A+ Computer Science

Iterators

What is a reference?

References

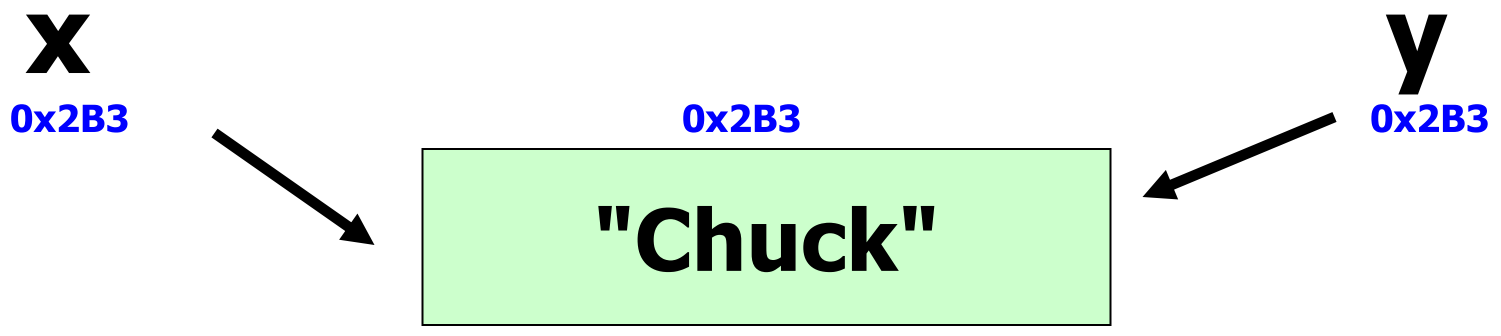
In Java, any variable that refers to an Object is a reference variable.

The variable stores the memory address of the actual Object.

References

```
String x = new String("Chuck");
String y = x;
```

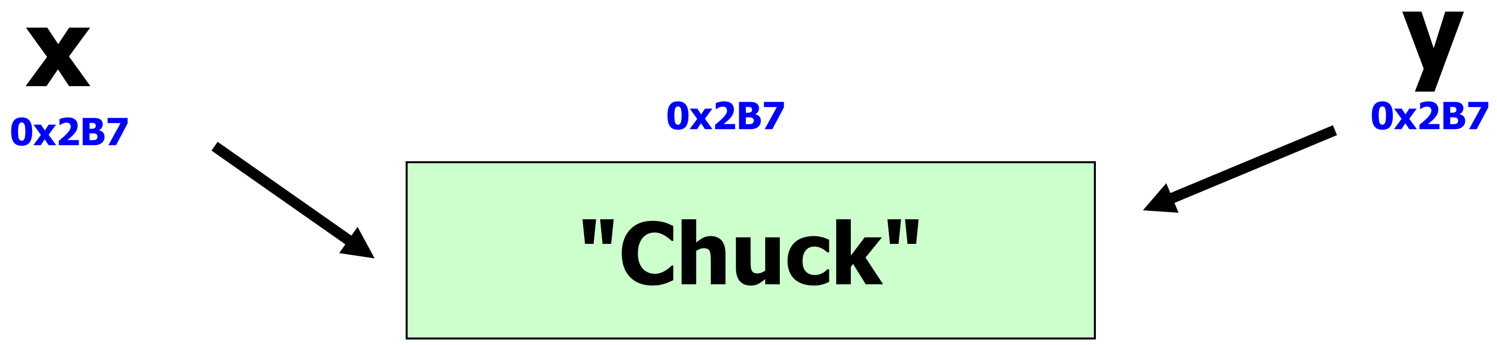
x and y store the same memory address.



References

```
String x = "Chuck";
String y = "Chuck";
```

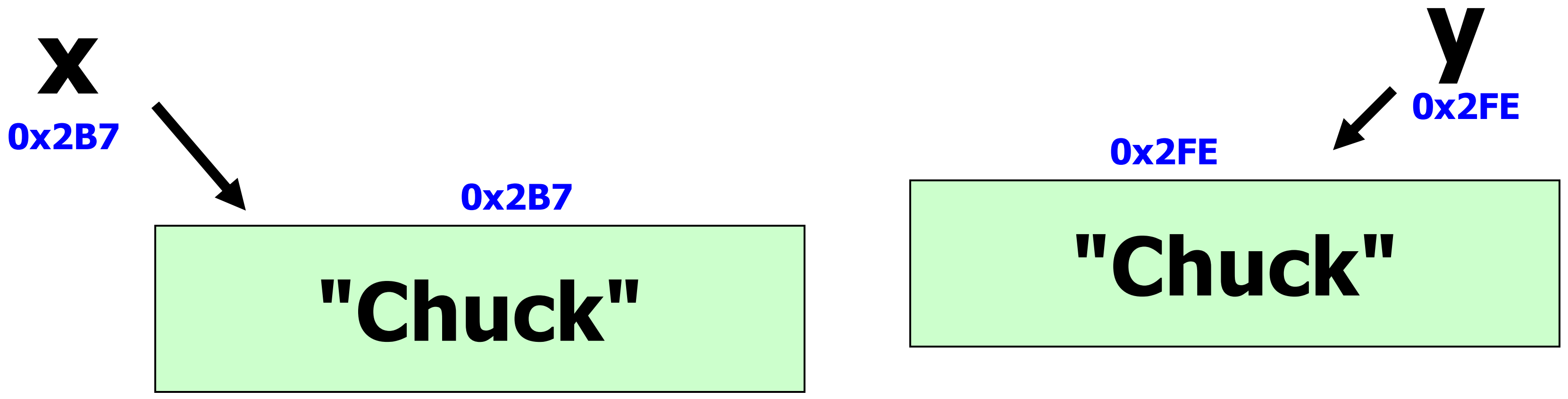
x and y store the same memory address.



References

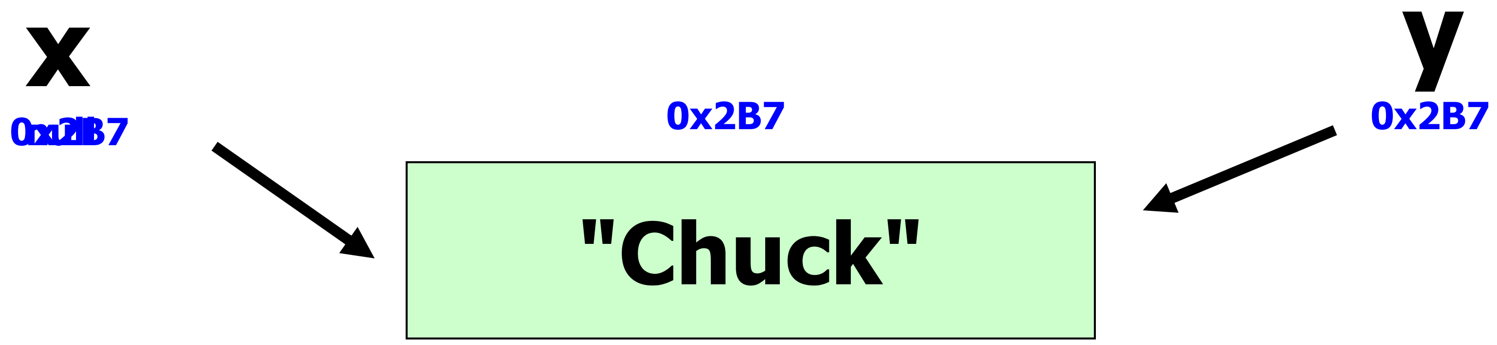
```
String x = new String("Chuck");
String y = new String("Chuck");
```

x and y store different memory addresses.



