TESTING - PROGRAMME

Beispiel - StackTest

```
import static org.junit.Assert.*;
import org.junit.Test;

public class StackTest {
   @Test  public void testEmptyStack(){
      Stack stack = new Stack();
      assertTrue(stack.isEmpty());
   }
}
class Stack {
     public boolean isEmpty() { return true;     }
}
```

```
import static org.junit.Assert.*;
import org.junit.Test;

public class StackTest {
   @Test  public void testPush() {
      Stack stack = new Stack();
      Integer i = new Integer(3);
      stack.push(i);
      assertFalse(stack.isEmpty());
      }
}
class Stack {
   private boolean isNotEmpty = false;
   public boolean isEmpty() {   return !isNotEmpty;  }
   public void push(Object o) { isNotEmpty = true;    }
}
```

```
import static org.junit.Assert.*;
import junit.framework.TestCase;
import org.junit.Test;

public class StackTest extends TestCase {
   private Stack stack;
   public void setUp() {   stack = new Stack(); }
   @Test  public void testPopOnEmptyStack() {
      try {
         stack.pop();
         fail("pop() worked on empty stack");
      } catch (StackEmptyException expected) { }
   }
}
class Stack {
   private boolean isNotEmpty = false;
   public boolean isEmpty() {   return !isNotEmpty;        }
   public void push(Object o) { isNotEmpty = true;    }
   public Object pop() throws StackEmptyException{throw new StackEmptyException();  }
}
class StackEmptyException extends Exception{    }
```

```
import junit.framework.TestCase;
import org.junit.Test;

public class StackTest extends TestCase {
   private Stack stack;
   public void setUp() {   stack = new Stack(); }
   @Test  public void testPopOnStackWithOneElement()
      throws StackEmptyException {
      Integer i = new Integer(3);
      stack.push(i);
      assertEquals(i, stack.pop());
      assertTrue(stack.isEmpty());
   }
}
class Stack {
   private boolean isNotEmpty = false;          private Object elem;
   public boolean isEmpty() {   return !isNotEmpty;  }
   public void push(Object o) { isNotEmpty = true;   this.elem = o;  }
   public Object pop() throws StackEmptyException {
      if (this.isEmpty()) {throw new StackEmptyException();
      } else {  isNotEmpty = false;    return this.elem;     }
   }
}
class StackEmptyException extends Exception{    }
```

```
import junit.framework.TestCase;
import org.junit.Test;

public class StackTest extends TestCase {
private Stack stack;
   public void setUp() {     stack = new Stack();    }
   @Test public void testStackWithMoreThanOneElement()
         throws StackEmptyException {
         Integer i = new Integer(3);
         Integer j = new Integer(42);
         stack.push(i);
         stack.push(j);
         assertEquals(j, stack.pop());
         assertEquals(i, stack.pop());
         assertTrue(stack.isEmpty());
         }
}
class Stack {
   private Element head;
   public boolean isEmpty() {     return (head == null); }
   public void push(Object o) {    head = new Element (o, head);     }
   public Object pop() throws StackEmptyException{
   if (this.isEmpty()) {     throw new StackEmptyException();
   } else {   Object h = head.getValue();     head = head.getNext(); return h;   }
   }
}
class StackEmptyException extends Exception{  }
class Element{ ... }
```

Beispiel Subscription

```
import junit.framework.TestCase;
import org.junit.* ;

public class SubscriptionTest extends TestCase{
      @Test public void test_returnEuro() {
            System.out.println("Test if pricePerMonth returns Euro...") ;
            Subscription S = new Subscription(200,2) ;
            assertTrue(S.pricePerMonth() == 100.0) ;
      }
      @Test public void test_roundUp() {
            System.out.println("Test if pricePerMonth rounds up correctly...") ;
            Subscription S = new Subscription(200,3) ;
            assertTrue(S.pricePerMonth() == 67) ;
      }
}
class Subscription {
      private int price ;    private int length ;
      public Subscription(int p, int n) {
            price = p ; length = n ;
      }
      public double pricePerMonth() { return price / length ;  }
      public void cancel() {      length = 0 ; }
}
```

Beispiel Node

```
package jUnit;

import junit.framework.TestCase;
import org.junit.Test;

public class NodeTest extends TestCase{
      Node a = new Node(13);
      @Test public void testEmptyTree() {
            assertTrue("tree is not empty", a.empty());
      }
}

