Iterators New For Loop

Lab 05

What is a reference?

References

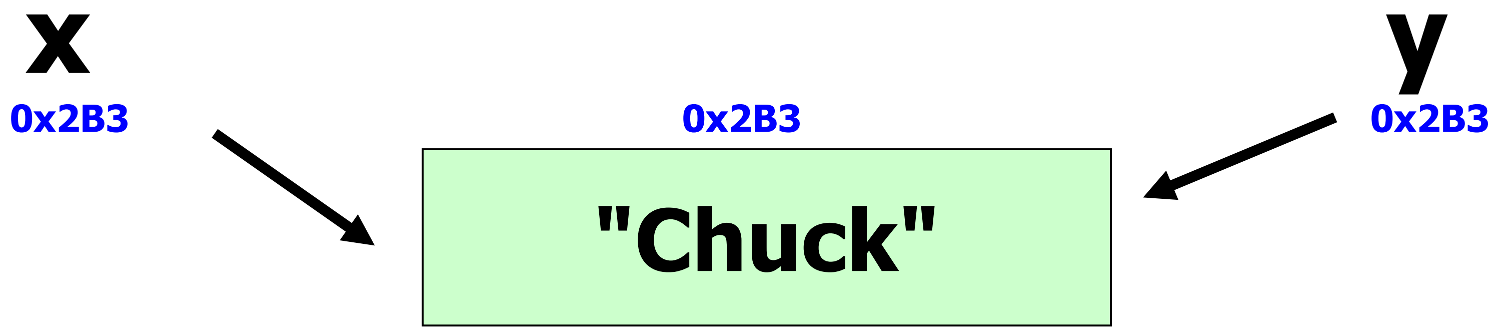
In Java, any variable that refers to an Object is a reference variable.

The variable stores the memory address of the actual Object.

References

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

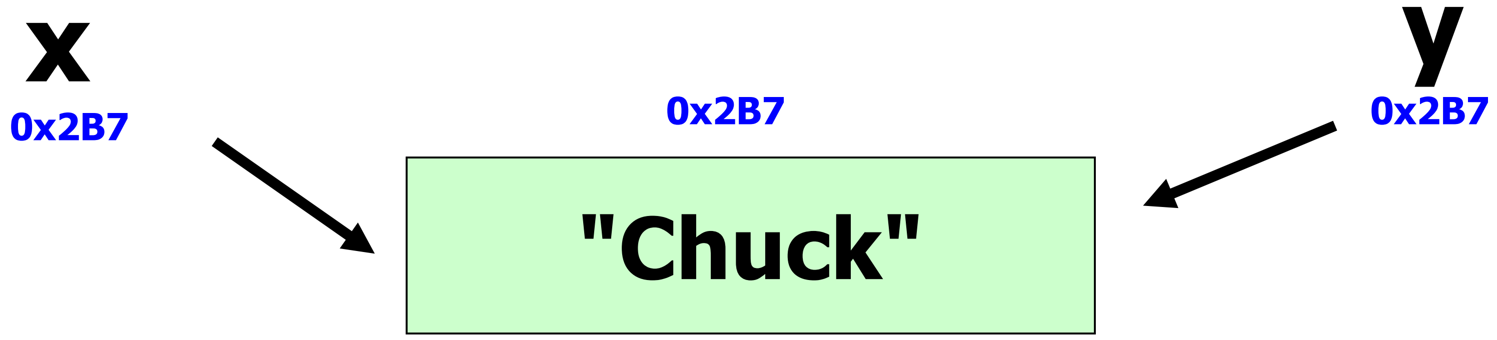
x and y store the same memory address.



References

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

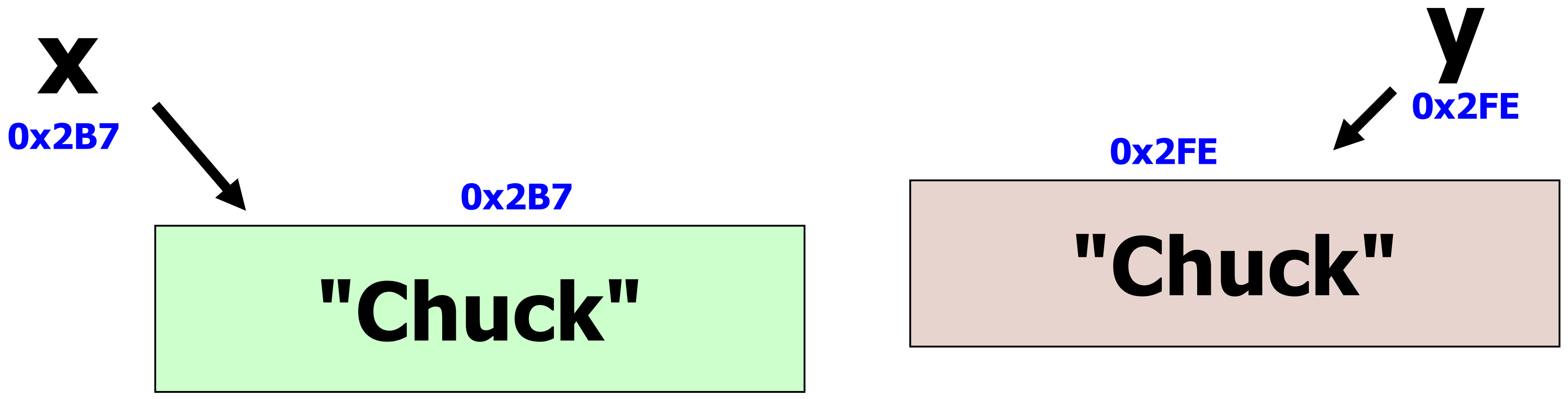
x and y store the same memory address.



