# Lab: B-Trees-2-3-Trees and AVL Trees

This document defines the lab for ["Data Structures – Advanced (Java)" course @ Software University](https://softuni.bg/trainings/4276/data-structures-advanced-with-java-september-2023). Please submit your solutions (source code) of all below described problems in [Judge](https://judge.softuni.bg/Contests/2333/01-B-Trees-2-3-Trees-and-AVL-Trees-Lab).

Write Java code for solving the tasks on the following pages. Code should compile under the Java 8 and above standards you can write and locally test your solution with the Java 13 standard, however **Judge will run the submission with Java 10 JRE**. Avoid submissions with **features included after Java 10** release doing **otherwise** will result in a **compile time error**.

Any code files that are part of the task are provided as **Skeleton**. In the beginning, import the project skeleton, do not change any of the interfaces or classes provided. You are free to add additional logic in the form of methods in both interfaces and implementations you are not allowed to delete or remove any of the code provided. Do not change the names of the files as they are part of the test logic. **Do not change the packages** or move any of the files provided inside the skeleton if you have to add new file add it in the same package of usage.

Some **tests may be provided** within the skeleton – use those for local **testing and debugging**, however there **is no guarantee that there are no hidden tests added inside Judge**.

Please follow the exact instructions on uploading the solutions for each task. Submit as **.zip archive** the files contained inside **"...\src\main\java"** folder this should work for all tasks regardless of the current DS implementation.

In order for the solution to compile the tests **successfully** the project **must** have **single** **Main.java** file containing single **public static void main(String[] args)** method even empty one within the **Main class**.

Some of the problems will have simple **Benchmark** **tests** inside the skeleton. You can try to run those with **different** **values** and **different** **implementations** in order to **observe** behavior. However **keep** in mind that the result comes **only as numbers** and this data may be **misleading** in some situations. Also, the tests are not started from the command prompt which may **influence** the **accuracy** of the results. Those tests are only added as an **example** of **different** **data** **structures** **performance** on their **common** operations.

The Benchmark tool we are using is **JMH** (Java Microbenchmark Harness) and which is Java harness for building, running, and analyzing, **nano/micro/milli/macro** benchmarks written in Java and other languages targeting the JVM.

**Additional** **information** can be found here: [JMH](https://openjdk.java.net/projects/code-tools/jmh/) and there are other examples over the **internet**.

**Important:** when importing the skeleton **select** **import** **project** and then **select** **from** the **maven** **module**, this way any following **dependencies** will be **automatically** **resolved**. The project has **NO** **default** **version** of **JDK so after the import you may (depending on some configurations) need to specify the SDK, you can download** **JDK 13** from [**HERE**](https://jdk.java.net/13/)**.**

## AVL Tree Insertion

For this task submit **only AVL.java**, **Main.java** and **Node.java** filesas **.zip**

You are given a skeleton that supports the following operations:

* Node<T> root 🡪 returns the root of the AVL tree
* bool contains(T item) 🡪 checks if an element exists
* void eachInOrder(Consumer<T> consumer) 🡪 performs an action in order on each element
* void insert(T item) 🡪 inserts an item into the tree

|  |
| --- |
| **public class** AVL<T **extends** Comparable<T>> {   **private** Node<T> **root**;   **public** Node<T> getRoot() {  **return this**.**root**;  }   **public boolean** contains(T item) {  Node<T> node = **this**.search(**this**.**root**, item);  **return** node != **null**;  }   **public void** insert(T item) {  **this**.**root** = **this**.insert(**this**.**root**, item);  }   **public void** eachInOrder(Consumer<T> consumer) {  **this**.eachInOrder(**this**.**root**, consumer);  }   **private void** eachInOrder(Node<T> node, Consumer<T> action) {  **if** (node == **null**) {  **return**;  }   **this**.eachInOrder(node.**left**, action);  action.accept(node.**value**);  **this**.eachInOrder(node.**right**, action);  }   **private** Node<T> insert(Node<T> node, T item) {  **if** (node == **null**) {  **return new** Node<>(item);  }   **int** cmp = item.compareTo(node.**value**);  **if** (cmp < 0) {  node.**left** = **this**.insert(node.**left**, item);  } **else if** (cmp > 0) {  node.**right** = **this**.insert(node.**right**, item);  }   **return** node;  }   **private** Node<T> search(Node<T> node, T item) {  **if** (node == **null**) {  **return null**;  }   **int** cmp = item.compareTo(node.**value**);  **if** (cmp < 0) {  **return** search(node.**left**, item);  } **else if** (cmp > 0) {  **return** search(node.**right**, item);  }   **return** node;  } } |

And a node class:

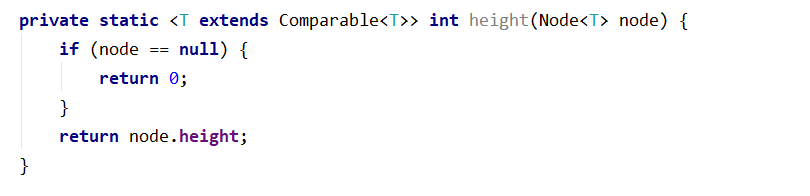
|  |
| --- |
| **public class** Node<T **extends** Comparable<T>> {   **public** T **value**;  **public** Node<T> **left**;  **public** Node<T> **right**;   **public int height**;   **public** Node(T value) {  **this**.**value** = value;  **this**.**height** = 1;  }  } |

Your task is to balance the tree after each insertion.

