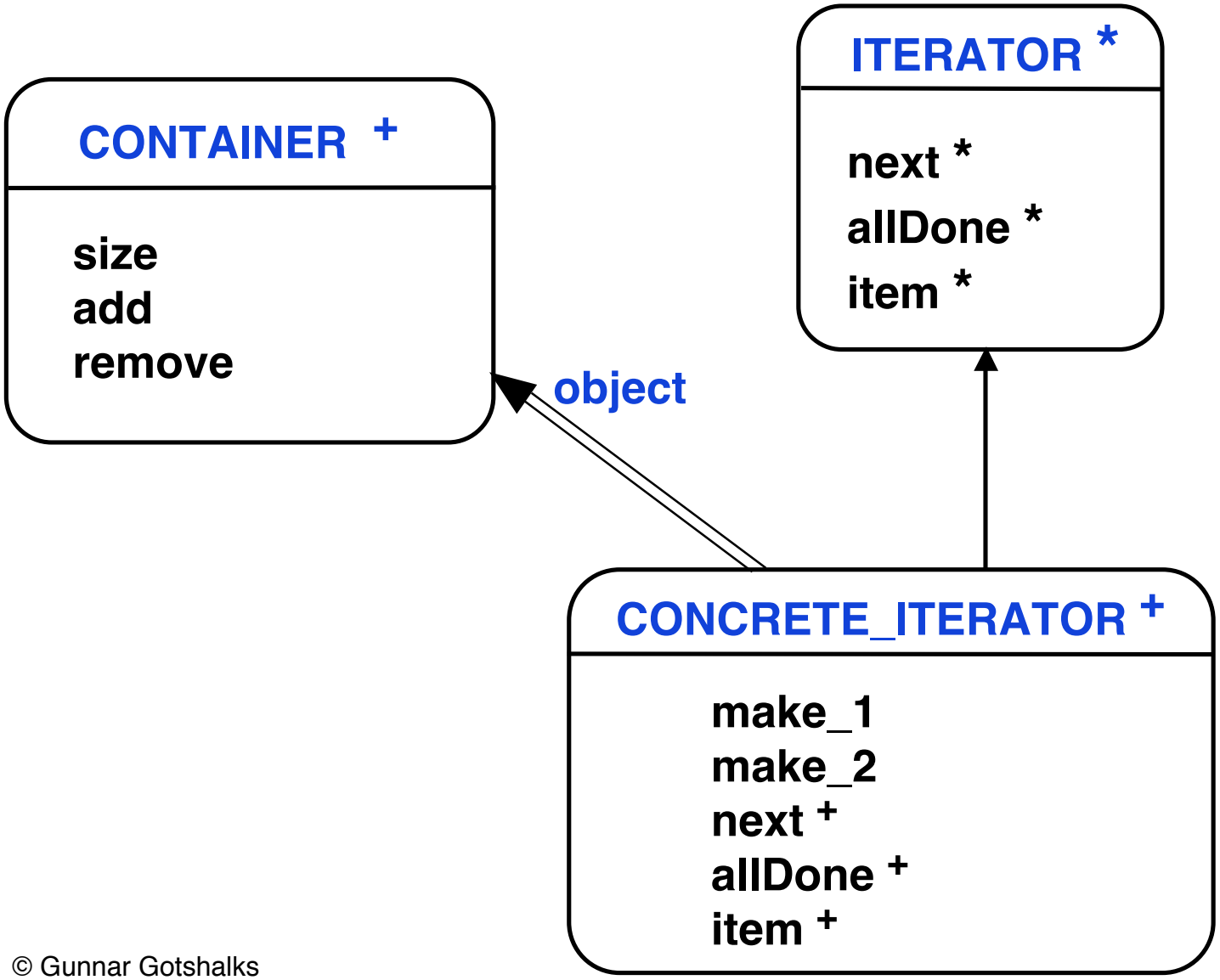


Iterator knows the internal structure – export all relevant features to it

Example Client

```
tree_items : TREE_ITERATOR
...
from create tree_items.make_inorder ( a_tree )
until tree_items.allDone
loop
    item := tree_items.item
    process ( item )
    tree_items.next
end
```

Abstract Architecture



Participants

- Iterator

  Defines interface for accessing and traversing a container's contents

- Concrete iterator

  - **Implements the iterator interface**
  - **Keeps track of the current position in the traversal**
  - **Determines next object in a sequence of the container's objects**

- Container

  Could provide a method to create an instance of an iterator

  Done in Java due to the poor export control

Applicability

- Access a container's contents without knowing about or using its internal representation

- Provide uniform interface for traversing a container's contents

  Support polymorphic iteration

Consequences

- Supports variations in the traversal of a container
  - » **Complex containers can be traversed in different ways**
    - **Trees and graphs**
  - » **Easy to change traversal order**
    - **Replace iterator instance with a different one**
- Iterators simplify the container interface
  - **Do not need iterator interface in container interface**
- Multiple simultaneous traversals
  - **Each iterator keeps track of its own state**

