Iterators
New For Loop

Lab 05

© A+ Computer Science - www.apluscompsci.com

What is a reference?

What is a

© A+ Computer Science - www.apluscompsci.com

References

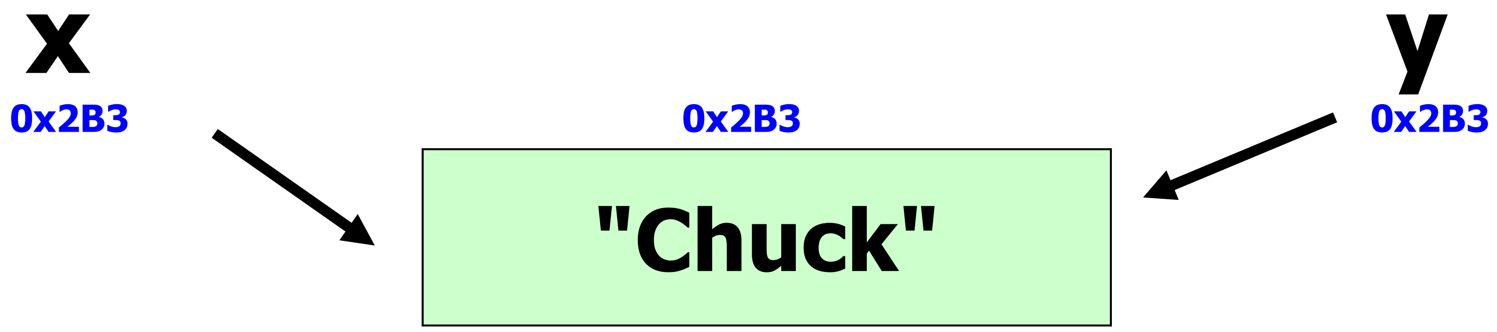
In Java, any variable that refers to an Object is a reference variable.

The variable stores the memory address of the actual Object.

References

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

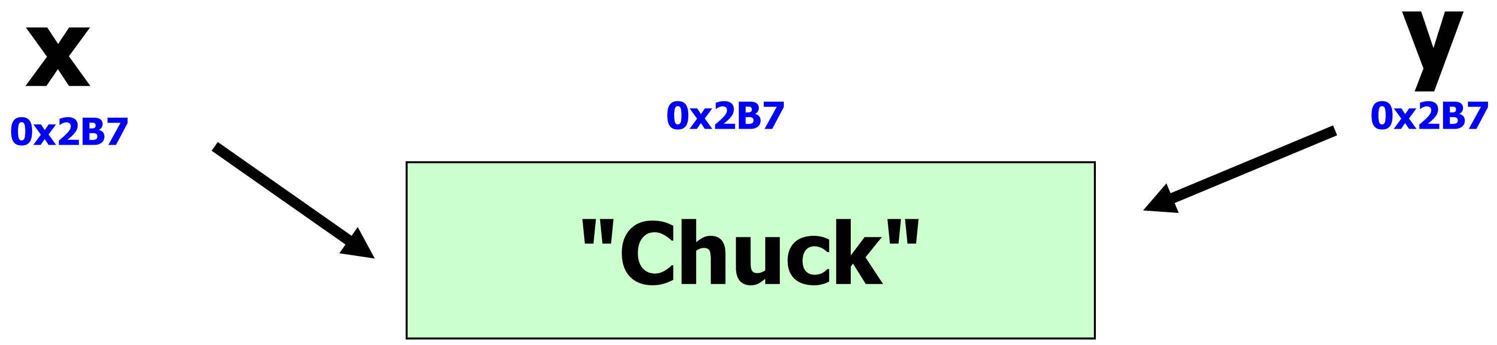
x and y store the same memory address.



References

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

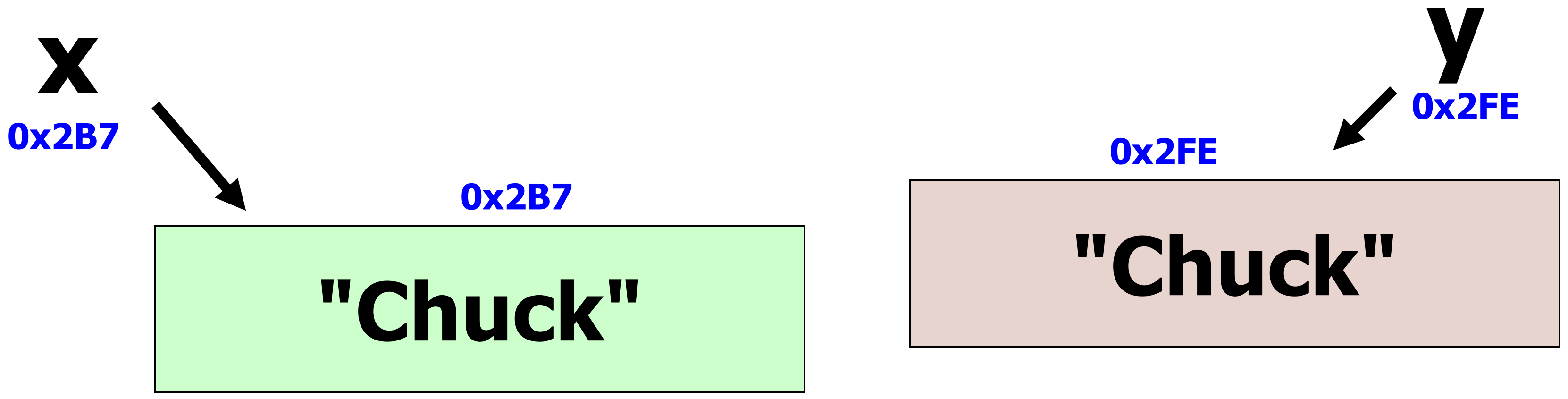
x and y store the same memory address.



References

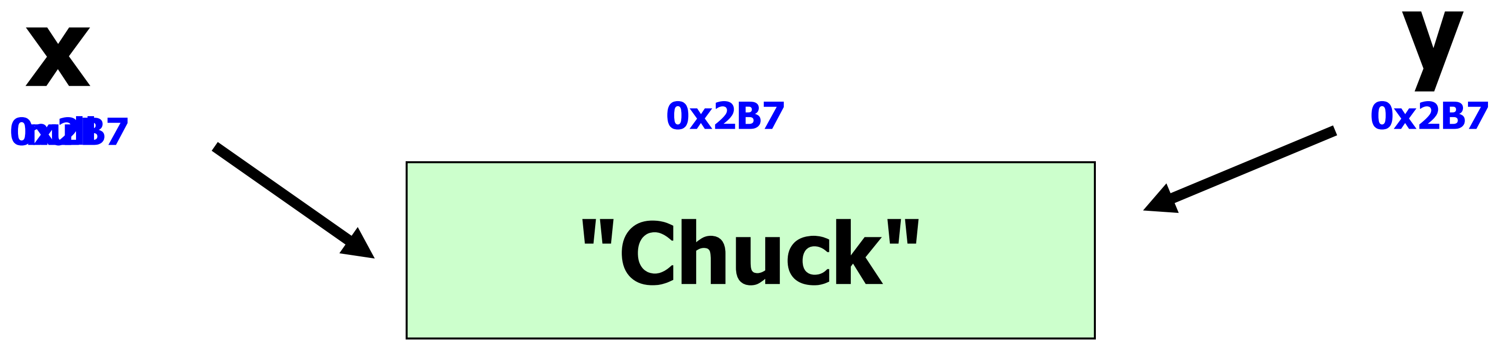
```
String x = new String("Chuck");
String y = new String("Chuck");
```

x and y store different memory addresses.



References

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

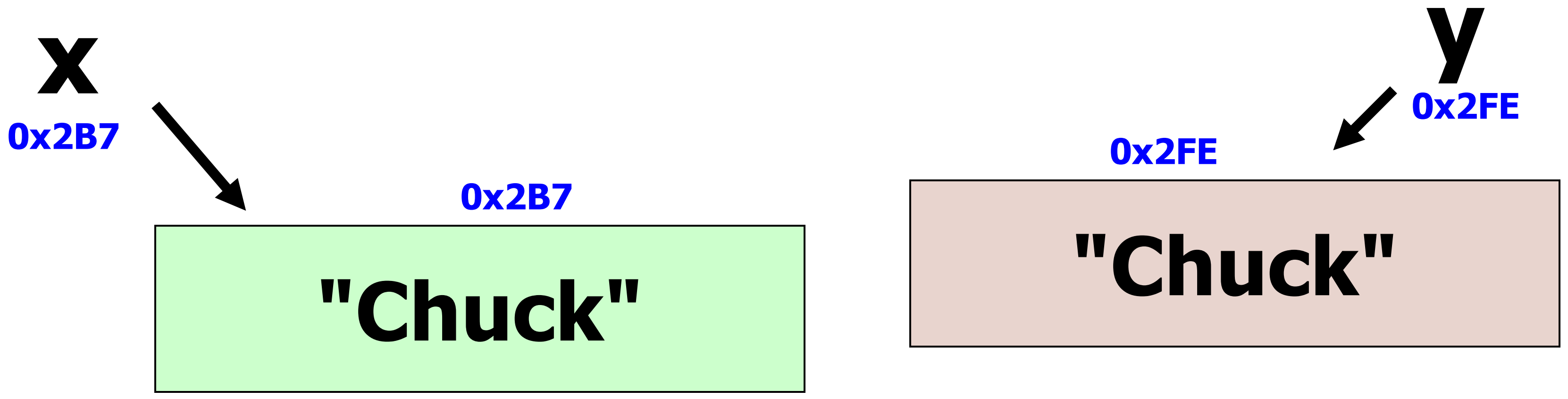


© A+ Computer Science - www.apluscompsci.com

References

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

x and y store different memory addresses.



