Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. R. Hennicker, Dr. A. Klarl

WS 2016/17
Übungsblatt 14

Übungen zu Einführung in die Informatik: Programmierung und Software-Entwicklung: Lösungsvorschlag

Aufgabe 14-1 **Binärbäume** Präsenz

Wir betrachten binäre Bäume, in deren Knoten Schlüssel/Element-Paare gespeichert werden. Die Elemente sind beliebige Objekte. Laden Sie von der Vorlesungswebseite die Klassen `BinTree` und `Node` herunter, und implementieren Sie folgende Methoden:

a) In der Klasse `BinTree` ist eine Methode `public Object findElement(int key)` implementiert, die einen Baum nach einem Knoten mit Schlüssel `key` durchsucht und den zugehörigen Wert zurückgibt. Die Methode ist wie folgt umgesetzt:

```
1 public Object findElement(int key) {
2   if (this.root != null) {
3     return this.root.findElement(key);
4   }
5   throw new NoSuchElementException();
6 }
```

Die Methode greift auf eine gleichnamige Methode `public Object findElement(int key)` in der Klasse `Node` zurück, welche die eigentliche Suche nach dem Knoten mit Schlüssel `key` realisieren soll. Die Methode soll für einen gegebenen Schlüssel `key` den zugehörigen Wert zurückgeben, falls ein Knoten mit dem Schlüssel `key` existiert. Ansonsten soll eine `NoSuchElementException` geworfen werden. Ergänzen Sie die Klasse `Node` um eine entsprechende Methode `public Object findElement(int key)`.

Lösung:
Es kann so vorgegangen werden, dass der Suchschlüssel zunächst mit dem Schlüssel des Wurzelknotens des betrachteten Teilbaums verglichen wird. Stimmt er mit diesem überein, wird der Wert des Wurzelknotens zurückgegeben. Andernfalls wird suchen wir im linken Teilbaum nach dem Schlüssel `key` (falls der linke Teilbaum nicht leer ist). Enthält der linke Teilbaum den Schlüssel `key` nicht, suchen wir im rechten Teilbaum nach dem Schlüssel `key` (falls der rechte Teilbaum nicht leer ist). Enthält auch der rechte Teilbaum den Schlüssel `key` nicht, werfen wir eine `NoSuchElementException`.

Implementierung der Methode `public Object findElement(int key)` in der Klasse `Node`:

```
1  public Object findElement(int key) throws NoSuchElementException {
2    if (key == this.key) {
3      return this.value;
4    }
5    if (this.left != null) {
6      try {
7        return this.left.findElement(key);
8      } catch (NoSuchElementException e) {
9        // Element konnte nicht im linken Teilbaum gefunden werden
10     }
11   }
12   if (this.right != null) {
13     try {
14       return this.right.findElement(key);
```

```
15      } catch (NoSuchElementException e) {
16        // Element konnte nicht im rechten Teilbaum gefunden werden
17      }
18    }
19    throw new NoSuchElementException();
20  }
```

b) In der Klasse `BinTree` ist eine Methode `public LinkedList<Object> toList()` implementiert, welche für einen Baum eine verkettete Liste vom Typ `LinkedList` zurückgibt, die die Werte der Knoten enthält. Die Methode ist wie folgt implementiert:

```
1 public LinkedList<Object> toList() {
2   LinkedList<Object> res = new LinkedList<Object>();
3   if (this.root != null) {
4     this.root.buildList(res);
5   }
6   return res;
7 }
```

Die Methode greift auf eine Methode `buildList(LinkedList<Object> res)` der Klasse `Node` zurück, welche die eigentliche Ansammlung von Knotenwerten in einer verketteten Liste `res` realisieren soll. Ergänzen Sie die Klasse `Node` um eine entsprechende Methode `buildList(LinkedList<Object> res)`. Gehen Sie dabei von der Java-Klasse `LinkedList<E>` aus.

