/* Author: Richard Eisenberg */

import java.util.*;

abstract class ListNode<A>
{
    public Nil() {
    }

    public String toString() {
        return "[";
    }
}

class Nil<A> extends ListNode<A>
{
    public Nil() {
    }

    public String toString() {
        return "[]";
    }
}

class Cons<A>
{

    public final A head;
    public final ListNode<A> tail;

    public Cons(A x, ListNode<A> xs) {
        head = x;
        tail = xs;
    }

    public String toString() {
        return head.toString() + ":{" + tail.toString() + "}";
    }
}

public class List
{
    public static <A> int length(ListNode<A> list) {
        // This 'if' is like Haskell’s pattern-match construct
        if(list instanceof Nil) {
            return 0;
        }
        else {
            final Cons<A> cons = (Cons<A>)list;
            return 1 + length(cons.tail);
        }
    }

    public static ListNode<Integer> increment(ListNode<Integer> list) {
        if(list instanceof Nil) {
            return new Nil<>();
        }
        else {
            final Cons<Integer> cons = (Cons<Integer>)list;
            return cons.head + cons.tail;
        }
    }
}
List.java

```java
73:     {
74:         final Cons<Integer> cons = (Cons<Integer>)list;
75:         return new Cons<>(cons.head + 1, increment(cons.tail));
76:     }
77: }
78: // Insertion sort requires the ability to compare elements. Java supports
79: // *bounded polymorphism* to accomplish this. It’s like Haskell’s class
80: // constraint mechanism.
81: public static <A extends Comparable<A>> ListNode<A> insert(A x, ListNode<A> xs)
82:     {
83:         if(xs instanceof Nil)
84:             return new Cons<>(x, new Nil<>());
85:         else
86:             final Cons<A> cons = (Cons<A>)xs;
87:             if(x.compareTo(cons.head) < 0)
88:                 return new Cons<>(x, cons);
89:             else
90:                 return new Cons<>(cons.head, insert(x, cons.tail));
91:     }
92: }
93: public static <A extends Comparable<A>> ListNode<A> insertionSort(ListNode<A> xs)
94:     {
95:         if(xs instanceof Nil)
96:             return new Nil<>();
97:         else
98:             final Cons<A> cons = (Cons<A>)xs;
99:             return insert(cons.head, insertionSort(cons.tail));
100:     }
101: }
102: public static <A> Optional<A> listToMaybe(ListNode<A> list)
103:     {
104:         if(list instanceof Nil)
105:             return Optional.empty();
106:         else
107:             final Cons<A> cons = (Cons<A>)list;
108:             return Optional.of(cons.head);
109:     }
110: }
111: // Test our functions.
112: public static void main(String[] args)
113:     {
114:         ListNode<Integer> nums = new Cons<>(8, new Cons<>(2, new Cons<>(10, new Nil<>())));
115:         System.out.println(nums);
116:         System.out.println(length(nums));
117:         System.out.println(increment(nums));
118:         System.out.println(insertionSort(nums));
119:         System.out.println(nums);
120:     }
121: ```