InterBit
Training & Consulting Ltd.

Iterator Pattern

Chapter Content

InterBit
Training & Consulting Ltd.

- The Iterator Pattern
- Iterator – UML Diagram
- Standard java iterators
- A Type-filtering Iterator
- Further considerations
- Concurrent changing

Intent

- Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
- The abstraction provided by the iterator pattern allows you to:
  - modify the collection implementation without making any changes outside of collection.
  - It enables you to create a general purpose GUI component that will be able to iterate through any collection of the application.

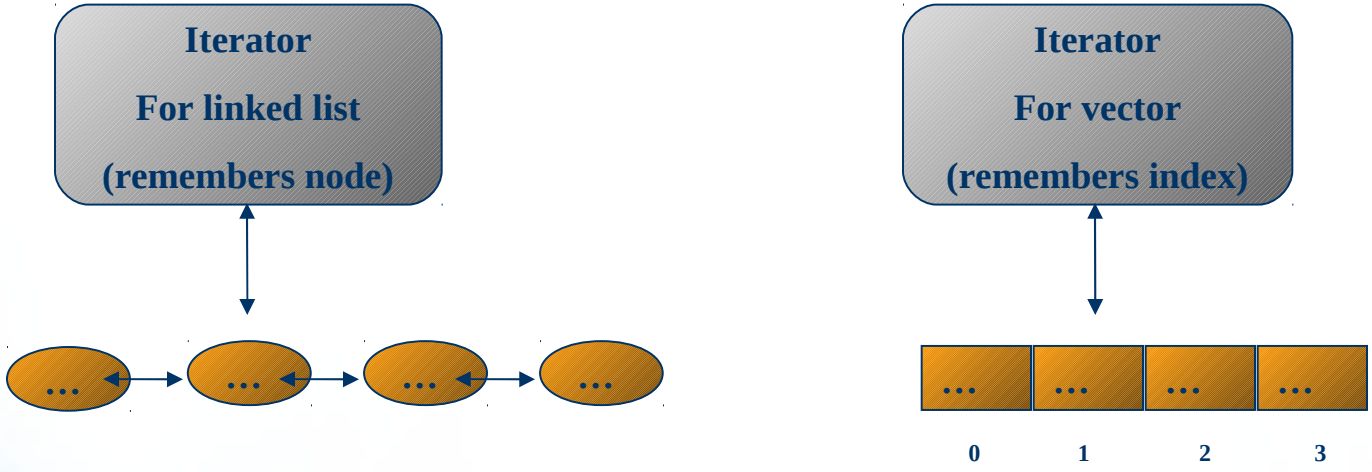
The Iterator Pattern

InterBit Training & Consulting Ltd.

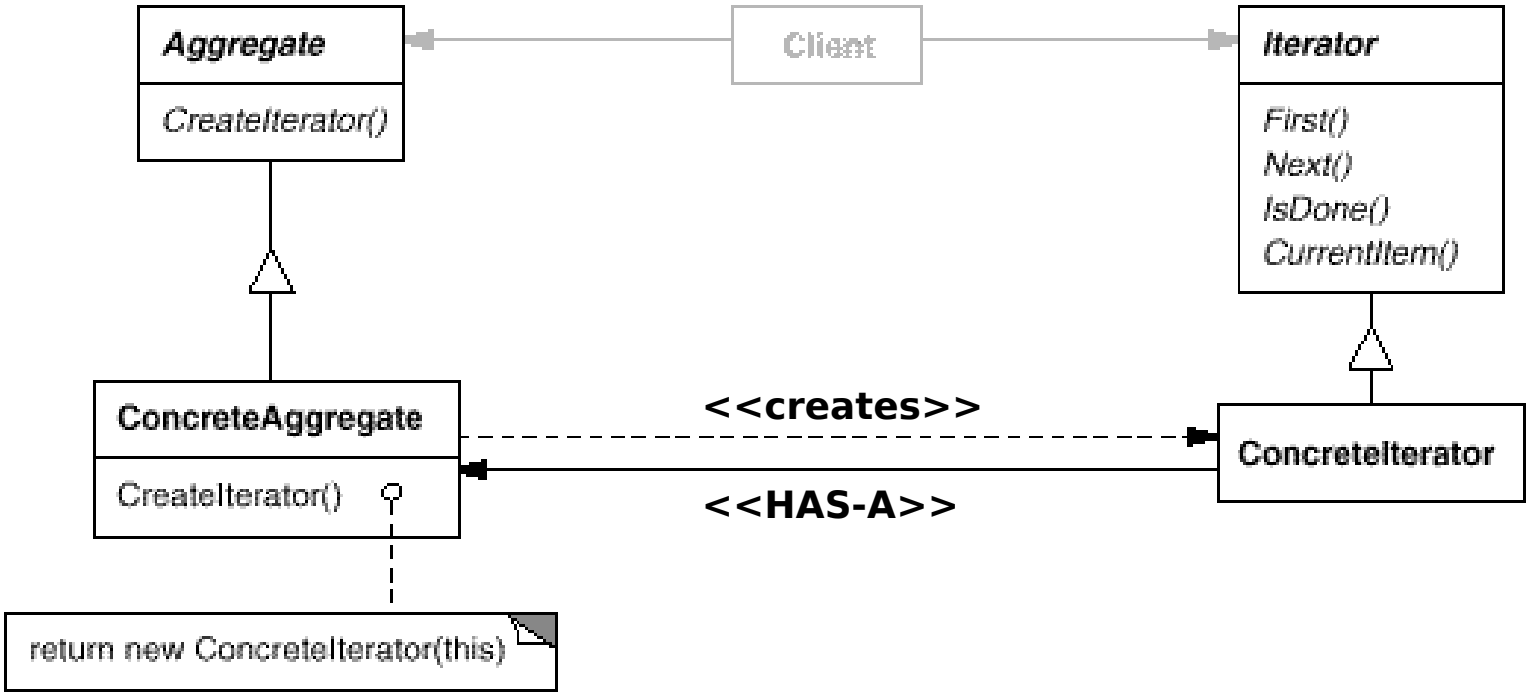
- A simple, frequently used pattern.
- Allows to **iterate through a collection of data without exposing** the internal representations of that collection.
- Iterator variation:
  - Iterators may do some processing on elements.
  - Two-way iterators.
  - Iterators that skip some elements.

