Iterators

An iterator permits you to examine the elements of a data structure one at a time.

---

Iterator Methods

```
Iterator ix = x.iterator();
```

constructs and initializes an iterator to examine the elements of `x`;
constructed iterator is assigned to `ix`

you must define the method `iterator` in the class for `x`

---

Iterator Methods

```
ix.hasNext()
```

returns `true` iff `x` has a next element

```
ix.next()
```

throws `NoSuchElementException` if there is no next element
returns next element otherwise

---

Optional Iterator Method

```
ix.remove()
```

removes last element returned by `ix.next()`
throws `UnsupportedMethodException` if method not implemented
throws `IllegalStateException` if `ix.next()` not yet called or did not return an element
Using An Iterator

```java
Iterator ix = x.iterator();
while (ix.hasNext())
    examine(ix.next());
```

vs

```java
for (int i = 0; i < x.size(); i++)
    examine(x.get(i));
```

Merits Of An Iterator

- It is often possible to implement the method `next` so that its complexity is less than that of `get`
- Many data structures do not have a `get by index` method
- Iterators provide a uniform way to sequence through the elements of a data structure

Java’s Array Linear List Class

```java
java.util.ArrayList
```

Cadillac version of our `ArrayLinearListWithIterator`

Linked Representation

- List elements are stored in memory, in an arbitrary order
- Explicit information (called a link) is used to go from one element to the next
Memory Layout

Layout of \( L = (a,b,c,d,e) \) using an array representation.

A linked representation uses an arbitrary layout.

Linked Representation

- FirstNode
- Pointer (or link) in e is null
- Use a variable firstNode to get to the first element a

Normal Way To Draw A Linked List

Chain

- A chain is a linked list in which each node represents one element.
- There is a link or pointer from one element to the next.
- The last node has a null pointer.
**Node Representation**

```java
package dataStructures;

class ChainNode {
    // package visible data members
    Object element;
    ChainNode next;

    // constructors come here
}
```

**Constructors Of ChainNode**

```java
ChainNode() {};

ChainNode(Object element) {
    this.element = element;
}

ChainNode(Object element, ChainNode next) {
    this.element = element;
    this.next = next;
}
```

**get(0)**

```java
checkIndex(0);
desiredNode = firstNode; // gets you to first node
return desiredNode.element;
```

**get(1)**

```java
checkIndex(1);
desiredNode = firstNode.next; // gets you to second node
return desiredNode.element;
```
get(2)

checkIndex(2);
desiredNode = firstNode.next.next; // gets you to third node
return desiredNode.element;

get(5)

checkIndex(5); // throws exception
desiredNode = firstNode.next.next.next.next.next;
return desiredNode.element; // null.element

NullPointerException

desiredNode =
firstNode.next.next.next.next.next.next;
// gets the computer mad
// you get a NullPointerException

Remove An Element

remove(0)

firstNode = firstNode.next;
remove(2)

firstNode

a

b

c

d

e

beforeNode

Step 1: get a node, set its data and link fields

ChainNode newNode = new ChainNode(new Character('f'), firstNode);

Step 2: update firstNode

firstNode = newNode;

add(0,'f')

firstNode

a

b

c

d

e

newNode

Step 1: get a node, set its data and link fields

ChainNode newNode = new ChainNode(new Character('f'), firstNode);

Step 2: update firstNode

firstNode = newNode;
One-Step add(0, 'f')

```java
firstNode = new ChainNode(new Character('f'), firstNode);
```

Two-Step add(3, 'f')

```java
beforeNode = firstNode.next.next;
beforeNode.next = new ChainNode(new Character('f'), beforeNode.next);
```

add(3, 'f')

1. first find node whose index is 2
2. next create a node and set its data and link fields
   ```java
   ChainNode newNode = new ChainNode(new Character('f'), beforeNode.next);
   ```
3. finally link beforeNode to newNode
   ```java
   beforeNode.next = newNode;
   ```