CS2400 - Data Structures and Advanced Programming
Module 5: Lists

Hao Ji
Computer Science Department
Cal Poly Pomona

Lists

- A way to organize data
    - A collection of objects in a specific order and having the same data type

- Examples
    - To-do list
    - Gift lists
    - Grocery Lists

- Items in list have position and may be added anywhere

ADT Bag vs ADT List

<<interface>>
Bag

+getCurrentSize(): integer
+isEmpty(): boolean
+add(newEntry: T): boolean
+remove(): T
+remove(anEntry: T): boolean
+clear(): void
+getFrequencyOf(anEntry: T): integer
+contains(anEntry: T): boolean
+toArray(): T[]

ADT Bag vs ADT List

<<interface>>
Bag

+getCurrentSize(): integer
+isEmpty(): boolean
+add(newEntry: T): boolean
+remove(): T
+remove(anEntry: T): boolean
+clear(): void
+getFrequencyOf(anEntry: T): integer
+contains(anEntry: T): boolean
+toArray(): T[]

<<interface>>
List

+getLength(): integer
+isEmpty(): boolean
+add(newEntry: T): void
+add(newPosition, newEntry): void
+remove(givenPosition): T
+replace(givenPosition, newEntry): T
+clear(): void
+getEntry(givenPosition): T
+contains(anEntry: T): boolean
+toArray(): T[]

ADT Bag vs ADT List

<<interface>>
Bag

+getCurrentSize(): integer
+isEmpty(): boolean
+add(newEntry: T): boolean
+remove(): T
+remove(anEntry: T): boolean
+clear(): void
+getFrequencyOf(anEntry: T): integer
+contains(anEntry: T): boolean
+toArray(): T[]

<<interface>>
List

+getLength(): integer
+isEmpty(): boolean
+add(newEntry: T): void
+add(newPosition, newEntry): void # Adds *newEntry* at position *newPosition* within the list.
+remove(givenPosition): T # Removes and returns the entry at position *givenPosition*.
+replace(givenPosition, newEntry): T # Replaces the entry at position *givenPosition* with *newEntry*.
+clear(): void
+getEntry(givenPosition): T # Retrieves the entry at position *givenPosition*
+contains(anEntry: T): boolean
+toArray(): T[]

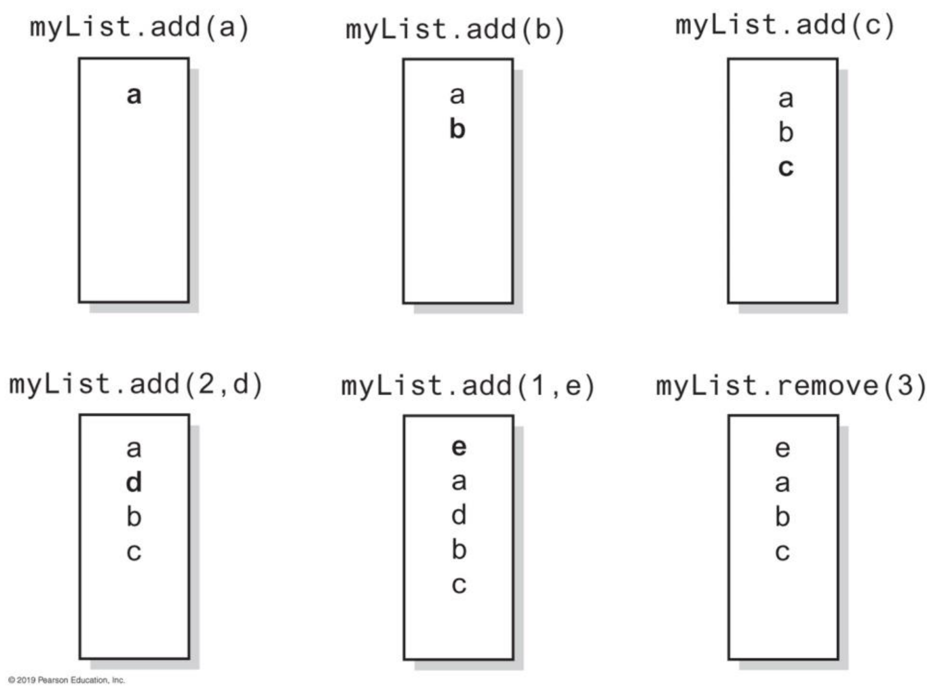
ADT Bag vs ADT List

<<interface>>
Bag

+getCurrentSize(): integer
+isEmpty(): boolean
+add(newEntry: T): boolean
+remove(): T
+remove(anEntry: T): boolean
+clear(): void
+getFrequencyOf(anEntry: T): integer
+contains(anEntry: T): boolean
+toArray(): T[]

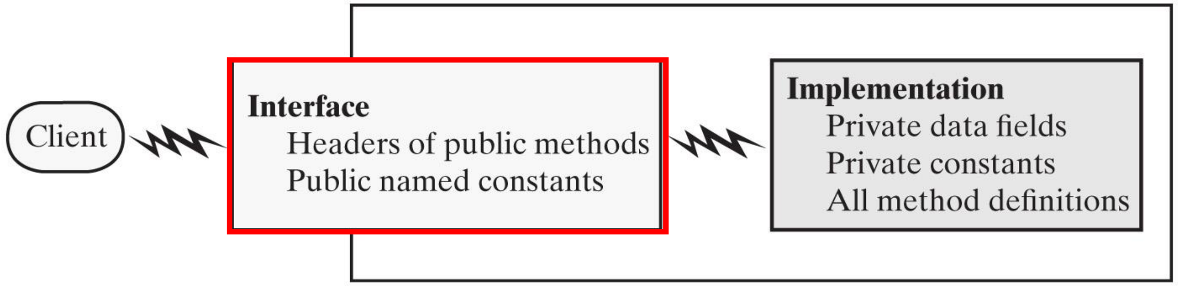
<<interface>>
List

+getLength(): integer
+isEmpty(): boolean
+add(newEntry: T): void
+add(newPosition, newEntry): void
+remove(givenPosition): T
+replace(givenPosition, newEntry): T
+clear(): void
+getEntry(givenPosition): T
+contains(anEntry: T): boolean
+toArray(): T[]



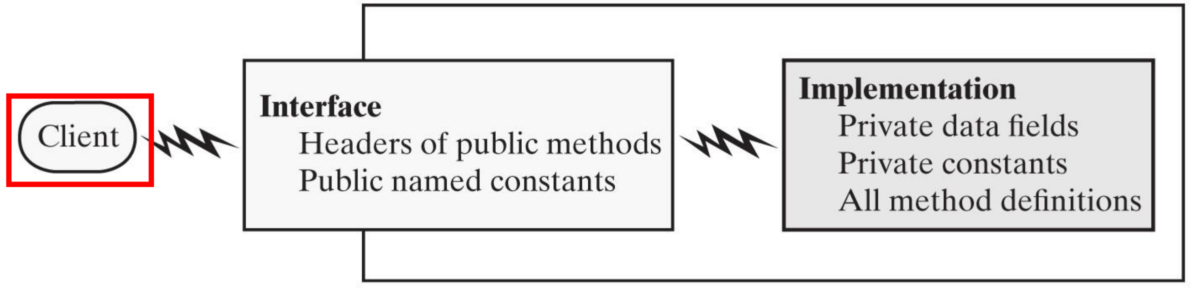
*In this course, **position 1 indicates the first entry in the list**.*

```java
** An interface ADT list. Entries in a list have positions that begin with 1.  */
public interface ListInterface<T>
{
 /** Adds a new entry to the end of this list. Entries currently in the list are unaffected. The list's size is
increased by 1.
     @param newEntry  The object to be added as a new entry. */
   public void add(T newEntry);

   /** Adds a new entry at a specified position within this list. Entries originally at and above the
specified position are at the next higher position within the list. The list's size is increased by 1.
      @param newPosition  An integer that specifies the desired position of the new entry.
      @param newEntry     The object to be added as a new entry.
      @throws  IndexOutOfBoundsException if either
             newPosition < 1 or newPosition > getLength() + 1. */
   public void add(int newPosition, T newEntry);

   /** Removes the entry at a given position from this list. Entries originally at positions higher than
the given position are at the next lower position within the list, and the list's size is decreased by 1.
      @param givenPosition  An integer that indicates the position of the entry to be removed.
      @return  A reference to the removed entry.
      @throws  IndexOutOfBoundsException if either
             givenPosition < 1 or givenPosition > getLength(). */
   public T remove(int givenPosition);

   /** Removes all entries from this list. */
   public void clear();

   /** Replaces the entry at a given position in this list.
      @param givenPosition  An integer that indicates the position of the entry to be replaced.
      @param newEntry  The object that will replace the entry at the position givenPosition.
      @return  The original entry that was replaced.
      @throws  IndexOutOfBoundsException if either
             givenPosition < 1 or givenPosition > getLength(). */
   public T replace(int givenPosition, T newEntry);
```



```java
   /** Retrieves the entry at a given position in this list.
      @param givenPosition  An integer that indicates the position of the desired entry.
      @return  A reference to the indicated entry.
      @throws  IndexOutOfBoundsException if either givenPosition < 1 or givenPosition
> getLength(). */
   public T getEntry(int givenPosition);

   /** Retrieves all entries that are in this list in the order in which they occur in the list.
      @return  A newly allocated array of all the entries in the list. If the list is empty, the
returned array is empty. */
   public T[] toArray();

   /** Sees whether this list contains a given entry.
      @param anEntry  The object that is the desired entry.
      @return  True if the list contains anEntry, or false if not. */
   public boolean contains(T anEntry);

   /** Gets the length of this list.
      @return  The integer number of entries currently in the list. */
   public int getLength();

   /** Sees whether this list is empty.
      @return  True if the list is empty, or false if not. */
   public boolean isEmpty();
} // end ListInterface
```

```
/**   A driver that uses a list to track the
  runners in a race as they cross the finish line. */
public class RoadRace {
      public static void main(String[] args)  {
      recordWinners();
      }  // end main

      public static void recordWinners() {
    ListInterface<String> runnerList = new AList<>();
 //  runnerList has only methods in ListInterface

    runnerList.add("16"); // Winner
    runnerList.add(" 4"); // Second place
    runnerList.add("33"); // Third place
    runnerList.add("27"); // Fourth place
    displayList(runnerList);
  } // end recordWinners

  public static void displayList(ListInterface<String> list) {
    int numberOfEntries = list.getLength();
    System.out.println("The list contains " + numberOfEntries +
                    " entries, as follows:");
    for (int position = 1; position <= numberOfEntries; position++)
      System.out.println(list.getEntry(position) +
                     " is entry " + position);
    System.out.println();
  } // end displayList
} // end RoadRace
```



```
The list contains 4 entries, as follows:
16 is entry 1
 4 is entry 2
33 is entry 3
27 is entry 4
```

Java Class Library: The Interface List

- The standard package **java.util** contains an interface **List** for an ADT list that is similar to the list that the interface describes.

```
public void add(int index, T newEntry)
public T remove(int index)
public void clear()
public T set(int index, T anEntry) // Like replace
public  T get(int index) // Like getEntry
public boolean contains(Object anEntry)
public int size() // Like getLength
public boolean isEmpty()
```

Java Class Library: The Class ArrayList

- The Java Class Library contains an implementation of the ADT list that uses a resizable array.
- This class, called ***ArrayList***, implements the interface *java.util.List*, which is in the package *java.util.*

```
public ArrayList()

public ArrayList(int initialCapacity)
```

Java Class Library: The Class Vector

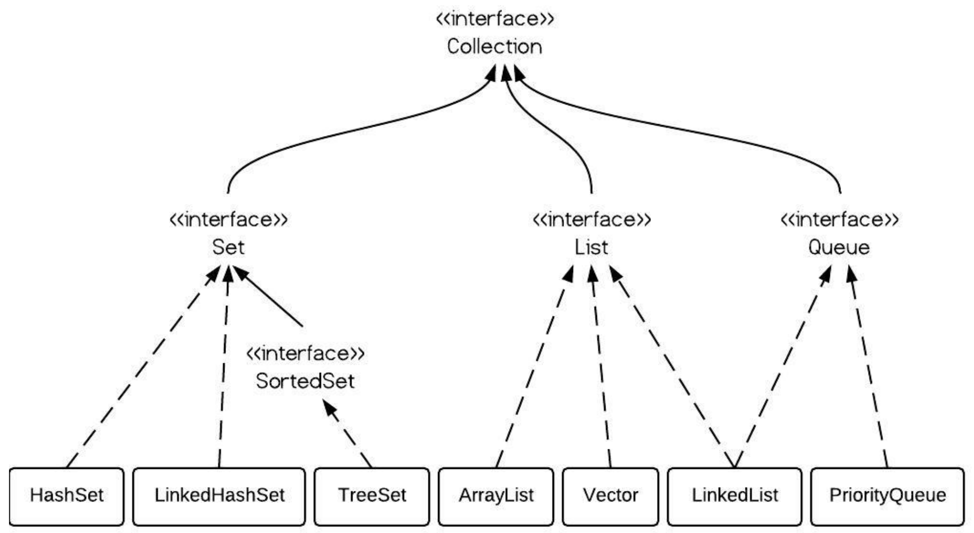
- In the Java Class Library ***java.util.Vector**,* is similar to ArrayList and implements the same interface *java.util.List.*

- You can use either **ArrayList** or **Vector** as an implementation of the interface List.

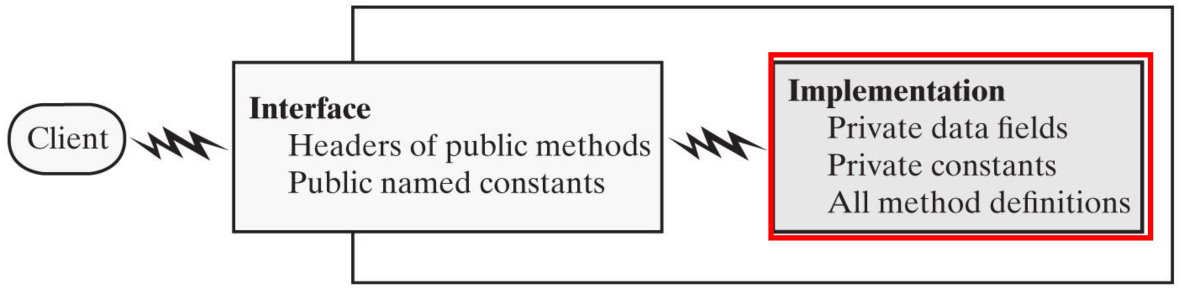
```
ListInterface<String> myList = new ArrayList<>();
```

or

```
ListInterface<String> myList = new Vector<>();
```



Implementations of a List

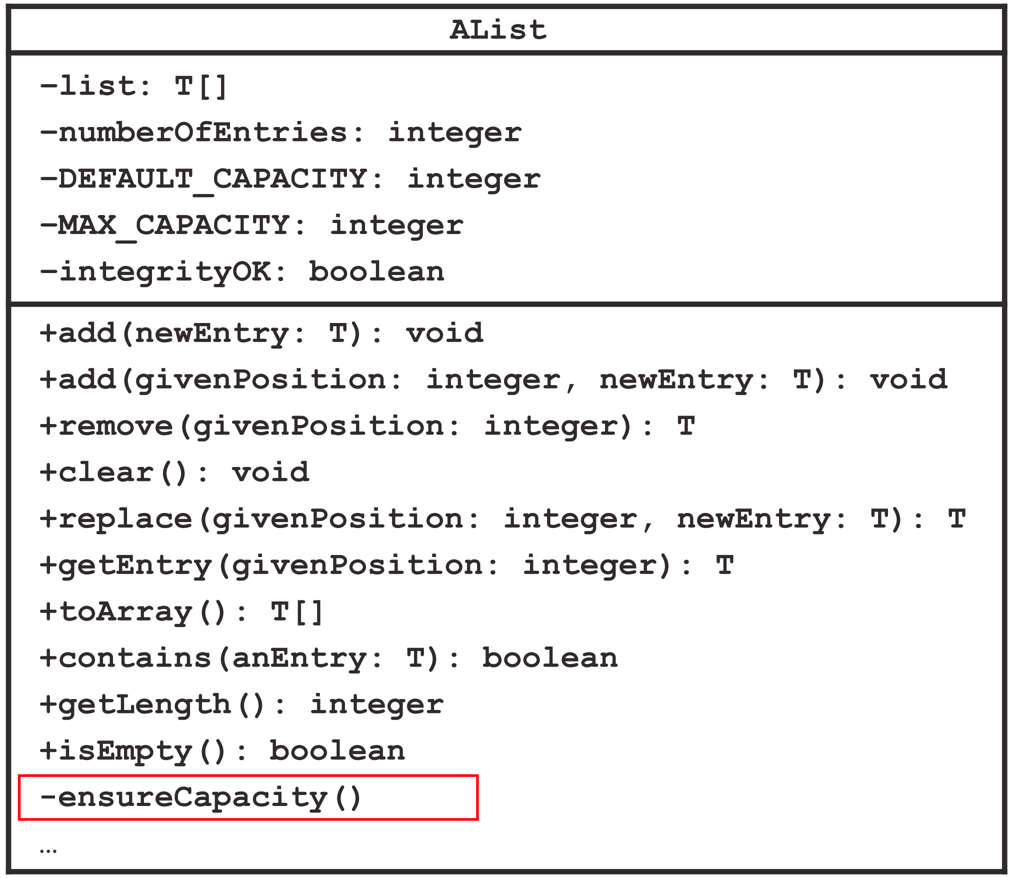


Implementations of a List

- Using Array Resizing
- Using Linked Data

Implementations of a List

- **Using Array Resizing**
- Using Linked Data



```
AList
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity()
…
```

