

Itérateur (patron de conception)

Un itérateur est un objet qui permet de parcourir tous les éléments contenus dans un autre objet, le plus souvent un conteneur (liste, arbre, etc).

Un synonyme d'itérateur est curseur, notamment dans le contexte des bases de données.

Itérateur (patron de conception)

Un itérateur ressemble à un pointeur disposant essentiellement de *deux/trois* primitives :

- *accéder* à l'élément pointé en cours (dans le conteneur), et *se déplacer* pour pointer vers l'élément suivant.

(`nextElement()` in Java)

- déterminer à tout moment si l'itérateur a *épuisé* la totalité des éléments du conteneur.

(`hasMoreElements()` in Java)

Bien sur, Il faut pouvoir créer un itérateur pointant sur le premier élément

Iterator Pattern - Contrasting with indexing

Although indexing may also be used with some object-oriented containers, the use of iterators may have some advantages:

- Counting loops are not suitable to all data structures, in particular to data structures with no or slow random access
- Iterators make the code more readable, reusable, and less sensitive to a change in the data structure.
- An iterator can enforce additional restrictions on access, such as ensuring that elements can not be skipped or that a previously visited element can not be accessed a second time.
- An iterator may allow the container object to be modified without invalidating the iterator. For instance, once an iterator has advanced beyond the first element it may be possible to insert additional elements into the beginning of the container with predictable results. With indexing this is problematic since the index numbers must change.

Iterators and collections in Java

There are multiple ways to iterate a collection in Java

Example:

```
ArrayList persons = new ArrayList();
```

```
Person p= new Person("john", "smith");  
persons.add(p);
```

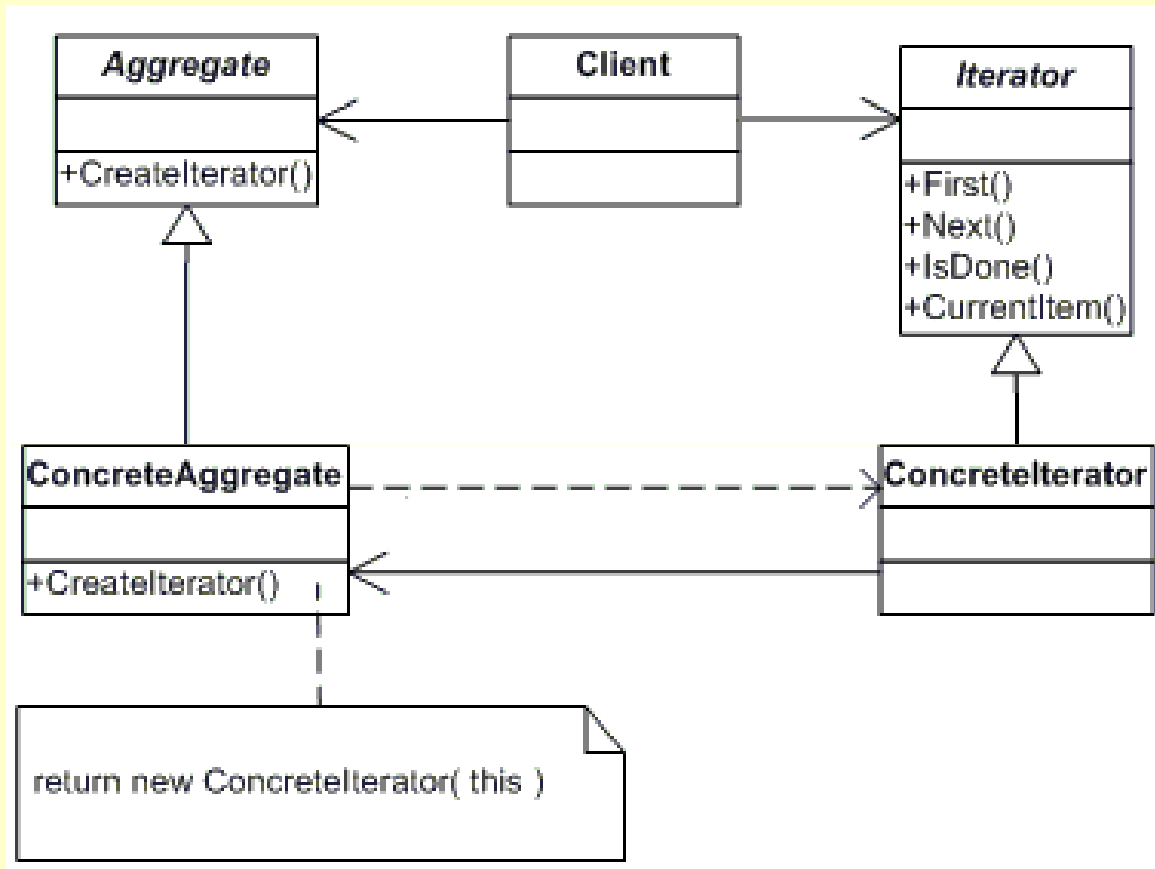
```
...
```

```
Iterator i = persons.iterator();
```

```
while(i.hasNext())  
{  
  
    Person p= (Person)i.next();  
  
    p.print();  
}
```