© A+ Computer Science - www.apluscompsci.com

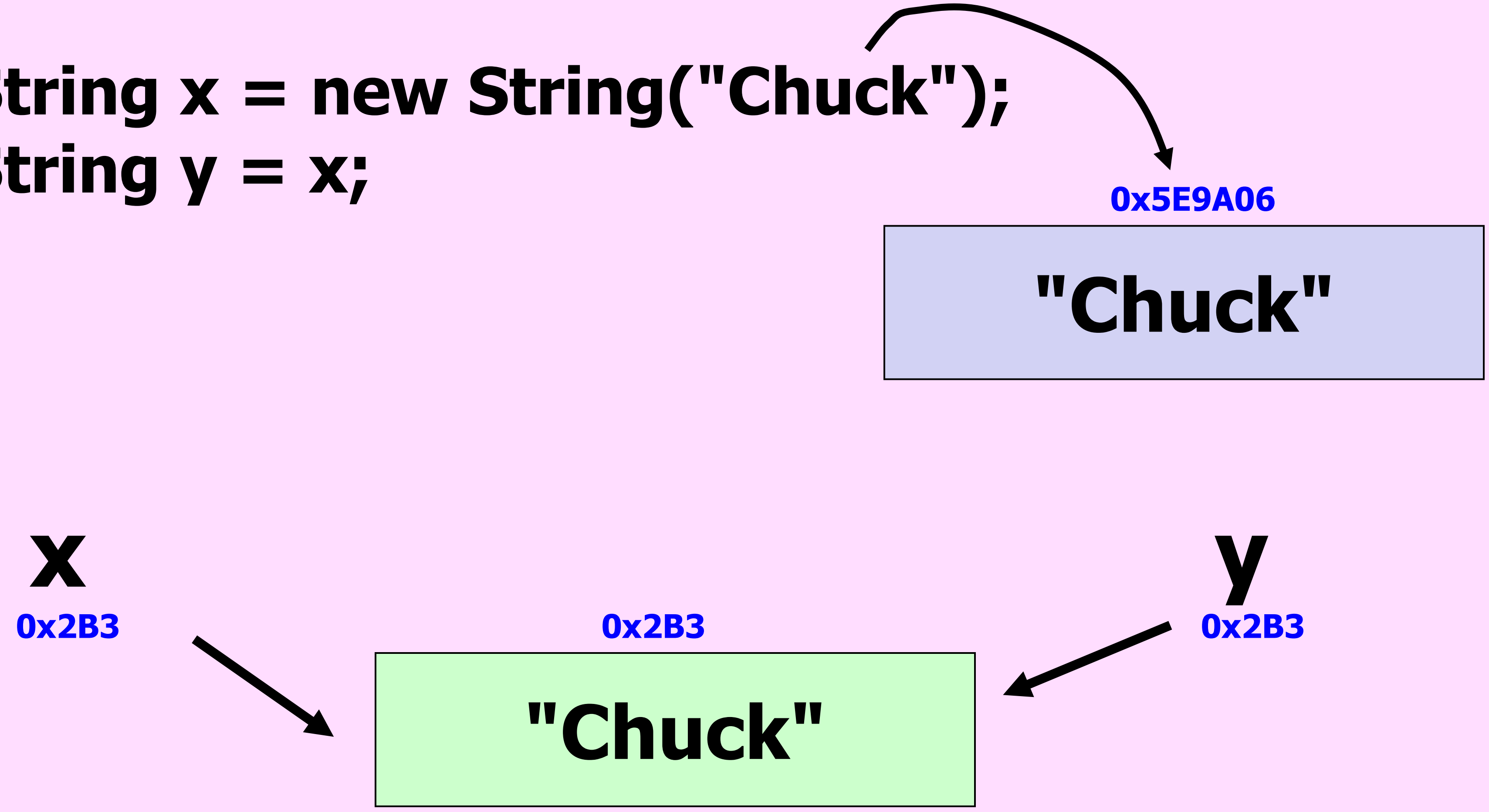
Contest Puzzlers

How many String object does this code create?

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

Contest Puzzlers

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



© A+ Computer Science - www.apluscompsci.com

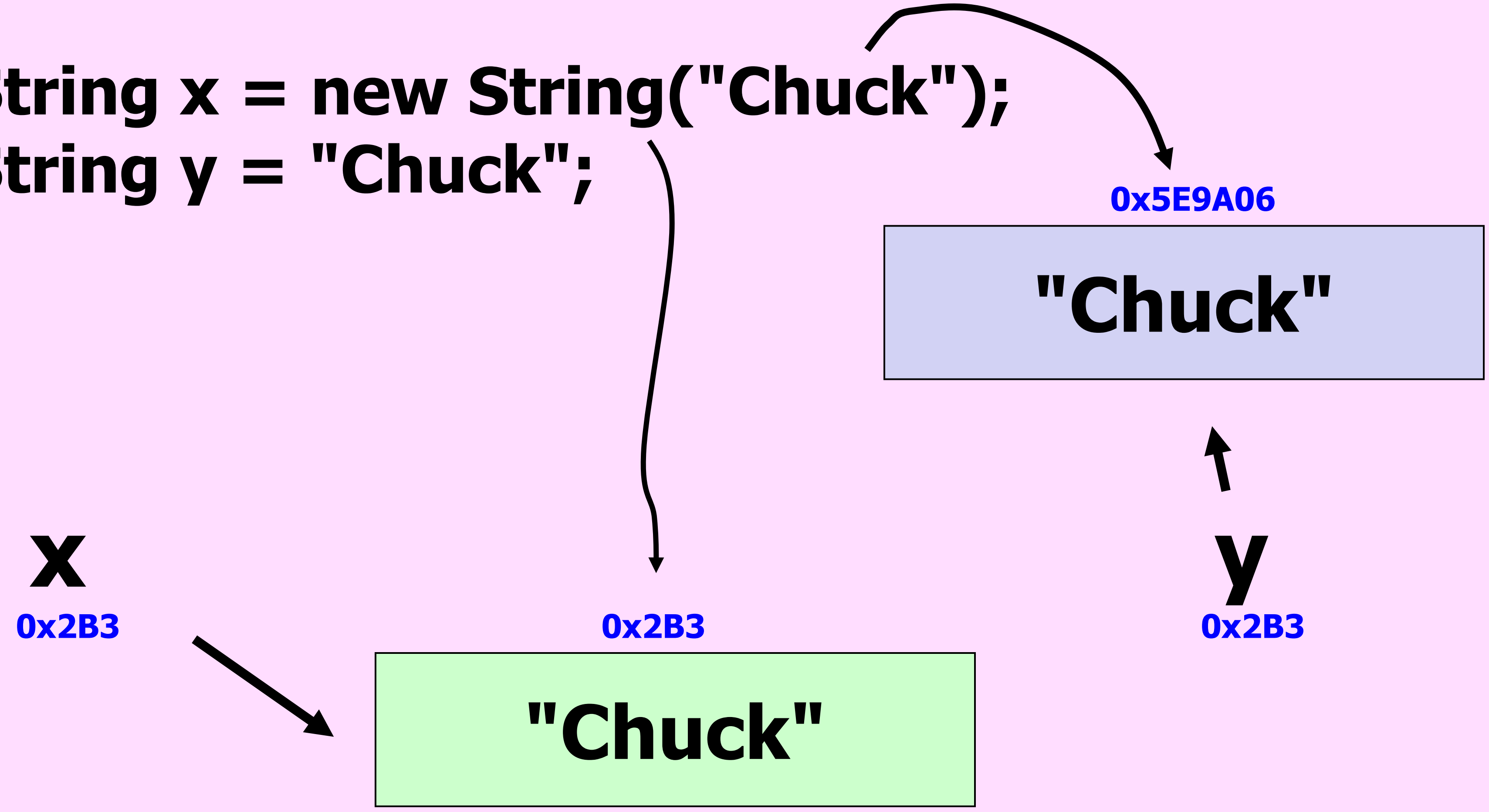
Contest Puzzlers

How many String object does this code create?