Implementations of a List

```java
import java.util.Arrays;
public class AList<T> implements ListInterface<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 AList()
   {
      this(DEFAULT_CAPACITY);
   } // end default constructor

   public AList(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
   …
}
```

AList
-list: T[] -numberOfEntries: integer -DEFAULT_CAPACITY: integer -MAX_CAPACITY: integer -integrityOK: boolean
+add(newEntry: T): void +add(givenPosition: integer, newEntry: T): void +remove(givenPosition: integer): T +clear(): void +replace(givenPosition: integer, newEntry: T): T +getEntry(givenPosition: integer): T +toArray(): T[] +contains(anEntry: T): boolean +getLength(): integer +isEmpty(): boolean -ensureCapacity() …

*In this course, **position 1 indicates the first entry in the list**.*

Implementations of a List

```
import java.util.Arrays;
public class AList<T> implements ListInterface<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 AList()
   {
      this(DEFAULT_CAPACITY);
   } // end default constructor

   public AList(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
   …
}
```

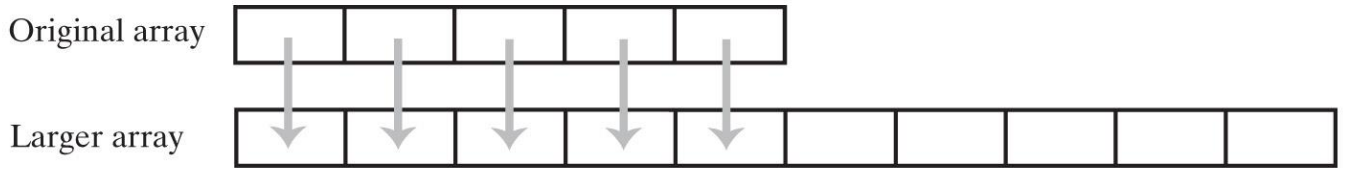
```
 // Throws an exception if this object is corrupt.
   private void checkIntegrity()
   {
      if (!integrityOK)
         throw new SecurityException ("AList object is corrupt.");
   } // end checkIntegrity

   // Throws an exception if the client requests a capacity that is too
large.
   private void checkCapacity(int capacity)
   {
      if (capacity > MAX_CAPACITY)
         throw new IllegalStateException("Attempt to create a list " +
                                         "whose capacity exceeds " +
                                         "allowed maximum.");
   } // end checkCapacity
```

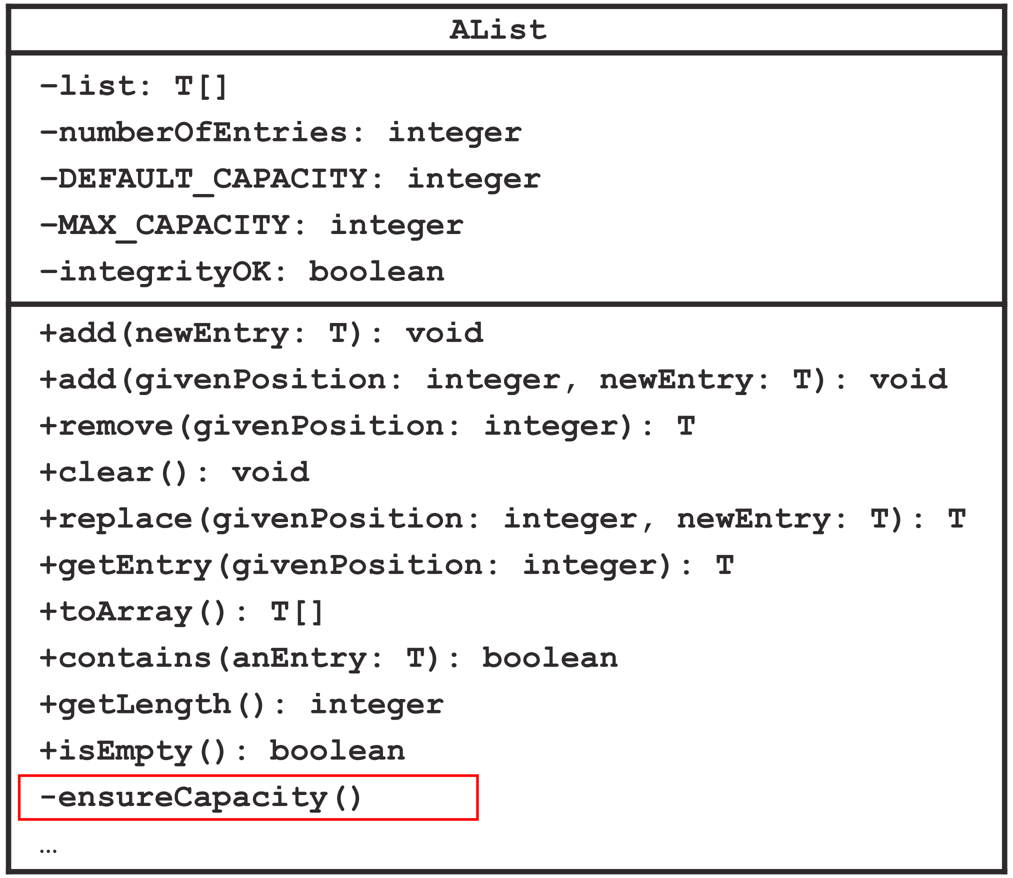
*In this course, **position 1 indicates the first entry in the list**.*

Implementations of a List

- **Resizing Array**

```
private void ensureCapacity()
  {
     int capacity = list.length - 1;
     if (numberOfEntries >= capacity)
     {
        int newCapacity = 2 * capacity;
        checkCapacity(newCapacity); // Is capacity too big?
        list = Arrays.copyOf(list, newCapacity + 1);
     } // end if
  } // end ensureCapacity
```

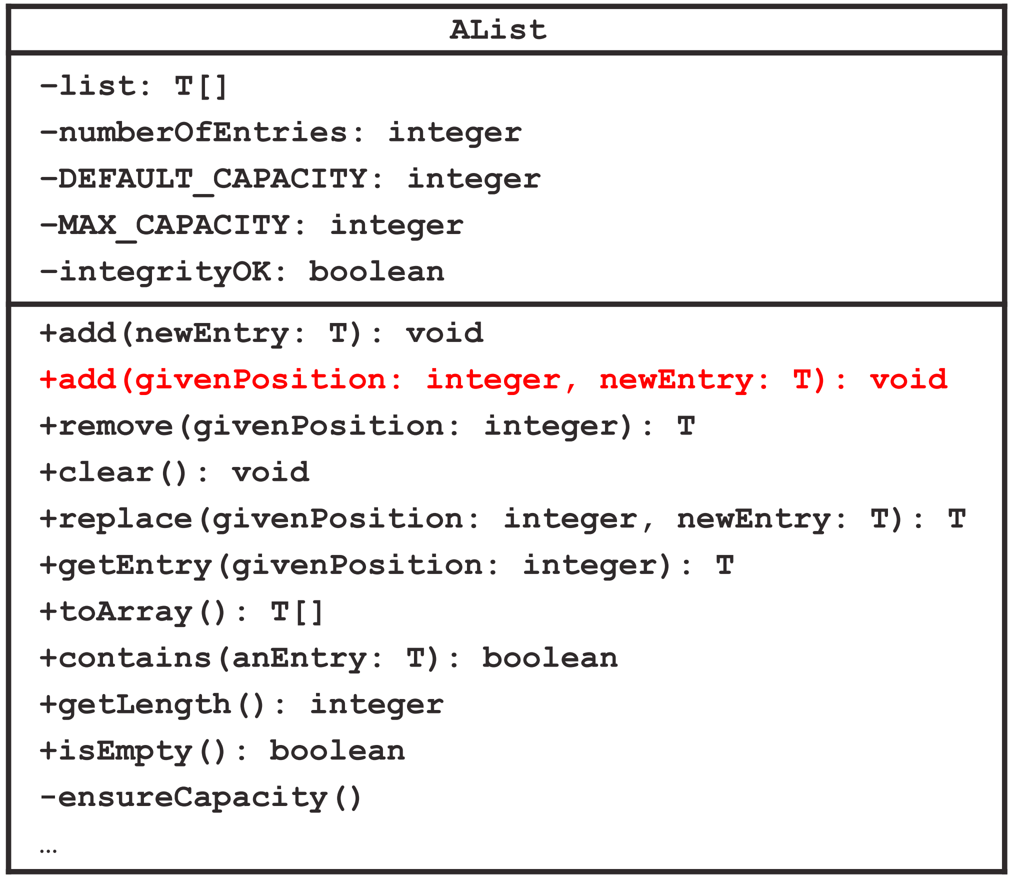




```
AList
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity()
…
```

Implementations of a List

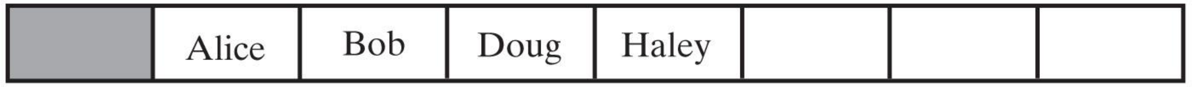
- Adding to a list at a given position



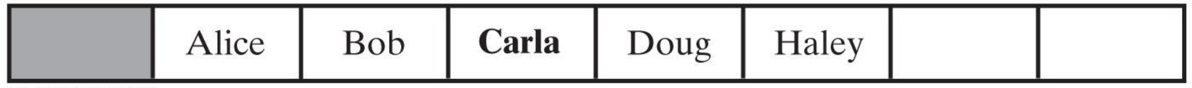
```
AList
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity()
…
```

Implementations of a List

- Adding to a list at a given position



Insert Carla as the third entry in an array

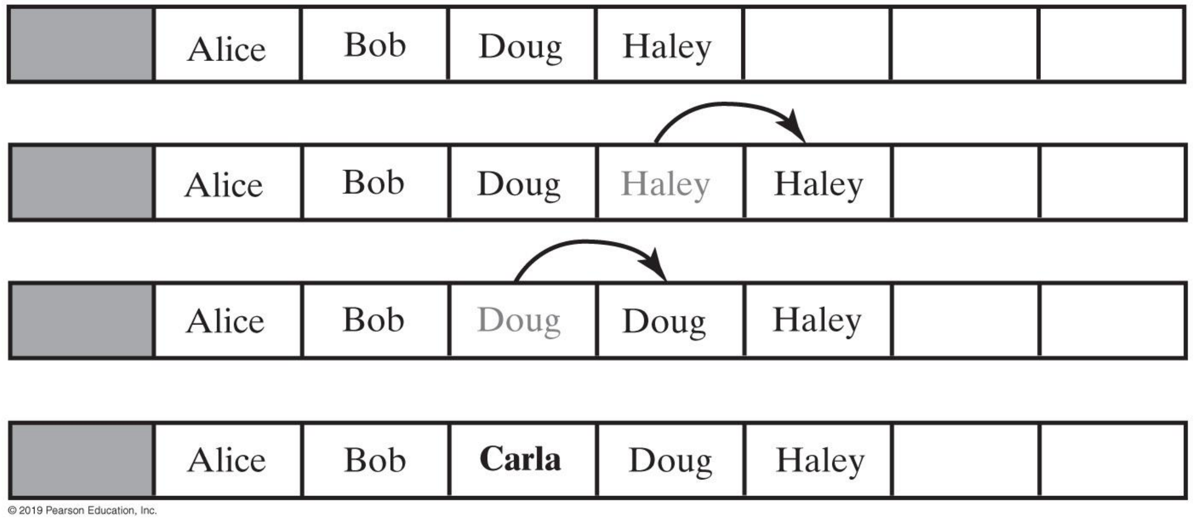


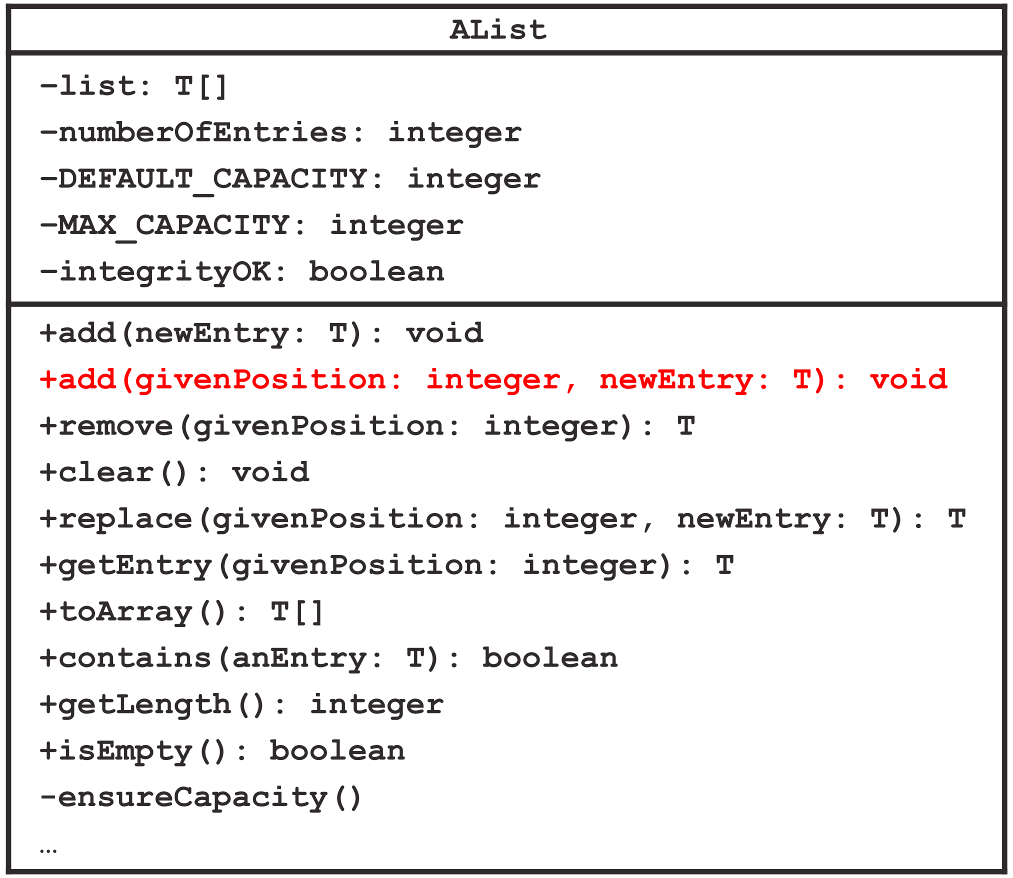
First of all: **Making room** to insert Carla as the third entry in an array

```
AList
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity()
…
```

Implementations of a List

- Adding to a list at a given position



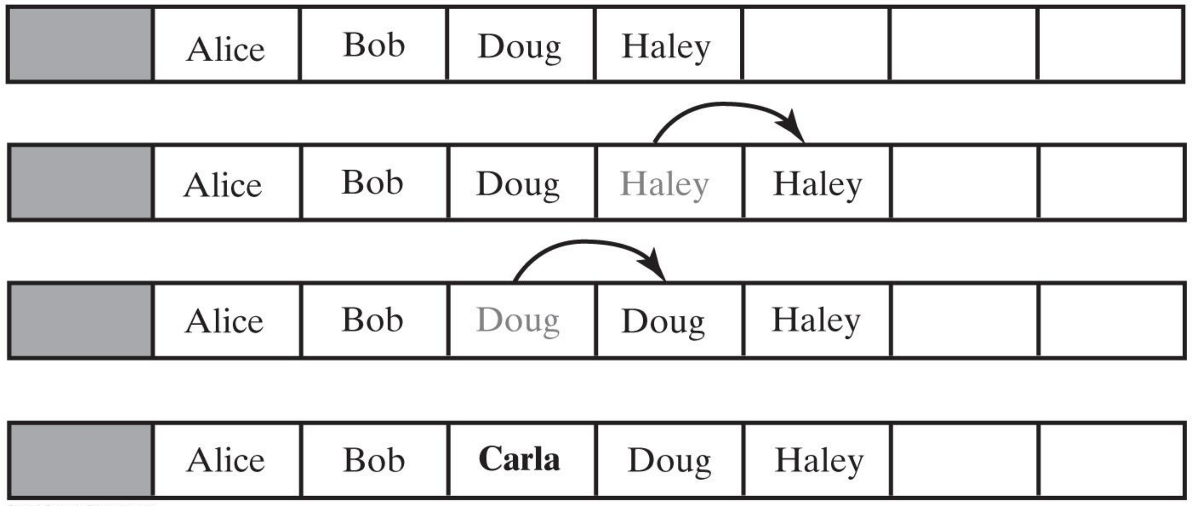


```
AList
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity()
…
```

