Lecture 20: Iterators

CS 0445: Data Structures

Constantinos Costa

http://db.cs.pitt.edu/courses/cs0445/current.term/

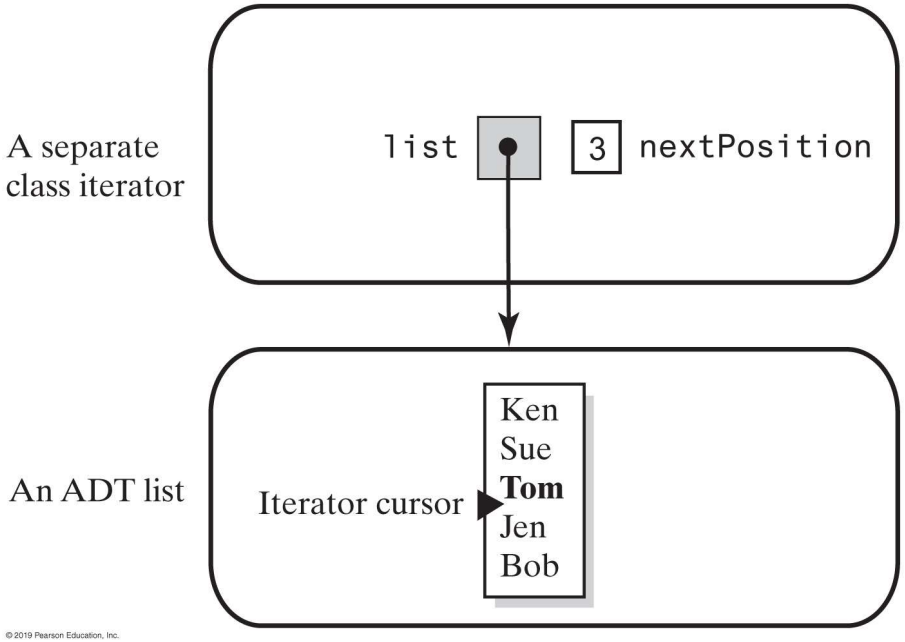
Oct 23, 2019, 8:00-9:15
University of Pittsburgh, Pittsburgh, PA

Iterators

- An iterator
  - An object that enables you to traverse entries in a data collection
- Possible way to provide an ADT with traversal operations
  - Define them as ADT operations
- Better way
  - Implement the iterator methods within their own class

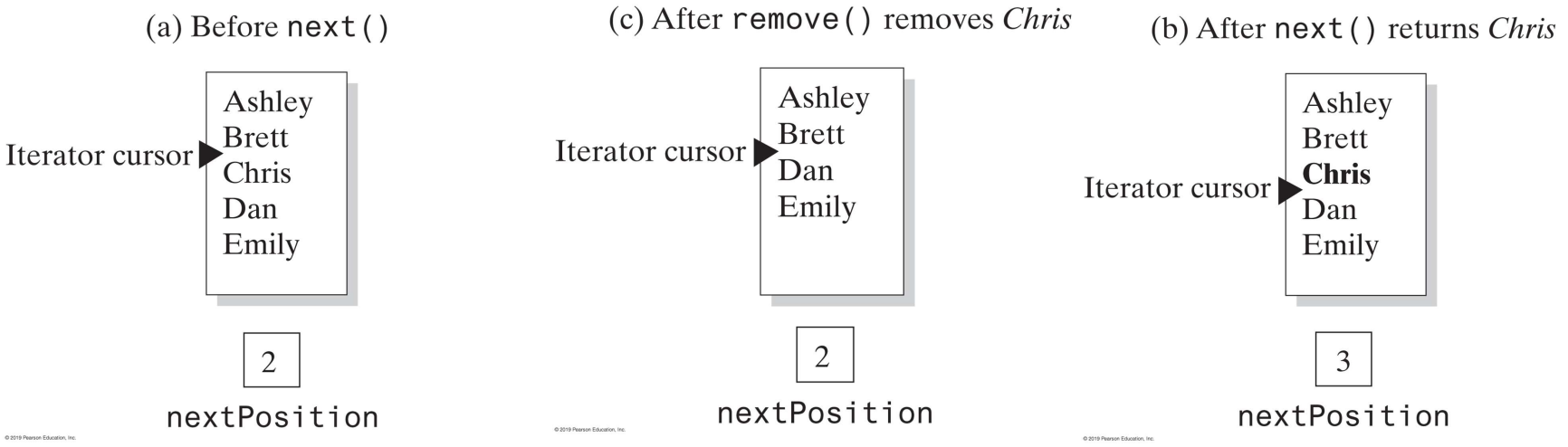
Separate Class Iterator

- A separate class iterator with a reference to an ADT, an indicator of its position within the iteration, and no knowledge of the ADT's implementation



