Iterators

Computer Science 112
Boston University

Christine Papadakis-Kanaris

Example:

Counting the Number of Occurrences of an Item

```
public class MyClass {
    public static int numOccur(List l, Object item) {
        int numOccur = 0;
        for (int i = 0; i < l.length(); i++) {
            Object itemAt = l.getItem(i);
            if (itemAt.equals(item)) {
                numOccur++;
            }
        }
        return numOccur;
    } ...
```

Example:

Counting the Number of Occurrences of an Item

```
public class MyClass {
    public static int numOccur(List l, Object item) {
        int numOccur = 0;
        for (int i = 0; i < l.length(); i++) {
            Object itemAt = l.getItem(i);
            if (itemAt.equals(item)) {
                numOccur++;
            }
        }
        return numOccur;
    } ...
```

Loop through the length of the list and compare each item returned by the method `getItem` with the item that was passed to the method.

Example:

Counting the Number of Occurrences of an Item

```
public class MyClass {
    public static int numOccur(List l, Object item) {
        int numOccur = 0;
        for (int i = 0; i < l.length(); i++) {
            Object itemAt = l.getItem(i);
            if (itemAt.equals(item)) {
                numOccur++;
            }
        }
        return numOccur;
    } ...
```

If we wanted to express this algorithm using Big-O notation, what operations do we want to measure?

Example:

Counting the Number of Occurrences of an Item

```
public class MyClass {
    public static int numOccur(List l, Object item) {
        int numOccur = 0;
        for (int i = 0; i < l.length(); i++) {
            Object itemAt = l.getItem(i);
            if (itemAt.equals(item)) {
                numOccur++;
            }
        }
        return numOccur;
    } ...
```

What does the method `getItem` do each time it is invoked?

Example:

Counting the Number of Occurrences of an Item

```
public class MyClass {
    public static int numOccur(List l, Object item) {
        int numOccur = 0;
        for (int i = 0; i < l.length(); i++) {
            Object itemAt = l.getItem(i);
            if (itemAt.equals(item)) {
                numOccur++;
            }
        }
        return numOccur;
    } ...
```

It depends on whether our list l is a reference to an instance of ArrayList or an instance of LLList…

Example:

Counting the Number of Occurrences of an Item

```
public class MyClass {
    public static int numOccur(List l, Object item) {
        int numOccur = 0;
        for (int i = 0; i < l.length(); i++) {
            Object itemAt = l.getItem(i);
            if (itemAt.equals(item)) {
                numOccur++;
            }
        }
        return numOccur;
    } ...
```

- This method works fine if we pass in an `ArrayList` object.
  - time efficiency (as a function of the length, n) = $O(n)$
- However, it's *not* efficient if we pass in an `LLList`.
  - each call to `getItem()` calls `getNode()`
  - to access item 0, `getNode()` accesses 2 nodes (dummy + node 0)
  - to access item 1, `getNode()` accesses 3 nodes
  - to access item i, `getNode()` accesses i+2 nodes
  - 2 + 3 + … + (n+1) = $O(n^2)$

A Solution:

Make `numOccur()` an `LLList` Method

```
public class LLList {
    public int numOccur(Object item) {
        int numOccur = 0;
        Node trav = head.next;  // skip the dummy head node
        while (trav != null) {
            if (trav.item.equals(item)) {
                numOccur++;
            }
            trav = trav.next;
        }
        return numOccur;
    } …
```

- Each node is only visited once, so we get *O*(n) efficiency.
- Problem: we can't anticipate all types of operations that clients may wish to perform.
- We'd like to provide the general ability to iterate over the list.

A Better Solution:

Provide an Iterator

```
public class MyClass {
    public static int numOccur(List l, Object item) {
        int numOccur = 0;
        ListIterator iter = l.iterator();
        while (iter.hasNext()) {
            Object itemAt = iter.next();
            if (itemAt.equals(item)) {
                numOccur++;
            }
        }
        return numOccur;
    } ...
```

- We add an `iterator()` method to the `List` interface.
  - it returns a separate *iterator object* that can efficiently iterate over the items in the list

A Better Solution:

Provide an Iterator

```
public class MyClass {
    public static int numOccur(List l, Object item) {
        int numOccur = 0;
        ListIterator iter = l.iterator();
        while (iter.hasNext()) {
            Object itemAt = iter.next();
            if (itemAt.equals(item)) {
                numOccur++;
            }
        }
        return numOccur;
    } ...
```

- We add an `iterator()` method to the `List` interface.
  - it returns a separate *iterator object* that can efficiently iterate over the items in the list
- The ListIterator (class) has two key methods:
  - `hasNext()`: tells us if there are items we haven't seen yet
  - `next()`: returns the next item *and* advances the iterator

A Better Solution:

Provide an Iterator

```
public class MyClass {
    public static int numOccur(List l, Object item) {
        int numOccur = 0;
        ListIterator iter = l.iterator();
        while (iter.hasNext()) {
            Object itemAt = iter.next();
            if (itemAt.equals(item)) {
                numOccur++;
            }
        }
        return numOccur;
    } ...
```

Compare the object returned by the iterator with the object passed to the method.

- We add an `iterator()` method to th
  - it returns a separate *iterator object* iterate over the items in the list
- The iterator (class) has two key methods:
  - `hasNext()`: tells us if there are items we haven't seen yet
  - `next()`: returns the next item *and* advances the iterator

A Better Solution:

Provide an Iterator

```
public class MyClass {
    public static int numOccur(List l, Object item) {
        int numOccur = 0;
        ListIterator iter = l.iterator();
        while (iter.hasNext()) {
            Object itemAt = iter.next();
            if (itemAt.equals(item)) {
                numOccur++;
            }
        }
        return numOccur;
    } ...
```

Continue until the iterator signals that there are no more items in the list!

- We add an `iterator()` method to
  - it returns a separate *iterator object* that
    iterate over the items in the list
- The iterator (class) has two key methods:
  - `hasNext()`: tells us if there are items we haven't seen yet
  - `next()`: returns the next item *and* advances the iterator

An Interface for List Iterators

- Here again, the interface only includes the method headers:

```
public interface ListIterator { // in ListIterator.java
    boolean hasNext();
    Object next();
}
```

- We can then implement this interface for each type of list:
  - `LLListIterator` for an iterator that works with `LLLists`
  - `ArrayListIterator` for an iterator for `ArrayLists`

We will implement the ListIterator interface as an inner class of the class we want to use it in.

An Inner Class for the Iterator

```
public class LLList … {
    private Node head;
    private int length;

    private class LLListIterator implements ListIterator {
        private Node nextNode;  // points to node with the next item

        public LLListIterator() {
            nextNode = head.next;  // skip over dummy head node
        }
        ...
    }

    public ListIterator iterator() {
        return new LLListIterator();
    }
    ...
```

Making this an inner class gives this class access to all the private data members of the LLList class!

An Inner Class for the Iterator

```
public class LLList … {
    private Node head;
    private int length;

    private class LLListIterator implements ListIterator {
        private Node nextNode;  // points to node with the next item

        public LLListIterator() {
            nextNode = head.next;  // skip over dummy head node
        }
        ...
    }

    public ListIterator iterator() {
        return new LLListIterator();
    }
    ...
```

Creates and returns a reference to a new instance of *LLListIterator* which contains…

An Inner Class for the Iterator

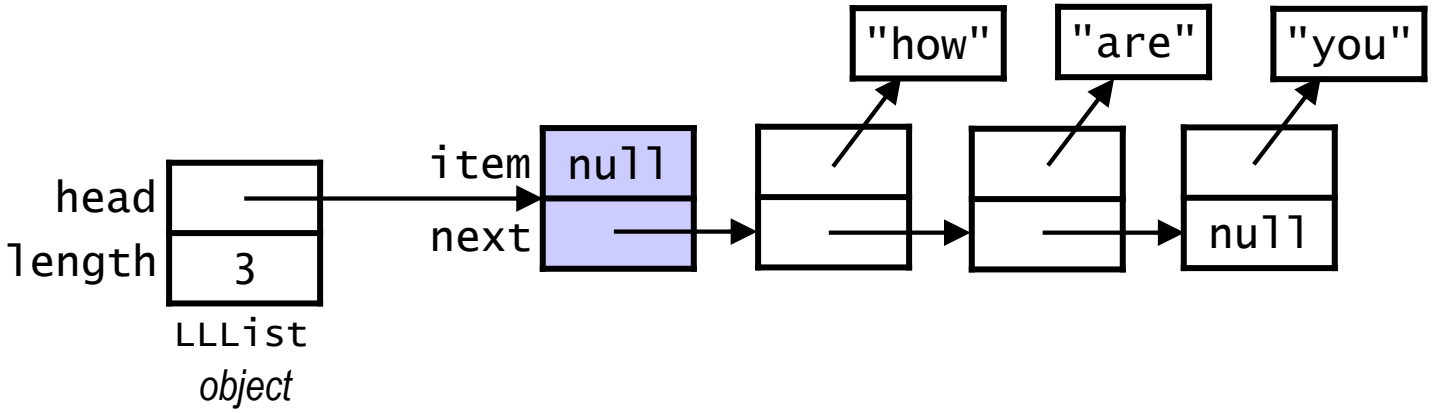
```
public class LLList … {
    private Node head;
    private int length;

    private class LLListIterator implements ListIterator {
        private Node nextNode;  // points to node with the next item

        public LLListIterator() {
            nextNode = head.next;  // skip over dummy head node
        }
        ...
    }

    public ListIterator iterator() {
        return new LLListIterator();
    }
    ...
```

a reference to the first node in the list!

Full `LLListIterator` Implementation:

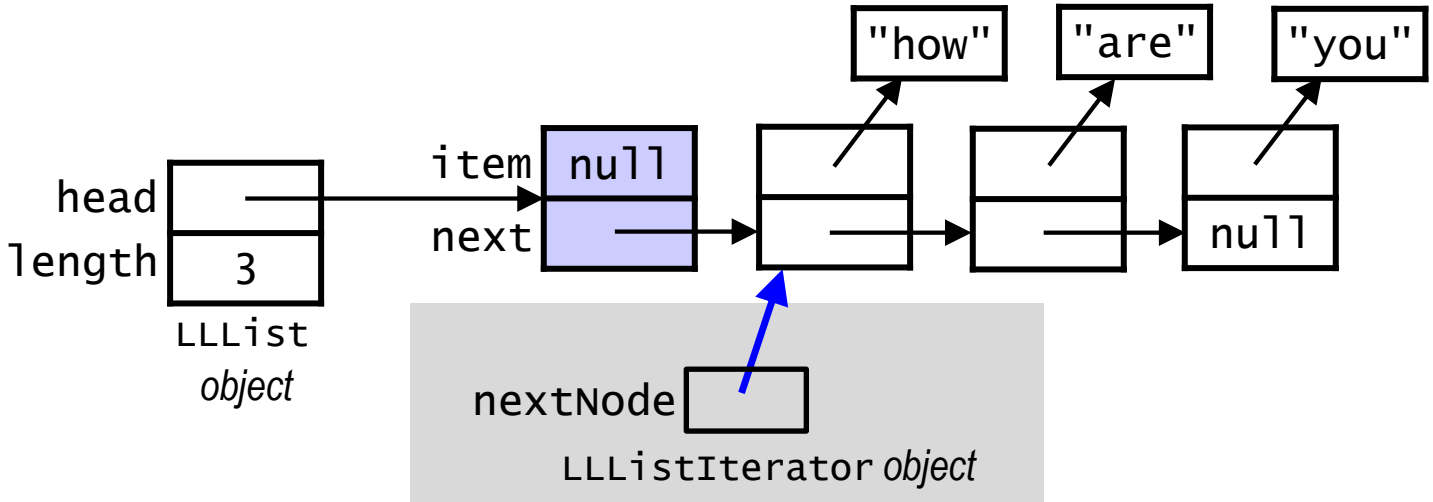
the inner class

```
private class LLListIterator implements ListIterator {
    private Node nextNode;     // points to node with the next item

    public LLListIterator() {
        nextNode = head.next;  // skip over the dummy head node
    }

    public boolean hasNext() {
        return (nextNode != null);
    }

    public Object next() {
        // throw an exception if nextNode is null

        Object item = ________________;
        nextNode = ________________;

        return item;
    }
}
```



