Bereich: Datenstrukturen (2)

Binärbaum	**Musterlösung**
Package: `de.dhbwka.java.exercise.collections`	**Klasse:** `BinaryTree`

```java
package de.dhbwka.java.exercise.collections;

import java.util.LinkedList;
import java.util.List;

/**
 * Part of lectures on 'Programming in Java'. Baden-Wuerttemberg
 * Cooperative State University.
 *
 * Thanks to Steven Kovcs for the idea!
 *
 * (C) 2016-2018 by W. Geiger, T. Schlachter, C. Schmitt, W. Suess
 *
 * @author DHBW lecturer
 * @version 1.2
 */
public class BinaryTree<T extends Comparable<T>> {

   private T value;
   private BinaryTree<T> left;
   private BinaryTree<T> right;

   public BinaryTree() {
   }

   /**
    * Recursive adding of newValue (no duplicates allowed).
    *
    * @param newValue
    *           new value to add
    *
    * @return <code>true</code> if value has been added, <code>false</code>
    *         otherwise
    */
   public boolean add( T newValue ) {
      if ( this.value == null ) {
         this.value = newValue;
      } else if ( this.value.compareTo( newValue ) == 0 ) {
         return false;
      } else if ( this.value.compareTo( newValue ) > 0 ) {
         return this.getLeft().add( newValue );
      } else if ( this.value.compareTo( newValue ) < 0 ) {
         return this.getRight().add( newValue );
      }
      return true;
   }


   // Continued on next page
```

```java
/**
 * Get the node's value
 */
public T getValue() {
   return this.value;
}

/**
 * Get left node, if none is present (<code>this.left==null</code>) create a
 * new node
 *
 * @return left
 */
private BinaryTree<T> getLeft() {
   if ( this.left == null ) {
      this.left = new BinaryTree<>();
   }
   return this.left;
}

/**
 * Get right node, if none is present (<code>this.left==null</code>) create a
 * new node
 *
 * @return right
 */
private BinaryTree<T> getRight() {
   if ( this.right == null ) {
      this.right = new BinaryTree<>();
   }
   return this.right;
}

/**
 * Traverse BinaryTree in ascending order and
 *
 * @return all values in an ordered List<T>
 */
public List<T> traverse() {
   List<T> l = new LinkedList<>();
   this.doTraverse( l );
   return l;
}

// Continued on next page
```

```
   /**
    * Traverse BinaryTree in ascending order and add all values to list.
    *
    * @param list
    *           list to add values to
    */
   private void doTraverse( List<T> list ) {
      if ( this.left != null ) {
         this.left.doTraverse( list );
      }
      if ( this.value != null ) {
         list.add( this.value );
      }
      if ( this.right != null ) {
         this.right.doTraverse( list );
      }
   }

}
```

```
package de.dhbwka.java.exercise.collections;

import java.util.Random;

/**
 * Part of lectures on 'Programming in Java'. Baden-Wuerttemberg
 * Cooperative State University.
 *
 * Thanks to Steven Kovcs for the idea!
 *
 * (C) 2016-2018 by W. Geiger, T. Schlachter, C. Schmitt, W. Suess
 *
 * @author DHBW lecturer
 * @version 1.1
 */
public class BinaryTreeTest {

   public static void main( String[] args ) {
      Random rnd = new Random();

      BinaryTree<Integer> tree = new BinaryTree<>();

      for ( int i = 0; i < 10; i++ ) {
         Integer newInt = rnd.nextInt( 20 );
         System.out.print( newInt );
         System.out.println( "  " + tree.add( newInt ) );
      }

      System.out.println( "------------" );
      for ( Integer t : tree.traverse() ) {
         System.out.println( t.toString() );
      }
   }

}
```