Objectives

- (Practice) To create and use hierarchical structures in Scala with pattern matching
- (Theory) To understand how to traverse such structures using recursion and HOFs

Resources

You should refer to the following resources accessible via Blackboard:

- Topic 6 Lecture videos 5A – 5D
- **Topic 6 folder** (zipped) – includes the notes and slides for this topic
- External website for Scala doc and other **Learning Resources** (see the folder on Blackboard).

Introduction

Trees are data structures in which a hierarchy can be embedded. Each node in the tree has a unique predecessor (except the root) and a number of potential successors (depending upon the arity of the tree). A binary tree, for example, will allow every node to have a maximum of two successor nodes (children) and one predecessor node (parent). Ternary trees would allow up to three successors. N-ary trees allow for an arbitrary number of successors for each node. This type of structure reflects typical folder (directory (Unix)) structures on a computer where each folder can contain any number of sub-folders etc. A unary tree in which any node can have at most one successor is just another name for a singly-linked list.

While the tree represents the general abstraction, these data structures often appear with other names that reflect the application domain. For example, it is typical to represent arithmetic expressions using trees. In this case the tree would be called something like "Expr" and its internal nodes would become operators ("+", "*", etc.) and its terminal nodes (leaves) would become operands ("1", "2", etc.) Trees can be used in compression algorithms (Huffman encoding, for example) or to represent the (single) inheritance hierarchy within an object-oriented language. An n-ary tree would appear as an organisation chart showing departments with sub-departments, and so on. The important point is that all of these structures share a common pattern and are accessed in similar ways.

The expression tree is a classic example from computer science. It is both intuitive and familiar. We will show how the data structure can be represented in Scala and we will construct operations (functions and methods) to manipulate expressions. This places the hierarchical data structure within the context of a well-understood domain.

Our expressions will be Boolean rather than arithmetic. We will be looking to model expressions such as: P => Q + R == R + ~P + Q. The logical connectives need to be expressed in ASCII so we will use the following symbols: True, False, ~ (not), * (and), + (or), >> (implies), == (equivalence). We will also restrict ourselves to the propositional variables P, Q, and R. It would be easy to add more variables, but these will be sufficient for the purpose of this example. You may extend it if you wish.

Our second example in this topic involves an n-ary tree. This is used to represent an organisational hierarchy. To keep it manageable, the hierarchy contains only names (of employees). The data structure is organised around the concept of a Team which could represent the whole organisation, or could represent a single individual (a team of one). The idea is that the line manager relationship is captured by the data structure. The chief executive officer is at the top of the hierarchy, followed by the second level tier of management, then the third level, etc. Because an employee can manage any number of direct reports, an n-ary tree is used. This means that every person in the organisation (i.e. every node in the tree) is associated with a list of reports. If this list is empty then the individual does not manage anyone. The second exercise sheet has examples of the data structure in use and then asks you to develop some extra functions to manipulate the organisation hierarchy.

Finally, the third demonstration contains the definition of a binary search tree. This is written in the object style which is consistent with the style adopted by Scala's own collection class library. The examples in this demonstration are there to be read and tried as much as they are there to be extended. You are encouraged to develop some BST-processing functions of your own.

Activities

3.1 Watch the videos

Watch the videos 6A and 6B. The accompanying slides can be found in the topic-6-folder. These videos give you an introduction to the two case studies used in the demonstration code that accompanies this topic. One is a binary tree and the other an n-ary tree.

3.2 Install and run the TreeDemo programs

Open your Scala IDE and within the **src** folder move to the **demo.tree** package (you may need to create this **tree** package if this is the first time).

Copy the Scala files **TreeDemo1.scala, TreeDemo2.scala** and **TreeDemo3.scala** into the **tree** package. The exercises are embedded within these programs. Follow the instructions in the comments.