Implementing a Sorted List through Inheritance

Another Approach to Implementing Sorted List

Inheritance

- Sorted list **is-a** list
- Inherit methods from parent class
- Must override methods that don't keep the list sorted (replace or add to the end)
- Must add methods that preserve the order of the sort (adding an entry in correct sorted position)

Inheritance to Implement a Sorted List

```
public class SortedList<T extends Comparable<? super T>>
             extends LList<T> implements SortedListInterface<T>
{
   public void add(T newEntry)
   {
      int newPosition = Math.abs(getPosition(newEntry));
      super.add(newPosition, newEntry);
   } // end add

   < Implementations of remove(anEntry) and getPosition(anEntry) go here. >
   . . .
} // end SortedList
```

If **`SortedList`** inherited methods from **`LList`**, we would not have to implement them again.

Inheritance to Implement a Sorted List

```
/** Adds newEntry to the list at position newPosition. */
public void add(int newPosition, T newEntry);

/** Replaces the entry at givenPosition with newEntry. */
public T replace(int givenPosition, T newEntry);
```

Although **`SortedList`** conveniently inherits methods such as **`isEmpty`** from **`LList`**, it also inherits two methods that a client can use to destroy the order of a sorted list.

Designing a Base Class

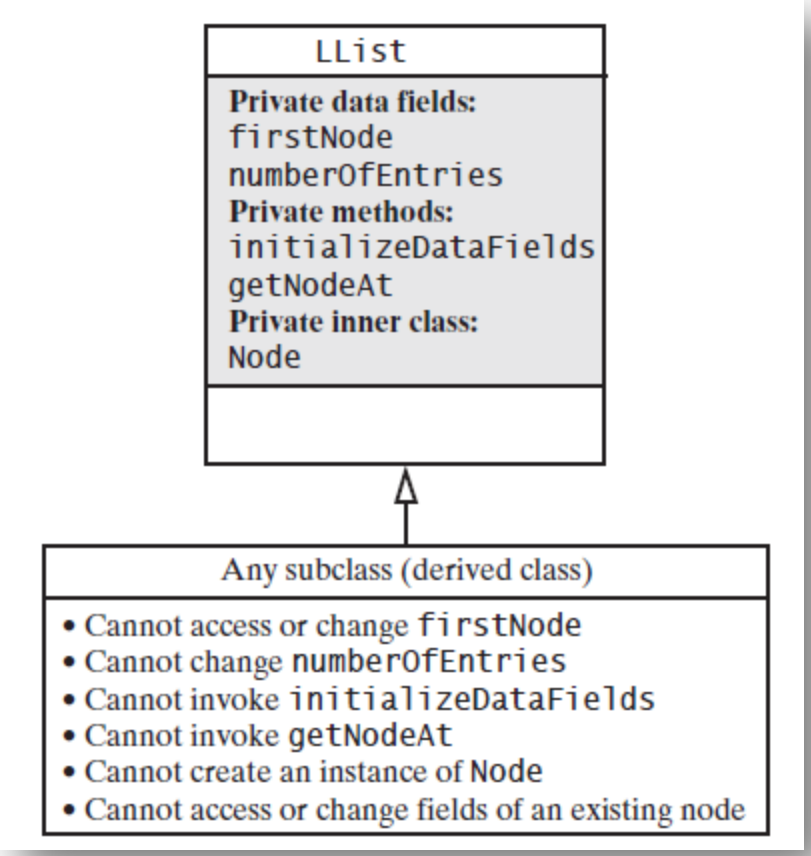
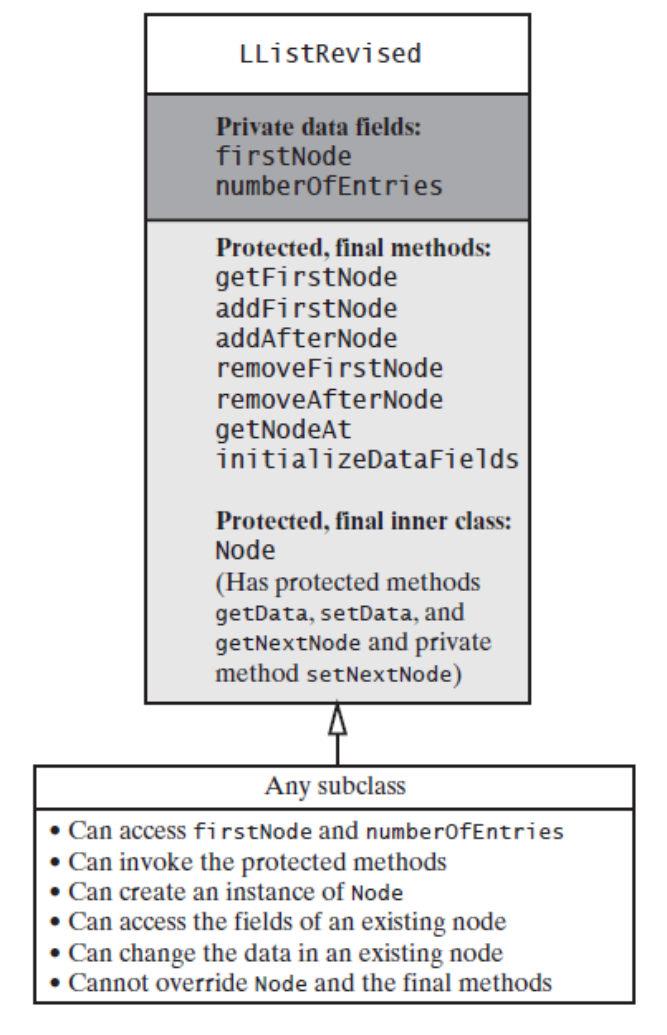


