HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

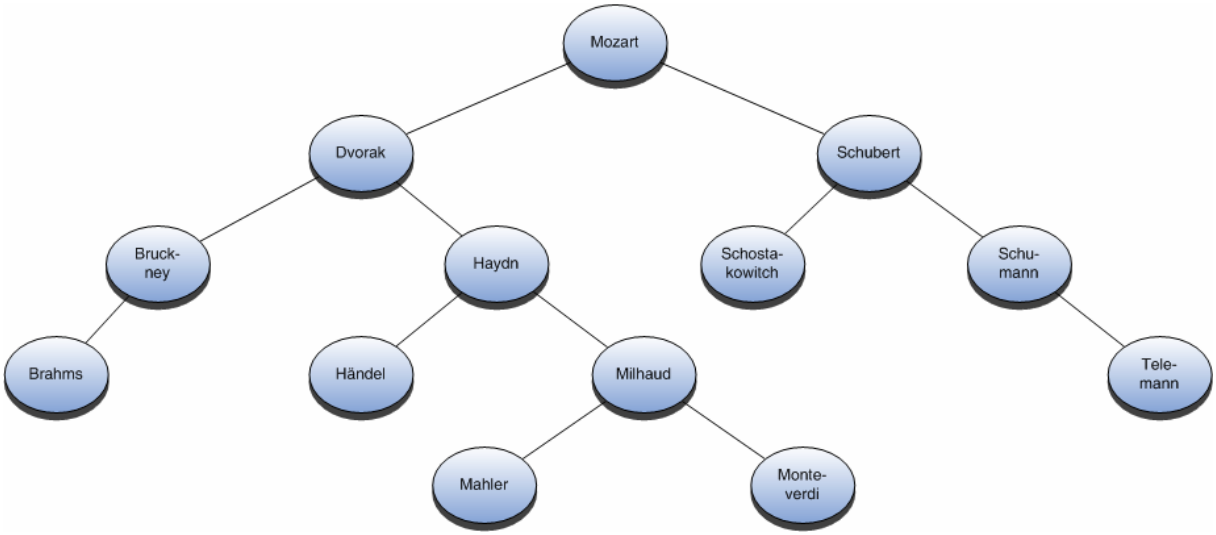
MUSTERLÖSUNGEN ÜBUNGSSERIE 3

Algorithmen & Datenstrukturen AD2 / HS 2019

AD2 Team

Aufgabe 1 (Einfügen in AVL-Bäume - Rotationen)

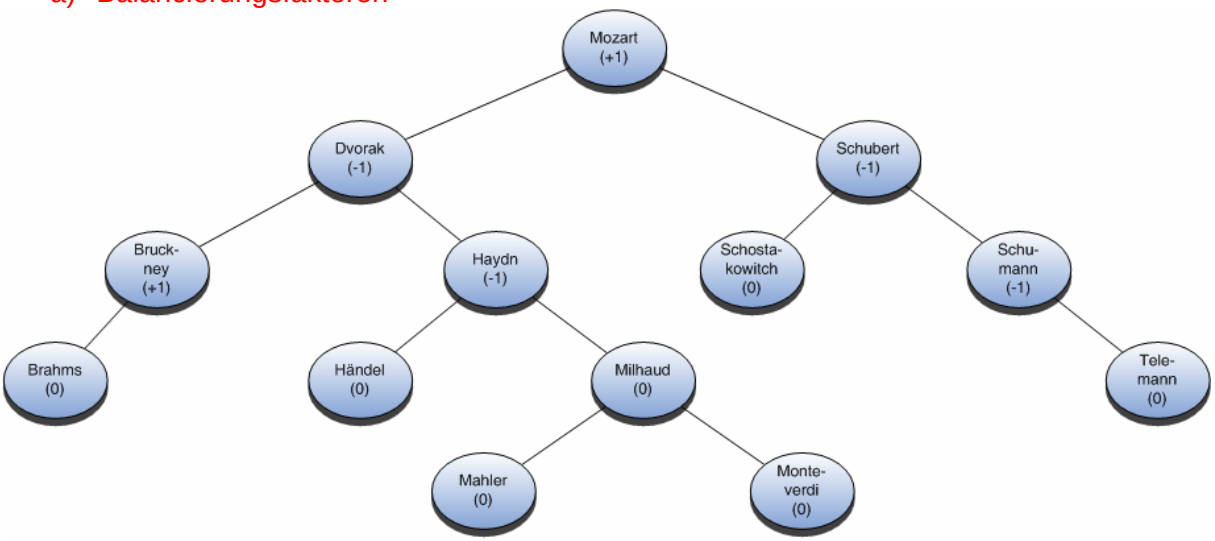
Gegeben sei nun der folgende AVL-Baum mit Komponistennamen:



