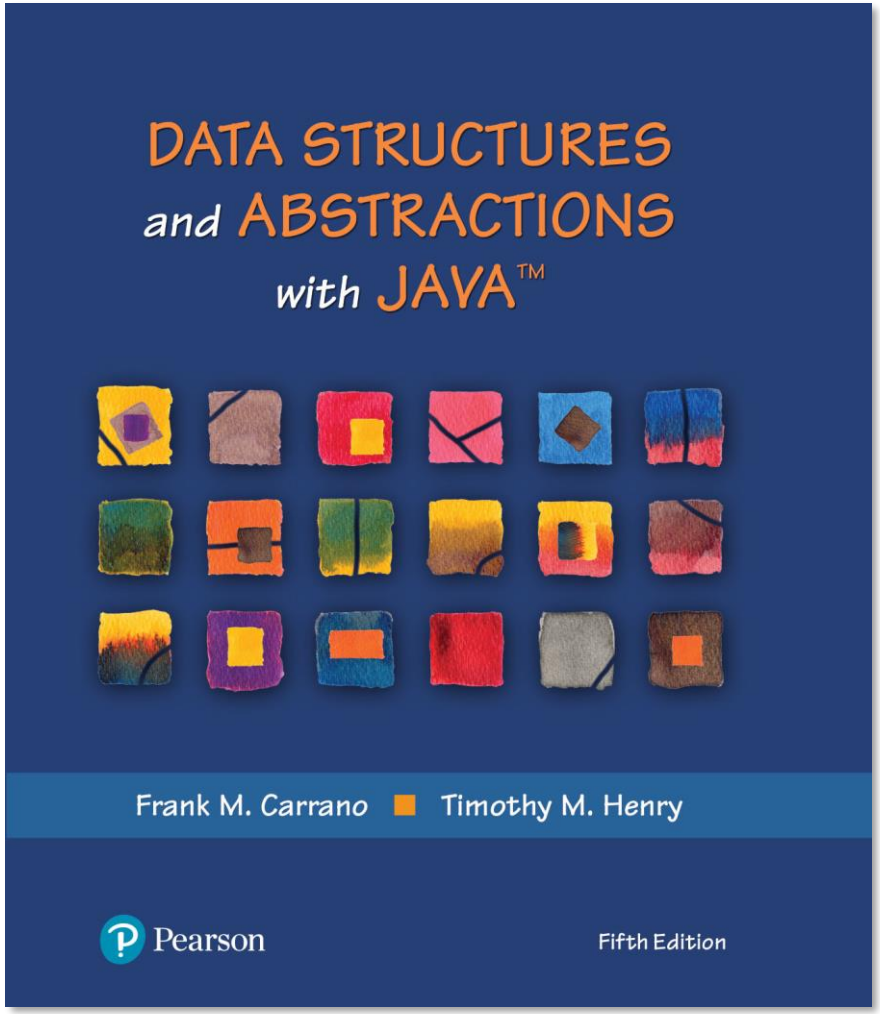
Data Structures and Abstractions with Java™

5th Edition



Java Interlude 4

Iterators

What Is an Iterator?

- An object that traverses a collection of data
- During iteration, each data item is considered once
  - Possible to modify item as accessed
- Should implement as a distinct class that interacts with the ADT

The Java Interface `Iterator`

```
package java.util;
public interface Iterator<T>
  {
    /** Detects whether this iterator has completed its traversal
     and gone beyond the last entry in the collection of data.
     @return True if the iterator has another entry to return. */
    public boolean hasNext();

    /** Retrieves the next entry in the collection and
     advances this iterator by one position.
     @return A reference to the next entry in the iteration,
     if one exists.
     @throws NoSuchElementException if the iterator had reached the
     end already, that is, if hasNext() is false. */
    public T next();

    /** Removes from the collection of data the last entry that
     next() returned. A subsequent call to next() will behave
     as it would have before the removal.
     Precondition: next() has been called, and remove() has not
     been called since then. The collection has not been altered
     during the iteration except by calls to this method.
     @throws IllegalStateException if next() has not been called, or
     if remove() was called already after the last call to next().
     @throws UnsupportedOperationException if the iterator does
     not permit a remove operation. */
    public void remove(); // Optional method
  } // end Iterator
```