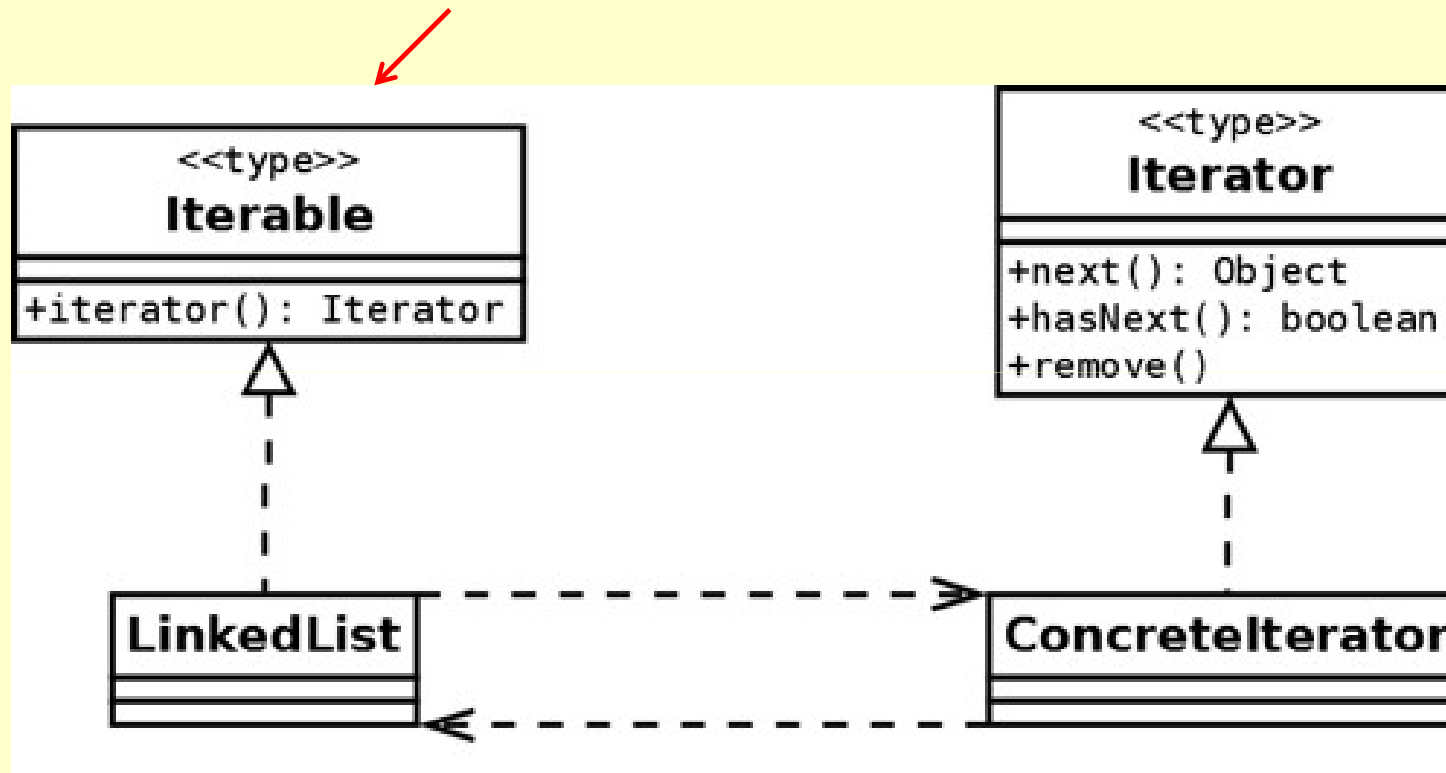
```
for(Person p :  
persons)  
{  
    p.print();  
} // since java 1.5
```