First of all: **Making room** to insert Carla as the third entry in an array

Implementations of a List

- Adding to a list at a given position



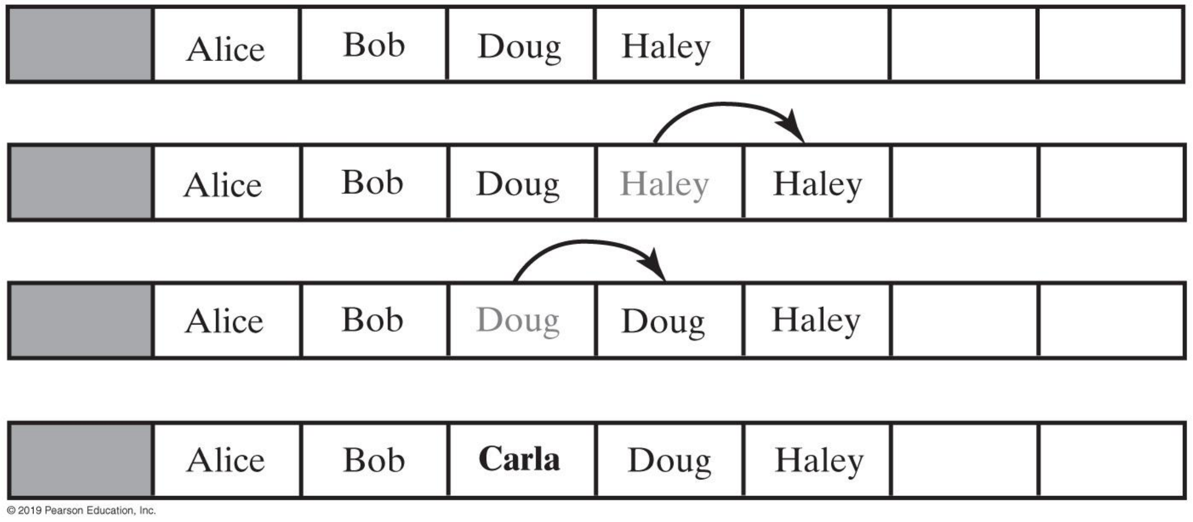
```
public void add(int givenPosition, T newEntry)
  {
     checkIntegrity();
     if ((givenPosition >= 1) && (givenPosition <=
           numberOfEntries + 1))
     {
        if (givenPosition <= numberOfEntries)
           makeRoom(givenPosition);
        list[givenPosition] = newEntry;
        numberOfEntries++;
        ensureCapacity(); // Ensure enough room for next add
     }
     else
        throw new IndexOutOfBoundsException("Given position of
           add's new entry is out of bounds.");
  } // end add
```

First of all: **Making room** to insert Carla as the third entry in an array

Implementations of a List

- Adding to a list at a given position

```
// Throws an exception if this object is corrupt.
   private void checkIntegrity()
   {
      if (!integrityOK)
         throw new SecurityException ("AList object is
            corrupt.");
   } // end checkIntegrity
```



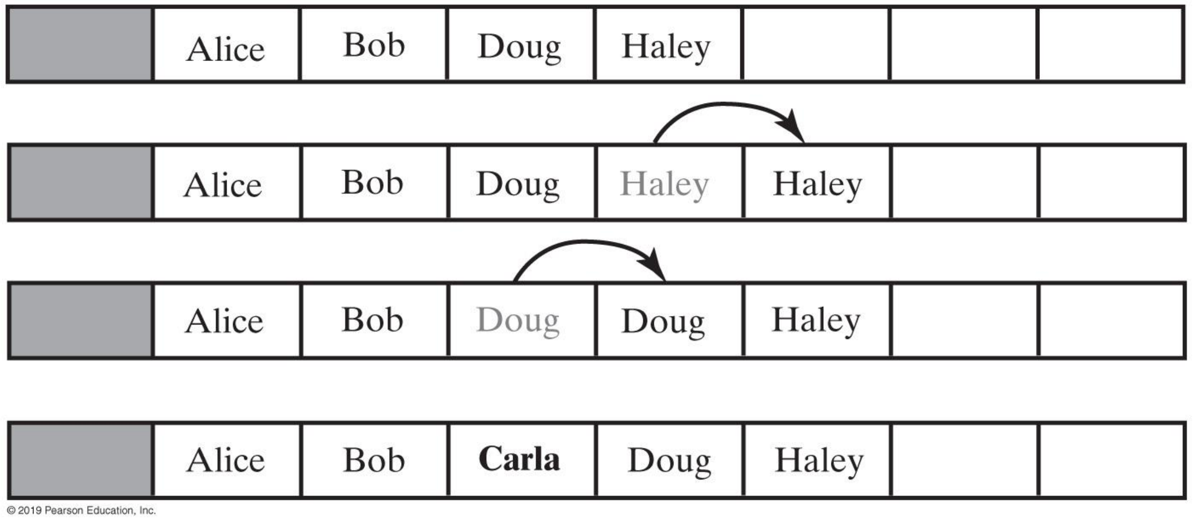
```
public void add(int givenPosition, T newEntry)
  {
     checkIntegrity();
     if ((givenPosition >= 1) && (givenPosition <=
           numberOfEntries + 1))
     {
        if (givenPosition <= numberOfEntries)
           makeRoom(givenPosition);
        list[givenPosition] = newEntry;
        numberOfEntries++;
        ensureCapacity(); // Ensure enough room for next add
     }
     else
        throw new IndexOutOfBoundsException("Given position of
           add's new entry is out of bounds.");
  } // end add
```

First of all: **Making room** to insert Carla as the third entry in an array

In-Class Exercises: Define makeRoom Method

- Adding to a list at a given position

```java
// Throws an exception if this object is corrupt.
  private void checkIntegrity()
  {
     if (!integrityOK)
        throw new SecurityException ("AList object is
           corrupt.");
  } // end checkIntegrity
```



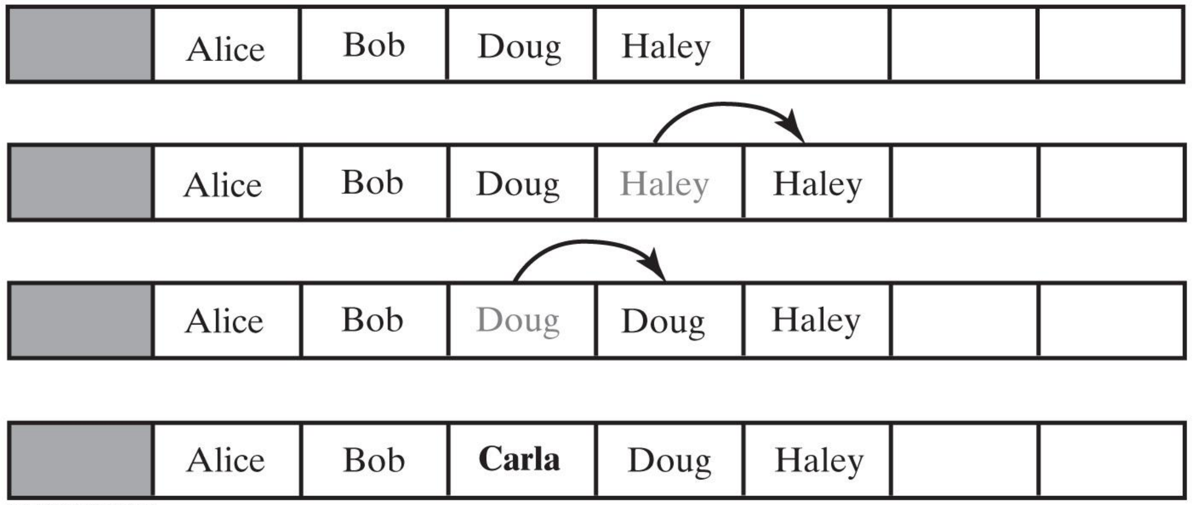
```java
public void add(int givenPosition, T newEntry)
  {
     checkIntegrity();
     if ((givenPosition >= 1) && (givenPosition <=
           numberOfEntries + 1))
     {
        if (givenPosition <= numberOfEntries)
           makeRoom(givenPosition);
        list[givenPosition] = newEntry;
        numberOfEntries++;
        ensureCapacity(); // Ensure enough room for next add
     }
     else
        throw new IndexOutOfBoundsException("Given position of
           add's new entry is out of bounds.");
  } // end add
```

First of all: **Making room** to insert Carla as the third entry in an array

Implementations o

- Adding to a list at a given position

```
 // Makes room for a new entry at newPosition.
   // Precondition: 1 <= newPosition <= numberOfEntries + 1;
//                 numberOfEntries is list's length before addition;
   //                 checkIntegrity has been called.
   private void makeRoom(int givenPosition)
   {
      // Assertion: (newPosition >= 1) && (newPosition <= numberOfEntries + 1)
      int newIndex = givenPosition;
      int lastIndex = numberOfEntries;

      // Move each entry to next higher index, starting at end of
      // list and continuing until the entry at newIndex is moved
      for (int index = lastIndex; index >= newIndex; index--)
         list[index + 1] = list[index];
   }  // end makeRoom
```

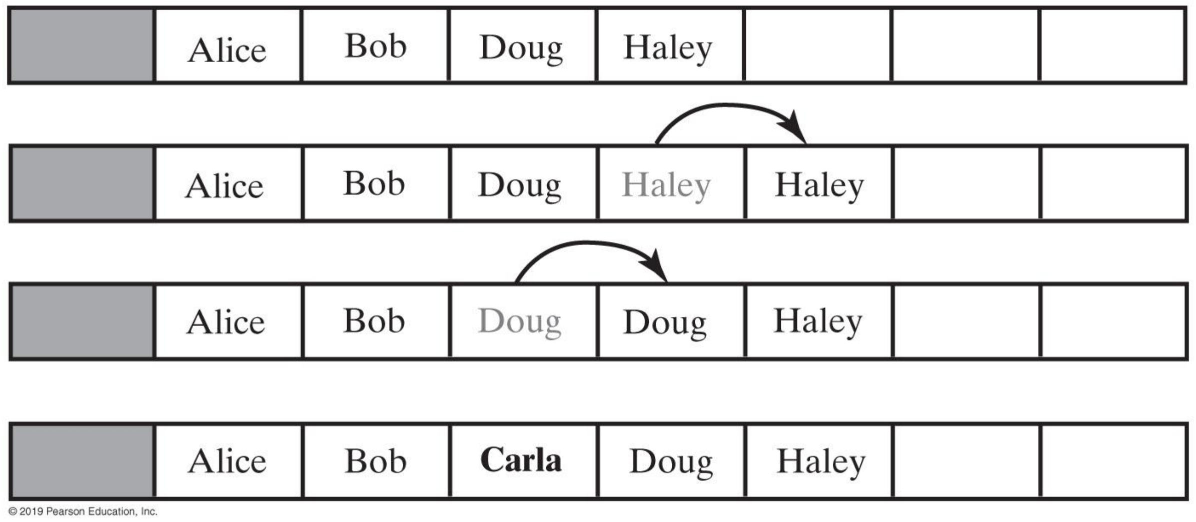


```
public void add(int givenPosition, T newEntry)
  {
     checkIntegrity();
     if ((givenPosition >= 1) && (givenPosition <=
           numberOfEntries + 1))
     {
        if (givenPosition <= numberOfEntries)
           makeRoom(givenPosition);
        list[givenPosition] = newEntry;
        numberOfEntries++;
        ensureCapacity(); // Ensure enough room for next add
     }
     else
        throw new IndexOutOfBoundsException("Given position of
           add's new entry is out of bounds.");
  } // end add
```

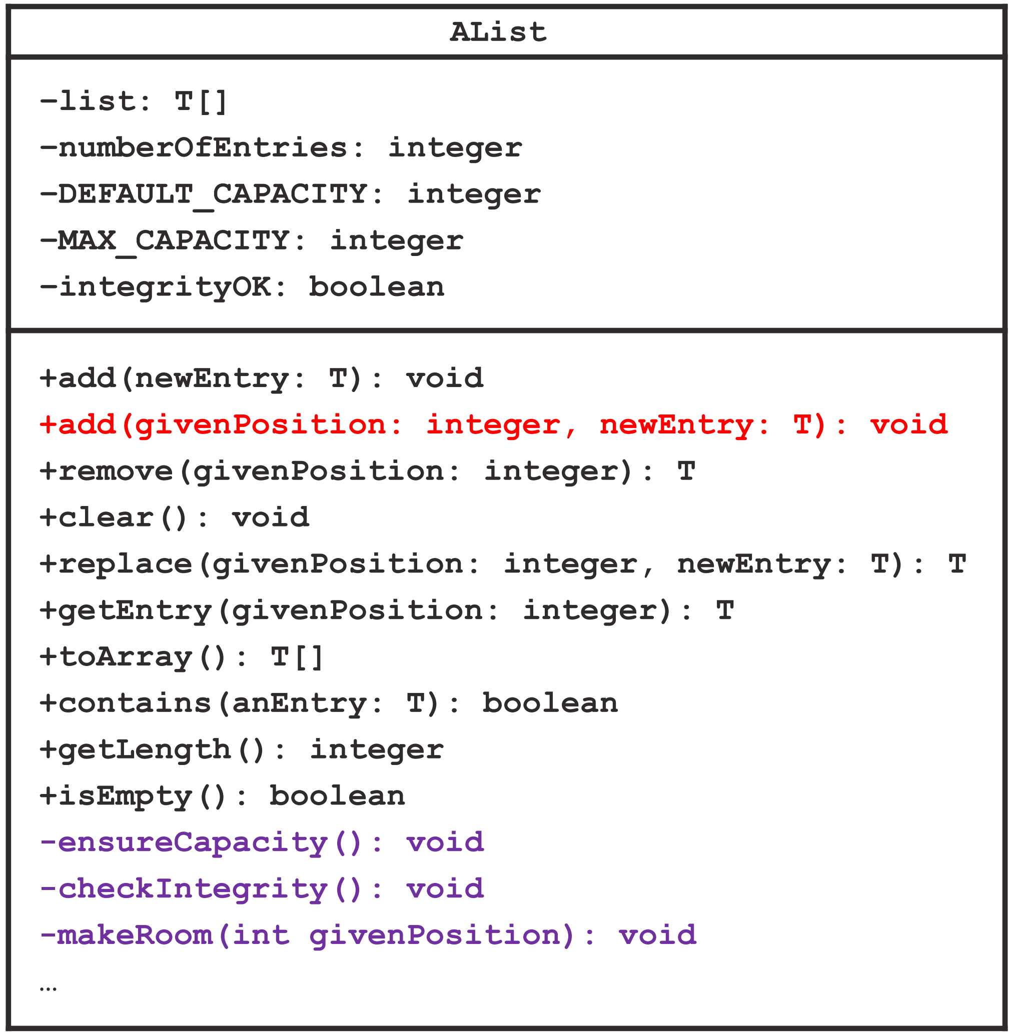
First of all: **Making room** to insert Carla as the third entry in an array

Implementations of a List

- Adding to a list at a given position



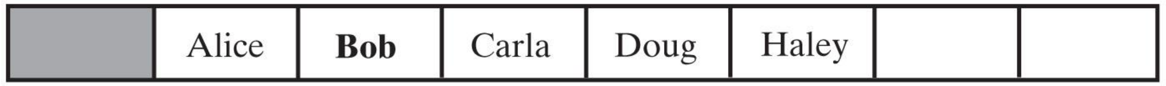
First of all: **Making room** to insert Carla as the third entry in an array



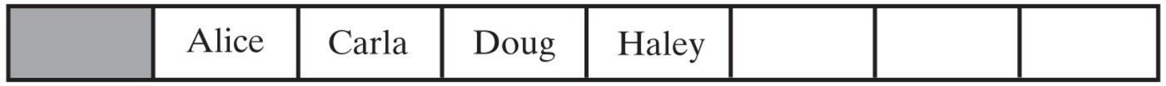
```
AList
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity(): void
-checkIntegrity(): void
-makeRoom(int givenPosition): void
…
```