Separate Class Iterator

- Changes to a list and `nextPosition` when removing Chris from the list



Separate Class Iterator

- An outline of the class SeparateIterator

```
/**  A class that represents an iterator for the ADT list. */
public class SeparateIterator<T> implements Iterator<T>
{
  private ListInterface<T> list;
  private int           nextPosition;  // Position of entry last returned by next()
  private boolean       wasNextCalled; // Needed by remove

  public SeparateIterator(ListInterface<T> myList)
  {
    list = myList;
    nextPosition = 0;
    wasNextCalled = false;
  } // end constructor

/* < Implementations of the methods hasNext, next, and remove go here. >
   . . . */
} // end SeparateIterator
```

Separate Class Iterator

- Implementation of `hasNext`

```
public boolean hasNext()
{
  return nextPosition < list.getLength();
} // end hasNext
```

Separate Class Iterator

- Implementation of `next`

```
public T next()
{
  if (hasNext())
  {
    wasNextCalled = true;
    nextPosition++;
    return list.getEntry(nextPosition);
  }
  else
    throw new NoSuchElementException("Illegal call to next(); " +
                                     "iterator is after end of list.");
} // end next
```

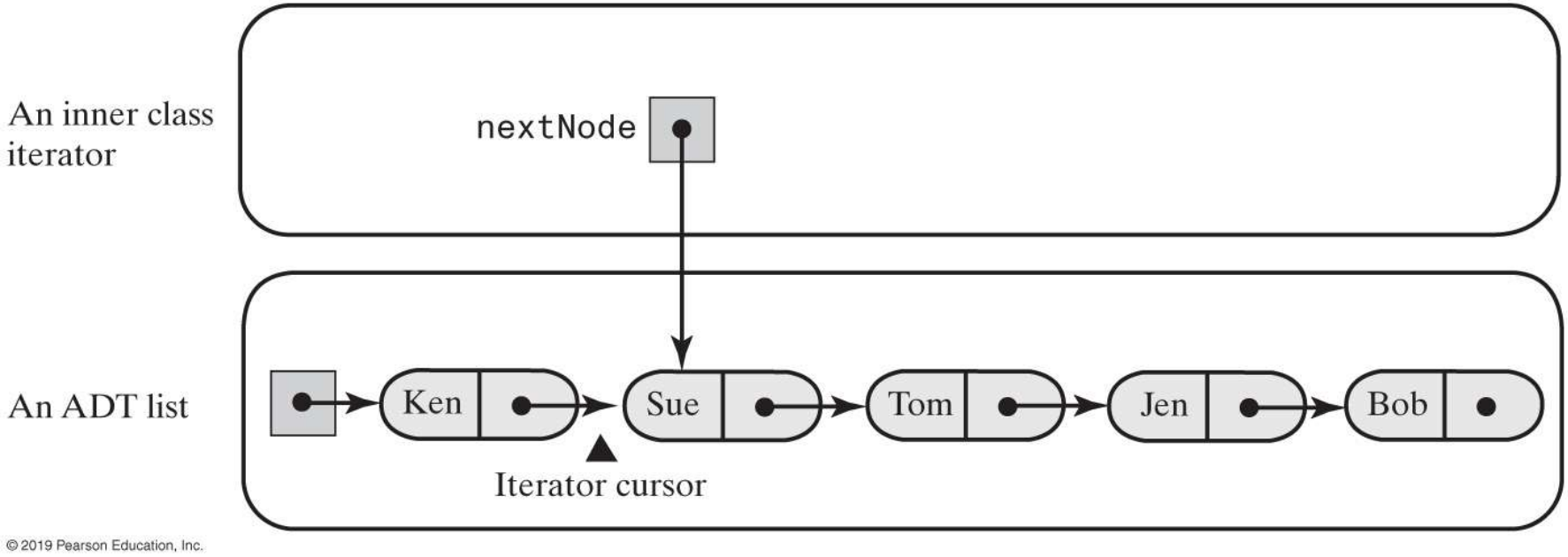
Separate Class Iterator

- Implementation of `remove`

```
public void remove()
{
  if (wasNextCalled)
  {
    // nextPosition was incremented by the call to next(), so
    // it is the position number of the entry to be removed
    list.remove(nextPosition);
    nextPosition--;      // A subsequent call to next() must be
                         // unaffected by this removal
    wasNextCalled = false; // Reset flag
  }
  else
    throw new IllegalStateException("Illegal call to remove(); " +
                                    "next() was not called.");
} // end remove
```

Inner Class Iterator

- An inner class iterator with direct access to the linked chain that implements the ADT



Inner Class Iterator

- The interface `ListWithIteratorInterface`

```
