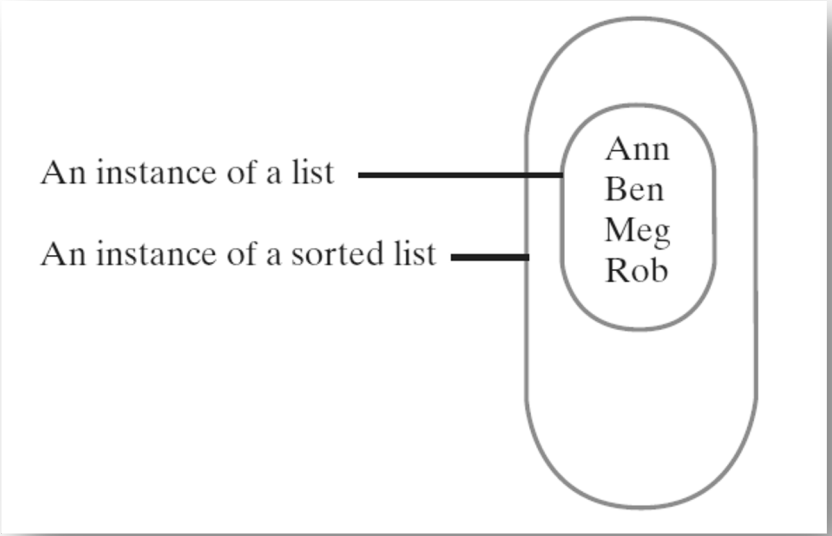
Implementing using Composition

Implementation That Uses the ADT List



- An instance of a sorted list that contains a list of its entries.
- Composition, the list is a field variable in the sorted list.
- Can also think of this as an example of a Wrapper Class or an Adapter Design Pattern

Implementation That Uses the ADT List

```
public class SortedList<T extends Comparable<? super T>>
              implements SortedListInterface<T>
{
   private ListInterface<T> list;
   public SortedList()
   {
      list = new LList<>();
   } // end default constructor
      . . .
} // end SortedList
```

LList is encapsulated inside the SortedList

Our class **`SortedList`** will implement the interface **`SortedListInterface`**

Remember the SortedListInterface

Specifications for ADT Sorted List as an interface

```
/** An interface for the ADT sorted list.
    Entries in the list have positions that begin with 1.
    @author Frank M. Carrano
/*
public interface SortedListInterface<T extends Comparable<? super T>>
{
   /** Adds a new entry to this sorted list in its proper order.
       The list's size is increased by 1.
       @param newEntry  The object to be added as a new entry. */
   public void add(T newEntry);

   /** Removes the first or only occurrence of a specified entry
       from this sorted list.
       @param anEntry  The object to be removed.
       @return  True if anEntry was located and removed; */
                otherwise returns false. */
   public boolean remove(T anEntry);

   /** Gets the position of an entry in this sorted list.
       @param anEntry  The object to be found.
       @return  The position of the first or only occurrence of anEntry
```

LISTING 16-1 The interface `SortedListInterface`

Specifications for ADT Sorted List as an interface

```
   public boolean remove(T anEntry);

   /** Gets the position of an entry in this sorted list.
       @param anEntry  The object to be found.
       @return  The position of the first or only occurrence of anEntry
                if it occurs in the list; otherwise returns the position
                where anEntry would occur in the list, but as a negative
                integer. */
   public int getPosition(T anEntry);

   // The following methods are described in Segment 12.9 of Chapter 12
   // as part of the ADT list:

   public T getEntry(int givenPosition);
   public boolean contains(T anEntry);
   public T remove(int givenPosition);
   public void clear();
   public int getLength();
   public boolean isEmpty();
   public T[] toArray();
} // end SortedListInterface
```

LISTING 16-1 The interface **`SortedListInterface`**

Implementation That Uses the ADT List

```
public void add(T newEntry)
{
   int newPosition = Math.abs(getPosition(newEntry));
   list.add(newPosition, newEntry);
} // end add
```

The method **add**.

Implementation That Uses the ADT List

```
public boolean remove(T anEntry)
{
   boolean result = false;
   int position = getPosition(anEntry);

   if (position > 0)
   {
      list.remove(position);
      result = true;
   } // end if

   return result;
} // end remove
```

The method **remove**.

Implementation That Uses the ADT List

```
public int getPosition(T anEntry)
{
   int position = 1;
   int length = list.getLength();

   // Find position of anEntry
   while ( (position <= length) &&
           (anEntry.compareTo(list.getEntry(position)) > 0) )
   {
      position++;
   } // end while

   // See whether anEntry is in list
   if ( (position > length) ||
        (anEntry.compareTo(list.getEntry(position)) != 0) )
   {
      position = -position; // anEntry is not in list
   } // end if

   return position;
} // end getPosition
```

The implementation of **`getPosition`**

Remember: Managing Sorting at List Implementation Level

ADT List Operation	Array	Linked
`getEntry(givenPosition)`	$O(1)$	$O(n)$
`add(newPosition, newEntry)`	$O(n)$	$O(n)$
`remove(givenPosition)`	$O(n)$	$O(n)$
`contains(anEntry)`	$O(n)$	$O(n)$
`toArray()`	$O(n)$	$O(n)$
`clear(), getLength(), isEmpty()`	$O(1)$	$O(1)$

FIGURE 16-8 The worst-case efficiencies of selected ADT list operations for array-based and linked implementations

Implementation That Uses the ADT List

ADT Sorted List Operation	List Implementation	
	Array	Linked
`add(new Entry)`	$O(n)$	$O(n^2)$
`remove(anEntry)`	$O(n)$	$O(n^2)$
`getPosition(anEntry)`	$O(n)$	$O(n^2)$
`getEntry(givenPosition)`	$O(1)$	$O(n)$
`contains(anEntry)`	$O(n)$	$O(n)$
`remove(givenPosition)`	$O(n)$	$O(n)$
`toArray()`	$O(n)$	$O(n)$
`clear(), getLength(), isEmpty()`	$O(1)$	$O(1)$

FIGURE 16-9 The worst-case efficiencies of the ADT sorted list operations when implemented using an instance of the ADT list