# Iterators

## What is an Iterator?

An ***iterator*** is an abstract data type that allows us to iterate through the elements of a collection one by one

# What is an Iterator?

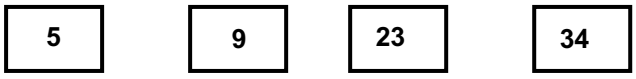
An ***iterator*** is an abstract data type that allows us to iterate through the elements of a collection one by one

Operations

- next: next element of the collection; ERROR if the element does not exist
- hasNext: true if there are more elements in the collection; false otherwise
- remove: removes the last element returned by the iterator



Consider an iterator for a collection storing the following elements:



Consider an iterator for a collection storing the following elements:

| 5 | 9 | 23 | 34 |
|---|---|---|---|

next:

2-5

Consider an iterator for a collection storing the following elements:

| 5 | 9 | 23 | 34 |
|---|---|---|---|

next: 5

2-6

Consider an iterator for a collection storing the following elements:

| 5 | 9 | 23 | 34 |
|---|---|---|---|

next:

2-7

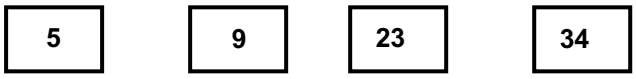
Consider an iterator for a collection storing the following elements:

| 5 | 9 | 23 | 34 |
|---|---|---|---|

next: 9

2-8

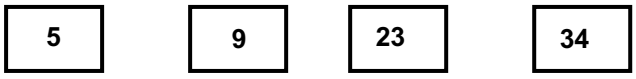
Consider an iterator for a collection storing the following elements:




hasNext:

2-9

Consider an iterator for a collection storing the following elements:




hasNext: true

2-10

Consider an iterator for a collection storing the following elements:

| | | | |
|---|---|---|---|
| 5 | 9 | 23 | 34 |

remove

9-11

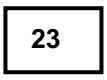
Consider an iterator for a collection storing the following elements:

| | | |
|---|---|---|
| 5 | 23 | 34 |

remove

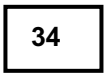
9-12

Consider an iterator for a collection storing the following elements:


next: 


2-13

Consider an iterator for a collection storing the following elements:


next: 


2-14

# **Iterator** Interface

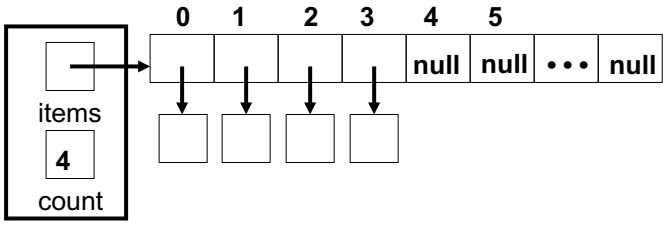
```
public interface Iterator<T> {
    public boolean hasNext( );
    public T next( );
    public void remove( ); // (optional operation)
}
```

It is in the java.util package of the Java API



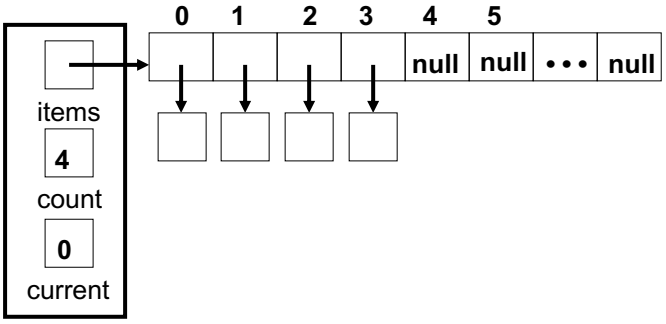
# Array Iterator

Consider a collection of data items stored in an array



# Array Iterator

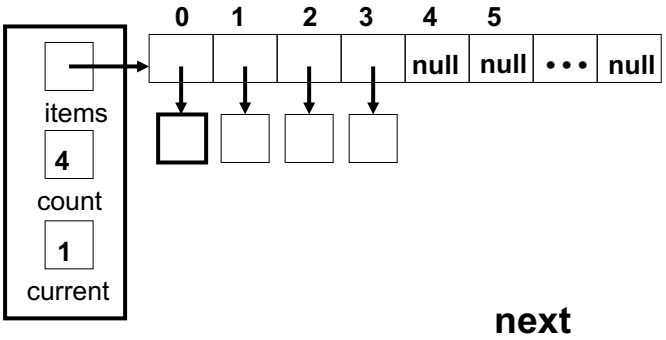
Consider a collection of data items stored in an array