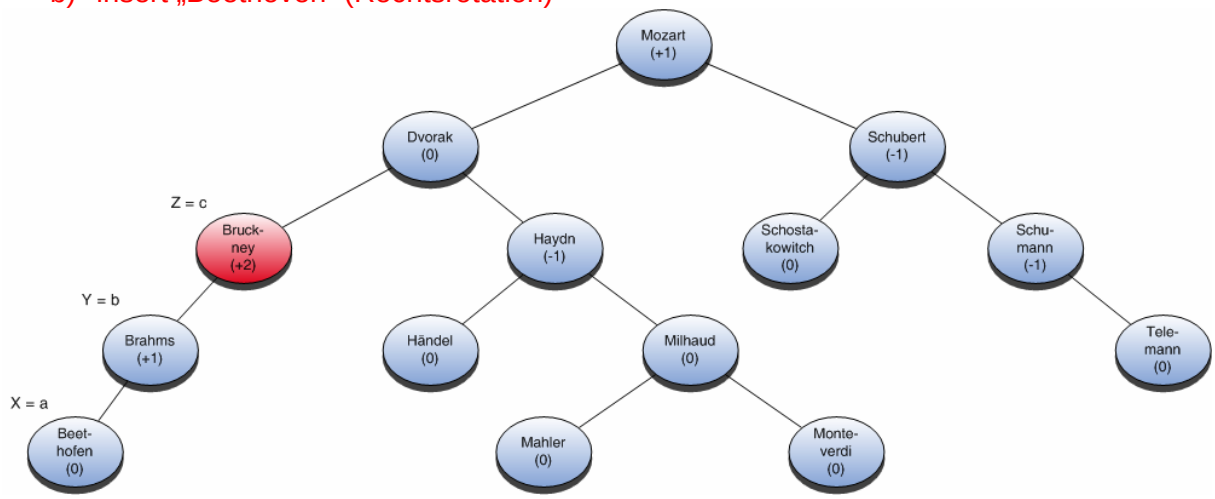
a) Notieren Sie für jeden Knoten des obigen Baumes den Balancierungsfaktor.
b) Fügen Sie Knoten mit den Schlüsseln "Beethoven" und "Zelenka" ein.
Aktualisieren Sie hierbei für jeden betroffenen Knoten die Balance, und führen Sie ggf. die nötigen Rebalancierungen durch, um die AVL-Eigenschaft zu erhalten.
c) Fügen Sie einen Knoten mit dem Schlüssel "Mendelssohn" ein.
Aktualisieren Sie auch hier die Balancierungsfaktoren und führen Sie die nötigen Rebalancierungen durch.
Zeichnen Sie den Zustand des Baumes nach jeder einzelnen Rotation.

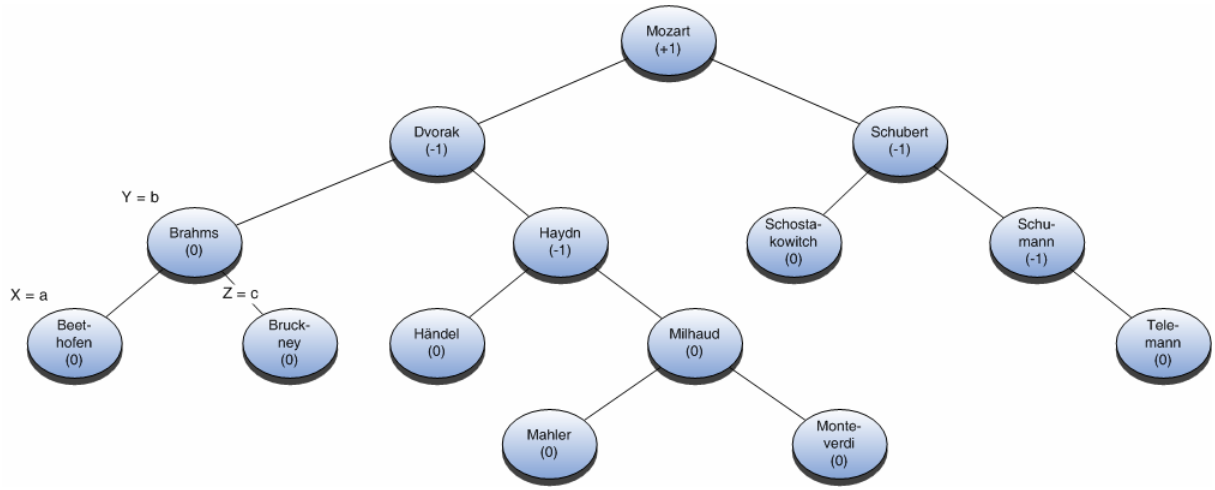
Lösungen:

