TP2

DE52

stephane.galland@utbm.fr

Implementation of a binary tree and a binary search tree

00 | Objectif

The objective of this lab session is to create an information tree structure by applying implementation principles derived from software engineering.

The configuration of Eclipse should be the same as for previous sessions.

This lab assignment will be completed over

TWO OR THREE SESSIONS.

01 | Exercises

01.1 Implementation of a generic tree

A tree is a computer data structure that can be represented as a hierarchy where each element is called a node (node), with the initial node being called the root. In a tree, each element has at most n child elements at the lower level. From the perspective of these child elements, the element from which they originate at the higher level is called the parent. Each node also has a reference to its parent node (the node above it in the hierarchy), except for the root node, which has no parent node.

A binary tree is a tree for which n = 2.

In the context of this lab, we will only consider a tree that contains information, its data. This data does not have a predetermined type. This type must therefore be considered as a generic type (T).

The UML class diagram below will give you the structure and relationships between the types to be implemented at the end of the lab.

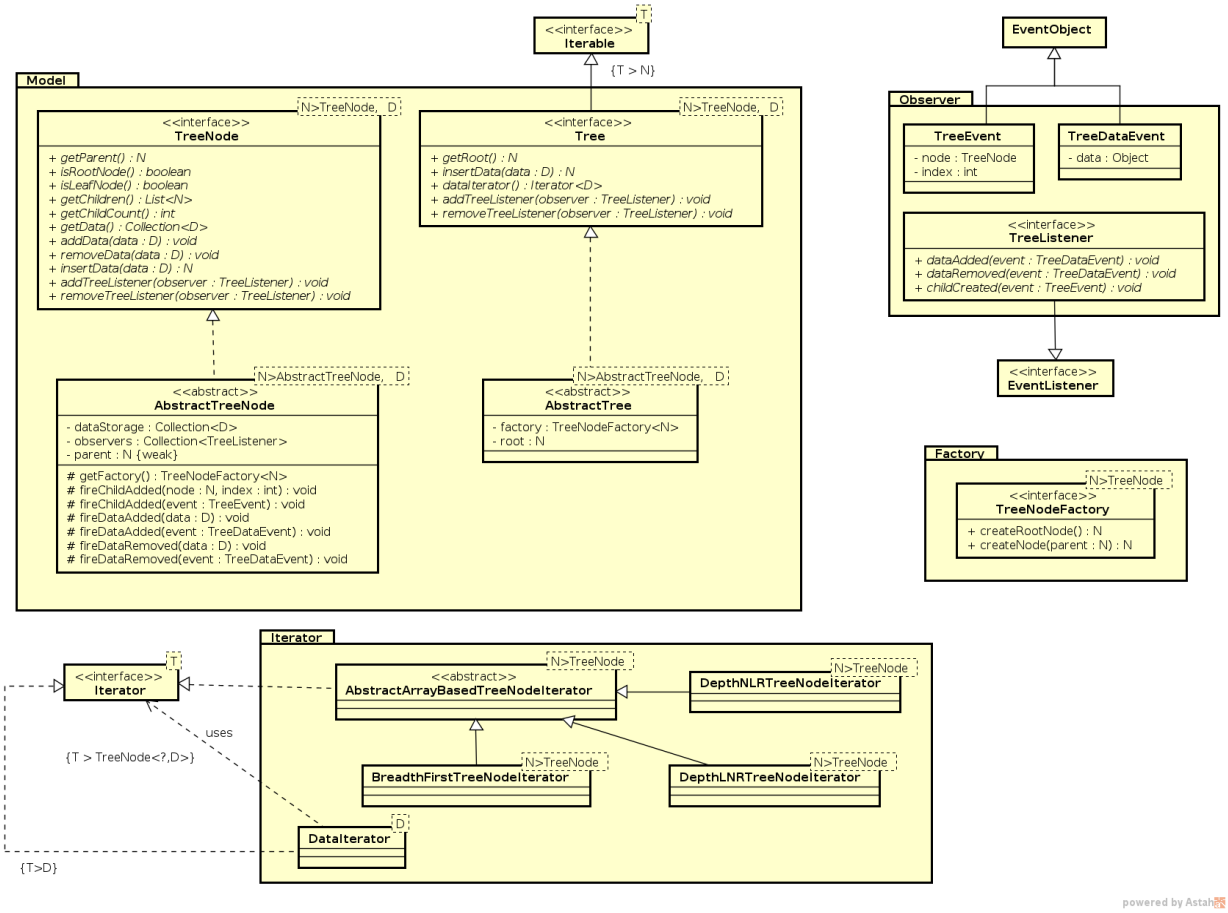

Figure 1: Class diagram for generic trees

You must adhere to the basic principles of programming inspired by software engineering best practices, namely:

