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:
Consider an iterator for a collection storing the following elements:

5  9  23  34

next: 5
Consider an iterator for a collection storing the following elements:

next:
Consider an iterator for a collection storing the following elements:

\[
\begin{array}{cccc}
5 & 9 & 23 & 34 \\
\end{array}
\]

\textbf{next:} 9
Consider an iterator for a collection storing the following elements:

```
5 9 23 34
```

hasNext:
Consider an iterator for a collection storing the following elements:

```
5 9 23 34
```

\texttt{hasNext}: true
Consider an iterator for a collection storing the following elements:

\[ \begin{array}{cccc}
5 & 9 & 23 & 34 \\
\end{array} \]

`remove`
Consider an iterator for a collection storing the following elements:

\[
\begin{array}{ccc}
5 & 23 & 34 \\
\end{array}
\]

\text{remove}
Consider an iterator for a collection storing the following elements:

```
next: 23
```

5 23 34
Consider an iterator for a collection storing the following elements:

5  23  34

next: 34
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
Consider a collection of data items stored in an array.
Array Iterator

Consider a collection of data items stored in an array

```
0  1  2  3  4  5
null null ••• null
```

- **items**: 4
- **count**: 1
- **current**: 1

next
// 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;
}
public boolean hasNext() {
    return (current < count);
}

public T next() {
    if (!hasNext())
        throw new NoSuchElementException();
    current++;
    return items[current - 1];
}
Linked Iterator

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

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

Consider a collection of data items stored in a linked list.
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;
    }
}
// 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;
}
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.
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);
}
/**
 * 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**

```java
public Iterator<T> iterator() {
    return new LinkedIterator<T>(list);
}
```

An application can declare an iterator as:

```java
LinkedList<String> list = new LinkedList<String>();
...
Iterator<String> iter = list.iterator();
```
Using an Iterator in an Application

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

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

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