References

```
String x = "Chuck";
String y = "Chuck";
x = null;
```



references.java

What is a iterator?

Java Iterators

Collection, List, and Set all have methods that return iterators.

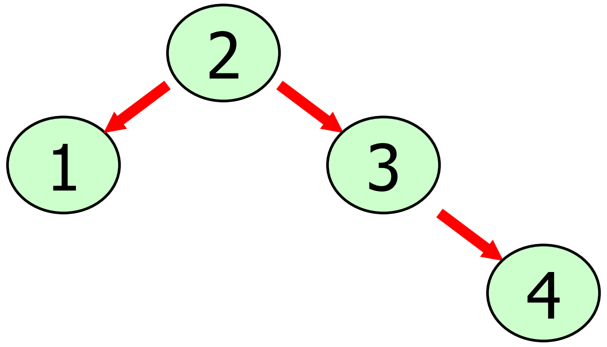
Iterators allow you to go from item to item through a collection.

Map does not have an iterator, but it does have a keySet() method that returns a Set of all keys. You can get an iterator from the Set.

What is an iterator?

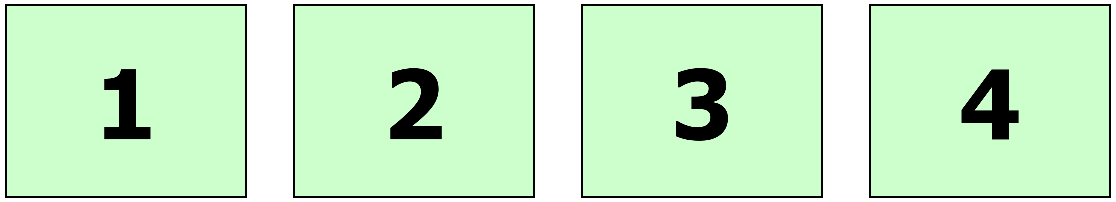
An Iterator provides a standard way to access all of the references stored in a collection.

For some Collections, TreeMap and HashSet for instance, the underlying data structures are not sequentially organized like an array. For example, a tree has nodes all over the place.



What is an iterator?

By using the Iterator, the references from a Collection can be accessed in a more standard sequential-like manner without having to manipulate the underlying Collection data structure.



Iteratator Interface

Iterator

frequently used methods

Name	Use
next()	returns a reference to the next item
remove()	removes the last ref returned by next
hasNext()	checks to see there are more items

```
import  java.util.Iterator;
```

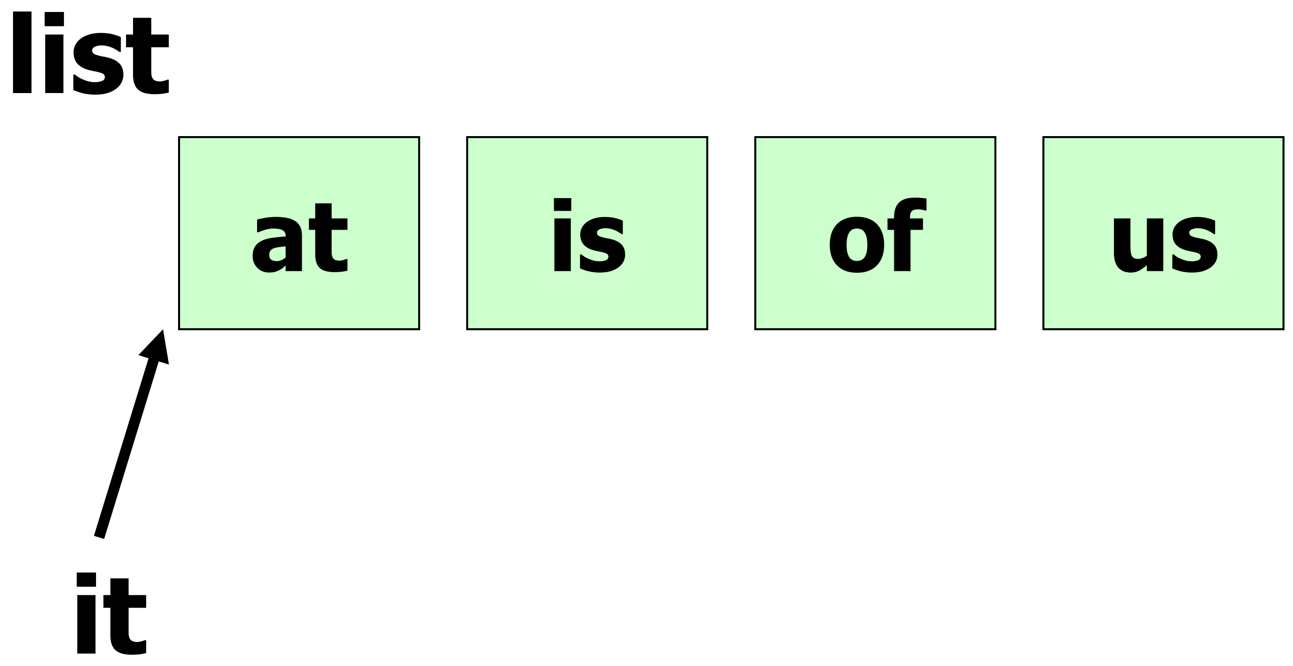
next() method

```
ArrayList<String> words;
words = new ArrayList<String>();
words.add("at");
words.add("is");
words.add("of");
words.add("us");

Iterator<String> it = words.iterator();
System.out.println(it.next());
```

