```
/** Chapter No. 15 - Exercise No. 6
   File Name:          LinkedList3.java
   Programmer:         Broderick Higby
   Date Last Modified: December 4, 2015
 *Algorithm:
 *1. Construct a linked list object of type Integer
 *2. Check to make sure the list is empty
 *3. Add each integer to the list individually
 *4. Then output the added integer to the linked list
 *4. Make sure the Linked list is empty for the Strings
 *5. Add each string to the linked list individually
 *6. Output the linked list with each individual value added
 *7.  Set the head node to null at the end of the list output
*/
public class LinkedListTester
{
    public static void main(String[] args)
    {
                    //Create new LinkedList of Integer values
                    LinkedList3<Integer> listInt = new LinkedList3();

                    //Numbers to test addSort
                    int num  = 4;
                    int num2 = 3;
                    int num3 = 7;

                    //Test if list is empty
                    System.out.println("Is list empty: ");
                    System.out.println(listInt.isEmpty());

                    //Add Values to list
                    System.out.println("Adding Values to List");
                    listInt.addToStart(6);
                    listInt.addToStart(5);
                    listInt.addToStart(2);
                    listInt.addToStart(1);
                    System.out.println("Output list with added values: ");
                    listInt.outputList();
                    System.out.println("Run addSort Method on integer value 4");
                    listInt.addSort(num);
                    System.out.println("Output updated list after add: ");
                    listInt.outputList();
                    System.out.println("Run addSort Method on integer value 3");
                    listInt.addSort(num2);
                    System.out.println("Output updated list after add: ");
                    listInt.outputList();
                    System.out.println("Run addSort Method on integer value 7");
                    listInt.addSort(num3);
                    System.out.println("Output updated list after add: ");
```

```
                listInt.outputList();

                //Create new LinkedList of Integer values
                LinkedList3<String> listStr = new LinkedList3();

                //Numbers to test addSort
                String word1 = "dog";
                String word2 = "fox";
                String word3 = "fish";
                String word4 = "fish";

                //Test if list is empty
                System.out.println("Is list empty: ");
                System.out.println(listStr.isEmpty());

                //Add Values to list
                System.out.println("Adding Values to List");
                listStr.addToStart("elephant");
                listStr.addToStart("cougar");
                listStr.addToStart("bee");
                listStr.addToStart("antelope");

                //Output Current List
                System.out.println("Output list with added values: ");
                listStr.outputList();
                System.out.println("Run addSort Method on String value dog");
                listStr.addSort(word1);
                System.out.println("Output updated list after add: ");
                listStr.outputList();
                System.out.println("Run addSort Method on String value fox");
                listStr.addSort(word2);
                System.out.println("Output updated list after add: ");
                listStr.outputList();
                System.out.println("Run addSort Method on String value fish");
                listStr.addSort(word3);
                System.out.println("Output updated list after add: ");
                listStr.outputList();
                System.out.println("Run addSort Method on String value");
                listStr.addSort(word4);
                System.out.println("Output updated list after add: ");
                listStr.outputList();
    }
}
public class LinkedList3<T extends Comparable>
{
    private class Node<T>
    {
        private T data;
        private Node<T> link;
```

```
    public Node()
    {
         data = null;
         link = null;
    }

    public Node(T newData, Node<T> linkValue)
    {
        data = newData;
        link = linkValue;
    }
 }//End of Node<T> inner class

private Node<T> head;

public LinkedList3( )
{
    head = null;
}

/**
 Adds a node at the start of the list with the specified data.
 The added node will be the first node in the list.
*/
public void addToStart(T itemData)
{
    head = new Node<T>(itemData, head);
}

/**
 Removes the head node and returns true if the list contains at least
 one node. Returns false if the list is empty.
*/
public boolean deleteHeadNode( )
{
    if (head != null)
    {
        head = head.link;
        return true;
    }
    else
        return false;
}

/**
 Returns the number of nodes in the list.
*/
public int size( )
```

```
{
    int count = 0;
    Node<T> position = head;
    while (position != null)
    {
        count++;
        position = position.link;
    }
    return count;
}

public boolean contains(T item)
{
    return (find(item) != null);
}

/**
 Finds the first node containing the target item, and returns a
 reference to that node. If target is not in the list, null is returned.
*/
private Node<T> find(T target)
{
    Node<T> position = head;
    T itemAtPosition;
    while (position != null)
    {
        itemAtPosition = position.data;
        if (itemAtPosition.equals(target))
            return position;
        position = position.link;
    }
    return null; //target was not found
}

/**
 Finds the first node containing the target and returns a reference
  to the data in that node. If target is not in the list, null is returned.
*/
public T findData(T target)
{
    return find(target).data;
}

public void outputList( )
{
    Node<T> position = head;
    while (position != null)
    {
        System.out.println(position.data);
```

```
        position = position.link;
    }
}

public boolean isEmpty( )
{
    return (head == null);
}

public void clear( )
{
    head = null;
}

/*
 For two lists to be equal they must contain the same data items in
 the same order. The equals method of T is used to compare data items.
*/
public boolean equals(Object otherObject)
{
    if (otherObject == null)
        return false;
    else if (getClass( ) != otherObject.getClass( ))
        return false;
    else
    {
        LinkedList3<T> otherList = (LinkedList3<T>)otherObject;
        if (size( ) != otherList.size( ))
            return false;
        Node<T> position = head;
        Node<T> otherPosition = otherList.head;
        while (position != null)
        {
            if (!(position.data.equals(otherPosition.data)))
                return false;
            position = position.link;
            otherPosition = otherPosition.link;
        }
        return true; //no mismatch was not found
    }
}



        public void addSort(T item)
        {
                Node<T> position = head;
                Node<T> previousPosition = null;
                Node<T> newNode;
                int added = 0;
```

```
                T itemAtPosition;

                while(position != null)
                {
                        itemAtPosition = position.data;
                        System.out.println("Current Node: " + itemAtPosition);
                        System.out.println("Comparable Interface Value: " +
                                                                item.compareTo(itemAtPosition));

                        if(item.compareTo(itemAtPosition) <= 0 && added == 0)
                        {
                                newNode = new Node(item, previousPosition.link);
                                previousPosition.link = newNode;
                                added++;
                        }
                        previousPosition = position;
                        position = position.link;
                }
                if(added == 0)
                {
                        newNode = new Node(item, previousPosition.link);
                        previousPosition.link = newNode;
                }
        }
}
```