References

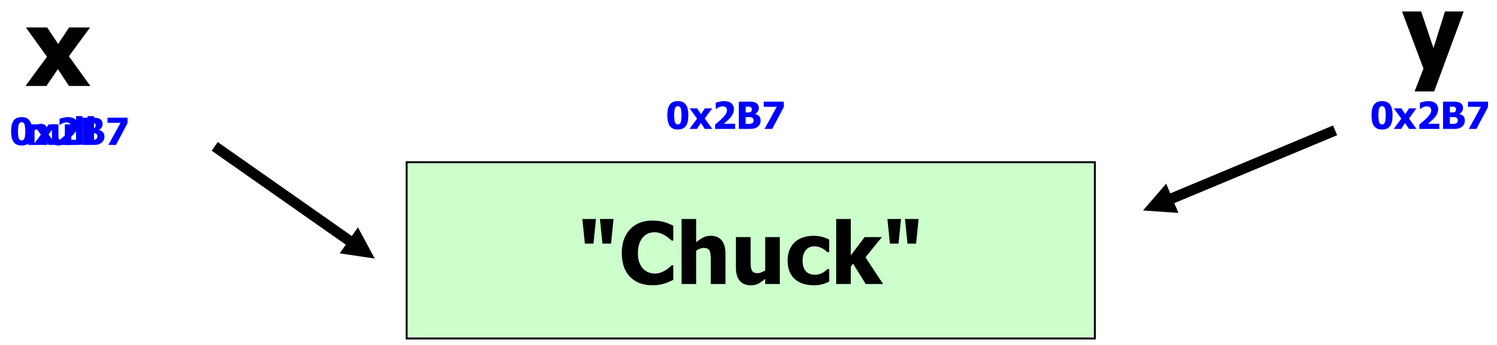
```
String x = new String("Chuck");
String y = new String("Chuck");
```

x and y store different memory addresses.



References

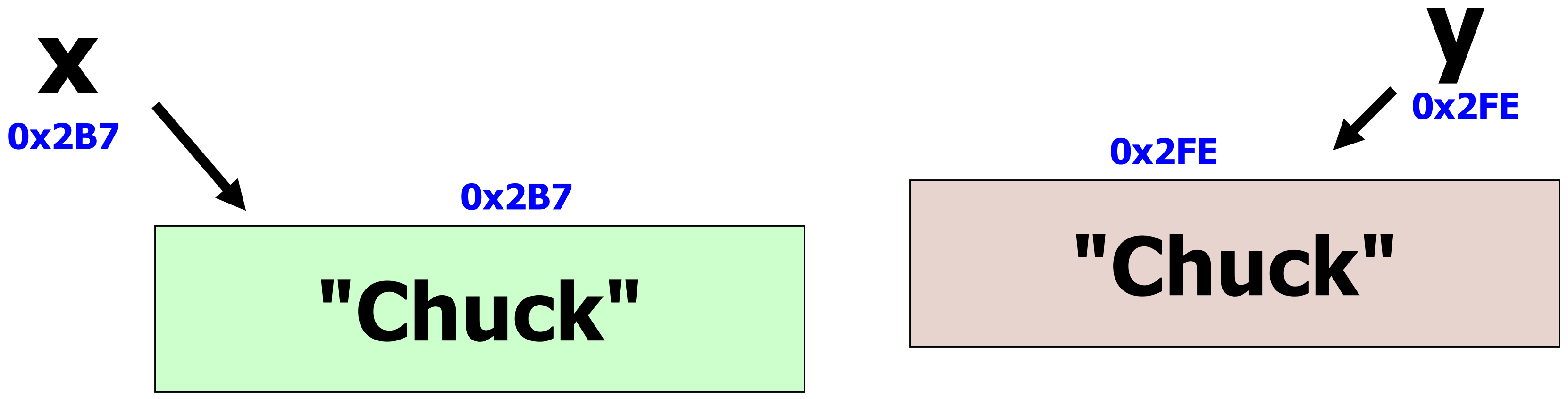
```
String x = "Chuck";
String y = "Chuck";
x = null;
```



References

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

x and y store different memory addresses.