FIGURE 17-1 A derived class of the original class **`LList`** cannot access or change anything that is private within **`LList`**

Designing a Base Class

FIGURE 17-2 Access available to a class derived from the class **LListRevised**



Creating an Abstract Base Class

LISTING 17-2 The abstract base class LinkedChainBase

```
public abstract class LinkedChainBase<T>
{
   private Node firstNode; // Reference to first node
   private int numberOfEntries;

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

   < Implementations of the public methods clear, getLength, isEmpty, and toArray go here. >
   . . .
   < Implementations of the protected, final methods getNodeAt, getFirstNode, addFirstNode,
     addAfterNode, removeFirstNode, removeAfterNode, and initializeDataFields
     go here. >
   . . .
   protected final class Node
   {
      private T data;    // Entry in list
      private Node next; // Link to next node

      protected Node(T dataPortion)
```

LISTING 17-2 The abstract base class **`LinkedChainBase`**

Creating an Abstract Base Class

```
18        private Node next; // Link to next node
19
20        protected Node(T dataPortion)
21        {
22           data = dataPortion;
23           next = null;
24        } // end constructor
25
26        private Node(T dataPortion, Node nextNode)
27        {
28           data = dataPortion;
29           next = nextNode;
30        } // end constructor
31        < Implementations of the protected methods getData, setData, and getNextNode go here. >
32        . . .
33        < Implementation of the private method setNextNode goes here. >
34        . . .
35     } // end Node
36  } // end LinkedChainBase
```

LISTING 17-2 The abstract base class **`LinkedChainBase`**

Creating an Abstract Base Class

```
public class LinkedChainList<T> extends LinkedChainBase<T>
                                implements ListInterface<T>
{
   public LinkedChainList()
   {
      super(); // Initializes the linked chain
   } // end default constructor

  < Implementations of the public methods add, remove, replace, getEntry, and contains
    go here. >
   . . .
} // end LinkedChainList
```

LISTING 17-3 A revision of **`LListRevised`** that extends **`LinkedChainBase`**

Efficient Implementation of a Sorted List

```
public class LinkedSortedList<T extends Comparable<? super T>>
             extends LinkedChainBase<T>
             implements SortedListInterface<T>
```

We want our class to extend **`LinkedChainBase`**

Efficient Implementation of a Sorted List

```
public void add(T newEntry)
{
   Node newNode = new Node(newEntry);
   Node nodeBefore = getNodeBefore(newEntry);
   if (nodeBefore == null) // No need to call isEmpty
      addFirstNode(newNode);
   else
      addAfterNode(nodeBefore, newNode);
} // end add
```

Details of a previous **`add`** method now hidden within the protected methods **`addFirstNode`** and **`addAfterNode`** of **`LinkedChainBase`**

Efficient Implementation of a Sorted List

```
private Node getNodeBefore(T anEntry)
{
   Node currentNode = getFirstNode();
   Node nodeBefore = null;

   while ((currentNode != null) &&
          (anEntry.compareTo(currentNode.getData()) > 0))
   {
      nodeBefore = currentNode;
      currentNode = currentNode.getNextNode();
   } // end while

   return nodeBefore;
} // end getNodeBefore
```

The private method **`getNodeBefore`**