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

Contest Puzzlers

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



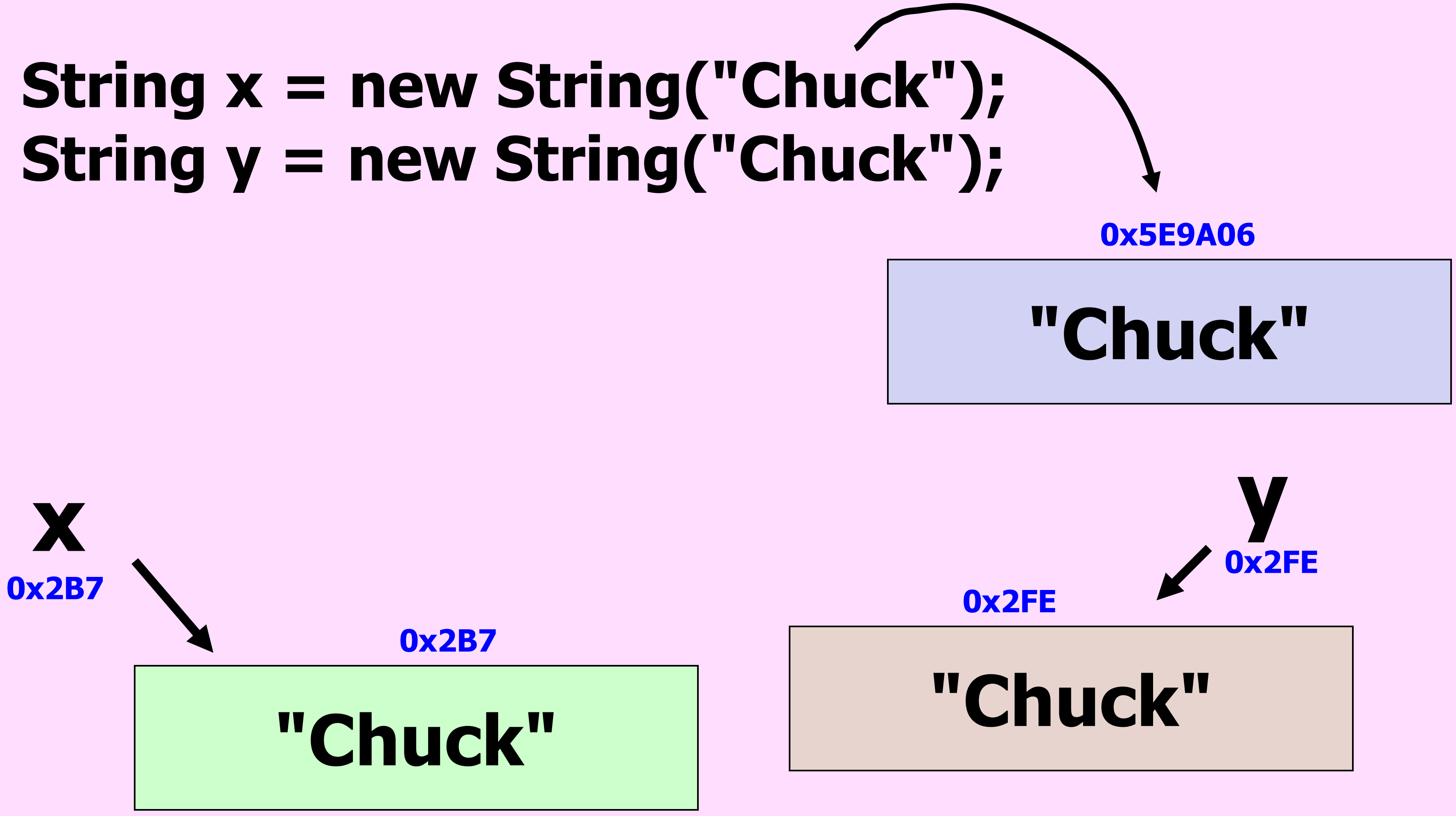
Contest Puzzlers

How many String object does this code create?

