JOHANNES KEPLER
UNIVERSITÄT LINZ

Algorithms and Data Structures 2
Winter term 2019
Stefan Grünberger, Dari Trendafilov

Assignment 04

Deadline: **Mon. 25.11.2019, 23:59**
Submission via: **www.pervasive.jku.at/Teaching/**

Trees

1. Binary search tree — 24 points

Implement a binary search tree **MyBinarySearchTree** according to the following interface **BinarySearchTree.**
Note: The insertion of duplicate elements is not allowed in this implementation of a binary search tree.

```
public interface BinarySearchTree {

/**
 * @param elem
 * @return
 * @throws IllegalArgumentException
 * Insert the element elem into the tree and return true if it was successful.
 * Elements with the same key are not allowed, in this case false is returned.
 * Null-elements are not allowed, in this case an exception is thrown.
 */
public boolean insert(Integer key, String elem) throws IllegalArgumentException;

/**
 * @param key
 * @return value
 * @throws IllegalArgumentException
 * Returns the value of the first found element with the given key, or null if element was not found.
 */
public String find(Integer key) throws IllegalArgumentException;

/**
 * @param key
 * @return success
 * @throws IllegalArgumentException
 * Removes the first element with the given key, and returns true if element was found AND removed.
 */
public boolean remove(Integer key) throws IllegalArgumentException;

/**
 * @return
 * Returns the number of elements stored in the binary tree.
 */
public int size();

/**
 * @return
 * Returns an array-representation of the stored elements (Postorder traversal).
 */
public Object[] toArrayPostOrder();

/**
 * @return
 * Returns an array-representation of the stored elements (Inorder traversal).
 */
public Object[] toArrayInOrder();

/**
 * @return
 * Returns an array-representation of the stored elements (Preorder traversal).
 */
public Object[] toArrayPreOrder();

/**
 * @param key
 * @return key of parent
 * @throws IllegalArgumentException
 * Search the parent node of the node with the given key,
 * and return its key (or null if not found).
 */
public Integer getParent(Integer key) throws IllegalArgumentException;
```

```
/**
 * @param key
 * @return
 * @throws IllegalArgumentException
 * Return true if the node with the given key is the root, otherwise return false.
 */
public boolean isRoot(Integer key) throws IllegalArgumentException;

/**
 * @param key
 * @return
 * @throws IllegalArgumentException
 * Return true if the node with the given key is an internal node, otherwise return false.
 */
public boolean isInternal(Integer key) throws IllegalArgumentException;

/**
 * @param key
 * @return
 * @throws IllegalArgumentException
 * Return true if the node with the given key is an internal node, otherwise return false.
 */
public boolean isExternal(Integer key) throws IllegalArgumentException;
}
```

For the nodes of the binary search tree use the class **BinaryTreeNode**:

```
public class BinaryTreeNode {
    /**
     * Reference to the left child node
     */
    public BinaryTreeNode left;

    /**
     * Reference to the right child node
     */
    public BinaryTreeNode right;

    /**
     * Key of the node
     */
    public Integer key;

    /**
     * Data of the node
     */
    public String elem;


    public BinaryTreeNode(Integer key, String elem) {
        this.key = key;
        this.elem = elem;
        left = null;
        right = null;
    }
}
```

For the implementation of `toArrayPostOrder` an auxiliary function

```
private int toArrayPostOrder (Object[] ret, int offset, BinaryTreeNode n) { … }
```

is useful, that starts writing to the array at index *offset* and returns the last index written to the temporary array *ret* (similar auxiliary functions could also be useful for the other toArray-traversal implementations). Consider if other help functions might be useful for reusing code.

Note: For the assignment 05 this tree structure will be extended with further functionality. Therefore it's highly recommended to work this assignment out.