The Java Interface `Iterator`

FIGURE J4-1 Possible positions of an iterator's cursor within a collection

The Java Interface Iterator

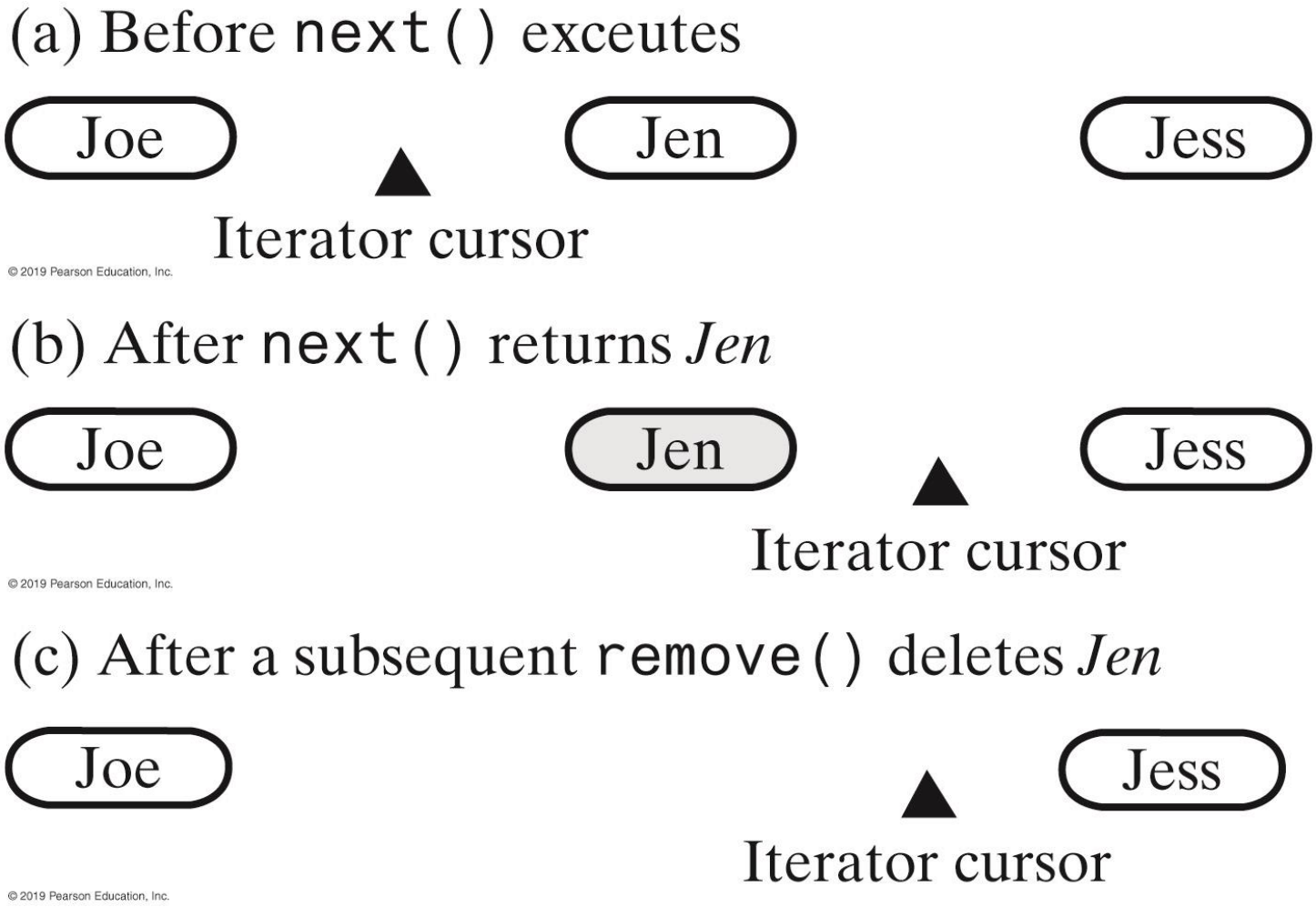


FIGURE J4-2 Effect on a collection's iterator by a call to `next` and subsequent call to `remove`