/** An interface for the ADT list that has an iterator.  */
public interface ListWithIteratorInterface<T>
                        extends ListInterface<T>, Iterable<T>
{
  public Iterator<T> getIterator();
} // end ListWithIteratorInterface
```

Inner Class Iterator (Part 1)

```
/**  A class that implements the ADT list by using a chain of linked nodes.
  The list has an iterator. The class is similar to LList. */
public class LinkedListWithIterator<T> implements ListWithIteratorInterface<T>
{
  private Node firstNode;
  private int  numberOfEntries;;

  public LinkedListWithIterator()
  {
    initializeDataFields();
  } // end default constructor

/*   < Implementations of the methods of the ADT list go here;
    you can see them in Chapter 12, beginning at Segment 12.7 >
  . . . */

  public Iterator<T> iterator()
  {
        return new IteratorForLinkedList();
  } // end iterator

     public Iterator<T> getIterator()
     {
       return iterator();
     } // end getIterator
```

Inner Class Iterator (Part 2)

- An outline of the class LinkedListWithIterator

```
 private class IteratorForLinkedList implements Iterator<T>
 {
private Node nextNode;

 private IteratorForLinkedList()
 {
       nextNode = firstNode;
 } // end default constructor

// Implementations of the methods in the interface Iterator go here.

 } // end IteratorForLinkedList

 private class Node
 {
// Implementations of the methods in inner class Node go here.
 } // end Node

} // end LinkedListWithIterator
```

Inner Class Iterator

- The method `next`.

```
public T next()
{
  T result;
  if (hasNext())
  {
    result = nextNode.getData();
    nextNode = nextNode.getNextNode(); // Advance iterator
  }
  else
    throw new NoSuchElementException("Illegal call to next(); " +
                                     "iterator is after end of list.");
  return result; // Return next entry in iteration
} // end next
```

Inner Class Iterator

- The method `hasNext`

```
public boolean hasNext()
{
  return nextNode != null;
} // end hasNext
```