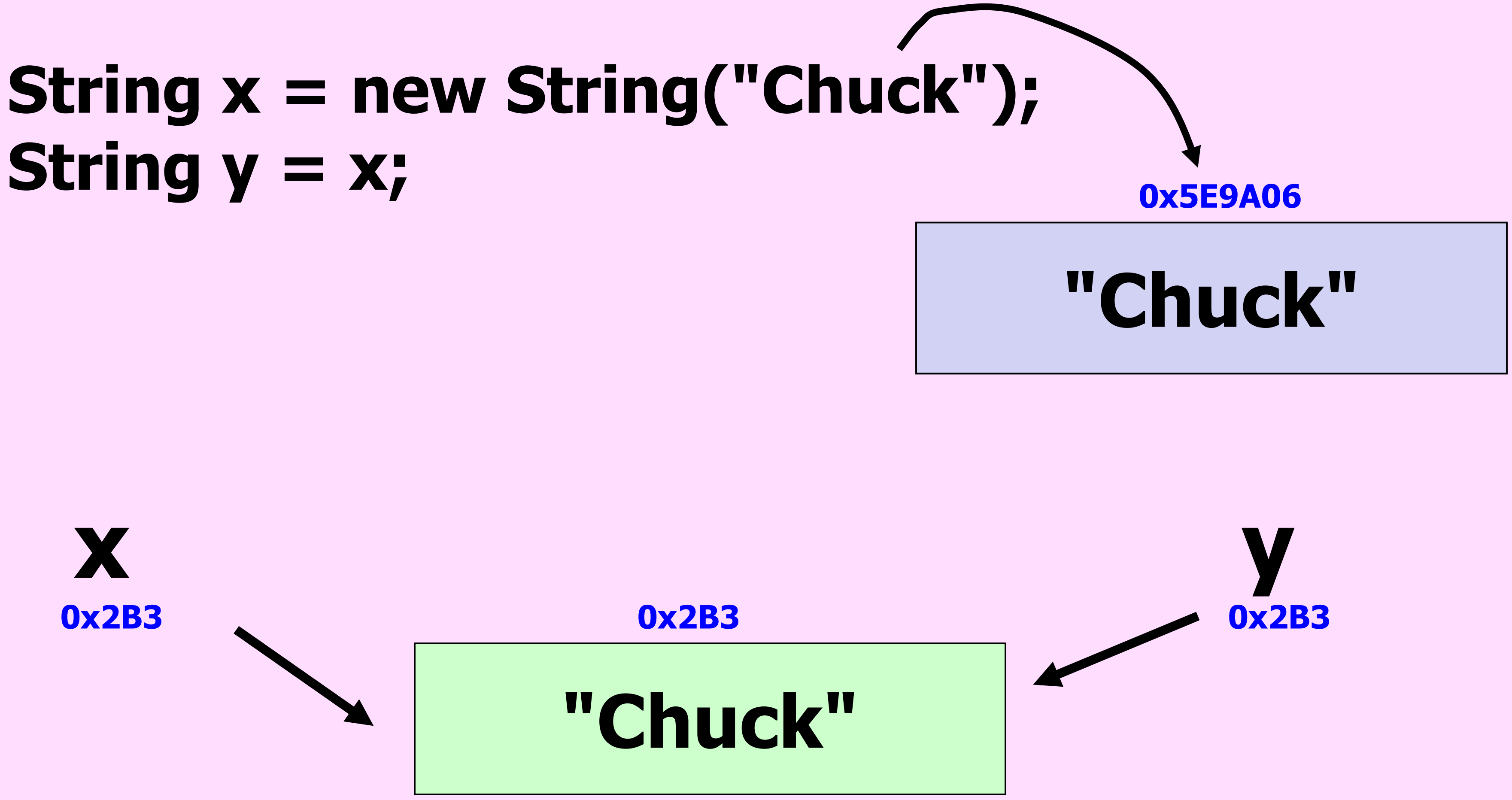
Contest Puzzlers

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;
```



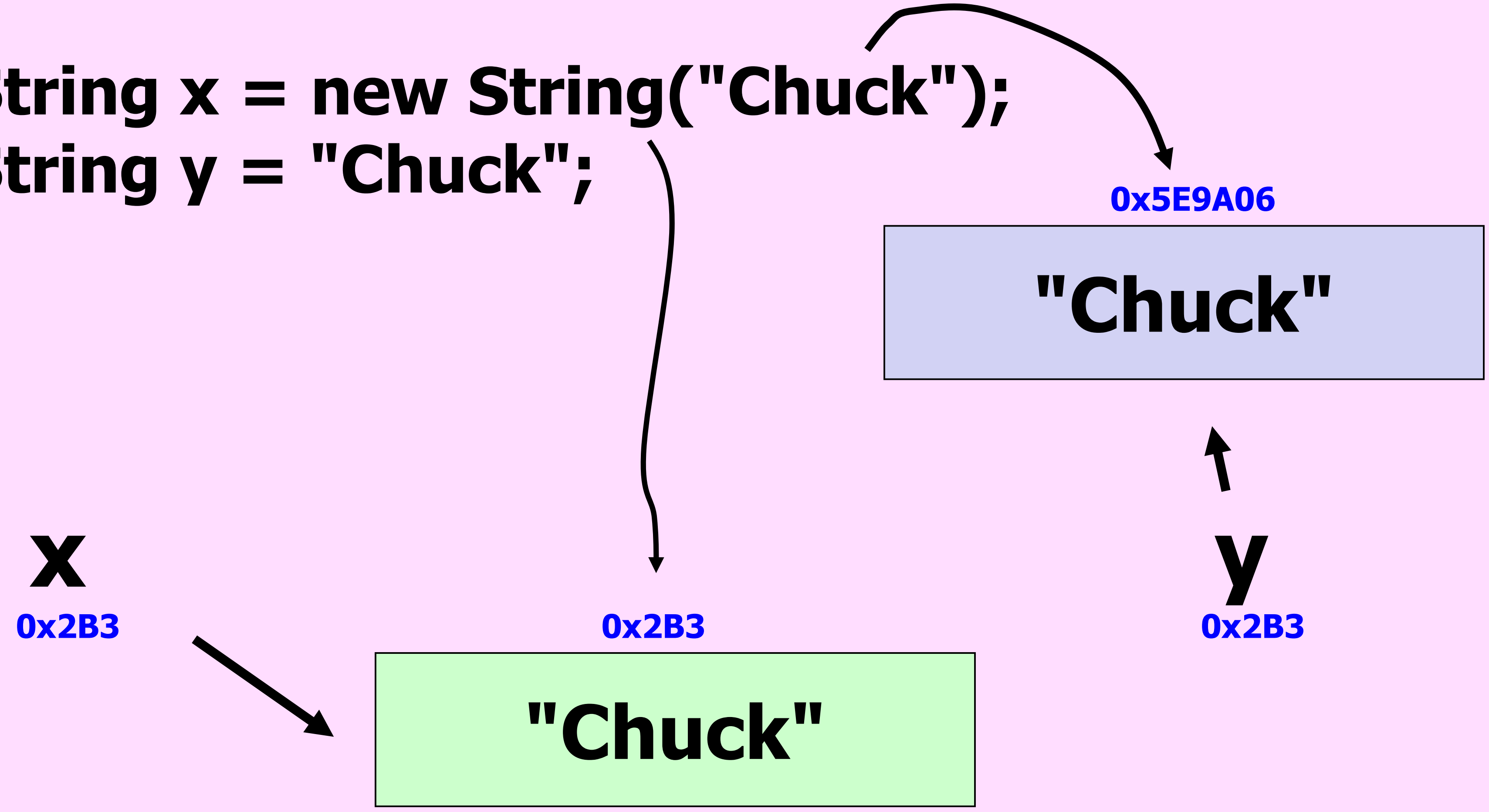
Contest Puzzlers

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";
```



