Übungsblatt 3

Ausgabe: 20.4.2020
Abgabe: 6.5.2020 (24:00 Uhr)
Testat: 8.5.2020 (10:00 –13:00 Uhr)

Aufgabe 1: ADT BinaryTree - *Programmieraufgabe* - 160 Punkte

Es ist ein ADT BinaryTree so zu implementieren, dass im Baum alle möglichen Objekte vom Typ des Interface `Element` gespeichert werden können.
Sie finden sowohl das Interface `BinaryTree` als auch das Interface `Element` im Repo "Team-00". Um tatsächlich Elemente im Baum zu speichern, definieren Sie 2 Klassen, die das Interface `Element` implementieren: Die Klasse `IntElement` und die Klasse `StringElement`.
Schreiben Sie ein Hauptprogramm mit einem Steuerungsmenü so, dass Sie in einem Baum **entweder** `IntElement`-Objekte **oder** `StringElement`-Objekte verwalten.

Der ADT BinaryTree (binärer Suchbaum) stellt über das Interface `BinaryTree` die folgende Funktionalität zur Verfügung:

`boolean insert (Element val)`	Fügt val in den Baum ein.
`boolean insert (String filename)`	Fügt die Element-Werte, die in der Datei stehen in den Baum ein.
`boolean contains(Element val)`	Testet, ob val im Baum vorhanden ist.
`int size()`	Ermittelt die Anzahl der Knoten im Baum.
`int height()`	Ermittelt die Höhe des Baums.
`Element getMax()`	Liefert das größte Element im Baum.
`Element getMin()`	Liefert das kleinste Element im Baum.
`boolean remove (Element val)`	Entfernt val aus dem Baum. Dabei wird das zu entfernende Element ersetzt durch kleinste Element im rechten Teilbaum.
`booelan isEmpty()`	true genau dann, wenn der Baum leer ist.
`void clear()`	Entfernt alle Elemente aus dem Baum.
`BinaryTree addAll(BinaryTree otherTree)`	Fügt alle Elemente des übergebenen Baums (`otherTree`) in den aktuellen Baum ein. Die Elemente aus `othertree` werden in PreOrder-Reihenfolge eingefügt. Das ist aus Testzwecken so.
`void printInorder()`	Gibt Baum in Inorder aus.
`void printPostorder()`	Gibt Baum in Postorder aus.
`void printPreorder()`	Gibt Baum in Preorder aus.
`void printLevelorder()`	Gibt Baum in Levelorder aus.
`boolean saveToFile (String filename)`	Speichert die Element-Werte des Baums in PreOrder in der Datei.
`BinaryTree clone()`	Erzeugt eine tiefe Kopie des Baums. Der Klon sieht exakt so aus wie das Original.
`boolean equal (BinaryTree otherTree)`	Testet, ob der aktuelle Baum und othertree inhaltlich gleich sind.
`boolean equals (Object other)`	Testet, ob der aktuelle Baum und other inhaltlich und strukturell gleich sind.
`void visualize ()`	Gibt den Baum graphisch aufbereitet aus.

Denken Sie an sinnvolle Kommentare!

Legen Sie fest (und beschreiben Sie im Kommentar), welche Ergebnisse (Rückgabewerte) herauskommen. Das Einlesen und Schreiben von Werten von/auf Datei mit den entsprechenden Methoden der Klasse `MakeItSimple` ist etwas holprig.

Hinweis:
Überschreiben Sie die Methode `equals` der Klasse `Object`, die zwei Bäume miteinander vergleicht. Das Ergebnis von `equals` ist `true`, wenn die beiden Bäume sowohl in der Struktur, als auch auf Elementebene gleich sind.
Schreiben Sie eine Methode `equal`, die zwei Bäume bezüglich ihres Inhalts miteinander vergleicht. Das Ergebnis von `equal` ist `true`, wenn die beiden Bäume inhaltlich gleich sind.

Beispiele:

> Betrachten Sie die drei Binärbäume `tree1`, `tree2` und `tree3`.
> In `tree1` werden nacheinander die Werte 5, 2 und 10 eingefügt.
> In `tree2` werden nacheinander die Werte 2, 5 und 10 eingefügt.
> In `tree3` werden nacheinander die Werte 5, 2 und 10 eingefügt.
> Dann liefern die Vergleiche folgende Ergebnisse:

```
tree1.equals (tree2) → false
tree1.equals (tree3) → true
tree1.equal (tree2) → true
tree1.equal (tree3) → true
```

Keine der Operationen auf Binärbäumen darf mit Hilfe des Herausschreibens und Lesens von Daten auf/von Datei realisiert werden.

Hinweis: Evt. wird das Programm künftig noch (mehrfach) verändert. Achten Sie deshalb darauf, dass die einzelnen Operation korrekt arbeiten (das algorithmische Gerüst bleibt quasi unverändert).

Schauen Sie gelegentlich in Moodle nach, ob die Aufgabe ergänzt bzw. präzisiert wurde.