Exercise 10.71

```
1
2  public class Person implements Comparable<Person> {
3
4          private int age;
5
6          public Person(int age)
7          {
8                  this.age = age;
9          }
10
11         public int getAge()
12         {
13                 return age;
14         }
15
16         public int compareTo(Person obj)
17         {
18                 return age - obj.getAge();
19         }
20
21         public String toString()
22         {
23                 return age + " years";
24         }
25 }
```

Exercise 'stack'

IStack interface

```
1  package inda3;
2
3  /**
4   * A stack data structure that has O(1) time complexity on all methods.
5   * @author Artem Los
6   *
7   * @param <T> The type to be stored in the Stack.
8   */
9  public interface IStack<T> {
10
11         /**
12          * Adds the object "o" on top of stack.
13          * @param o The object to add on top of the stack.
14          */
15         void push(T o);
16
17         /**
18          * Removes and returns the the top element in the stack.
19          * @return Top element in the stack.
20          * @throws StackEmptyException.
21          */
22         T pop() throws StackEmptyException;
23
24         /**
25          * Returns the top element in the stack without removing it.
26          * @return Top element in the stack.
27          * @throws StackEmptyException.
28          */
29         T top() throws StackEmptyException;
30
31         /**
32          * The number of items in the stack (i.e. the size).
33          * @return size.
34          */
```

```
35          int size();
36
37          /**
38           * If the stack is empty, this will return true and false otherwise.
39           * @return True or false.
40           */
41          boolean isEmpty();
42  }
```

Stack class

```
1   package inda3;
2
3   import java.io.StringWriter;
4
5   public class Stack<T> implements IStack<T>
6   {
7       private ListElement<T> first;    // First element in list.
8       private ListElement<T> last;     // Last element in list.
9       private int size;                // Number of elements in list.
10
11      /**
12       * A list element.
13       */
14      private static class ListElement<T> {
15          public T data;
16          public ListElement<T> next;
17
18          public ListElement(T data) {
19              this.data = data;
20              this.next = null;
21          }
22      }
23
24
25      public Stack()
26      {
27          first=null;
28          last=null;
29          size=0;
30      }
31
32          @Override
33          /**
34           * Adds the object "o" on top of stack.
35           * @param o The object to add on top of the stack.
36           */
37          public void push(T o) {
38          ListElement<T> newElement = new ListElement<T>(o);
39
40          if(first == null)
41          {
42                  first = newElement;
43                  last = first; // switched from first=last (remember same in addLast)
44          }
45          else
46          {
47                  newElement.next = first;
48                  first = newElement;
49          }
50          size++;
51
52          }
53
54          @Override
55          /**
56           * Removes and returns the the top element in the stack.
57           * @return Top element in the stack.
```

```
58          * @throws StackEmptyException.
59          */
60         public T pop() throws StackEmptyException {
61 
62         if(size == 0 || first == null)
63                 throw new StackEmptyException();
64 
65         ListElement<T> temp = first;
66 
67         first = first.next;
68         size--;
69 
70         return temp.data;
71         }
72 
73         @Override
74         /**
75          * Returns the top element in the stack without removing it.
76          * @return Top element in the stack.
77          * @throws StackEmptyException.
78          */
79         public T top() throws StackEmptyException {
80                 // TODO Auto-generated method stub
81 
82                 if(size == 0)
83                         throw new StackEmptyException();
84                 /*
85         if(first == null)
86                 return null;
87         */
88 
89         return first.data;
90         }
91 
92         @Override
93 
94         /**
95          * The number of items in the stack (i.e. the size).
96          * @return size.
97          */
98     public int size() {
99         // TODO
100        return size;
101    }
102 
103        @Override
104        /**
105         * If the stack is empty, this will return true and false otherwise.
106         * @return True or false.
107         */
108        public boolean isEmpty() {
109                // TODO Auto-generated method stub
110                if(size == 0)
111                        return true;
112                else
113                        return false;
114        }
115 
116    /**
117     * Returns a string representation of this list. The string
118     * representation consists of a list of the elements enclosed in
119     * square brackets ("[]"). Adjacent elements are separated by the
120     * characters ", " (comma and space). Elements are converted to
121     * strings by the method toString() inherited from Object.
122     */
123    public String toString() {
124 
125        StringWriter out = new StringWriter();
126        out.write("[");
127 
```

```
128         ListElement<T> current;
129
130         current = first;
131
132         if(current == null)
133                 return "[]";
134
135         for (int i = 0; i < size-1; i++) {
136                 out.write( current.data.toString() + ",␣");
137                 current = current.next;
138
139                 }
140                 out.write(current.data.toString() + "]");
141
142         return out.toString();
143
144     }
145
146 }
```