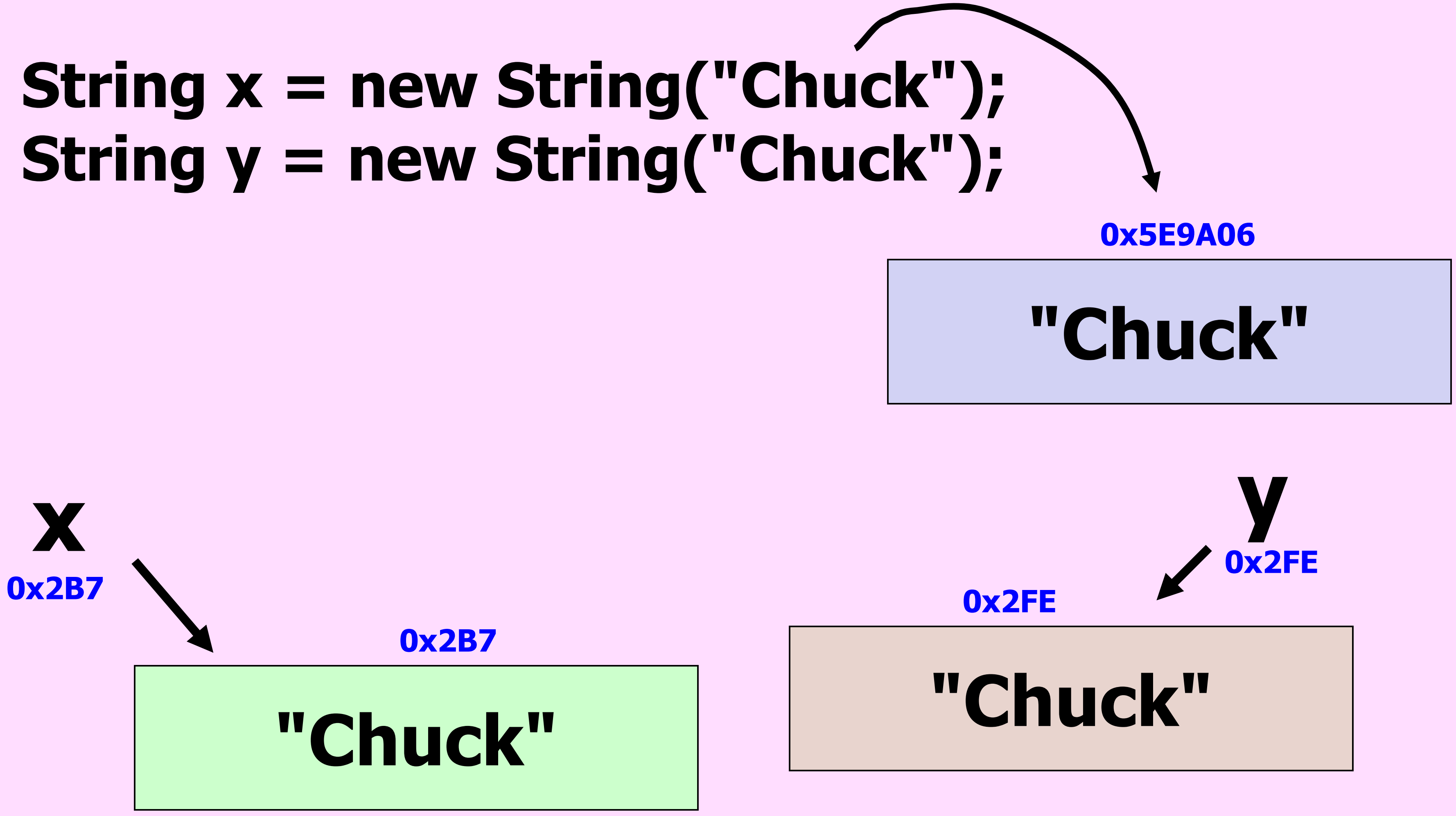
Contest Puzzlers

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");
```



references.java

Iterators

Java Iterators

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

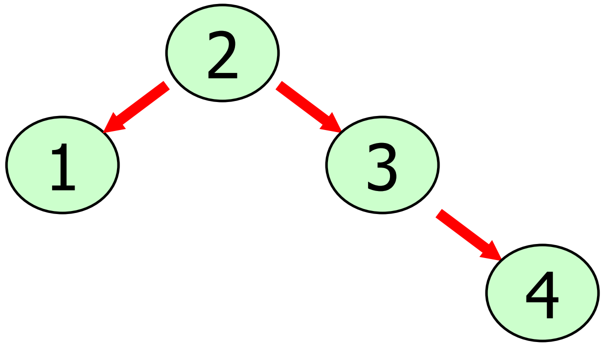
Iterators allow you to go from item to item through a collection.

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?

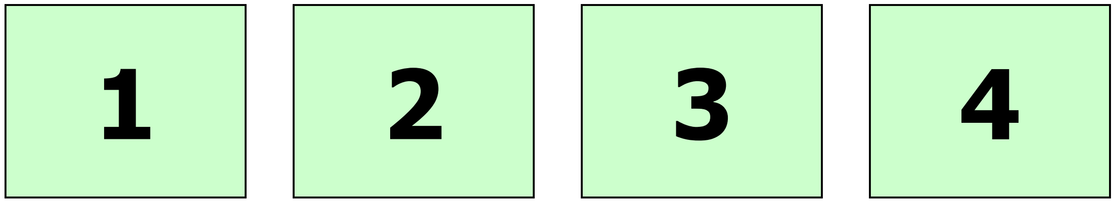
An Iterator provides a standard way to access all of the references stored in a collection.

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.



Iterator Interface

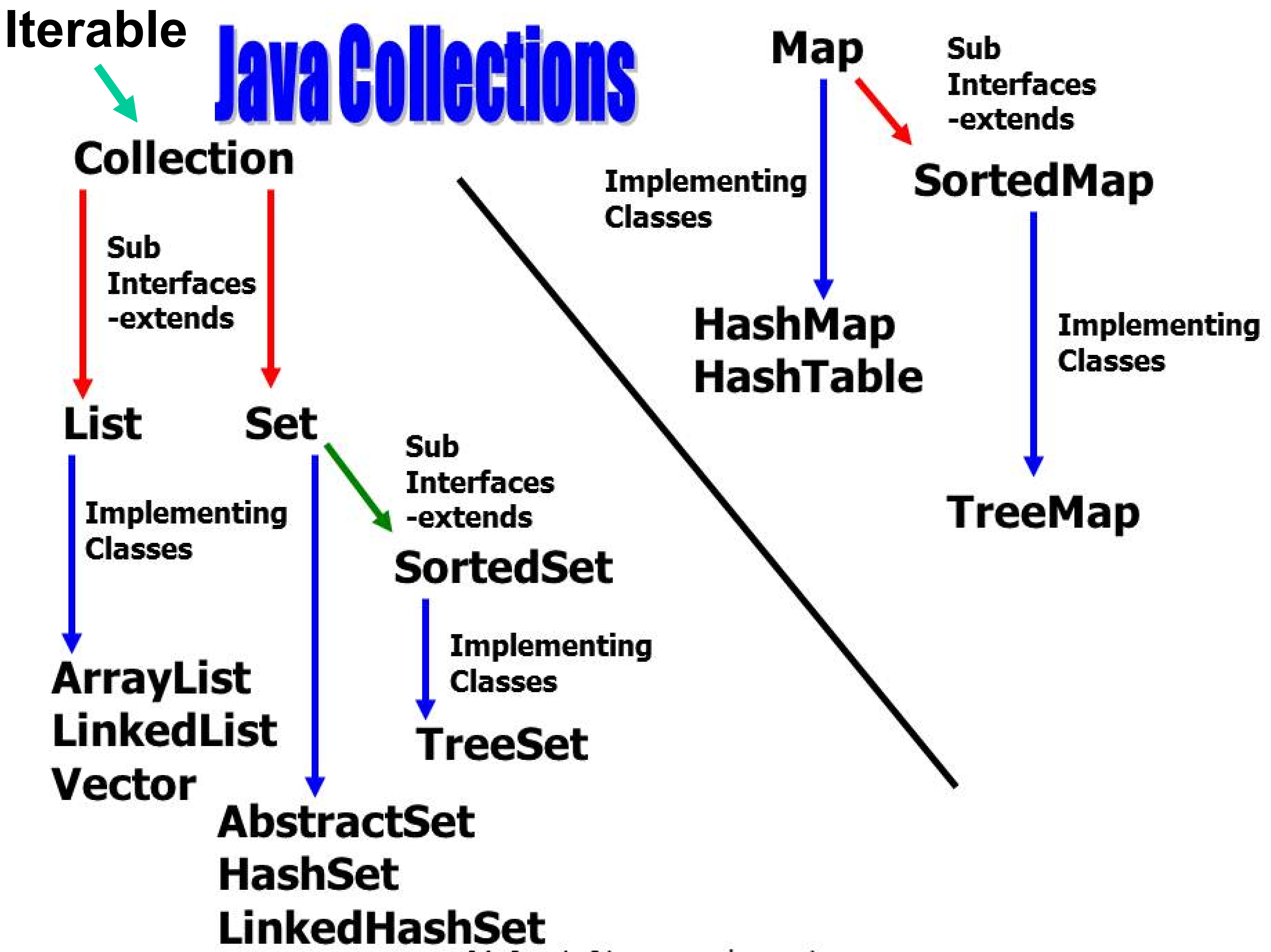
next()

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

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

OUTPUT