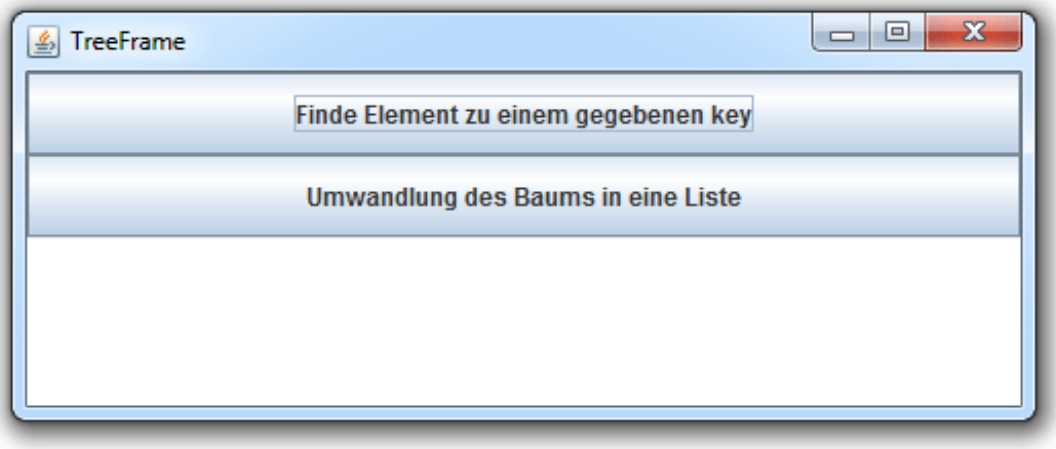
Lösung:
Es wird so vorgegangen, dass sowohl der Wert des Wurzelknotens des betrachteten Teilbaums als auch rekursiv die Werte aller nicht-leeren Unterbäume des Wurzelknotens der Liste hinzugefügt werden. Wir geben dabei immer zuerst den Wurzelknoten aus, gefolgt vom linken Teilbaum und rechten Teilbaum (Tiefensuche).

Implementierung der Methode `public LinkedList<Object> buildList()` in der Klasse `Node`:

```
1 public void buildList(LinkedList<Object> res) {
2   res.addLast(value);
3   if (this.left != null) {
4     this.left.buildList(res);
5   }
6   if (this.right != null) {
7     this.right.buildList(res);
8   }
9 }
```

c) Implementieren Sie nun ein Programm mit einer grafischen Benutzeroberfläche, welches die Methoden `public Object findElement(int key)` und `public LinkedList<Object> toList()` testet. Der zu testende Baum soll in diesem Programm als Konstante vom Typ `BinTree` vorliegen . Er soll einen Wurzelknoten haben mit dem Eintrag `(2, "Hello")` und einem Kindknoten mit dem Eintrag `(7, "World")`.

Die grafische Benutzeroberfläche soll wie folgt aussehen:



Es soll zwei Buttons, einen für die Suche eines Elements zu einem gegebenen Schlüssel und einen zum Umwandeln des Baums in eine Liste, geben. Darunter soll ein Ausgabebereich platziert werden, in dem Rückmeldung über die Ergebnisse gegeben wird.

Schreiben Sie eine Klasse `TreeFrame`, die die Hauptklasse dieser grafischen Benutzeroberfläche sein soll und das Fenster erzeugt. Um Ihr Programm ausführen zu können, schreiben Sie eine weitere Klasse `TreeFrameMain`, die Sie wie gewohnt im gleichen Ordner wie Ihre Klasse `TreeFrame` abspeichern.

