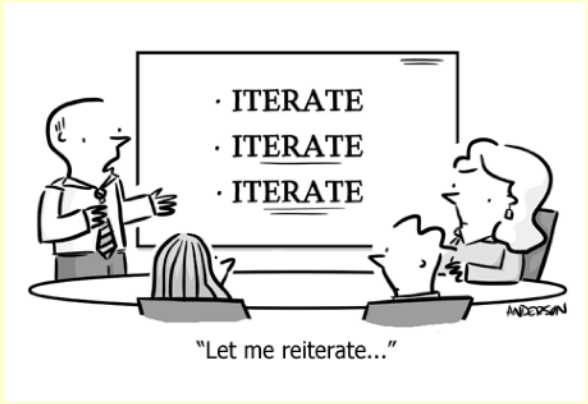
CSC7203 : Advanced Object Oriented Development

J Paul Gibson, D311



Iterator Design Pattern

…/~gibson/Teaching/CSC7203/CSC7203-AdvancedOO-L2-Iterator.pdf

Iterator

An **iterator** is an object that facilitates traversal of a container/ collection of objects

Various types of iterators are often provided via a container's interface.

Though the interface and semantics of a given iterator are fixed, iterators are tightly coupled to the container implementation in order to enable the operational semantics of the iterator.

Note that an iterator performs traversal and also gives access to data elements in a container, but does not perform iteration.

QUESTION: Have you already seen this in Java?

Iterator Pattern

See - http://sourcemaking.com/design_patterns/iterator

- Intent
  - Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
  - The C++ and Java standard library abstraction that makes it possible to decouple collection classes and algorithms.
  - Promote to "full object status" the traversal of a collection.
  - Polymorphic traversal

- Problem

Need to "abstract" the traversal of wildly different data structures so that algorithms can be defined that are capable of interfacing with each transparently.

Iterator Pattern

See - http://sourcemaking.com/design_patterns/iterator

Relation to other patterns

•The abstract syntax tree of Interpreter is a Composite (therefore Iterator and Visitor are also applicable).
•Iterator can traverse a Composite. Visitor can apply an operation over a Composite.
•Polymorphic Iterators rely on Factory Methods to instantiate the appropriate Iterator subclass.
•Memento is often used in conjunction with Iterator. An Iterator can use a Memento to capture the state of an iteration. The Iterator stores the Memento internally.

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", "mith");
persons.add(p);
...

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

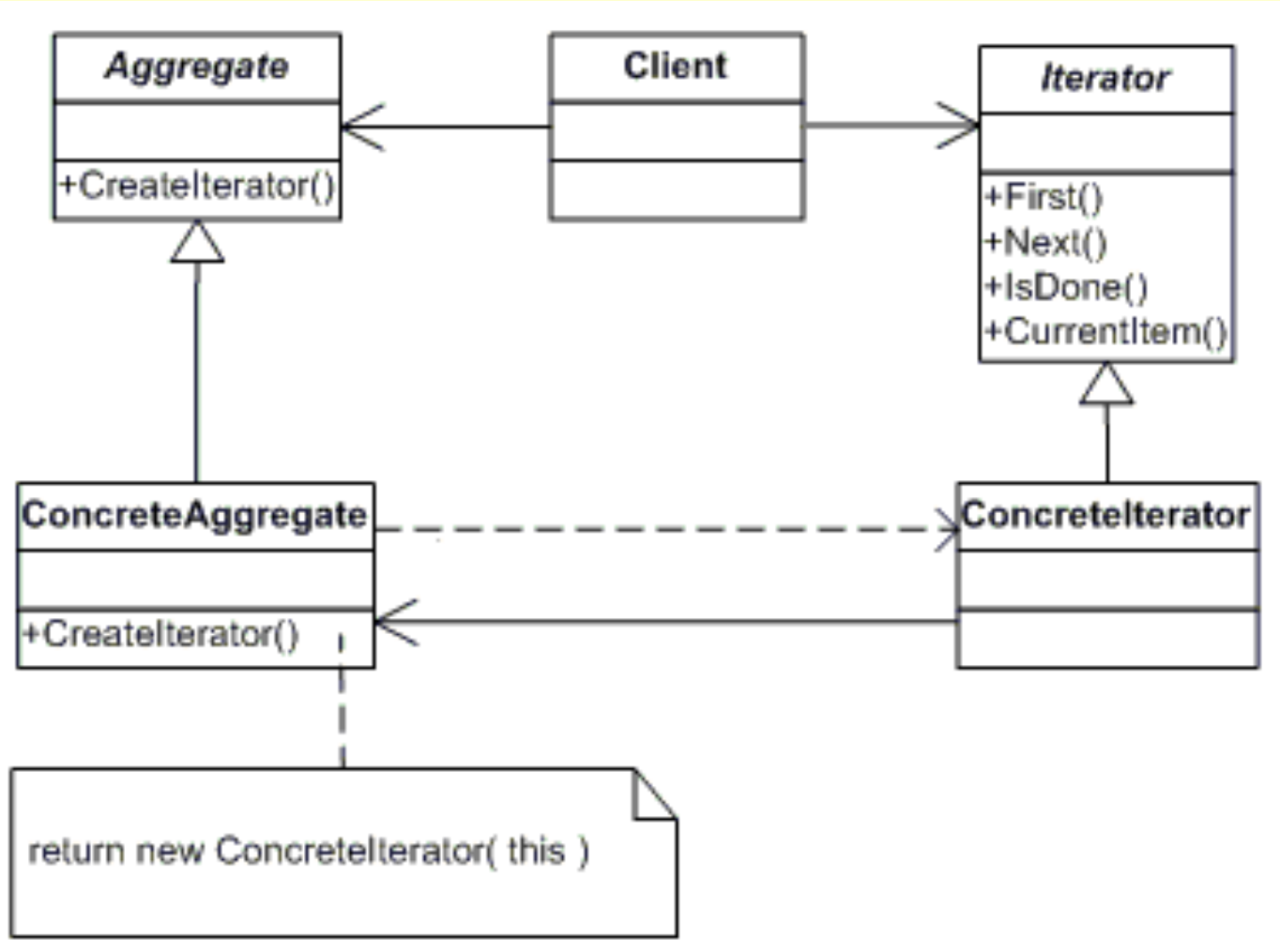
```
while(i.hasNext())
{

    Person p= (Person)i.next();


    p.print();
}
```