Stack error

```
1  package inda3;
2
3  public class StackEmptyException extends Exception {
4
5          public StackEmptyException(){}
6
7          public StackEmptyException(String message)
8          {
9                  super(message);
10         }
11 }
```

MathStack class

```
1  package inda3;
2
3  public class MathStack extends Stack<Integer> {
4
5          // we could use Stack<Number> but that would require explicit definitions of
               each number type (of operators).
6          public MathStack()
7          {
8                  super();
9          }
10
11         /**
12          * Adds the recent two values in the stack.
13          * @throws StackEmptyException
14          */
15         public void add() throws StackEmptyException
16         {
17                 int a = this.pop();
18                 int b = this.pop();
19                 this.push(a+b);
20         }
21
22         /**
23          * Subtracts the recent two values in the stack.
24          * @throws StackEmptyException
25          */
26         public void sub() throws StackEmptyException
27         {
28                 int a = this.pop();
```

```
                int b = this.pop();
                this.push(b-a);
        }

        /**
         * Multiplies all values in the stack.
         * @throws StackEmptyException
         */
        public void mul() throws StackEmptyException
        {
                int a = this.pop();
                int b = this.pop();
                this.push(a*b);
        }

        /**
         * Divides the recent two values in the stack.
         * @throws StackEmptyException
         */
        public void div() throws StackEmptyException
        {
                int a = this.pop();
                int b = this.pop();
                this.push(b/a);
        }



}
```

Stack Test

```
package inda3;
import static org.junit.Assert.*;

import org.junit.Test;


public class StackTest {

        @Test
        public void PushPopTopTest() throws StackEmptyException {
                Stack<String> a = new Stack<String>();
                a.push("hi");
                a.push("hi2");
                a.push("hi3");
                a.push("hi4");

                assertEquals(a.pop(), "hi4");
                assertEquals(a.pop(), "hi3");
                assertEquals(a.pop(), "hi2");
                assertEquals(a.pop(), "hi");
                assertTrue(a.isEmpty());

                a.push("hi");
                assertEquals("hi", a.top());
                assertEquals("hi", a.top());

        }
        @Test
        public void EmptyListTest()
        {
                Stack<String> b = new Stack<String>();

                System.out.println(b.toString());
        }

        @Test
```

```
37          public void AddTest() throws StackEmptyException
38          {
39                  MathStack a = new MathStack();
40                  a.push(3);
41                  a.push(2);
42                  a.add();
43                  assertTrue(5 == a.top());
44
45                  a.push(3);
46                  a.add();
47                  assertTrue(8 == a.top());
48          }
49
50          @Test(expected=StackEmptyException.class)
51          public void PopFail() throws StackEmptyException
52          {
53                  Stack<String> a = new Stack<String>();
54                  String b = a.pop();
55          }
56
57          @Test(expected=StackEmptyException.class)
58          public void TopFail() throws StackEmptyException
59          {
60                  Stack<String> a = new Stack<String>();
61                  String b = a.top();
62          }
63
64  }
```

Exercise 'Postfix'

Postfix class

```
1   package inda3;
2
3
4   /**
5    * The Postfix class implements an evaluator for integer postfix expressions.
6    *
7    * Postfix notation is a simple way to define and write arithmetic expressions
8    * without the need for parentheses or priority rules. For example, the postfix
9    * expression "1 2 - 3 4 + *" corresponds to the ordinary infix expression
10   * "(1 - 2) * (3 + 4)". The expressions may contain decimal 32-bit integer
11   * operands and the four operators +, -, *, and /. Operators and operands must
12   * be separated by whitespace.
13   *
14   * @author  Artem Los (artem@artemlos.net)
15   * @version 2013-02-01
16   */
17  public class Postfix {
18          /**
19           * Evaluates the given postfix expression.
20           *
21           * @param expr  Arithmetic expression in postfix notation
22           * @return      The value of the evaluated expression
23           * @throws StackEmptyException
24           * @throws      A subclass of RuntimeException if the expression is wrong
25           */
26          public static int evaluate(String expr) throws InvalidExpressionException,
                StackEmptyException {
27                  // TODO
28                  //expr = expr.replace("(\\S+)?(\\t)?","");
29                  String[] tokens = expr.split("\\s+");
30
31                  MathStack stack = new MathStack();
32
33                  for (int i = 0; i < tokens.length; i++) {
```

```
34
35                      if(isInteger(tokens[i]))
36                      {
37                              try
38                              {
39                                      stack.push(Integer.parseInt(tokens[i]));
40                              }
41                              catch(Exception e)
42                              {
43                                      throw new InvalidExpressionException();
44                              }
45                      }
46                      else if(isOperator(tokens[i]))
47                      {
48                              switch (tokens[i]) {
49                              case "+":
50                                      stack.add();
51                                      break;
52                              case "-":
53                                      stack.sub();
54                                      break;
55                              case "*":
56                                      stack.mul();
57                                      break;
58                              case "/":
59                                      stack.div();
60                                      break;
61                              default:
62                                      break;
63                              }
64                      }
65                      else
66                      {
67                              if(!tokens[i].matches("(\\s)?(\\t)?"))
68                              {
69                                      throw new InvalidExpressionException();
70                              }
71                      }
72              }
73
74              if(stack.size() == 1)
75              {
76                      return stack.top();
77              }
78              else
79              {
80                      throw new InvalidExpressionException();
81              }
82      }
83
84      /**
85       * Returns true if s is an operator.
86       * An operator is one of '+', '-', '*', '/'.
87       */
88      private static boolean isOperator(String s) {
89
90              return s.matches("(\\+)?(\\-)?(\\*)?(\\/)?") && s.length() == 1; //can
                    be simplified (+-*/)
91      }
92
93      /**
94       * Returns true if s is an integer.
95       *
96       * We accept two types of integers:
97       *
98       * - the first type consists of an optional '-'
99       *   followed by a non-zero digit
100      *   followed by zero or more digits,
101      *
102      * - the second type consists of an optional '-'
```

```
103             *   followed by a single '0'.
104             */
105         private static boolean isInteger(String s) {
106 
107                 return s.matches("((((\\-)*?[1-9](\\d+)*)|(\\-)*?0)");
108         }
109 }
```