The Iterator Pattern

- Different collections will have different implementations of Iterator.
- but all iterators have the same interface.



Iterator – UML Diagram



Aggregate is a class holding a collection of objects.
When client needs to iterate over this collection, it asks for an Iterator.

Different concrete implementations of Aggregate may have different implementation for Iterator, but all iterators have a common interface for obtaining current element and for progressing.

Standard java iterators

InterBit Training & Consulting Ltd.

```
List  lst = new ArrayList();  // or new LinedList()
lst.add(new Integer(10));
lst.add(new Double(20));
Iterator it = lst.iterator() ;
while(it.hasNext()) {
     Number num= (Number)it.next();
     System.out.println(num);
}
```

- **ListIterators** are two-way (next/previous).

A Type-filtering Iterator

InterBit Training & Consulting Ltd.

```
// iterator that only returns elements of a given type, skipping all others:
class TypedIterator implements Iterator<Serializable> {
    private Iterator<Serializable> impl; // Concrete iterator (for arraylist/linkedlist…)
    private Class<?> type;               // Return objects of this type (skip the rest)
    private Serializable datum;          // Object to return when  next()  is called

    public TypedIterator(Iterator<Serializable> impl, Class<?> type) {
        this.impl = impl;
        this.type = type;
        findNext();
    }
    public boolean hasNext() {
        return (datum != null);
    }

    public Serializable next() {
        Serializable result = datum;
        findNext();
        return result;
    }
```

.A Type-filtering Iterator – cont

```
    public void remove() {
        throw new UnsupportedOperationException();
        // difficult, unless impl has a  previous()  method
    }
    private void findNext() {
        datum = null;
        while (impl.hasNext()) {
            Serializable tmp = impl.next();
            if (tmp.getClass() == type) {
                datum = tmp;
                break;
            }
        }
    }
}
```

Usage:

```
Collection<Serializable> collection = new LinkedList<Serializable>();
collection.add("1");              // populate with any Serializable (String)
collection.add(2.25D);            // populate with any Serializable (Double)
Iterator it=new TypedIterator(collection.iterator(),  String.class) ;  // only strings
```

Problems and implementation

InterBit Training & Consulting Ltd.

Iterator and multithreading

- Several problems may appear when collections are added from different threads.
  - Step one: the collection return a new iterator. Usually this step is not affected when it is used in multithreading environments because it returns a new iterator object.
  - Step two: The iterator is used for iterating through the objects. Since the iterators are different objects this step is not a problematic one in multithreading environments.
- So far, the iterator does not raise special problems when a collection is used from different threads. The iterator does not raise special problems when the collection is used from different threads **as long the collection is not changed**.

Problems and implementation

InterBit Training & Consulting Ltd.