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

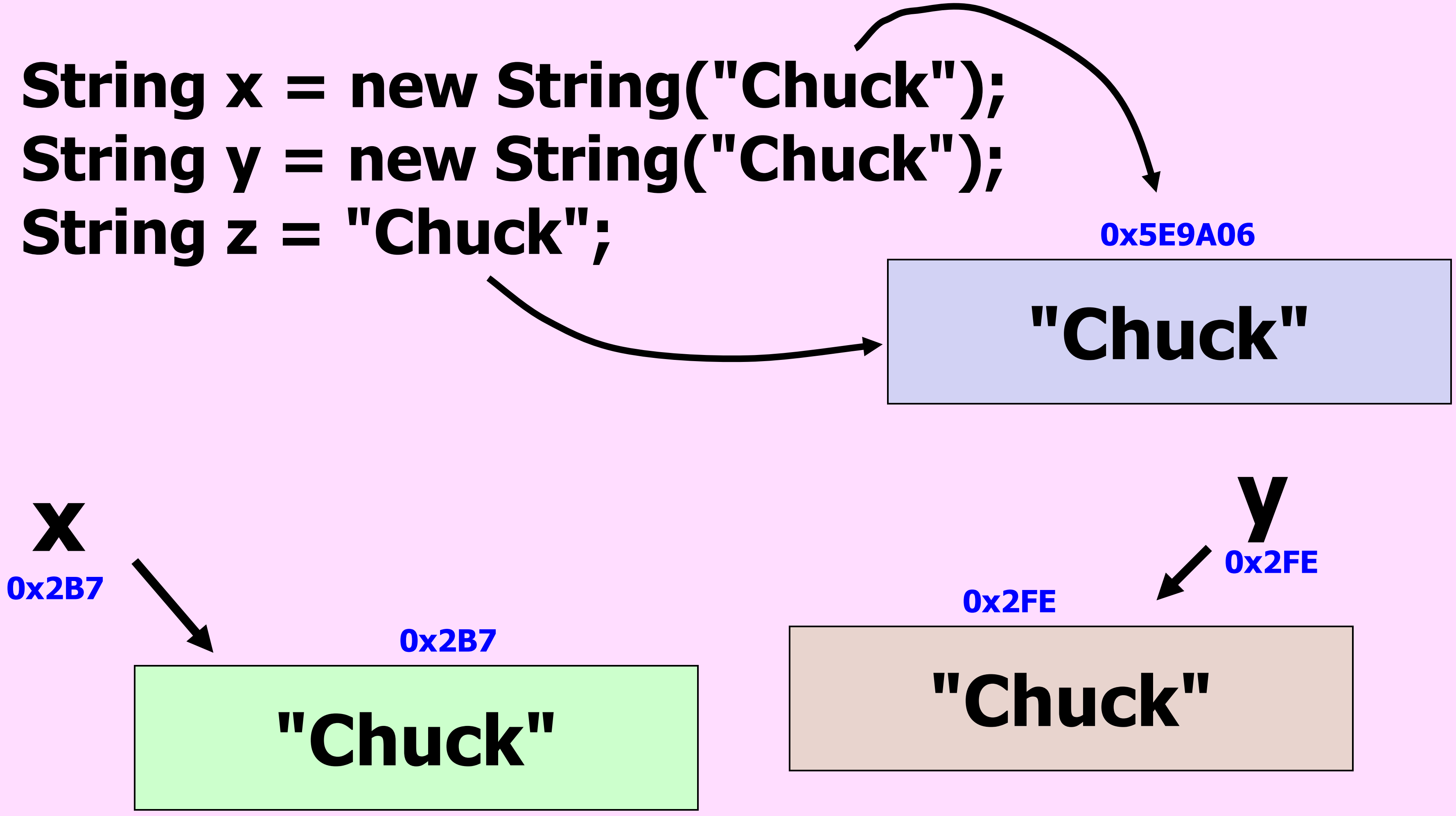
Contest Puzzlers

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



Contest Puzzlers

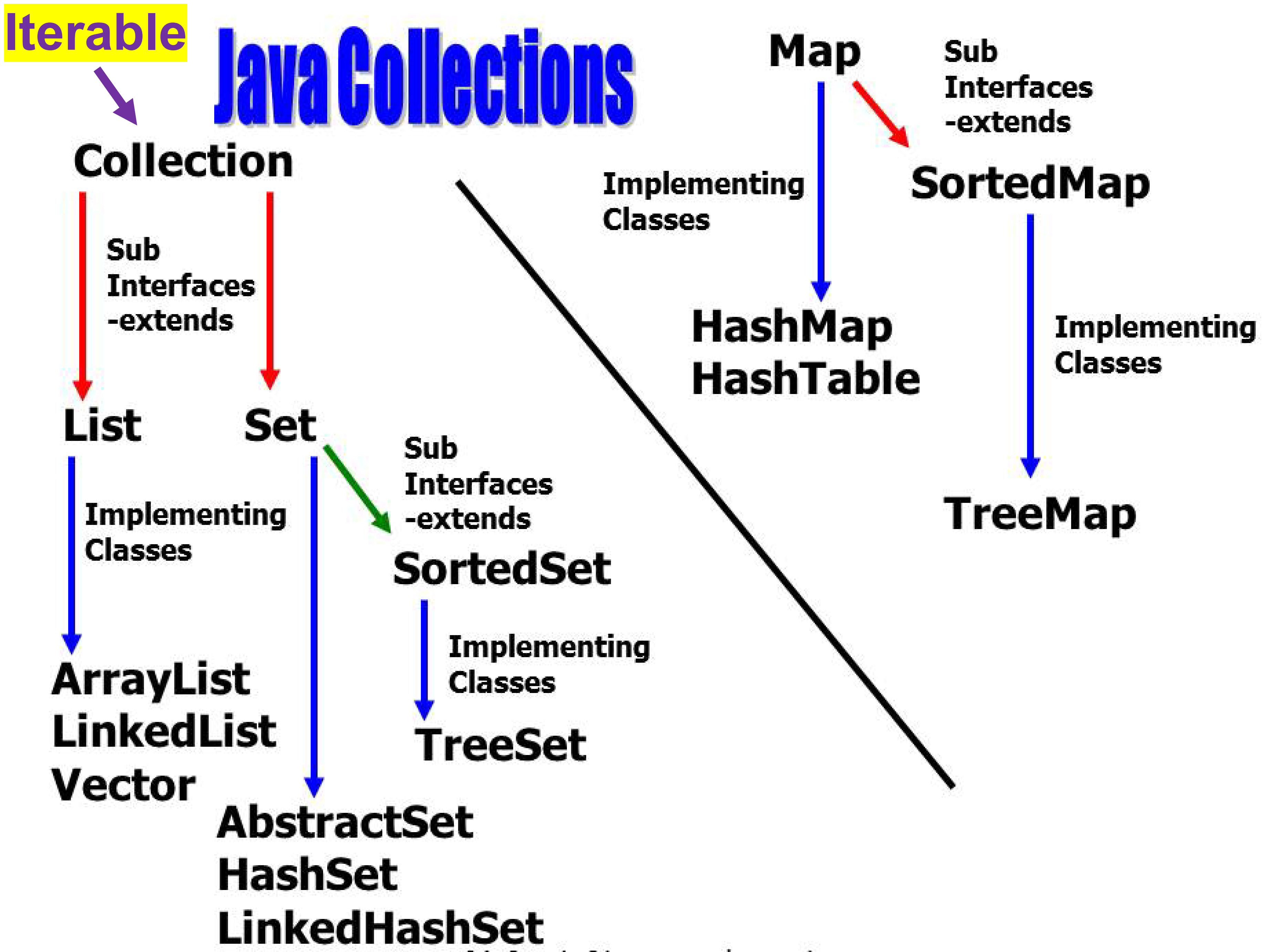
```
String x = new String("Chuck");
String y = new String("Chuck");
String z = "Chuck";
```



references.java

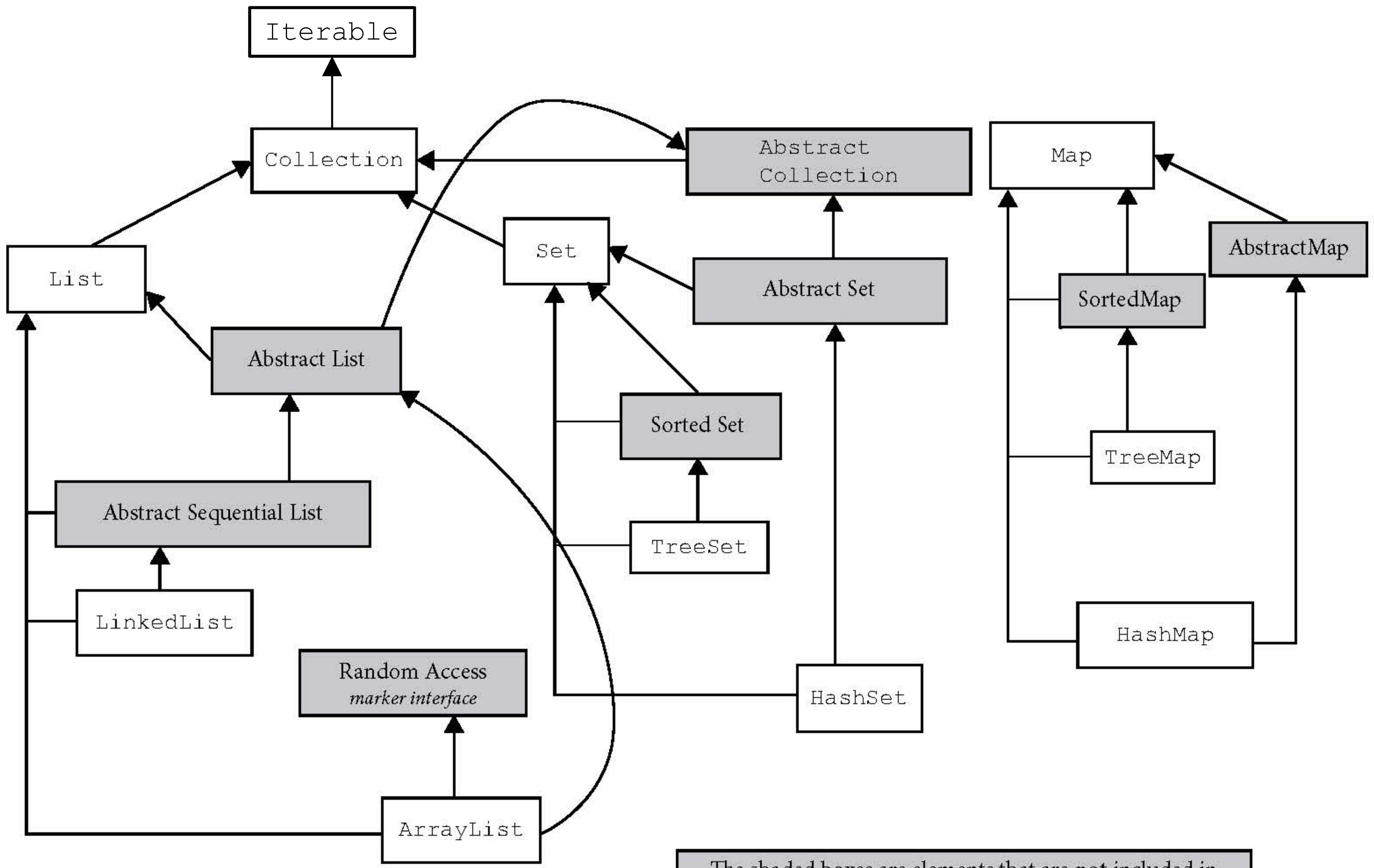
Iterators

Java Collections



© A+ Computer Science - www.apluscompsci.com

The Java Collections Hierarchy



The shaded boxes are elements that are **not** included in the AP-testable subset.

Java Iterators

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

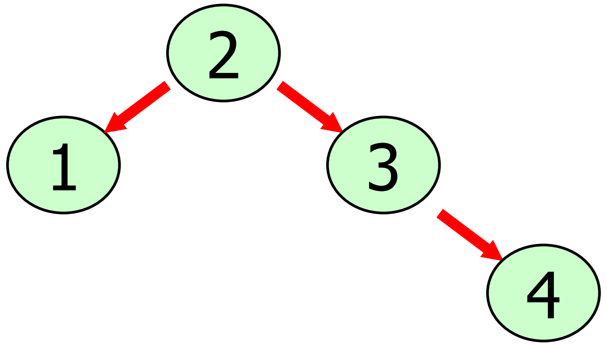
Iterators allow you to go from item to item through a collection.

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.

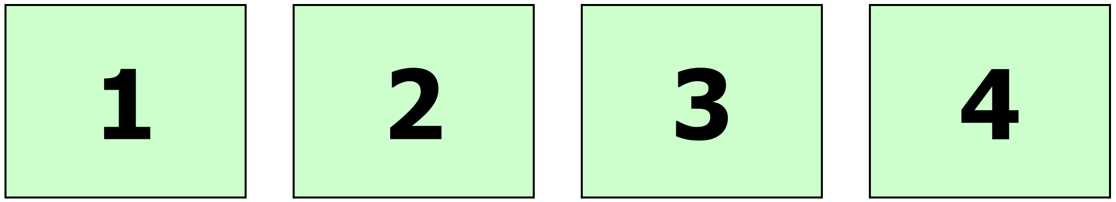
For some Collections, TreeMap and HashSet for instance, the underlying data structures are not sequentially organized like an array. For example, a tree has nodes all over the place.