```
at
```

Java Collections



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;
```

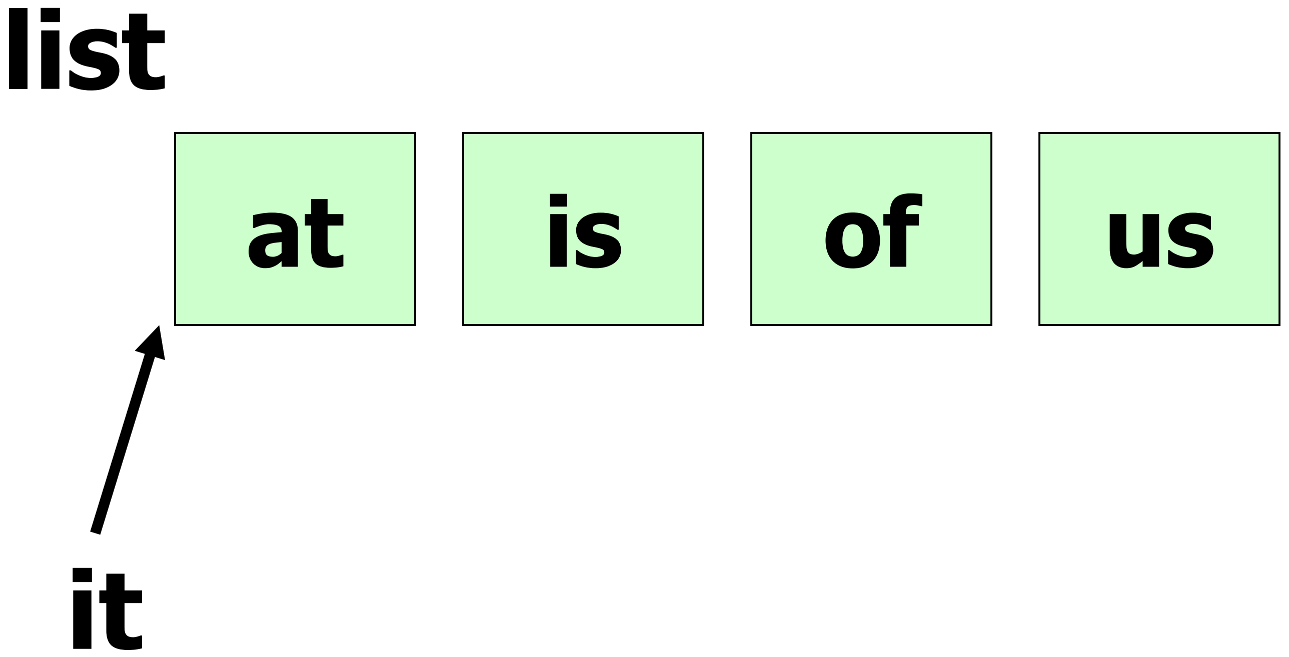
next()

```
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()

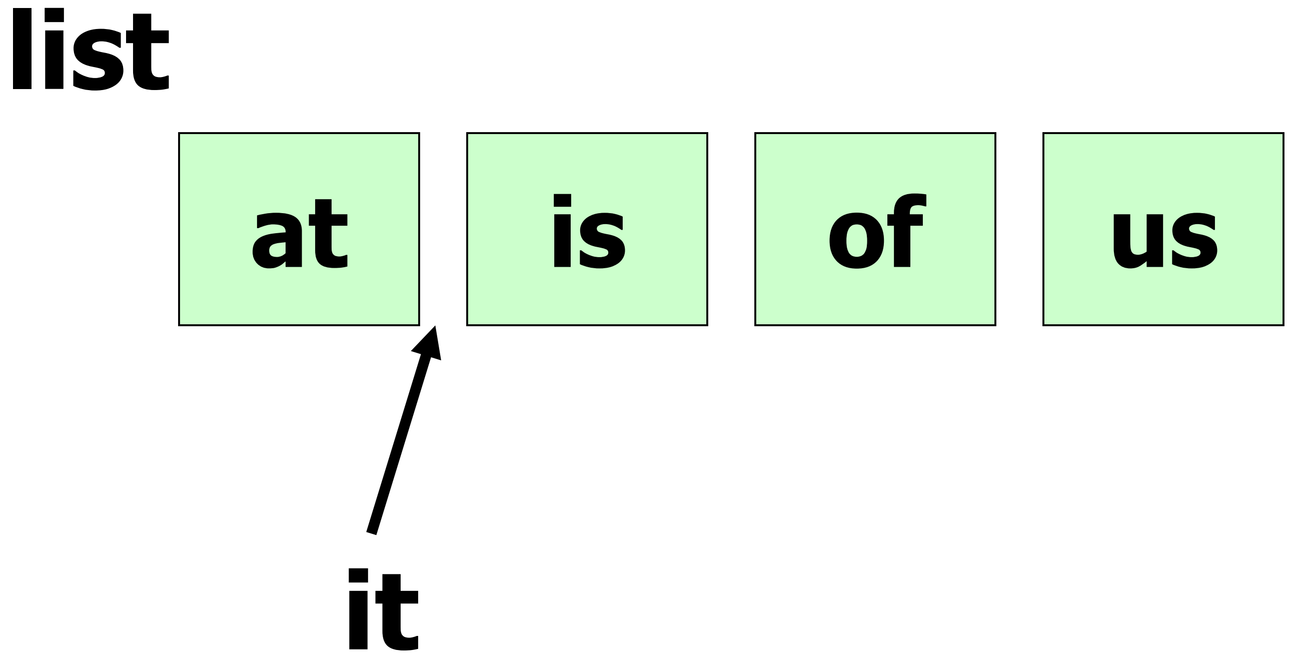


