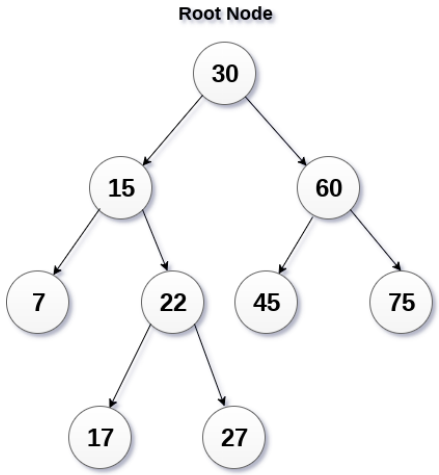
Die Klasse BinarySearchTree
<ContentType extends ComparableContent <ContentType>>
in Java - Implementation (generisch) an GUI (JavaFX)
Standardmethoden + Traversierung
+ weitere spezielle Methoden

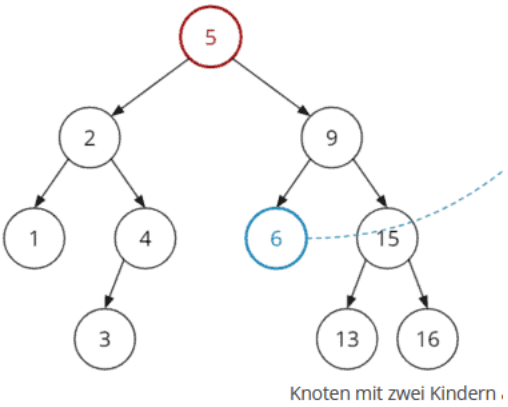
```
public interface ComparableContent <ContentType> {
  public boolean isGreater(ContentType pContent);
  public boolean isEqual(ContentType pContent);
  public boolean isLess(ContentType pContent);
}

public class Entry implements ComparableContent <Entry> {
  private int Wert;
  public Entry (int Zahl) { Wert = Zahl; }
  public int getWert() { return Wert; }
  public boolean isLess(Entry pContent) { return (getWert() < pContent.getWert()); }
  public boolean isEqual(Entry pContent) { return (getWert() == pContent.getWert()); }
  public boolean isGreater(Entry pContent) { return (getWert() > pContent.getWert()); }
}

public class BinarySearchTree <ContentType extends ComparableContent <ContentType>> {
//aus offizieller Abitur-Vorgabe - allerdings Quelltext ein wenig verändert: "this" gelöscht, teils
//unnötige Klammern entfernt, teils Attributen- durch Methodenaufrufe ersetzt, "remove" ohne
//Hilfsmethoden, ...! außerdem im unteren Teil weitere nützliche Methoden ergänzt!

  //innere Klasse (Wurzelknoten)
  private class BSTNode <CT extends ComparableContent <CT>> {
    private CT content;
    private BinarySearchTree <CT> left, right;

    public BSTNode(CT pContent) {
      content = pContent;
      left = new BinarySearchTree<>();
      right = new BinarySearchTree<>();
    }
  }
  //Ende innere Klasse

  private BSTNode <ContentType> node;

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



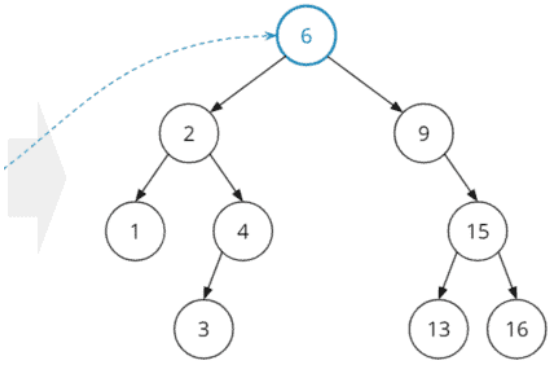
Binary Search Tree

```
public void insert(ContentType pContent) {
  if (pContent != null) {
    if (isEmpty()) node = new BSTNode <>(pContent);
    else if (pContent.isLess(getContent())) getLeftTree().insert(pContent);
    else if(pContent.isGreater(getContent())) getRightTree().insert(pContent);
  }
}

public ContentType search(ContentType pContent) {
  if (isEmpty() || pContent == null) return null;
  else if (pContent.isLess(getContent())) return getLeftTree().search(pContent);
  else if (pContent.isGreater(getContent())) return getRightTree().search(pContent);
  else return pContent;
}