OUTPUT

```
at
```

next() method

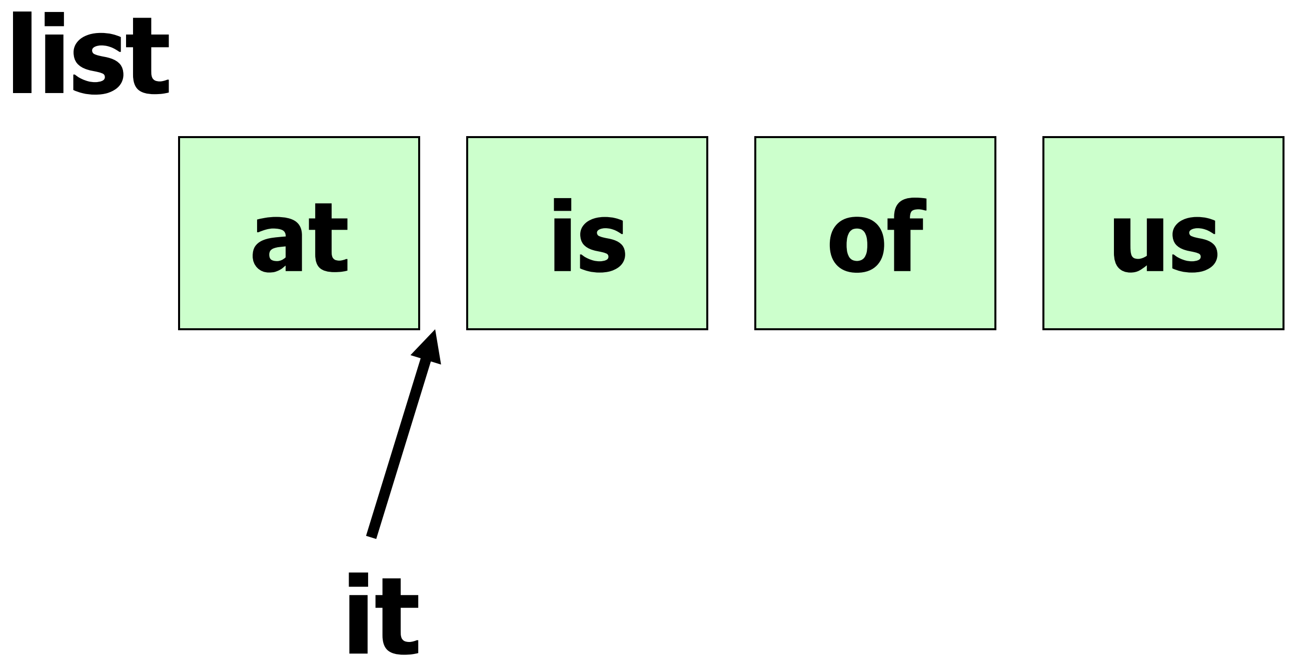


Iterator it = list.iterator();

next() method

```
method next()
{
  oldRef = currRef
  currRef = next ref in the collection
  return oldRef
}
```

next() method



```
it.next();
```

next moves the iterator up one spot and returns a reference to the 1st item.

next() method

```
ArrayList<String> words;
words = new ArrayList<String>();
words.add("at");
words.add("is");
words.add("of");
words.add("us");

Iterator<String> it = words.iterator();
System.out.println(it.next());
System.out.println(it.next());
System.out.println(it.next());
System.out.println(it.next());
```

OUTPUT

```
at
is
of
us
```

iteratorone.java

hasNext() method

```
ArrayList<String> words;
words = new ArrayList<String>();

words.add("at");
words.add("is");
words.add("of");
words.add("us");

Iterator<String> it = words.iterator();
while(it.hasNext())
{
  System.out.println(it.next());
}
```

OUTPUT

```
at
is
of
us
```

hasnext.java

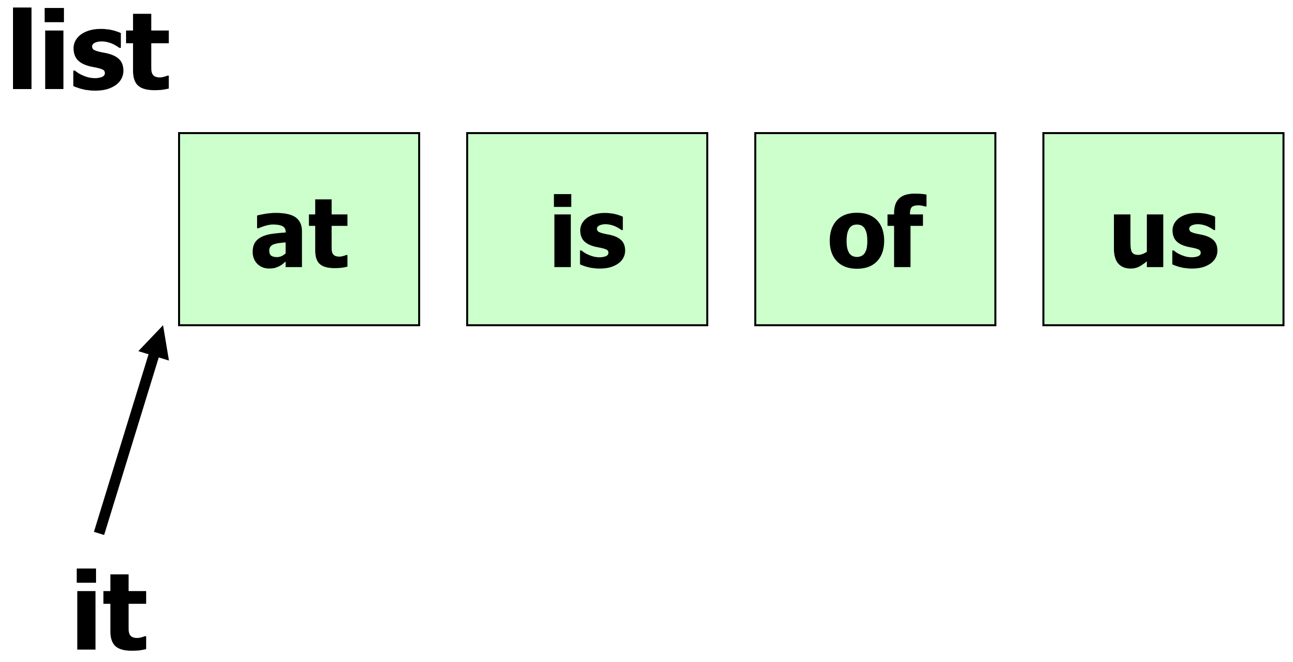
remove() method

```
ArrayList<String> words;
words = new ArrayList<String>();

words.add("at");
words.add("is");
words.add("of");

Iterator<String> it = words.iterator();
System.out.println(it.next());
it.remove();
System.out.println(it.next());
System.out.println(words);
```