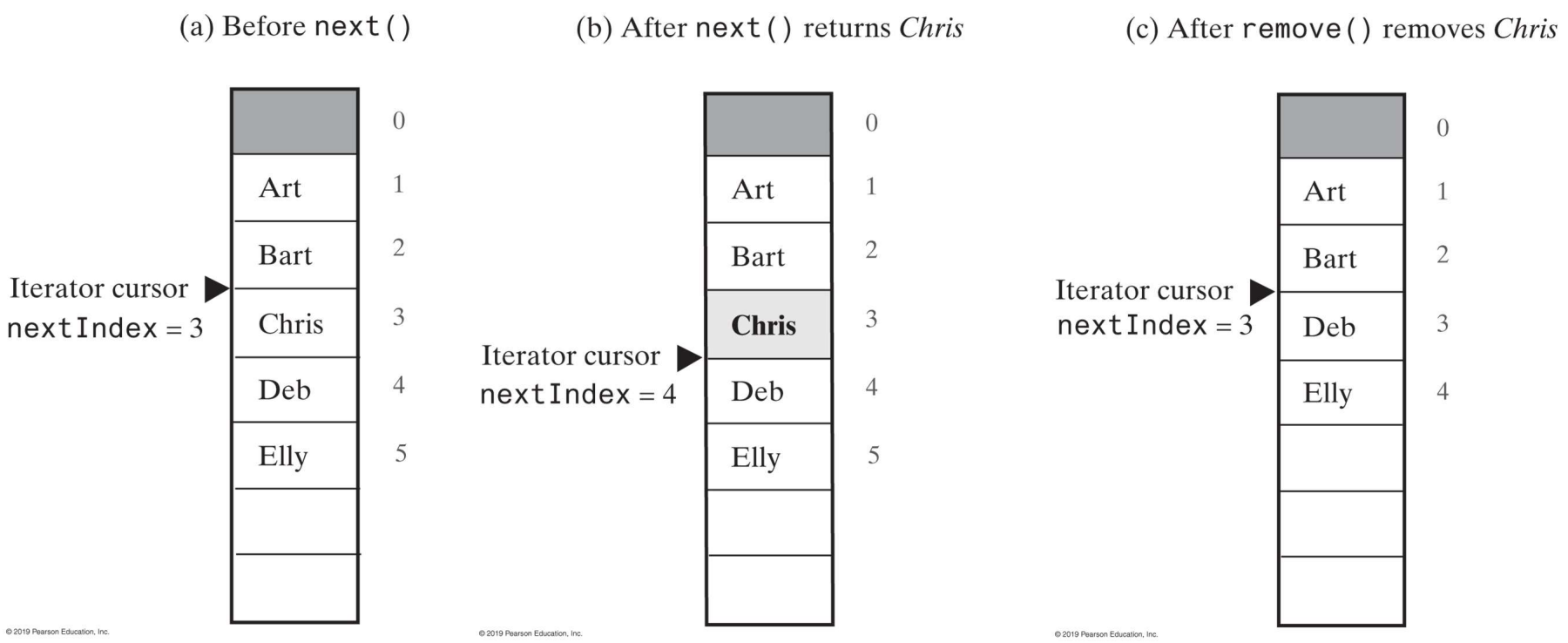
Inner Class Iterator

- The method `remove`.

```
public void remove()
{
  throw new UnsupportedOperationException("remove() is not supported " +
                                          "by this iterator");
} // end remove
```

Iterators for Array-Based Lists

- Changes to the array of list entries and `nextIndex` when removing Chris from the list



Iterators for Array-Based Lists (Part 1)

- An outline of the class `ArrayListWithIterator`

```
/**  A class that implements the ADT list by using a resizable array and
  gives it an iterator. */
public class ArrayListWithIterator<T> implements ListWithIteratorInterface<T>
{
      private T[] list; // Array of list entries; ignore list[0]
      private int numberOfEntries;
   private boolean integrityOK = false;
      private static final int DEFAULT_CAPACITY = 25;
      private static final int MAX_CAPACITY = 10000;

      public ArrayListWithIterator()
      {
      this(DEFAULT_CAPACITY);
      } // end default constructor
```

Iterators for Array-Based Lists (Part 2)

- An outline of the class `ArrayListWithIterator`

```
  public ArrayListWithIterator(int initialCapacity)
  {
 integrityOK = false;

 // Is initialCapacity too small?
 if (initialCapacity < DEFAULT_CAPACITY)
   initialCapacity = DEFAULT_CAPACITY;
 else // Is initialCapacity too big?
   checkCapacity(initialCapacity);

 // The cast is safe because the new array contains null entries
 @SuppressWarnings("unchecked")
 T[] tempList = (T[])new Object[initialCapacity + 1];
 list = tempList;
 numberOfEntries = 0;
 integrityOK = true;
  } // end constructor

/* < Implementations of the methods of the ADT list go here;
   you can see them in Chapter 11, beginning at Segment 11.5. */
```

Iterators for Array-Based Lists (Part 3)

- An outline of the class `ArrayListWithIterator`

```
public Iterator<T> iterator()
{
  return new IteratorForArrayList();
} // end iterator

public Iterator<T> getIterator()
{
  return iterator();
} // end getIterator

private class IteratorForArrayList implements Iterator<T>
{
  private int     nextIndex;     // Index of next entry in the iteration
  private boolean wasNextCalled; // Needed by remove

  private IteratorForArrayList()
  {
    nextIndex = 1;       // Iteration begins at list's first entry
    wasNextCalled = false;
  } // end default constructor
  // Implementations of the methods in the interface Iterator go here.
} // end IteratorForArrayList
} // end ArrayListWithIterator
```

Iterators for Array-Based Lists

- Method `hasNext`

```
public boolean hasNext()
{
  return nextIndex <= numberOfEntries;
} // end hasNext
```