<terminated> LinkedListTester (1) [Java Application] /Library/Java/JavaVirtualMachir

```
Is list empty:
true
Adding Values to List
Output list with added values:
1
2
5
6
Run addSort Method on integer value 4
Current Node: 1
Comparable Interface Value: 1
Current Node: 2
Comparable Interface Value: 1
Current Node: 5
Comparable Interface Value: -1
Current Node: 6
Comparable Interface Value: -1
Output updated list after add:
1
2
4
5
6
Run addSort Method on integer value 3
Current Node: 1
Comparable Interface Value: 1
Current Node: 2
Comparable Interface Value: 1
Current Node: 4
Comparable Interface Value: -1
Current Node: 5
Comparable Interface Value: -1
Current Node: 6
Comparable Interface Value: -1
```

<terminated> LinkedListTester (1) [Java Application] /Library/Java/JavaVirtualM

```
bee
cougar
dog
elephant
fox
Run addSort Method on String value fish
Current Node: antelope
Comparable Interface Value: 5
Current Node: bee
Comparable Interface Value: 4
Current Node: cougar
Comparable Interface Value: 3
Current Node: dog
Comparable Interface Value: 2
Current Node: elephant
Comparable Interface Value: 1
Current Node: fox
Comparable Interface Value: -6
Output updated list after add:
antelope
bee
cougar
dog
elephant
fish
fox
Run addSort Method on String value
Current Node: antelope
Comparable Interface Value: 5
Current Node: bee
Comparable Interface Value: 4
Current Node: cougar
Comparable Interface Value: 3
Current Node: dog
```

```
Comparable Interface Value: -1
Output updated list after add:
1
2
3
4
5
6
Run addSort Method on integer value 7
Current Node: 1
Comparable Interface Value: 1
Current Node: 2
Comparable Interface Value: 1
Current Node: 3
Comparable Interface Value: 1
Current Node: 4
Comparable Interface Value: 1
Current Node: 5
Comparable Interface Value: 1
Current Node: 6
Comparable Interface Value: 1
Output updated list after add:
1
2
3
4
5
6
7
Is list empty:
true
Adding Values to List
```

<terminated> LinkedListTester (1) [Java Application] /Library/Java/JavaVirtualMa

```
Output list with added values:
antelope
bee
cougar
elephant
Run addSort Method on String value dog
Current Node: antelope
Comparable Interface Value: 3
Current Node: bee
Comparable Interface Value: 2
Current Node: cougar
Comparable Interface Value: 1
Current Node: elephant
Comparable Interface Value: -1
Output updated list after add:
antelope
bee
cougar
dog
elephant
Run addSort Method on String value fox
Current Node: antelope
Comparable Interface Value: 5
Current Node: bee
Comparable Interface Value: 4
Current Node: cougar
Comparable Interface Value: 3
Current Node: dog
Comparable Interface Value: 2
Current Node: elephant
Comparable Interface Value: 1
Output updated list after add:
antelope
bee
```