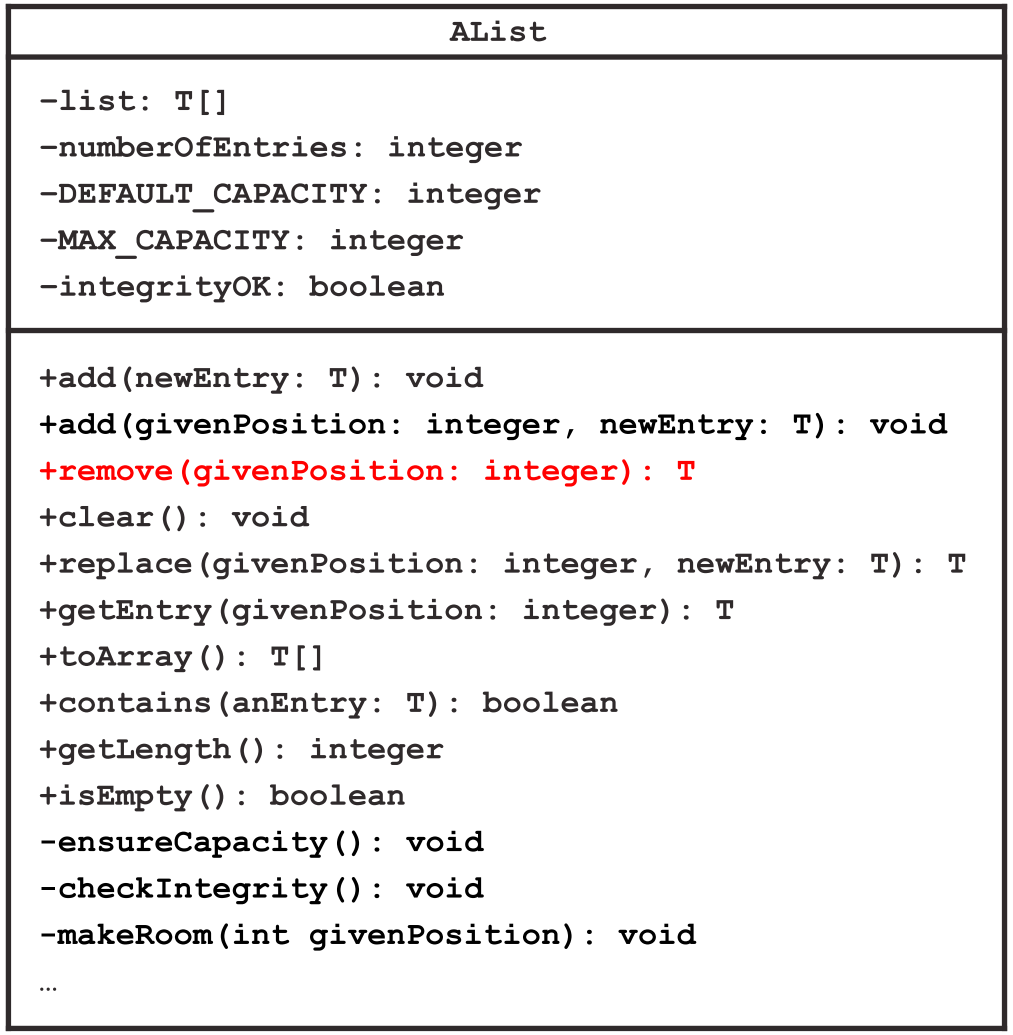
Implementations of a List

- The method **remove**



Remove Bob from the list

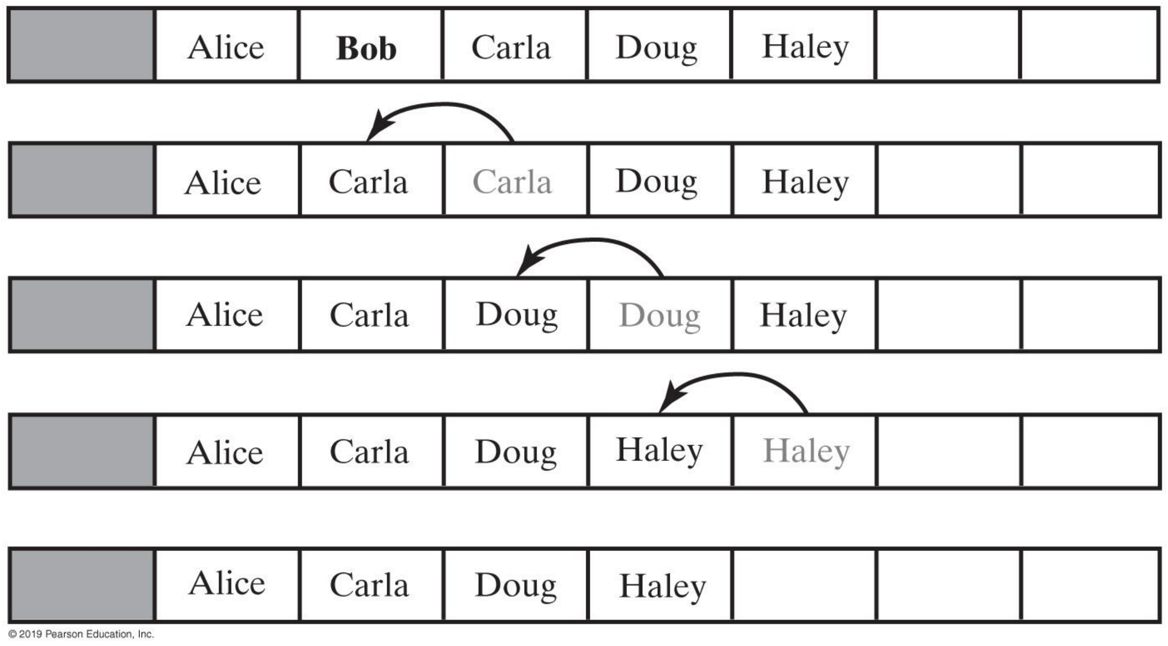




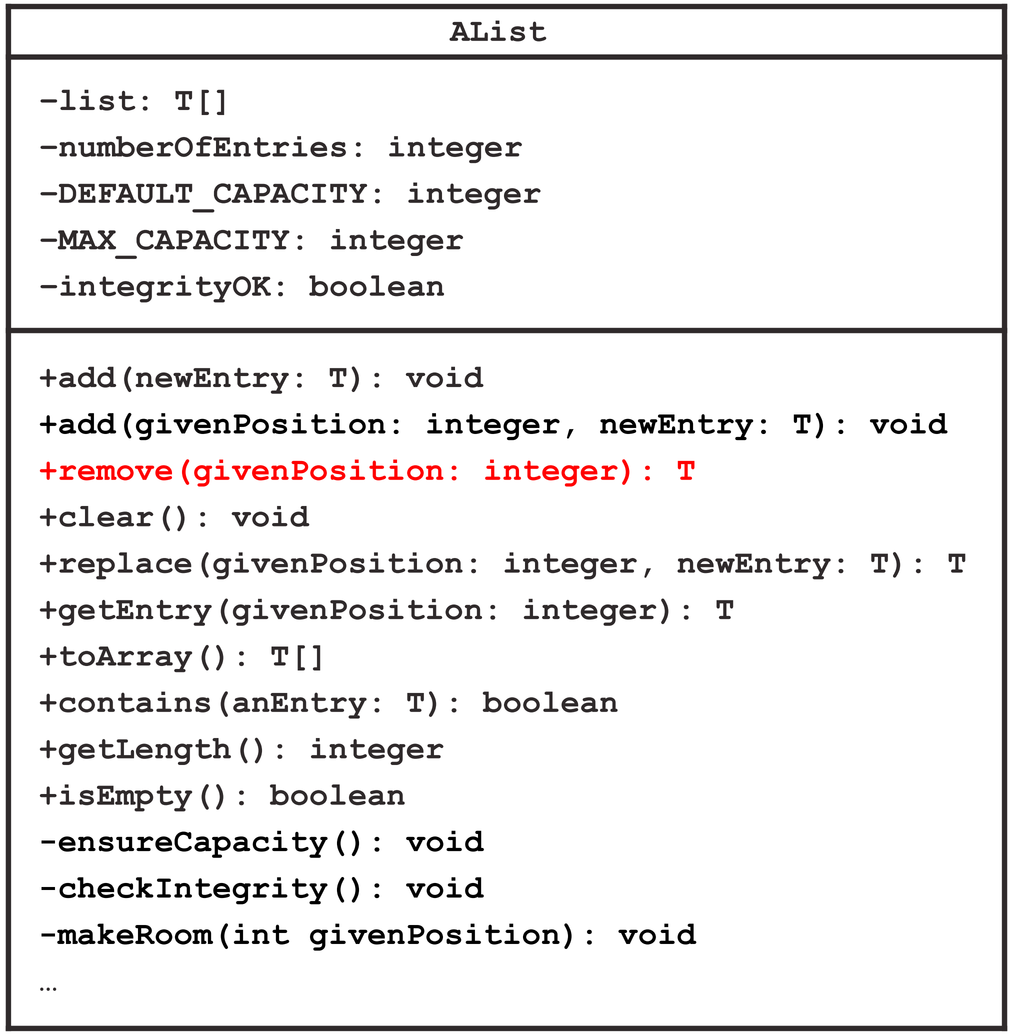
```
AList
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity(): void
-checkIntegrity(): void
-makeRoom(int givenPosition): void
…
```

Implementations of a List

- The method **remove**



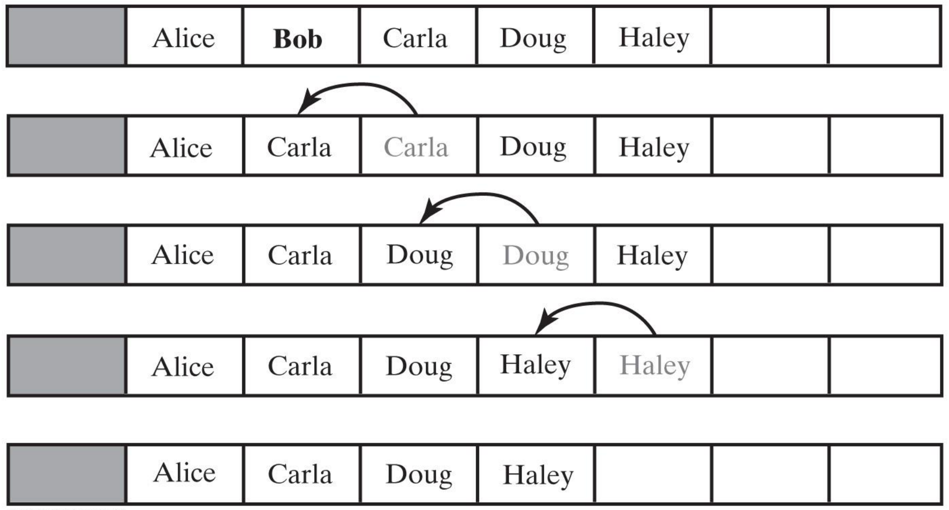
Shifts entries that are beyond the entry to be removed to the next lower position.



```
AList
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity(): void
-checkIntegrity(): void
-makeRoom(int givenPosition): void
…
```

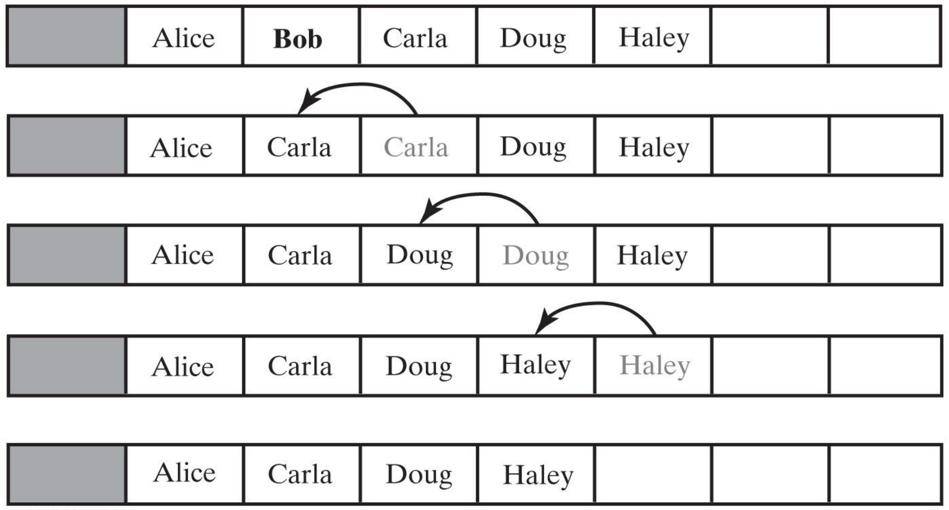
Implementations of a List

- The method **remove**



```java
public T remove(int givenPosition)
  {
     checkIntegrity();
     if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
     {
        // Assertion: The list is not empty
        T result = list[givenPosition]; // Get entry to be removed

        // Move subsequent entries towards entry to be removed,
        // unless it is last in list
        if (givenPosition < numberOfEntries)
           removeGap(givenPosition);
        list[numberOfEntries] = null;
        numberOfEntries--;
        return result;                    // Return reference to removed entry
     }
     else
        throw new IndexOutOfBoundsException("Illegal position given to remove operation.");
  } // end remove
```

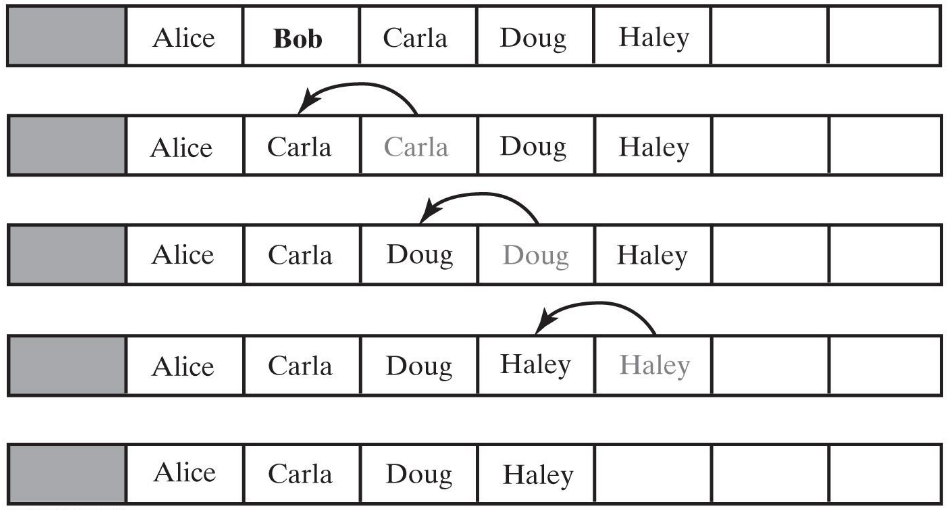
In-Class Exercises: Define removeGap Method



```
public T remove(int givenPosition)
  {
     checkIntegrity();
     if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
     {
        // Assertion: The list is not empty
        T result = list[givenPosition]; // Get entry to be removed

        // Move subsequent entries towards entry to be removed,
        // unless it is last in list
        if (givenPosition < numberOfEntries)
           removeGap(givenPosition);
        list[numberOfEntries] = null;
        numberOfEntries--;
        return result;                    // Return reference to removed entry
     }
     else
        throw new IndexOutOfBoundsException("Illegal position given to remove operation.");
  } // end remove
```