OUTPUT

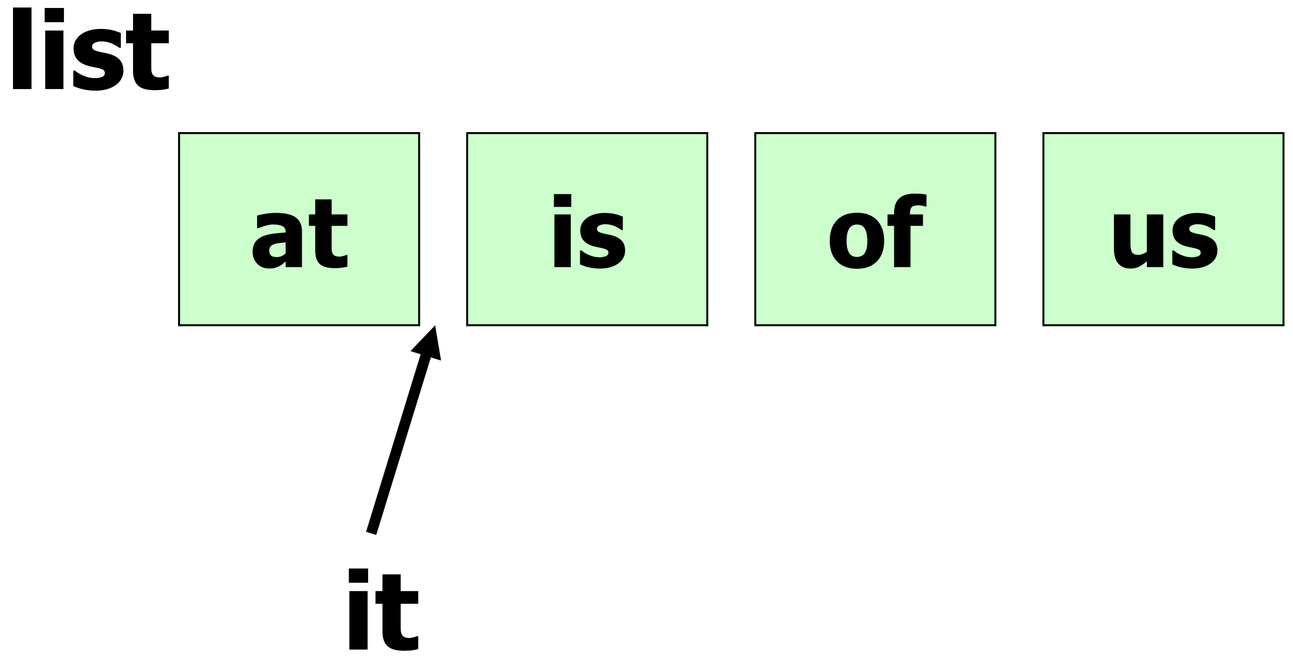
```
at
is
[is, of]
```

remove() method



```
Iterator it = list.iterator();
```

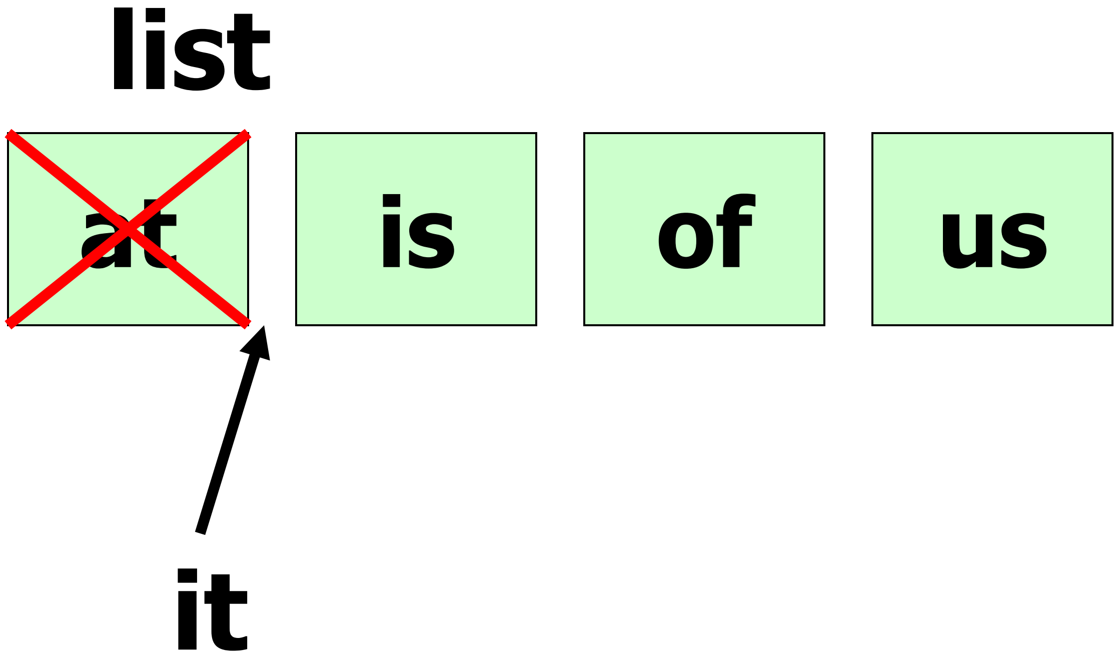
remove() method



it.next();

next moves the iterator up one spot and returns a reference to the 1st item.

remove() method



it.remove();

remove always modifies the last reference returned by next.

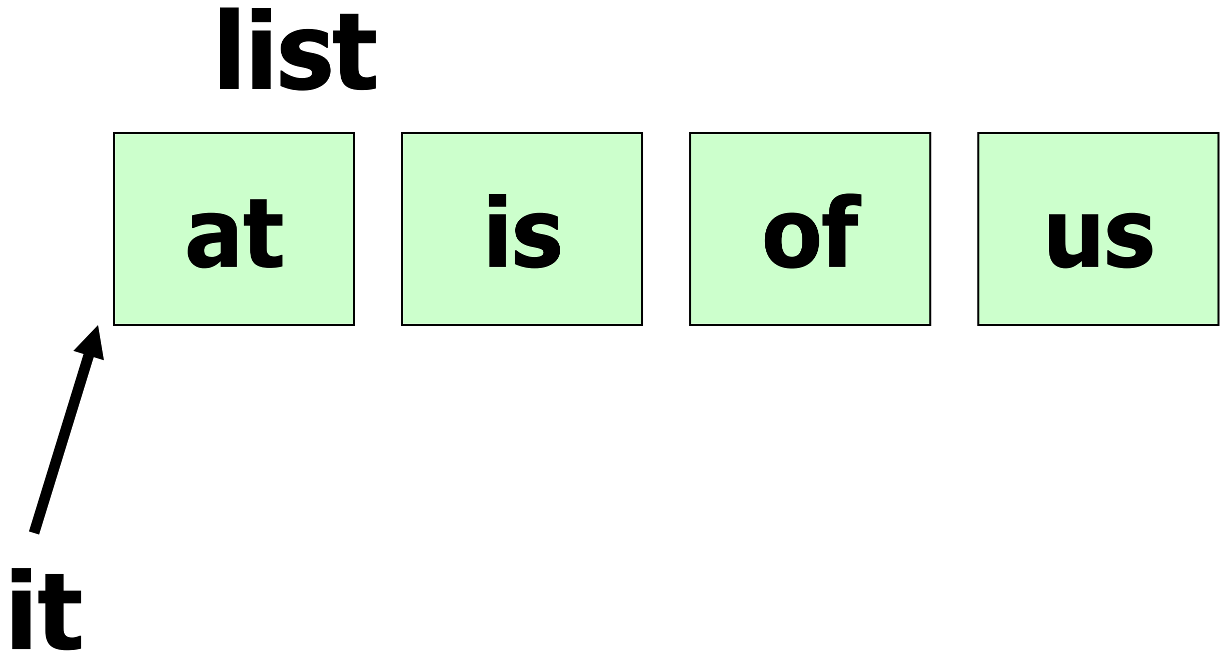
remove() method

```
ArrayList<String> words;
words = new ArrayList<String>();

words.add("at");
words.add("is");
words.add("of");

Iterator<String> it = words.iterator();
System.out.println(it.next());
it.remove();
it.remove();
```

