

CS 213 – Software Methodology

Spring 2017

Sesh Venugopal

Lecture 18: Mar 28

Design Patterns – 3
Iterator

Iterator Design Pattern: Behavioral

```
public class LinkedList<T> {
    public static class Node<E> {
        public E data;
        public Node<E> next;
    }
    public Node<T> front;
    . . .
}
```

Solution 1: Iterate by directly accessing nodes

```
LinkedList<String> list =
    new LinkedList<String>();

for (LinkedList.Node<String> ptr = list.front;
     ptr != null; ptr = ptr.next) {
    System.out.println(ptr.data);
}
```

Only works if `Node` and `front` are accessible to clients, which means they must be made public. Not a good design idea!

Need something like this instead



```
public class LinkedList<T> {
    protected static class Node<E> {
        protected E data;
        protected Node<E> next;
    }
    protected Node<T> front;
    . . .
}
```

Iterator: Behavioral

Solution 2: Iterate via method invocation

Basic Iteration using solution 2

```
public class LinkedList<T> {
    . . .
    protected Node<T> curr;

    public void reset() {
        curr = front;
    }

    public T next() {
        T ret=null;
        if (curr != null) {
            ret = curr.data;
            curr = curr.next;
        }
        return ret;
    }

    public boolean hasNext() {
        return curr != null;
    }
}
```

```
LinkedList<String> list = new LinkedList<String>();
. . .
for (list.reset(); list.hasNext();) {
    System.out.println(list.next());
}
```

E.g. Print #links from each web page to all other web pages

```
LinkedList<URL> list = new LinkedList<URL>();
// populate with web pages . . .
for (list.reset(); list.hasNext();) {
    URL wp1 = list.next();
    for (list.reset(); list.hasNext();) {
        URL wp2 = list.next();
        int n = numLinks(wp1, wp2);
        System.out.println("#links from " + wp1 +
                           " to " + wp2 + " = " + n);
    }
}
```

This won't work – the inner loop thrashes the state of the outer!

Iterator: Behavioral

Solution 3: Separate the Iterator from the LinkedList

```
// in same package as LinkedList
public class LinkedListIterator<T> {

    protected LinkedList.Node<T> curr;

    public LinkedListIterator(
        LinkedList<T> list) {
        curr = list.front;
    }

    public T next() {
        T ret = null;
        if (curr != null) {
            ret = curr.data;
            curr = curr.next;
        }
        return ret;
    }

    public boolean hasNext() {
        return curr != null;
    }
}
```

**Print #links from each web page
to all other web pages**

```
LinkedList<URL> list =
    new LinkedList<URL>();

// populate with web pages . . .

LinkedListIterator<URL> iter1 =
    new LinkedListIterator<URL>(list);
LinkedListIterator<URL> iter2 =
    new LinkedListIterator<URL>(list);

while (iter1.hasNext()) {
    URL wp1 = iter1.next();
    while (iter2.hasNext()) {
        URL wp2 = iter2.next();
        int n = numLinks(wp1, wp2);
        . . .
    }
}
```

Iterator: Behavioral

Solution 4: Generalization with Interface

Have the `LinkedListIterator` class
implement an interface

`java.util`

```
public interface Iterator<T> {
    boolean hasNext();
    T next();
    void remove();
}
```

This is a default method in the Java 8 version of the `Iterator` interface, which throws this exception. So this particular implementation need not be coded since it is the same as the default



```
class LinkedListIterator<T>
    implements Iterator<T> {
    protected LinkedList<T> list;
    protected LinkedList.Node<T> curr;

    LinkedListIterator(LinkedList<T> list) {
        this.list = list;
        curr = list.front;
    }

    public T next() {
        T ret = null
        if (curr != null) {
            ret = curr.data;
            curr = curr.next;
        }
        return ret;
    }

    public boolean hasNext() {
        return curr != null;
    }

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

Iterator: Behavioral

Solution 4: Generalization with Interface

Finish up by having the `LinkedList` class implement a method that will return an instance of the `LinkedListIterator`

```
public class LinkedList<T> {  
    . . .  
    public Iterator<T> iterator() {  
        return new  
            LinkedListIterator<T>(this);  
    }  
    . . .  
}  
  
LinkedList<URL> list =  
    new LinkedList<URL>();  
// populate with web pages . . .  
  
Iterator<URL> iter1 = list.iterator();  
  
while (iter1.hasNext()) {  
    URL wp1 = iter1.next();  
    Iterator<URL> iter2 = list.iterator();  
    while (iter2.hasNext()) {  
        URL wp2 = iter2.next();  
        int n = numLinks(wp1, wp2);  
        . . .  
    }  
}
```

Iterator: Behavioral

- Access the contents of a collection without exposing its internal representation
- Support overlapping multiple traversals
- Provide a uniform interface for traversing different collections – support polymorphic iteration