a) Balancierungsfaktoren

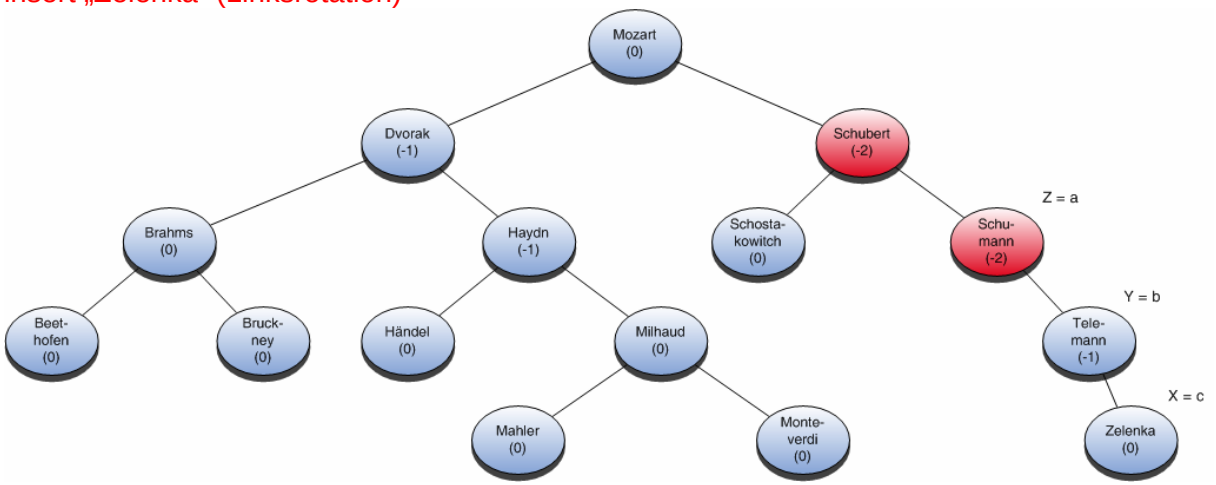


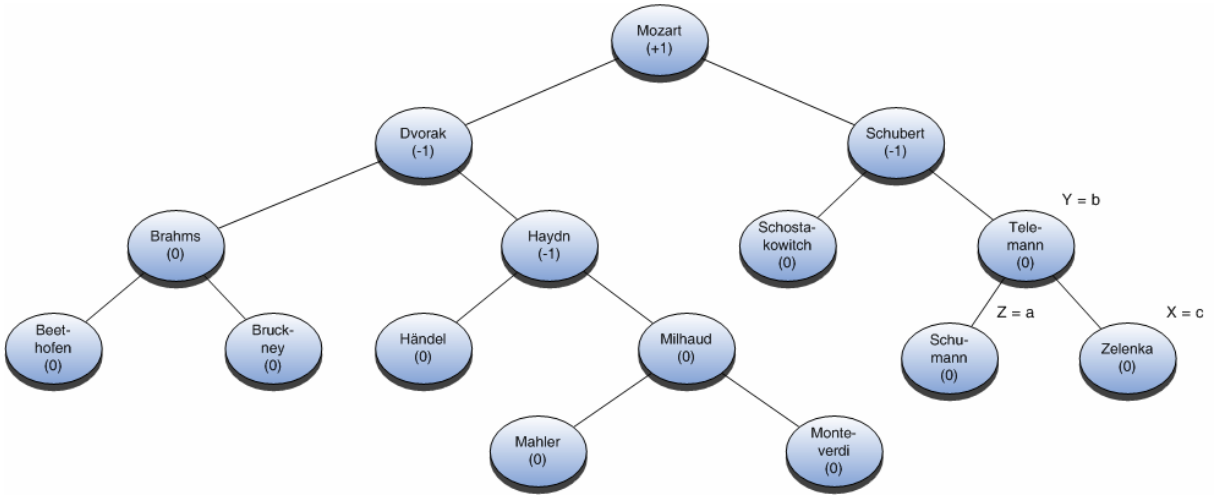
b) insert „Beethoven“ (Rechtsrotation)



