Iterator

A behavioral pattern

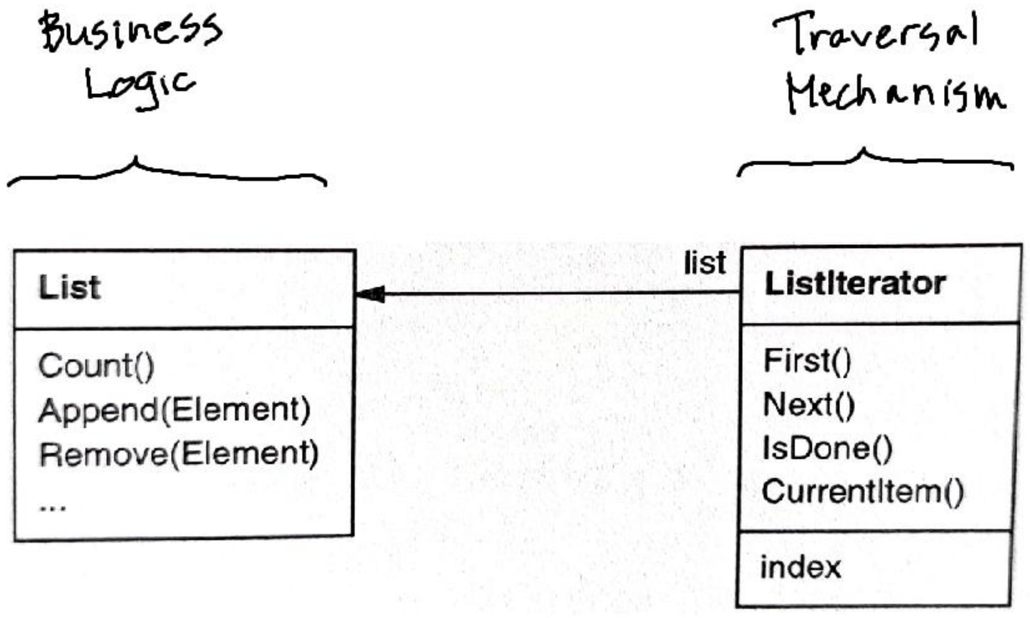
Learning goals

1. Learn the idea, structure, and Java implementation of the Iterator design pattern.
2. Learn to apply the Iterator DP in your own programming.

Idea of Iterator

- The Iterator design pattern provides a standard way to access and traverse elements in a collection.
- It hides the underlying structure of the collection being accessed and traversed.
- Goals:
  1. **Abstraction**: Separate the traversal mechanism from the business logic of the collection.
  2. **Uniformity**: Provide a uniform interface for traversing different types of collections.
  3. **Control**: Encapsulate the details of accessing and traversing the collection.

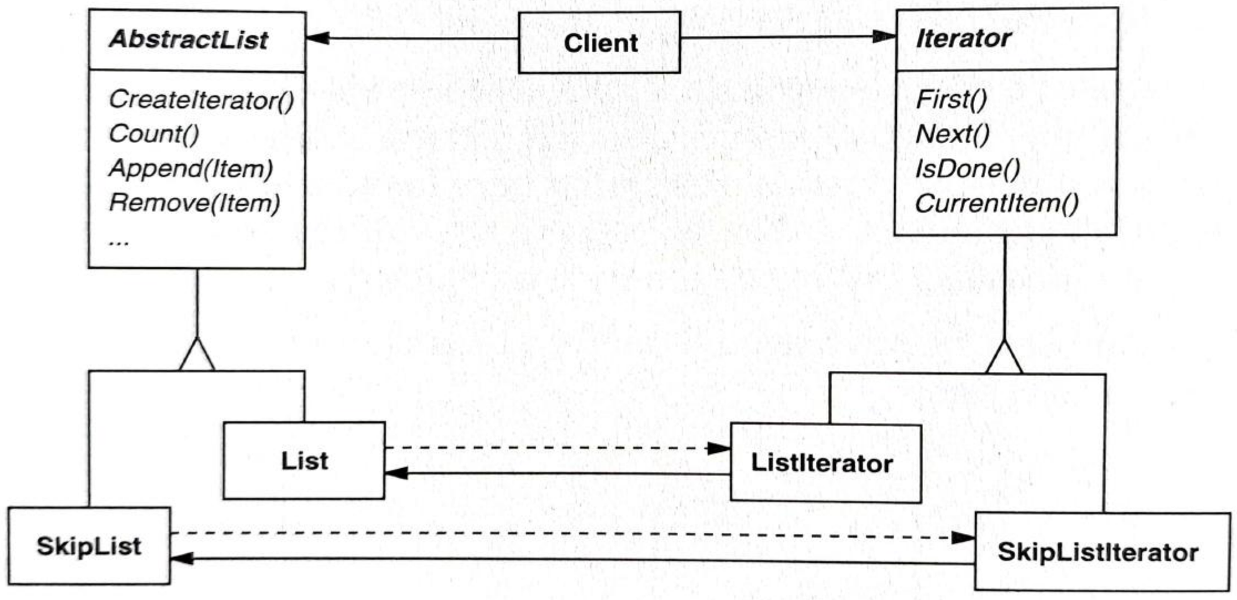
Example: List



- Managing the list content is a separate problem of traversing and accessing the list.
- They are worth separating from each other.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 257

Example: List



- Each List implementation may need to have its own technical implementation for traversing the list.
- This should, however, be hidden from the client.
- Solution: A List is able to create its Iterator.
- Lists and their Iterators are accessed via an itnterface.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 258