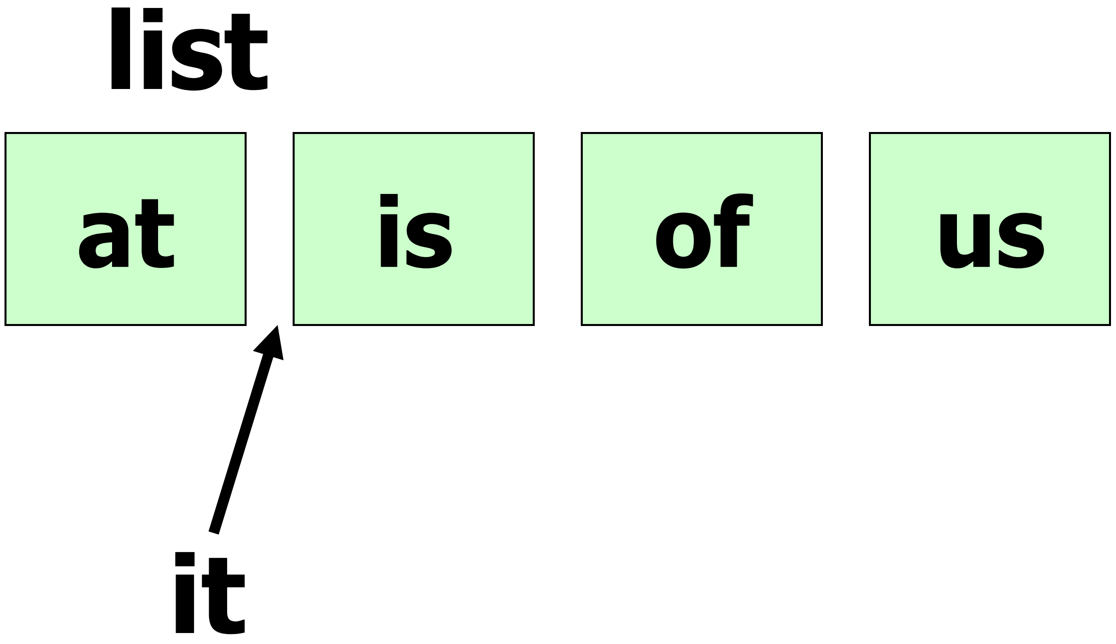
OUTPUT

```
at
error
```

remove() method



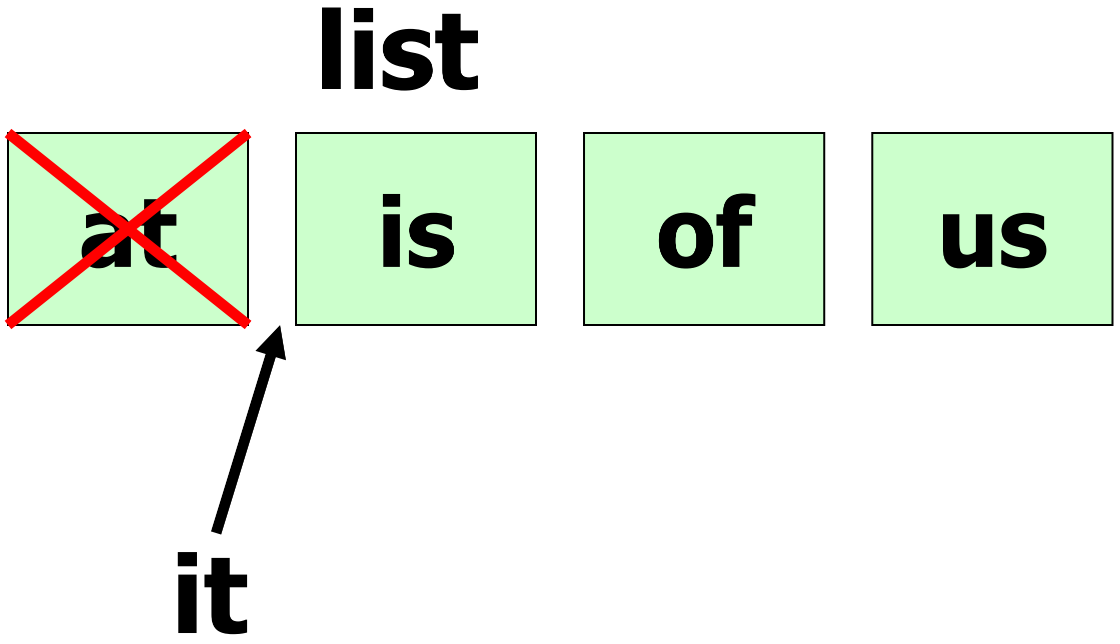
remove() method



it.next();

next moves the iterator up one spot and returns a reference to the 1st item.

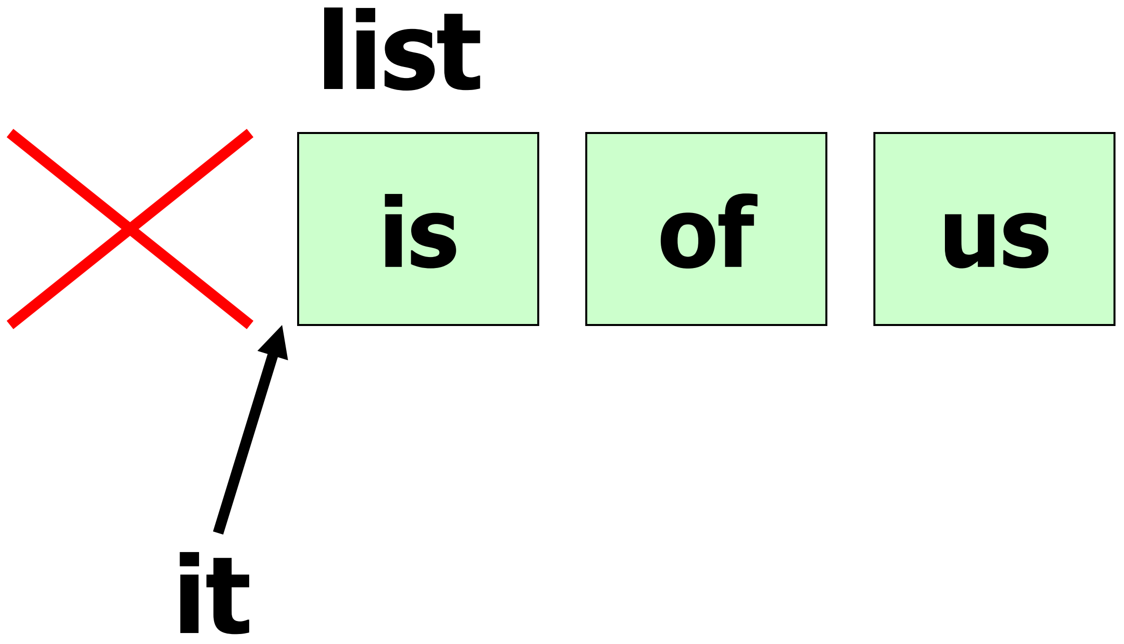
remove() method



it.remove();

remove always modifies the last reference returned by next.

remove() method



it.remove();

remove call blows up because there was no call to next; thus, there was no reference to modify.

removeone.java
removetwo.java

ListIteratator Interface

ListIterator

frequently used methods

Name	Use
next()	returns a reference to the next item
remove()	removes the last ref returned by next or previous
hasNext()	checks to see there are more items
add()	adds in a new item
set()	sets the last ref returned by next or previous
previous()	goes back and returns a ref to prev item

```
import  java.util.ListIterator;
```

ListIterator

```
ArrayList<String> words;
words = new ArrayList<String>();

words.add("at");
words.add("is");
words.add("of");
words.add("us");

ListIterator<String> it = words.listIterator();
System.out.println(it.next());
System.out.println(it.next());
```

OUTPUT

```
at
is
```

listiteratorone.java

previous() method

```
ArrayList<String> words;
words = new ArrayList<String>();

words.add("at");
words.add("is");
words.add("of");
words.add("us");

ListIterator<String> it = words.listIterator();
System.out.println(it.next());
System.out.println(it.next());
System.out.println(it.previous());
```

OUTPUT

```
at
is
is
```

previousone.java

previous() method

```
ArrayList<String> words;
words = new ArrayList<String>();
words.add("at");
words.add("up");
words.add("or");

ListIterator<String> it = words.listIterator();
it.next();
it.next();
it.next();
System.out.println(it.previous());
System.out.println(it.previous());
it.set("33");
System.out.println(words);
```