1. Use clear names in English, as your code must be readable by any developer, regardless of their native language.

2. Correct the code so that there are no errors or warnings, as each error or warning indicates an anomaly related to good programming practices.

3. Write clear documentation in English (you will have warnings to help you).

4. Apply design patterns as much as possible: https://en.wikipedia.org/wiki/Design_Pattern

Work to be done:

A) Generic Tree Structure

- Write the Tree and TreeNode interfaces representing a tree and its nodes. These types must take 2 generic parameters:

    1. N: represents the type of nodes

2. D: represents the type of data stored in the nodes

- -There is a difference between the addData and insertData functions:

1. addData(data: D) adds the data to the current node (the node on which the function is called). It does not attempt to add the data to another node.

2. insertData(data: D) adds the data to the subtree whose root is the current node. In other words, this function adds the data either to a child node or to the current node, depending on the semantics and type of tree implemented (binary, ternary, search tree, etc.).

- -Write the abstract classes corresponding to each of the two interfaces implemented above.

B) Design Pattern: Factory

- The 'Factory' design pattern allows the creation of objects (new operator) to be externalized in a specific class called Factory.
- Define the TreeNodeFactory interface.
- Integrate a field containing the factory into the classes specified in A).

C) Design Pattern: Observer

- Write the TreeDataEvent interface as an object containing information corresponding to a modification of the data in the tree.
- Write the TreeEvent interface as an object containing information corresponding to a modification of the tree structure.
- Write the TreeListener interface representing an observer on the tree or on a node of the tree. The events to be considered are:

1. the addition of data to a node
2. the deletion of data from a node
3. the creation of a child node.

- Integrate the 'Observer' design pattern into the structure of the tree and nodes. Observers can observe a node or the entire tree.

D) Design Pattern: Iterator

- Two implementations of the 'Iterator' design pattern are necessary: one for iterating over the nodes of the tree, and a second for iterating over the data stored in the tree.
- Iteration over the nodes in the tree: there are different algorithms for traversing the nodes of a tree (https://en.wikipedia.org/wiki/Tree_traversal)

1. Create the breadth-first traversal iterator, which returns the nodes level by level from top (the root) to bottom (the leaves). Each call to the iterator returns a node.

2. Create the pre-order depth-first traversal iterator (NLR) that returns the parent node before its child nodes.

3. Create the in-order depth-first traversal iterator (LNR) that returns the parent node after its left node and before its right node. This iterator only works for binary trees.

- Create the iterator for the data stored in the tree using a reference to an iterator over the nodes. Indeed, to traverse all the data in a tree, you must first iterate over the nodes themselves, as the data is stored in the nodes.

01.2 Implementation of a binary tree

A binary tree is a tree for which n = 2.

The upper part of the UML class diagram below will give you the structure and relationships between the types to be implemented at the end of the lab for exercise 01.2.

