

Design Patterns for Iteration

Note Title

2/13/2007

□ Iterators

purpose

example use

implementation for pointer-based + array-based data structures

□ Concurrent Modification

why it's a problem

detection with modification counts

□ Locators — robust iterators

□ The Valet Parking Problem

□ Markers

□ Trackers

□ Visitors

* Discuss Lab 2 Handout

Iterators

- efficient traversal of a structure
- hide internal rep.

```
class Foo<T> implements Iterable<T> {  
    public Iterator<T> iterator() {  
        return new FooIterator();  
    }  
    // other stuff  
}
```

```
f = new Foo<String>();  
Iterator<String> it = f.iterator();  
while (it.hasNext()) {  
    String s = it.next();  
    doSomethingWith(s);  
}
```

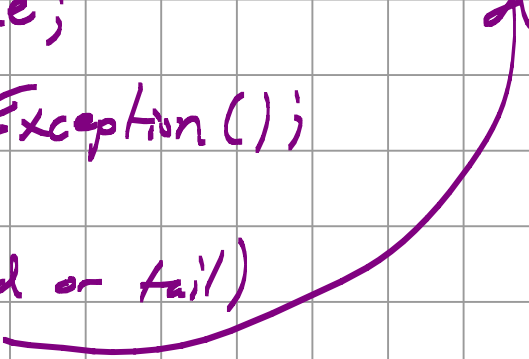
} ⇒ for (String s: f)
doSomethingWith(s);

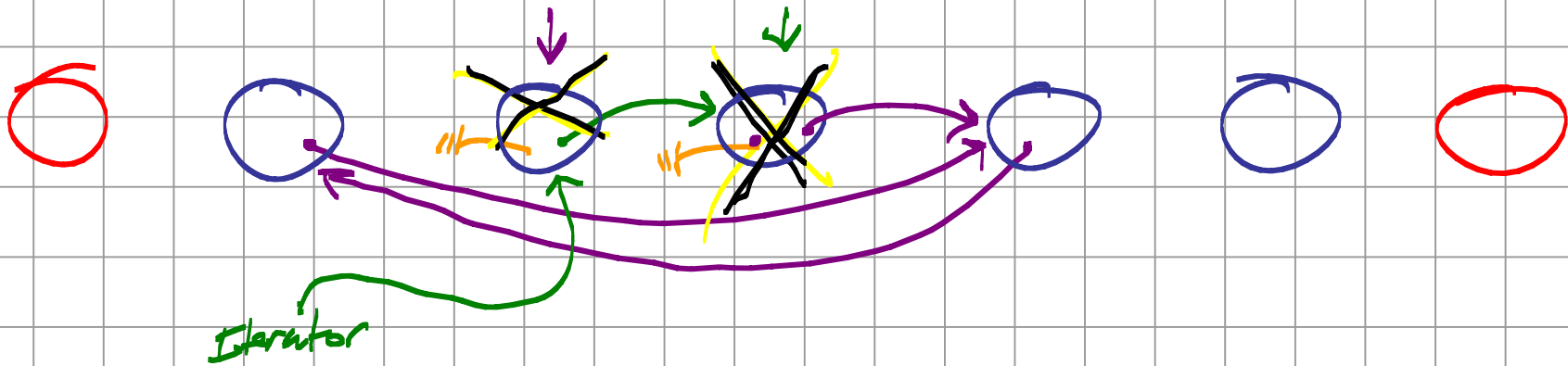
in a LinkedList (doubly linked, dummy/sentinel head & tail)

⇒ Inner class in the LinkedList class:

```
class MyLinkedListIterator<E> implements Iterator<E> {  
    ListItem currentPosition = head;  
    public boolean hasNext() {  
        return currentPosition.next != tail;  
    }  
    public E next() {  
        if (hasNext()) {  
            currentPosition = currentPosition.next;  
            return currentPosition.value;  
        } else {  
            throw new NoSuchElementException();  
        }  
    }  
    public void remove() {  
        if (currentPosition is not head or tail)  
            remove(currentPosition);  
    }  
}
```

This would splice the list item out of the list (without changing next in the list item)





- Use `previous ptr == null` to indicate a deleted list item
- advance in `hasNext` until reaching a non-deleted item (or fail)

In an array-based implementation

Iterator keep an index of current position



index = 2

Start iterator at index = 1
~~a b c d e f // // //~~

hasNext:

true when $index < size - 1$

next:

```
if (hasNext()) {  
    index++;  
    return a[index];  
}
```

Collection has:
remove(int index)

remove:

check if not at -1 & that
next has been called since last
call to remove
call remove(index), then index--;

a b c d e f // //

↑
index = 2

remove()