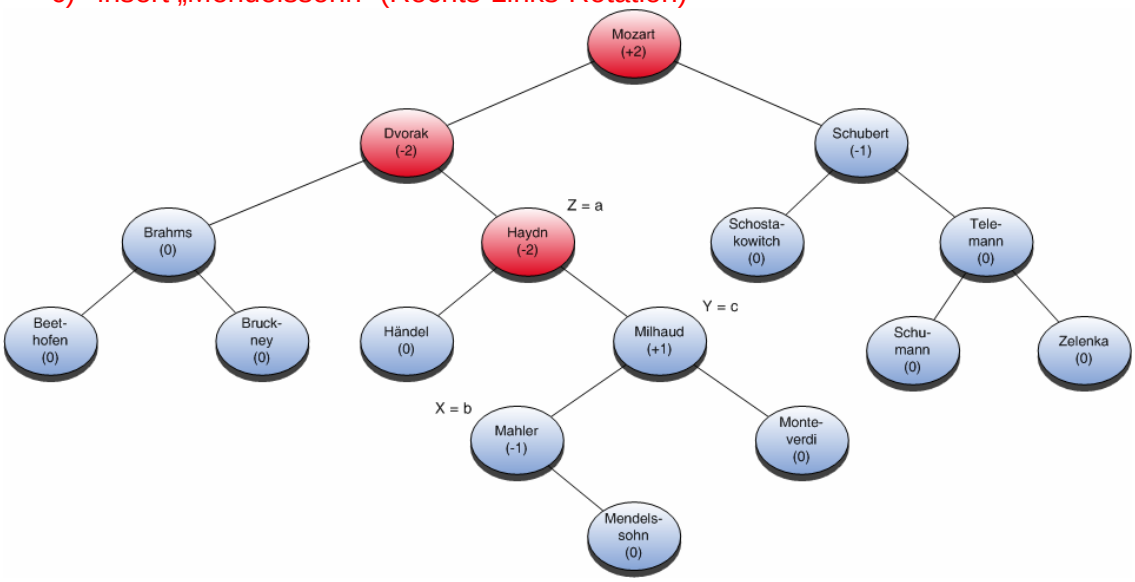


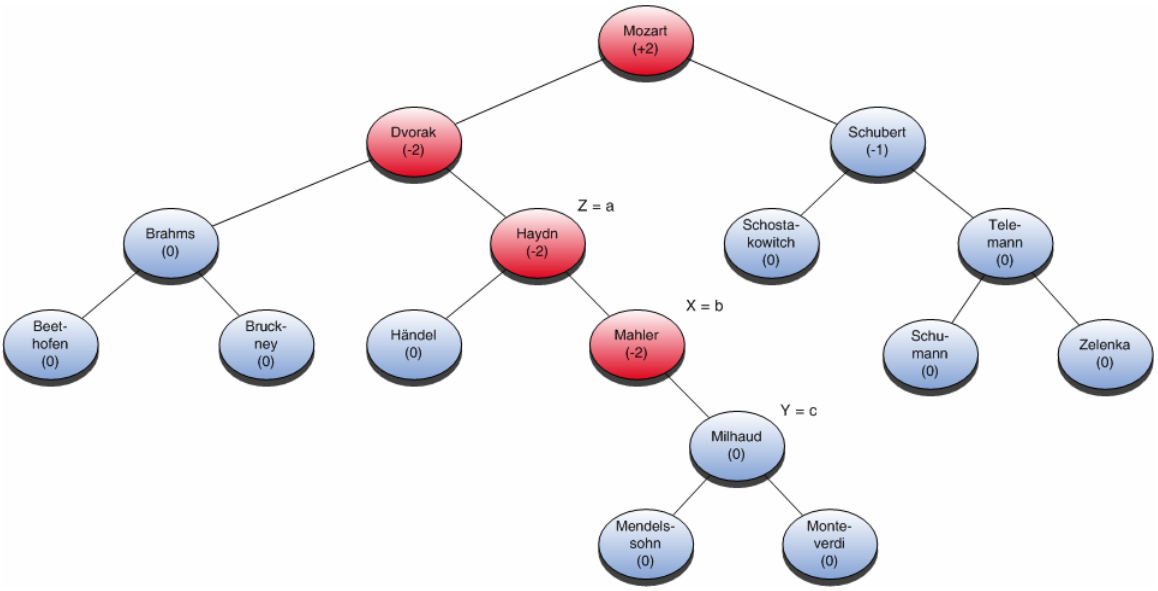
insert „Zelenka“ (Linksrotation)