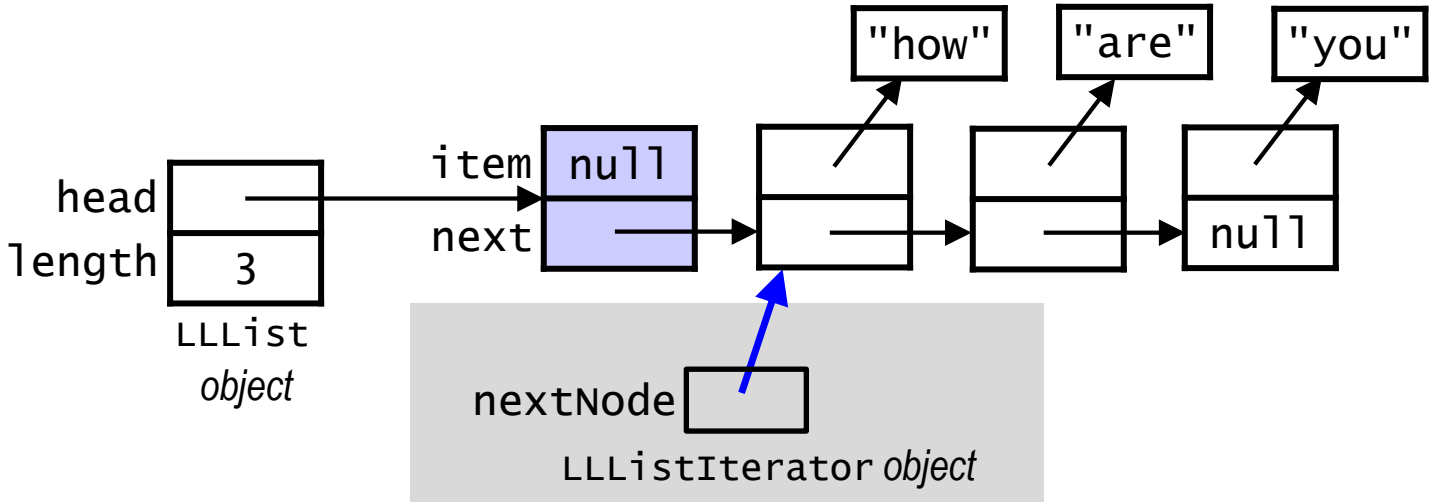
Full `LLListIterator` Implementation

```
private class LLListIterator implements ListIterator {
    private Node nextNode;     // points to node with the next item

    public LLListIterator() {
        nextNode = head.next;  // skip over the dummy head node
    }

    public boolean hasNext() {
        return (nextNode != null);
    }

    public Object next() {
        // throw an exception if nextNode is null

        Object item = ______________;
        nextNode = ______________;

        return item;
    }
}
```



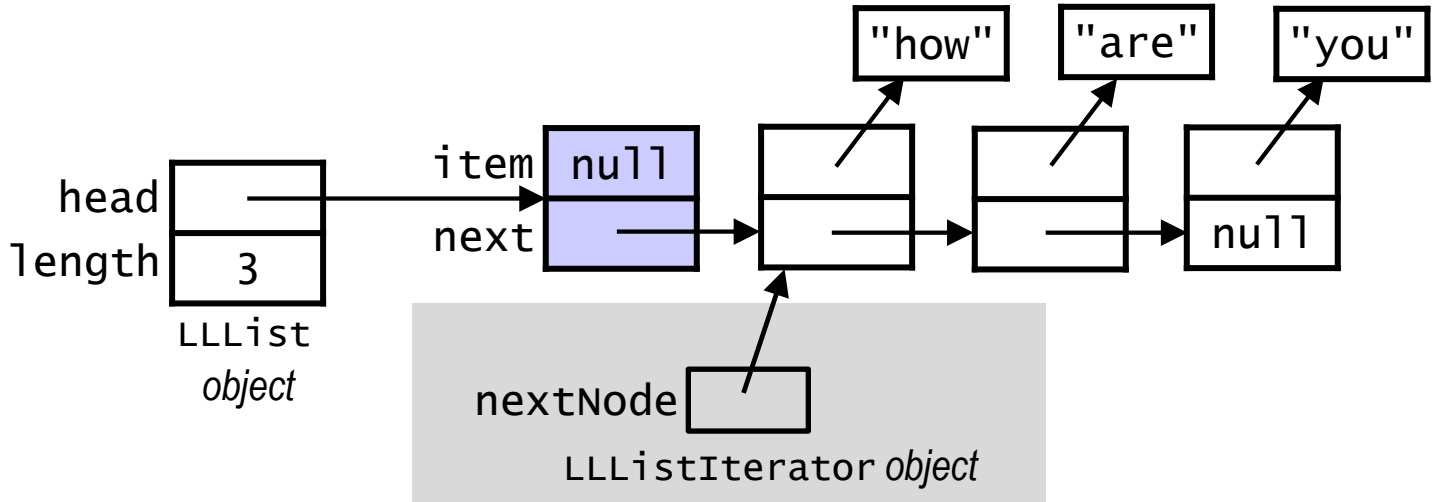
Full `LLListIterator` Implementation

```
private class LLListIterator implements ListIterator {
    private Node nextNode;     // points to node with the next item

    public LLListIterator() {
        nextNode = head.next;  // skip over the dummy head node
    }

    public boolean hasNext() {
        return (nextNode != null);
    }

    public Object next() {
        // throw an exception if nextNode is null

        Object item = ________________;
        nextNode = ________________;

        return item;
    }
}
```



Full `LLListIterator` Implementation

```
private class LLListIterator implements ListIterator {
    private Node nextNode;     // points to node with the next item

    public LLListIterator() {
        nextNode = head.next;  // skip over the dummy head node
    }

    public boolean hasNext() {
        return (nextNode != null);
    }

    public Object next() {
        // throw an exception if nextNode is null

        Object item = ______________;
        nextNode = ______________;

        return item;
    }
}
```



Full `LLListIterator` Implementation

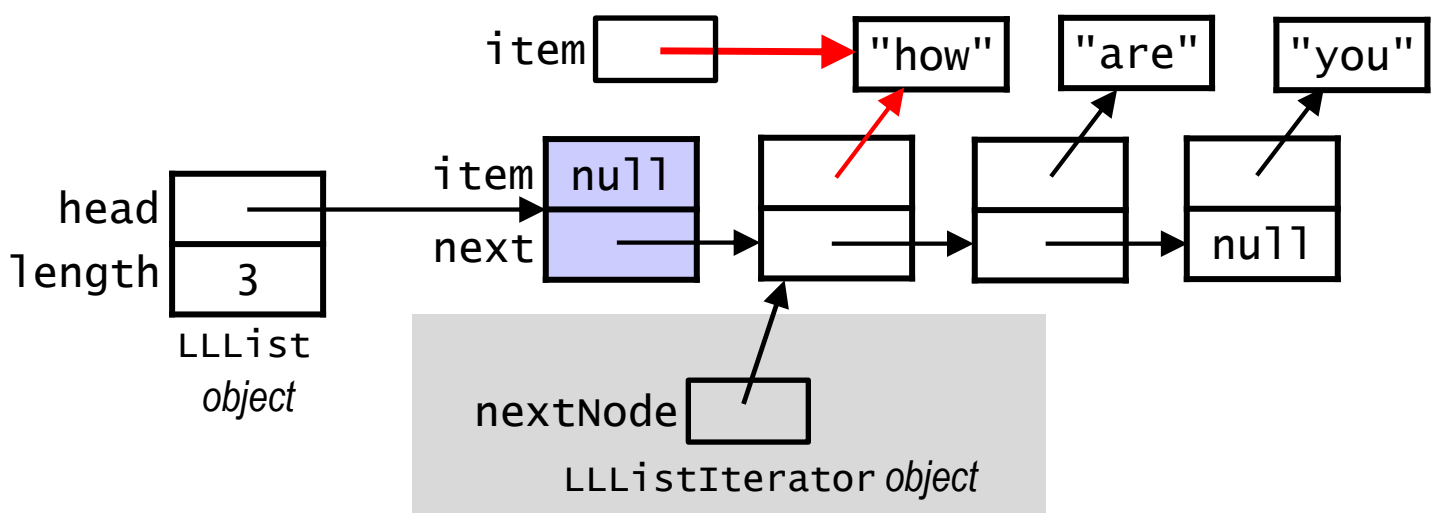
```
private class LLListIterator implements ListIterator {
    private Node nextNode;     // points to node with the next item

    public LLListIterator() {
        nextNode = head.next;  // skip over the dummy head node
    }

    public boolean hasNext() {
        return (nextNode != null);
    }

    public Object next() {
        // throw an exception if nextNode

        Object item = nextNode.item;
        nextNode = ______________;

        return item;
    }
}
```

A. `nextNode.next;`

B. **`nextNode.item;`**

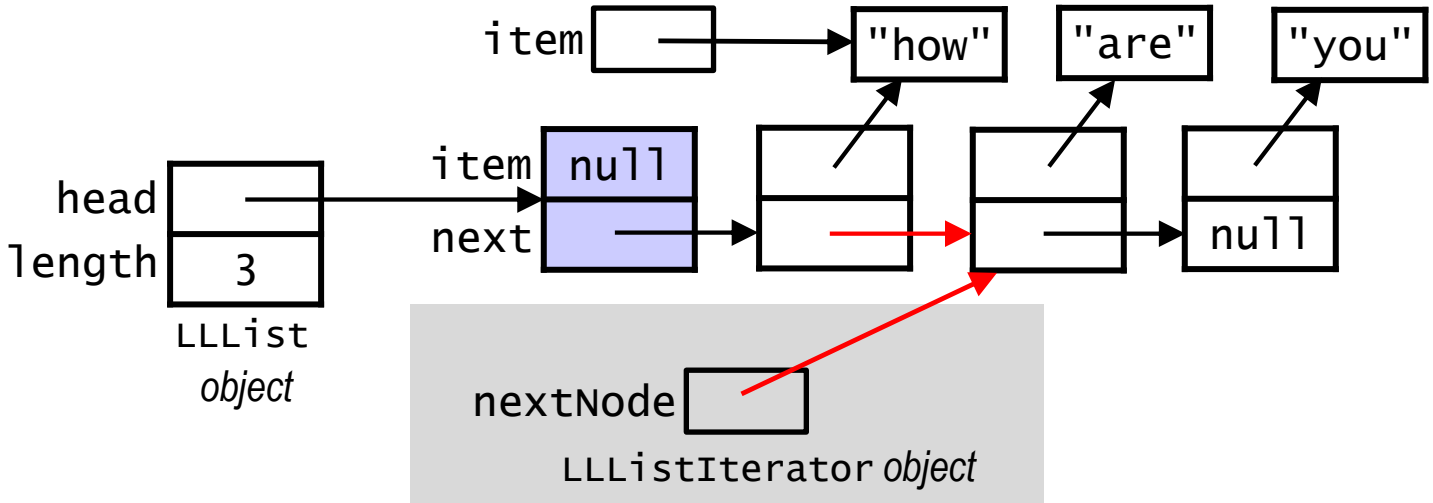
C. `nextNode.next.next;`

D. `nextNode.next.item;`



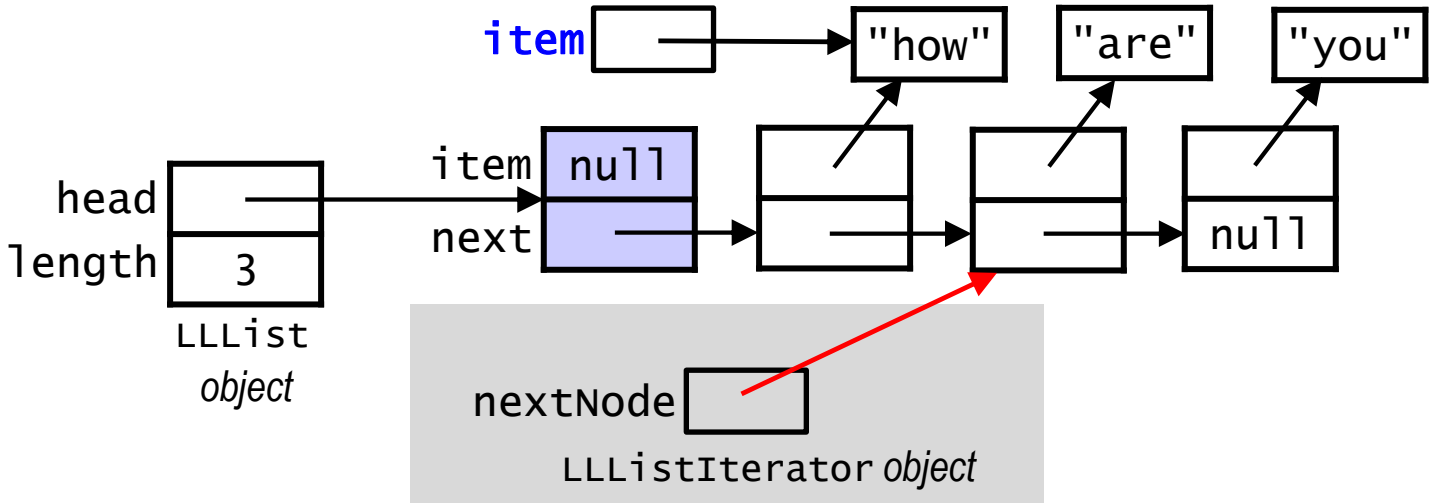
Full `LLListIterator` Implementation

```
private class LLListIterator implements ListIterator {
    private Node nextNode;     // points to node with the next item

    public LLListIterator() {
        nextNode = head.next;  // skip over the dummy head node
    }

    public boolean hasNext() {
        return (nextNode != null);
    }

    public Object next() {
        // throw an exception if nextNode is null

        Object item = nextNode.item;
        nextNode = nextNode.next;

        return item;
    }
}
```



