

# Iterator Pattern

Definition

Real-world Usage

Code Example

The Problem: Data exposure to the Client

Class Diagram

Structure of the Iterator Design Pattern

Implementation(Example: Library)

Output

▼ Resources

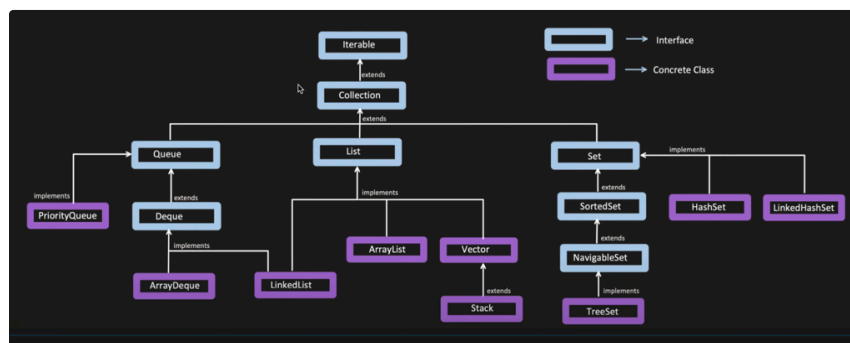
- [41. All Behavioral Design Patterns | Strategy, Observer, State, Template, Command, Visitor, Memento](#)
- [33. Iterator Design Pattern Explained with Example | Low Level Design](#)

## Definition

The Iterator design pattern is one of the behavioral design patterns that **provides a way to access elements of a Collection sequentially** without exposing the underlying representation of the collection.

## Real-world Usage

In the Java Collections Framework, the collection interface includes methods, such as `iterator()`, that allow a client to obtain an Iterator object from any collection that implements it. All collection classes have iterator implementation, e.g., (`ArrayList.iterator()`, `HashSet.iterator()`, `TreeSet.descendingIterator()`, etc).



Java Collections Framework

## Collection Class – ArrayList

```

public class ArrayList<E> extends AbstractList<E>
    implements List<E>, RandomAccess, Cloneable, java.io.Serializable
{
    // ...

    public interface Iterator<E> {
        Returns true if the iteration has more elements. (In other words, returns true if next would
        return an element rather than throwing an exception.)
        Returns true if the iteration has more elements
        @Contract(pure = true)
        boolean hasNext();

        Returns the next element in the iteration.
        Returns: the next element in the iteration
        Throws: NoSuchElementException -- if the iteration has no more elements
        @Contract(mutates = "this")
        E next();
    }

    // ...

    private class Itr implements Iterator<E> {
        int cursor; // Index of next element to return
        int lastRet = -1; // Index of last element returned; -1 if no such
        int expectedModCount = modCount;

        // prevent creating a synthetic constructor
        Itr() {}

        public boolean hasNext() {
            return cursor != size;
        }

        @SuppressWarnings("unchecked")
        public E next() {
            checkForComodification();
            int i = cursor;
            if (i >= size)
                throw new NoSuchElementException();
            Object[] elementData = ArrayList.this.elementData;
            if (i >= elementData.length)
                throw new ConcurrentModificationException();
            cursor = i + 1;
            return (E) elementData[lastRet = i];
        }
    }
}

```

ArrayList implementation of Iterator

Choose Implementation of Iterator (226 found)