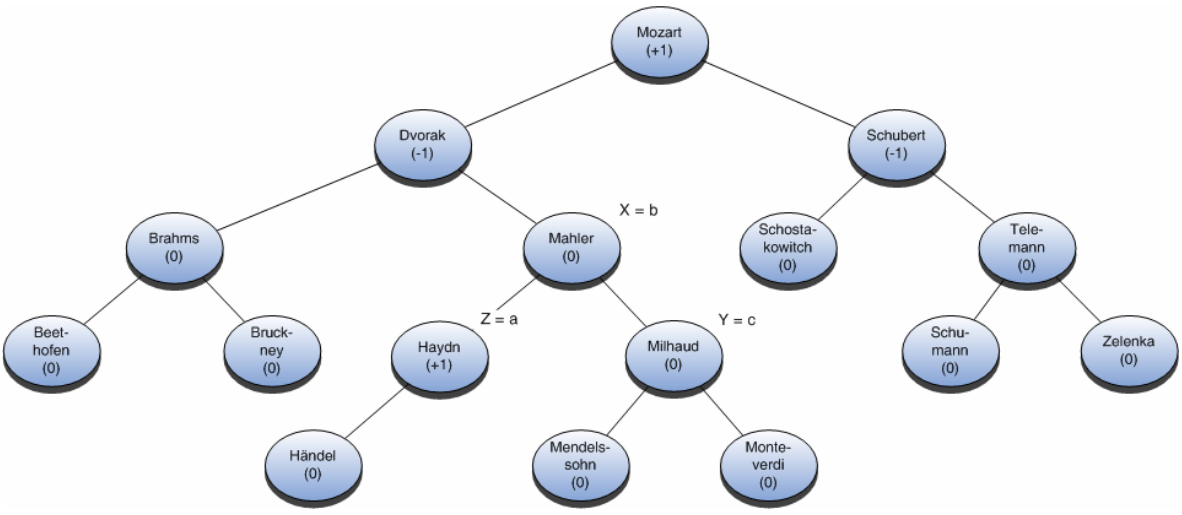




c) insert „Mendelssohn“ (Rechts-Links-Rotation)







Aufgabe 2 (Einfügen in AVL-Bäume – Cut/Link)

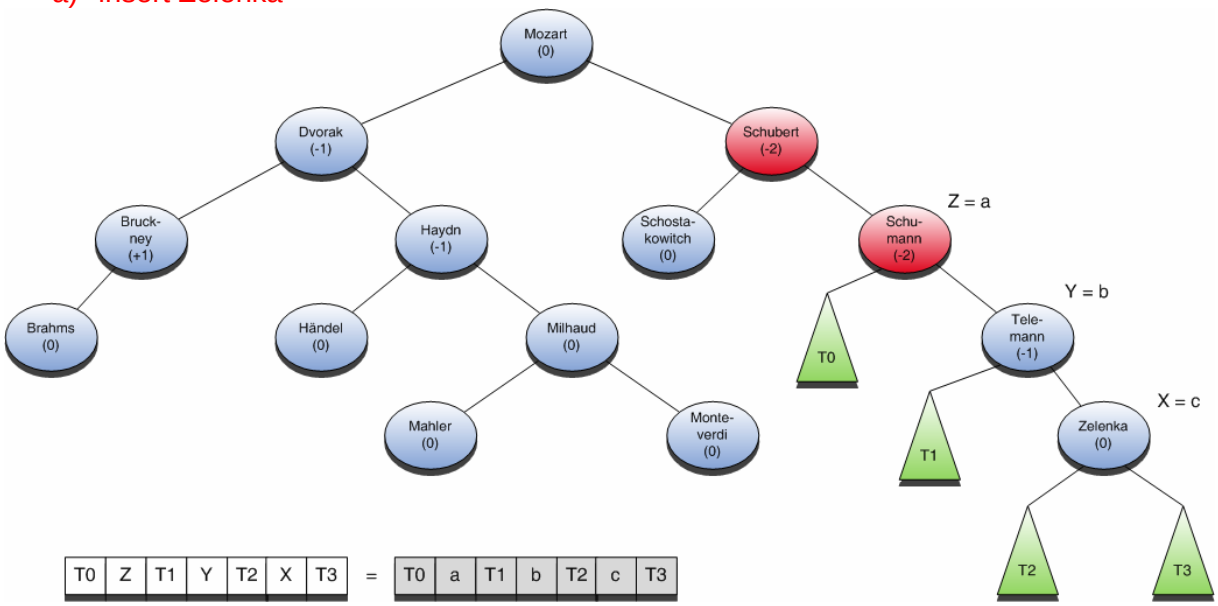
Gegeben sei nun der Ausgangsbaum von Aufgabe 1 (siehe Grafik):

