```java
//--------------------------------------------------------------------------
// ArrayUnsortedList.java        by Dale/Joyce/Weems                Chapter 6
//
// Implements the ListInterface using an array.
//
// Null elements are not permitted on a list.
//
// Two constructors are provided: one that creates a list of a default
// original capacity, and one that allows the calling program to specify the
// original capacity.
//--------------------------------------------------------------------------

package lists;

public class ArrayUnsortedList<T> implements ListInterface<T>
{
  protected final int DEFCAP = 100; // default capacity
  protected int origCap;            // original capacity
  protected T[] list;               // array to hold this list's elements
  protected int numElements = 0;    // number of elements in this list
  protected int currentPos;         // current position for iteration

  // set by find method
  protected boolean found;  // true if element found, otherwise false
  protected int location;   // indicates location of element if found

  public ArrayUnsortedList()
  {
    list = (T[]) new Object[DEFCAP];
    origCap = DEFCAP;
  }

  public ArrayUnsortedList(int origCap)
  {
    list = (T[]) new Object[origCap];
    this.origCap = origCap;
  }

  protected void enlarge()
  // Increments the capacity of the list by an amount
  // equal to the original capacity.
  {
    // Create the larger array.
    T[] larger = (T[]) new Object[list.length + origCap];

    // Copy the contents from the smaller array into the larger array.
    for (int i = 0; i < numElements; i++)
    {
      larger[i] = list[i];
    }

    // Reassign list reference.
    list = larger;
  }

  protected void find(T target)
  // Searches list for an occurence of an element e such that
```

```
58  // e.equals(target). If successful, sets instance variables
59  // found to true and location to the array index of e. If
60  // not successful, sets found to false.
61  {
62    location = 0;
63    found = false;
64
65    while (location < numElements)
66    {
67      if (list[location].equals(target))
68      {
69        found = true;
70        return;
71      }
72      else
73        location++;
74    }
75  }
76
77  public void add(T element)
78  // Adds element to this list.
79  {
80    if (numElements == list.length)
81      enlarge();
82    list[numElements] = element;
83    numElements++;
84  }
85
86  public boolean remove (T element)
87  // Removes an element e from this list such that e.equals(element)
88  // and returns true; if no such element exists, returns false.
89  {
90    find(element);
91    if (found)
92    {
93      list[location] = list[numElements - 1];
94      list[numElements - 1] = null;
95      numElements--;
96    }
97    return found;
98  }
99
100 public int size()
101 // Returns the number of elements on this list.
102 {
103   return numElements;
104 }
105
106 public boolean contains (T element)
107 // Returns true if this list contains an element e such that
108 // e.equals(element); otherwise, returns false.
109 {
110   find(element);
111   return found;
112 }
113
114 public T get(T element)
```

```
115   // Returns an element e from this list such that e.equals(element);
116   // if no such element exists, returns null.
117   {
118     find(element);
119     if (found)
120       return list[location];
121     else
122       return null;
123   }
124
125   public String toString()
126   // Returns a nicely formatted string that represents this list.
127   {
128     String listString = "List:\n";
129     for (int i = 0; i < numElements; i++)
130       listString = listString + "   " + list[i] + "\n";
131     return listString;
132   }
133
134   public void reset()
135   // Initializes current position for an iteration through this list,
136   // to the first element on this list.
137   {
138     currentPos  = 0;
139   }
140
141   public T getNext()
142   // Preconditions: The list is not empty
143   //                The list has been reset
144   //                The list has not been modified since the most recent reset
145   //
146   // Returns the element at the current position on this list.
147   // If the current position is the last element, it advances the value
148   // of the current position to the first element; otherwise, it advances
149   // the value of the current position to the next element.
150   {
151     T next = list[currentPos];
152     if (currentPos == (numElements - 1))
153       currentPos = 0;
154     else
155       currentPos++;
156     return next;
157   }
158
159   // Checks if list is empty
160   // Returns boolean value
161   public boolean isEmpty() {
162       return this.size() == 0;
163   }
164
165   // Removes all list items by traversing through list
166   // and individually removing each list item and decrementing
167   // until the size of the list is 0.
168   // If the list is null or empty, throw a NullPointerException.
169   public void removeAll() {
170       if (!this.isEmpty())
171           {
```

```
172           currentPos = 0;
173           while(!this.isEmpty()) {
174               if (list[currentPos] != null)
175                   remove(list[currentPos]);
176               else
177                   numElements--;
178               }
179           }
180       else
181           throw new NullPointerException();
182       }
183
184 }
185
```