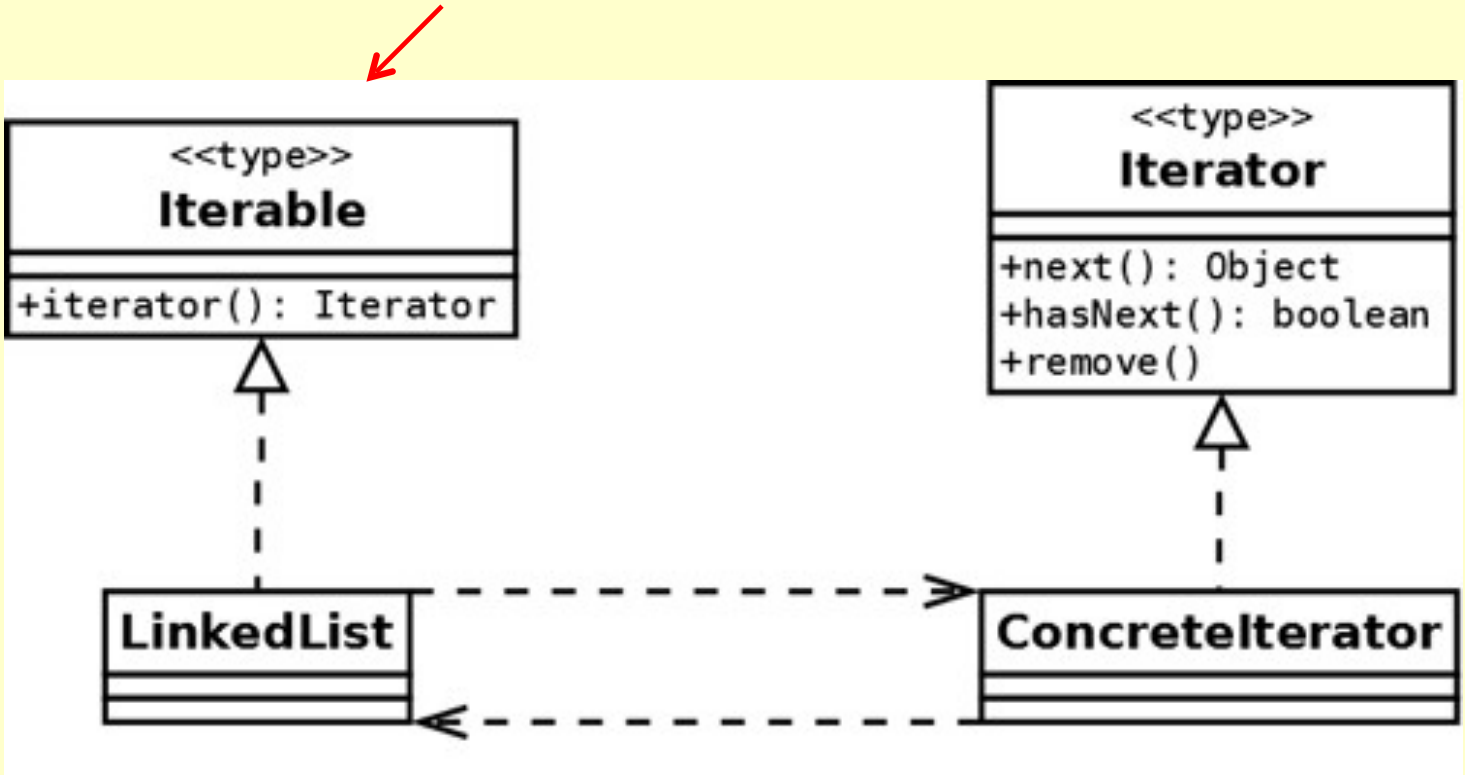
```
for(Person p : persons)
{
   p.print();
} // since java 1.5
```