- ☐ Itr in AbstractList (java.util)
- ☐ Itr in ArrayBlockingQueue (java.util.concurrent)
- ☒ Itr in ArrayList (java.util)
- ☐ Itr in ConcurrentLinkedDeque (java.util.concurrent)
- ☐ Itr in ConcurrentLinkedQueue (java.util.concurrent)
- ☐ Itr in DelayQueue (java.util.concurrent)
- ☐ Itr in DelayedWorkQueue in ScheduledThreadPoolExecutor (java.util.concurrent)
- ☐ Itr in EventSetImpl (com.sun.tools.jdi)
- ☐ Itr in LinkedBlockingDeque (java.util.concurrent)
- ☐ Itr in LinkedBlockingQueue (java.util.concurrent)
- ☐ Itr in LinkedTransferQueue (java.util.concurrent)
- ☐ Itr in PriorityBlockingQueue (java.util.concurrent)
- ☐ Itr in PriorityQueue (java.util)
- ☐ Itr in Vector (java.util)
- ☒ KeyIterator in ConcurrentHashMap (java.util.concurrent)
- ☐ KeyIterator in ConcurrentSkipListMap (java.util.concurrent)
- ☐ KeyIterator in EnumMap (java.util)
- ☒ KeyIterator in HashMap (java.util)
- ☐ KeyIterator in IdentityHashMap (java.util)
- ☐ KeyIterator in TreeMap (java.util)
- ☐ KeyIterator in WeakHashMap (java.util)
- ☒ ValueIterator in ConcurrentHashMap (java.util.concurrent)
- ☐ ValueIterator in ConcurrentSkipListMap (java.util.concurrent)
- ☐ ValueIterator in EnumMap (java.util)
- ☐ ValueIterator in HashMap (java.util)
- ☐ ValueIterator in IdentityHashMap (java.util)
- ☐ ValueIterator in TreeMap (java.util)
- ☐ ValueIterator in WeakHashMap (java.util)

Iterator Interface Implementations

## Code Example

```

1 // Java Collections Usage Example
2 public class LinkedHashSetExample {
3     public static void main(String[] args) {
4         Set<Integer> intSet = new LinkedHashSet<>();
5         intSet.add(2);
6         intSet.add(77);
7         intSet.add(82);
8         intSet.add(63);
9         intSet.add(5);
10
11         // Common to all Collection Classes
12         Iterator<Integer> iterable = intSet.iterator();
13         while (iterable.hasNext()) {
14             int value = iterable.next();
15             System.out.println(value);
16         }
17     }
18 }

```

## The Problem: Data exposure to the Client

```

1 public class Book {
2     private final String title;
3     private final String author;
4     private final String isbn;

```

```

5
6     public Book(String tittle, String author, String isbn) {
7         this.title = tittle;
8         this.author = author;
9         this.isbn = isbn;
10    }
11
12    public static List<Book> getBooks() {
13        List<Book> books = List.of(
14            new Book("To Kill a Mockingbird", "Harper Lee", "978-
150-74-7356-5"),
16            new Book("The Great Gatsby", "F. Scott Fitzgerald",
17"778-0-24-7156-5"),
18            new Book("The Catcher in the Rye", "J.D. Salinger",
19"333-0-28-7446-8"),
20            new Book("The Hobbit", "J.R.R. Tolkien", "783-0-14-
211951-8"),
22            new Book("Rich Dad Poor Dad", "Robert Kiyosaki", "183-
230-12-1491-8"),
24            new Book("Pride and Prejudice", "Jane Austen", "289-0-
2512-1678-8")
26        );
27        return books;
28    }
29
30    @Override
31    public String toString() {
32        return "Book [" + "Title='" + title + ", Author='" + author +
33", age=" + "ISBN=" + isbn + "];"
34    }
35 }

```

```

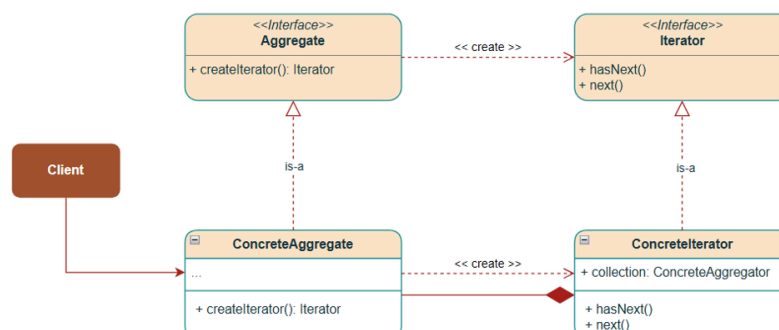
1 public class Client {
2     public static void main(String[] args) {
3         System.out.println("\n##### Problem without Iterator Pattern
4 Demo #####");
5         // Client has access to the entire book list in the library
6         List<Book> bookList = Book.getBooks();
7         for (Book book : bookList) {
8             System.out.println(book);
9         }
10    }

```

Problems with the above code are that:

