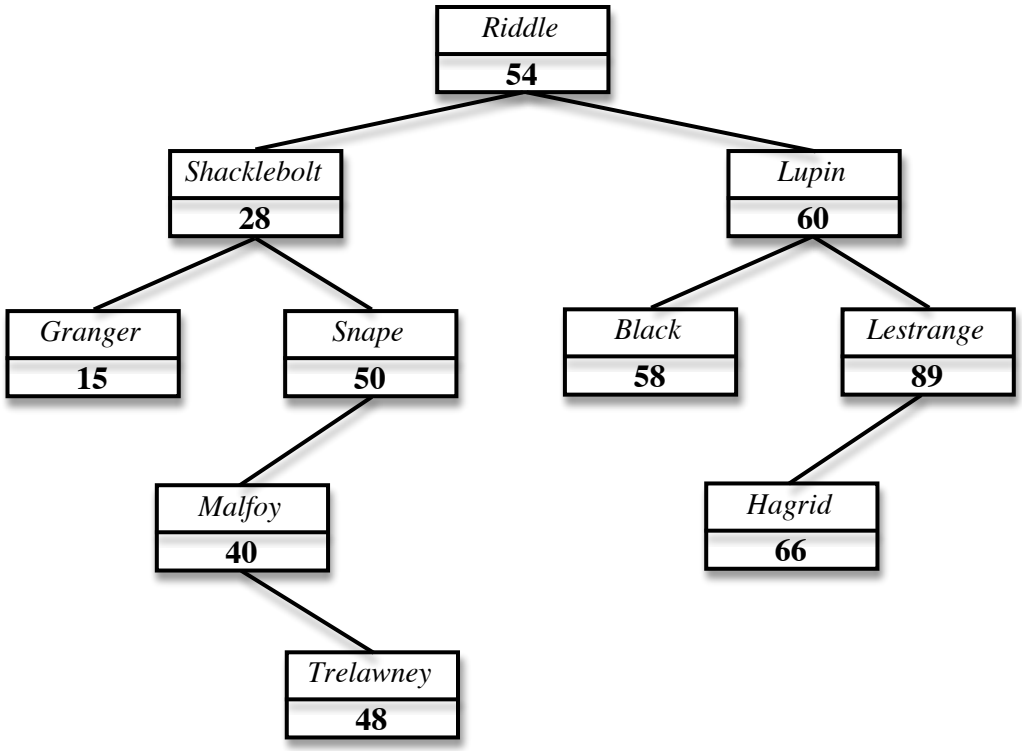
GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computing & Mathematics

CP2SD – Data Structures & Algorithms
Lab 3: Binary Trees

Binary trees are hierarchical data structures in which each node contains at most two child nodes. Nodes with children are parent nodes, and child nodes may or may not contain references to their parents. Child nodes may be labeled as left or right to denote their relationship with a parent. Due to its binary structure, a search of a binary tree can be accomplished in **O**(log *n*) time using a simple recursive algorithm. In this practical, we will create a binary tree using an ***insert()*** method and examine how the tree can be transversed in depth-first, pre, in and post order. We will also implement the binary search algorithm to search the tree in sub-linear time.



- Download the Zip archive ***lab6BinaryTrees.zip*** from Moodle and extract the files into a folder on your hard disk. As the classes involved are all packaged, ensure that you create the correct directory structure and study the source code before you compile.

- Examine the class *BinaryNode.java*. A node represents either a parent or a child in the binary tree. The instance variable *item* is declared as an *Item* type, meaning that any class that implements the *Item* interface can be added to the binary tree. In this example, we will be adding *Person* objects (*Person* implements *Item*) to the tree.

- The interface defined in ***Visitor.java*** declares a single method ***visit(BinaryNode node)*** and is implemented by the classes *PreOrderTransverser*, *InOrderTransverser* and *PostOrderTranverser*. Examine each of these classes and make sure you understand how the pseudocode (in comments at the top of each class) is actually implemented in Java.

- Execute the class ***BinaryTreeTest.java.*** The class constructs the binary tree in the ***init()*** method by repeatedly calling ***insert()*** at the root node. Examine the insert() and search() method and make sure you understand how the pseudocode is implemented in each method.

- Change the line *Visitor pot = new PreOrderTransverser()* to an in-order and post-order transversal and examine the order in which the nodes are visited.

Exercises

Using the following algorithm create a ***remove()*** method that is capable of removing a node from the binary search tree. Note that you will have to add left/right labeling to the *BinaryNode* class.

```
If root is Null
  Item is not in tree. Return Null.
Compare item to data at local root
If item < data at local root
  Return result of deleting from left subtree
Else if item > data at local root
  Return result of deleting from right subtree
Else //Equals. Item same as data in local root
  Store the data in local root in variable deletedLocal
  If local root has NO children
      Set parent of local root to Null
  Else If local root has ONE child
      Set parent of local root to reference that child
  Else //Node to remove has TWO children. Find in-order predecessor
      If left child has no right child //Found in-order predecessor
         Set parent of local root to reference left child
      Else
         Find rightmost node in right subtree of left child
         Copy its data to local root's data.
         Remove by setting its parent to reference its left child
```