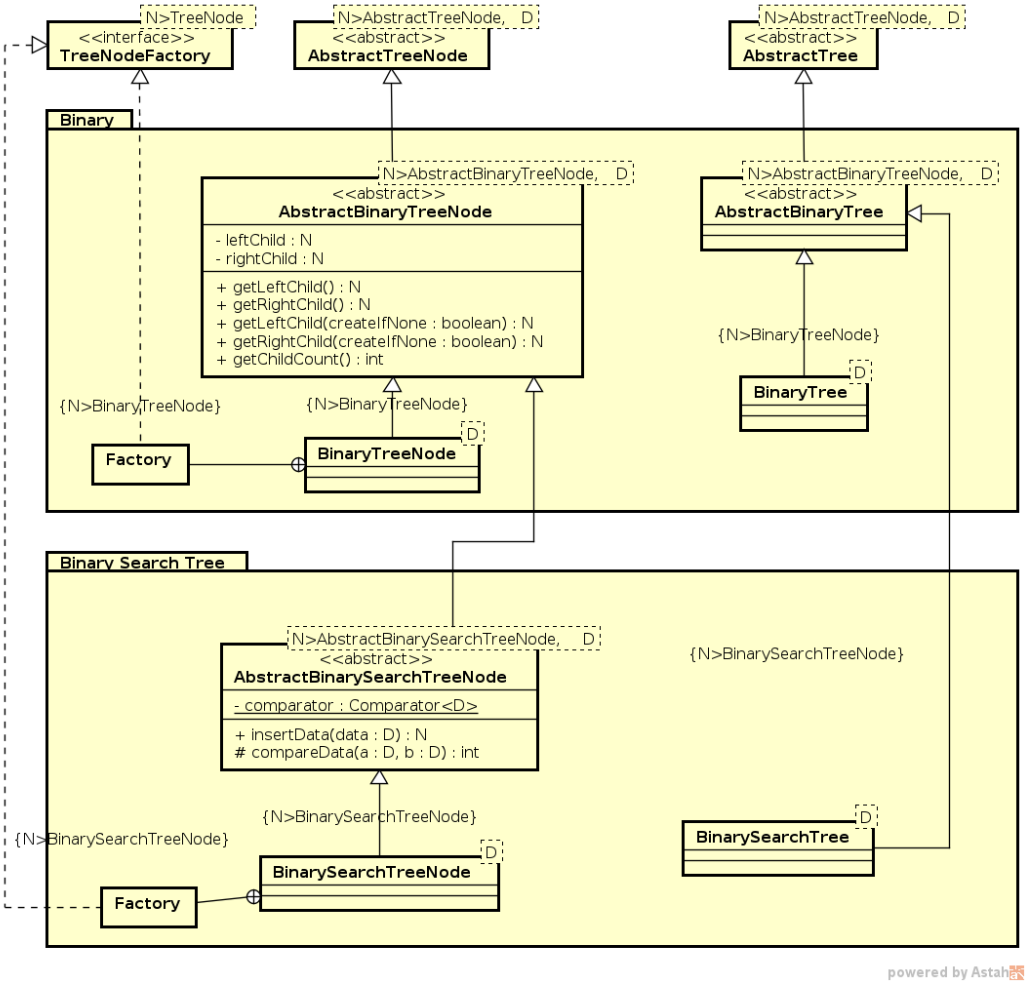

Figure 2: Class diagram for binary trees

A) Binary Tree Structure

- Create the abstract classes common to all binary tree implementations: AbstractBinaryTreeNode, AbstractBinaryTree.
- Create the concrete classes for a binary tree: BinaryTreeNode and BinaryTree.
- Create the Factory of BinaryTreeNode as an 'inner class' of the BinaryTreeNode class and use it to create the nodes of a binary tree.

01.3 Implementation of a binary search tree

A binary search tree is a binary tree where each node contains data, and the data stored in the left branch is less than the data stored in the root, and the data stored in the right branch is strictly greater than the data stored in the root. This notion of order is recursively repeated for each node in the tree (https://en.wikipedia.org/wiki/Binary_search_tree).

A) Binary Search Tree Structure

- Create the abstract class AbstractBinarySearchTreeNode including tools for comparing data according to their natural order (integer, etc.) or using a comparator (tools from the Java API). These tools will serve as support for the notion of order intrinsic to binary search trees.
- Create the concrete classes for a binary search tree: BinarySearchTreeNode and BinarySearchTree.
- Create the Factory of BinarySearchTreeNode as an 'inner class' of the BinarySearchTreeNode class and use it to create the nodes of a binary search tree.

B) Test Program

Write a 'main' function that performs the following actions:

1. Create a binary search tree of integers.
2. Add random integers to the tree.
3. Display the sorted list of integers stored in the tree using one of the iterators implemented in 01.1.

AT THE END OF THE TWO/THREE LAB SESSIONS, SEND A ZIP FILE CONTAINING THE JAVA CODE YOU PRODUCED DURING THE LAB