OUTPUT

```
or
up
[at, 33, or]
```

previoustwo.java

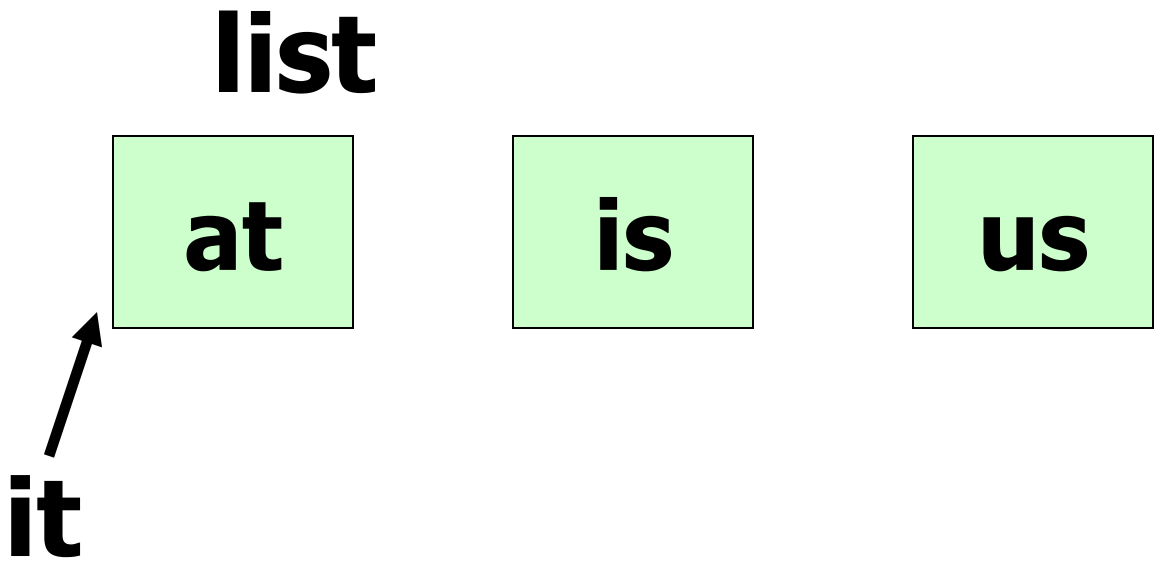
set() method

```
ArrayList<String> words;
words = new ArrayList<String>();

words.add("at");
words.add("is");
words.add("us");

ListIterator<String> it = words.listIterator();
System.out.println(it.next());
it.set("###");
System.out.println(it.next());
System.out.println(words);
```

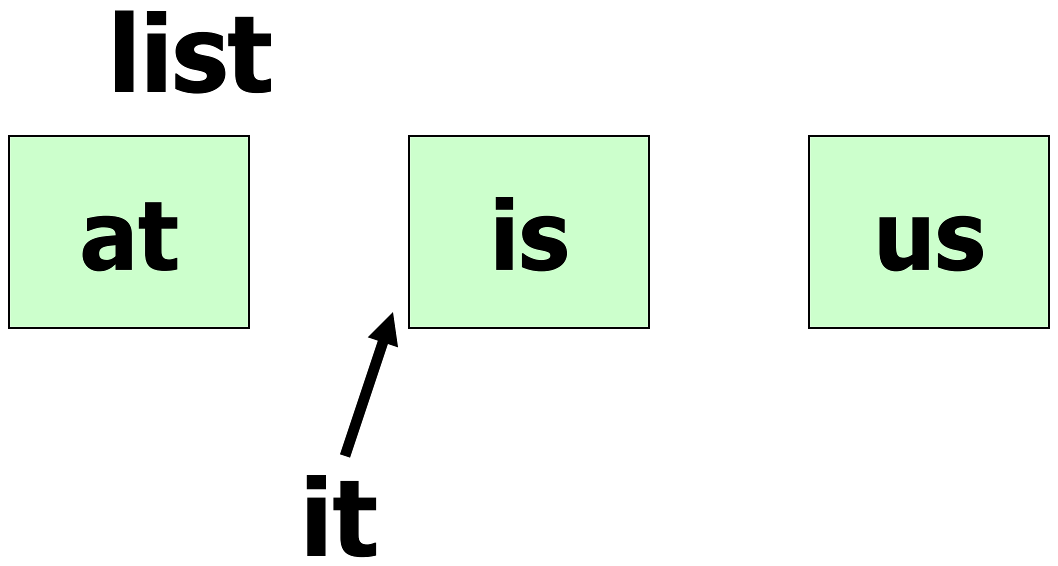
OUTPUT

```
at
is
[###, is, us]
```

set() method



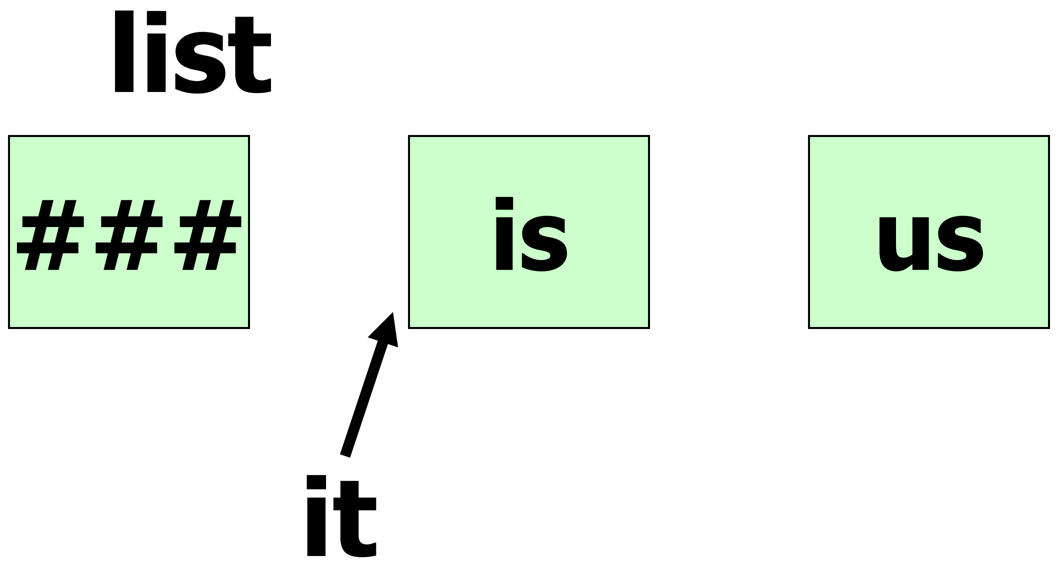
set() method



it.next();

next moves the iterator up one spot and returns a reference to the 1st item.