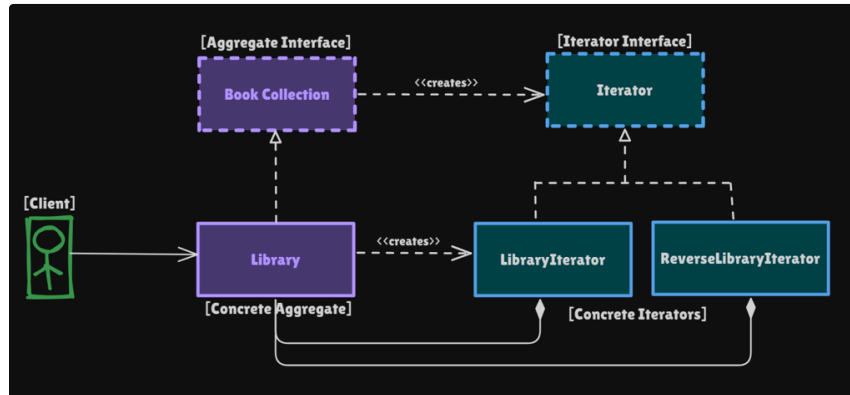
- No data encapsulation allows the client can modify data and access hidden information.
- Tight coupling of traversal logic to data structure and client.
- Violates SOLID principles if we try to implement new traversal logic.

## Class Diagram



## Structure of the Iterator Design Pattern

Let's understand the Structure of the Iterator Design Pattern using a Library example:



- **Iterator Interface** (`Iterator<T>`): Defines the contract for traversing a collection with `hasNext()` and `next()` methods.
- **Concrete Iterator**: Implements the traversal logic for a specific collection type. It tracks the current position while traversing the collection.
  - **LibraryIterator** → traverses books from first to last.
  - **DescendingLibraryIterator** → traverses books from last to first by starting at `numberOfBooks - 1` and decrementing the position.
- **Aggregate Interface** (`BookCollection`): Declares a method to create iterators (`createIterator()` and `createReverseIterator()`).
- **Concrete Aggregate** (`Library`): The actual collection of books (`Library` class) that implements the aggregate interface (creates and returns **Concrete Iterators** `LibraryIterator` and `DescendingLibraryIterator`).

## Implementation(Example: Library)

```
1 // Book class representing individual books in a library
2 public class Book {
3     private final String title;
4     private final String author;
5     private final String isbn;
6     private int price;
7
8     public Book(String title, String author, String isbn) {
9         this.title = title;
10        this.author = author;
11        this.isbn = isbn;
12    }
13
14    public String getTitle() {
15        return title;
16    }
17
18    public String getAuthor() {
19        return author;
20    }
21
22    public String getIsbn() {
23        return isbn;
24    }
25 }
```

```

25
26     public int getPrice() {
27         return price;
28     }
29
30     @Override
31     public String toString() {
32         return "Book [Title=" + title + ", Author=" + author + ",
ISBN=" + isbn + "]\n";
33     }
34 }

```

```

1 // Aggregate interface
2 public interface BookCollection {
3     Iterator<Book> createIterator();
4
5     Iterator<Book> createReverseIterator();
6 }

```

```

1 // Iterator interface
2 public interface Iterator<T> {
3     boolean hasNext();
4
5     T next();
6 }

```

```

1 // Concrete Aggregate
2 public class Library implements BookCollection {
3
4     private final List<Book> books;
5
6     public Library(List<Book> books) {
7         this.books = books;
8     }
9
10    @Override
11    public Iterator<Book> createIterator() {
12        return new LibraryIterator(books);
13    }
14
15    @Override
16    public Iterator<Book> createReverseIterator() {
17        return new ReverseLibraryIterator(books);
18    }
19 }

```

```

1 // Concrete Iterator - for Library
2 public class LibraryIterator implements Iterator<Book> {
3     private final List<Book> books;
4     private int position = 0;
5
6     public LibraryIterator(List<Book> books) {
7         this.books = books;
8     }
9
10    @Override
11    public boolean hasNext() {
12        return position < books.size();
13    }
14
15    @Override
16    public Book next() {
17        return books.get(position++);
18    }
19 }

