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

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

vs

```
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.util.ArrayList

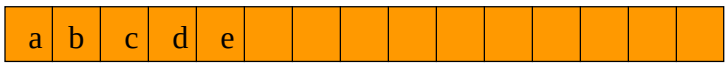
Cadillac version of our
ArrayLinearListWithIterator

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

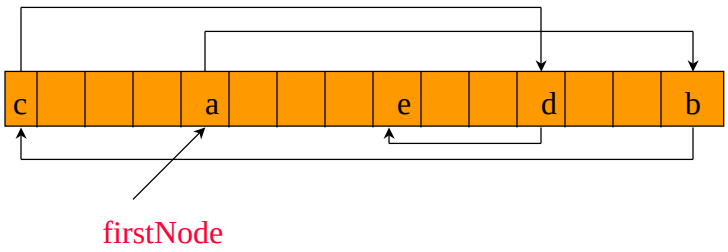
Memory Layout

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



A linked representation uses an arbitrary layout.

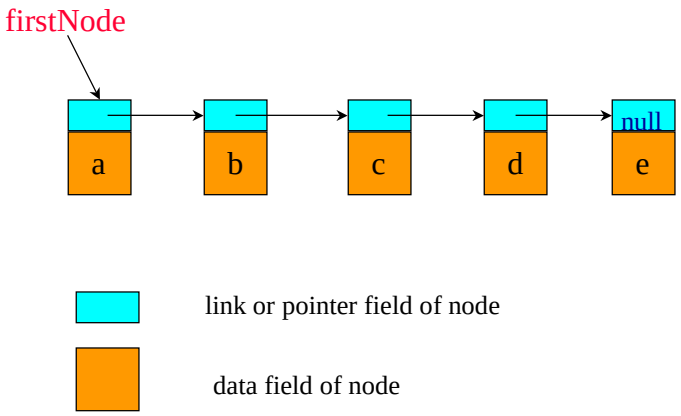
Linked Representation



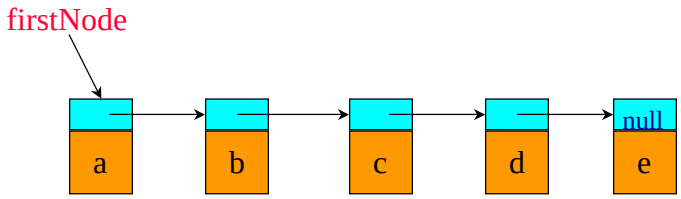
pointer (or link) in e is null

use a variable firstNode to get to the first element a

Normal Way To Draw A Linked List



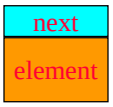
Chain



- 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.

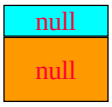
Node Representation

```
package dataStructures;

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

  // constructors come here
}
```

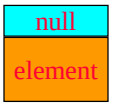


Constructors Of ChainNode

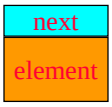
ChainNode() {}



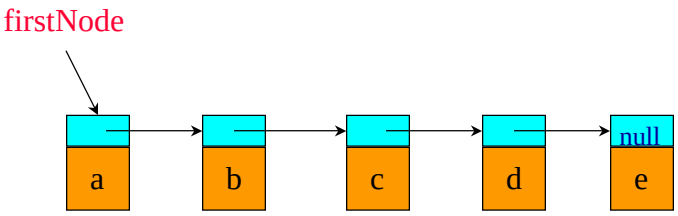
ChainNode(Object element)
{this.element = element;}



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

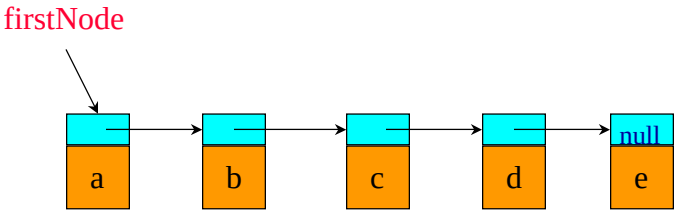


get(0)



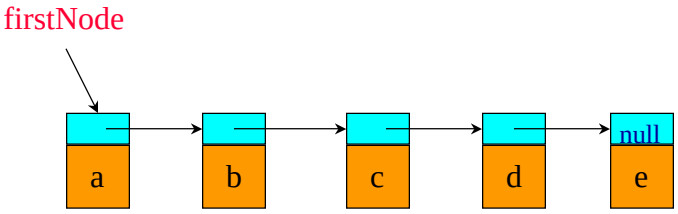