© A+ Computer Science - www.apluscompsci.com

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.



© A+ Computer Science - www.apluscompsci.com

Iterator Interface

© A+ Computer Science - www.apluscompsci.com

next()

You don't call a constructor to create an iterator. You create your collection and then ask it for an iterator:

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

Iterator<String> it = words.iterator();
```

© A+ Computer Science - www.apluscompsci.com

Iterator frequently used methods	
Name	**Use**
E **next()**	**returns a reference to the next item**
void **remove()**	**removes the last ref returned by next**
boolean **hasNext()**	**checks to see there are more items**

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

next()

```
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()

```
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
```

next()

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

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

OUTPUT

```
at
error
```

A NoSuchElementException is thrown.

iteratorone.java

hasnext.java

hasNext()

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

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

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

OUTPUT

```
at
is
of
```

hasNext()

Note that if you don’t use generics, Java only knows it’s some kind of Object.

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

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

This would output the same thing:

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

OUTPUT

```
at
is
of
```

© A+ Computer Science - www.apluscompsci.com

hasNext()

But I couldn't call any String methods:

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

hasNext()

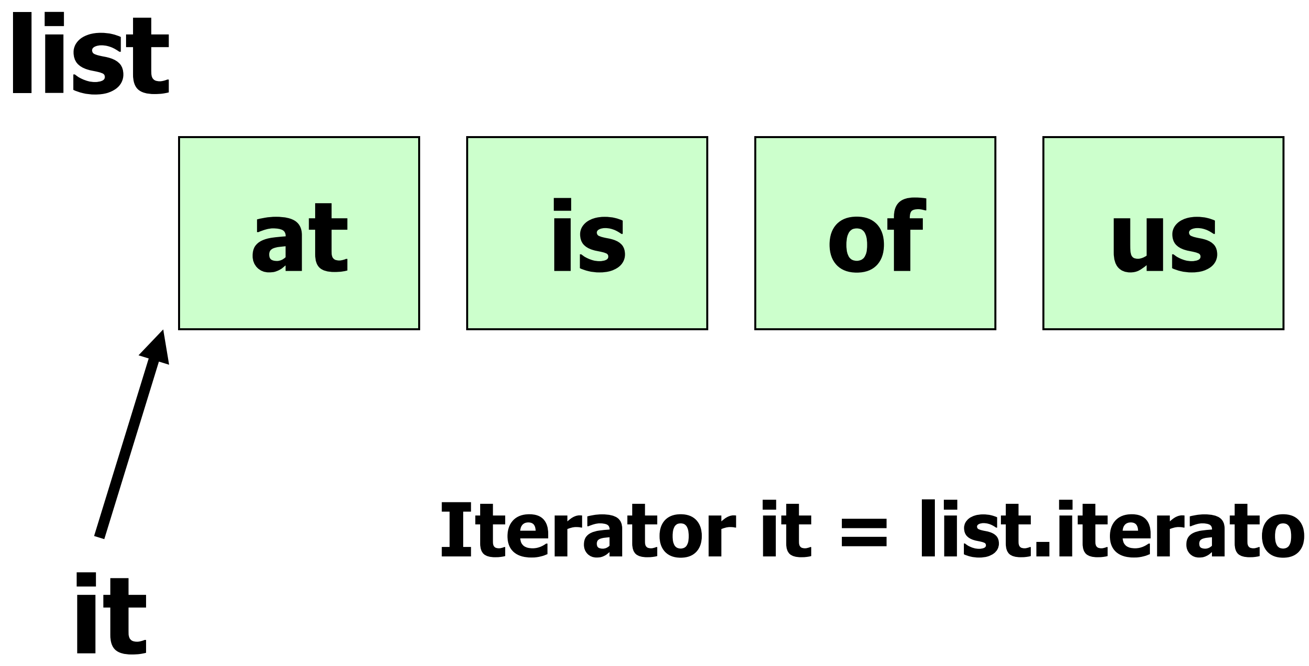
Now I can:

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

© A+ Computer Science - www.apluscompsci.com

Inside the next() method

next()



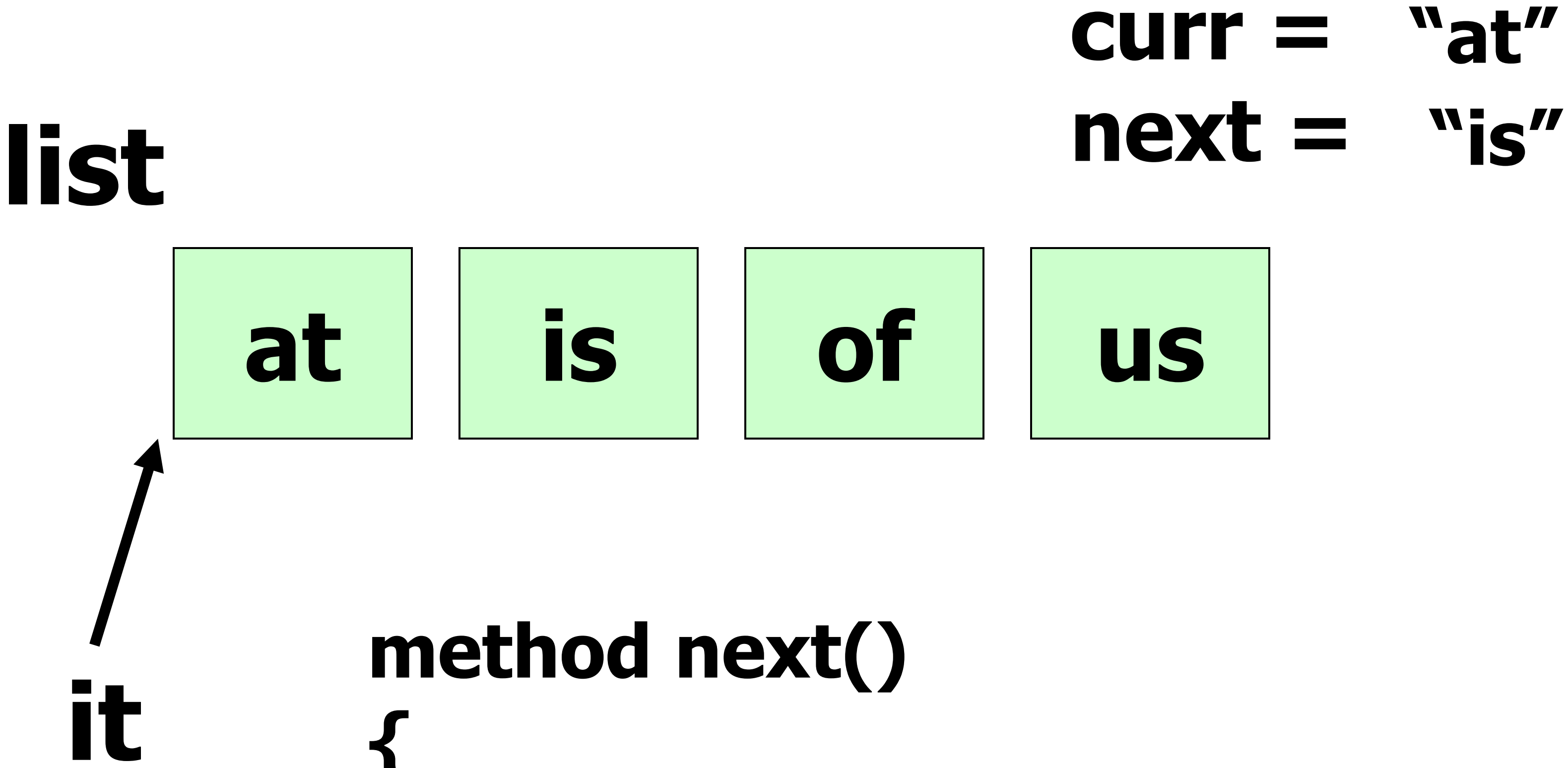
```
Iterator it = list.iterator();
```

```
curr = null
next = "at"
```

next()