In-Class Exercises: Define removeGap Method

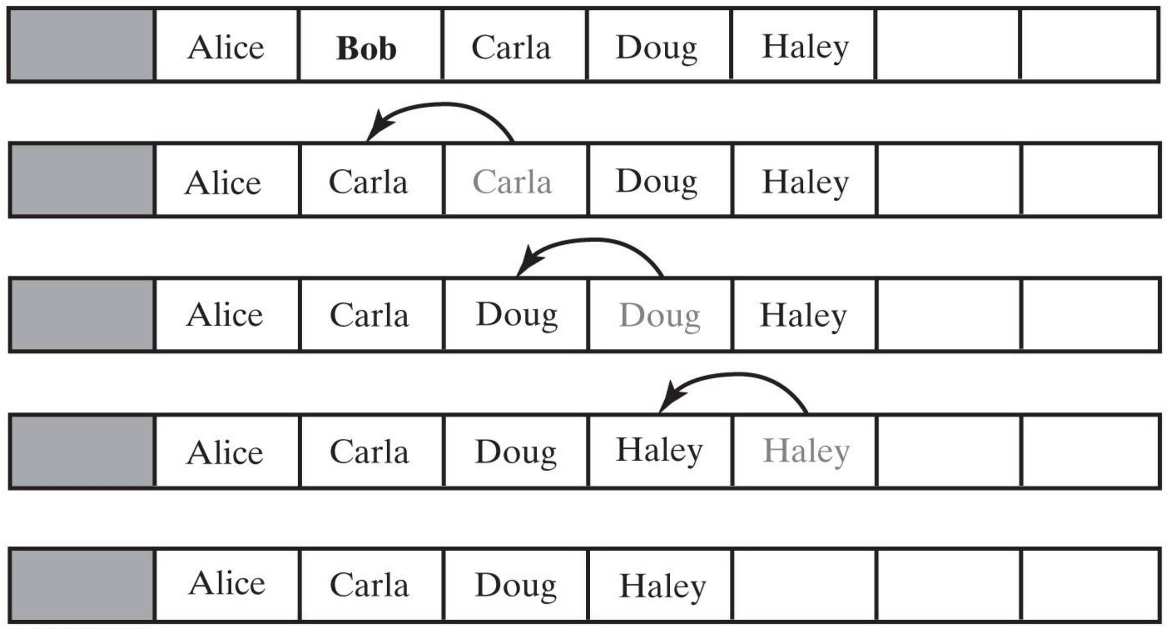
```
// Shifts entries that are beyond the entry to be removed to the
  // next lower position.
  // Precondition: 1 <= givenPosition < numberOfEntries;
  //               numberOfEntries is list's length before removal;
  //               checkIntegrity has been called.
  private void removeGap(int givenPosition)
  {
     int removedIndex = givenPosition;
     for (int index = removedIndex; index < numberOfEntries; index++)
        list[index] = list[index + 1];
  } // end removeGap
```



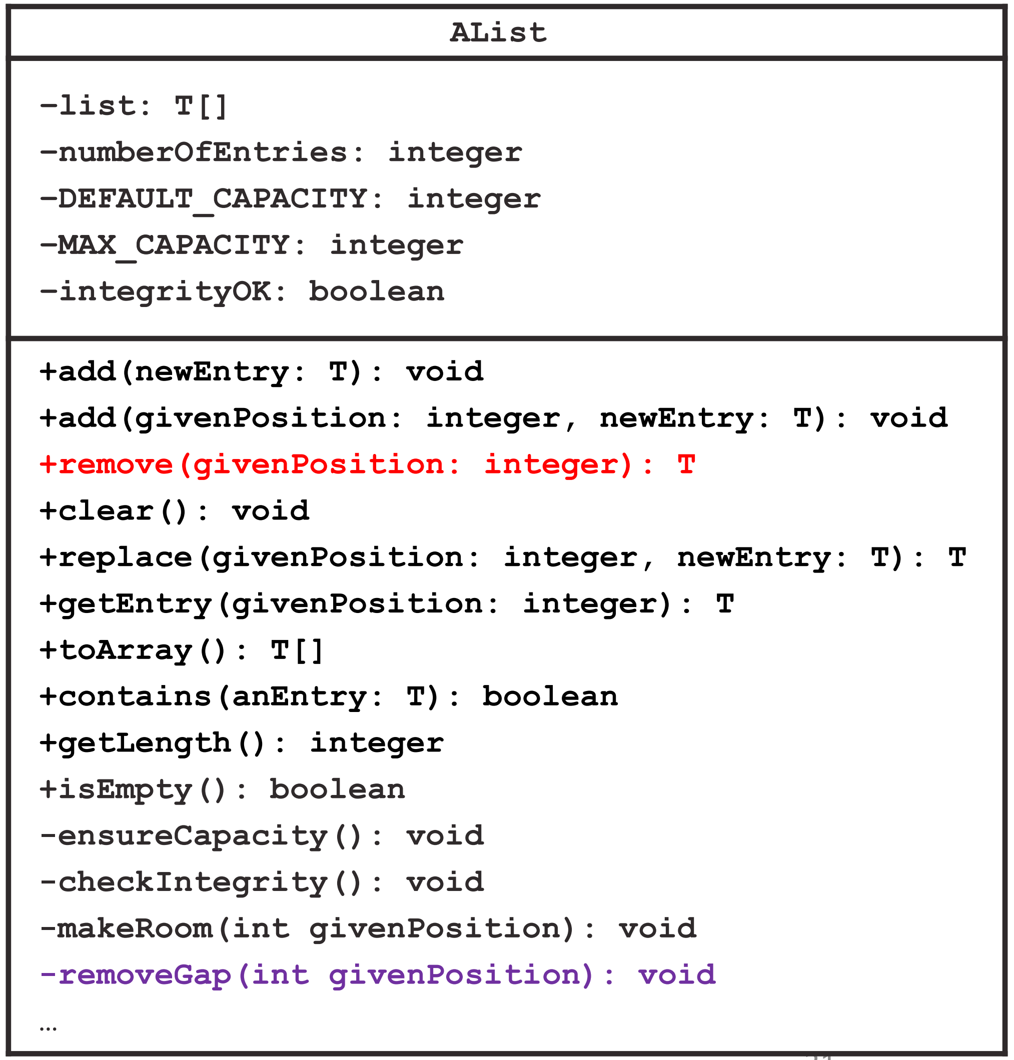
```
public T remove(int givenPosition)
  {
     checkIntegrity();
     if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
     {
        // Assertion: The list is not empty
        T result = list[givenPosition]; // Get entry to be removed

        // Move subsequent entries towards entry to be removed,
        // unless it is last in list
        if (givenPosition < numberOfEntries)
           removeGap(givenPosition);
        list[numberOfEntries] = null;
        numberOfEntries--;
        return result;                    // Return reference to removed entry
     }
     else
        throw new IndexOutOfBoundsException("Illegal position given to remove operation.");
  } // end remove
```

Implementations of a List

- The method **remove**



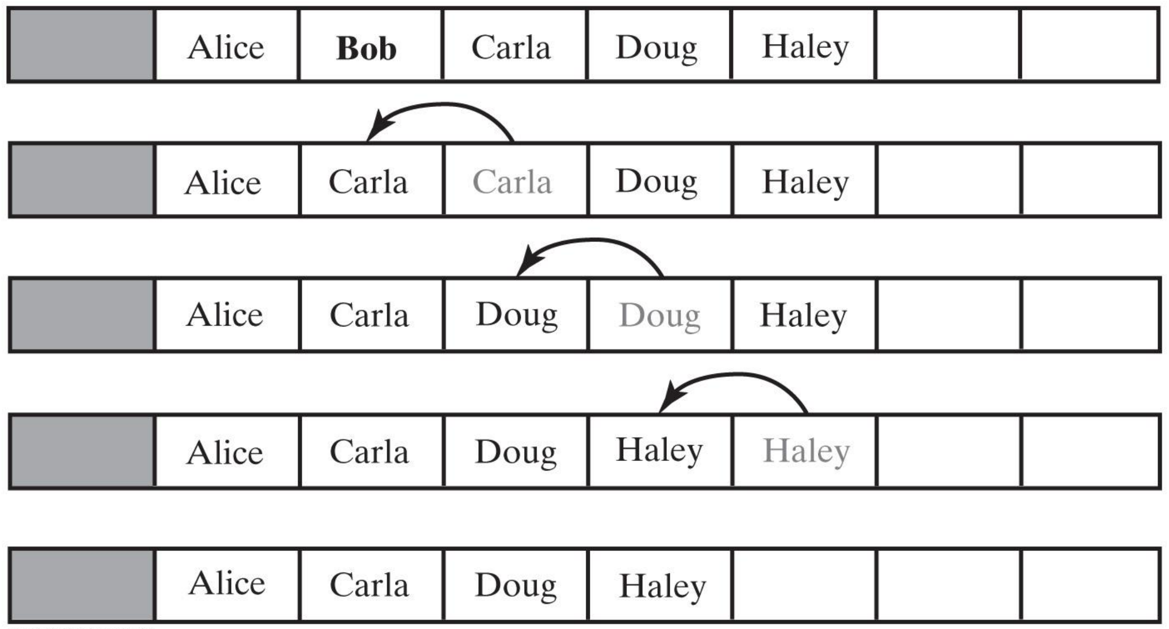
Shifts entries that are beyond the entry to be removed to the next lower position.



```
AList
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity(): void
-checkIntegrity(): void
-makeRoom(int givenPosition): void
-removeGap(int givenPosition): void
…
```

Implementations of a List

- The method **remove**


Alice Bob Carla Doug Haley
Alice Carla Carla Doug Haley
Alice Carla Doug Doug Haley
Alice Carla Doug Haley Haley
Alice Carla Doug Haley

Shifts entries that are beyond the entry to be removed to the next lower position.

```
AList
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity(): void
-checkIntegrity(): void
-makeRoom(int givenPosition): void
-removeGap(int givenPosition): void
…
```

Implementations of a List

- Another **add** method adding a given entry to a list

```
public void add(T newEntry)
{
      add(numberOfEntries + 1, newEntry);
} // end add
```

- The **toArray** method

```
public T[] toArray()
 {
    checkIntegrity();

    // The cast is safe because the new array contains null entries
    @SuppressWarnings("unchecked")
    T[] result = (T[])new Object[numberOfEntries]; // Unchecked cast
    for (int index = 0; index < numberOfEntries; index++)
    {
       result[index] = list[index + 1];
    } // end for

    return result;
 } // end toArray
```

```
AList
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity(): void
-checkIntegrity(): void
-makeRoom(int givenPosition): void
-removeGap(int givenPosition): void
…
```

```java
public void clear()
{
   checkIntegrity();

      // Clear entries but retain array; no need to create a new array
   for (int index = 1; index <= numberOfEntries; index++) // Loop is part of Q4
   list[index] = null;

   numberOfEntries = 0;
} // end clear

public boolean contains(T anEntry)
{
   checkIntegrity();
   boolean found = false;
   int index = 1;
   while (!found && (index <= numberOfEntries))
   {
      if (anEntry.equals(list[index]))
         found = true;
      index++;
   } // end while
   return found;
} // end contains

public int getLength()
{
   return numberOfEntries;
} // end getLength

public boolean isEmpty()
{
   return numberOfEntries == 0; // Or getLength() == 0
} // end isEmpty
```

AList
-list: T[]   -numberOfEntries: integer   -DEFAULT_CAPACITY: integer   -MAX_CAPACITY: integer   -integrityOK: boolean
+add(newEntry: T): void   +add(givenPosition: integer, newEntry: T): void   +remove(givenPosition: integer): T   +clear(): void   +replace(givenPosition: integer, newEntry: T): T   +getEntry(givenPosition: integer): T   +toArray(): T[]   +contains(anEntry: T): boolean   +getLength(): integer   +isEmpty(): boolean   -ensureCapacity(): void   -checkIntegrity(): void   -makeRoom(int givenPosition): void   -removeGap(int givenPosition): void   …

In-Class Exercises: Define replace and getEntry

- The **replace** method
  - Replaces the entry at position *givenPosition* with *newEntry*
- The **getEntry** method
  - Retrieves the entry at position *givenPosition*

```
AList
--------------------------------------------------------------
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
--------------------------------------------------------------
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity(): void
-checkIntegrity(): void
-makeRoom(int givenPosition): void
-removeGap(int givenPosition): void
…
```

In-Class Exercises: Define replace and getEntry

```
public T replace(int givenPosition, T newEntry)
{
checkIntegrity();
      if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
      {
         // Assertion: The list is not empty
         T originalEntry = list[givenPosition];
         list[givenPosition] = newEntry;
         return originalEntry;
      }
      else
         throw new IndexOutOfBoundsException("Illegal position given to
            replace operation.");
   } // end replace
```

```
public T getEntry(int givenPosition)
{
checkIntegrity();
      if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
      {
         // Assertion: The list is not empty
         return list[givenPosition];
      }
      else
         throw new IndexOutOfBoundsException("Illegal position given to
            getEntry operation.");
} // end getEntry
```

```
AList
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity(): void
-checkIntegrity(): void
-makeRoom(int givenPosition): void
-removeGap(int givenPosition): void
…
```

In-Class Exercises: Algorithm Analysis

- What is the **Big Oh** of each list method in the best case and the worst case?

```
AList
-----------------------------------------------------------
-list: T[]
-numberOfEntries: integer
-DEFAULT_CAPACITY: integer
-MAX_CAPACITY: integer
-integrityOK: boolean
-----------------------------------------------------------
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-ensureCapacity(): void
-checkIntegrity(): void
-makeRoom(int givenPosition): void
-removeGap(int givenPosition): void
…
```

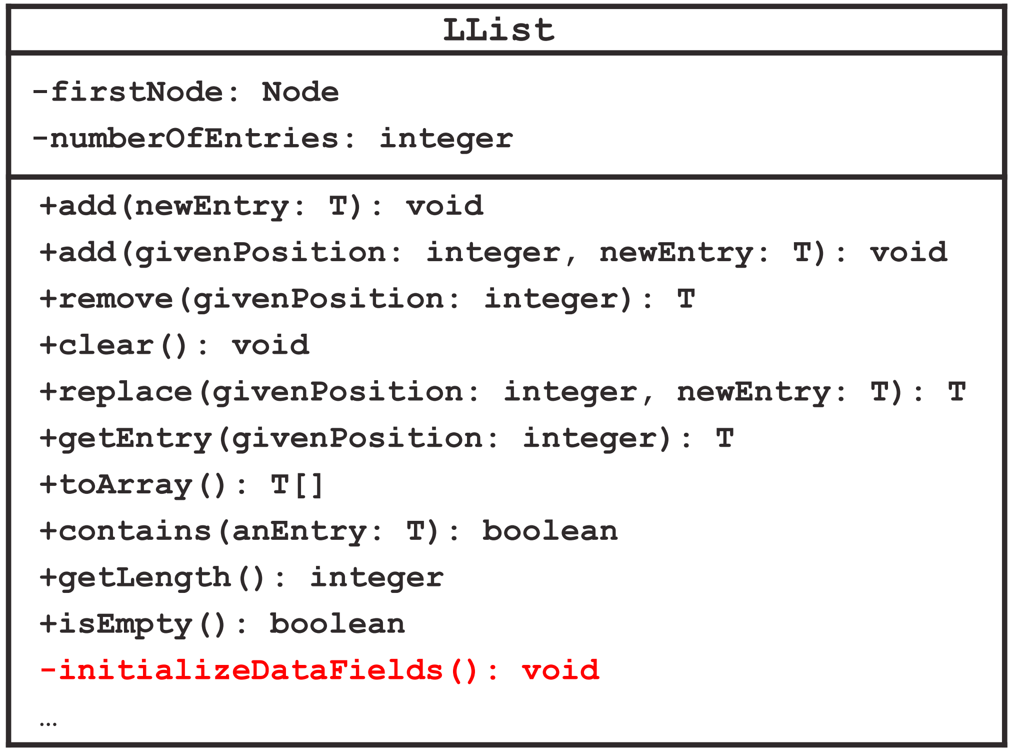
Implementations of a List

- Using Array Resizing
- **Using Linked Data**
  - Uses memory only as needed
  - When entry removed, unneeded memory returned to system
  - Avoids moving data when adding or removing entries

```
LList
-firstNode: Node
-numberOfEntries: integer

+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
…
```

Implementations of a List

```
/** A linked implemention of the ADT list. */
public class LList<T> implements ListInterface<T>
{
        private Node firstNode;         // Reference to first node of chain
        private int  numberOfEntries;

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

        public void clear()
        {
                initializeDataFields();
        } // end clear

       /*  < Implementations of the public methods add, remove, replace, getEntry, contains, getLength, isEmpty, and
toArray go here. > . . . */

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

        private class Node
        {
                ….
        }
}
```



```
LList
-firstNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
…
```