Iterator and multithreading (continued)

- Let's analyze each case:
  - A new element is added to the collection (at the end). The iterator should be aware of the new size of the collection and to iterate till the end.
  - A new element is added to the collection before the current element. In this case all the iterators of the collection should be aware of this.
- The same actions should occur when an element is removed from the collection. The iterators should be aware of the changes.

Problems and implementation

InterBit Training & Consulting Ltd.

Iterator and multithreading (continued)

- The main task when creating a multithreading iterator is to create a robust iterator (that allows insertions and deletions without affection transversal)
- Then the blocks which are changing or accessing resources changed by another thread have to be synchronized

Problems and implementation

InterBit Training & Consulting Ltd.

Concurrent Modification

- An important, non-trivial issue:
- **What happens if the data-structure is modified while Iterator is attempting to iterate over it?**
- With **java.util.Iterator**:
  - Not guaranteed to function correctly on such conditions (usually throws an exception).
  - *Is* guaranteed to work if changes are made through the Iterator, e.g: *Iterator.remove().*

Problems and implementation

InterBit Training & Consulting Ltd.

External vs. internal iterators.

- **External** Iterators - when the iteration is controlled by the collection object we say that we have an external Iterator.
- In the following example an external iterator is used:

```
// using iterators for a clloection of String objects:
// using in a for-each loop (syntax available from java 1.5 and above)
Iterator name = options.iterator();
for (Object item : options)
    System.out.println(((String)item));
```

Problems and implementation

InterBit Training & Consulting Ltd.

External vs. internal iterators (continued)

- **Internal** Iterators - When the iterator controls it we have an internal iterator. Implementing and using internal iterators is really difficult. When an internal iterator is used it means that the code to be ran is delegated to the aggregate object.
- For example in languages that offer support for this is easy to call internal iterators:

```
collection do: [:each | each doSomething] (Smalltalk)
```

- The main idea is to pass the code to be executed to the collection. Then the collection will call internally the *doSomething* method on each of the components.

Problems and implementation

InterBit Training & Consulting Ltd.

External vs. internal iterators (continued)

- In C++ it's possible to send the *doMethod* method as a pointer. In C# .NET or VB.NET it is possible to send the method as a delegate

- In Java, the **Command** design pattern has to be used to create a base Interface with only one method (*doSomething*). Then the method will be implemented in a class which implements the interface and the class will be passed to the collection to iterate.

Problems and implementation

InterBit Training & Consulting Ltd.

Who defines the traversal algorithm?

- The algorithm for traversing the aggregate can be implemented in the iterator or in the aggregate itself.
- When the traversal algorithm is defined in the aggregate, the iterator is used only to store the state of the iterator.
  - This kind of iterator is called a cursor because it points to the current position in the aggregate.

Problems and implementation

InterBit Training & Consulting Ltd.

Who defines the traversal algorithm? (continued)

- The other option is to implement the traversal algorithm in the iterator. This option offers certain advantages and some disadvantages.
- For example it is easier to implement different algorithms to reuse the same iterators on different aggregates and to subclass the iterator in order to change its behavior.
- The main disadvantage is that the iterator will have to access internal members of the aggregate
  - In Java and .NET this can be done, without violating the encapsulation principle, by making the iterator an inner class of the aggregate class

Problems and implementation

InterBit Training & Consulting Ltd.

Robust Iterators

- Can the aggregate be modified while a traversal is ongoing?
- An iterator that allows insertion and deletions without affecting the traversal and without making a copy of the aggregate is called a robust iterator.
- A robust iterator will make sure that when elements are added or removed from an aggregate during iteration; elements are not accessed twice or ignored.

Problems and implementation

InterBit Training & Consulting Ltd.

Robust Iterators (continued)

- Lets' say we don't need a robust iterator:
- If the aggregate can not be modified (because the iteration is started), it should be made explicitly, meaning that the client should be aware of it.
- We can just return a false value what an element is added to the collection stating that the operation has failed
  - or we can throw an exception

Problems and implementation

InterBit Training & Consulting Ltd.

Robust Iterators (continued)

- An alternative solution is to add functions to change the aggregate in the iterator itself. For example we can add the following methods to our iterator:

```
boolean remove();
boolean insertAfer();
boolean insertBefore();
```

- In this case the iterator handles the changes of the aggregate and the operation to change the iteration should be added to the iterator interface or base class - **not** to the implementation only, in order to have a general mechanism for the entire app.

Problems and implementation

InterBit Training & Consulting Ltd.

Mechanism provided by the language

- The iterator pattern can be implemented from scratch in Java or .NET, but there is already built-in support for Iterator Pattern:
  - Iterator/Collection in JAVA
  - and IEnumerator/IEnumerable in .NET