Helmut Schauer
Educational Engineering Lab
Department for Informatics
University of Zurich

Algorithmen und Datenstrukturen
FS 2008

Bäume

Binary Tree in Java (1)

```
class BinaryTree {
  class Node {
    static Node empty = new Node(null) {
      boolean isEmpty() {return true;}
    };
    Object info;
    Node left, right;
    Node(Object x) {info = x; left = empty; right = empty;}
    boolean isEmpty() {return false;}
    ...
  }
  Node root;
  BinaryTree() {root = empty;}
  boolean isEmpty() {return root.isEmpty();}
  ...
}
```

Binary Tree in Java (2)
Anzahl der Knoten

```
int count() {
    if (isEmpty()) return 0;
    else return 1+left.count()+right.count();
}
```

Binary Tree in Java (3)
Maximale Höhe

```
int height() {
    if (isEmpty()) return 0;
    else return 1+Math.max(left.height(),right.height());
}
```

Preorder Traversieren (rekursiv)

```
preOrder() {
    if (!isEmpty()) {
            visit(info);
            left.preOrder();
            right.preOrder();
    }
}
```

Postorder Traversieren (rekursiv)

```
postOrder() {
      if (!isEmpty()) {
              left.postOrder();
              right.postOrder();
              visit(info);
      }
}
```

Inorder Traversieren (rekursiv)

```
inOrder() {
        if (!isEmpty()) {
                left.inOrder();
                visit(info);
                right.inOrder();
        }
}
```

Preorder Traversieren (iterativ)

```
preOrder() {
    if (isEmpty()) return;
    Stack stack = new Stack();
    stack.push(root);
    do {
        Node t = (Node) stack.pop();
        visit(t.info);
        if (!t.right.isEmpty())
            stack.push(t.right);
        if (!t.left.isEmpty())
            stack.push(t.left);
    } while (!stack.isEmpty());
}
```

Levelorder Traversieren (iterativ)

```
levelOrder() {
    if (isEmpty()) return;
    Queue queue = new Queue();
    queue.put(root);
    do {
        Node t = (Node) queue.get();
        visit(t.info);
        if (!t.left.isEmpty())
            queue.put(t.left);
        if (!t.right.isEmpty())
            queue.put(t.right);
    } while (!queue.isEmpty());
}
```

Binäres Suchen in einem Sortierbaum

```
Object search(Comparable x) {
    if (isEmpty())
        return null; // x not found
    if (x.compareTo(info)<0)
        return left.search(x);
    if (x.compareTo(info)>0)
        return right.search(x);
    else
        return info; // x found
}
```

Einfügen in einen Sortierbaum

```
Node insert(Comparable x) {
        if (isEmpty())
                return new Node(x);
        if (x.compareTo(info)<0)
                left = left.insert(x);
        else
                right = right.insert(x);
        return this;
}
```

Entfernen aus einem Sortierbaum

```
Node remove(Comparable x) {
        if (isEmpty()) return this;
        if (x.compareTo(info)<0)
                left = left.remove(x);
        else if (x.compareTo(info)>0)
                right = right.remove(x);
        else if (left.isEmpty()) return right;
        else if (right.isEmpty()) return left;
        else {
                Node t = right;
                while (!t.left.isEmpty()) t = t.left;
                info = t.info;
                right = right.remove((Comparable)info);
        }
        return this;
}
```