Implementations of a List

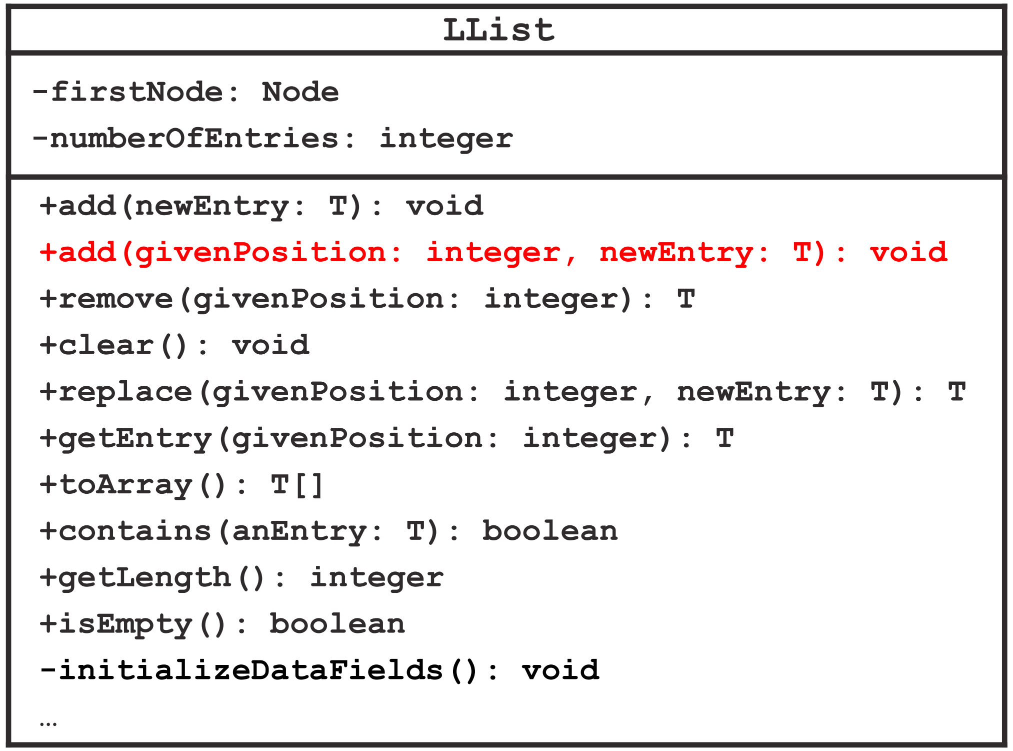
- Adding a Node at Various Positions

```
LList
-----------------------------------------------------------
-firstNode: Node
-numberOfEntries: integer
-----------------------------------------------------------
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
…
```

Implementations of a List

- Adding a Node at Various Positions

- Possible cases:
  - Case 1: Chain is empty
  - Case 2: Adding node at chain's beginning
  - Case 3: Adding node between adjacent nodes
  - Case 4: Adding node to chain's end

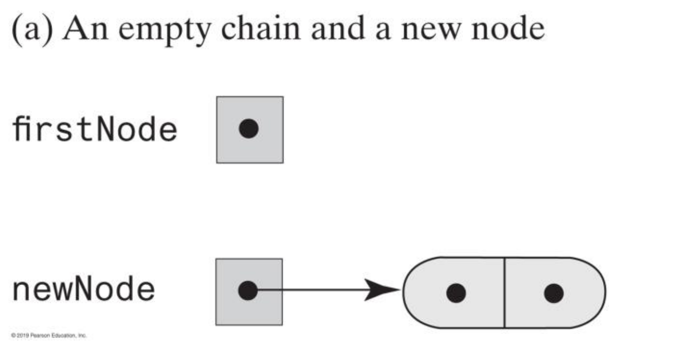


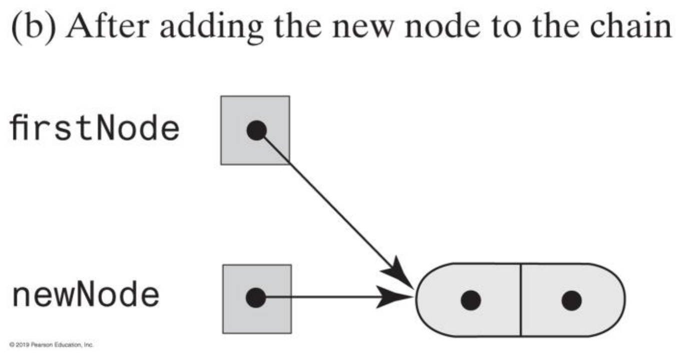
```
LList
-firstNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
…
```

Implementations of a List

- Adding a Node at Various Positions

- Possible cases:
  - **Chain is empty**
  - Adding node at chain's beginning
  - Adding node between adjacent nodes
  - Adding node to chain's end


(a) An empty chain and a new node
firstNode
newNode
© 2019 Pearson Education, Inc.


(b) After adding the new node to the chain
firstNode
newNode
© 2019 Pearson Education, Inc.

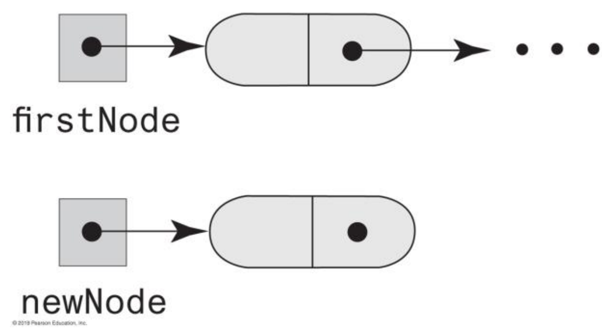
```
Node newNode = new Node(newEntry);
firstNode = newNode;
```

Implementations of a List

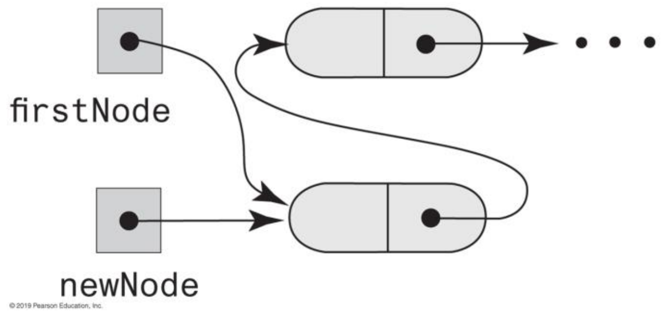
- Adding a Node at Various Positions

- Possible cases:
    - Chain is empty
    - **Adding node at chain's beginning**
    - Adding node between adjacent nodes
    - Adding node to chain's end

(a) A chain of nodes and a new node



(b) After adding the new node to the beginning of the chain



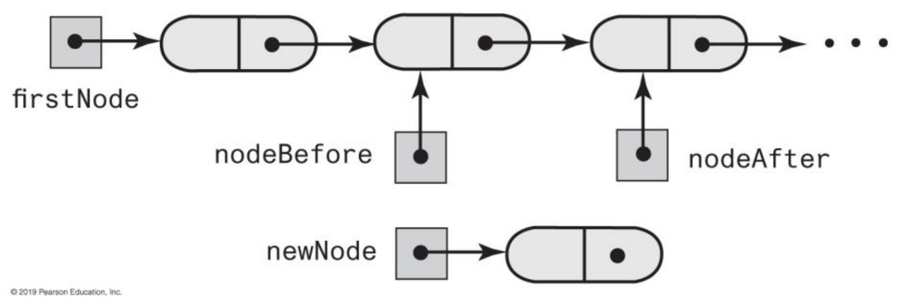
```
Node newNode = new Node(newEntry);
newNode.setNextNode(firstNode);
firstNode = newNode;
```

Implementations of a List

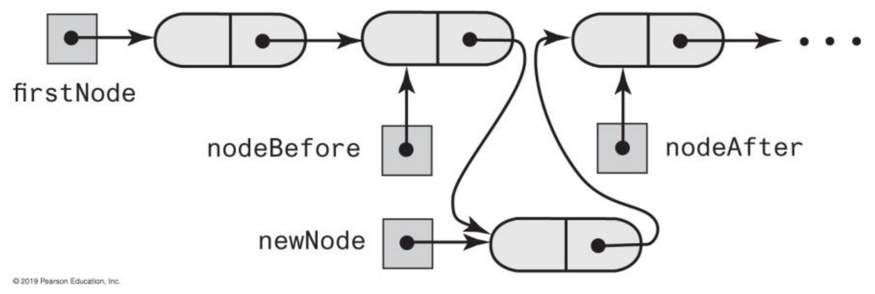
- Adding a Node at Various Positions

- Possible cases:
    - Chain is empty
    - Adding node at chain's beginning
    - **Adding node between adjacent nodes**
    - Adding node to chain's end

(a) A chain of nodes and a new node



(b) After adding the new node between adjacent nodes



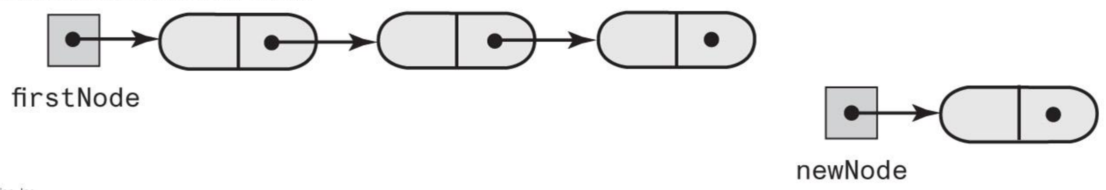
```
Node newNode = new Node(newEntry);
Node nodeBefore = getNodeAt(newPosition - 1);
Node nodeAfter = nodeBefore.getNextNode();
newNode.setNextNode(nodeAfter);
nodeBefore.setNextNode(newNode);
```

Implementations of a List

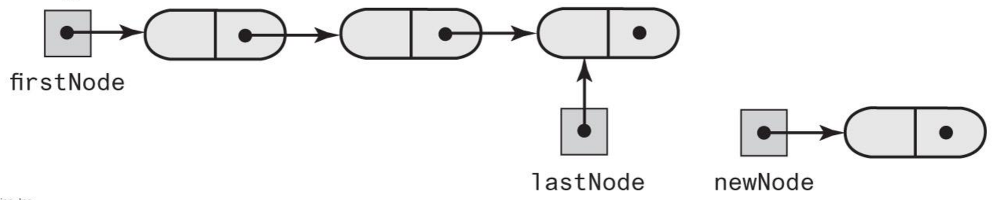
- Adding a Node at Various Positions

- Possible cases:
  - Chain is empty
  - Adding node at chain's beginning
  - Adding node between adjacent nodes
  - **Adding node to chain's end**

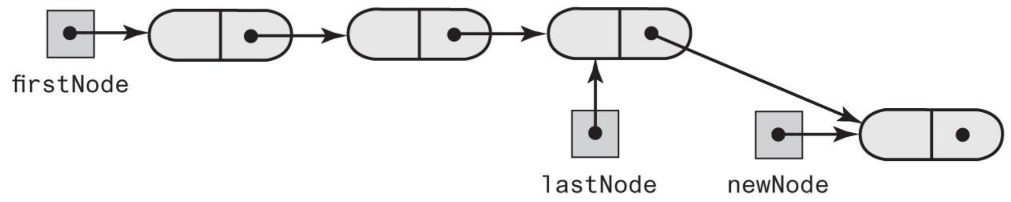
(a) A chain of nodes and a new node



(b) After locating the last node



(c) After adding the new node to the end of the chain



```
Node newNode = new Node(newEntry);
Node lastNode = getNodeAt(numberOfNodes);
lastNode.setNextNode(newNode);
```

Implementations of a List

- Operations on a chain depended on the method ***getNodeAt***

```
private Node getNodeAt(int givenPosition)
{
     // Assertion: (firstNode != null) &&
     //            (1 <= givenPosition) && (givenPosition <= numberOfEntries)
     Node currentNode = firstNode;

     // Traverse the chain to locate the desired node
     // (skipped if givenPosition is 1)
     for (int counter = 1; counter < givenPosition; counter++)
        currentNode = currentNode.getNextNode();

     // Assertion: currentNode != null
     return currentNode;
 } // end getNodeAt
```

```
LList
-----------------------------------------------------------
-firstNode: Node
-numberOfEntries: integer
-----------------------------------------------------------
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
-getNodeAt(int givenPosition): Node
…
```

Implementations of a List

```java
public void add(int givenPosition, T newEntry) // OutOfMemoryError possible
  {
     if ((givenPosition >= 1) && (givenPosition <= numberOfEntries + 1))
     {
        Node newNode = new Node(newEntry);
        if (givenPosition == 1)                // Cases 1 and 2
        {
           newNode.setNextNode(firstNode);
           firstNode = newNode;
        }
        else                                    // Cases 3 and 4: list is not empty
        {                                       // and givenPosition > 1
           Node nodeBefore = getNodeAt(givenPosition - 1);
           Node nodeAfter = nodeBefore.getNextNode();
           newNode.setNextNode(nodeAfter);
           nodeBefore.setNextNode(newNode);
        } // end if
        numberOfEntries++;
     }
     else
        throw new IndexOutOfBoundsException("Illegal position given to add
           operation.");
  } // end add
```

```
LList
-firstNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
-getNodeAt(int givenPosition): Node
…
```

Case 1: Adding the new node to the beginning of the chain.
Case 2: Adding the new node at a position other than the beginning of the chain.

In-Class Exercises

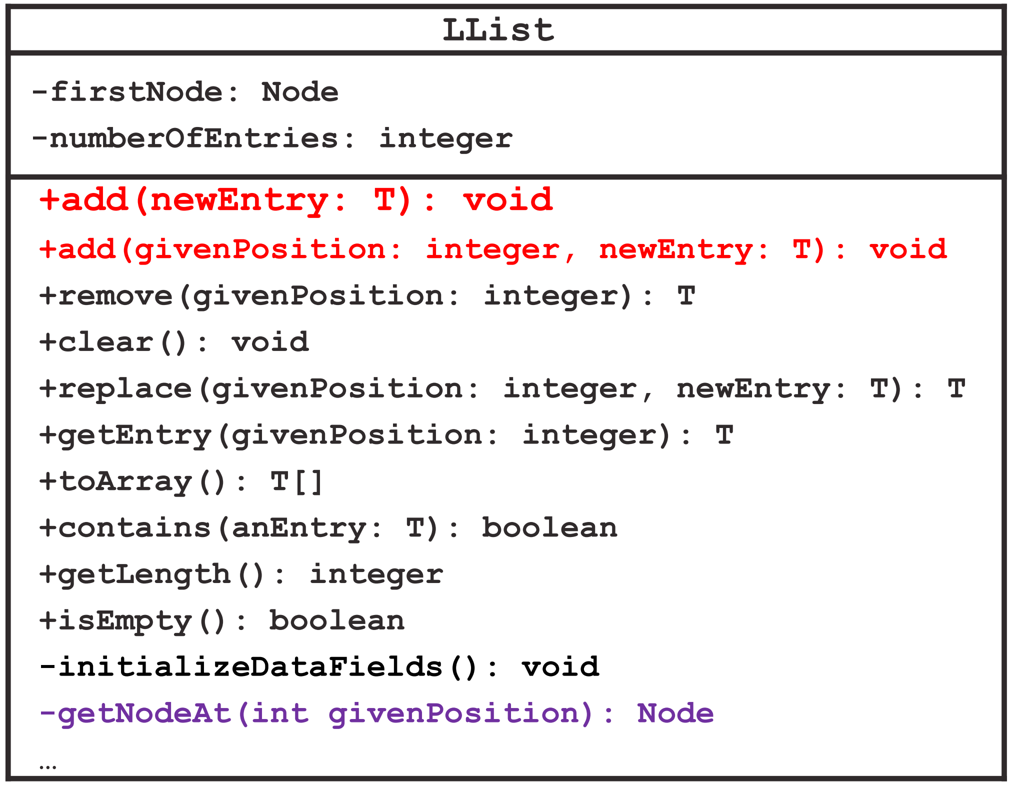
- How to implement another **add** method?

```
LList
-firstNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
-getNodeAt(int givenPosition): Node
…
```

In-Class Exercises

- How to implement another **add** method?

```
public void add(T newEntry)          // OutOfMemoryError possible
  {
     Node newNode = new Node(newEntry);

     if (isEmpty())
        firstNode = newNode;
     else                                 // Add to end of nonempty list
     {
        Node lastNode = getNodeAt(numberOfEntries);
        lastNode.setNextNode(newNode); // Make last node reference new node
     } // end if

     numberOfEntries++;
  } // end add
```

