CH7.
LIST AND ITERATOR ADTS

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH DATA STRUCTURES AND ALGORITHMS IN JAVA, GOODRICH, TAMASSIA AND GOLDWASSER (WILEY 2016)
ITERATORS

• An **iterator** is a software design pattern that abstracts the process of scanning through a sequence of elements, one element at a time.

  hasNext(): Returns true if there is at least one additional element in the sequence, and false otherwise.

  next(): Returns the next element in the sequence.

• Some iterators offer a third operation: **remove()** to modify the data structure while scanning its elements
USES OF ITERATORS

- Abstracts a series or collection of elements
  - A container, e.g., List or PositionalList
  - A stream of data from a network or file
  - Data generated by a series of computations, e.g., random numbers

- Facilitate generic programming of algorithms to operate on any source of data, e.g., finding the minimum element in the data

- Why?
  - While it is true we could just reimplement minimum as many times as needed, it is better to use a trusted single implementation for: (1) correctness – no silly typos and (2) efficiency – professional libraries are often better than what you could implement on your own.
THE ITERABLE INTERFACE

• Java defines a parameterized interface, named `Iterable`, that includes the following single method:
  • `iterator()`: Returns an iterator of the elements in the collection.

• An instance of a typical collection class in Java, such as an `ArrayList`, is `Iterable` (but not itself an iterator); it produces an iterator for its collection as the return value of the `iterator()` method.

• Each call to `iterator()` returns a new iterator instance, thereby allowing multiple (even simultaneous) traversals of a collection.
EXAMPLE IN PSEUDOCODE

• The following algorithm will compute the minimum of an iterable collection:

Algorithm minimum

Input: Iterable collection $I$ of comparable Elements

1. Iterator $it \leftarrow I$.iterator()
2. Element $min \leftarrow null$
3. while $it$.hasNext() do
   4. Element $e \leftarrow it$.next()
   5. if $e$.compareTo($min) < 0$ then
      6. $min \leftarrow e$
4. return $min$
EXAMPLE IN JAVA

The following code will compute the minimum of an Iterable collection:

```java
1. public static <E extends Comparable<E>> E minimum(
   Iterable<E> iterable) {
2.   Iterator<E> it = iterable.iterator();
3.   E min = null;
4.   while(it.hasNext()) {
5.     E e = it.next();
6.     if(e.compareTo(min) < 0) {
7.       min = e;
8.     }
9.   } return min;
10.}
```
EXERCISE

• Write an algorithm and a Java program using iterators to compute whether a collection contains only unique elements.
  • Test your generic method with both a Java ArrayList and a Java LinkedList
THE FOR-EACH LOOP

• Java’s Iterable class also plays a fundamental role in support of the “for-each” loop syntax:

```java
for (ElementType variable : collection) {
    loopBody
} // may refer to "variable"
```

• is equivalent to:

```java
Iterator<ElementType> iter = collection.iterator();
while (iter.hasNext()) {
    ElementType variable = iter.next();
    loopBody
} // may refer to "variable"
```
EXAMPLE IN PSEUDOCODE

• The following algorithm will compute the minimum of an iterable collection:

Algorithm minimum
Input: Iterable collection $I$ of comparable Elements
1. Element $min \leftarrow \text{null}$
2. for all Element $e \in I$ do
3. if $e$.$\text{compareTo}(min) < 0$ then
4. $min \leftarrow e$
5. return $min$
EXAMPLE IN JAVA

• The following code will compute the minimum of an Iterable collection:

```java
1. public static <E extends Comparable<E>> E minimum(
    Iterable<E> iterable) {
2.   E min = null;
3.   for (E e : iterable) {
4.     if (e.compareTo(min) < 0) {
5.       min = e;
6.     }
7.   return min;
8. }
```
EXERCISE

• Simplify your algorithm and Java program using the for-each loop construct to determine whether a collection contains only unique elements.
FOR-EACH VS ITERATORS

• For-each is not always a replacement for iterators
  • In fact it only replaces the most common use of iterators – iterating entirely through a collection
  • When you can’t use a for-each loop, use iterators
    • Essentially, when you need more power, use more power

• Remember this is about generic programming. Iterators abstract the underlying collection. When you know your collection, you might be able to do something different.