Lösung:
Bei der Implementierung der grafischen Benutzeroberfläche geht man wie folgt vor: Deklarieren Sie eine Klasse `TreeFrame`, die die Hauptklasse Ihrer grafischen Benutzeroberfläche wird und deshalb von der Klasse `JFrame` erbt. Ihre Klasse `TreeFrame` soll drei Attribute haben:

- je ein Attribut vom Klassentyp `JButton` für die Suche eines Elements zu einem gegebenen Schlüssel und einen zum Umwandeln des Baums in eine Liste,
- eine Konstante vom Klassentyp `JTextArea`, das als Ausgabebereich für die spätere Rückmeldung über das Ergebnis der Suche bzw. die Ausgabe der Werte des Baums als Liste dient.

Schreiben Sie einen Konstruktor, der den `TreeFrame` mit einem entsprechenden Titel und Größe (wir wählen hier 500x200 Pixel) initialisiert. In dem Konstruktor sollen weiterhin alle Attribute korrekt initialisiert werden. Die Buttons werden zunächst in einem `JPanel` mit einem `GridLayout` mit zwei Zeilen und einer Spalte gruppiert. Dann wird das Layout des `ContentPane` des Frames auf ein `GridLayout` mit zwei Zeilen und einer Spalte initialisiert und die Gruppe der Buttons sowie der Ausgabebereich darauf platziert. Fügen Sie abschließend noch ein, dass das Programm ordnungsgemäß beendet wird, falls der `TreeFrame` geschlossen wird. Benutzen Sie dazu die Methode `setDefaultCloseOperation`.

Auf der Webseite finden Sie die Implementierung der Klassen `TreeFrame` und `TreeFrameMain` einschließlich der Lösungen der folgenden Aufgaben.

d) Erweitern Sie Ihre Klasse `TreeFrame` um eine Ereignisbehandlung für den Button für die Suche eines Elements zu einem gegebenen Schlüssel. Wird dieser Button gedrückt, soll der Benutzer in einer Methode `findeElement` mit Hilfe der Klasse `JOptionPane` nach dem Schlüssel des gesuchten Elements gefragt werden. Der Wert des gefundenen Elements soll anschließend im Ausgabebereich angezeigt werden. Falls kein Element mit dem gesuchten Schlüssel im Baum vorhanden ist (und somit eine `NoSuchElementException` geworfen wird) soll das Programm die Fehlermeldung “Ein Element mit diesem Schlüssel ist im vorgegebenen Baum nicht vorhanden.” ausgeben.

Lösung:
Implementierung der Methode `findeElement()`:

```
1  private void findeElement() {
2    String einlesenKey = JOptionPane
3        .showInputDialog("Key zu gesuchtem Element eingeben: ");
4    int key = Integer.parseInt(einlesenKey);
5
6    this.ausgabeBereich.setText("");
7    String gefundenesElement = "";
8    try {
9      gefundenesElement = tree.findElement(key).toString();
10     this.ausgabeBereich.setText(gefundenesElement);
11   } catch (NoSuchElementException e) {
12     this.ausgabeBereich
13         .setText("Ein Element mit diesem Schlüssel ist"
14             + " im vorgegebenen Baum nicht vorhanden.");
15   }
16 }
```

e) Erweitern Sie Ihre Klasse `TreeFrame` um eine Ereignisbehandlung für den Button für das Umwandeln des vorgegebenen Baums in eine Liste der Werte der Knoten des Baums vom Typ `LinkedList<Object>`. Wird dieser Button gedrückt, so soll eine Methode `umwandelnBaum` die Umwandlung des Baums übernehmen und die Werte im Ausgabebereich anzeigen.

Lösung:
Implementierung der Methode `umwandelnBaum()`:

```
1  private void umwandelnBaum() {
2    LinkedList<Object> ausgabeliste = tree.toList();
3
4    this.ausgabeBereich.setText("Der Baum als Liste: \n");
5    for (Object knoten : ausgabeliste) {
6      this.ausgabeBereich.append(knoten.toString() + "\n");
7    }
8  }
```

Besprechung der Präsenzaufgabe in den Übungen vom 01.02.2017 bis 08.02.2017.