The Interface `Iterable`

```
package java.lang;
public interface Iterable<T>
   {
      /** @return An iterator for a collection of objects of type T. */
            Iterator<T> iterator();
   } // end Iterable
```

LISTING JI5-2 The interface `java.lang.Iterable`

Using the Java Interface Iterator

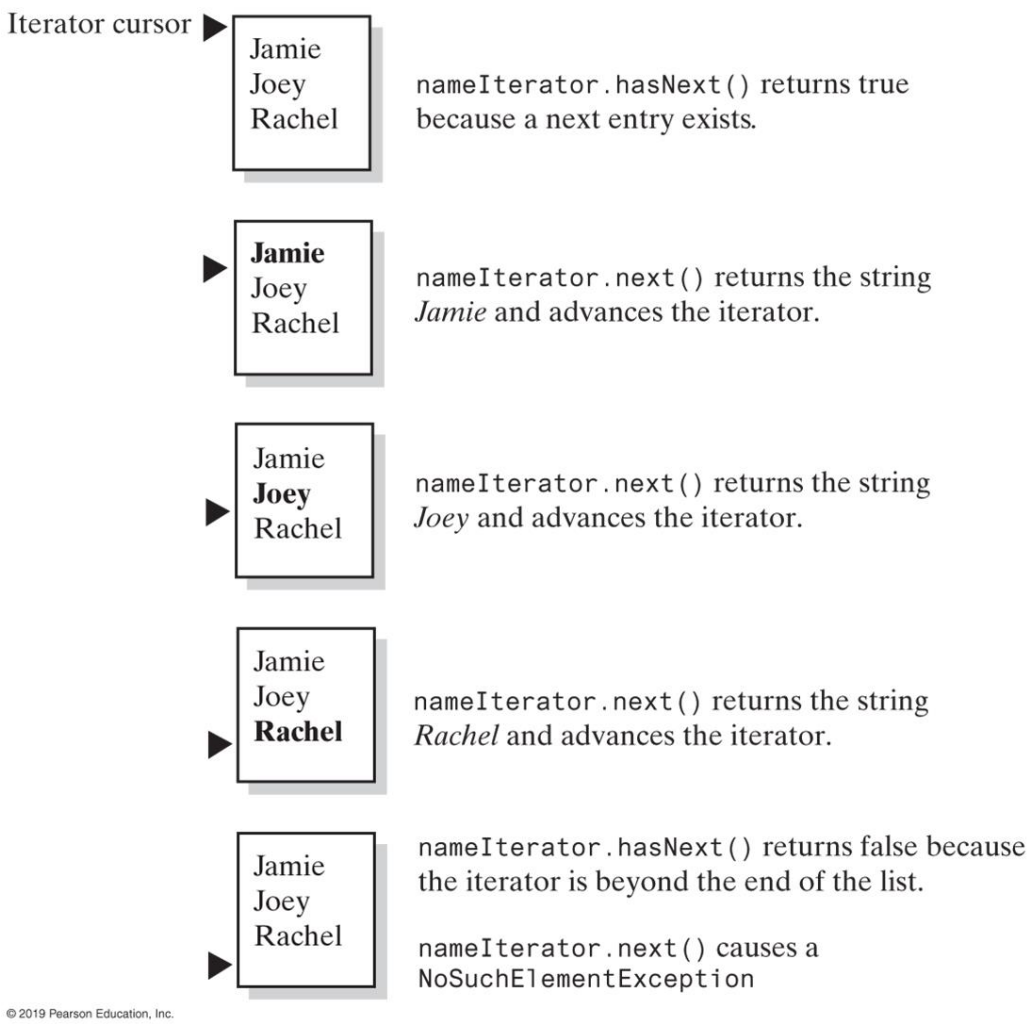


FIGURE J4-3 The effect of the iterator methods `hasNext` and `next` on a list