Iterators for Array-Based Lists

- Method `next`

```
public T next()
{
  checkIntegrity();
  if (hasNext())
  {
    wasNextCalled = true;
    T nextEntry = list[nextIndex];
    nextIndex++; // Advance iterator
    return nextEntry;
  }
  else
    throw new NoSuchElementException("Illegal call to next(); " +
                                     "iterator is after end of list.");
} // end next
```

Iterators for Array-Based Lists

- Implementation of `remove`

```
public void remove()
{
  checkIntegrity();
  if (wasNextCalled)
  {
    // nextIndex was incremented by the call to next, so it is
    // 1 larger than the position number of the entry to be removed
    ArrayListWithIterator.this.remove(nextIndex – 1);
    nextIndex––;            // Index of next entry in iteration
    wasNextCalled = false;    // Reset flag
  }
  else
    throw new IllegalStateException("Illegal call to remove(); " +
                                    "next() was not called.");
} // end remove
```

Why Are Iterator Methods in Their Own Class? (Part 1)

- An outline of the class `ListWithTraversal`

```
/**  A linked implementation of the ADT list that
  includes iterator operations as ADT operations. */
public class ListWithTraversal<T> implements ListInterface<T>, Iterator<T>
{
      private Node firstNode;
      private int  numberOfEntries;
      private Node nextNode; // Node containing next entry in iteration

      public ListWithTraversal()
      {
      initializeDataFields();
      } // end default constructor

/* < Implementations of the remaining methods of the ADT list go here . */

  // Initializes the class's data fields to indicate an empty list.
  private void initializeDataFields()
  {
      firstNode = null;
      numberOfEntries = 0;
      nextNode = null;
  } // end initializeDataFields
```

Why Are Iterator Methods in Their Own Class? (Part 2)

- An outline of the class `ListWithTraversal`

```
// Methods in the interface Iterator go here

    /** Sets the traversal to the beginning of the list.
        This method is NOT in the interface Iterator. */
  public void resetTraversal()
  {
    nextNode = firstNode;
  } // end resetTraversal

    private class Node
    {
   // Methods in the inner Node class go here:
    } // end Node
} // end ListWithTraversal
```

Why Are Iterator Methods in Their Own Class?

- These traversal methods can execute quickly
  - They have direct access to the underlying data structure
- Disadvantages
  - Only one traversal at a time
  - Operation such as resetTraversal necessary – “interface bloat”

Array-Based Implementation of the Interface `ListIterator`

- The interface `ListWithListIteratorInterface`

```
/**  An interface for the ADT list that has an iterator implementing
  the interface ListIterator.  */
public interface ListWithListIteratorInterface<T>
                                  extends Iterable<T>, ListInterface<T>
{
  public ListIterator<T> getIterator();
} // end ListWithListIteratorInterface
```

Array-Based Implementation of the Interface `ListIterator` (Part 1)

- An outline of the class `ArrayListWithListIterator`

```
/** A class that implements the ADT list by using an array.
  The list has entries that are numbered beginning at 1.
  The list has an iterator that implements the interface ListIterator.
  Iterator positions (indexes) are numbered beginning at 0. */
public class ArrayListWithListIterator<T>
        implements ListWithListIteratorInterface<T>
{
  private T[] list;  // Array of list entries; ignore list[0]
  private int numberOfEntries;
  private boolean integrityOK;
  private static final int DEFAULT_CAPACITY = 25;
     private static final int MAX_CAPACITY = 10000;

  public ArrayListWithListIterator()
  {
    this(DEFAULT_CAPACITY);
  } // end default constructor
```

Array-Based Implementation of the Interface `ListIterator` (Part 2)

- An outline of the class `ArrayListWithListIterator`

```
public ArrayListWithListIterator(int initialCapacity)
{
  integrityOK = false;

  // Is initialCapacity too small?
  if (initialCapacity < DEFAULT_CAPACITY)
    initialCapacity = DEFAULT_CAPACITY;
  else // Is initialCapacity too big?
    checkCapacity(initialCapacity);

  // The cast is safe because the new array contains null entries
  @SuppressWarnings("unchecked")
  T[] tempList = (T[])new Object[initialCapacity + 1];
  list = tempList;
  numberOfEntries = 0;
  integrityOK = true;
} // end constructor

/* < Implementations of the methods of the ADT list go here; */
```