PostfixTest class

```
1  package inda3;
2  
3  import static org.junit.Assert.*;
4  
5  import org.junit.Before;
6  import org.junit.Test;
7  
8  public class PostfixTest {
9  
10         @Test
11         public void test() throws StackEmptyException, InvalidExpressionException {
12 
13                 assertTrue(Postfix.evaluate("0") == 0);
14                 assertTrue( Postfix.evaluate("-0") == -0);
15                 assertTrue( Postfix.evaluate("1234567890") == 1234567890);
16                 assertTrue( Postfix.evaluate("-1234567890") == -1234567890);
17                 assertTrue( Postfix.evaluate("1␣23␣+") == 1 + 23);
18                 assertTrue( Postfix.evaluate("1␣23␣␣␣␣␣␣+") == 1 + 23); // tabs instead
                       of spaces
19                 assertTrue( Postfix.evaluate("0␣1␣/") == 0 / 1);
20                 assertTrue( Postfix.evaluate("1␣2␣+␣-3␣*") == (1 + 2) * -3);
21                 assertTrue( Postfix.evaluate("12␣34␣-␣56␣-78␣+␣*") == (12 - 34) * (56 +
                       -78));
22                 assertTrue( Postfix.evaluate("1␣2␣+␣3␣*␣4␣-␣5␣/") == (((1 + 2) * 3) - 4)
                        / 5);
23                 assertTrue( Postfix.evaluate("2␣3␣4␣-0␣+␣-␣*") == 2 * (3 - (4 + -0)));
24                 assertTrue( Postfix.evaluate("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣1␣␣␣␣␣␣␣-2␣␣␣␣␣␣␣+␣") ==
                        1 - 2); // tabs and spaces
25 
26                 assertTrue( explodes(""));
27                 assertTrue( explodes("+"));
28                 assertTrue( explodes("--1"));
29                 assertTrue( explodes("-1-0"));
30                 assertTrue( explodes("-0-1"));
31                 assertTrue( explodes("1␣+"));
32                 assertTrue( explodes("1␣2␣,"));
33                 assertTrue( explodes("1␣2␣."));
34                 assertTrue( explodes("1␣2␣3␣+"));
35                 assertTrue( Postfix.evaluate("4") == 4);
36                 assertTrue( explodes("1␣2␣+␣+"));
37                 assertTrue( explodes("017"));
38                 assertTrue( explodes("0x17"));
39                 assertTrue( explodes("-03"));
40                 assertTrue( explodes("x"));
41                 assertTrue( explodes("1234L"));
42                 assertTrue( explodes("9876543210")); // larger than maxint
43                 assertTrue( explodes("1␣0␣/"));
44                 assertTrue( explodes("1␣2+"));
45                 assertTrue( explodes("1␣2␣3␣+*"));
46         }
47 
48         /**
49          * Returns true if <code>evaluate(expr)</code> throws
50          * a subclass of RuntimeException.
51          */
52         private static boolean explodes(String expr) {
53                 try {
54                         Postfix.evaluate(expr);
```

```
                } catch (Exception e) {
                        return true;
                }
                return false;
        }

}
```

InvalidExpressionException class

```
package inda3;

public class InvalidExpressionException extends Exception {

        public InvalidExpressionException(){}

        public InvalidExpressionException(String message)
        {
                super(message);
        }
}
```