Using the Java Interface Iterator

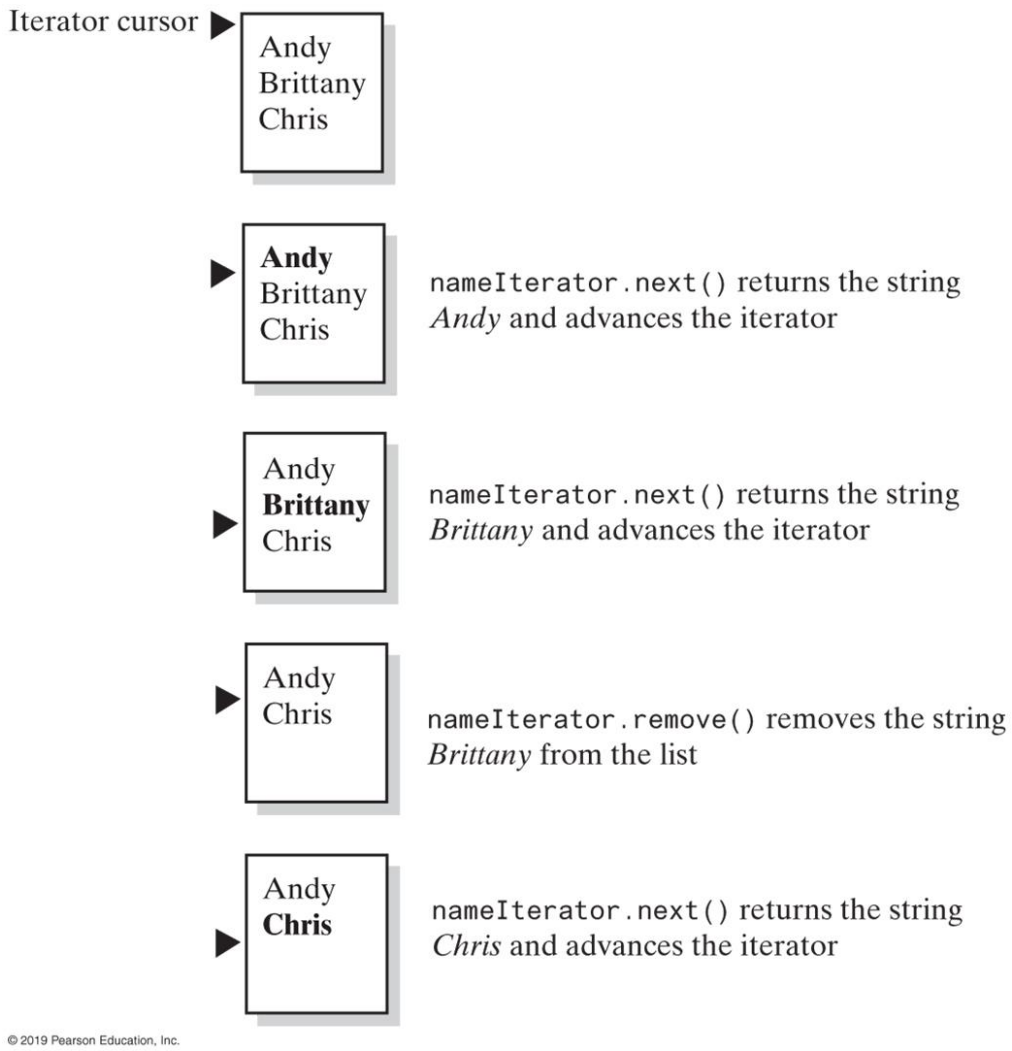


FIGURE J4-4 The effect of the iterator methods `next` and `remove` on a list

Using the Java Interface Iterator

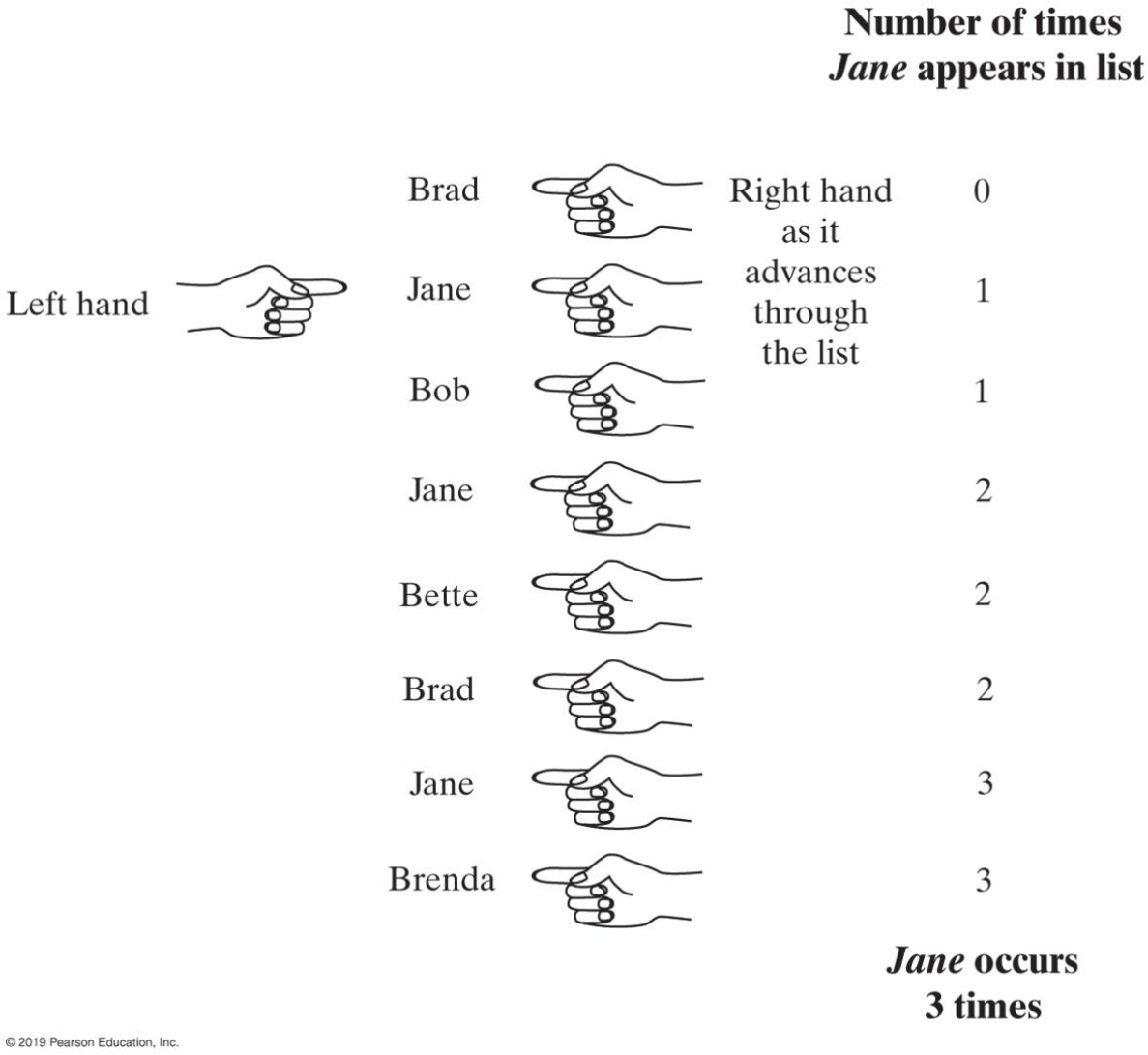


FIGURE J4-5 Counting the number of times that Jane appears in a list of names

Multiple Iterators

```
Iterator<String> nameIterator = namelist.iterator();
while (nameIterator.hasNext())
{
  String currentName = nameIterator.next();

  int nameCount = 0;

  Iterator<String> countingIterator = namelist.iterator();
  while (countingIterator.hasNext())
  {
    String nextName = countingIterator.next();
    if (currentName.equals(nextName))
      nameCount++;

  } // end while

  System.out.println(currentName + " occurs " + nameCount + " times.");
} // end while
```