Iterator in UML class diagram



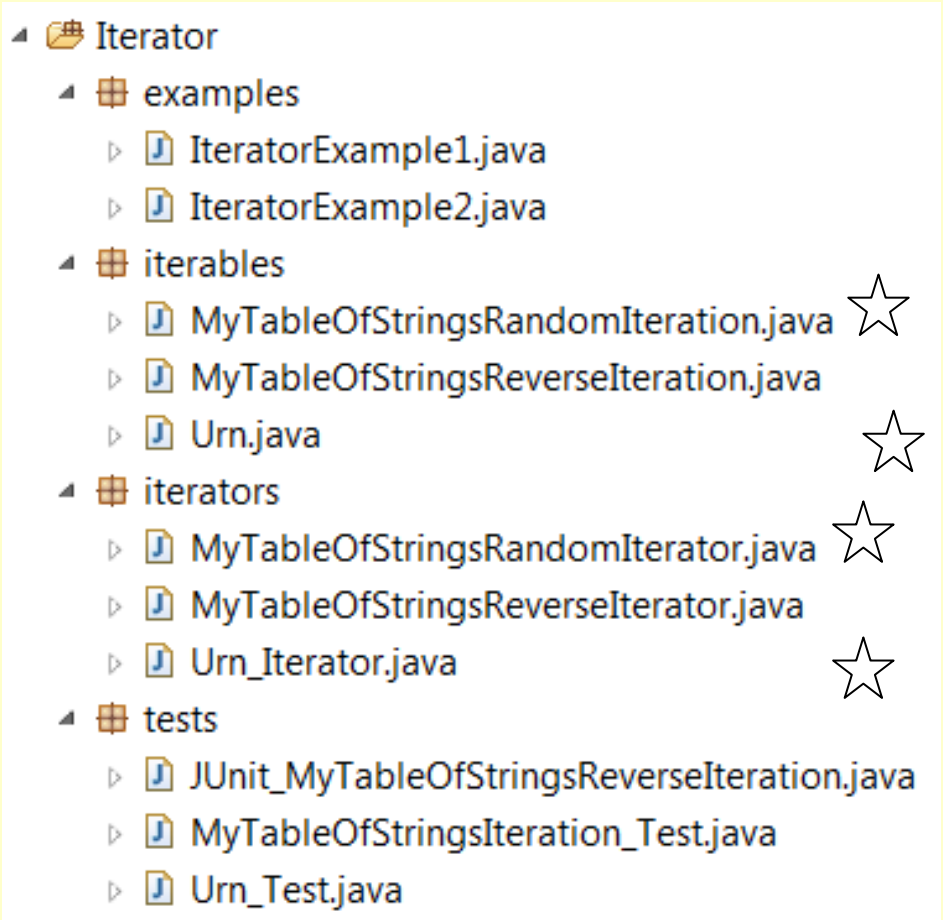
Iterator in UML class diagram (linked list example in Java)

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



Java Iterable Example (in Iterator folder/package)

(~gibson/Teaching/CSC7203/Code/Iterator.zip)



☆ TO DO

Java Iterable Example (in Iterator folder/package)

(~gibson/Teaching/CSC7203/Code/Iterator.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

Look at the `Urn_Test` Code and write the `Urn` and `Urn_Iterator` classes appropriately.

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

```
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);}
}
}
```

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