```
Iterator it = list.iterator();
```

next()

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

next()



it.next();

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

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
```

iteratorone.java

hasNext()

```
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

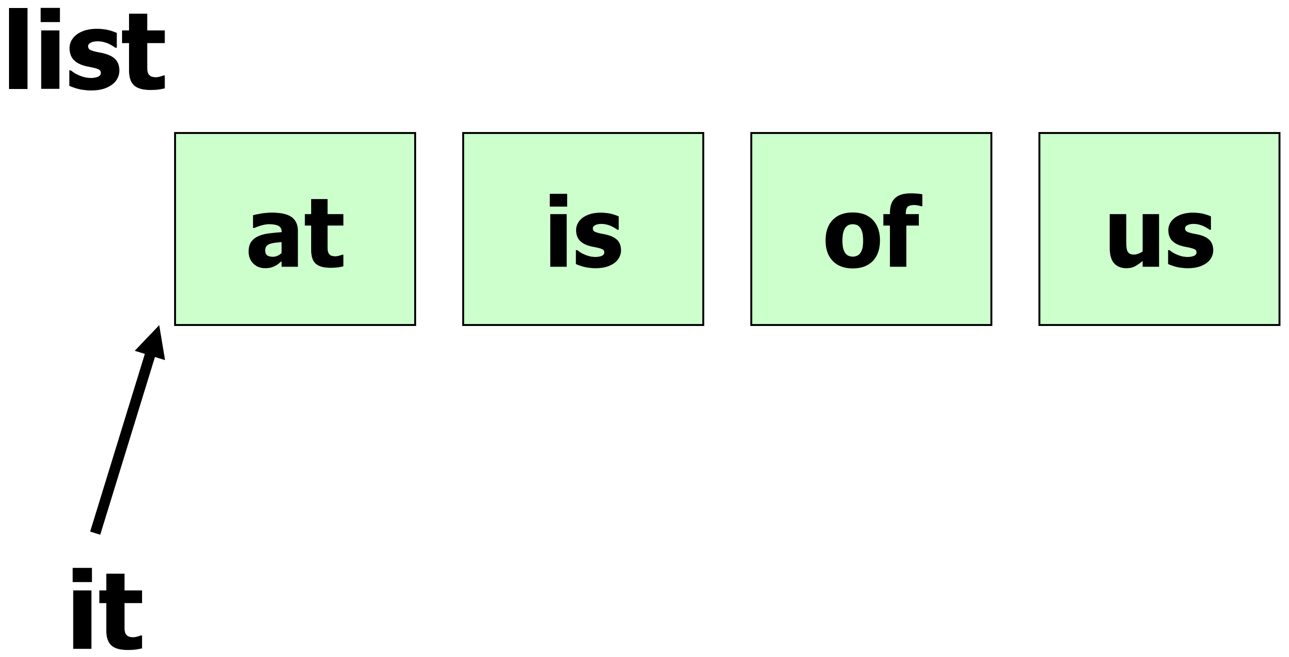
remove()

```
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

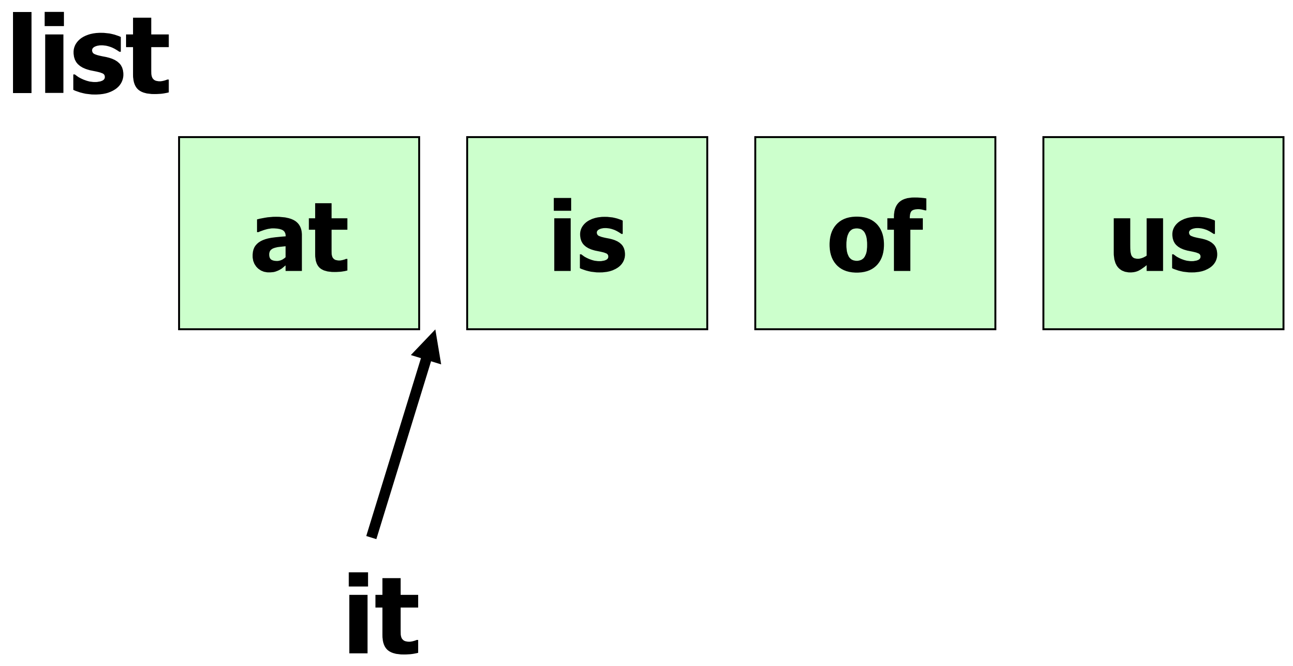
```
at
is
[is, of]
```

remove()



