Iterator Design Pattern

Jim Fawcett
CSE776 – Design Patterns
Fall 2014

Intent

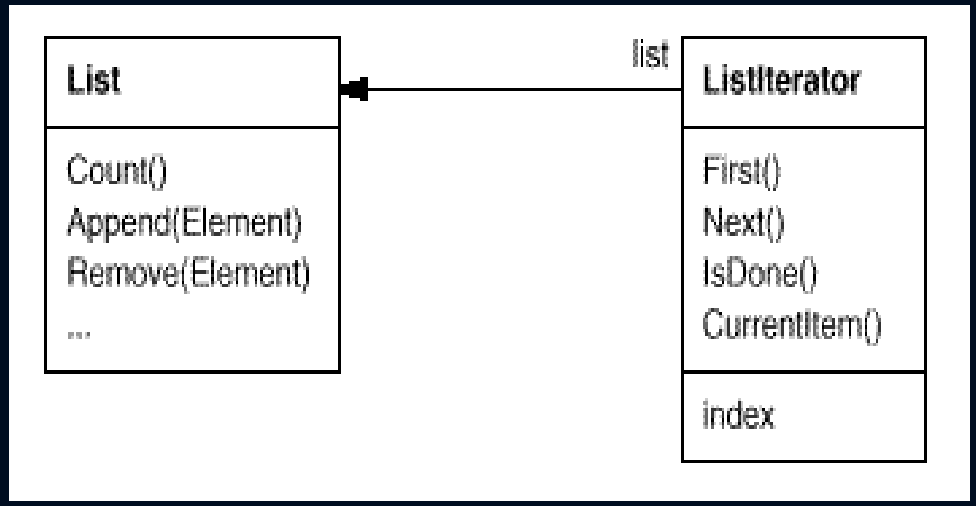
- Provide a way to access elements of an aggregate object without exposing its underlying representation
- Traverse many different data structures in a uniform way
- Place “bookmarks” in a large collection of data

Motivation

- Abstract the traversal of different data structures so that algorithms can interface with each transparently

Motivation

- Instance of List class is traversed with a ListIterator
- ListIterator provides an interface for accessing List's elements
- An iterator is responsible for keeping track of the current element



Forces

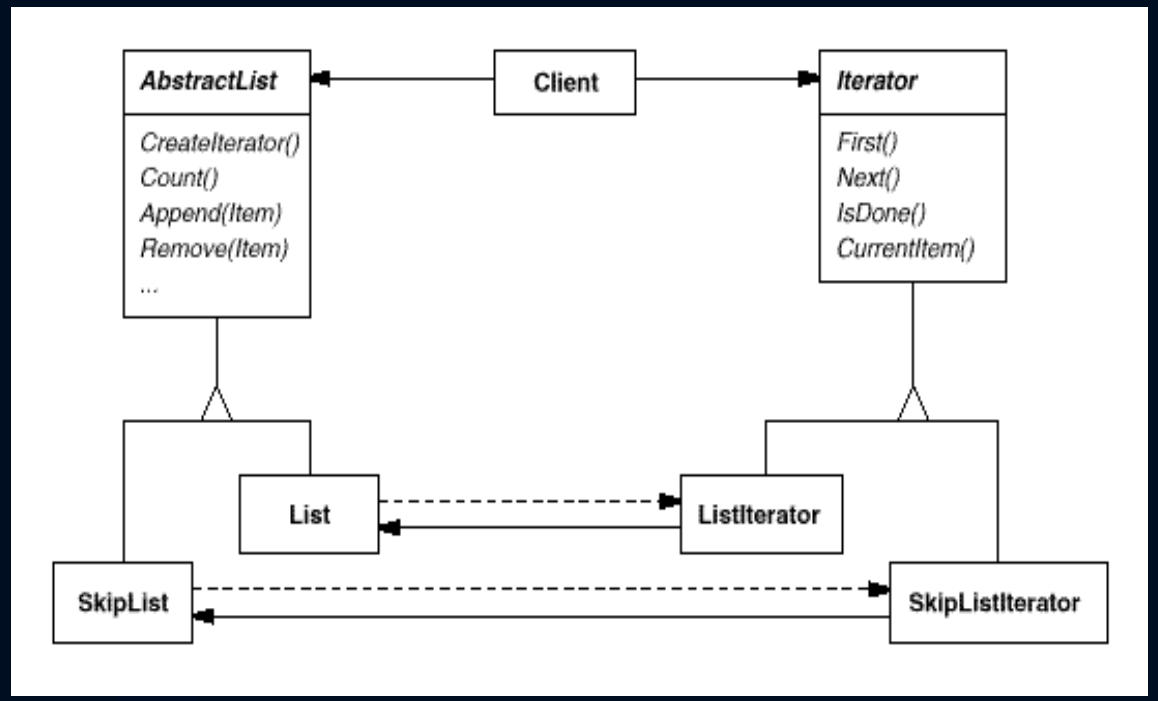
- Separating traversal mechanism from the aggregate object lets us define iterators for different traversal policies without enumerating them in List interface
  - Forward iteration, reverse iteration, preorder, postorder, …
- The iterator and aggregate are coupled
  - Iterator needs to know aggregate structure
  - Aggregate may need to give iterator special permissions, e.g., friend status in C++

Polymorphic Iteration

- We want the code using iteration to be independent of the type of aggregate being used.
  - C++ Standard Template Library (STL) uses, by convention, a standard interface for iteration based on pointer syntax:
    - ++iter, *iter, iter->fun(arg)
  - .Net provides IEnumerator interface
    - Current property
    - MoveNext, Reset methods

