

CMSC 433 – Programming Language Technologies and Paradigms Fall 2003

Iterators and Design Patterns
September 25, 2003

1

Iteration

- Goal: Loop through all objects in an aggregate

```
class Node { Element elt; Node next; }  
Node n = ...;  
while (n != null) { ...; n = n.next; }
```

- Problems:
 - Depends on implementation details
 - Varies from one aggregate to another

2

Iterators in Java

```
public Interface Iterator {  
    // returns true if the iteration has more elts  
    public boolean hasNext();  
  
    // returns the next element in the iteration  
    public Object next() throws NoSuchElementException;  
}
```

(plus optional remove method)

- Implementation of aggregate not exposed
- Generic for wide variety of aggregates
- Supports multiple traversal strategies

3

Generic Iterators in Java 1.5

```
public Interface Iterator<A> {  
    // returns true if the iteration has more elts  
    public boolean hasNext();  
  
    // returns the next element in the iteration  
    public A next() throws NoSuchElementException;  
}
```

So-- what's a generic? Well, let me digress ...

4

Using Iterators

```
Iterator<Element> i =  
    c.iterator();  
while (i.hasNext()) {  
    Element e = i.next();  
    // do stuff with e  
}
```

// alternatively use for

```
for (Iterator i = c.iterator();
```

5

The Queue Class

```
class Queue<Element> {  
    class Entry { // Java inner class  
        Element elt; Entry next;  
        Entry(Element i) { elt = i; next = null; }  
    }  
  
    Entry theQueue;  
  
    void enqueue(Element e) {  
        if (theQueue == null) theQueue = new Entry(e);  
        else {  
            Entry last = theQueue;  
            while (last.next != null) last = last.next;  
            last.next = new Entry(e);  
        }  
    }  
    ...  
}
```

6

The Queue Class (cont'd)

```
class Queue<Element> {
    ...
    Element dequeue() throws EmptyQueueException {
        if (theQueue == null)
            throw new EmptyQueueException();
        Element e = theQueue.elt;
        theQueue = theQueue.next;
        return e;
    }
}
```

7

next() Shouldn't Mutate Aggregate

```
class Queue<Element> {
    ...
    class QueueIterator implements Iterator<Element> {
        Entry rest;

        QueueIterator(Entry q) { rest = q; }
        boolean hasNext() { return rest != null; }
        Element next() throws NoSuchElementException {
            if (rest == null)
                throw new NoSuchElementException();
            Element e = rest.elt;
            rest = rest.next; // queue data intact
            return e;
        }
    }
}
```

8

Evil Mutating Clients

- But a client could mutate the data structure ...

```
HashMap h = ...;
...
Iterator i = h.entrySet().iterator();
System.out.println(i.next());
System.out.println(i.next());
h.put("Foo", "Bar"); // hash table resize!
System.out.println(i.next()); // prints ???
```

9

Defensive (Proactive) Copying

- Solution 1: Iterator copies data structure

```
class QueueIterator implements Iterator<Element> {
    Entry rest;

    QueueIterator(Queue q) {
        // copy q.theQueue to rest
    }
}
```

- Pro: Works even if queue is mutated
- Con: Expensive to construct iterator

10

Timestamps

- Solution 2: Track Mutations

```
class Queue<Element> {
    ...
    int modCount = 0;
    void enqueue(Element e) { ... modCount++; }
    Element dequeue() { ... modCount++; }
    ...
}
```

11

Timestamps (cont'd)

```
...
class QueueIterator implements Iterator<Element> {
    int expectedModCount = modCount; // set at iterator
    // construction time

    Element next() {
        if (expectedModCount != modCount)
            throw new ConcurrentModificationException();
        ...
    }
    // does hasNext() need to be modified?
}
```

- Pro: Iteration construction cheap
- Con: Doesn't allow any mutation

12

Comments

- Neither solution tracks mutations to container elts
 - Could use clone(), but tricky

13

What if Mutation is Allowed?