9-17

# Array Iterator

Consider a collection of data items stored in an array




9-18

```
// Represents an iterator over the elements of an array
import java.util.*;
public class ArrayIterator<T> implements Iterator<T> {

   // Attributes
   private int count;   // number of elements in collection
   private int current; // current position in the iteration
   private T[ ] items;   // items in the collection

   // Constructor: sets up this iterator using the
   // specified items
   public ArrayIterator (T[ ] collection, int size) {
         items = collection;
         count = size;
         current = 0;
   }
```

2-19

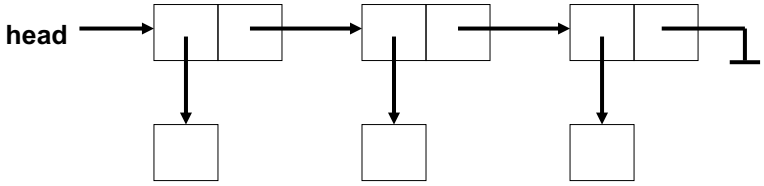
```
   // Returns true if this iterator has at least one
   // more element to deliver in the iteration
   public boolean hasNext( )  {
         return (current < count);
   }

   // Returns the next element in the iteration.
   // If there are no more elements in this iteration,
   // throws an exception.
   public T next( )  {
         if (! hasNext( ))
           throw new NoSuchElementException( );
         current++;
         return items[current - 1];
   }
}
```

2-20

# Linked Iterator

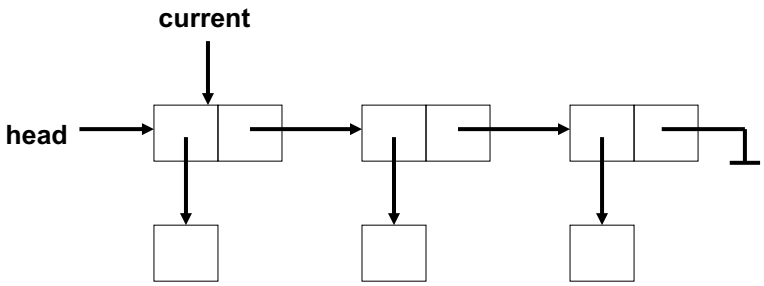
Consider a collection of data items stored in a linked list.




9-21

# Linked Iterator

Consider a collection of data items stored in a linked list.

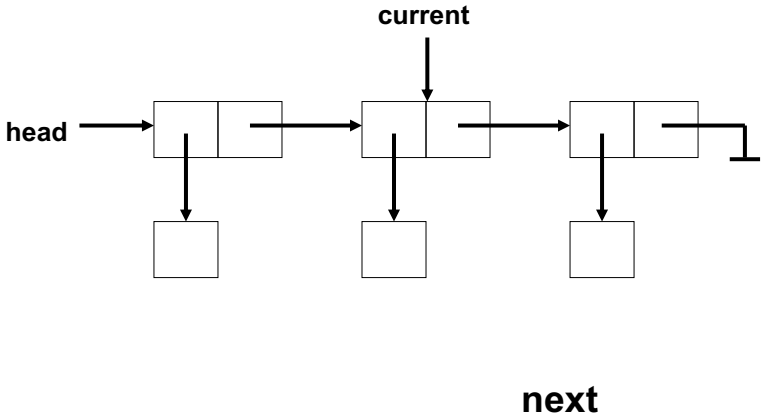



**next**

9-22

# Linked Iterator

Consider a collection of data items stored in a linked list.