Code that counts the occurrences of each name

LISTING JI4-3 Java's interface `java.util.ListIterator` (Part 1)

```
package java.util;
public interface ListIterator<T> extends Iterator<T>
  {
    /** Detects whether this iterator has gone beyond the last
     entry in the list.
     @return True if the iterator has another entry to return when
     traversing the list forward; otherwise returns false. */
    public boolean hasNext();

    /** Retrieves the next entry in the list and
     advances this iterator by one position.
     @return A reference to the next entry in the iteration,
     if one exists.
     @throws NoSuchElementException if the iterator is at the end,
     that is, if hasNext() is false. */
    public T next();

    /** Removes from the list the last entry that either next()
     or previous() has returned.
     Precondition: next() or previous() has been called, but the
     iterator's remove() or add() method has not been called
     since then. That is, you can call remove only once per
     call to next() or previous(). The list has not been altered
     during the iteration except by calls to the iterator's
     remove(), add(), or set() methods.
     @throws IllegalStateException if next() or previous() has not
     been called, or if remove() or add() has been called
     already after the last call to next() or previous().
     @throws UnsupportedOperationException if the iterator does not
     permit a remove operation. */
    public void remove(); // Optional method
```

These three methods are in the interface **`Iterator`**; they are duplicated here for reference and to show new behavior for **`remove`**.

LISTING JI4-3 Java's interface `java.util.ListIterator` (Part 2)

```
/** Detects whether this iterator has gone before the first
 entry in the list.
 @return True if the iterator has another entry to visit when
 traversing the list backward; otherwise returns false. */
public boolean hasPrevious();

/** Retrieves the previous entry in the list and moves this
 iterator back by one position.
 @return A reference to the previous entry in the iteration, if
 one exists.
 @throws NoSuchElementException if the iterator has no previous
 entry, that is, if hasPrevious() is false. */
public T previous();

/** Gets the index of the next entry.
 @return The index of the list entry that a subsequent call to
 next() would return. If next() would not return an entry
 because the iterator is at the end of the list, returns
 the size of the list. Note that the iterator numbers
 the list entries from 0 instead of 1. */
public int nextIndex();

/** Gets the index of the previous entry.
 @return The index of the list entry that a subsequent call to
 previous() would return. If previous() would not return
 an entry because the iterator is at the beginning of the
 list, returns –1. Note that the iterator numbers the
 list entries from 0 instead of 1. */
public int previousIndex();
```

LISTING JI4-3 Java's interface `java.util.ListIterator` (Part 3)

```
    /** Adds an entry to the list just before the entry, if any,
     that next() would have returned before the addition. This
     addition is just after the entry, if any, that previous()
     would have returned. After the addition, a call to
     previous() will return the new entry, but a call to next()
     will behave as it would have before the addition.
     Further, the addition increases by 1 the values that
     nextIndex() and previousIndex() will return.
     @param newEntry An object to be added to the list.
     @throws ClassCastException if the class of newEntry prevents the
     addition to the list.
     @throws IllegalArgumentException if some other aspect of
     newEntry prevents the addition to the list.
     @throws UnsupportedOperationException if the iterator does not
     permit an add operation. */
    public void add(T newEntry); // Optional method

    /** Replaces the last entry in the list that either next()
     or previous() has returned.
     Precondition: next() or previous() has been called, but the
     iterator's remove() or add() method has not been called since then.
     @param newEntry An object that is the replacement entry.
     @throws ClassCastException if the class of newEntry prevents the
     addition to the list.
     @throws IllegalArgumentException if some other aspect of newEntry
                                      prevents the addition to the list.
     @throws IllegalStateException if next() or previous() has not been called,
                                   or if remove() or add() has been called already
                                   after the last call to next() or previous().
     @throws UnsupportedOperationException if the iterator does not permit a set operation. */
    public void set(T newEntry); // Optional method
} // end ListIterator
```