a b d e f // // //

↑
index = 2

next() ⇒

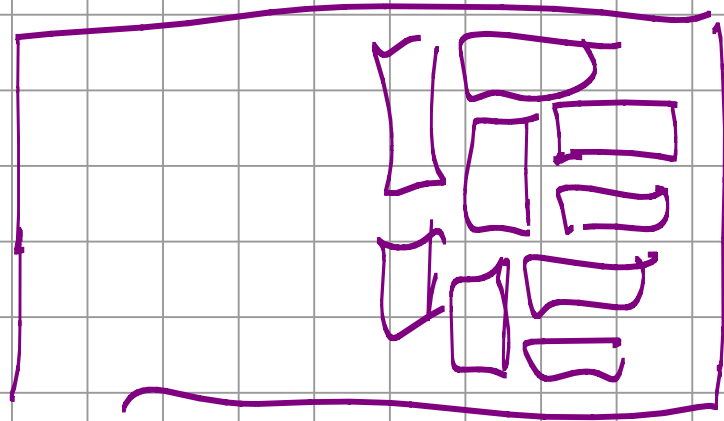
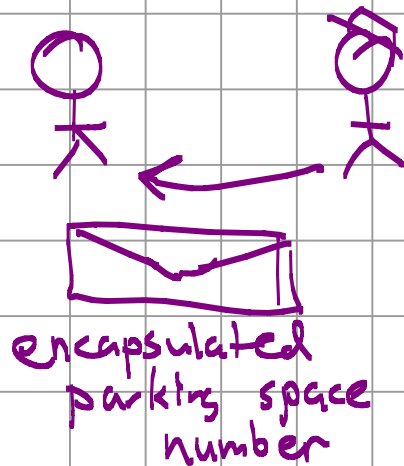
Java's solution to concurrent modifications:
"Fail-fast"

get a `ConcurrentModificationException` if
try to use an iterator on a collection
that has been modified

Need a detection mechanism

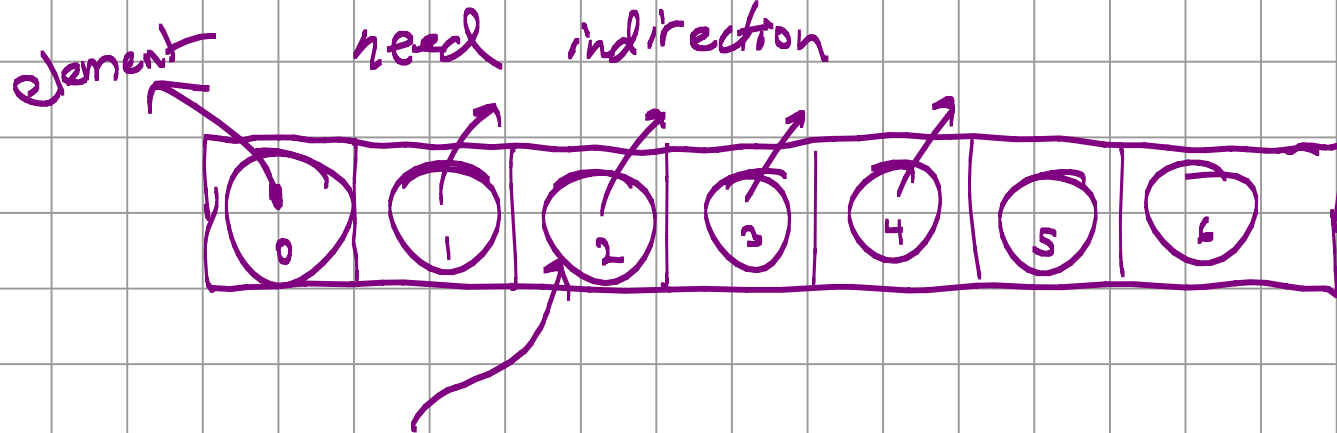
- ① let the collection keep a modification count
- ② create iterator — save copy of current modification count in iterator
- ③ when try to use iterator, it compares its count to the one in data structure
if iterator's is less \Rightarrow iterator is stale
throw `ConcurrentModificationExc...`

The Valet Parking Problem:



- ① Cars go in and out but don't switch spaces
Envelope is a "Marker" — Marks a particular spot
(like earlier array-based iterator)
- OR
- ② Cars get shifted around over time
Envelope is a "Tracker" — Keeps track of where the original element is.
(like earlier linked list iterator)

How would you implement a tracker for an array?

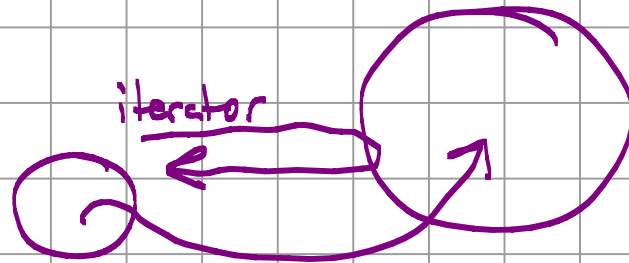


Collection $\langle E \rangle$
has: Array of
Container $\langle E \rangle$

Tracker
ptr to container for its element (container knows its index)

If containers move, update their indices

With iterators, the control flow (loop) is external to the data structure; data structure provides a way to access its internal structure



With visitors, the control flow is inside the data structure