9-23

```
import java.util.*;
public class LinkedIterator<T> implements Iterator<T> {

  // Attributes
   private LinearNode<T> current; // current position

   //  Constructor: Sets up this iterator
   public LinkedIterator (LinearNode<T> collection){
          current = collection;
}
```

9-24

```
  //  Returns true if this iterator has at least one more element
  //  to deliver in the iteration.
  public boolean hasNext( ) {
    return (current != null);
  }
  //  Returns the next element in the iteration. If there are no
  //  more elements in this iteration, throws an exception.
  public T next( ) {
    if (! hasNext( ))
        throw new NoSuchElementException( );
    T result = current.getElement( );
    current = current.getNext( );
    return result;
  }
}
```

9-25

# Iterators for a Collection

A List ADT can be implemented using, for example, an array or a linked list. For each implementation we can add an iterator operation that returns an iterator for the corresponding list.

9-26

# iterator method for ArrayList

```
/**
* Returns an iterator for the elements currently in this list.
*
* @return  an iterator for the elements in this list
*/
   public Iterator<T> iterator()  {
      return new ArrayIterator<T> (list, size);
   }
```



# iterator method for ArrayList

```
/**
* Returns an iterator for the elements currently in this list.
*
* @return  an iterator for the elements in this list
*/
   public Iterator<T> iterator()  {
      return new ArrayIterator<T> (list, size);
   }
```

An application can then declare an iterator as

```
ArrayList<String> a = new ArrayList<String>();
. . .
Iterator<String> iter = a.iterator();
```

## iterator method for LinkedList

```
/**
* Returns an iterator for the elements currently in this list.
* @return  an iterator for the elements in this list
*/
public Iterator<T> iterator( ) {
   return new LinkedIterator<T> (list);
}
```

An application can declare an iterator as

```
LinkedList<String> list = new LinkedList<String>();
. . .
Iterator<String> iter = list.iterator();
```

9-29

## Using an Iterator in an Application

If we want to print the elements in the iterator we can use this code:

```
while(iter.hasNext()) {
        System.out.println(iter.next());
}
```

This will work regardless of whether iter was obtained from theArrayList or from the LinkedList!

9-30

# Iterators

- An iterator abstracts the process of scanning through a collection of elements
- It maintains a cursor that sits between elements in the list, or before the first or after the last element
- Methods of the Iterator ADT:
  - hasNext(): returns true so long as the list is not empty and the cursor is not after the last element
  - next(): returns the next element
- Extends the concept of position by adding a traversal capability
- Implementation with an array or singly linked list



# Iterable Classes

- An iterator is typically associated with an another data structure, which can implement the Iterable ADT
- We can augment the Stack, Queue, Vector, List and Sequence ADTs with method:
  - Iterator<E> iterator(): returns an iterator over the elements
  - In Java, classes with this method extend Iterable<E>
- Two notions of iterator:
  - snapshot: freezes the contents of the data structure at a given time
  - dynamic: follows changes to the data structure
  - In Java: an iterator will fail (and throw an exception) if the underlying collection changes unexpectedly

# The For-Each Loop

- Java provides a simple way of looping through the elements of an Iterable class:
  - for (type name: expression)

    loop_body
  - For example:

```
List<Integer> values;
int sum=0
for (Integer i : values)
  sum += i;   // boxing/unboxing allows this
```



# Implementing Iterators

- Array based
  - array A of the elements
  - index i that keeps track of the cursor
- Linked list based
  - doubly-linked list L storing the elements, with sentinels for header and trailer
  - pointer p to node containing the last element returned (or the header if this is a new iterator).
- We can add methods to our ADTs that return iterable objects, so that we can use the for-each loop on their contents

# List Iterators in Java

- Java uses a the ListIterator ADT for node-based lists.
- This iterator includes the following methods:
  - add(e): add e at the current cursor position
  - hasNext(): true if there is an element after the cursor
  - hasPrevious: true if there is an element before the cursor
  - previous(): return the element e before the cursor and move cursor to before e
  - next(): return the element e after the cursor and move cursor to after e
  - set(e): replace the element returned by last next or previous operation with e
  - remove(): remove the element returned by the last next or previous method



# Why use Iterators?

- Traversing through the elements of a collection is very common in programming, and iterators provide a *uniform* way of doing so.
- Advantage? Using an iterator, we don't need to know how the collection is implemented!