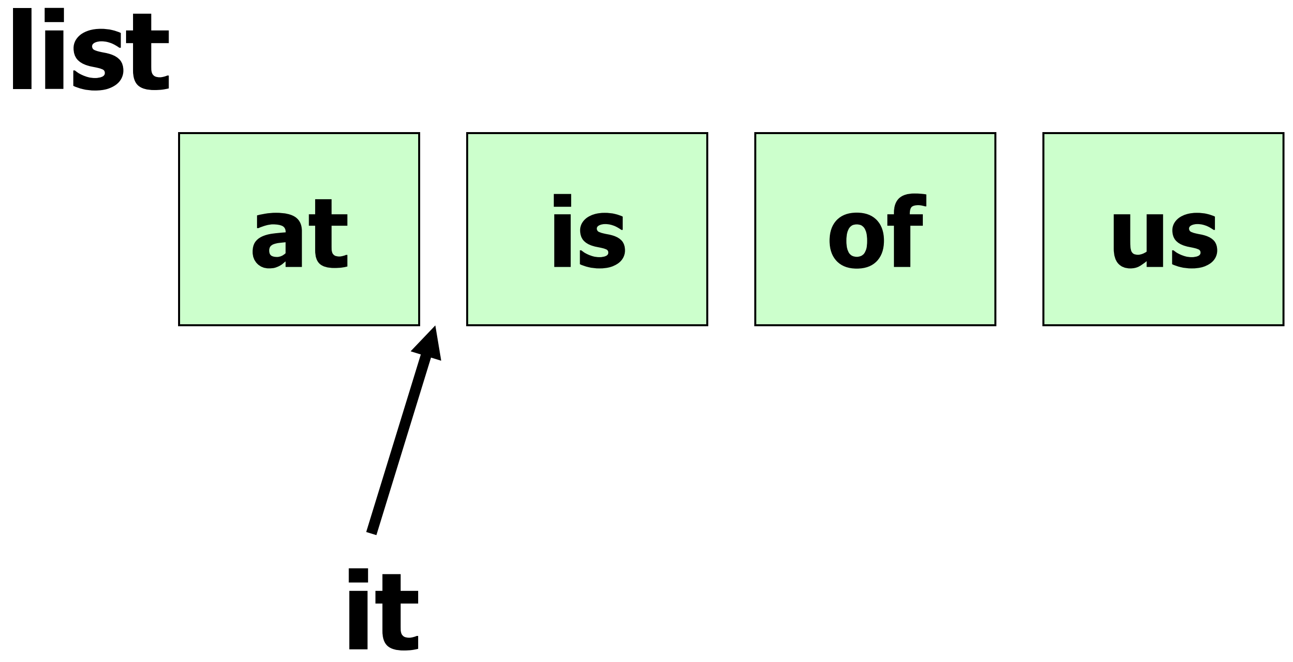
```
method next()
{
   curr = next
   next = next ref in the collection
   return curr
}
```

next()



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

next()



it.next();

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

remove()

The Iterator remove method removes the last element iterated over.

I.e. the last element returned by the next() method.