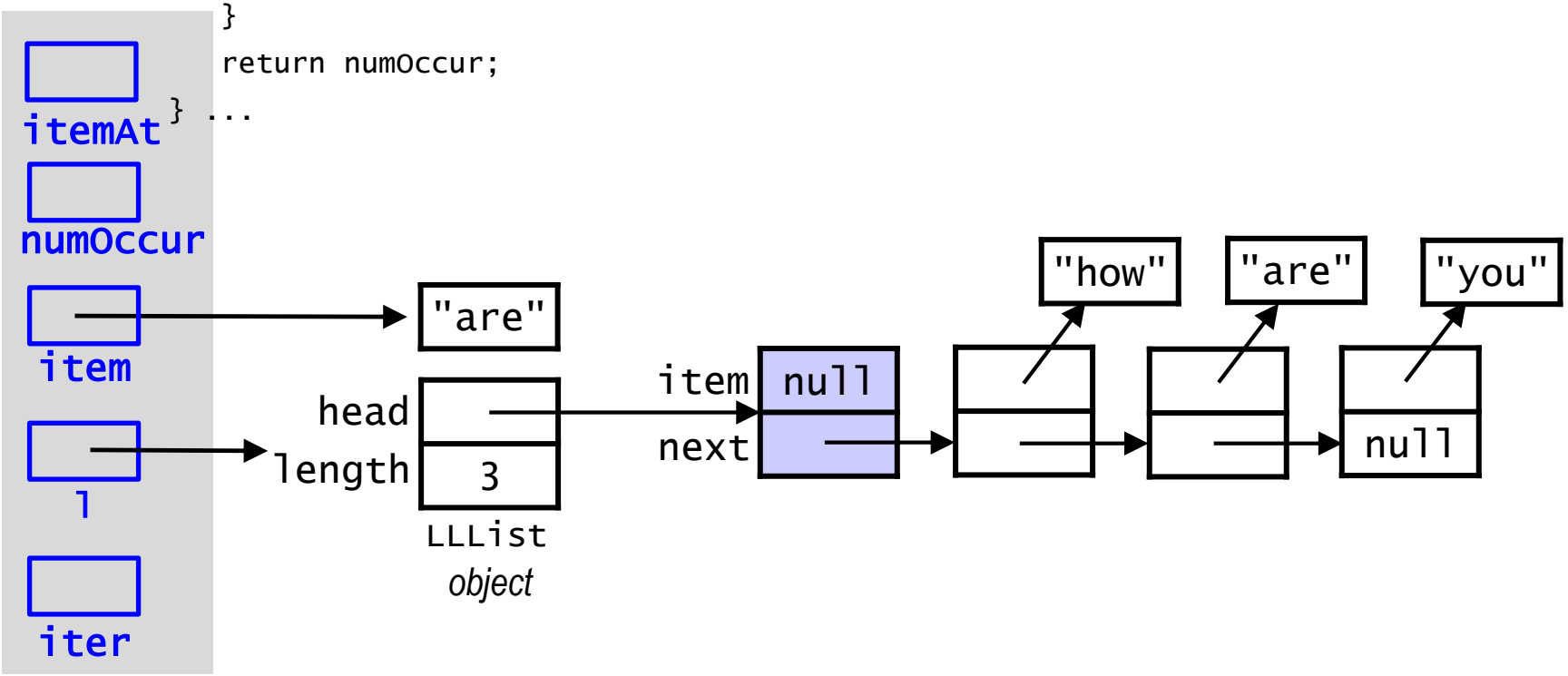
Full `LLListIterator` Implementation

```
private class LLListIterator implements ListIterator {
    private Node nextNode;     // points to node with the next item

    public LLListIterator() {
        nextNode = head.next;  // skip over the dummy head node
    }

    public boolean hasNext() {
        return (nextNode != null);
    }

    public Object next() {
        // throw an exception if nextNode is null

        Object item = nextNode.item;
        nextNode = nextNode.next;

        return item;
    }
}
```



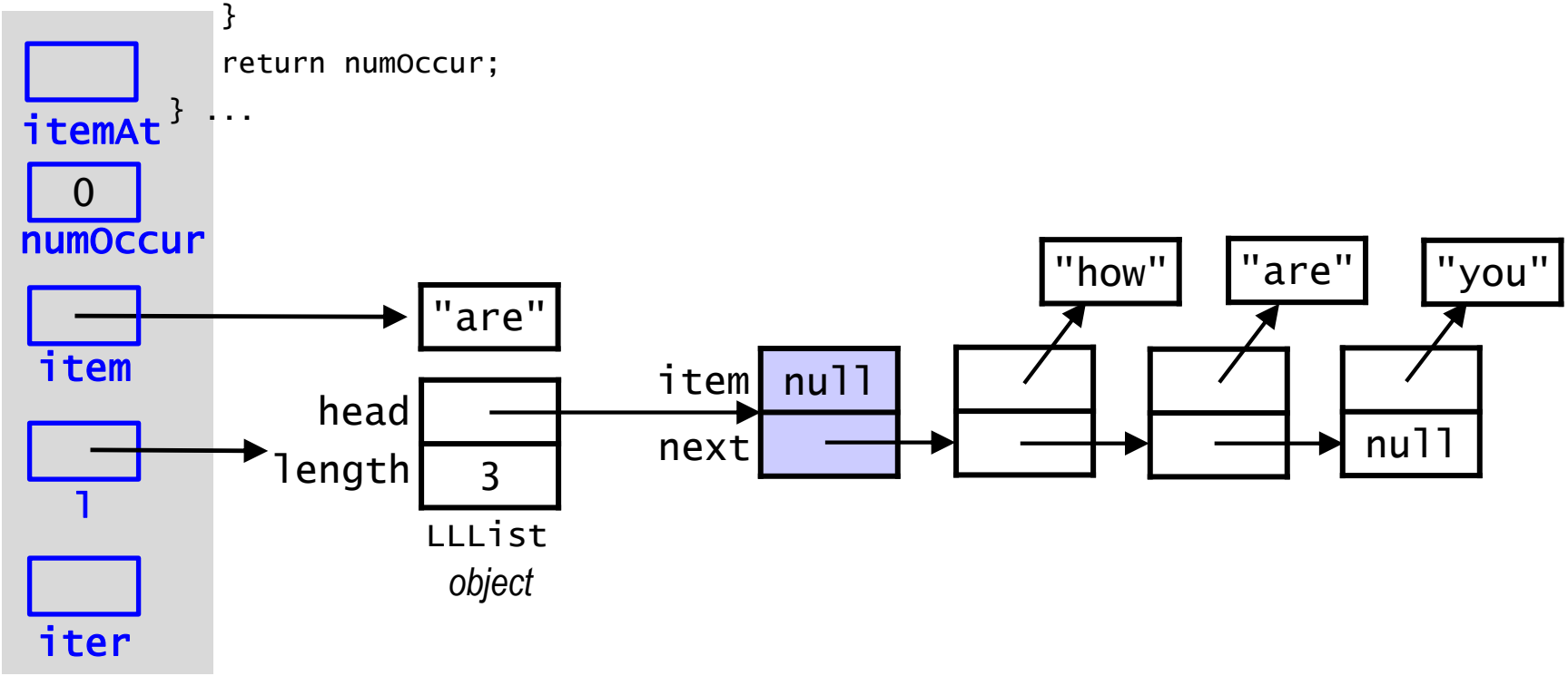
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



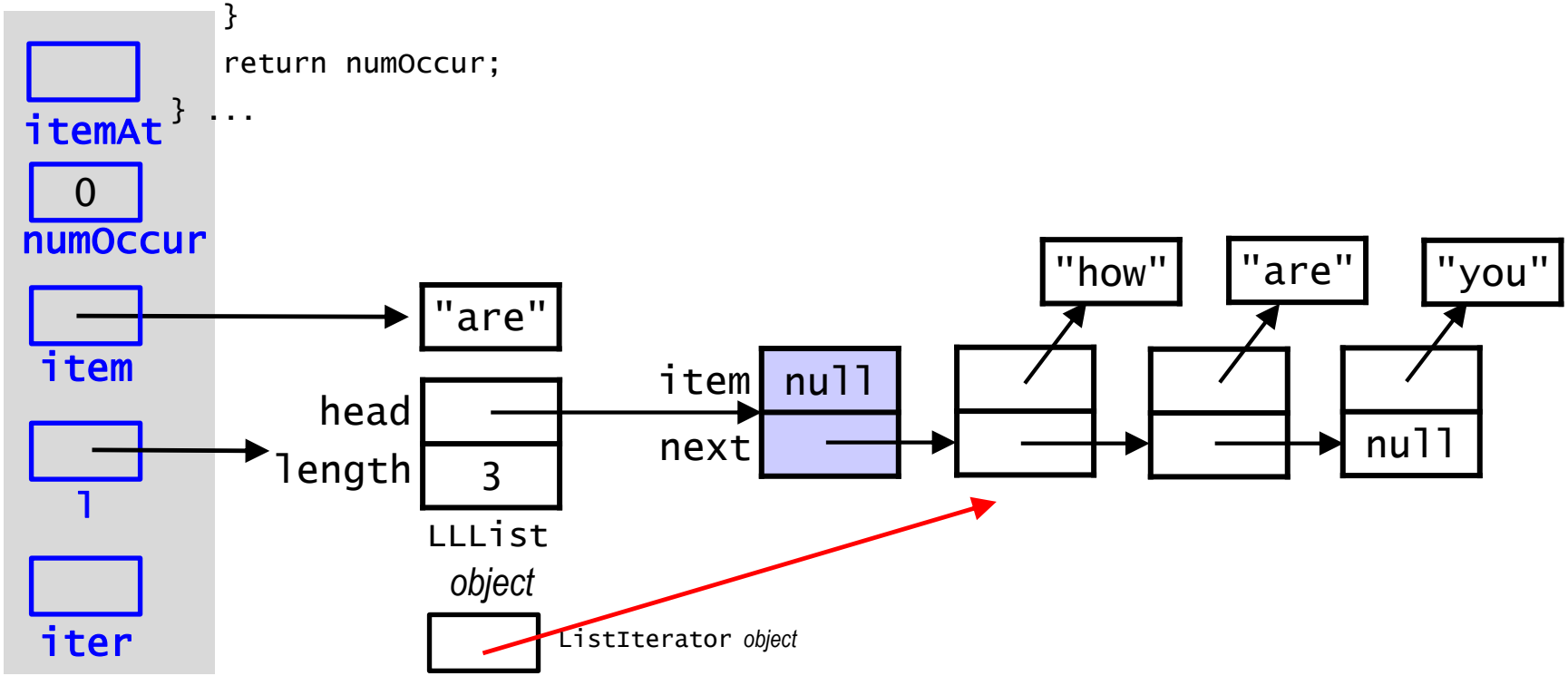
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



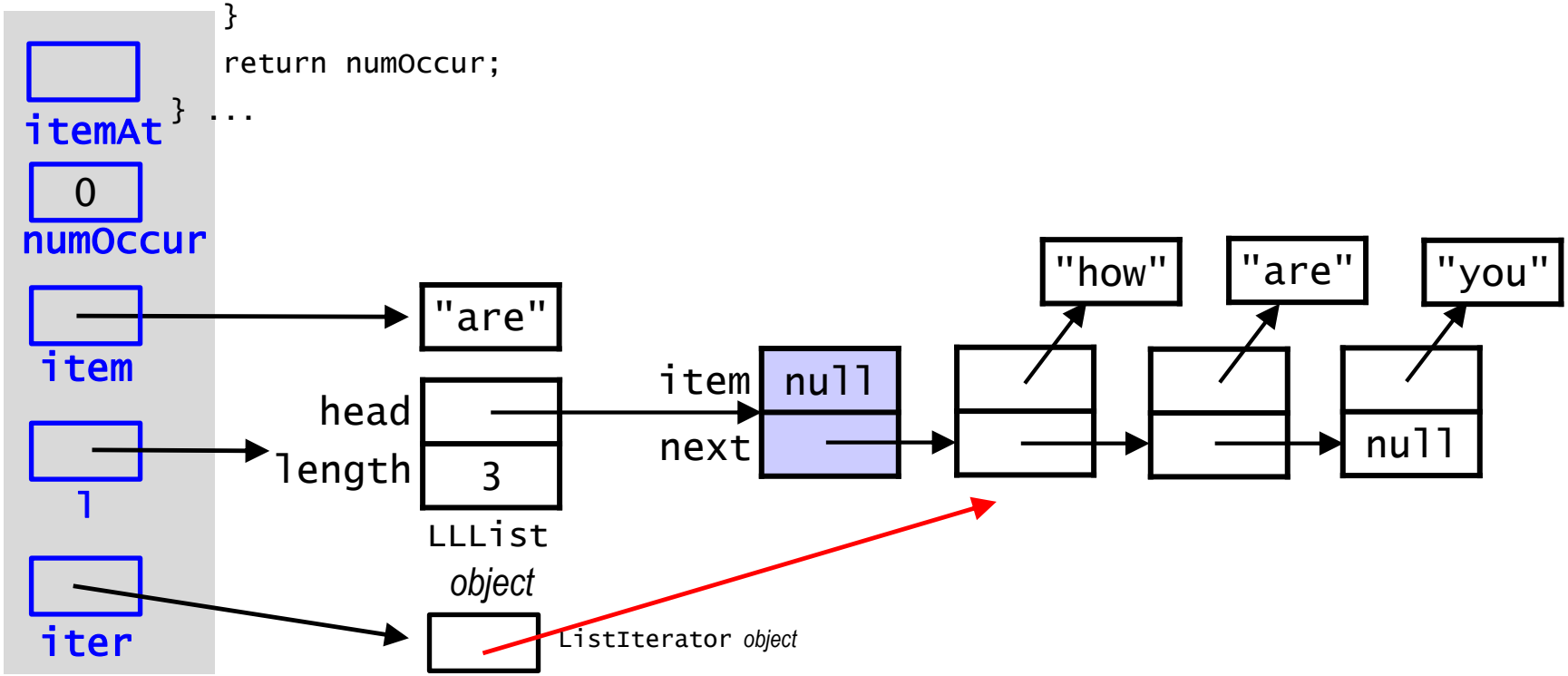
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