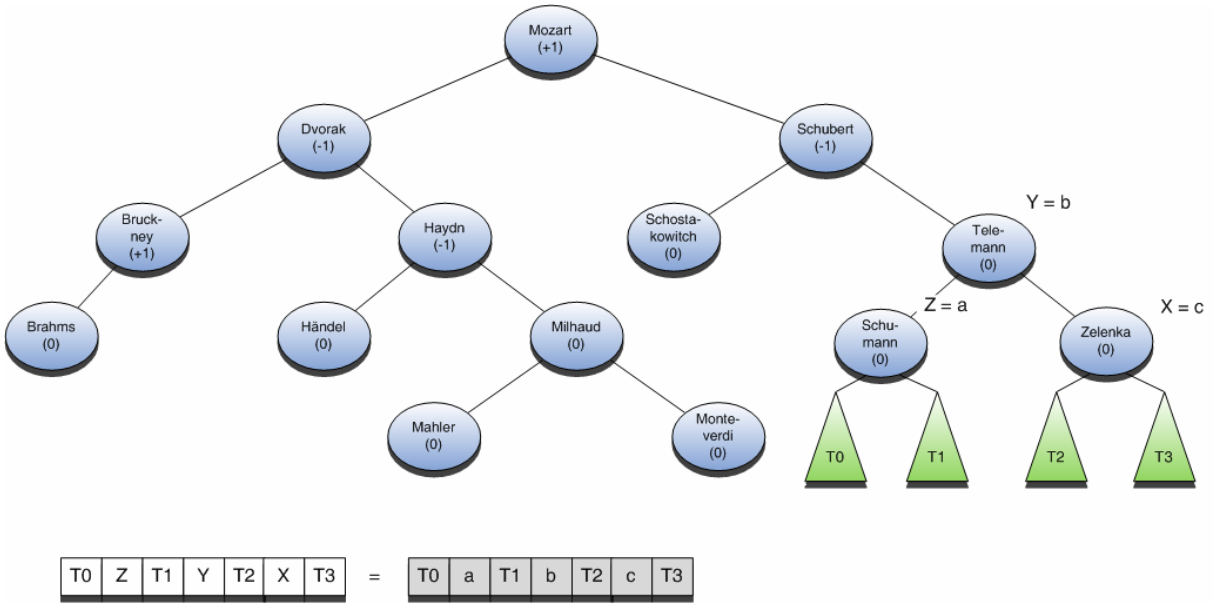
a) Fügen Sie Knoten mit dem Schlüssel "Zelenka" ein.
Führen Sie ggf. die nötige Restrukturierung mittels Cut/Link-Algorithmus durch, um die AVL-Eigenschaft zu erhalten, und geben Sie die Order-Arrays an.

b) Fügen Sie einen Knoten mit dem Schlüssel "Mendelssohn" ein.
Führen Sie auch hier die nötigen Restrukturierungen mittels Cut/Link-Algorithmus durch.

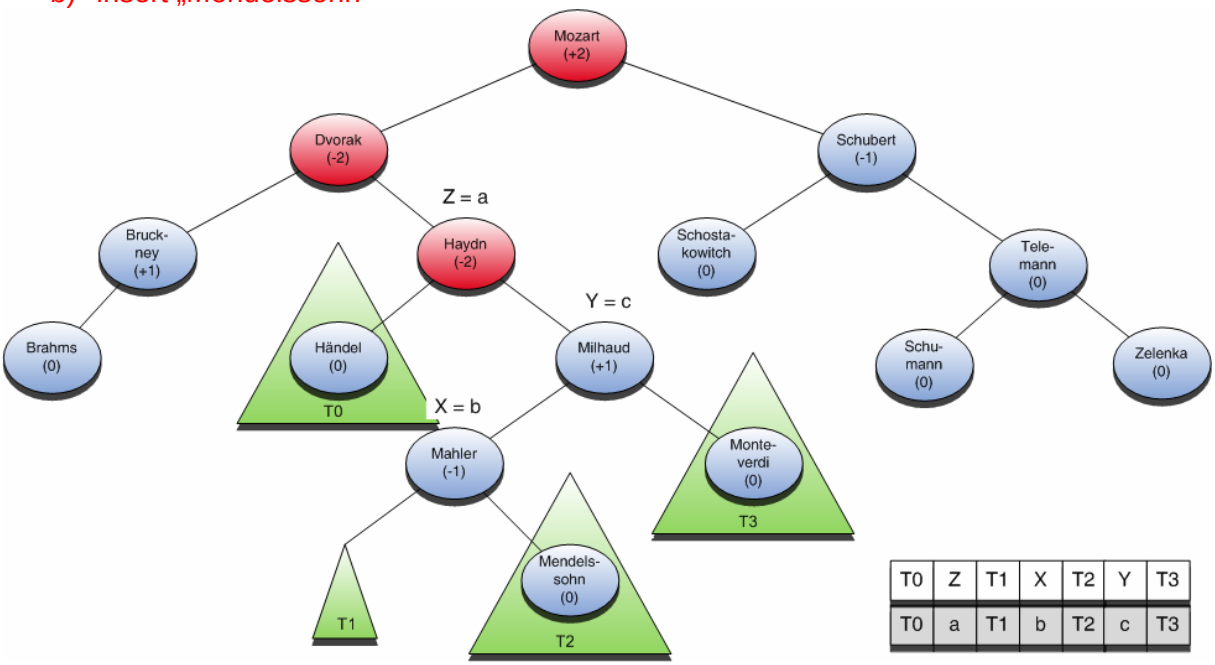
Lösung:

a) insert Zelenka

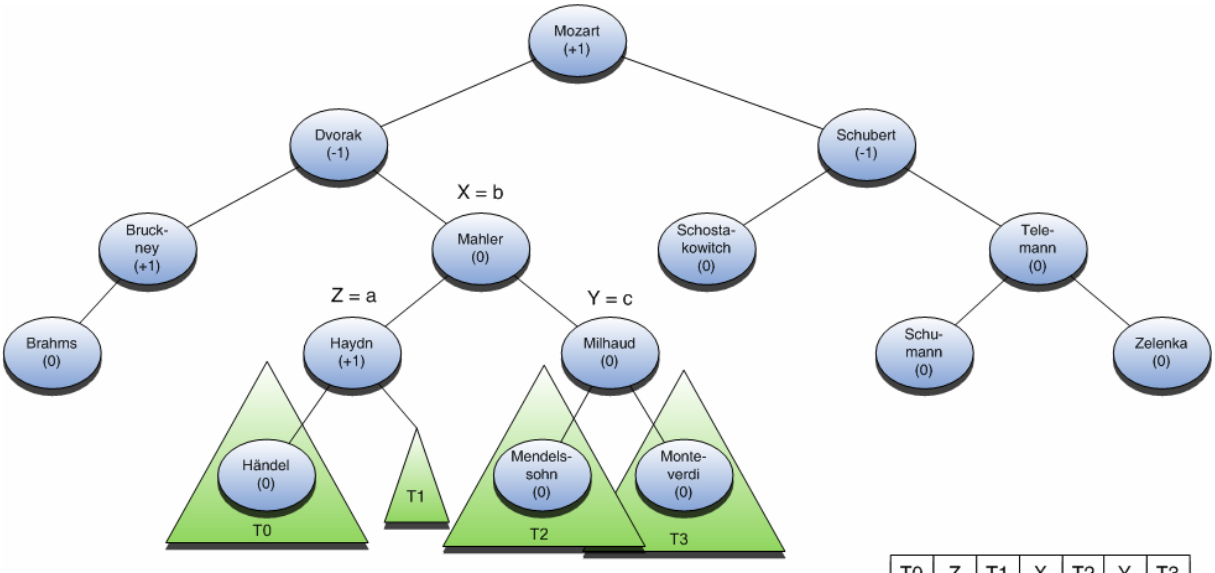




b) insert „Mendelssohn



T0	Z	T1	X	T2	Y	T3
T0	a	T1	b	T2	c	T3



T0	Z	T1	X	T2	Y	T3
T0	a	T1	b	T2	c	T3

Aufgabe 3 (Implementation AVL-Baum mit Rotationen: Teil 1)

Implementieren Sie einen AVL-Baum.
Beachten sie, dass es sich gemäss den Vorgaben um eine *Map* handeln soll.

Dabei soll der AVL-Baum auf dem Binary-Search-Tree von Übung 2 aufbauen (ohne das an jenem Binary-Search-Tree Anpassungen gemacht werden!).

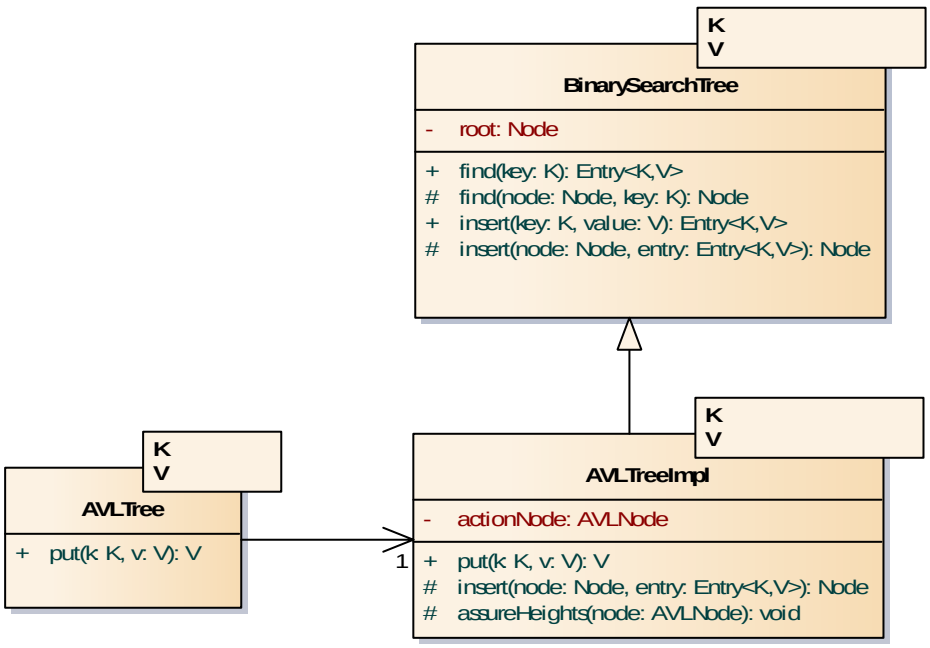
In einem ersten Teilschritt (diese Übung 3) sollen nur die Operationen zum *Einfügen* und *Suchen* realisiert werden, und zwar noch ohne Rotationen (dies wird dann in einem zweiten Teilschritt (Übung 4) realisiert).