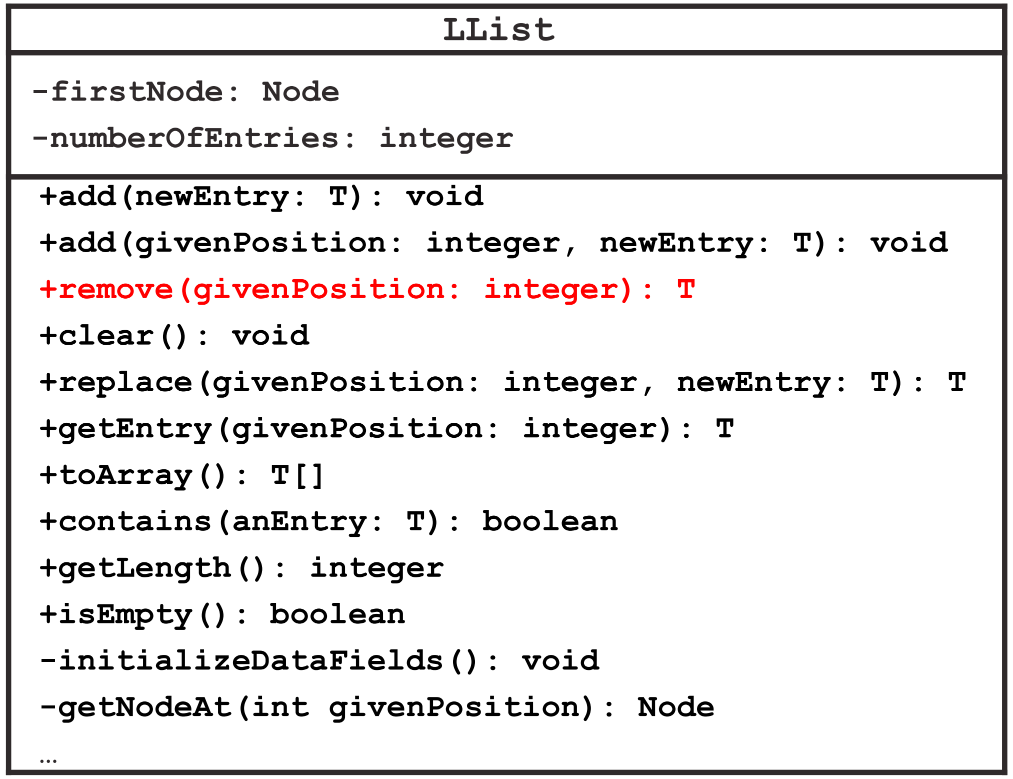


```
LList
-firstNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
-getNodeAt(int givenPosition): Node
…
```

Implementations of a List

- Removing a Node from Various Positions

- Possible cases
  - Removing the first node
  - Removing a node other than first one

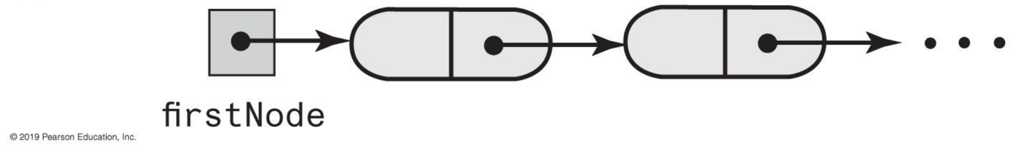


```
LList
-firstNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
-getNodeAt(int givenPosition): Node
…
```

Implementations of a List

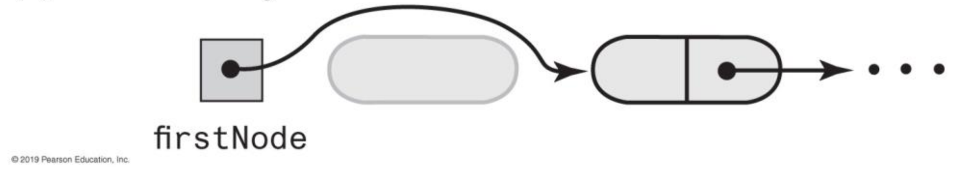
- Removing a Node from Various Positions

(a) A chain of nodes



- Possible cases
    - **Removing the first node**
    - Removing a node other than first one

(b) After removing the first node

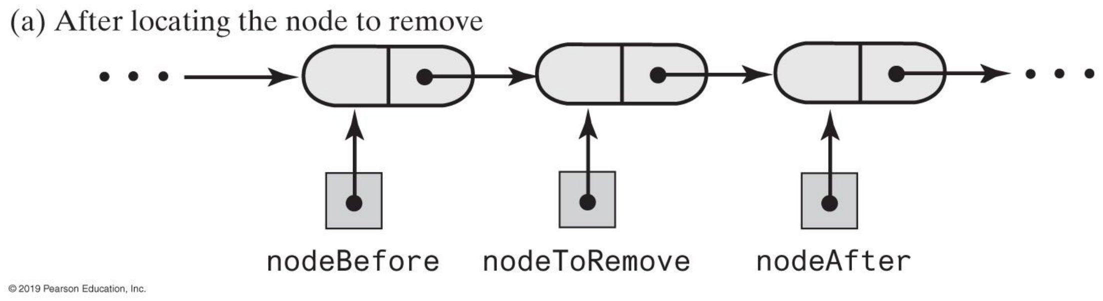


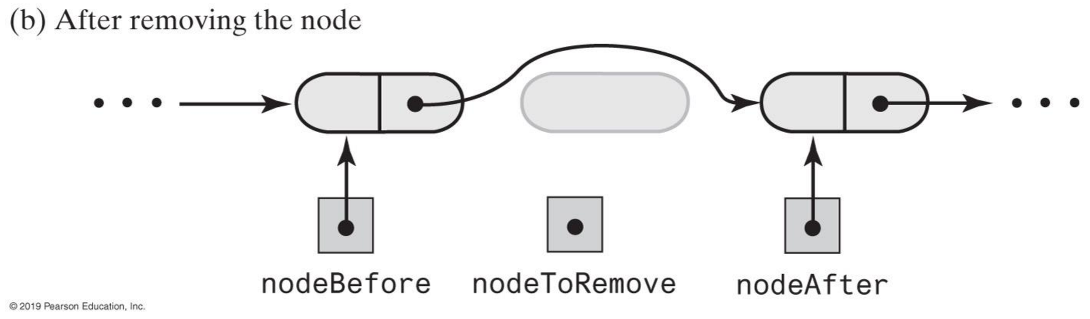
```
firstNode = firstNode.getNextNode();
```

Implementations of a List

- Removing a Node from Various Positions

- Possible cases
  - Removing the first node
  - **Removing a node other than first one**


(a) After locating the node to remove
nodeBefore
nodeToRemove
nodeAfter
© 2019 Pearson Education, Inc.



```
Node nodeBefore = getNodeAt(givenPosition - 1);
Node nodeToRemove = nodeBefore.getNextNode();
Node nodeAfter = nodeToRemove.getNextNode();
nodeBefore.setNextNode(nodeAfter);
nodeToRemove = null;
```

Implementations of a List

- Removing a Node from Various Positions

```java
public T remove(int givenPosition)
{
      T result = null;                          // Return value
      if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
      {
         // Assertion: !isEmpty()
         if (givenPosition == 1)                // Case 1: Remove first entry
         {


         }
         else                                   // Case 2: Not first entry
         {





        } // end if
         numberOfEntries--;                     // Update count
         return result;                         // Return removed entry
      }
      else
         throw new IndexOutOfBoundsException("Illegal position given to remove
            operation.");
} // end remove
```

```
LList
-firstNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
-getNodeAt(int givenPosition): Node
…
```

Implementations of a List

- Removing a Node from Various Positions

```
public T remove(int givenPosition)
{
      T result = null;                         // Return value
      if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
      {
         // Assertion: !isEmpty()
         if (givenPosition == 1)               // Case 1: Remove first entry
         {
            result = firstNode.getData();       // Save entry to be removed
            firstNode = firstNode.getNextNode(); // Remove entry
         }
         else                                  // Case 2: Not first entry
         {
            Node nodeBefore = getNodeAt(givenPosition - 1);
            Node nodeToRemove = nodeBefore.getNextNode();
            result = nodeToRemove.getData();    // Save entry to be removed
            Node nodeAfter = nodeToRemove.getNextNode();
            nodeBefore.setNextNode(nodeAfter);  // Remove entry
        } // end if
         numberOfEntries--;                    // Update count
         return result;                        // Return removed entry
      }
      else
         throw new IndexOutOfBoundsException("Illegal position given to remove
            operation.");
} // end remove
```

```
LList
-firstNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
-getNodeAt(int givenPosition): Node
…
```

Implementations of a List

- The **toArray** method

```
public T[] toArray()
{
   // The cast is safe because the new array contains null entries
   @SuppressWarnings("unchecked")
   T[] result = (T[])new Object[numberOfEntries];

   int index = 0;
   Node currentNode = firstNode;
   while ((index < numberOfEntries) && (currentNode != null))
   {
      result[index] = currentNode.getData();
      currentNode = currentNode.getNextNode();
      index++;
   } // end while

   return result;
} // end toArray
```

LList
-firstNode: Node -numberOfEntries: integer
+add(newEntry: T): void +add(givenPosition: integer, newEntry: T): void +remove(givenPosition: integer): T +clear(): void +replace(givenPosition: integer, newEntry: T): T +getEntry(givenPosition: integer): T +toArray(): T[] +contains(anEntry: T): boolean +getLength(): integer +isEmpty(): boolean -initializeDataFields(): void -getNodeAt(int givenPosition): Node …

Implementations of a List

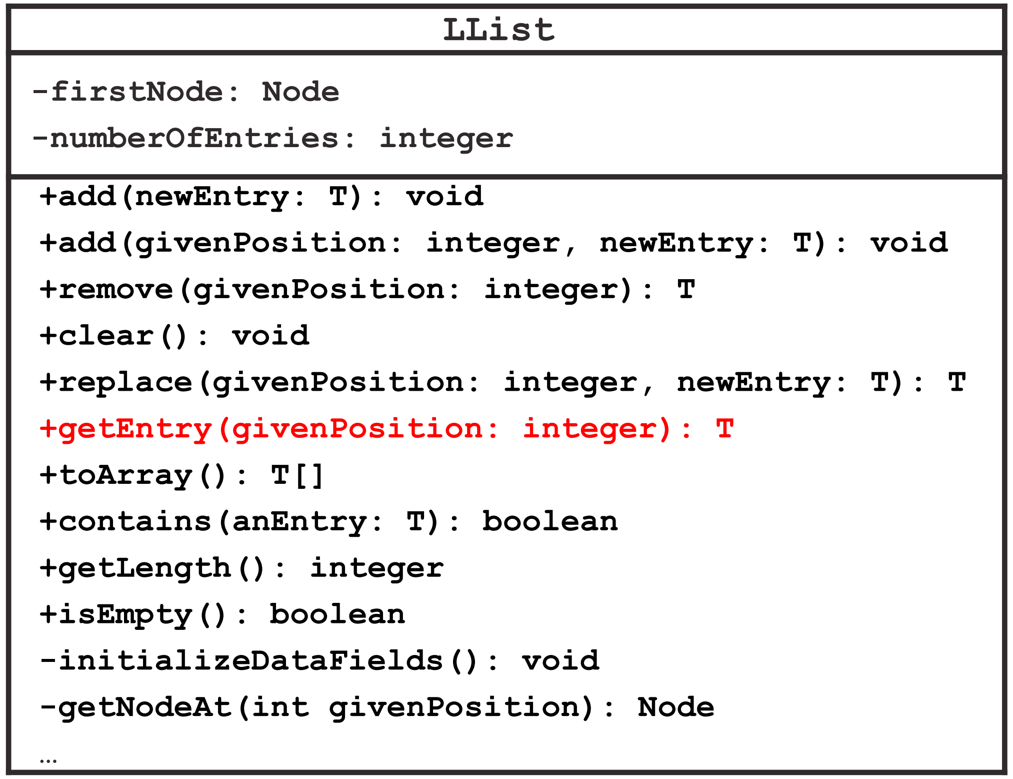
- The **contains** method

```java
public boolean contains(T anEntry)
{
   boolean found = false;
   Node currentNode = firstNode;

   while (!found && (currentNode != null))
   {
      if (anEntry.equals(currentNode.getData()))
         found = true;
      else
         currentNode = currentNode.getNextNode();
   } // end while

   return found;
} // end contains
```

```
LList
-firstNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
-getNodeAt(int givenPosition): Node
…
```

In-Class Exercises

- The **getEntry** method:



```
LList
-firstNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
-getNodeAt(int givenPosition): Node
…
```

In-Class Exercises

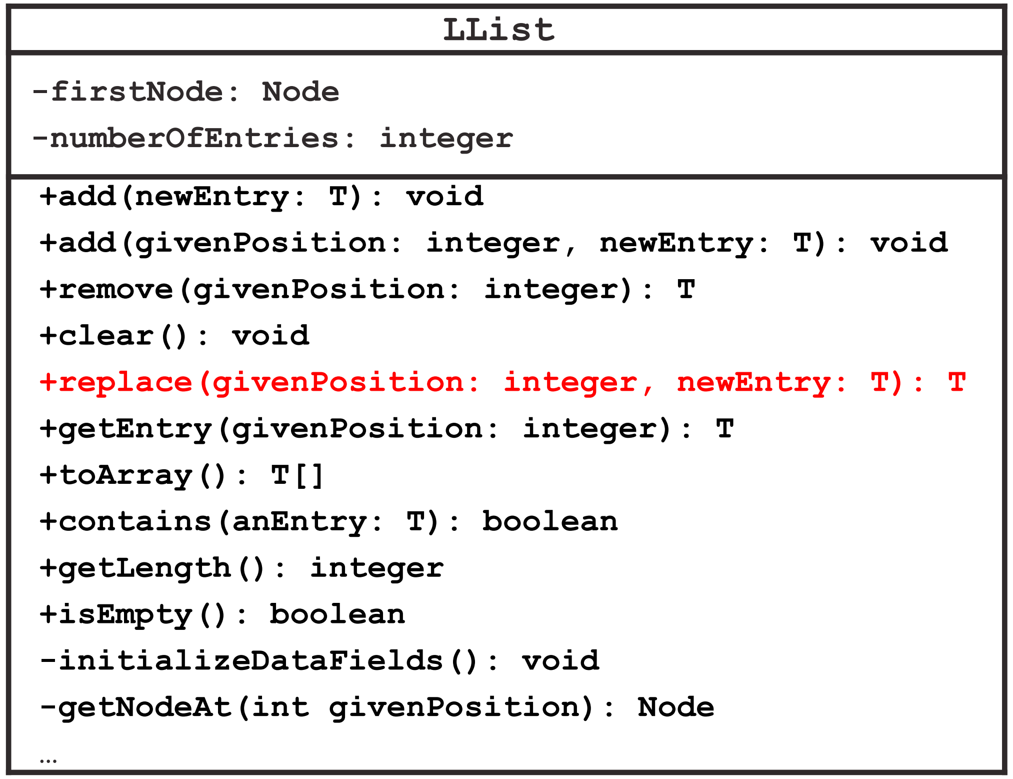
- The **getEntry** method:

```
public T getEntry(int givenPosition)
  {
     if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
     {
        // Assertion: !isEmpty()
        return getNodeAt(givenPosition).getData();
     }
     else
        throw new IndexOutOfBoundsException("Illegal position given to
           getEntry operation.");
  } // end getEntry
```

```
LList
-firstNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
-getNodeAt(int givenPosition): Node
…
```

In-Class Exercises

- The **replace** method: Replacing a list entry



```
LList
-firstNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
-getNodeAt(int givenPosition): Node
…
```

In-Class Exercises

- The **replace** method: Replacing a list entry

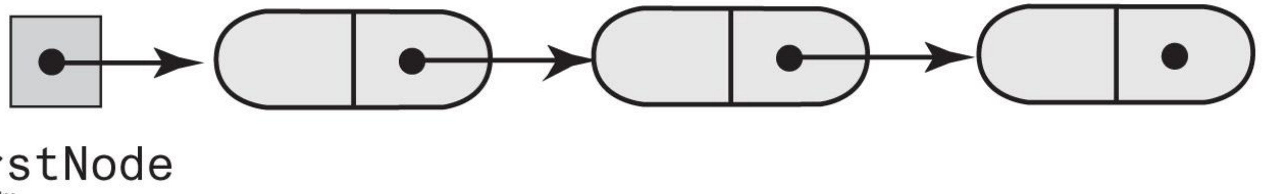
```
public T replace(int givenPosition, T newEntry)
{
   if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
   {
      // Assertion: !isEmpty()
      Node desiredNode = getNodeAt(givenPosition);
      T originalEntry = desiredNode.getData();
      desiredNode.setData(newEntry);
      return originalEntry;
   }
   else
      throw new IndexOutOfBoundsException("Illegal position given to
         replace operation.");
} // end replace
```

LList
-firstNode: Node -numberOfEntries: integer
+add(newEntry: T): void +add(givenPosition: integer, newEntry: T): void +remove(givenPosition: integer): T +clear(): void +replace(givenPosition: integer, newEntry: T): T +getEntry(givenPosition: integer): T +toArray(): T[] +contains(anEntry: T): boolean +getLength(): integer +isEmpty(): boolean -initializeDataFields(): void -getNodeAt(int givenPosition): Node …

A Refined Implementation (Tail Reference)

