Iterator interface,
Iterable interface

Two very useful Interfaces
Need for Iterators:  

to simplify the step of processing all the elements of a collection

- Very often we need to have a method that accesses ALL the elements in a collection in a simple way
- So far we do not access the elements of a Graph or a Queue with the same code.

- Given a collection, (e.g., Stack, Queue, Graph, Tree, HashTable) we would like to use the same code to access each and every element in the collection!
- Like: get me another element that I did not process before

- An iterator is an object that allows the user to acquire and use each element in a collection

- It works neatly with a collection, but is a separate object
Example of an Iterator use

Suppose WCDorms is an LinkedList of String objects

```java
LinkedList<String> WCDorms = new LinkedList<>();
Iterator<String> itr = WCDorms.iterator();
while (itr.hasNext()) {
    System.out.println(itr.next());
}
```

The first line obtains the iterator, then the loop uses hasNext and next to access and print each dorm

What does this code remind you of?
9.3 – The Iterator Interface

An iterator is an object that provides a means of processing a collection of objects, one at a time.

By implementing the `Iterator` interface, a class formally establishes that objects of that type are iterators.

Now, the `for-each` version of the `for` loop can be used to process the items in the iterator.

```java
java.util

Interface Iterator<E>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean hasNext()</td>
<td>Returns true if the iteration has more elements.</td>
</tr>
<tr>
<td>E next()</td>
<td>Returns the next element in the iteration.</td>
</tr>
<tr>
<td>void remove()</td>
<td>Removes from the underlying collection the last element returned by the iterator (optional operation).</td>
</tr>
</tbody>
</table>

Scanner

<<Iterator>>
Iterators work with “for-each” loops

- A for-each loop can be used for the same goal:

  ```java
  for (String dorm : WCDorms)
      System.out.println(dorm);
  ```

- The for-each loop uses an iterator behind the scenes

- The for-each loop can be used on any object that is Iterable
Different Data Structure, same way of processing all elements

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

/**
 * Example on how to use iterator of a LinkedList.
 * @author (TM)
 * @version (2022.11.09)
 */
public class LinkedListIteratorExample
{
    public static void main(String[] args)
    {
        LinkedList<String> dorms = new LinkedList<String>();
        dorms.add("Beebe");
        dorms.add("Clafflin");
        dorms.add("Stone-Davis");

        Iterator<String> it = dorms.iterator(); // Let's see what's in LL
        System.out.println("Linked List elements:");
        while(it.hasNext()){
            System.out.println(it.next());
        }

        dorms.add("Science Center ;");
        dorms.add("Casa Cervantes");
        dorms.add("Tower Court");

        it = dorms.iterator();
        System.out.println("\nNew LinkedList iterator:");
        while(it.hasNext()){
            System.out.println(it.next());
        }
    }
}

import java.util.Vector;
import java.util.Iterator;

/**
 * Example on how to use iterator of a Vector.
 * @author (TM)
 * @version (2022.11.09)
 */
public class VectorIteratorExample
{
    public static void main(String args[]){
        Vector<String> dorms = new Vector<String>();
        dorms.add("Beebe");
        dorms.add("Clafflin");

        Iterator<String> it = dorms.iterator();
        System.out.println("Vector elements:");
        while(it.hasNext()) {
            String obj = it.next();
            System.out.println(obj);
        }

        dorms.add("Stone-Davis");
        dorms.add("Science Center ;");
        dorms.add("Casa Cervantes");
        dorms.add("Tower Court");

        it = dorms.iterator();
        System.out.println("\nNew Vector iterator:");
        while(it.hasNext()) {
            System.out.println(it.next());
        }
    }
}
```
Iterators

- You shouldn't assume that an iterator will deliver the elements in any particular order unless the documentation explicitly says you can.

- Also, remember that an iterator is accessing the elements stored in the collection.

- The structure of the underlying collection should not be changed while an iterator is being used.

- Most iterators in the Java API are *fail-fast*, meaning they throw an exception if the collection is modified while the iterator is active.
How about Iterable?

There are two key interfaces in the Java API related to iterators:
- `Iterator` – used to define an iterator
- `Iterable` – used to define a collection that provides an iterator

When we call a collection `Iterable`, it means it will provide an `Iterator` when requested

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Iterator&lt;E&gt; iterator()</code></td>
<td>Returns an iterator over a set of elements of type E.</td>
</tr>
</tbody>
</table>
Using Java Foundations’ ArrayIterator

It is a little different; it is more flexible:
It can remember where was the next() element
Using JavaFoundation’s ArrayIterator

```java
import javafoundations.ArrayIterator;

/**< *
 * Client program for testing ArrayIterator.
 * *
 * @author CS230 (TM)
 * @version 2020.03.29
 */

public class IteratorTest {

    public static void main(String args[]){
        ArrayIterator<String> dormsIter = new ArrayIterator<String>();
        dormsIter.add("Beebe");
        dormsIter.add("Claflin");
        dormsIter.add("Stone-Davis");

        while (dormsIter.hasNext())
            System.out.println(dormsIter.next());

        dormsIter.add("Dower");
        dormsIter.add("Casa Cervantes");
        dormsIter.add("Tower Court");

        while (dormsIter.hasNext())
            System.out.println(dormsIter.next());
    }
}
```
Implementing JF’s Iterator

It uses an array to store and access the elements. We could also have another implementation that uses a LinkedList.
Implementing an iterator using array

```java
import java.util.*;

class ArrayIterator<T> implements Iterator<T> {
    private int DEFAULT_CAPACITY = 10;
    private int count; // the number of elements in the iterator
    private int current; // the current position in the iteration
    private T[] items; // the iterator's storage for elements

    public ArrayIterator() {
        items = (T[]) (new Object[DEFAULT_CAPACITY]);
        count = 0;
        current = 0;
    }

    // more...
```
Implementing an iterator using array

```java
// Adds the specified item to this iterator.

public void add (T item)
{
    if (count == items.length)
        expandCapacity();
    items[count] = item;
    count++;
}

private void expandCapacity()
{
    T[] larger = (T []) (new Object[items.length*2]);

    int location = 0;
    for (T element : items)
        larger[location++] = element;

    items = larger;
}

(more...)
```
Implementing an iterator using array

```java
// 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, a NoSuchElementException is thrown.
public T next() {
    if (!hasNext())
        throw new NoSuchElementException();
    current++;
    return items[current - 1];
}

// The remove operation is not supported in this collection.
public void remove() throws UnsupportedOperationException {
    throw new UnsupportedOperationException();
}
```