```java
/*
 * HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
 * Version: Sun Sep 22 16:04:12 CEST 2024
 */

package uebung02.as.aufgabe03;

/**
 * Interface for a stack: a collection of objects that are inserted and removed
 * according to the last-in first-out principle.
 *
 * @author Roberto Tamassia
 * @author Michael Goodrich
 * @see EmptyStackException
 */

public interface Stack<T> {

  /**
   * Return the number of elements in the stack.
   *
   * @return Number of elements in the stack.
   */
  public int size();

  /**
   * Return whether the stack is empty.
   *
   * @return True if the stack is empty, false otherwise.
   */
  public boolean isEmpty();

  /**
   * Inspect the element at the top of the stack.
   *
   * @return Top element in the stack.
   * @exception EmptyStackException
   *              If the stack is empty.
   */
  public T top() throws EmptyStackException;

  /**
   * Insert an element at the top of the stack.
   *
   * @param element
   *          Element to be inserted.
   */
  public void push(T element);

  /**
   * Remove the top element from the stack.
   *
   * @return Element removed.
   * @exception EmptyStackException
   *              If the stack is empty.
   */
  public T pop() throws EmptyStackException;

  /**
   * Prints the contents of the stack to the console.
   */
  public void print();

}
```



```java
/*
 * HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
 * Version: Sun Sep 22 16:04:12 CEST 2024
 */

package uebung02.as.aufgabe03;


public class StackImpl<T> implements Stack<T> {

  /**
   * Nodes of a simple linked list.
   */
  private class Node<E> {

    private E element;
    private Node<E> next;

    /**
     * Constructs a new unlinked node.
     *
     * @param elem
     *          Element for the node.
     */
    public Node(E elem) {
      element = elem;
      next = null;
    }

    /**
     * Adds the node next to this node.
     *
     * @param next
     *          The next node.
     */
    public void appendNode(Node<E> next) {
      this.next = next;
    }

    public Node<E> getNext() {
      return next;
    }

    public E getElement() {
      return element;
    }
  }

  private Node<T> top;

  private int size;

  @Override
  public int size() {
    // TODO: Implement here...
    return -1;
  }

  @Override
  public boolean isEmpty() {
    // TODO: Implement here...
    return true;
  }

  @Override
  public T top() throws EmptyStackException {
    // TODO: Implement here...
    return null;
  }
```

```
70
71    @Override
72    public void push(T element) {
73      // TODO: Implement here...
74    }
75
76    @Override
77    public T pop() throws EmptyStackException {
78      // TODO: Implement here...
79      return null;
80    }
81
82    @Override
83    public void print() {
84      System.out.println("Stack: (" + toString(top, "") + ")");
85    }
86
87    private String toString(Node<T> node, String content) {
88      if (node == null) {
89        return content;
90      }
91      if (!content.equals("")) {
92        content += ", ";
93      }
94      content += node.getElement();
95      return toString(node.getNext(), content);
96    }
97
98
99    public static void main(String[] args) {
100     Stack<Integer> stack = new StackImpl<>();
101     stack.print();
102     final int TEST_SIZE = 4;
103     for (int i = 0; i < TEST_SIZE; i++) {
104       stack.push(i);
105       stack.print();
106       if (stack.size() != i+1) {
107         System.out.println("ERROR: Stack.size() != " + (i+1));
108         return;
109       }
110       if (stack.top() != i) {
111         System.out.println("ERROR: Stack.top() != " + i);
112         return;
113       }
114     }
115     for (int i = TEST_SIZE-1; i > 0; i--) {
116       if (stack.pop() != i) {
117         System.out.println("ERROR: Stack.pop() != " + i);
118         return;
119       }
120       stack.print();
121       if (stack.size() != i) {
122         System.out.println("ERROR: Stack.size() != " + i);
123         return;
124       }
125       if (stack.top() != i-1) {
126         System.out.println("ERROR: Stack.top() != " + (i-1));
127         return;
128       }
129     }
130     if (stack.pop() != 0) {
131       System.out.println("ERROR: Stack.pop() != 0");
132       return;
133     }
134     stack.print();
135     if (!stack.isEmpty()) {
136       System.out.println("ERROR: Stack.empty() != true");
137       return;
138     }
```



```
139     if (stack.size() != 0) {
140       System.out.println("ERROR: Stack.size() != 0");
141       return;
142     }
143     try {
144       stack.top();
145       System.out.println("ERROR: no EmptyStackException for stack.top()!");
146       return;
147     }
148     catch(EmptyStackException e) {
149       e = null;
150     }
151     try {
152       stack.pop();
153       System.out.println("ERROR: no EmptyStackException for stack.pop()!");
154       return;
155     }
156     catch(EmptyStackException e) {
157       e = null;
158     }
159   }
160
161 }
162
163
164 /* Session-Log:
165
166 Stack: ()
167 Stack: (0)
168 Stack: (1, 0)
169 Stack: (2, 1, 0)
170 Stack: (3, 2, 1, 0)
171 Stack: (2, 1, 0)
172 Stack: (1, 0)
173 Stack: (0)
174 Stack: ()
175
176 */
177
```

```
1  /*
2   * HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
3   * Version: Sun Sep 22 16:04:12 CEST 2024
4   */
5
6  package uebung02.as.aufgabe03;
7
8  /**
9   * Runtime exception thrown when one tries to perform operation top or pop on an
10  * empty stack.
11  */
12
13 public class EmptyStackException extends RuntimeException {
14
15   private static final long serialVersionUID = 1L;
16
17   public EmptyStackException(String err) {
18     super(err);
19   }
20 }
21
22
23
```