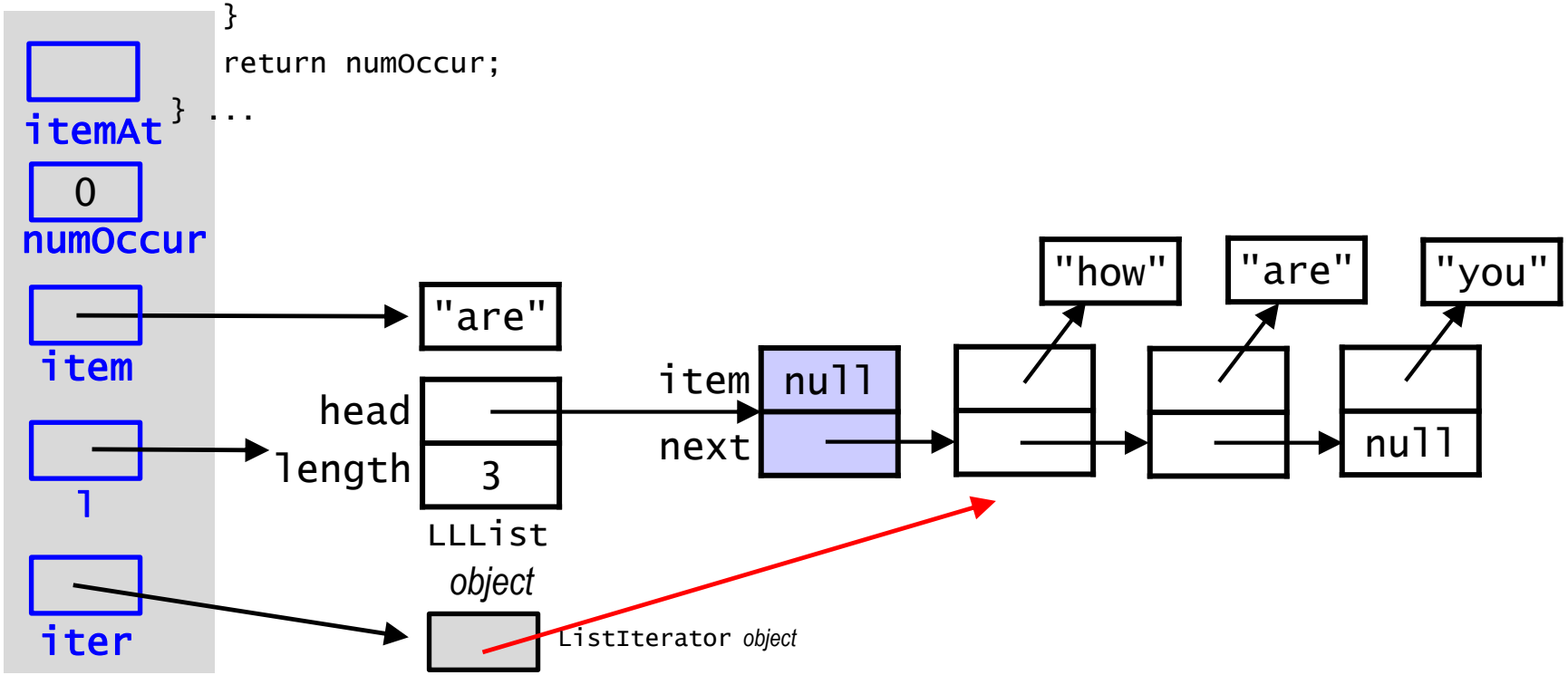
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



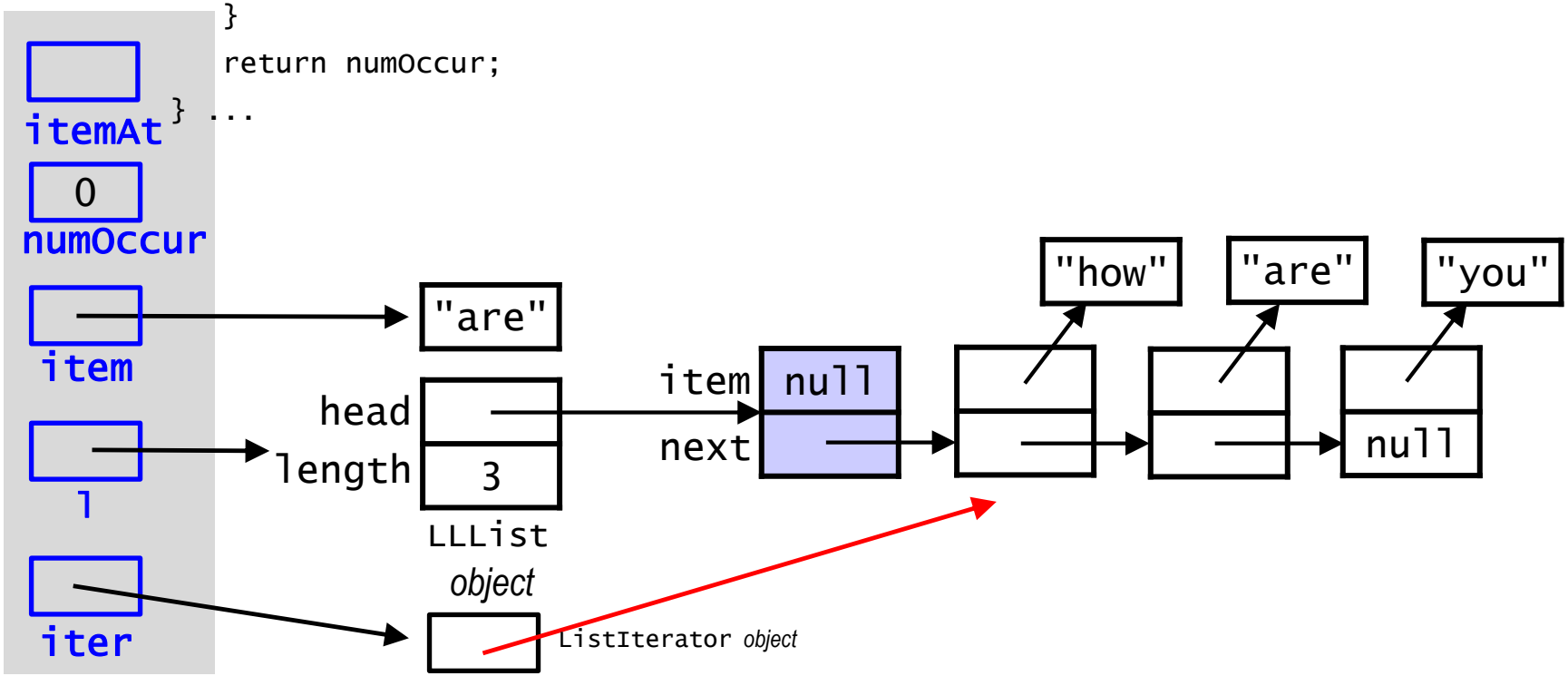
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



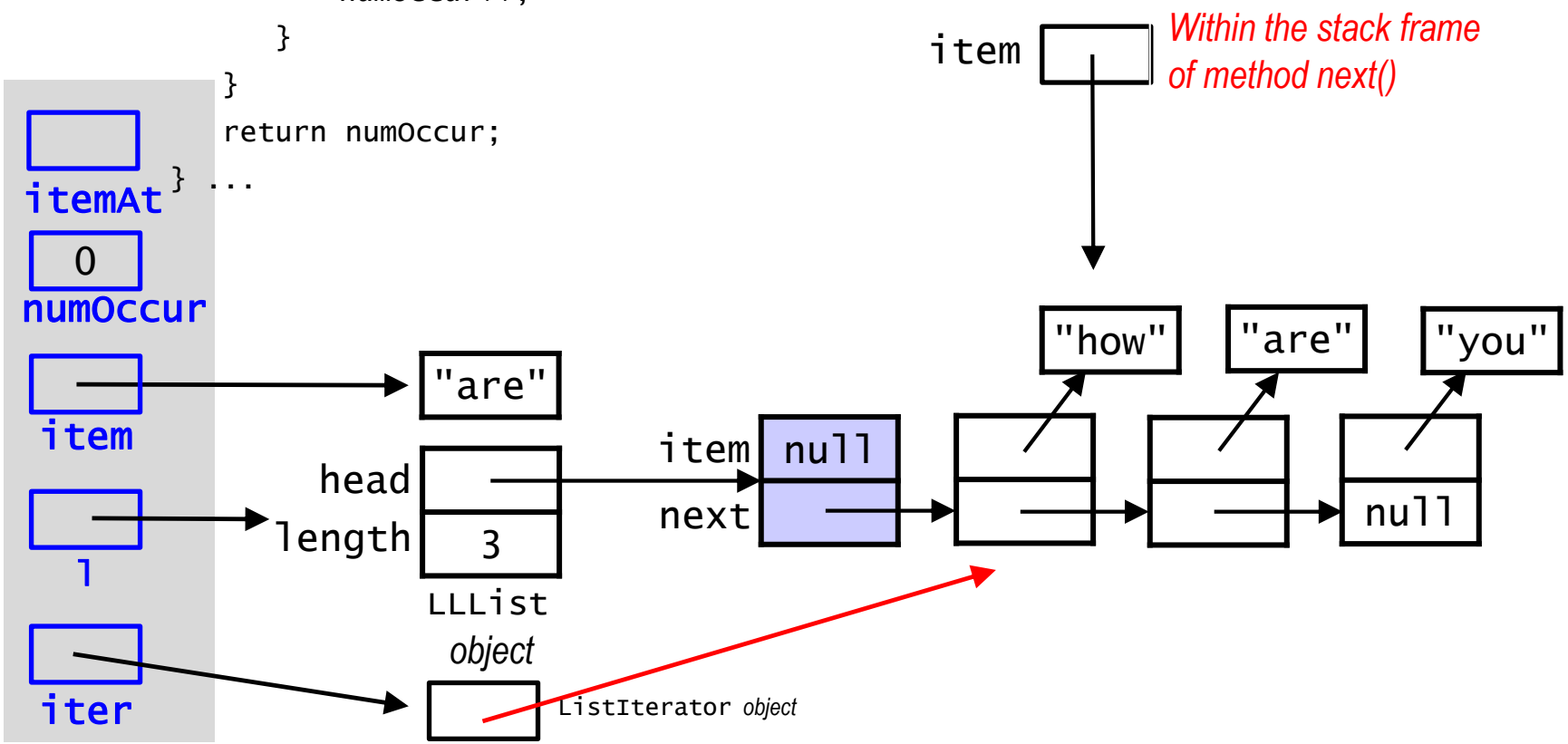
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



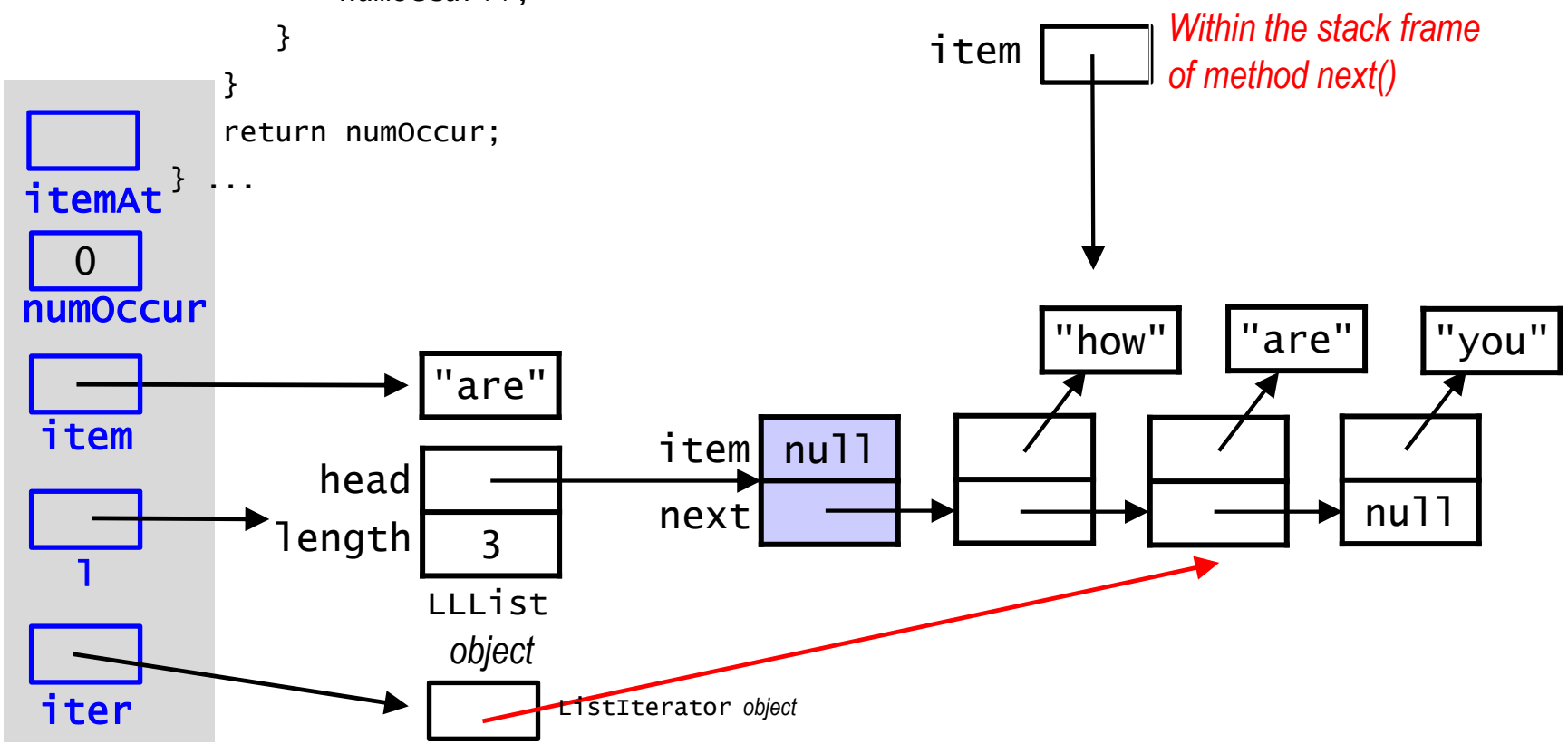
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