Array-Based Implementation of the Interface `ListIterator` (Part 3)

- LISTING 13-7 An outline of the class `ArrayListWithListIterator`

```
public ListIterator<T> getIterator()
{
  return new ListIteratorForArrayList();
} // end getIterator

public Iterator<T> iterator()
{
  return getIterator();
} // end iterator

private class ListIteratorForArrayList implements ListIterator<T>
{
  // The details of this class begin with Segment 13.24.
} // end ListIteratorForArrayList
} // end ArrayListWithListIterator
```

Inner Class Iterator for Array-Based Lists (Part 1)

- Possible contexts in which the method `remove` of the iterator traversal throws an exception when called

(a)
```
traverse.remove();
```
← Causes an exception; neither `next` nor `previous` has been called

(b)
```
traverse.next();
traverse.remove();
```
← Legal

```
traverse.remove();
```
← Causes an exception

(c)
```
traverse.previous();
traverse.remove();
```
← Legal

```
traverse.remove();
```
← Causes an exception

Inner Class Iterator for Array-Based Lists (Part 2)

- Possible contexts in which the method `remove` of the iterator traversal

(d)
```
traverse.next();
traverse.add(...);
traverse.remove();
```
← Causes an exception

(e)
```
traverse.previous();
traverse.add(...);
traverse.remove();
```
← Causes an exception

The Inner Class

- Beginning of the inner class.

```
private enum Move {NEXT, PREVIOUS}

private class ListIteratorForArrayList implements ListIterator<T>
{
  private int nextIndex; // Index of next entry in the iteration
  private boolean isRemoveOrSetLegal;
  private Move lastMove;

  private ListIteratorForArrayList()
  {
    nextIndex = 1; // Iteration begins at list's first entry
    isRemoveOrSetLegal = false;
    lastMove = null;
  } // end default constructor
```

The Inner Class

- Method `hasNext`.

```
public boolean hasNext()
{
  return nextIndex <= numberOfEntries;
} // end hasNext
```

The Inner Class

- Implementation of `next`.

```
public T next()
{
  if (hasNext())
  {
    lastMove = Move.NEXT;
    isRemoveOrSetLegal = true;

    T nextEntry = list[nextIndex];
    nextIndex++; // Advance iterator

    return nextEntry;
  }
  else
    throw new NoSuchElementException("Illegal call to next(); " +
                                     "iterator is after end of list.");
} // end next
```

The Inner Class

- Methods `hasPrevious` and `previous`.

```
public boolean hasPrevious()
{
  return (nextIndex > 1) && (nextIndex <= numberOfEntries + 1);
} // end hasPrevious

public T previous()
{
  if (hasPrevious())
  {
    lastMove = Move.PREVIOUS;
    isRemoveOrSetLegal = true;

    T previousEntry = list[nextIndex - 1];
    nextIndex--; // Move iterator back
    return previousEntry;
  }
  else
    throw new NoSuchElementException("Illegal call to previous(); " +
                                     "iterator is before beginning of list.");
} // end previous
```

The Inner Class

- Methods `nextIndex` and `previousIndex`

```
public int nextIndex()
{
  int result;

  if (hasNext())
    result = nextIndex - 1;   // Change to zero-based numbering of iterator
  else
    result = numberOfEntries; // End-of-list flag

  return result;
} // end nextIndex

public int previousIndex()
{
  int result;

  if (hasPrevious())
    result = nextIndex - 2; // Change to zero-based numbering of iterator
  else
    result = -1;          // Beginning-of-list flag

  return result;
} // end previousIndex
```