Siehe Vorlage auf dem Skripte-Server.

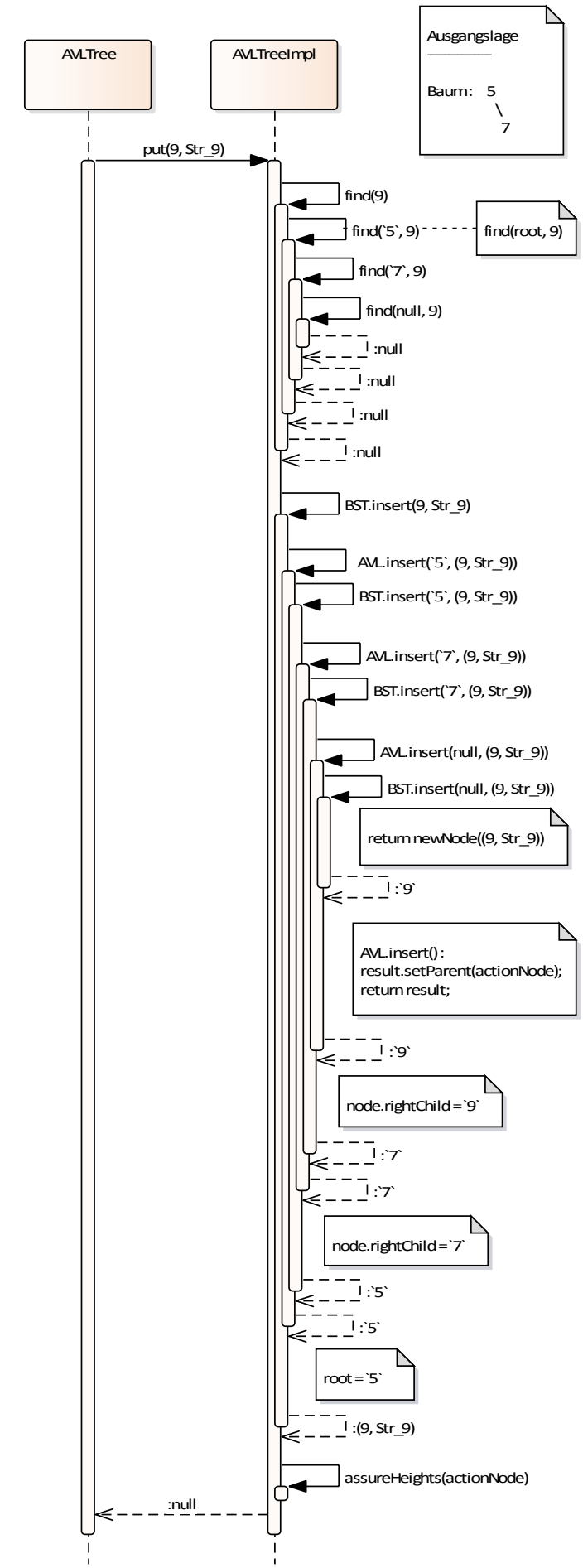
Lösung:

siehe „AVLTreeImpl.java“

Im Weiteren ein Klassendiagramm mit den beteiligten Methoden für ein *AVL.put()* :



Sowie ein Sequenzdiagramm für die Situation eines *put()* mit noch nicht bestehendem *Key* :


AVLTree
AVLTreeImpl
Ausgangslage
Baum: 5
\
7
put(9, Str_9)
find(9)
find(`5`, 9)
find(root, 9)
find(`7`, 9)
find(null, 9)
:null
:null
:null
:null
BST.insert(9, Str_9)
AVL.insert(`5`, (9, Str_9))
BST.insert(`5`, (9, Str_9))
AVL.insert(`7`, (9, Str_9))
BST.insert(`7`, (9, Str_9))
AVL.insert(null, (9, Str_9))
BST.insert(null, (9, Str_9))
return newNode((9, Str_9))
:`9`
AVL.insert():
result.setParent(actionNode);
return result;
:`9`
node.rightChild = `9`
:`7`
:`7`
node.rightChild = `7`
:`5`
:`5`
root = `5`
:(9, Str_9)
assureHeights(actionNode)
:null