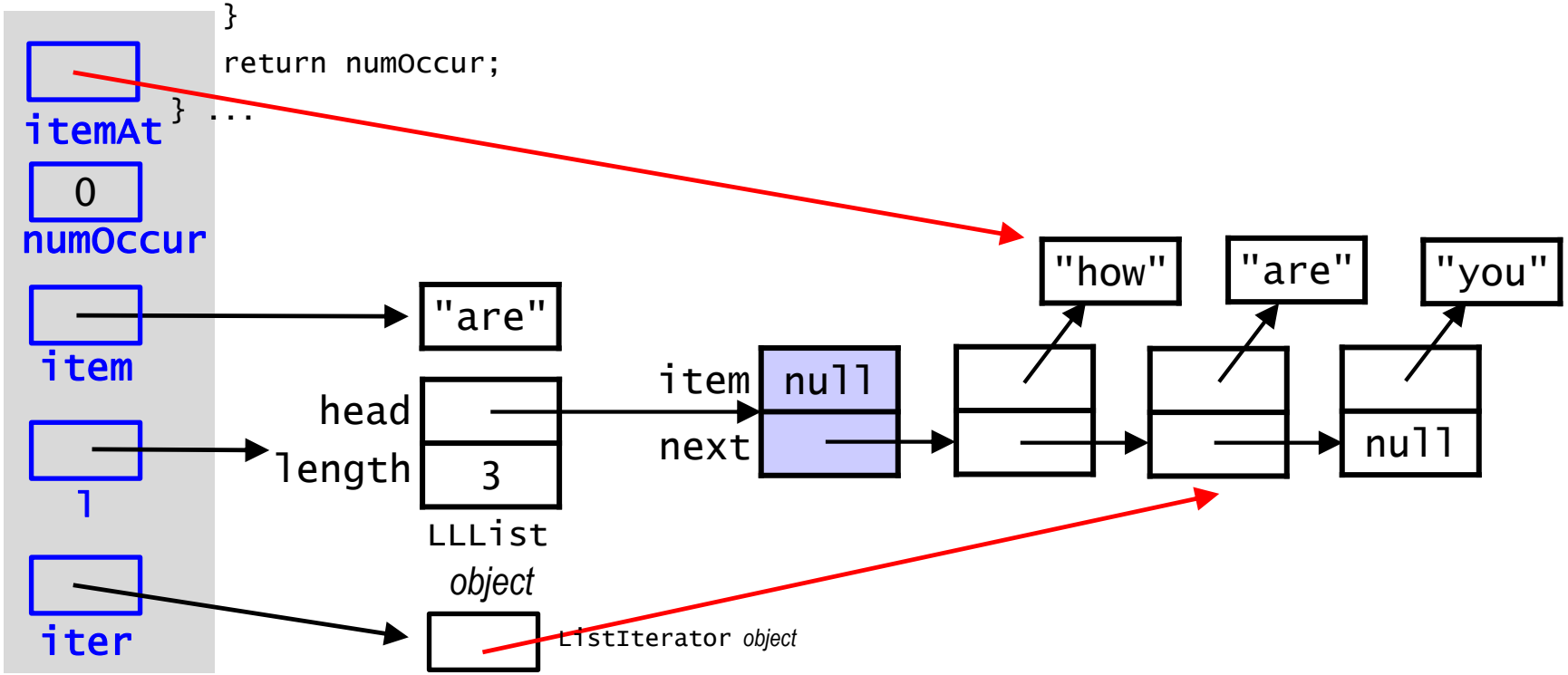
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



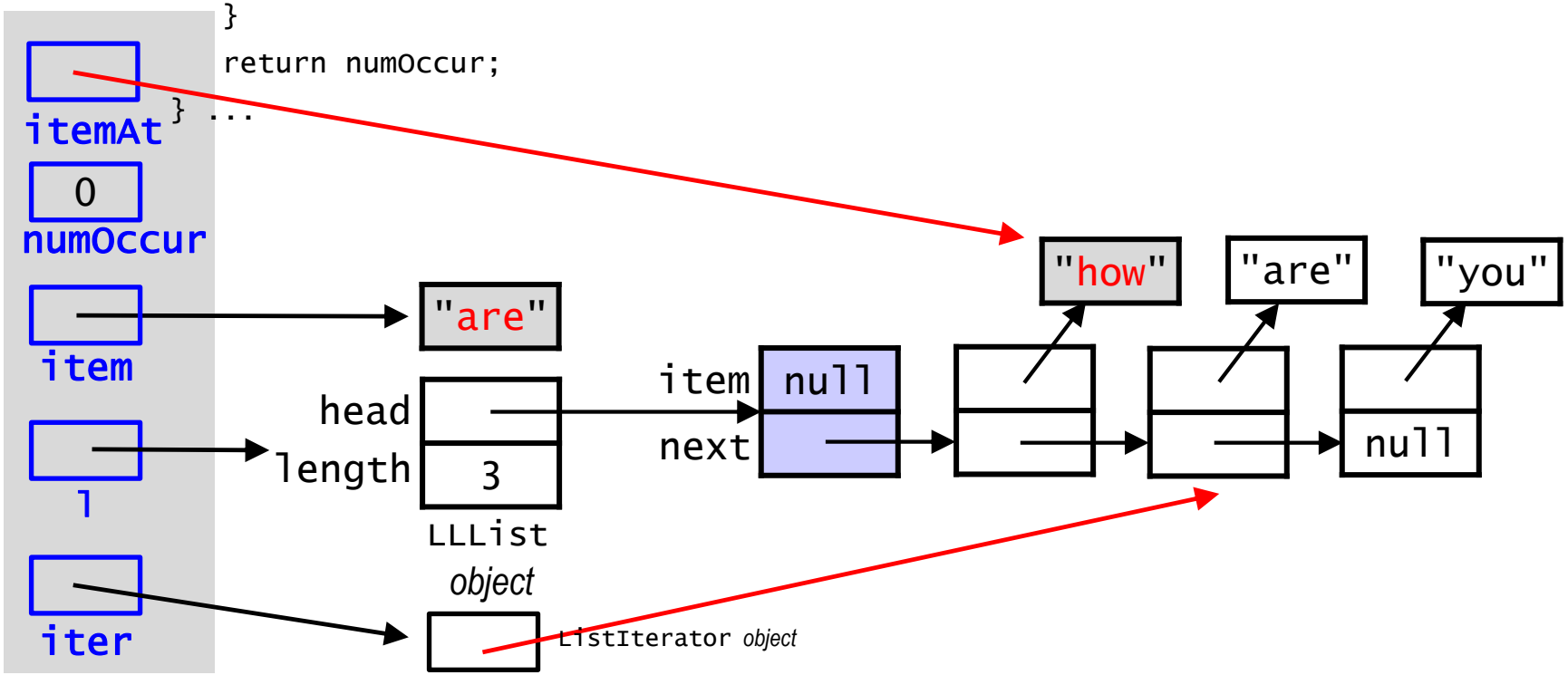
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



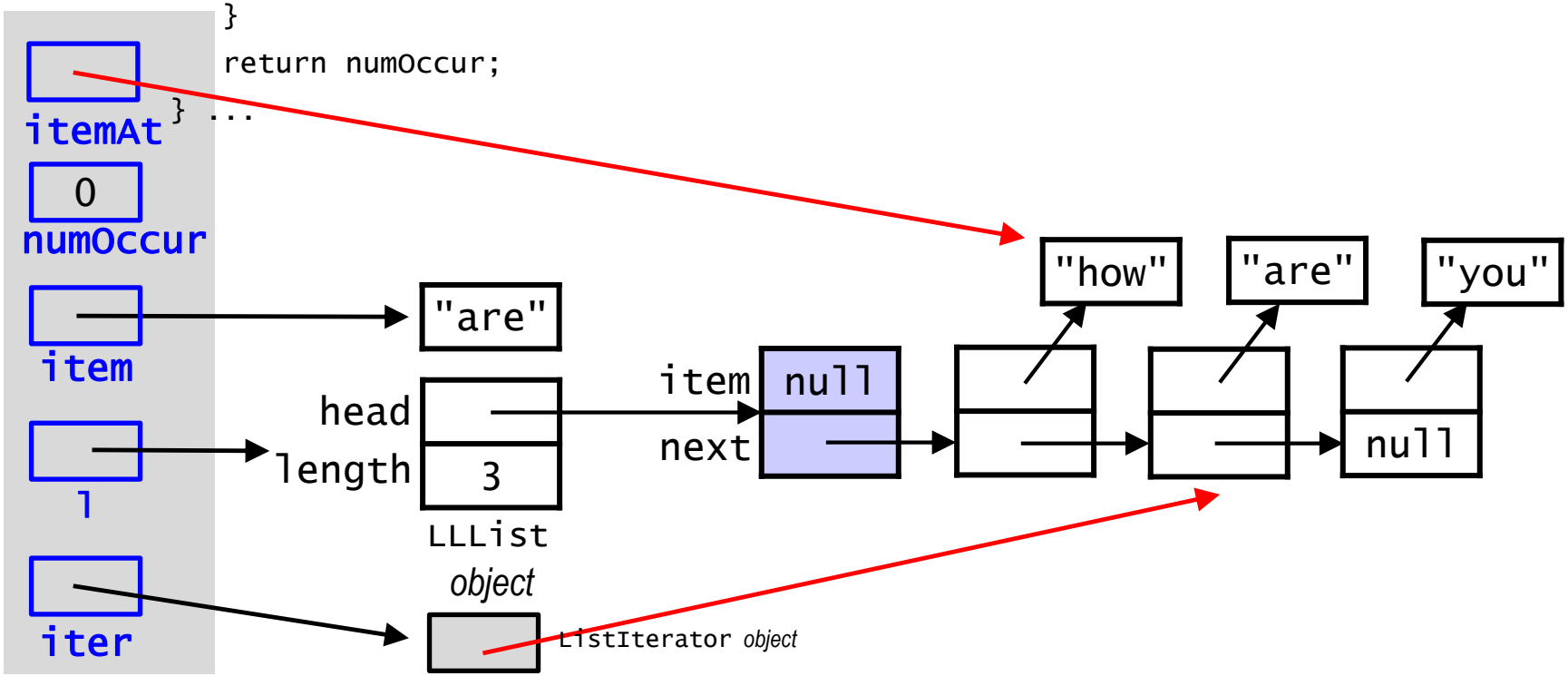
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



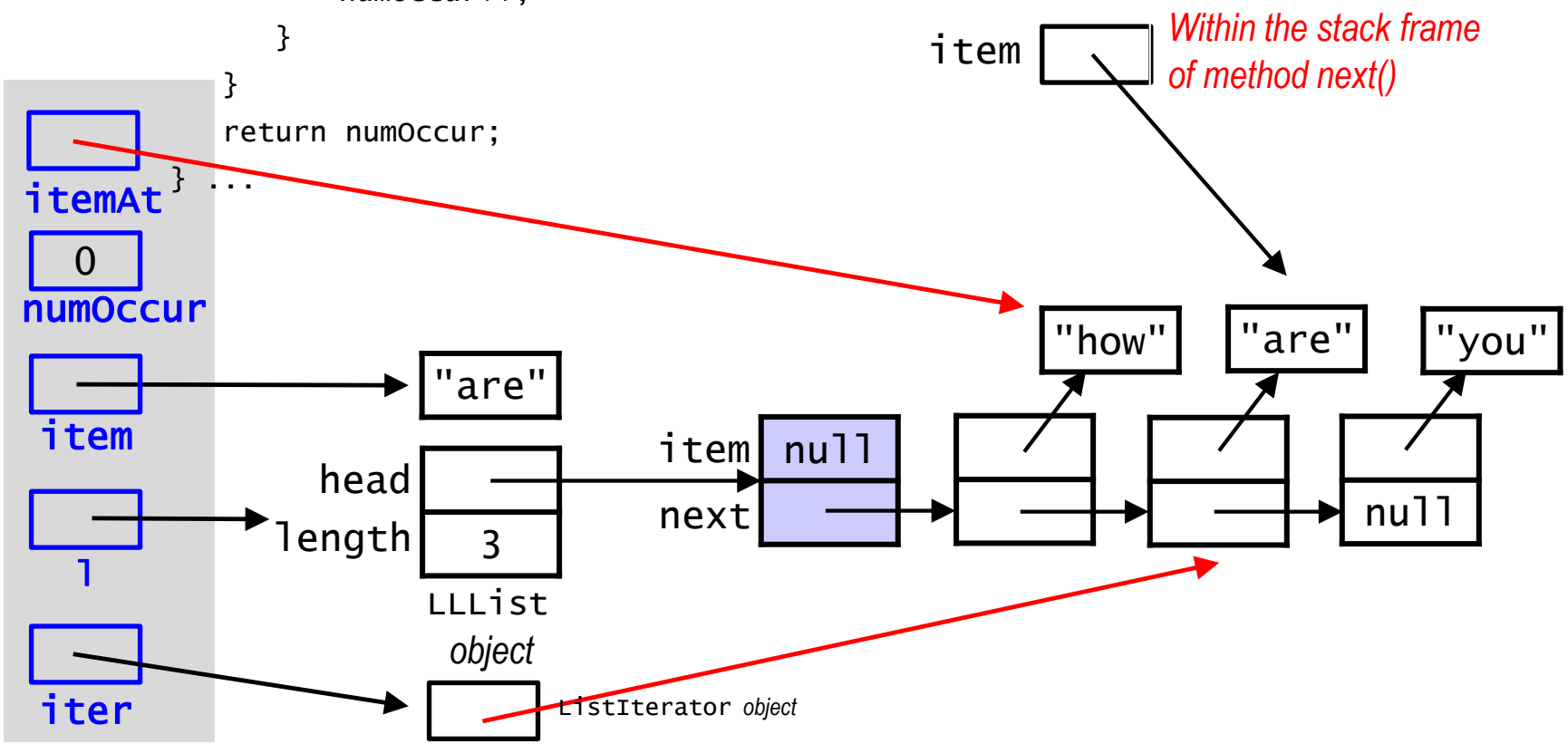
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