Using the Java Interface `ListIterator`

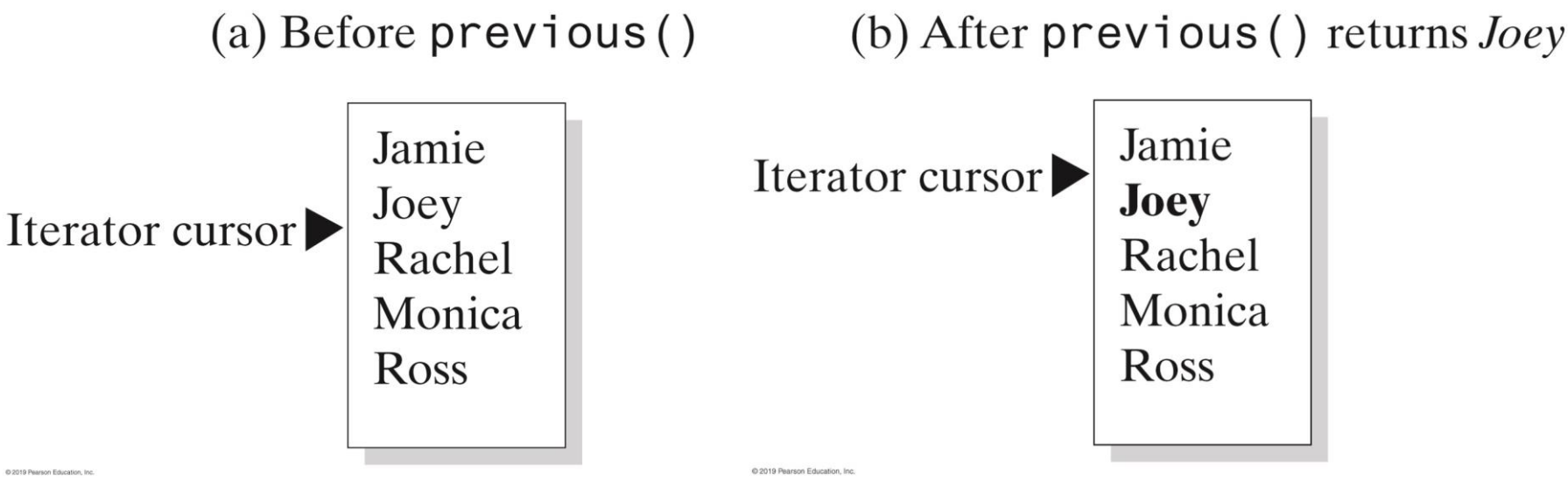


FIGURE J4-6 The effect of a call to previous on a list

Using the Java Interface `ListIterator`

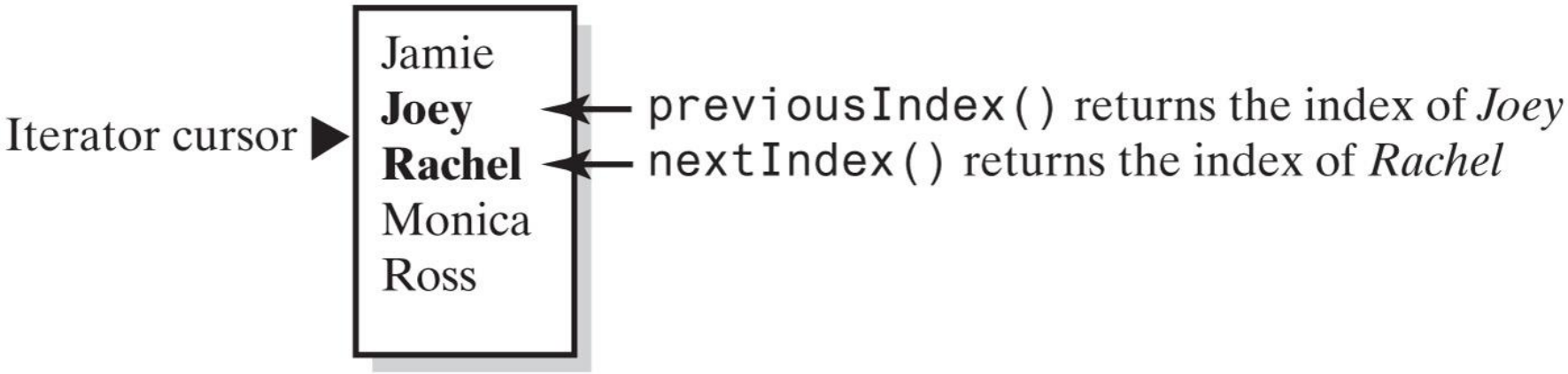


FIGURE J4-7 The indices returned by the methods `nextIndex` and `previousIndex`

The Interface `List` Revisited

- Method set replaces entry that either next or previous just returned.
- Method add inserts an entry into list just before iterator's current position
- Method remove removes list entry that last call to either next or previous returned

End

Java Interlude 4