The Inner Class

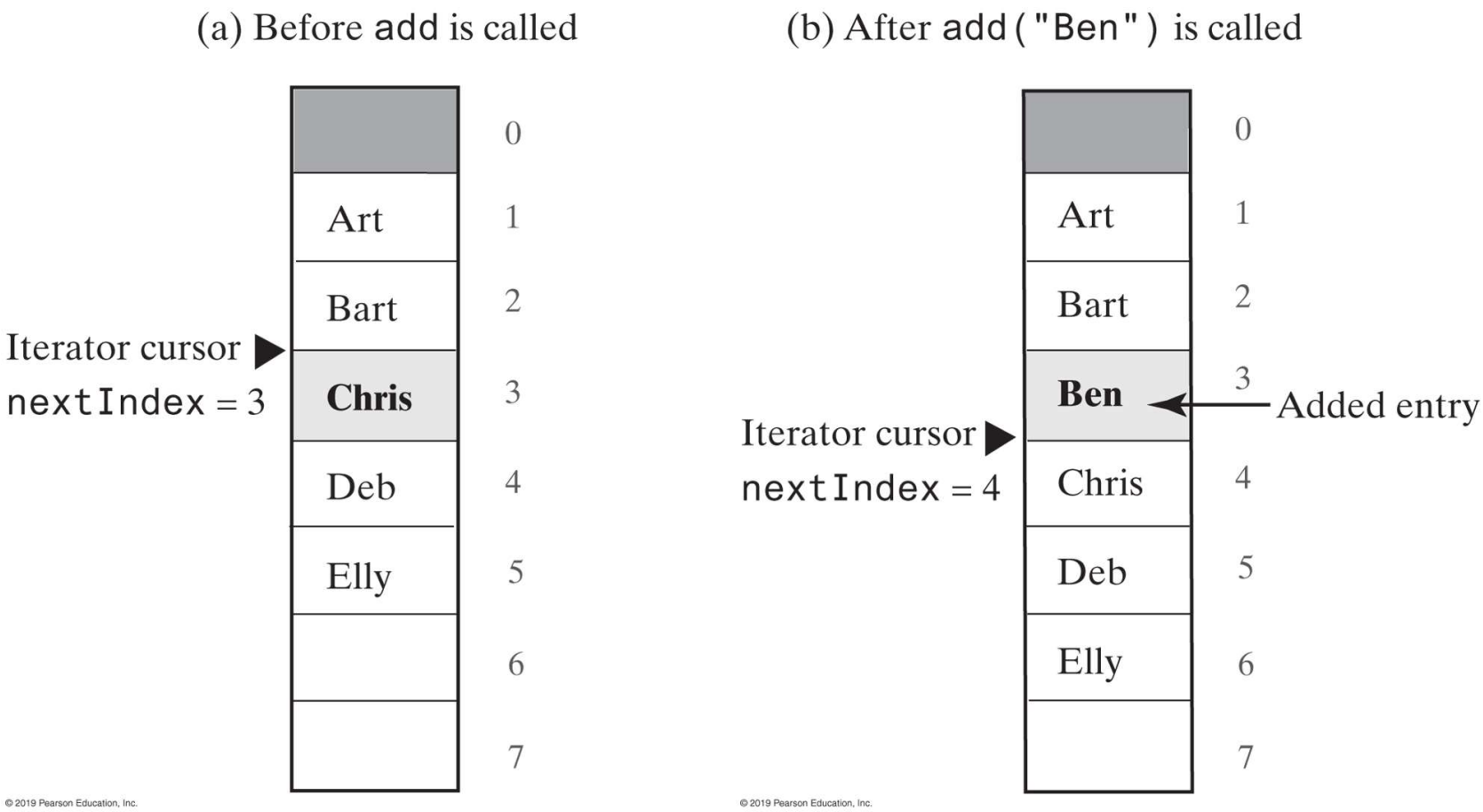
- The method `add`.

```
public void add(T newEntry)
{
  isRemoveOrSetLegal = false;

  // Insert newEntry immediately before the the iterator's current position
  ArrayListWithListIterator.this.add(nextIndex, newEntry);
  nextIndex++;
} // end add
```