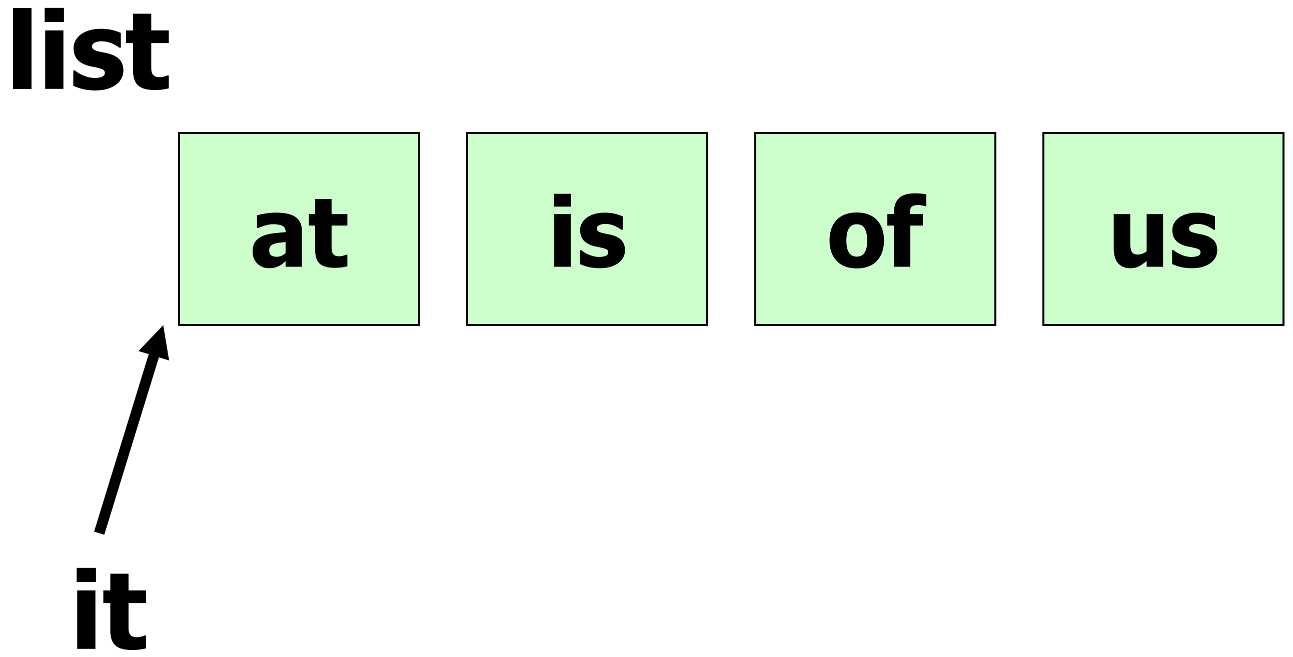
remove()

```
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

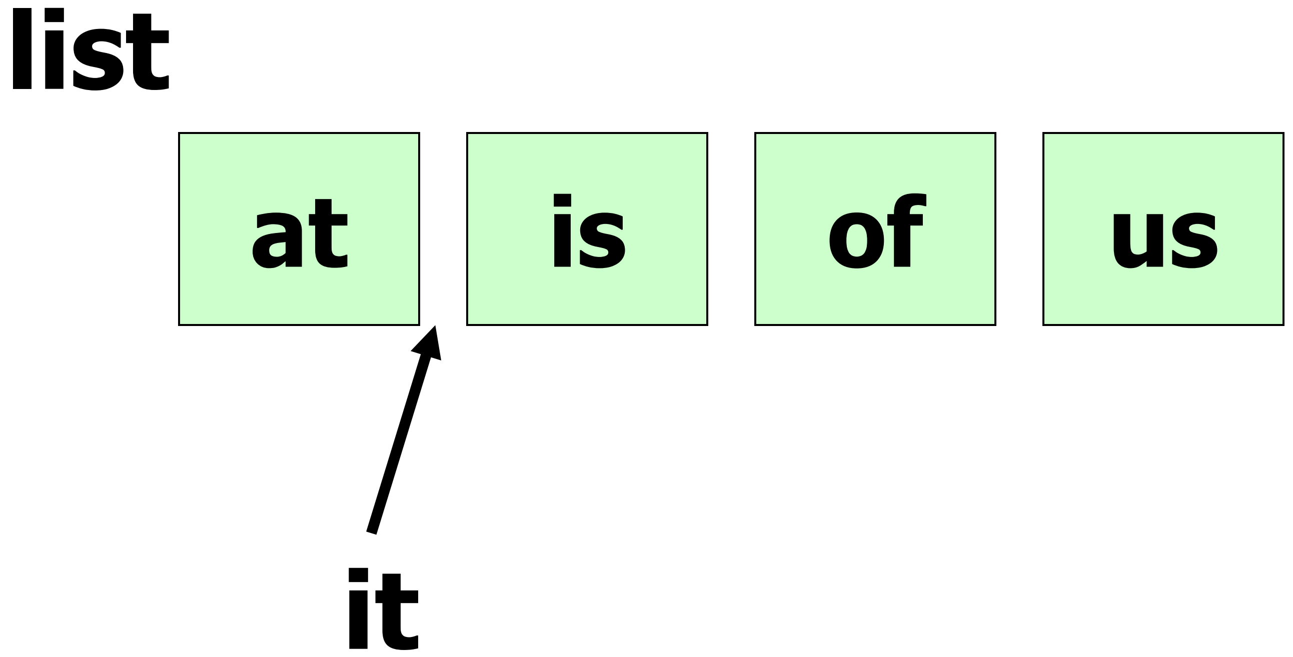
```
at
is
[is, of]
```

remove()



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

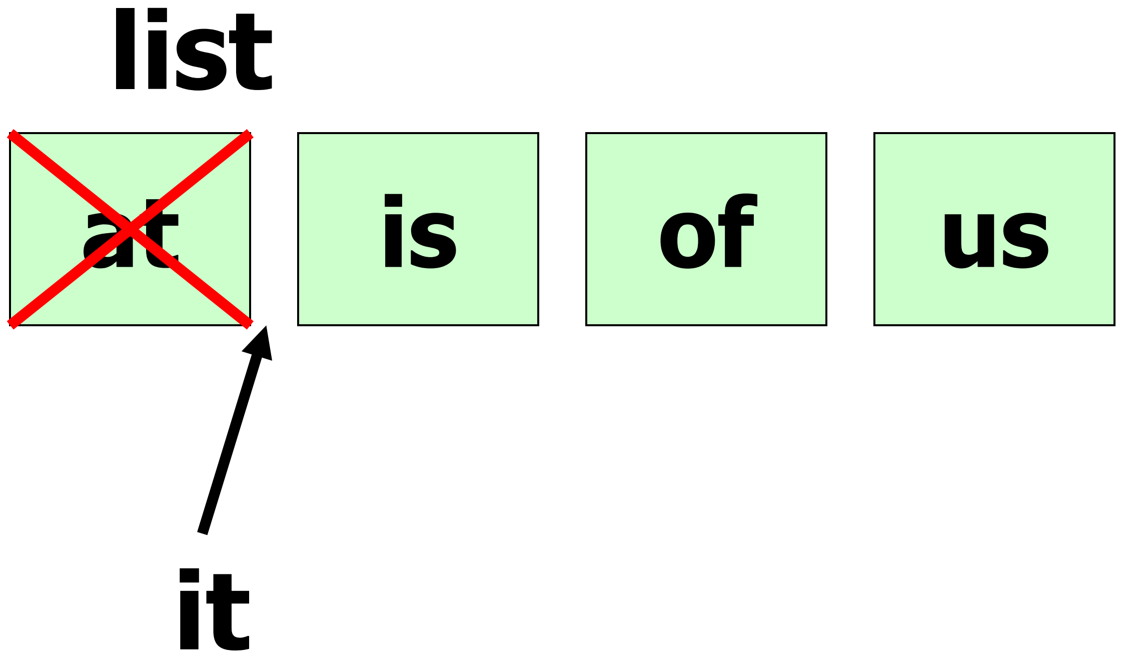
remove()



it.next();

next moves the iterator one spot and returns a reference to at.

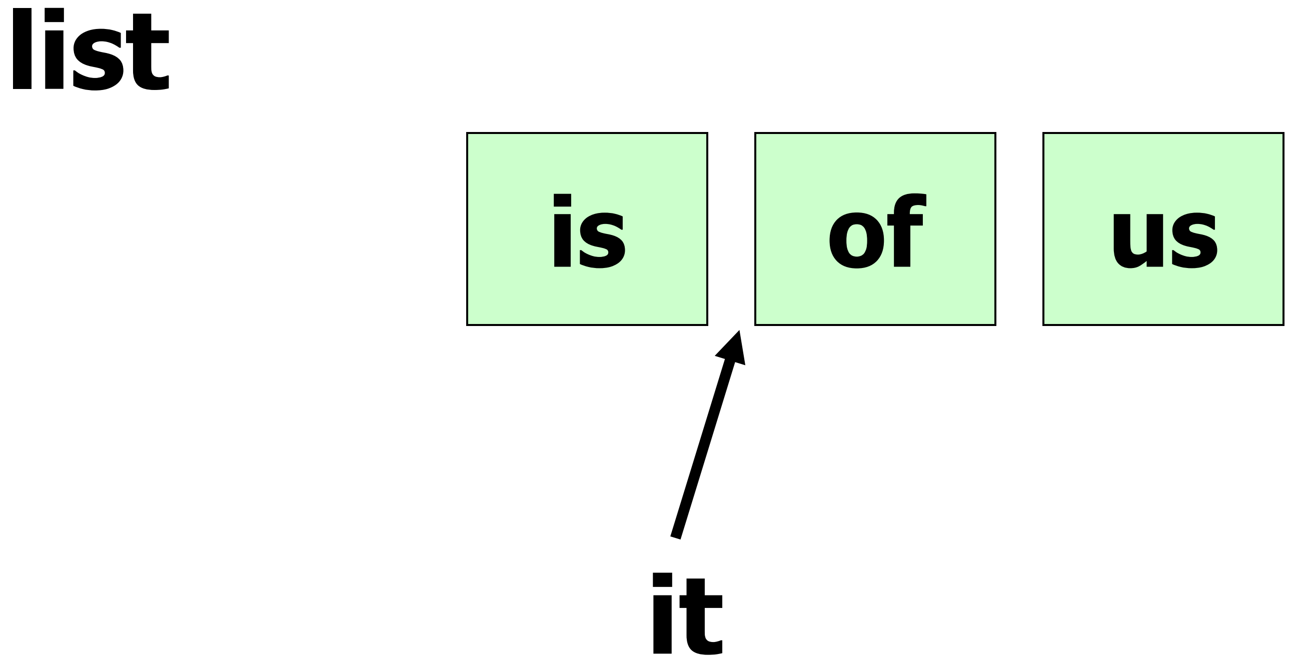
remove()



it.remove();

remove always modifies the last reference returned by next.

remove()



it.next();

next moves the iterator one spot and returns a reference to **is**.

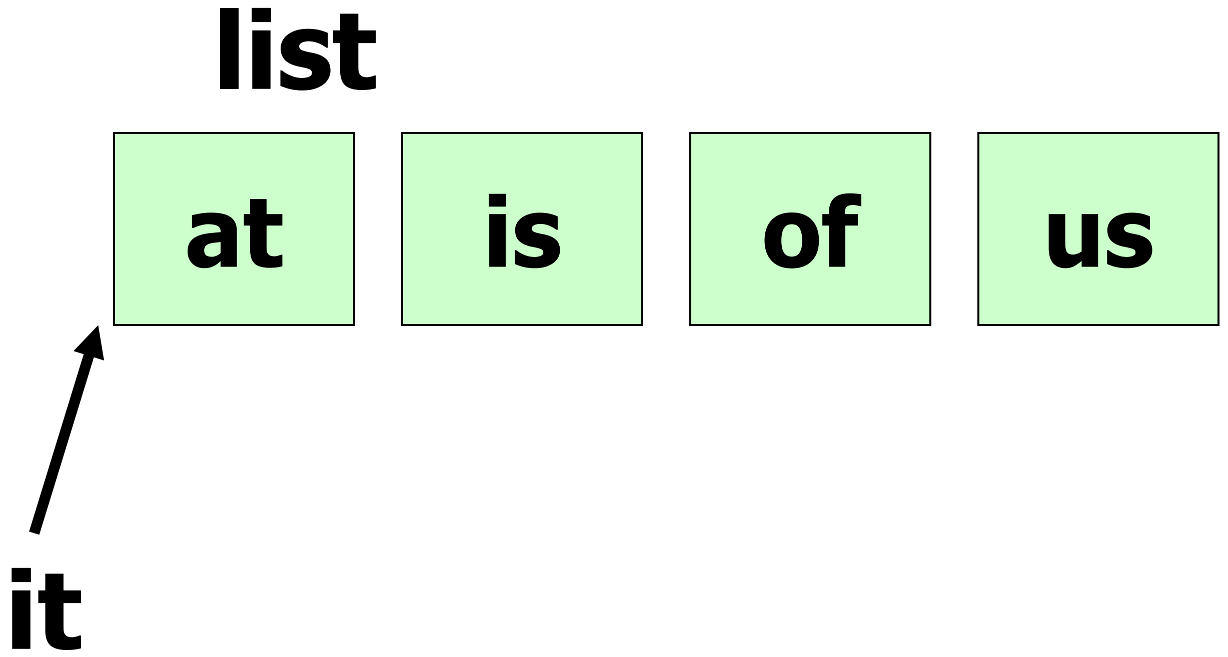
remove()

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

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

Iterator<String> it = words.iterator();
                                  curr = null, next = "at"
out.println(it.next());           curr = "at", next = "is"
it.remove();                      curr = null, next = "is"
it.remove();
```

