Die Klasse BinaryTree <Content Type> in Java

Implementation (generisch) an GUI (JavaFX)

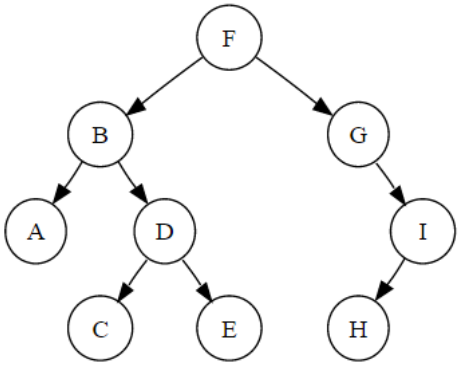
Standardmethoden + Traversierung

+ weitere spezielle Methoden

//offizielle Abiturklasse (im 1. Teil - weiter unten ergänzt durch nützliche Methoden!)
public class BinaryTree <ContentType> {

```
    //innere Klasse (Wurzelknoten)
    private class BTNode {
      private ContentType content;
      private BinaryTree <ContentType> left, right;

      public BTNode(ContentType pContent) {
        content = pContent;
        left = new BinaryTree <>();
        right = new BinaryTree <>();
      }
    }
    //Ende innere Klasse
```




```
private BTNode node; //ein Baum besteht aus einer Wurzel (mit Teilbäumen - s.o.)

public BinaryTree() { //Konstruktor 1
  node = null;
}

public BinaryTree(ContentType pContent) { //Konstruktor 2
  if (pContent != null) node = new BTNode(pContent); else node = null;
}

public BinaryTree(ContentType pContent, BinaryTree <ContentType> pLeftTree, BinaryTree
<ContentType> pRightTree) { //Konstruktor 3
  if (pContent != null) {
    node = new BTNode(pContent);
    if (pLeftTree != null) node.left = pLeftTree; else node.left = new BinaryTree <>();
    if (pRightTree != null) node.right = pRightTree; else node.right = new BinaryTree <>();
  } else node = null;
}

public boolean isEmpty() {
  return node == null;
}
```

```
public void setContent(ContentType pContent) {
 if (pContent != null) {
  if (isEmpty()) {
   node = new BTNode(pContent);
   node.left = new BinaryTree <>(); node.right = new BinaryTree <>();
  }
  node.content = pContent;
 }
}

public ContentType getContent() {
 if (isEmpty()) return null; else return node.content;
}

public void setLeftTree(BinaryTree <ContentType> pTree) {
 if (!isEmpty() && pTree != null) node.left = pTree;
}

public void setRightTree(BinaryTree <ContentType> pTree) {
 if (!isEmpty() && pTree != null) node.right = pTree;
}

public BinaryTree <ContentType> getLeftTree() { if (!isEmpty()) return node.left; else return null; }

public BinaryTree <ContentType> getRightTree() { if (!isEmpty()) return node.right; else return null; }
//Ende offizielle Abiturklasse

//sinnvolle Erweiterungen (für jeden Binärbaum!)
//erst verschiedene Traversierungsalgorithmen
public String inorder() {
 if (!isEmpty()) return getLeftTree().inorder() + getContent() + " " + getRightTree().inorder();
 else return "";
}

public String preorder() {
 if (!isEmpty()) return getContent() + " " + getLeftTree().preorder() + getRightTree().preorder();
 else return "";
}

public String postorder() {
 if (!isEmpty()) return getLeftTree().postorder() + getRightTree().postorder() + getContent() + " ";
 else return "";
}
```

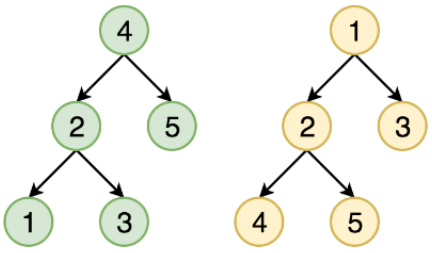
```
public String levelorderMitQueue() {
//unten eine Version ohne Queue (rekursiv mit 2 Methoden)
//untere ist kürzer, speichert aber durch Rekursion auch Zwischenwerte auf Stack!
//diese hier ist evtl. etwas intuitiver/ leichter nachzuvollziehen!
  String s;
  Queue <BinaryTree <ContentType>> q = new Queue <>();
  s = "";
  q.enqueue(this); //kompletter Baum wird in die Schlange eingefügt
  while (!q.isEmpty()) {
    BinaryTree <ContentType> tree = q.front();
    s = s + tree.getContent() + " "; //Wurzel des 1. Baums in der Schlange wird zu s hinzugefügt
    if (!tree.getLeftTree().isEmpty()) {
      q.enqueue(tree.getLeftTree()); //linker Teilbaum wird an die Schlange angehangen
    }
    if (!tree.getRightTree().isEmpty()) {
      q.enqueue(tree.getRightTree()); //rechter Teilbaum wird an die Schlange angehangen
    }
    q.dequeue(); //1. Element der Schlange wird gelöscht
  }
  return s;
}

public String levelorder(){
  String s = "";
  for (int i=1; i<=height(); i++) s = s + giveLevel(i);
  return s;
}

private String giveLevel(int level) {
  if (isEmpty()) return "";
  else if (level == 1) return getContent() + " ";
  else return getLeftTree().giveLevel(level - 1) + getRightTree().giveLevel(level - 1);
}
```

DFS
Inorder
Left -> Node -> Right

BFS
Left -> Right
Top -> Bottom




```
//weitere nützliche Extras
public int number() {
  if (isEmpty()) return 0; else return getLeftTree().number() + 1 + getRightTree().number();
}

public int height() {
  if (isEmpty()) return 0;
  else if (getLeftTree().height() > getRightTree().height()) return getLeftTree().height() + 1;
  else return getRightTree().height() + 1;
}
}
```

Controller-Klasse

```
import javafx.fxml.FXML;
…
public class Controller {
   @FXML private Canvas myCanvas;
   …
   private BinaryTree Baum = new BinaryTree(); private BinaryTree Baum1, Baum2;
   …
   public void btZufall_onClick() {
    int[] a = new int[10];
    for (int i=0; i<=9 ; i++) { Random rand = new Random(); a[i] = rand.nextInt(7)+7*i; }
    Baum = new BinaryTree(a[1], new BinaryTree(a[0]), new BinaryTree(a[2]));
    Baum1 = new BinaryTree(a[5], new BinaryTree(a[4]), new BinaryTree());
    Baum2 = new BinaryTree(a[8], new BinaryTree(a[7]), new BinaryTree(a[9]));
    Baum2 = new BinaryTree(a[6], Baum1, Baum2); Baum = new BinaryTree(a[3], Baum, Baum2);
    paint(myCanvas.getGraphicsContext2D());
   }
   public void btInorder_onClick() { tfInorder.setText(Baum.inorder()); }
   …
   public void btPreorder_onClick() { tfPreorder.setText(Baum.preorder()); }
   public void btPostorder_onClick() { tfPostorder.setText(Baum.postorder()); }
   public void btLevelorder_onClick() { tfLevelorder.setText(Baum.levelorder()); }
   public void btAnzahl_onClick() { tfAnzahl.setText("" + Baum.number()); }
   public void btHoehe_onClick() { tfHoehe.setText("" + Baum.height()); }
   …
```