```
checkIndex(0);
desiredNode = firstNode; // gets you to first node
return desiredNode.element;
```

get(1)



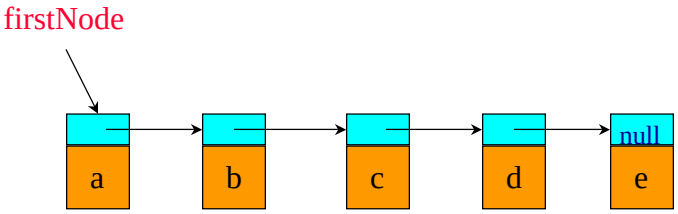

```
checkIndex(1);
desiredNode = firstNode.next; // gets you to second node
return desiredNode.element;
```

get(2)



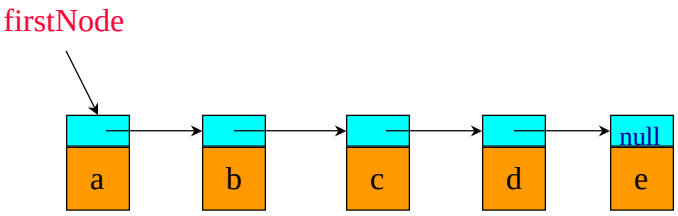

```
checkIndex(2);
desiredNode = firstNode.next.next; // gets you to third node
return desiredNode.element;
```

get(5)



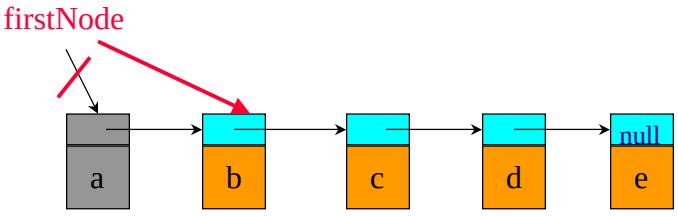