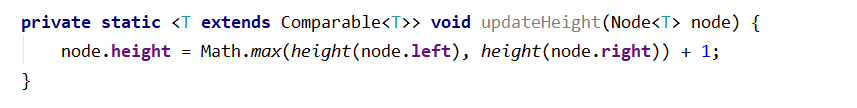
### Height

First, you should update the height of all nodes along an insertion path.

You will need a method to find a node's height.

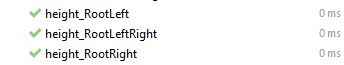


And a method to update a node's height.



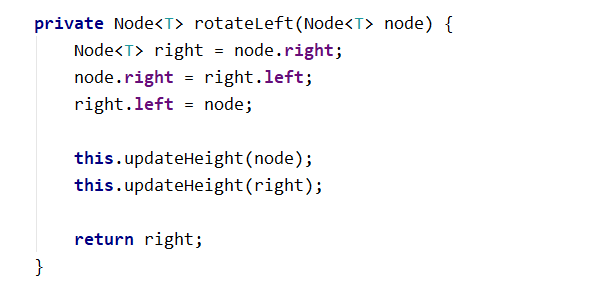
Consider when it is appropriate to update the height of a node.

Check if Height tests pass.

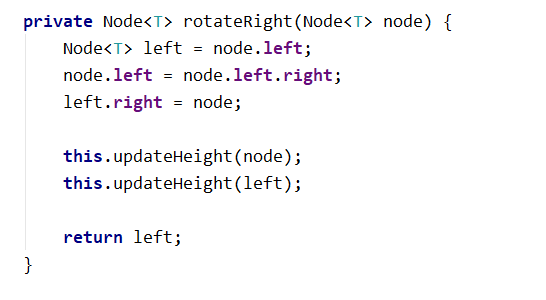


### Rotations

If you find it difficult to imagine the links that need to be updated in a rotation, refer to the presentation.

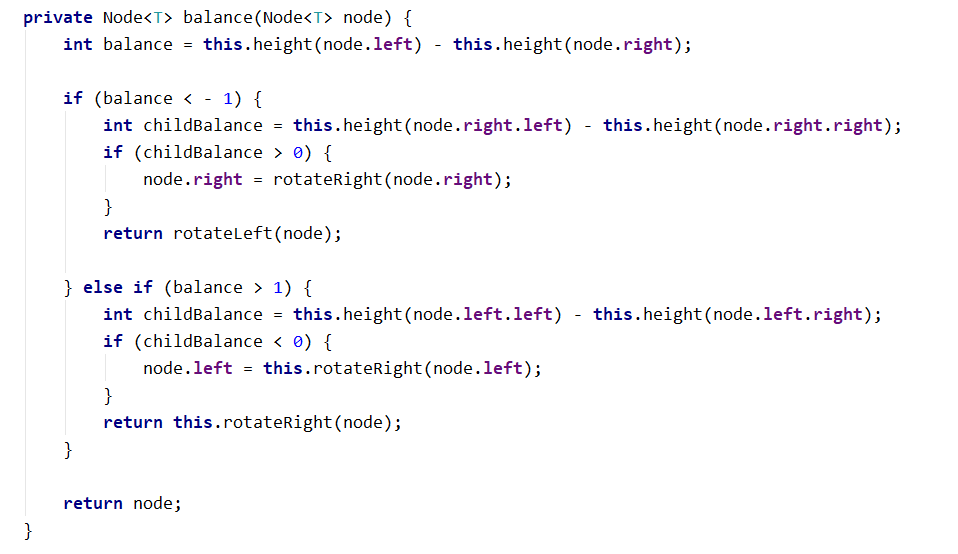


The rotation are analogous.

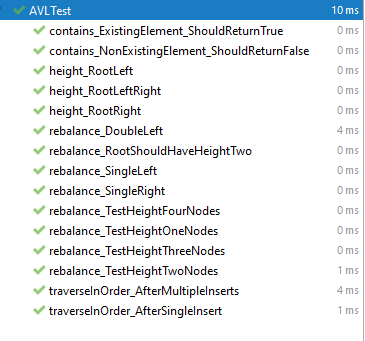


### Balancing

Start by creating the method.



Make sure that all tests pass:



Congratulations, you have completed the lab for AVL Trees.