class Node {
      int val;    Node left, right;
      public Node (int val) {      this.val = val;   }

      public boolean empty() {
            return true;
      }

      public void insert (int val, Node node) {
            if (val < node.val) {
                  if (node.left != null) node.left = new Node(val);
                  else insert(val, node.left);
            }
            else {
                  if (node.right != null) node.right = new Node(val);
                  else insert(val, node.left);
            }
      }
}
```

Beispiel - Binary Tree

```
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class TreeTest {

   Tree tree;
   @Before public void setUp() {
     tree = new Tree();    System.out.print("Beginn Test ");
   }
   @After public void endTest() {
      tree.printTree();
      System.out.println("End Test ----------------------------------------------\n");
    }
   @Test public void testEmptyTree() {
     System.out.println("Empty Tree -----------------------------------------------");
     assertTrue("tree is not empty", tree.empty());
   }
   @Test public void testFillTree(){
     System.out.println("Fill Tree ------------------------------------------------");
     assertTrue("tree is not empty", tree.empty());
     tree.insert(2);
     assertFalse("tree is empty", tree.empty());
     tree.insert(10); tree.insert(5); tree.insert(7);
     tree.insert(6); tree.insert(-1); tree.insert(1);
   }
   @Test public void testGetItem(){
     System.out.println("getItem ------------------------------------------------");
     tree.insert(2); tree.insert(4); tree.insert(3); tree.insert(5);
     tree.insert(0); tree.insert(-1); tree.insert(1);
     assertEquals("wrong node return != 5", tree.getItem(5).getVal(), 5);
   }
   @Test(expected=ItemNotFoundException.class)
   public void testException() throws ItemNotFoundException{
     System.out.println("Exception ------------------------------------------------");
     Node node = new Node(1);
     node.insert(1);
   }
   @Test public void testLookUp(){
     System.out.println("LookUp -------------------------------------------------");
     tree.insert(2); tree.insert(4); tree.insert(3); tree.insert(5); tree.insert(0);
     assertTrue("item 2 does  exist",Tree.lookUp(2, tree.root));
     assertTrue("item 4 does  exist",Tree.lookUp(4, tree.root));
     assertTrue("item 3 does  exist",Tree.lookUp(3, tree.root));
     assertTrue("item 5 does  exist",Tree.lookUp(5, tree.root));
     assertTrue("item 0 does  exist",Tree.lookUp(0, tree.root));
     assertFalse("item 10 does not exist",Tree.lookUp(10, tree.root));
   }
   @Test
   public void testLookUp2(){
     System.out.println("LookUp2 ------------------------------------------------");
     tree.insert(2); tree.insert(4); tree.insert(3); tree.insert(5);
     tree.insert(0); tree.insert(-1); tree.insert(1);
     Node testNode = tree.getItem(4);
     assertTrue("item 5 does  exist",Tree.lookUp(5, testNode));
     assertTrue("item 3 does  exist",Tree.lookUp(3, testNode));
     assertFalse("item 10 does not exist",Tree.lookUp(10, testNode));
     assertFalse("item 1 does not exist",Tree.lookUp(1, testNode));
   }
   @Test public void testToString(){
     System.out.println("ToString -----------------------------------------------");
     tree.insert(2); tree.insert(4); tree.insert(3); tree.insert(5);
     tree.insert(0); tree.insert(6); tree.insert(1);
     assertTrue("incorrect output",tree.toString().equals("0123456"));
   }
   @Test public void testGetHeigth(){
      System.out.println("getHeigth -------------------------------------------------");
      tree.insert(2); tree.insert(4); tree.insert(3); tree.insert(5);
      tree.insert(0); tree.insert(6); tree.insert(1); tree.insert(9);
      assertEquals(5, tree.getHeigth());
   }
   @Test
   public void testRemove(){
      System.out.println("remove -------------------------------------------------");
      tree.insert(1); tree.insert(3); tree.insert(2); tree.insert(5); tree.insert(4);
      tree.insert(7); tree.insert(6); tree.insert(9); tree.insert(8);
      System.out.println(tree.remove(5));
      assertTrue("item 1 does  exist",Tree.lookUp(1, tree.root));
      assertTrue("item 2 does  exist",Tree.lookUp(2, tree.root));
      assertTrue("item 3 does  exist",Tree.lookUp(3, tree.root));
      assertTrue("item 4 does  exist",Tree.lookUp(4, tree.root));