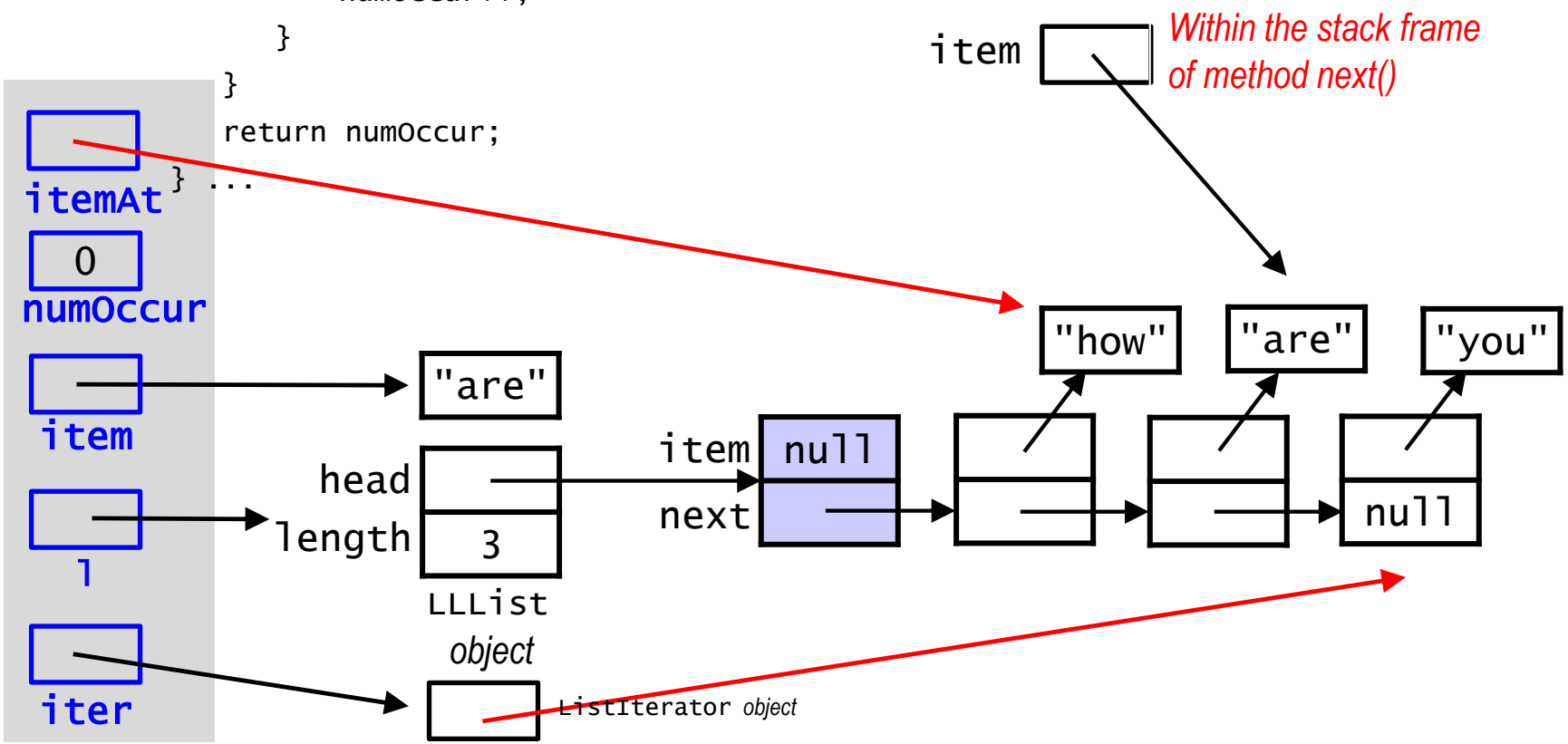
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



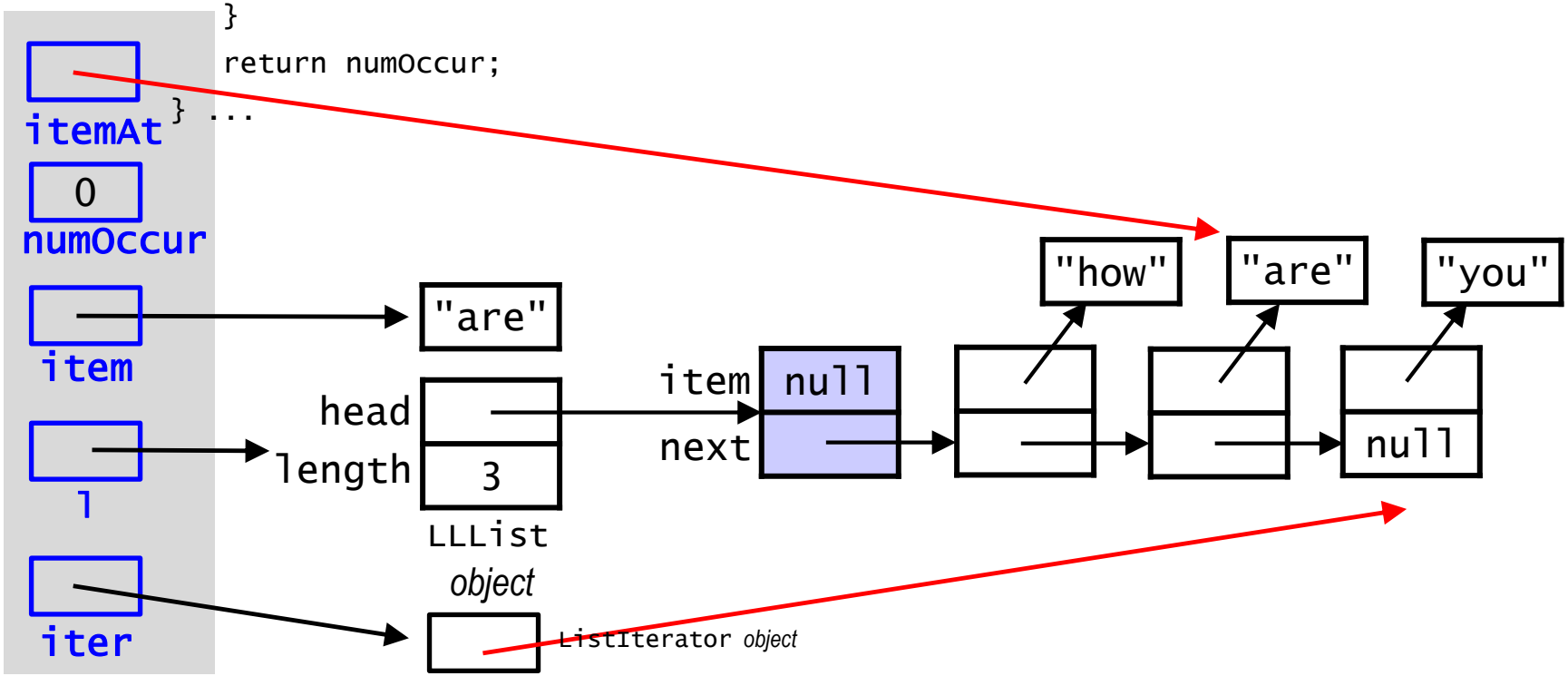
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



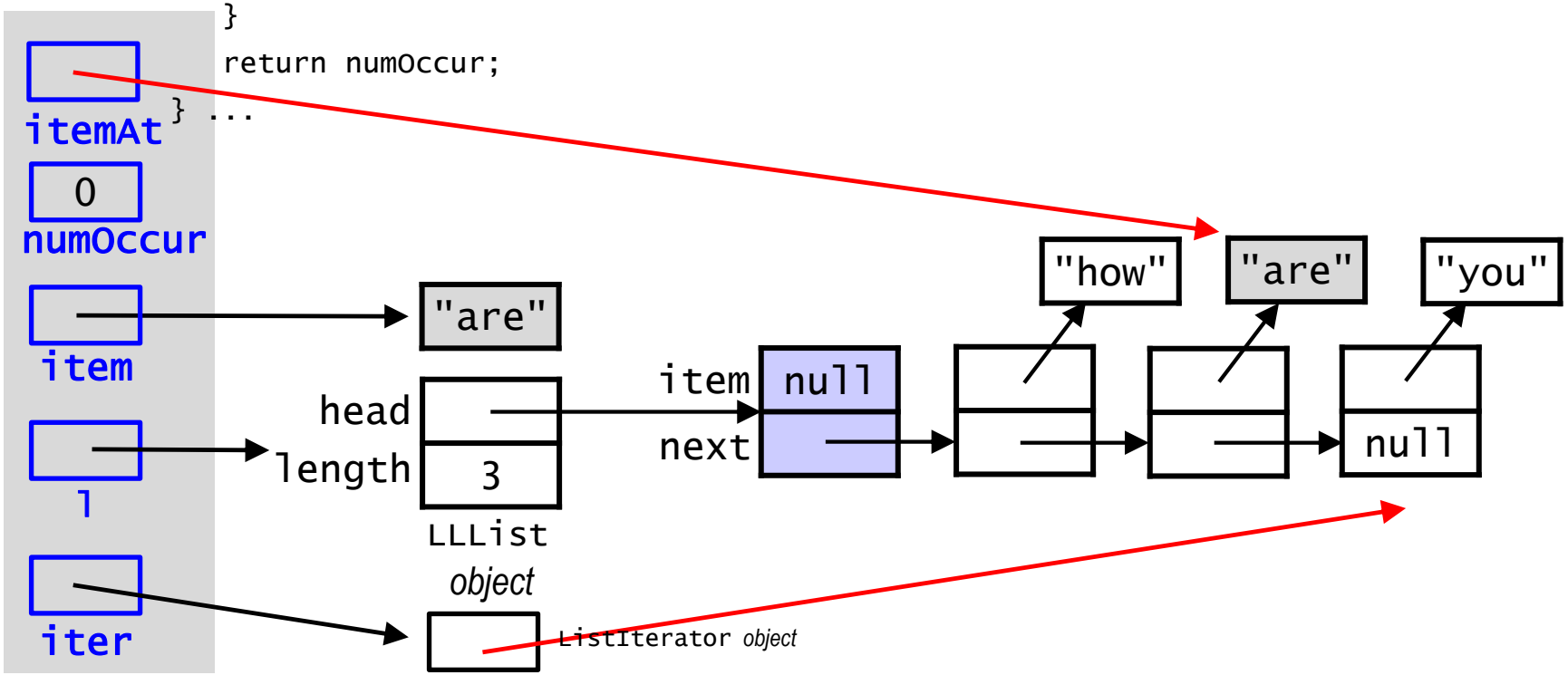
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



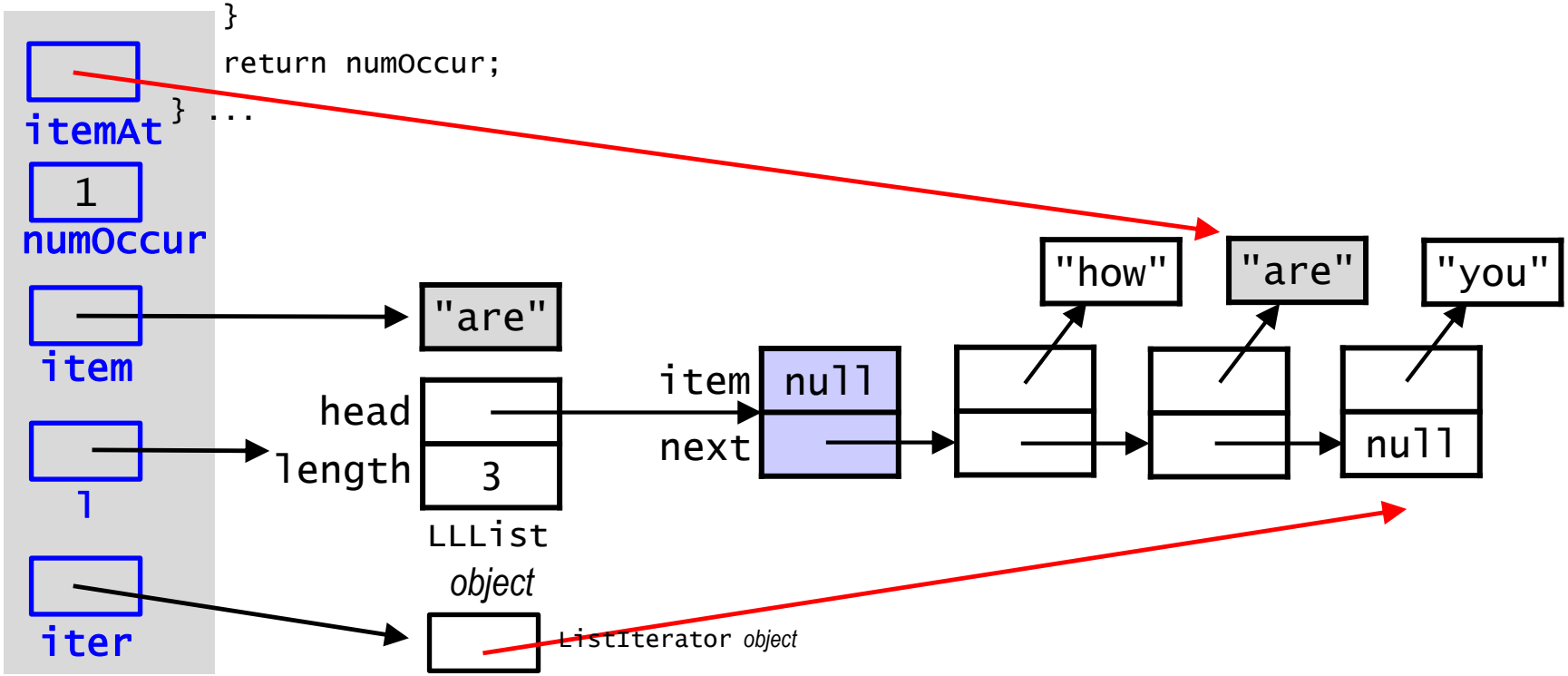
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