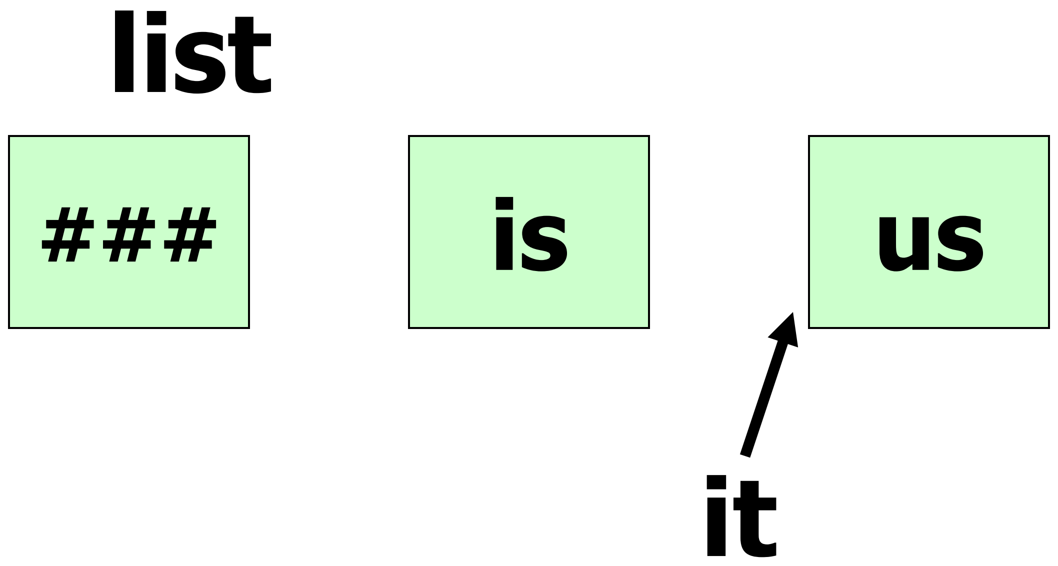
set() method



```
it.set("###");
```

set always modifies the last reference returned by next or previous.

set() method



it.next();

next moves the iterator up one spot and returns a reference to the 2nd item.

setone.java
settwo.java

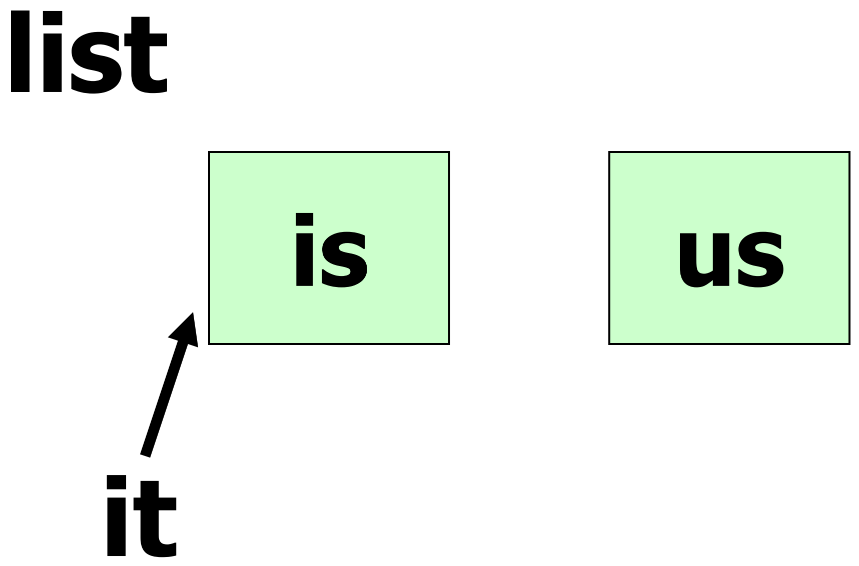
add() method

```
ArrayList<String> words;
words = new ArrayList<String>();

words.add("is");
words.add("us");

ListIterator<String> it = words.listIterator();
it.add("##");
System.out.println(it.next());
System.out.println(it.next());
System.out.println(it.previous());
it.set("##");
System.out.println(words);
```

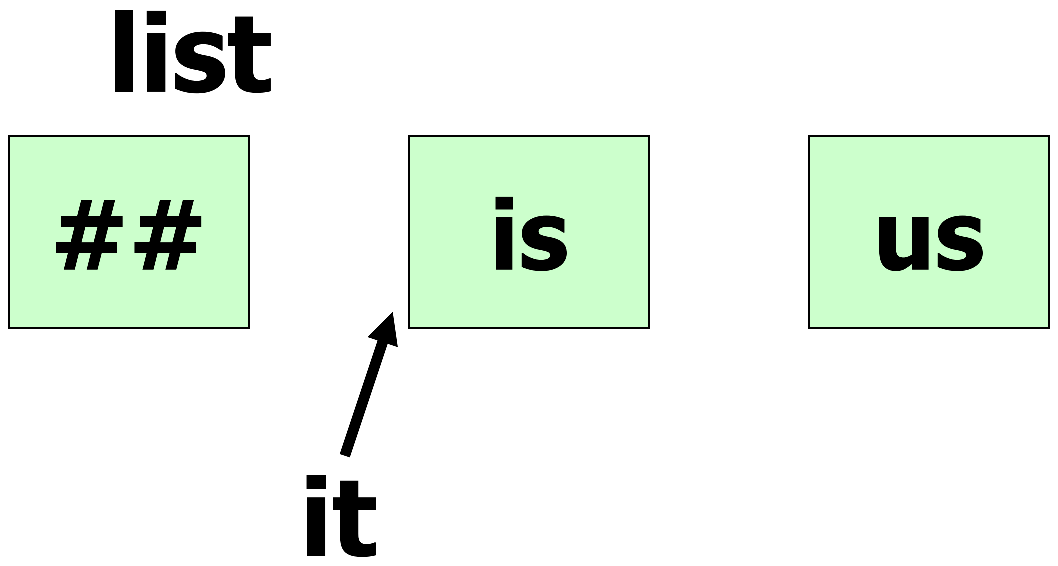
OUTPUT

```
is
us
us
[##, is, ##]
```

add() method



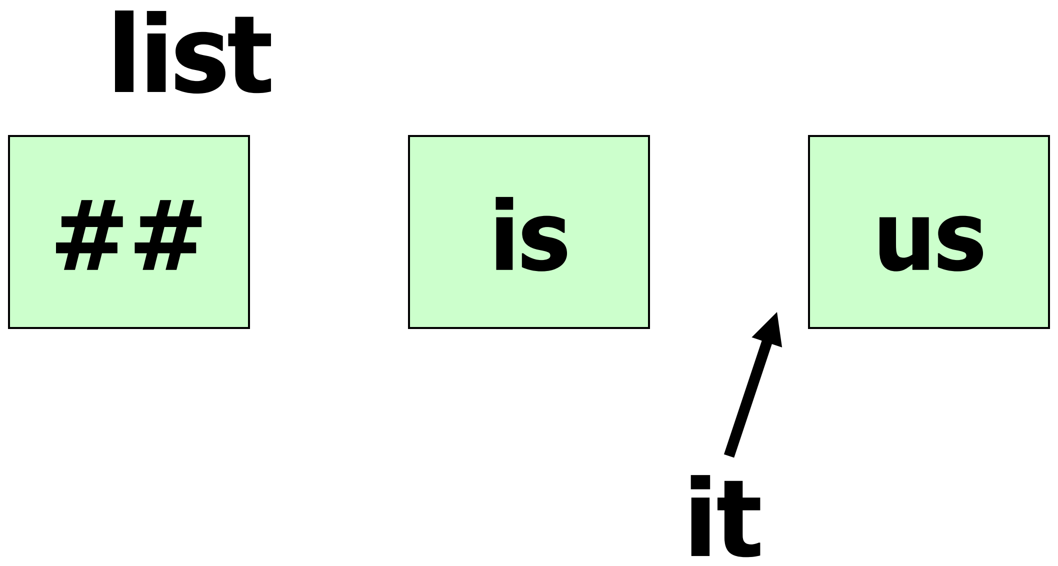
add() method



```
it.add("##");
```

add always adds the new item in front of the current spot.