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

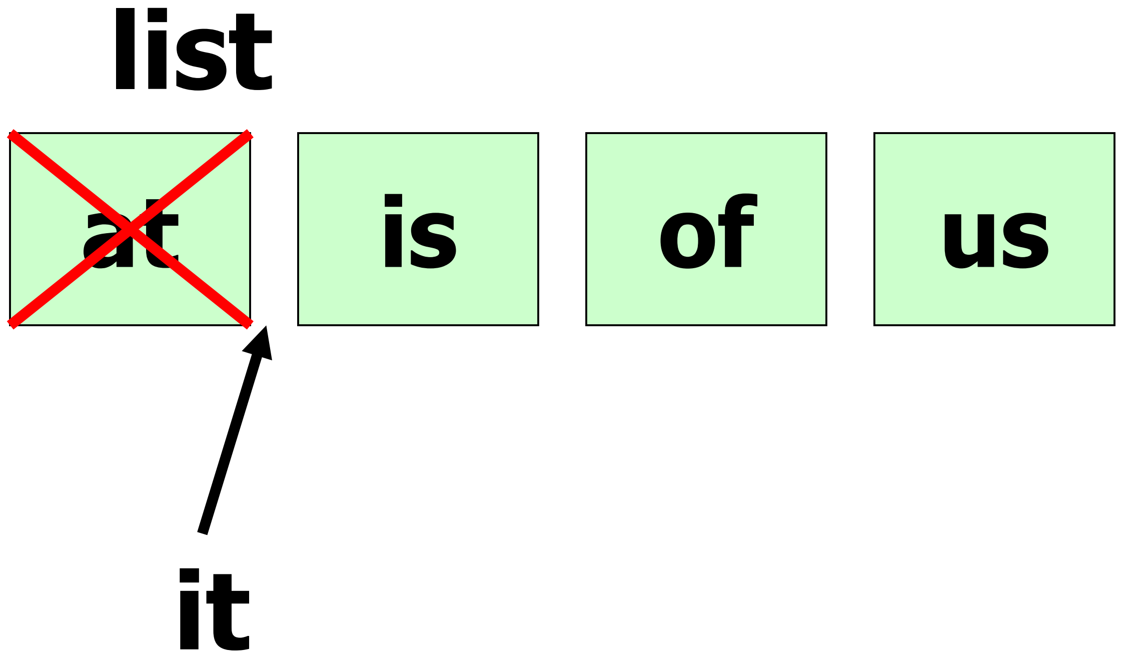
remove()



it.next();

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

remove()



it.remove();

remove always modifies the last reference returned by next.

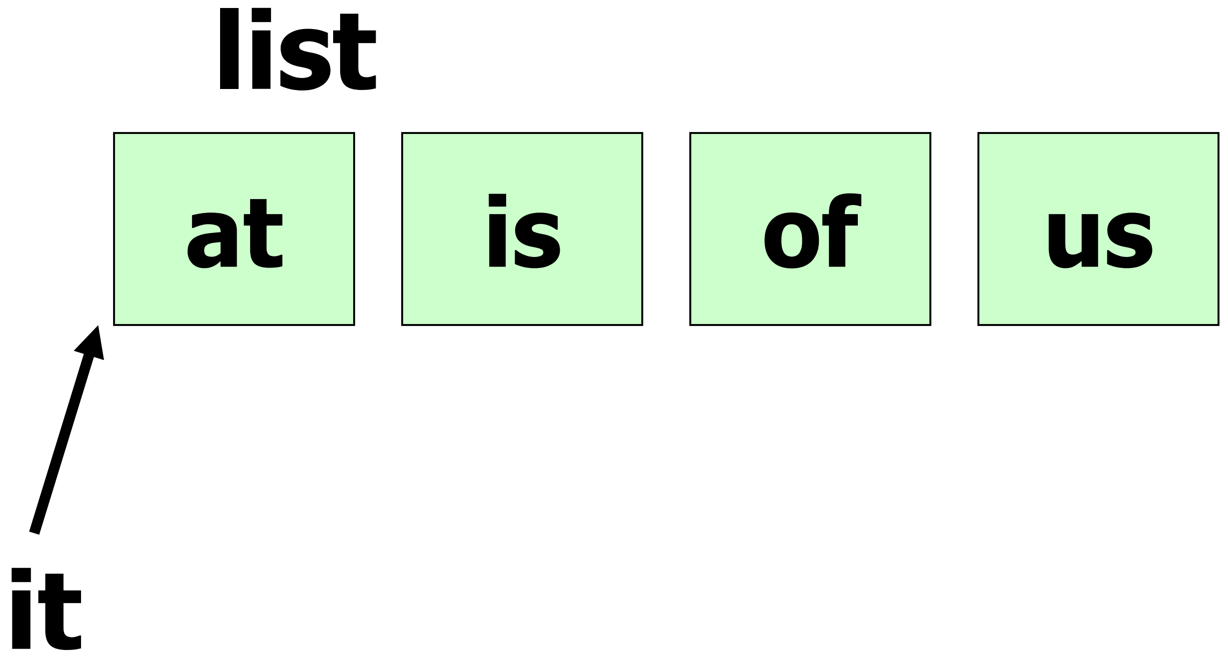
remove()