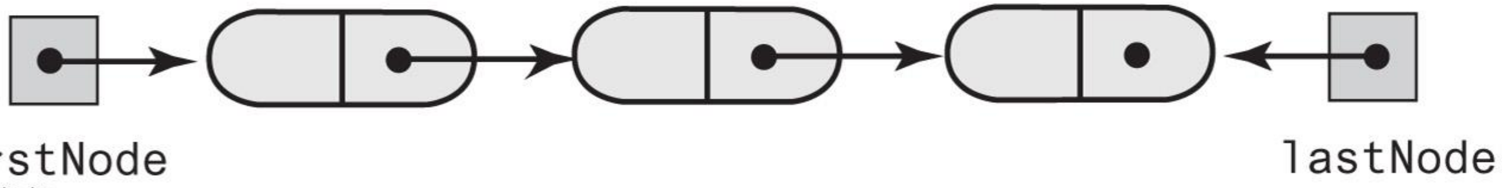
- Previous implementation

(a) With only a head reference



- Refined implementation

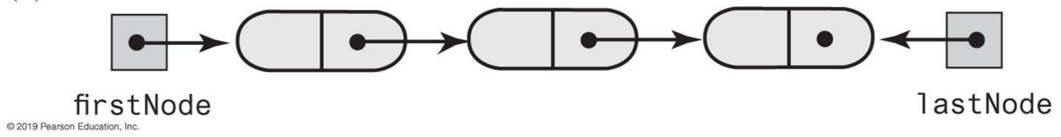
(b) With both a head reference and a tail reference



A Refined Implementation (Tail Reference)

- Using Tail Reference

(b) With both a head reference and a tail reference



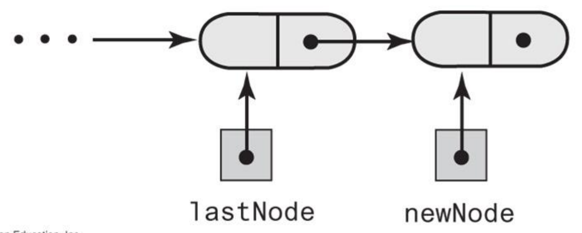
```
private void initializeDataFields()
{
        firstNode = null;
        lastNode = null;
        numberOfEntries = 0;
} // end initializeDataFields
```

```
LList
-firstNode: Node
-lastNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
-getNodeAt(int givenPosition): Node
…
```

A Refined Implementation (Tail Reference)

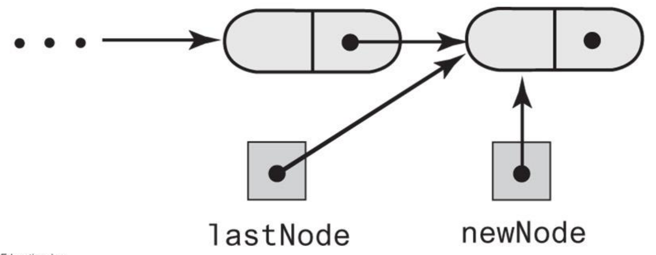
- Adding a node to the end of a nonempty chain that has a tail reference

(a)



After executing
`lastNode.setNextNode(newNode);`

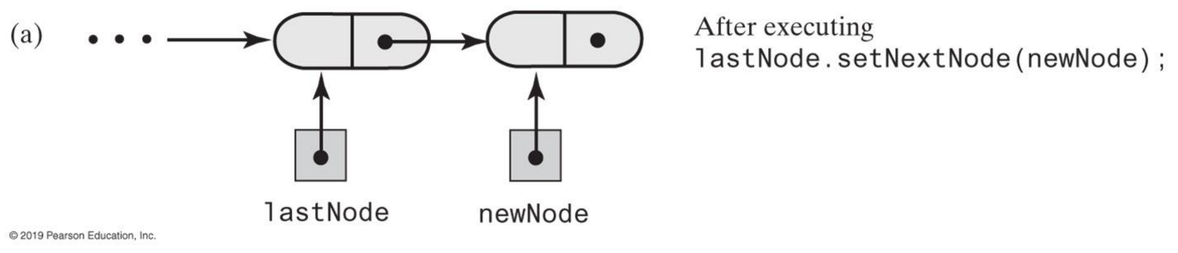
(b)

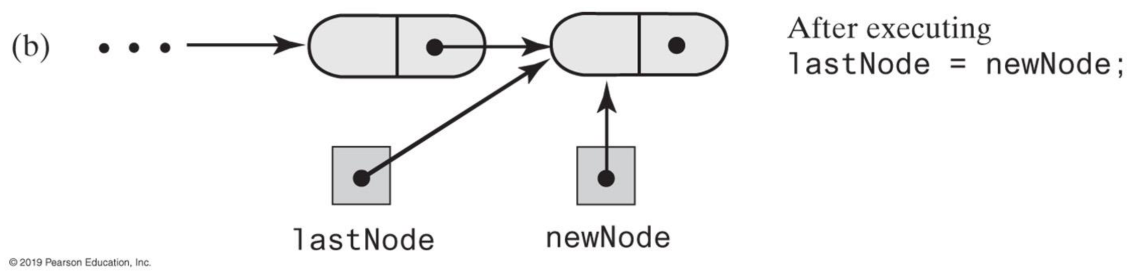


After executing
`lastNode = newNode;`

A Refined Implementation (Tail Reference)

- Adding a node to the end of a nonempty chain that has a tail reference

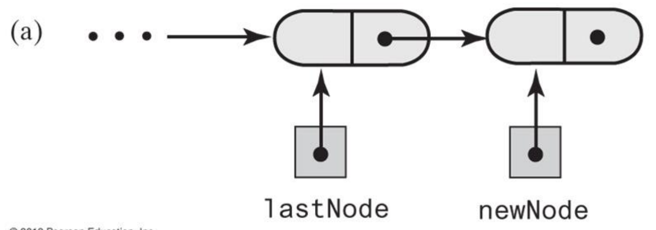

(a)
After executing
lastNode.setNextNode(newNode);
lastNode
newNode
© 2019 Pearson Education, Inc.



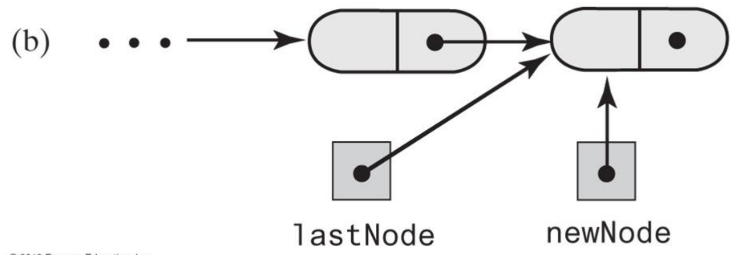
```
public void add(T newEntry)
{
   Node newNode = new Node(newEntry);

   if (isEmpty())
      firstNode = newNode;
   else
      lastNode.setNextNode(newNode);

   lastNode = newNode;

   numberOfEntries++;
} // end add
```

A Refined Implementa

- Adding to the list at a given position



After executing
`lastNode.setNextNode(newNode);`

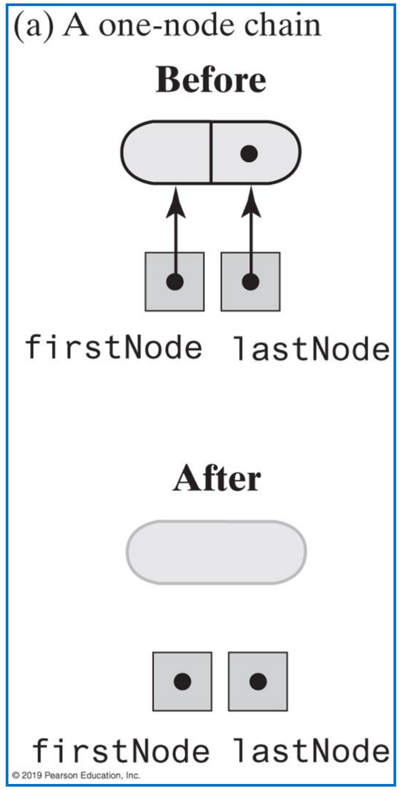


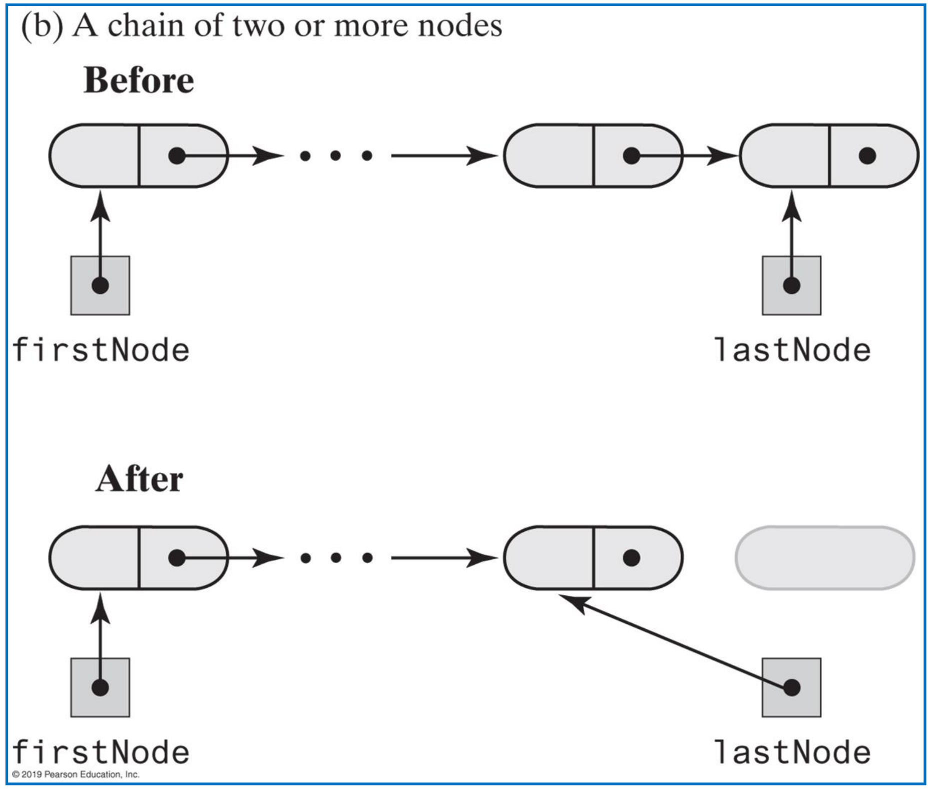
After executing
`lastNode = newNode;`

```
public boolean add(int newPosition, T newEntry)
{
   boolean isSuccessful = true;
   if ((newPosition >= 1) && (newPosition <= numberOfEntries + 1))
   {
      Node newNode = new Node(newEntry);
      if (isEmpty())
      {
         firstNode = newNode;
         lastNode = newNode;
      }
      else if (newPosition == 1)
      {
         newNode.setNextNode(firstNode);
         firstNode = newNode;
      }
      else if (newPosition == numberOfEntries + 1)
      {
         lastNode.setNextNode(newNode);
         lastNode = newNode;
      }
      else
      {
         Node nodeBefore = getNodeAt(newPosition - 1);
         Node nodeAfter = nodeBefore.getNextNode();
         newNode.setNextNode(nodeAfter);
         nodeBefore.setNextNode(newNode);
      } // end if
      numberOfEntries++;
   }
   else
      isSuccessful = false;
   return isSuccessful;
} // end add
```

A Refined Implementation (Tail Reference)

- Removing a node from a list that has a tail reference

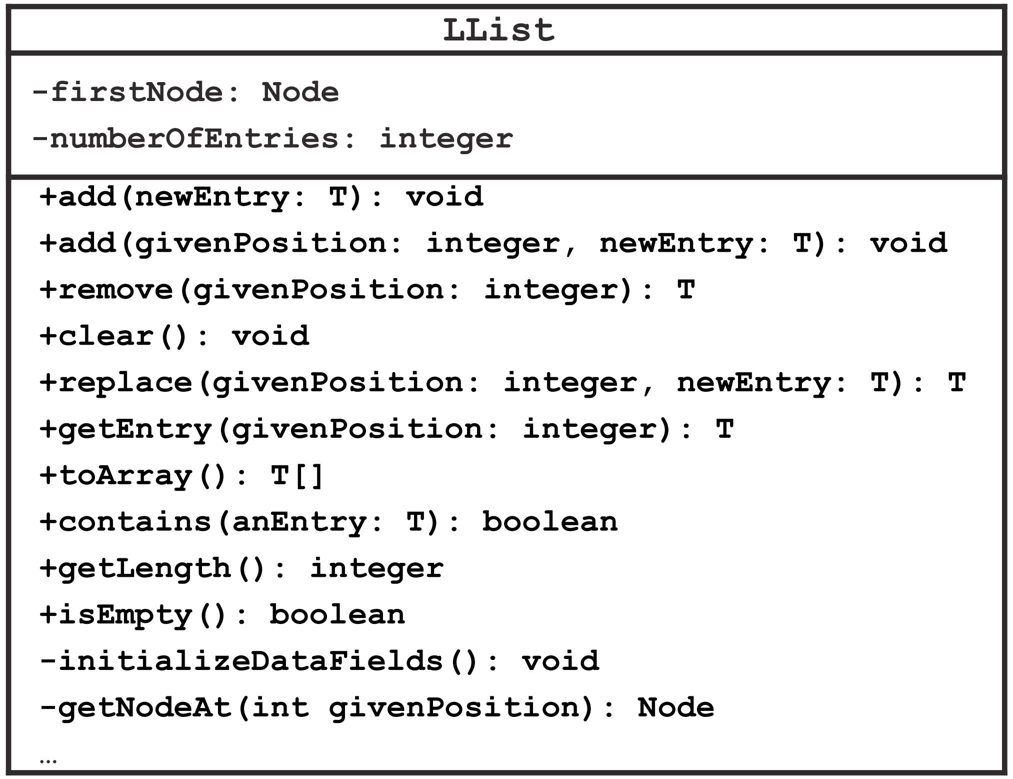



(b) A chain of two or more nodes
Before
firstNode
lastNode
After
firstNode
lastNode
© 2019 Pearson Education, Inc.

```
public T remove(int givenPosition)
{
   T result = null;                           // return value

   if ((givenPosition >= 1) && (givenPosition <= numberOfEntries))
   {
      assert !isEmpty();
      if (givenPosition == 1)                 // case 1: remove first entry
      {
         result = firstNode.getData();        // save entry to be removed
         firstNode = firstNode.getNextNode();
         if (numberOfEntries == 1)
            lastNode = null;                  // solitary entry was removed
      }
      else                                    // case 2: not first entry
      {
         Node nodeBefore = getNodeAt(givenPosition - 1);
         Node nodeToRemove = nodeBefore.getNextNode();
         Node nodeAfter = nodeToRemove.getNextNode();
         nodeBefore.setNextNode(nodeAfter);
         result = nodeToRemove.getData();     // save entry to be removed

         if (givenPosition == numberOfEntries)
            lastNode = nodeBefore;            // last node was removed
      } // end if

      numberOfEntries--;
   } // end if

   return result;                             // return removed entry, or
                                              // null if operation fails
} // end remove
```

In-Class Exercises: Algorithm Analysis

- What is the **Big Oh** of each list method in the best case and the worst case?



```
LList
-firstNode: Node
-numberOfEntries: integer
+add(newEntry: T): void
+add(givenPosition: integer, newEntry: T): void
+remove(givenPosition: integer): T
+clear(): void
+replace(givenPosition: integer, newEntry: T): T
+getEntry(givenPosition: integer): T
+toArray(): T[]
+contains(anEntry: T): boolean
+getLength(): integer
+isEmpty(): boolean
-initializeDataFields(): void
-getNodeAt(int givenPosition): Node
…
```

Algorithm Analysis on Three Implementations

Operation	Alist	LList	LListWithTail
`add(newEntry)`	O(1)	O(*n*)	O(1)
`add(givenPosition, newEntry)`	O(*n*); O(n); O(1)	O(1); O(*n*)	O(1); O(*n*); O(1)
`toArray()`	O(*n*)	O(*n*)	O(*n*)
`remove(givenPosition)`	O(*n*); O(*n*); O(1)	O(1); O(*n*)	O(1); O(*n*)
`replace(givenPosition, newEntry)`	O(1)	O(1); O(*n*)	O(1); O(*n*); O(1)
`getEntry(givenPosition)`	O(1)	O(1); O(*n*)	O(1); O(*n*); O(1)
`contains(anEntry)`	O(*n*)	O(*n*)	O(*n*)
`clear(), getLength(), isEmpty()`	O(1)	O(1)	O(1)

Java Class Library: The Class LinkedList

- Implements the interface **`List`**

- **`LinkedList`** defines more methods than are in the interface `List`

- You can use the class **`LinkedList`** as implementation of ADT
  - **queue**
  - **deque**
  - or **list**.

Summary

- Lists
- Implementations of a List

What I Want You to Do

- Review class slides
- Review Chapters 10, 11, and 12

- Next Topics
  - ADT Stack
  - Implementations of a Stack