      assertTrue("item 6 does  exist",Tree.lookUp(6, tree.root));
      assertTrue("item 7 does  exist",Tree.lookUp(7, tree.root));
      assertTrue("item 8 does  exist",Tree.lookUp(8, tree.root));
      assertTrue("item 9 does  exist",Tree.lookUp(9, tree.root));
      assertFalse("item 5 does not  exist",Tree.lookUp(5, tree.root));
   }
}

class Tree {
   public Node root;
   public void insert(int val) {
      try {
         if (empty()) root = new Node(val);
         else root.insert(val);
      } catch (Exception e) {
         System.out.println(e.toString());
      }
   }
   public boolean empty() { return root == null; }
   public void printTree() {
      if (root == null) return;  root.print(0);
   }
   public Node getItem(int val) {
      if (root == null) return null;
      return root.getItem(val);
   }
   public static boolean lookUp(int val, Node node) {
      return node.lookUp(val);
   }
   public String toString() {
      if (root == null) return null;
      return root.toString();
   }
   public int getHeigth() {
      if (root == null) return 0;
      return root.getHeigth();
   }
   public boolean remove(int val) {
      if (root == null) return false;
      else {
         if (root.getVal() == val) {
            Node auxRoot = new Node(0);
            auxRoot.left = root;
            boolean result = root.remove(val, auxRoot);
            root = auxRoot.left;
            return result;
         } else { return root.remove(val, null);
         }
      }
   }
}
```

```java
class Node {
   int val; Node left, right;
   public Node(int val) {  this.val = val; }
   public void insert(int val) throws ItemNotFoundException {
      if (this.val == val) throw new ItemNotFoundException(val);
      if (this.val < val) {
         if (left == null) left = new Node(val);
         else left.insert(val);
      } else {
         if (right == null) right = new Node(val);
         else right.insert(val);
      }
   }
   void print(int indent) {
      if (left != null) {   left.print(indent + 1);    }
      printIndent(indent);
      System.out.println(val);
      if (right != null) { right.print(indent + 1); }
   }
   private void printIndent(int indent) {
      for (int i = 0; i < indent; i++)
         System.out.print("    ");
   }
   public int getVal() { return val; }
   public Node getItem(int val) {
      if (this.val == val) return this;
      if (this.val < val) {
         if (left == null) return null;
         else return left.getItem(val);
      } else {
         if (right == null) return null;
         else return right.getItem(val);
      }
   }
   public boolean lookUp(int val) {
      if (this.val == val)
         return true;
      if (this.val < val) {
         if (left == null)
            return false;
         else
            return left.lookUp(val);
      } else {
         if (right == null)
            return false;
         else
            return right.lookUp(val);
      }
   }
   public String toString() {
      StringBuffer sb = new StringBuffer();
      if (right != null)    sb.append(right.toString());
      sb.append(val);
      if (left != null) sb.append(left.toString());
      return sb.toString();
   }
   int getHeigth() {
      int leftHeigth = 0;
      int rightHeigth = 0;
      if (left != null) leftHeigth = left.getHeigth();
      if (right != null) rightHeigth = right.getHeigth();
      return Math.max(leftHeigth, rightHeigth) + 1;
   }
   public boolean remove(int value, Node parent) {
      if (value > this.val) {
         if (left != null) return left.remove(value, this);
         else return false;
      } else if (value < this.val) {
         if (right != null) return right.remove(value, this);
         else return false;
      } else {
         if (left != null && right != null) {
            this.val = right.minValue();
            right.remove(this.val, this);
         } else if (parent.left == this) {
            parent.left = (left != null) ? left : right;
         } else if (parent.right == this) {
            parent.right = (left != null) ? left : right;
         }
         return true;
      }
   }

   public int minValue() {
      if (left == null) return val;
      else return left.minValue();
   }
}

class ItemNotFoundException extends Exception{
   private int val;
   public String toString() {   return val + " allready exists !"; }
   ItemNotFoundException(int val){    this.val = val; }
}
```