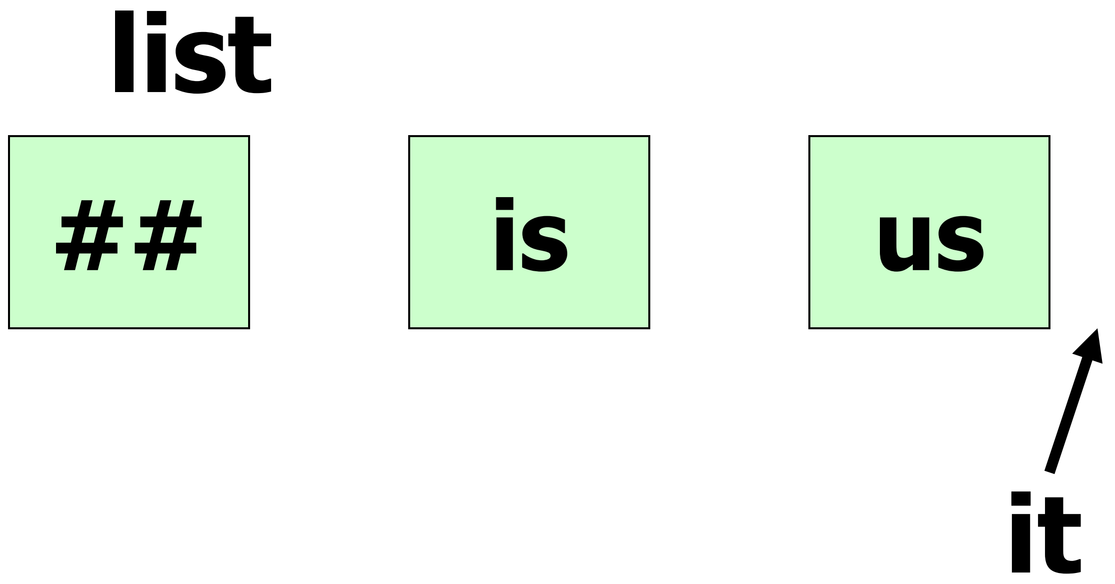
add() method



it.next();

next moves the iterator up one spot and returns a reference to the 2nd item.

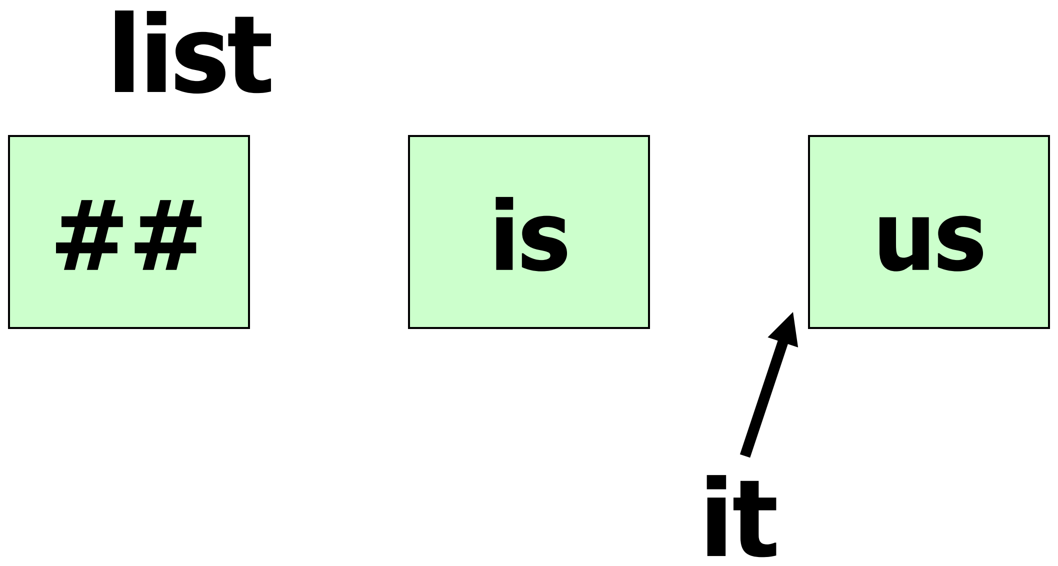
add() method



```
it.next();
```

next moves the iterator up one spot and returns a reference to the 3rd item.

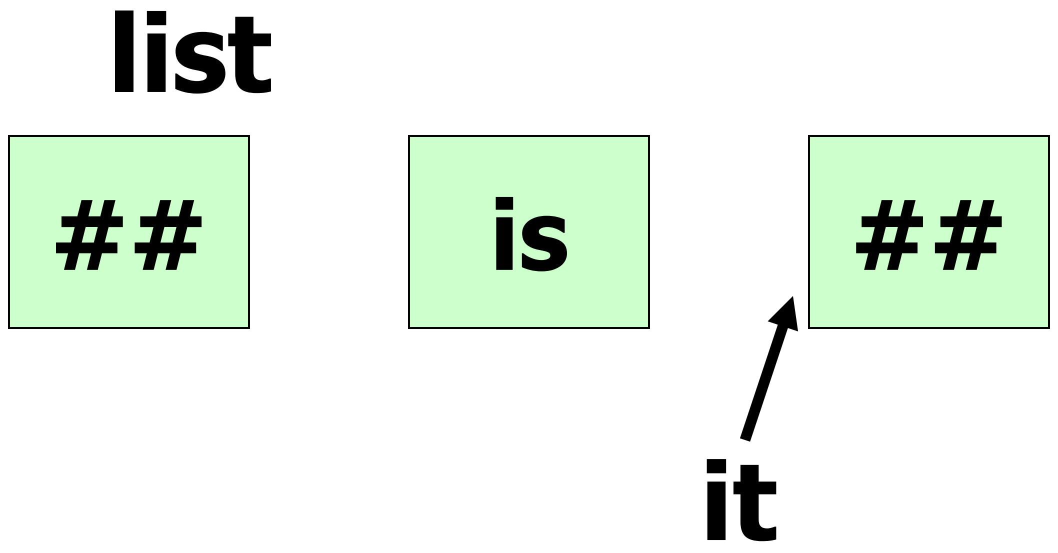
add() method



it.previous();

previous backs the iterator up one spot and returns a reference to the 3rd item.

add() method



```
it.set("##");
```

set always modifies the last reference returned by next or previous.

addone.java
addtwo.java

Modification rule

Modifications through an Iterator or ListIterator are always applied to the reference returned by the last next or previous call.

Pay attention to the direction you are going.

Iterator only goes one direction. ListIterator can go either direction.

For each loop

Counter based for loop

```
int[] array = {4,5,6,7};
int sum = 0;

for(int i=0; i<array.length; i++)
{
  sum += array[i];
}
```

for each loop

```
int array[] = {4,9,6,2,3};
int sum = 0;

for (int num  : array)
    sum = sum + num;
System.out.println(sum);
```

for each loop

```
ArrayList<Integer>  list;
list  = new ArrayList<Integer>();
list.add(3);
list.add(9);

for (Integer  num  : list)
     System.out.print(num + " ");
```

for each loop

```
ArrayList<Integer>  list;
list  = new  ArrayList<Integer>();
list.add(3);
list.add(9);

for (int num  : list)
    System.out.print(num + " ");
```

foreachloop.java
arraylistsplit.java

Work on Programs!

Crank

Some Code!

A+ Computer Science

Iterators