Motivation Structure

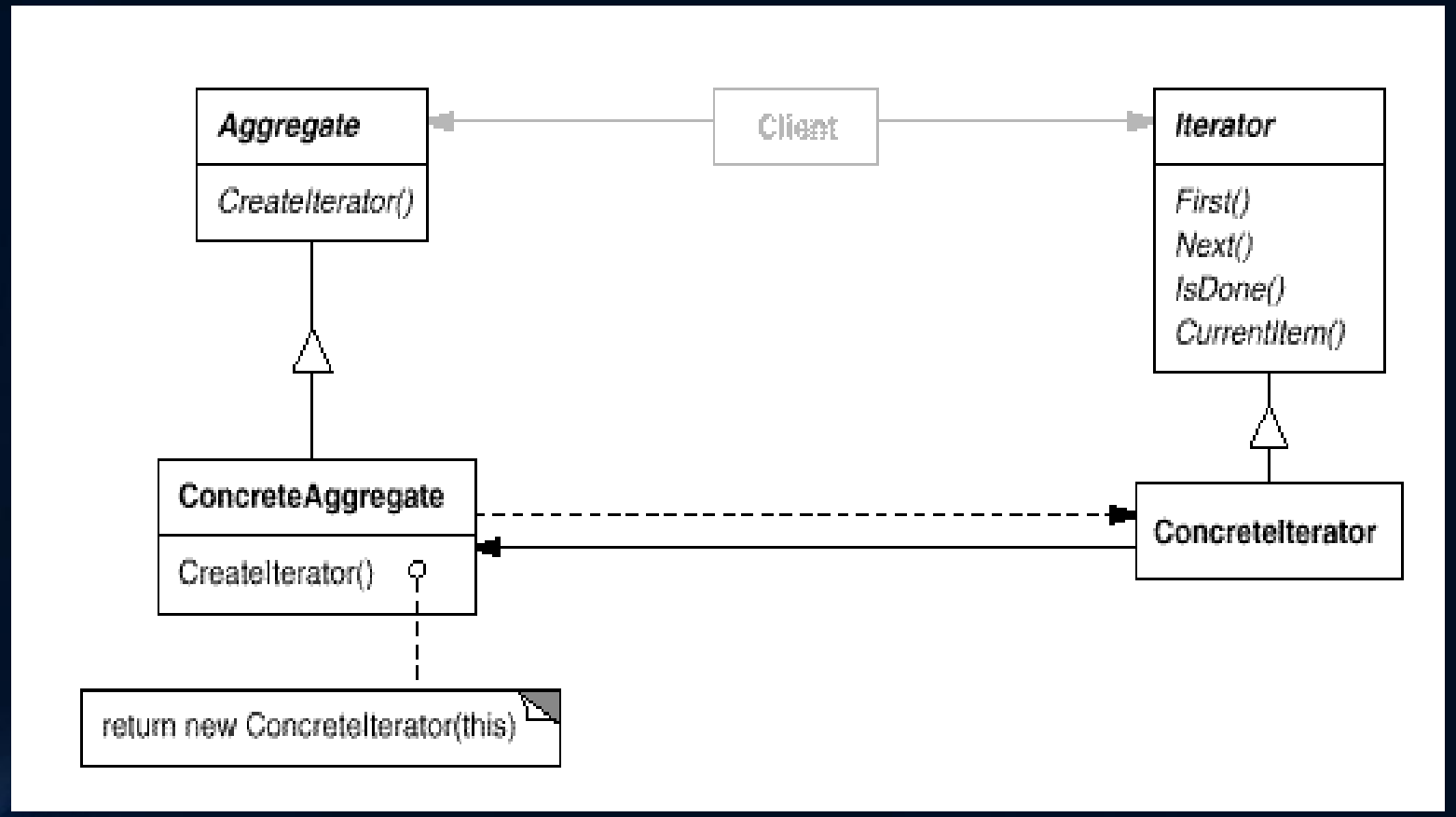
- CreateIterator() is a factory method
- We use it to let client ask for an iterator appropriate for the aggregate being used
- .Net uses the IEnumerable.GetEnumerator()
- C++ uses Container::Iterator



Applicability

- Use the Iterator Pattern to:
  - Access an aggregate object's contents without exposing its internal representation
  - Support multiple traversals of aggregate's objects
  - Provide a uniform interface for traversing different aggregate structures

Structure


Aggregate
CreateIterator()
Client
Iterator
First()
Next()
IsDone()
CurrentItem()
ConcreteAggregate
CreateIterator()
ConcreteIterator
return new ConcreteIterator(this)

Participants

- Iterator
  - Defines interface for accessing and traversing elements
- ConcreteIterator
  - Implements the Iterator interface
  - Keeps track of current position in traversal
- Aggregate
  - Defines interface for creating an Iterator object
- ConcreteAggregate
  - Implements the Iterator creation interface to return an instance of its associated ConcreteIterator

Collaborations

- ConcreteIterator keeps track of the current object in the aggregate
- Makes current element accessible to client
- Computes the succeeding object in traversal

Consequences

- Supports variations in the traversal of an aggregate
- Simplifies the Aggregate interface
- More than one traversal can be pending on an aggregate

Implementation

- Who controls iteration?
  - Client – successive steps
  - Aggregate – DFS on Tree
- Who defines the traversal algorithm?
  - Preorder, postorder
- How robust is the iterator?
- Additional Iterator operations
- Using polymorphic iterators in C++
- Iterators may need priviledged access
- Iterators for composites
- Null iterators

Sample Code

- Iterator Skeleton
- STL Iterator Examples
- .Net Iterator Examples

Known Uses

- C++
  - STL <vector>, <list>, <string>, <iostreams>, …
  - Arrays using native pointers
- .Net
  - System.Collections, System.Collections.Generic
- Java
  - ListIterator<E>, XMLEventReader