Iterator in UML class diagram



Iterator in UML class diagram (linked list example in Java)

Notice that the **Aggregate** is an **Iterable** (in Java)



Java Iterable Example (in Iterator folder/package)

<http://www-public.int-evry.fr/~gibson/Teaching/DesignPatterns/IteratorSource.zip>

```
public class MyTableOfStrings implements
Iterable<String> {

protected String[] data;

public MyTableOfStrings(String [] data) {
this.data = data;
}

public int length(){return data.length;}

public Iterator<String> iterator() {
return new MyTableOfStrings_Iterator(this);
}

}
```

```
public class MyTableOfStrings_Iterator implements Iterator<String> {

    private int index;
    private MyTableOfStrings table;

    public MyTableOfStrings_Iterator(MyTableOfStrings tab) {
        index = tab.length()-1;
        table = tab;
    }

    public String next() {
        index--;
        return table.data[index +1];
    }

    public boolean hasNext() {
        return index >= 0;
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }

}
```

```
public class MyTableOfStrings_Test {

public static void main(String[] s) {

String [] data = {"one", "two", "three"};
MyTableOfStrings t = new MyTableOfStrings(data);

System.out.println("Iterate over original data array");
for (String value : data) {
System.out.println(" "+value);
}
System.out.println("\nIterate over same data in MyTableOfStrings");
for (String value : t) {
System.out.println(" "+value);
}
}

}
```

p_Iterator.MyTableOfStrings_Test

Version:

1 A simple test for the [MyTableOfStrings](#)

EXPECTED OUTPUT :

```
Iterate over original data array
one
two
three
```

```
Iterate over same data in MyTableOfStrings
three
two
one
```

Author:

J Paul Gibson

TO DO : Compile and execute the test class

Random Iteration: Reservoir Sampling

In the previous example we saw how the `Iterator` code decides the order in which to visit the elements.

By default Java iterates through arrays from the 1st to the last elements. In the example we iterate through `MyTableOfStrings` in reverse order.

TO DO:

Change the iterator code so that the elements are visited in random order. Do not do this by shuffling the elements as this may be expensive for a large number of elements.

A more complex data structure: an urn/ballot box of bulletins/votes

© p_Iterator.Urn

Version:

1 An urn of votes, where each vote is a table of strings, eg:

```
[["gibson", "smith", "hughes"],  
 ["jones", "bell"],  
 ["raffy", "lallet"]]
```

represents three preferential votes with the first vote being -
first preference for gibson, second preference for smith and third preference for hughes

Author:

J Paul Gibson

TO DO: Your task is to iterate through the Strings in the Urn

First, look at the Urn_Test Code

```

public class Urn_Test {

public static void main(String[] s) {

String [] preferences1 = {"gibson", "smyth", "hughes"};
MyTableOfStrings vote1 = new MyTableOfStrings( preferences1);

String [] preferences2 = {"jones", "bell"};
MyTableOfStrings vote2 = new MyTableOfStrings( preferences2);

String [] preferences3 = {"raffy", "lallet"};
MyTableOfStrings vote3 = new MyTableOfStrings( preferences3);

MyTableOfStrings [] votes = { vote1, vote2, vote3};

Urn urn = new Urn (votes);

System.out.println("\nIterate over strings on bulletins in Urn");
for (String value : urn) {System.out.println(" "+value);}
}
}

```

Write the Urn and Urn_Iterator classes appropriately.

```
public class Urn implements Iterable<String>{
```

```
    \\ TO DO
```

```
}
```

```
public class Urn_Iterator implements Iterator<String> {
```

```
    \\ TO DO
```

```
}
```

Check that the test, executed on your code, produces the expected results

p_Iterator.Urn_Test

Version:

1 A simple test for the [Urn](#)

EXPECTED OUTPUT :

```
Iterate over strings on bulletins in Urn
hughes
smyth
gibson
bell
jones
lallet
raffy
```

Author:

J Paul Gibson