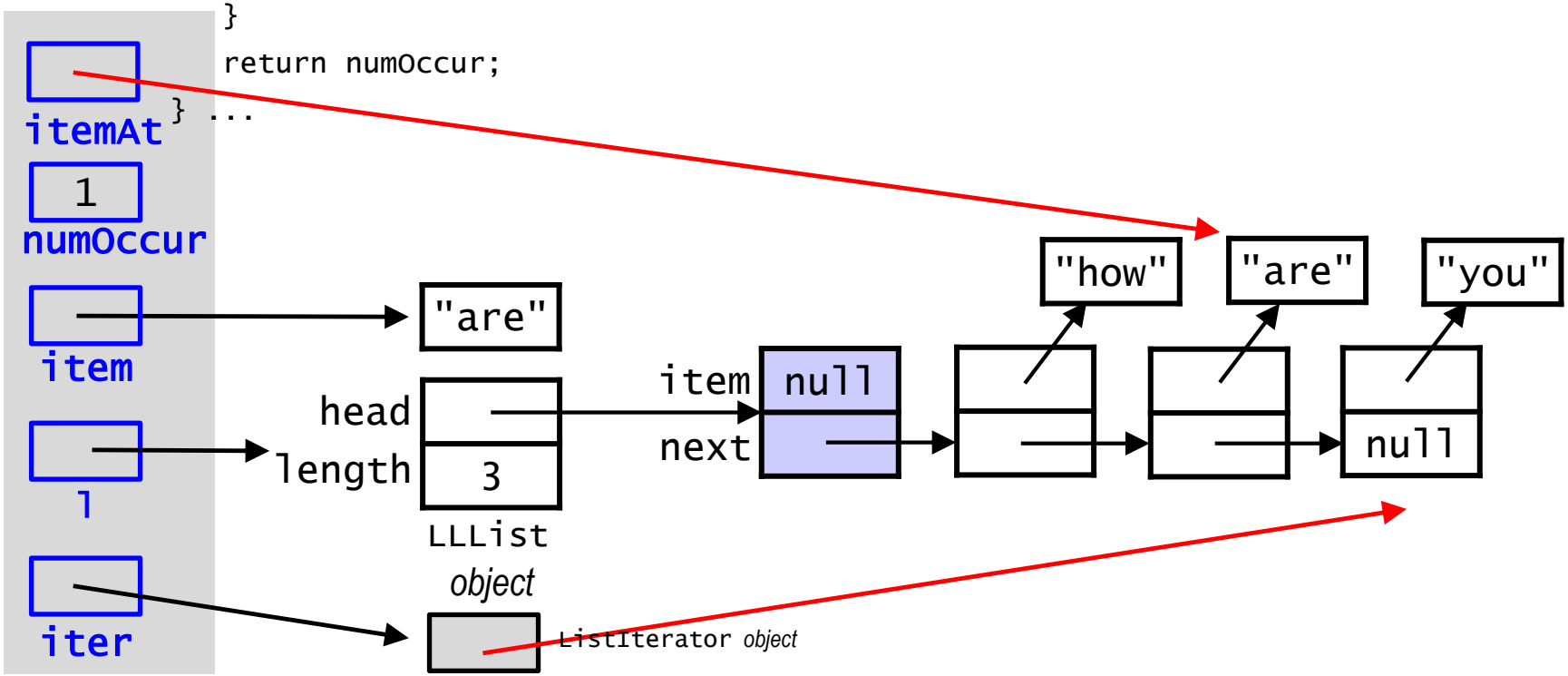
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



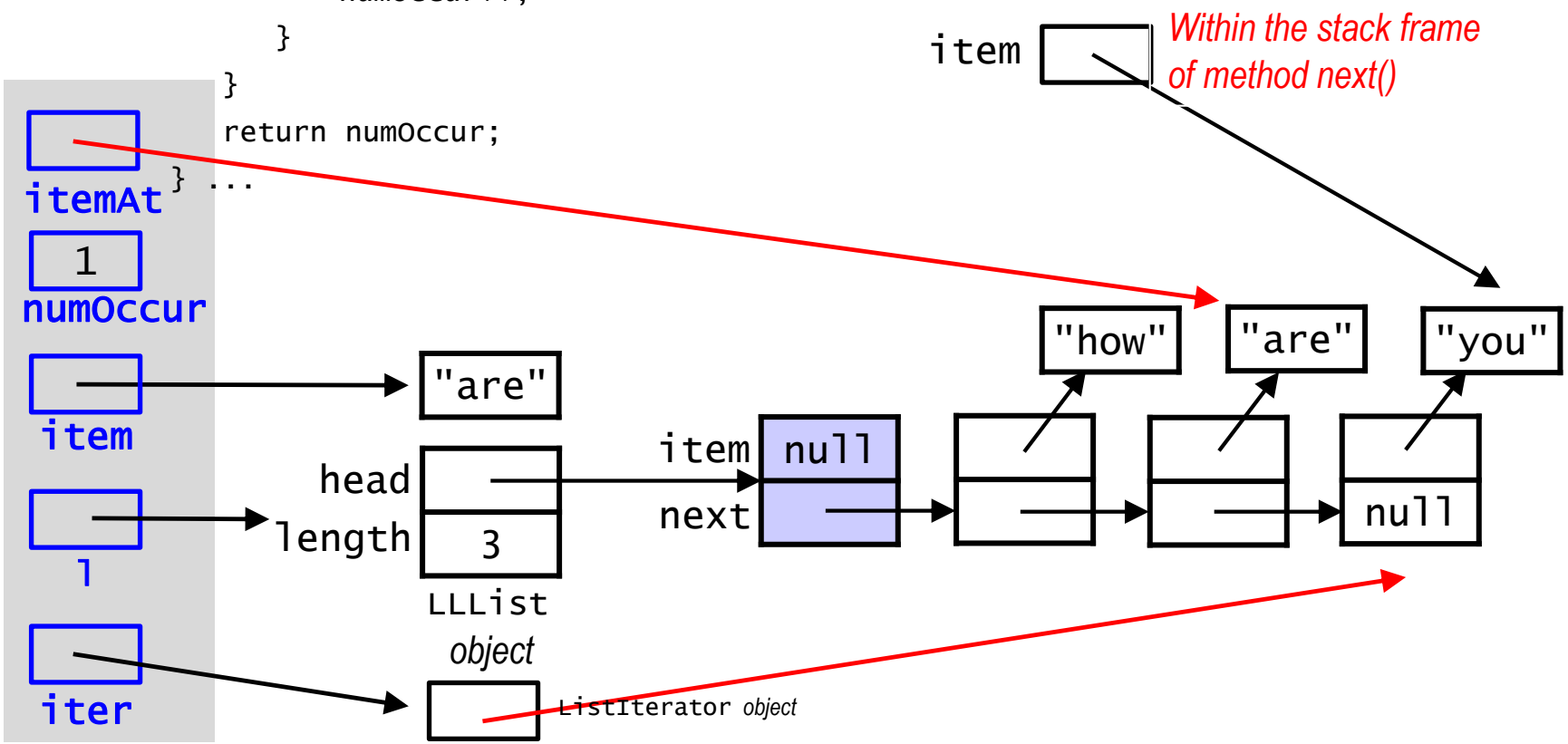
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



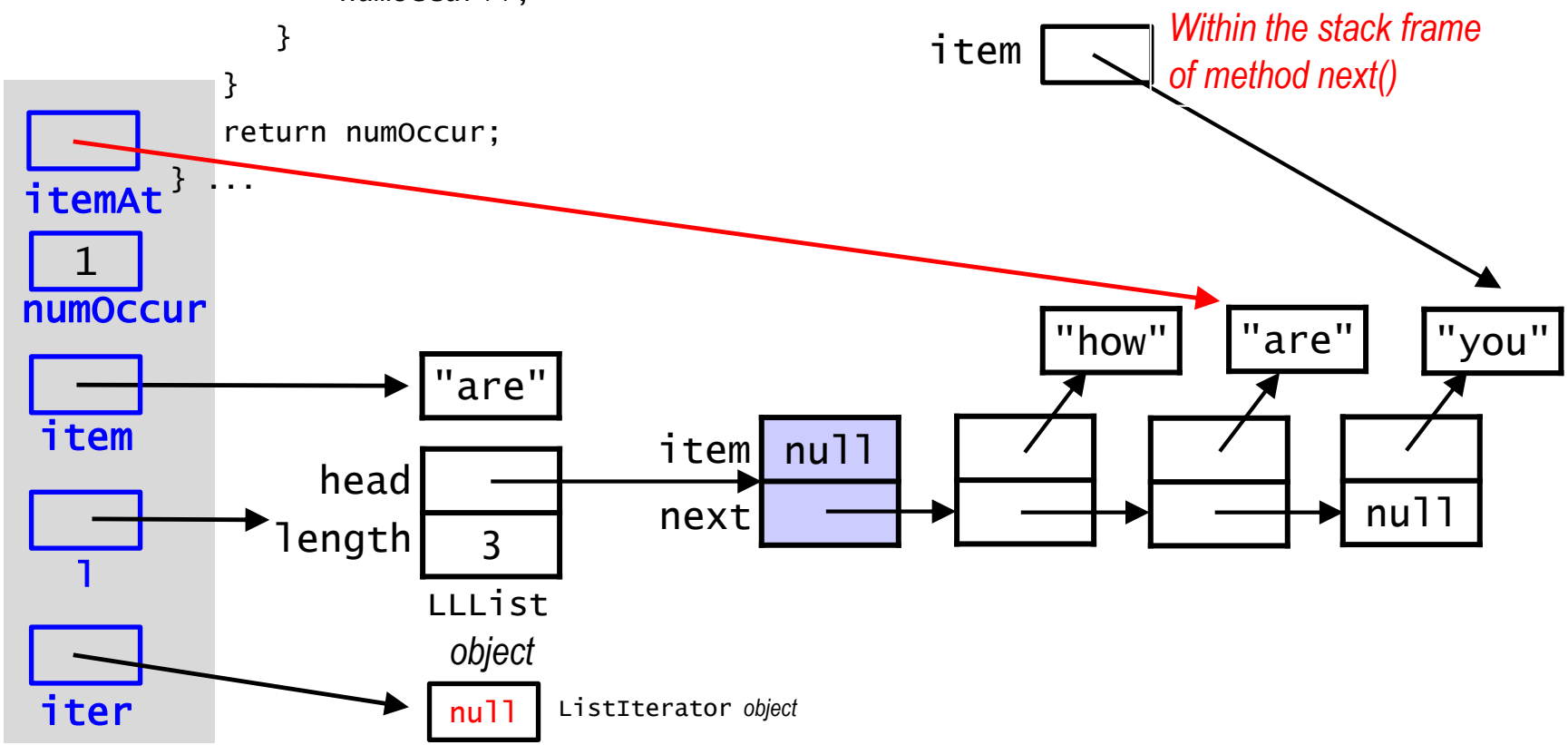
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