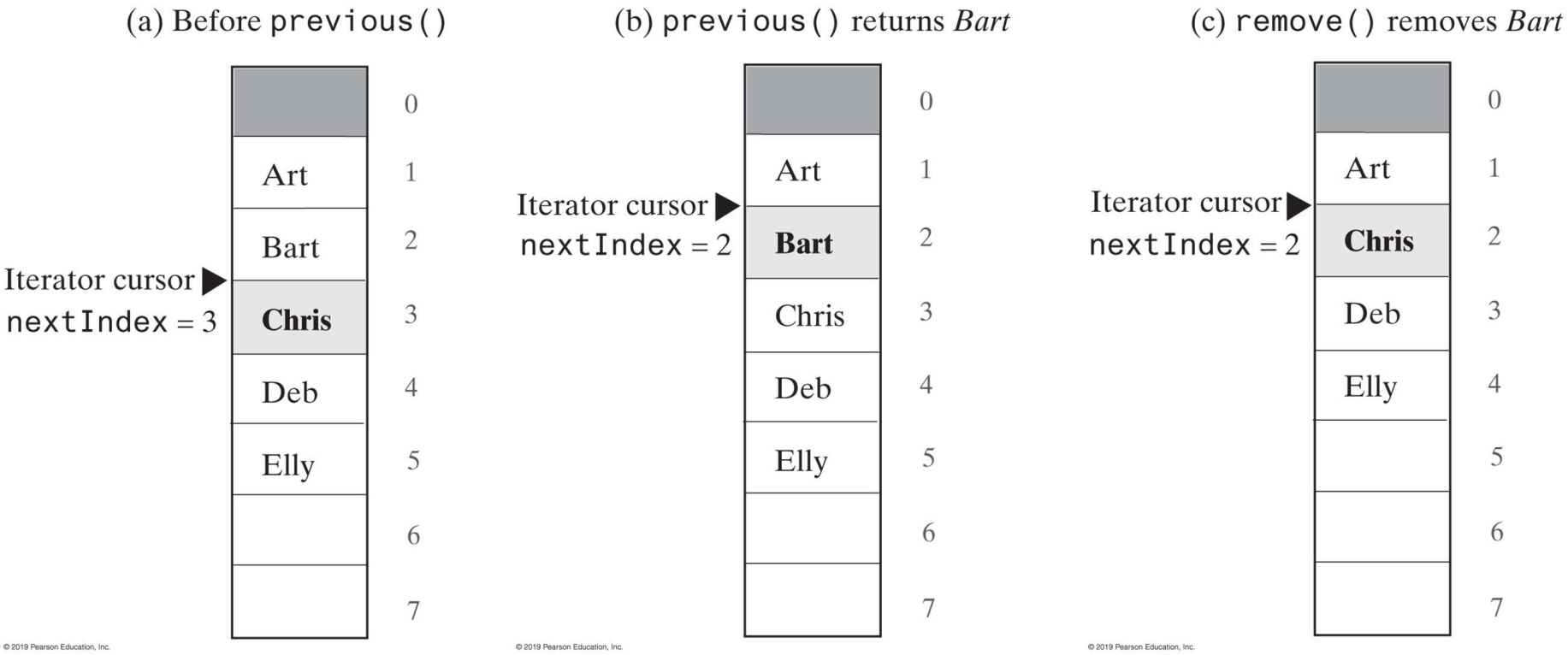
The Inner Class

- Changes to the array of list entries and `nextIndex` when adding Ben to the list



The Inner Class

- Changes to the array of list entries and `nextIndex` when removing Chris from the list



The Inner Class

- An implementation of `remove`.

```
public void remove()
{
  if (isRemoveOrSetLegal)
  {
    isRemoveOrSetLegal = false;

    if (lastMove.equals(Move.NEXT))
    {
      // next() called, but neither add() nor remove() has been called since.
      // Remove entry last returned by next().
      // nextIndex is 1 more than the index of the entry
      // returned by next()
      ArrayListWithListIterator.this.remove(nextIndex - 1);
      nextIndex--; // Move iterator back
    }
    else
    {
      // previous() called, but neither add() nor remove() has been called since
      // Remove entry last returned by previous().
      // nextIndex is the index of the entry returned by previous().
      ArrayListWithListIterator.this.remove(nextIndex);
    } // end if
  }
  else
    throw new IllegalStateException("Illegal call to remove(); " + "next() or previous() not called, OR " +
                                    "add() or remove() called since then.");
} // end remove
```

The Inner Class

- An implementation of method `set`.

```
public void set(T newEntry)
{
  if (isRemoveOrSetLegal)
  {
    if (lastMove.equals(Move.NEXT))
      list[nextIndex - 1] = newEntry; // Replace entry last returned by next()
    else
    {
      // Assertion: lastMove.equals(Move.PREVIOUS)
      list[nextIndex] = newEntry; // Replace entry last returned by previous()
    } // end if
  }
  else
    throw new IllegalStateException("Illegal call to set(); " +
                                    "next() or previous() not called, OR " +
                                    "add() or remove() called since then.");
} // end set
```