public void remove(ContentType pContent) {
  if (!isEmpty() && pContent != null ) {
    if (pContent.isLess(getContent())) {
      getLeftTree().remove(pContent);
    } else if (pContent.isGreater(getContent())) {
      getRightTree().remove(pContent);
    } else { //gefunden
      if (getLeftTree().isEmpty()) {
        if (getRightTree().isEmpty()) { //kein Nachfolger
          node = null;
        } else { //nur rechts Nachfolger
          node = getRightTree().node;
        }
      } else if (getRightTree().isEmpty()) { //nur links Nachfolger
        node = getLeftTree().node;
      } else { //links und rechts Nachfolger - suche den nächstgrößten!
        BinarySearchTree <ContentType> tree = getRightTree();
        while (!tree.getLeftTree().isEmpty()) tree = tree.getLeftTree();
        ContentType hilfe = tree.getContent();
        remove(hilfe); //Methode wird rekursiv aufgerufen - sorgt für Nachrücken des Restbaumes
        node.content = hilfe;
      }
    }
  }
}

public ContentType getContent() { if (isEmpty()) return null; else return node.content; }
public BinarySearchTree <ContentType> getLeftTree() { if (isEmpty()) return null; else return node.left; }
public BinarySearchTree <ContentType> getRightTree() { if (isEmpty()) return null; else return node.right; }
```

//Ende offizielle (angepasste) Abiturklasse



Knoten mit zwei Kindern



aus binärem Suchbaum löschen

```
 //weitere nützliche Methoden (siehe teils auch BinaryTree)
 public void inorder(Queue <ContentType> queue) { //zur Abwechslung mal mit Queue implementiert!
  if (!isEmpty()) {
   getLeftTree().inorder(queue); queue.enqueue(getContent()); getRightTree().inorder(queue);
  }
 }
 …  (preorder und postorder laufen genauso!)
 public void levelorder(Queue <ContentType> queue) { for (int i=1; i<=height(); i++) giveLevel(i, queue); }
 public void giveLevel(int level, Queue <ContentType> queue) {
  if (!isEmpty()) {
   if (level == 1) queue.enqueue(getContent());
   else { getLeftTree().giveLevel(level-1, queue); getRightTree().giveLevel(level-1, queue); }
  }
 }
 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;
 }
 public ContentType minimum() {
  if (isEmpty()) return null;
  else if (getLeftTree().isEmpty()) return getContent(); else return getLeftTree().minimum();
 }
 … (maximum wird analog implementiert!)
}
```

Controller-Klasse

```
import javafx.fxml.FXML;
…
public class Controller {
   @FXML private Canvas myCanvas;
    …
   private BinarySearchTree <Entry> Baum = new BinarySearchTree();
   private Queue <Entry> queue = new Queue();

   public void btBsp_onClick() {
    Baum = new BinarySearchTree(); int[] Zahlen = {10, 5, 15, 2, 7, 13, 17, 8, 9, 6, 4, 14, 16, 1, 22, 11, 12};
    for (int i = 0; i < Zahlen.length; i++) Baum.insert(new Entry(Zahlen[i]));
    paint(myCanvas.getGraphicsContext2D());
   }
   …
   public void btInorder_onClick() {
    queue = new Queue(); Baum.inorder(queue); tfInorder.setText(AusgabeQueue());
   } …
```

```
public String AusgabeQueue() {
 String s = ""; while (!queue.isEmpty()) { s = s + queue.front().getWert() + " "; queue.dequeue(); }
 return s;
}
public void btInsert_onClick() {
 Baum.insert(new Entry(Integer.parseInt(tfInsert.getText()))); paint(myCanvas.getGraphicsContext2D());
}
public void btSearch_onClick() {
 Entry hilfe = new Entry(Integer.parseInt(tfSearch.getText()));
 if (Baum.search(hilfe) == null) tfSearchAusgabe.setText("nicht vorhanden!");
 else tfSearchAusgabe.setText("vorhanden!");
} …
public void btNumber_onClick() { tfNumber.setText("" + Baum.number()); } …
public void btBuildTree_onClick() { //Aufbau eines ausgeglichenen Baums aus Zahlenreihe
 String[] s = tfBuildTree.getText().split(" ", 0); //aufsplitten an " ", Teile in String-array
 int laenge = s.length; int[] Reihe = new int[laenge];
 for (int i  = 0; i < laenge; i++) Reihe[i] = Integer.parseInt(s[i]);
 Reihe = bubblesort(Reihe); //sortierte Liste, damit Baum maximal ausgeglichen
 Baum = new BinarySearchTree(); FuegeEin(Reihe, 0, laenge - 1);
 paint(myCanvas.getGraphicsContext2D());
}
public int[] bubblesort(int[] toSort) {
 int temp; for(int i = 1; i < toSort.length; i++) for(int j = 0; j < toSort.length - i; j++)
                if(toSort[j] > toSort[j+1]) { temp = toSort[j]; toSort[j]=toSort[j+1]; toSort[j+1]=temp; }
 return toSort;
}
public void FuegeEin(int[] Feld, int l, int r) {
 if (l <= r) {
    int m = (l+r) / 2; Baum.insert(new Entry(Feld[m])); FuegeEin(Feld, l, m-1); FuegeEin(Feld, m+1, r);
 }
}
```