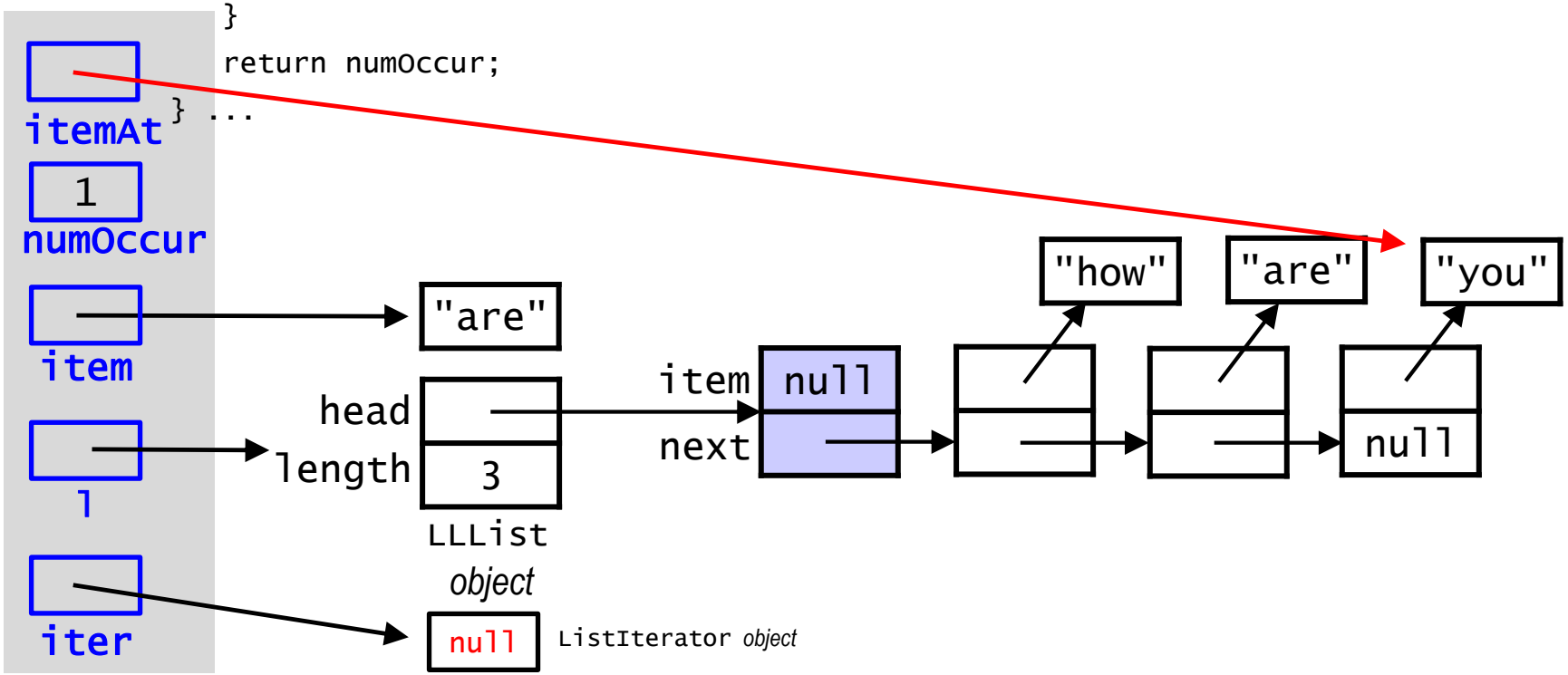
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



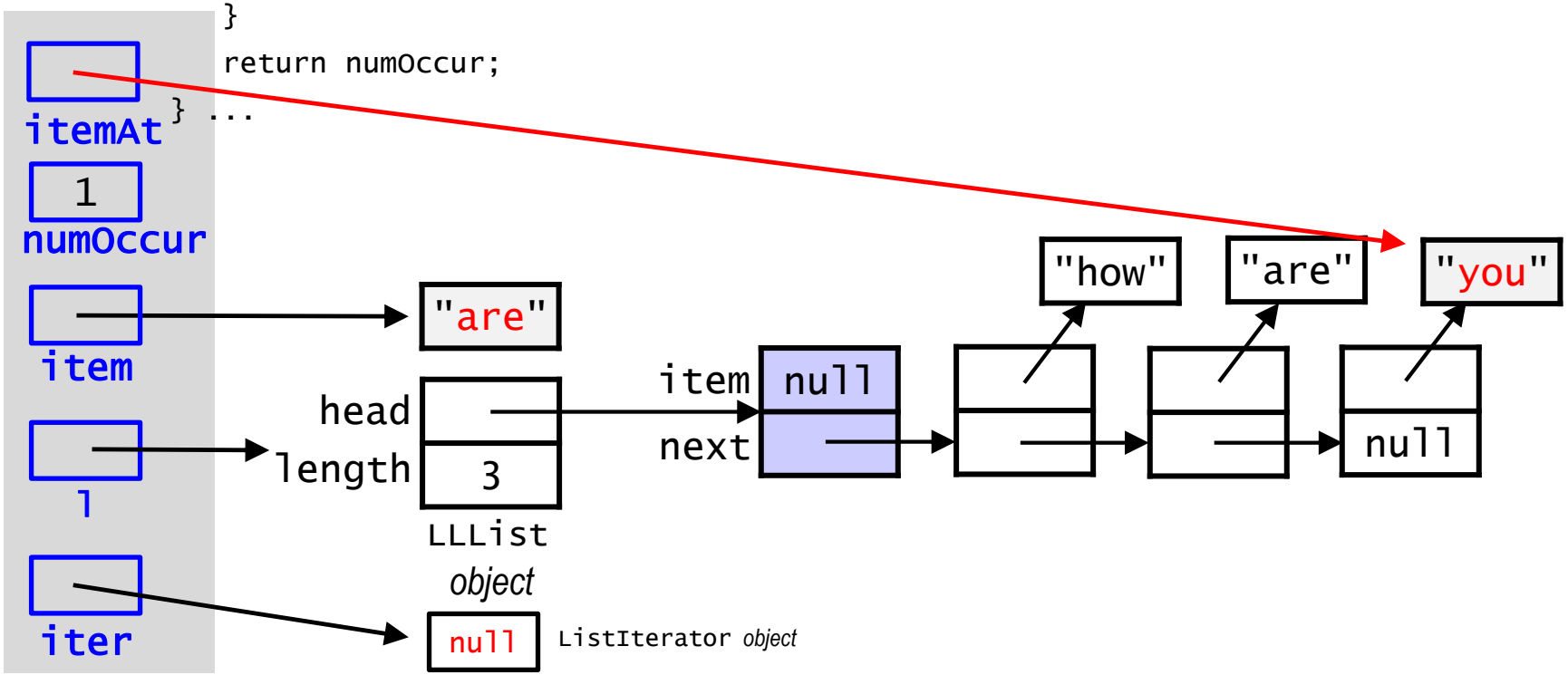
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



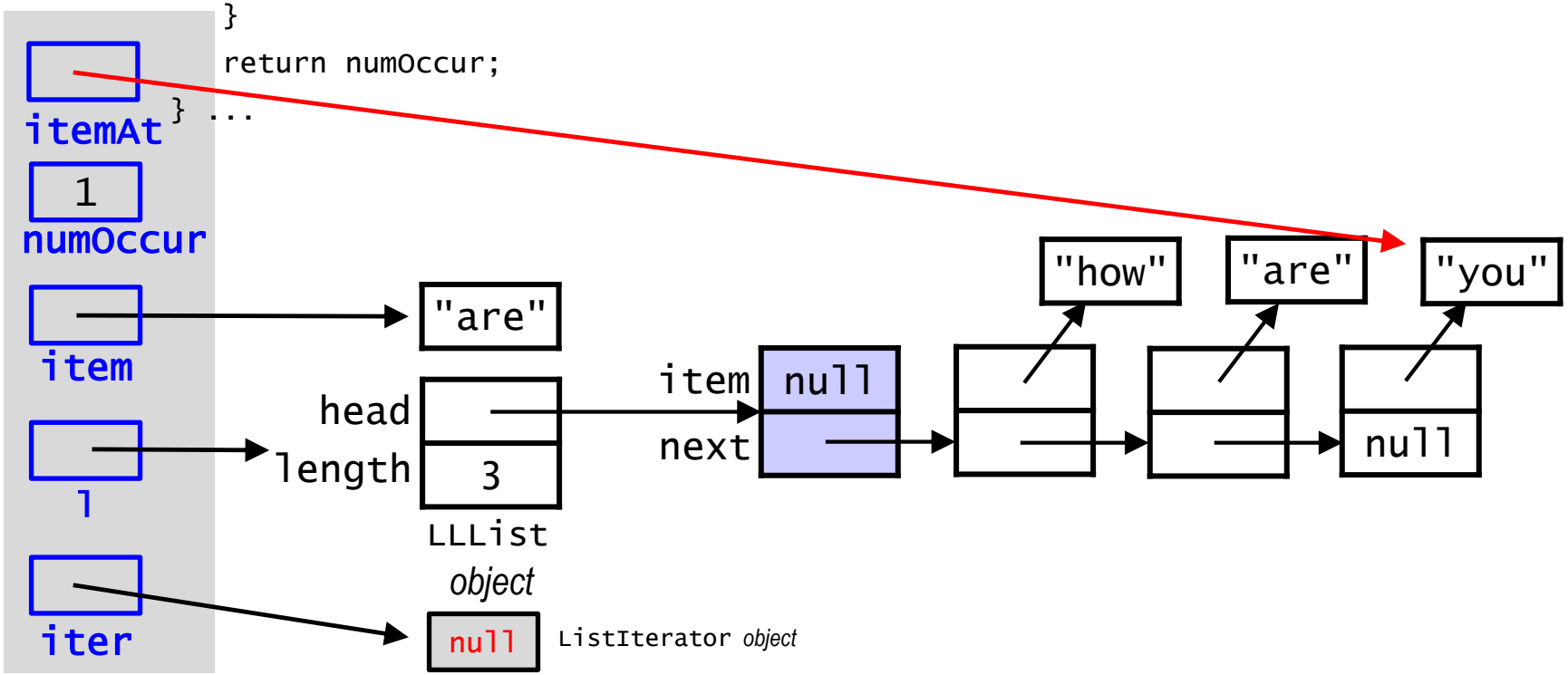
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



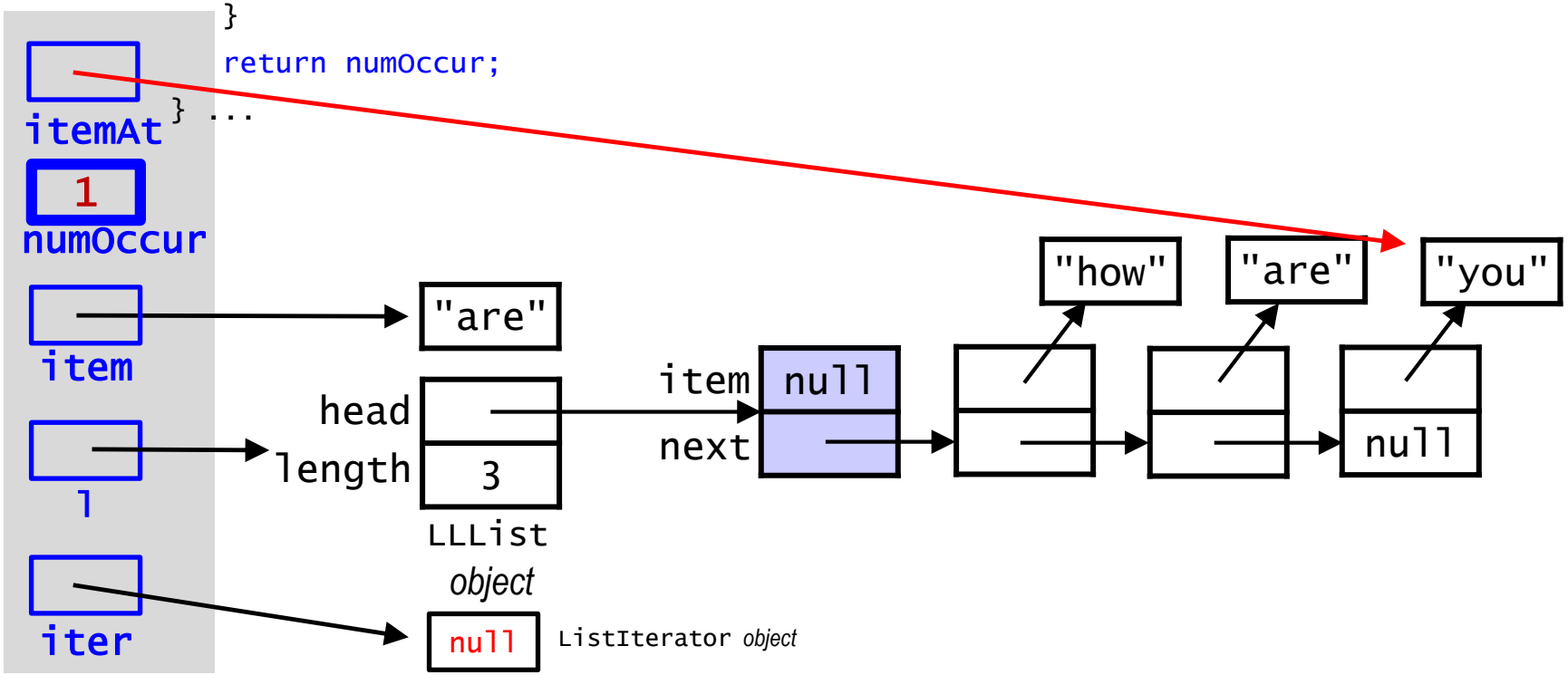
Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



Example: *LLList* list

```
public class MyClass {
    public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
       while ( iter.hasNext() ) {
           Object itemAt = iter.next();
           if (itemAt.equals(item)) {
               numOccur++;
           }
       }
       return numOccur;
    } ...
```



An Interface for List Iterators:

summary

Once the iterator interface has been implemented, we can create an instance of it and use it to externally traverse the list - regardless of the specific implementation of the List:

```
public class MyClass {
   public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
      while ( iter.hasNext() ) {
         Object itemAt = iter.next();

         if (itemAt.equals(item)) {
            numOccur++;
         }
      }
      return numOccur;
   } ...
```

Note that this method can be called on any implementation of the List Interface.
What are the implications of this?

An Interface for List Iterators:

summary

Once the iterator interface has been implemented, we can create an instance of it and use it to externally traverse the list - regardless of the specific implementation of the List:

```
public class MyClass {
   public static int numOccur(List l, Object item) {
      int numOccur = 0;
      ListIterator iter = l.iterator();
      while ( iter.hasNext() ) {
         Object itemAt = iter.next();

         if (itemAt.equals(item)) {
            numOccur++;
         }
      }
      return numOccur;
   } ...
```

Note that this method can be called on any implementation of the List Interface.

What are the implications of this?