```

```

1 // Concrete Iterator - for Library
2 public class ReverseLibraryIterator implements Iterator<Book> {
3     private final List<Book> books;
4     private int position;
5
6     public ReverseLibraryIterator(List<Book> books) {
7         this.books = books;
8         this.position = books.size() - 1;
9     }
10
11    @Override
12    public boolean hasNext() {
13        return position >= 0;
14    }
15
16    @Override
17    public Book next() {
18        return books.get(position--);
19    }
20 }

```

```

5
6     public ReverseLibraryIterator(List<Book> books) {
7         this.books = books;
8         this.position = books.size() - 1;
9     }
10
11     @Override
12     public boolean hasNext() {
13         return position >= 0 && books.get(position) != null;
14     }
15
16     @Override
17     public Book next() {
18         if (!hasNext()) {
19             return null;
20         }
21         return books.get(position--); // Return current book and move
backward
22     }
23 }

```

```

1 // Client
2 public class LibraryIteratorDemo {
3
4     public static void main(String[] args) {
5         System.out.println("\n##### Iterator Design Pattern #####");
6
7         // Create Library
8         Library library = getLibrary();
9
10        // Forward iteration
11        Iterator<Book> iterator = library.createIterator();
12        System.out.println("\n==> Forward iteration:");
13        displayLibrary(iterator);
14
15        // Reverse iteration
16        Iterator<Book> reverseIterator =
library.createReverseIterator();
17        System.out.println("\n==> Reverse iteration:");
18        displayLibrary(reverseIterator);
19    }
20
21    private static Library getLibrary() {
22        List<Book> books = List.of(
23            new Book("To Kill a Mockingbird", "Harper Lee", "978-
0-74-7356-5"),
24            new Book("The Great Gatsby", "F. Scott Fitzgerald",
"778-0-24-7156-5"),
25            new Book("The Catcher in the Rye", "J.D. Salinger",
"333-0-28-7446-8"),
26            new Book("The Hobbit", "J.R.R. Tolkien", "783-0-14-
1951-8"),
27            new Book("Rich Dad Poor Dad", "Robert Kiyosaki", "183-
0-12-1491-8"),
28            new Book("Pride and Prejudice", "Jane Austen", "289-0-
12-1678-8")
29        );
30        Library library = new Library(books);
31        return library;
32    }
33
34    private static void displayLibrary(Iterator<Book> iterator) {
35        while (iterator.hasNext()) {
36            Book book = iterator.next();
37            System.out.println(book);
38        }
39    }
40 }

```

## Output

```
##### Iterator Design Pattern #####
```

```
==> Forward iteration:
```

```
Book [Title=To Kill a Mockingbird, Author=Harper Lee, ISBN=978-0-74-7356-5]  
Book [Title=The Great Gatsby, Author=F. Scott Fitzgerald, ISBN=778-0-24-7156-5]  
Book [Title=The Catcher in the Rye, Author=J.D. Salinger, ISBN=333-0-28-7446-8]  
Book [Title=The Hobbit, Author=J.R.R. Tolkien, ISBN=783-0-14-1951-8]  
Book [Title=Rich Dad Poor Dad, Author=Robert Kiyosaki, ISBN=183-0-12-1491-8]  
Book [Title=Pride and Prejudice, Author=Jane Austen, ISBN=289-0-12-1678-8]
```

```
==> Reverse iteration:
```

```
Book [Title=Pride and Prejudice, Author=Jane Austen, ISBN=289-0-12-1678-8]  
Book [Title=Rich Dad Poor Dad, Author=Robert Kiyosaki, ISBN=183-0-12-1491-8]  
Book [Title=The Hobbit, Author=J.R.R. Tolkien, ISBN=783-0-14-1951-8]  
Book [Title=The Catcher in the Rye, Author=J.D. Salinger, ISBN=333-0-28-7446-8]  
Book [Title=The Great Gatsby, Author=F. Scott Fitzgerald, ISBN=778-0-24-7156-5]  
Book [Title=To Kill a Mockingbird, Author=Harper Lee, ISBN=978-0-74-7356-5]
```

```
Process finished with exit code 0
```