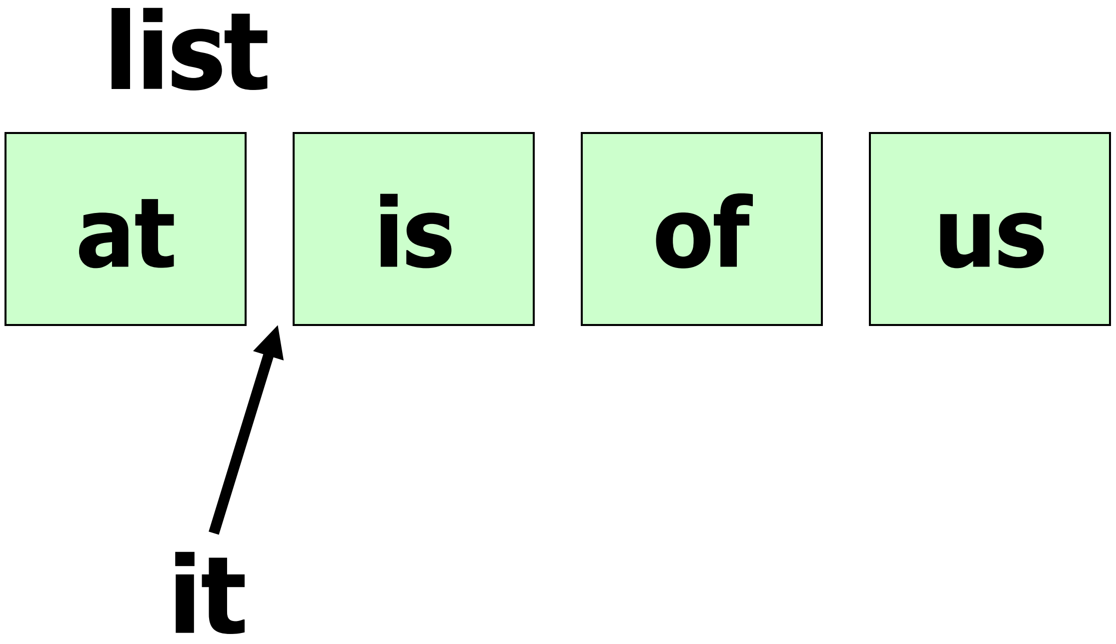
OUTPUT

```
at
error
```

remove()



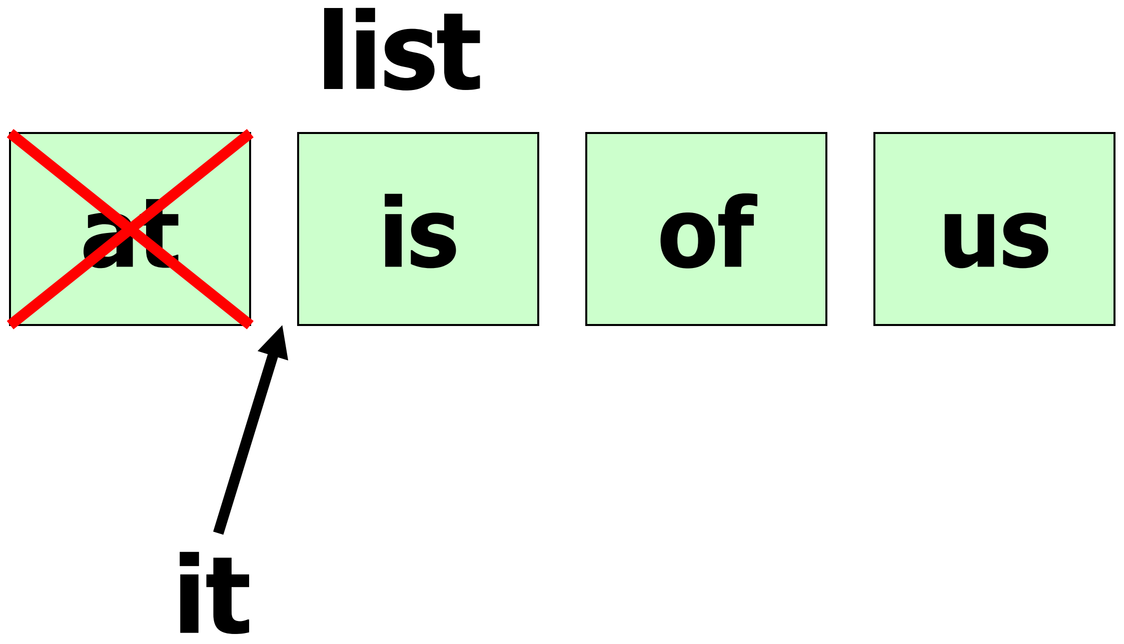
remove()



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

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

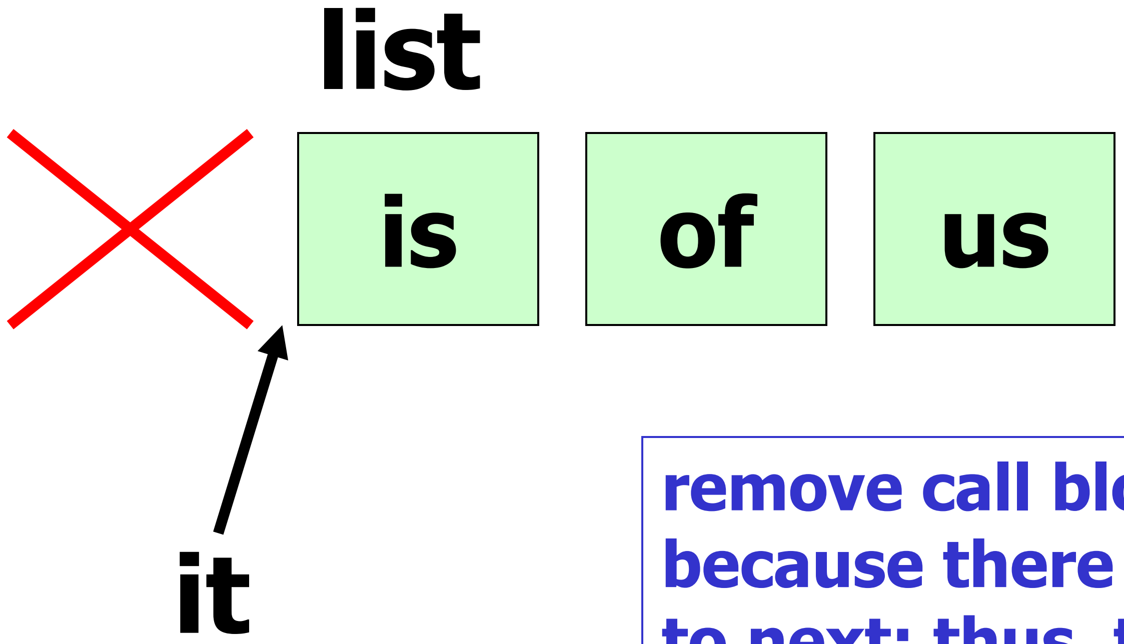
remove()



```
it.remove();
```

remove always modifies the last reference returned by next.

remove()



```
it.remove();
```

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

Throws an IllegalStateException

© A+ Computer Science - www.apluscompsci.com

AN ITERATOR DOESN'T STORE ANYTHING!

When you make an iterator, it's not storing a separate copy of the list. It simply manages the list for you and <u>expects you to use it exclusively</u>.

ConcurrentModificationException

```
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());
words.remove(1);
System.out.println(it.next());
```

OUTPUT

```
at
error
```

No error if you don’t use Iter

```
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());
words.remove(1);
```

OUTPUT

```
at
```

Okay … but not okay

```
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());
words.remove(1);
it = words.iterator();
System.out.println(it.next());
System.out.println(it.next());
```

OUTPUT

```
at
at
of
```

What wrong with this?

```
List <String>list = new ArrayList<String>();
list.add("1");
list.add("2");
list.add("3");
Iterator <String>iter = list.iterator ();
iter.remove();
iter = list.iterator();
while (iter.hasNext())
    System.out.print((String)iter.next());
```

IllegalStateException is thrown
You must call next() before remove()!

```
List <String>list = new ArrayList<String>();
list.add("1");
list.add("2");
list.add("3");
Iterator <String>iter = list.iterator ();
iter.remove();
iter = list.iterator();
while (iter.hasNext())
    System.out.print((String)iter.next());
```

What’s wrong with this loop?

```
int totalLengths(ArrayList<String> list) {
   Iterator<String> iter = data.iterator();
   int total = 0;
   while (iter.hasNext()) {
      if (iter.next().length() < 5)
         total += iter.next().length();
   }
   return total;
}
```

The fix: call next() once

```
int totalLengths(ArrayList<String> list) {
   Iterator<String> iter = data.iterator();
   int total = 0;
   while (iter.hasNext()) {
      String word = iter.next();
      if (word.length() < 5)
         total += word.length();
   }
   return total;
}
```

removeone.java

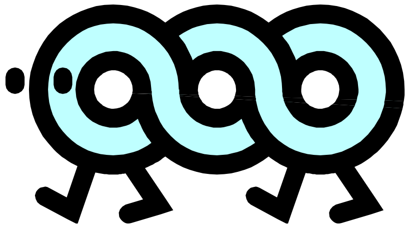
removetwo.java

the new for loop

new for loop

In an enhanced for loop, Java will automatically create an Iterator and use it to iterate through the array or Collection.

```
for (int num : array)

for (Integer value : list)
```



new for loop

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

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

OUTPUT

```
3 9 2
```

newforone.java

Start work on Lab 5

© A+ Computer Science - www.apluscompsci.com