- Allowed mutation must be part of iterator spec

```
public void remove()  
    throws IllegalStateException;
```

- Removes from the underlying collection the last element returned by the iterator (optional operation). This method can be called only once per call to next.
- The behavior of an iterator is unspecified if the underlying collection is modified while the iteration is in progress in any way other than by calling this method.

14

Iterators

- Key ideas
 - Separate aggregate structure from traversal protocol
 - Support additional kinds of traversals
 - E.g., smallest to largest, largest to smallest, unordered
 - Multiple simultaneous traversals
 - Though many Java Collections do not provide this
- Structure
 - Iterator interface defines traversal protocol
 - Concrete Iterator implementations for each aggregate
 - And for each traversal strategy
 - Aggregate instances create Iterator object instances

15

Design Patterns

- Iterators are an example of a *design pattern*:
 - Design pattern = problem + solution in context
 - Iterators: solution for providing generic traversals
- Design patterns capture software architectures and designs
 - Not code reuse!
 - Instead, solution/strategy reuse
 - Sometimes, interface reuse

16

Gang of Four

- The book that started it all
- Community refers to authors as the “Gang of Four”
- Figures and some text in these slides come from book
- On reserve in CS library (3rd floor AVW)



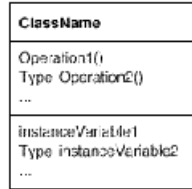
17

Object Modeling Technique (OMT)

- Used to describe patterns in GO4 book
- Graphical representation of OO relationships
 - **Class diagrams** show the static relationship between classes
 - **Object diagrams** represent the state of a program as series of related objects
 - **Interaction diagrams** illustrate execution of the program as an interaction among related objects

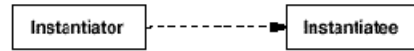
18

Classes



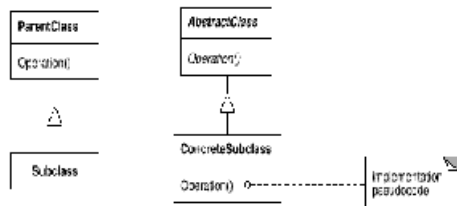
19

Object instantiation



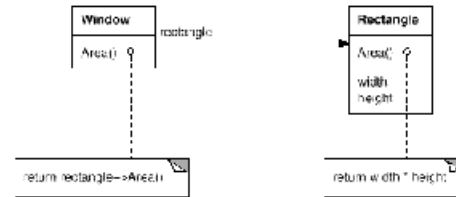
20

Subclassing and Abstract Classes



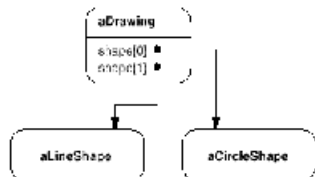
21

Pseudo-code and Containment



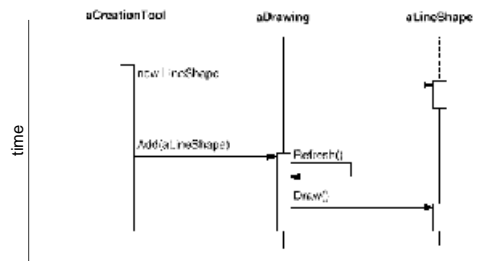
22

Object diagrams



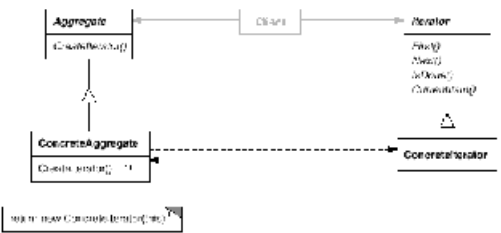
23

Interaction diagrams



24

Structure of Iterator (Cursor) Pattern



This document was created with Win2PDF available at <http://www.daneprairie.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.