Exercise #7

If you haven't already, read the relevant parts of the course materials and try to answer the questions at the end of the chapter.

book up until page 286, including chapter 4.5.4 about Iterators.

Binary tree

The given data structure is a binary tree that contains integer values. Extend that data structure by adding the methods mentioned below. Make sure to not add any new instance variables to the class, only use the ones already provided.

Listing 1: binary tree

```
public class BinaryTree {

    private Node root;

    private class Node {
        private int value;
        private Node left;
        private Node right;

        // TODO: implement methods here
    }

    // TODO: implement methods here
}
```

Make sure to add comments to your code as documentation. Also, write a testing program for each method you implement.

Aufgabe 1: Querying and adding

Add the method `void add(int val)` to `BinaryTree`.

Also, add the method `boolean contains(int val)`, which returns `true` if the tree contains the provided value and `false` otherwise.

Aufgabe 2: Size

Add the method `int size()`, which returns the number of elements contained within the tree. Also, add the method `boolean empty()`, which returns `true` if the tree is empty and `false` if it is not.

Aufgabe 3: Traversals

There are three ways to traverse a binary tree:

Inorder: Output the left subtree, then the current element and finally the right subtree.

Preorder: Output the current element, then the left and finally the right subtree.

Postorder: Output the left subtree, then the right subtree and finally the current element.

Each of these traversals is done recursively until the whole tree has been traversed.

Write three Methods which should return the content of the whole tree in Inorder, Preorder and Postorder, respectively.

Aufgabe 4: Estimate algorithmic complexity

Estimate the runtime and memory comsumption (both average and maximum) of every method you implemented.

Hashtable

Aufgabe 5: Generic hashtable

We presented a hashtable `Hashtable` for `int` values in our lecture. Modify that class in a way that allows it to handle objects of an arbitrary (but equal) type. Should a collision occur, use a linked list to save all elements mapped to the same hashcode.

guaranteed by a type parameter

Write a testing program that creates `Hashtable`s for `Integer`, `String` and for a class you have written yourself (e.g. `Ladybird`). Make sure to also add two elements with the same `hashCode()`, to make sure that the class behaves correctly in the event of collisions.

Theory questions

Answer the following questions:

Aufgabe 6: Algorithms and data structures

What is an algorithm, and what is a data structure? How are these two terms connected? When do we call two algorithms or data structures equal and when do we not? Name at least five different data structures and characterize them.

Aufgabe 7: Algorithmic complexity estimation

How can we estimate the complexity of an algorithm? What do $O(1)$, $O(\log(n))$, $O(n)$, $O(n \cdot \log(n))$, $O(n^2)$ and $O(2^n)$ mean? How do the runtime and memory consumption of an algorithm change if we double n or if we multiply it by 100. Why may we ignore constant factors when estimating the complexity of an algorithm?

Aufgabe 8: Searching and sorting

What is binary search? How do Bubblesort, Mergesort and Quicksort work? What are their respective average and worst case complexities?

Aufgabe 9: Generic programming

What is the difference between generic and non-generic classes? What is the difference between a type and a type parameter? Can we use types and type parameters equally? How can we use primitive types like `int` as type parameters in Java's generic containers?

Aufgabe 10: Iterators

What are iterators, and why do we use them? What difficulties may arise when using iterators in conjunction with recursion, and how can we overcome them? Which special language feature of Java helps us use iterators effectively?