General structure

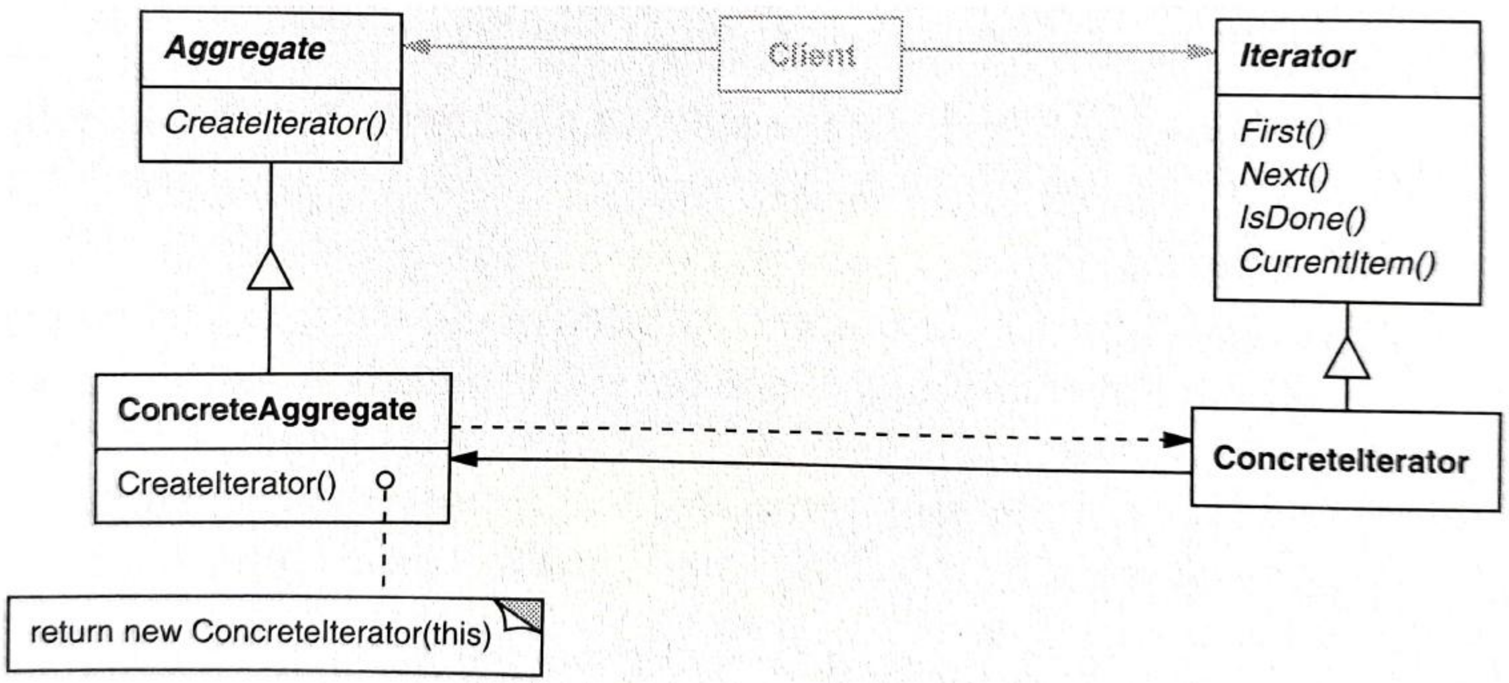


Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 259

Roles

- **Aggregate**[1] declares an interface for creating an Iterator object.
- **ConcreteAggregate**[1] implements the Iterable interface and returns a ConcreteIterator instance that can iterate over the collection.
- **Iterator** provides an interface for accessing and traversing elements, typically with methods hasNext() and next().
- **ConcreteIterator** implements the Iterator interface. Manages the iteration state and ensures the correct traversal over the ConcreteAggregate.
  - Keeps track of its current position in iterating the Aggregate.
- **Client** uses the Aggregate to obtain an Iterator. It then uses this iterator to traverse the elements in the ConcreteAggregate.

1) Aggregate and ConcreteAggregate are the original terms used by Gamma et al. They are often called Iterable and ConcreteIterable instead. This highlights the fact that the Iterator DP can be used to iterate not only stored objects but also dynamically created sequences.

Java’s Interfaces and the Iterator DP

- The Java Iterator
  - Adheres to Gamma’s Iterator Design Pattern.
  - **java.util.Iterator** acts as the ready-to-use Iterator interface.
  - Provides a consistent method to traverse elements with hasNext() and next().
- The Java Collection
  - Goes beyond the simple Iterator pattern.
  - **java.util.Collection** combines the roles of managing business logic and providing a traversal mechanism.
    - Is feature-rich, requiring implementations of a comprehensive set of methods for manipulating a collection.

Java's Interfaces and the Iterator DP

- The Collection interface may be too complex for simple needs.
- Implementing it can be burdensome for a small application
  - It requires a large number of methods to be implemented.
- For pure iteration needs, Iterator is sufficient and straightforward.
- When full collection functionality is needed, Collection provides a robust framework.

Practical issues

- The Iterator pattern streamlines the process of sequential access to elements within a collection or sequence.
  - Avoids the need for complex loop constructs and index management within client code.
  - Eliminates the risk of errors such as off-by-one mistakes or boundary overruns.
- Provides the flexibility to define various iteration strategies.
  - The Aggregate may offer a choice of iterators.
  - Examples: A forward iterator, a backward iterator, a specialized filtering iterator etc.
- Once the client retrieves an iterator from a collection, it interacts solely with the iterator.
  - The client remains agnostic to the internal organization of the collection.
- Modifying the collection which is being iterated may have unexpected side effects to their ongoing iteration.
  - Iterators can be implemented with such robustness in mind. This increases code complexity.