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

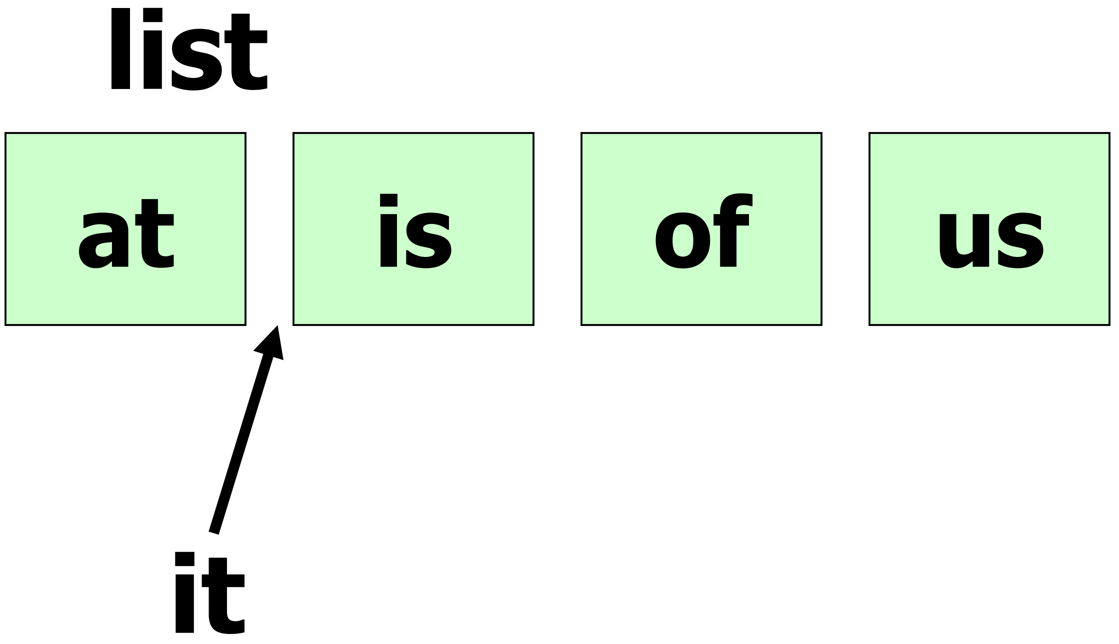
OUTPUT

```
at
error
```

remove()



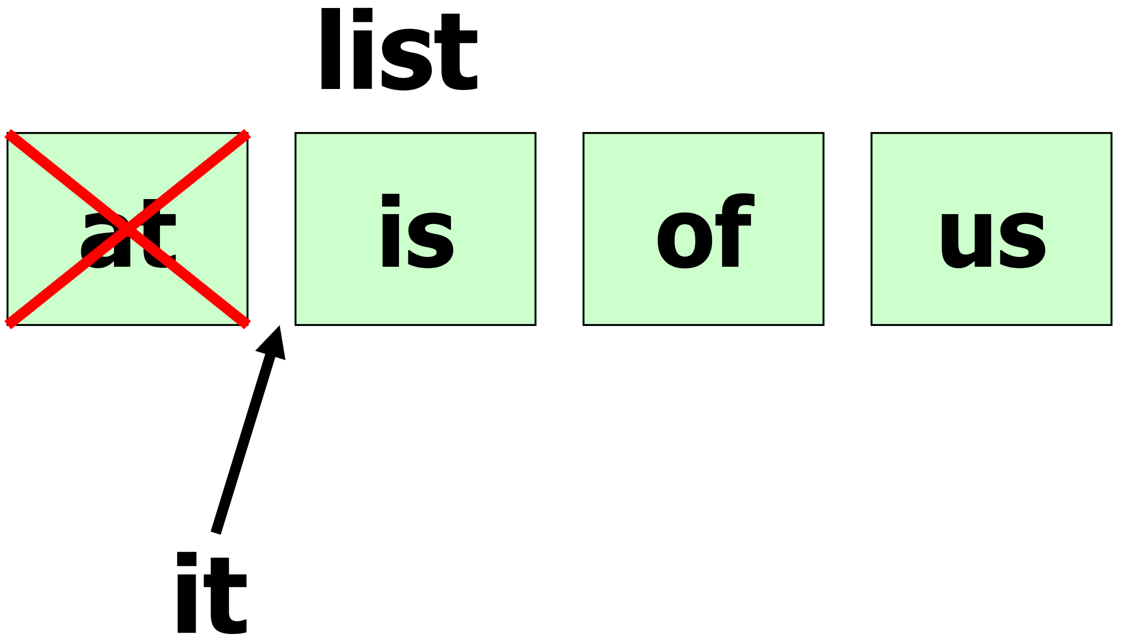
remove()



it.next();

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

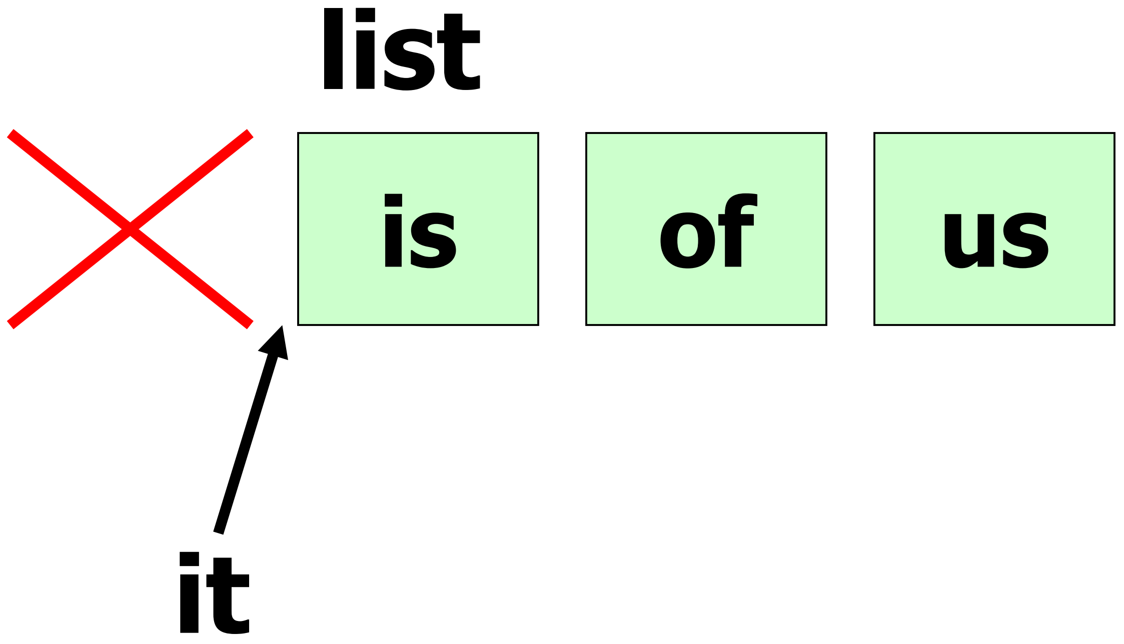
remove()



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.

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
```

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);
```

OUTPUT

```
at
```

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);
it = words.iterator();
it.next();
it.next();
```

OUTPUT

```
at
```

What is the output of the code segment?

a. 1 b. 12 c. 13 d. error

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

What is the output of the code segment?

a. 1 b. 12 c. 13 d. error

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

removeone.java

removetwo.java

String □ Array □ ArrayList

```
String[] words = "abc cde fgh".split(" ");
ArrayList<String> list;
list = new ArrayList<String>(Arrays.asList(words));
```

arraylistsplit.java

Start work on Lab 5