Implementation

- Can implement null iterators

  allDone is always True

  » **Useful in traversing tree structures**

  > **At each level use iterator over children**

  > **At leaf level automatically get a null iterator**

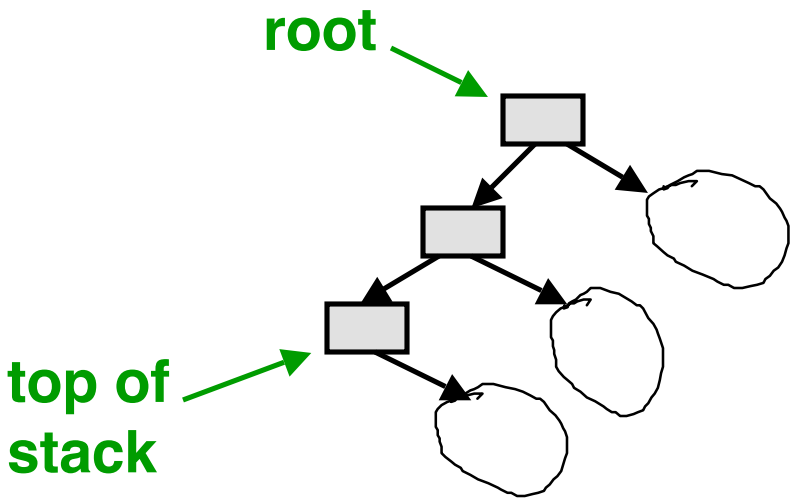
  > **No exceptions at the boundary**

Inorder Traversal Binary Tree

```
public Enumeration inOrderLRtraversal() {
    return new Enumeration() {
    Declare variables needed by the enumeration
        {
            Initialization program for the enumerator
        }

     public boolean hasMoreElements() {
            Provide the definition
        }
        public Object nextElement() {
            Provide the definition
        }
}
```

Inorder Traversal Binary Tree – 2

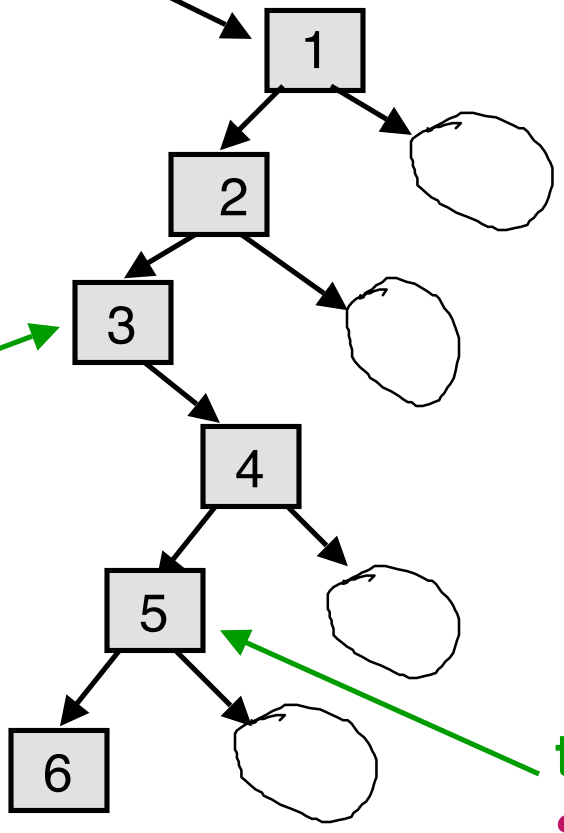
```
// Declare variables needed by the enumeration
    private Stack btStack = new Stack();

    { // Initialization program for the enumerator
      // Simulate recursion by programming our own
      // stack.  Need to get to the leftmost node as it
      // is first in the enumeration
        Node node = tree;

        while (node != null) {
            btStack.add(node);
            node = node.left;
         }
    }
```



Inorder Traversal Binary Tree – 3

```
public boolean hasMoreElements() {
    return ! btStack.isEmpty();
}
```

Inorder Traversal Binary Tree – 4



InitStack	after call 1	after call 2
3	6	5
2	5	4
1	4	2
	2	1
	1	

An enumerator is always 1 element ahead of the user

top of stack after second call to nextElement

top of stack after first call to nextElement

Inorder Traversal Binary Tree – 5

```
public Object nextElement() {
    if (btStack.isEmpty())
                        throw new NoSuchElementException();
    Node node = (Node) btStack.remove();
    Object result = node.datum;   // next data to return
    if (node.right != null) {          // Find next sequence node
       node = node.right;
       do { btStack.add(node);      // Get leftmost node in
         node = node.left;            // right subtree
       } while (node != null);
    }
    return result;
}
```

Notice that an enumerator is always 1 element ahead

What iterators do not do

- Create a copy, modify it and iterate over the copy
  - » **Sometimes efficiency may dictate a compromise**
- Iterators do not modify the original
  - » **Leave a "bookmark"**
  - » **Fold the page of a book**

Should iterate aggregate contents?

- If aggregation is complex (binary tree traversal) and multiple iterations are needed, execution can be more efficient
  - » **Aggregate contains pointers to original**
- Can be expensive in storage space for large collections
- If an object has multiple roles (occurs multiple times in the collection, e.g. Leila is the CEO and an engineer), then could lose a role with aggregation

Related Patterns

- Iterators are frequently applied to Composites
- Polymorphic iterators rely on factory methods to instantiate the appropriate Iterator subclass
- Memento is often used in conjunction with the Iterator pattern. An iterator can use a memento to capture the state of an iteration. The iterator stores the memento internally.

Iterator in Java API

- The example binary tree iterator in previous slides shows that the Java class Enumerator is an instantiation of the Iterator pattern

- Java also has the class Iterator with the following methods
  - » **next(), hasNext() and remove()**