```
checkIndex(5);              // throws exception
desiredNode = firstNode.next.next.next.next.next;
                            // desiredNode = null
return desiredNode.element;   // null.element
```

NullPointerException




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

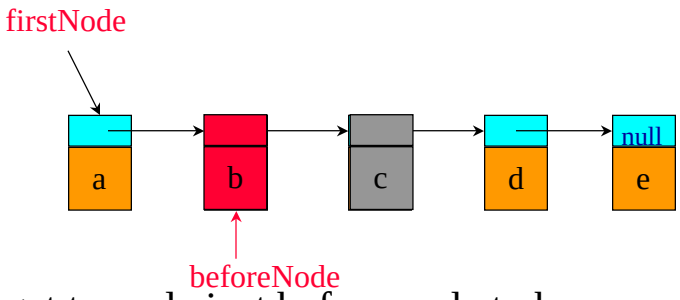
Remove An Element



remove(0)

firstNode = firstNode.next;

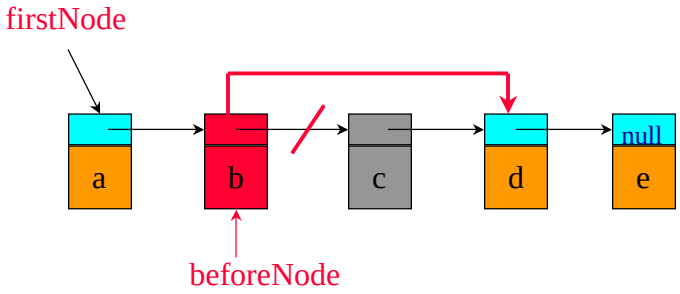
remove(2)



first get to node just before node to be removed

beforeNode = firstNode.next;

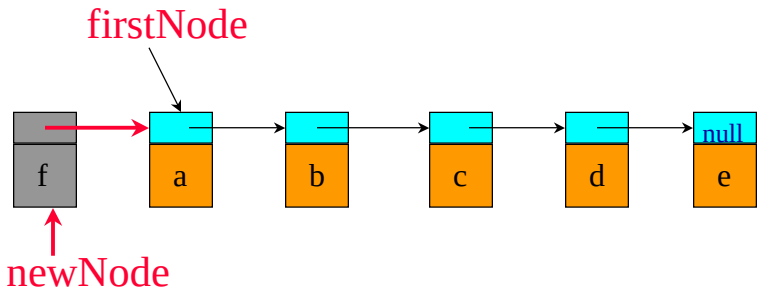
remove(2)



now change pointer in beforeNode

beforeNode.next = beforeNode.next.next;

add(0,'f')

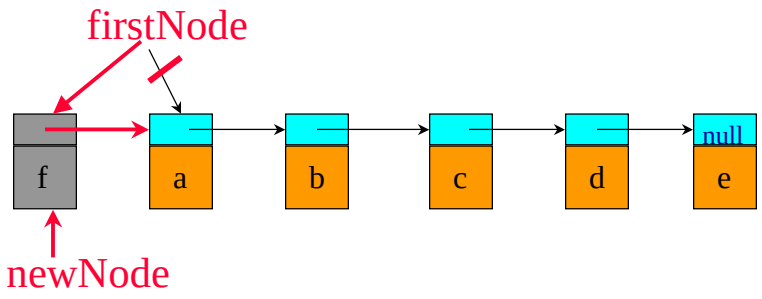


Step 1: get a node, set its data and link fields

ChainNode newNode =

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

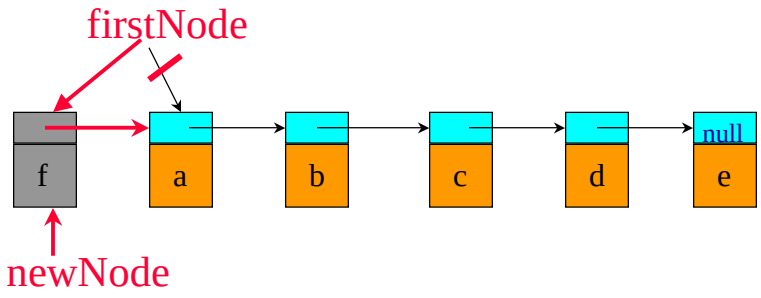
add(0,'f')



Step 2: update firstNode

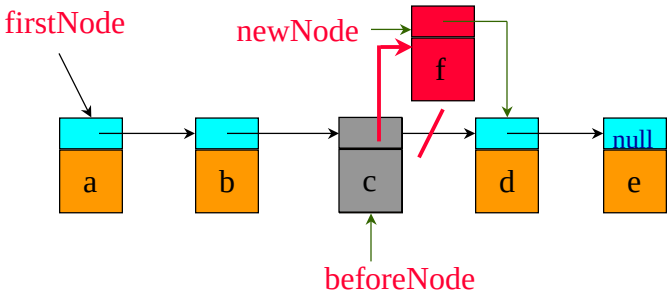
firstNode = newNode;

One-Step add(0,'f')




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

add(3,'f')



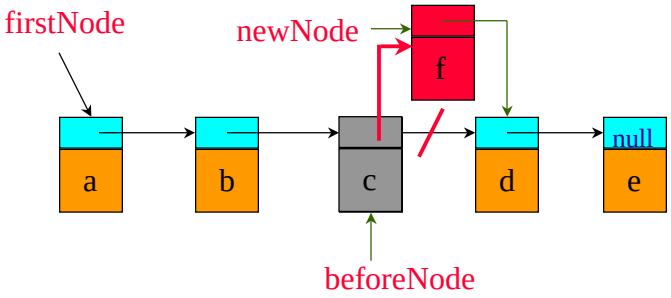
- first find node whose index is 2
- next create a node and set its data and link fields

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

- finally link beforeNode to newNode

```
beforeNode.next = newNode;
```

Two-Step add(3,'f')




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