Bereich: Datenstrukturen

Binärbaum

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

Aufgabenstellung:

Im Foliensatz zu „Datenstrukturen“ gibt es eine Folie „Unsere erste generische Klasse“ (Nr. 49), in der eine Klasse `MyLinkedList<T>` vorgestellt wird.

Schreiben Sie nach diesem Vorbild eine Klasse `BinaryTree<T>`, die einen Knoten in einem Binärbaum beschreiben soll. (Eine Referenz auf die Wurzel erreicht also den ganzen Baum.)

- Jeder `BinaryTree<T>` soll einen Wert vom Typ `T` speichern können
- Jeder `BinaryTree<T>` hat genau zwei Nachfolger: `left` und `right`
  (Jeder Nachfolger darf auch `null` sein)
- `T` soll mindestens das Interface `Comparable` implementieren

Für jeden Knoten `k` im Binärbaum soll gelten:

- Der Wert von `k` muss definiert sein (`<>null`)
- Hat `k` einen Nachfolger `left` (`<>null`), so ist dessen Wert echt kleiner als der von `k`
- Hat `k` einen Nachfolger `right` (`<>null`), so ist dessen Wert echt größer als der von `k`

Korollar: In diesem Binärbaum gibt es keine Duplikate.

`BinaryTree<T>` soll folgende Methoden zur Verfügung stellen:

```
public boolean add(T newValue)
```
fügt einen neuen Wert nach den obigen Regeln in den Binärbaum ein, sofern dieser noch nicht vorhanden ist. Ist der Wert bereits vorhanden, wird false zurückgegeben, sonst true.

```
public T getValue()
```
liefert den Wert eines Knotens zurück.

```
public List<T> traverse()
```
traversiert den Baum und liefert eine **sortierte** (!) Liste aller enthaltenen Werte zurück.
Hinweis: Wählen Sie den Algorithmus der Traversierung so, dass „automatisch“ eine sortierte Liste entsteht. `Collections.sort` ist hier verboten!

Testen Sie Ihre Klasse:

1. Erzeugen Sie einen leeren Binärbaum!
2. Fügen Sie 10 Zufallszahlen ein (und geben sie diese zur Kontrolle auf der Konsole aus). Hinweis: Wählen oder erzeugen Sie einen geeigneten Datentyp dafür!
3. Traversieren Sie den Baum und geben das Ergebnis ebenfalls auf der Konsole aus!