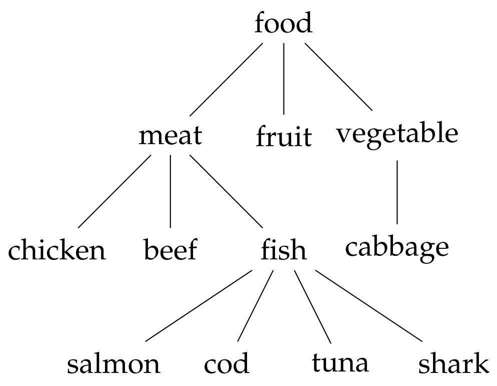
The basic recursive definition of a general tree (for computer scientists) which stores values from some class `T` is that a tree consists of a root which contains a value from `T`, together with a list (possibly empty) of trees that also store values from `T`.

In this lab you'll be completing the implementation of such a class providing further practice with recursive data structures and recursive methods on them. In the following descriptions, all examples given will be with reference to the tree below (with the labels thought of as `String`):



Part one (1%)

A basic implementation of the `Tree<T>` class has been provided for you in the directory `/home/cshome/coursework/241/pickup/10`. It includes some completed methods and some methods that need to be implemented by you. The declarations of the incomplete methods are listed below. Complete their implementation so that they have the intended behaviour.

`size()` Returns the number of nodes in the tree (in the example, 12).

`maxDegree()` Returns the largest number of children of any node (in the example, 4).

`add(Tree<T> child)` Adds the tree `child` as a new subtree below the root at the end of the list of children. The example tree could have been obtained by adding the tree with root "vegetable" and child "cabbage" to a tree with root "food", and subtrees as indicated rooted at "meat" and "fruit".

`postOrder()` Returns an `ArrayList<T>` which gives the contents of the nodes for the postorder traversal. In the example this would be:
chicken, beef, salmon, cod, tuna, shark, fish, meat, fruit, cabbage, vegetable, food
(as an `ArrayList<String>` of course.)

Part two (1%)

Another way to represent the contents of a tree as a string is using an indented list. Here, we first represent the root, and then each of its child subtrees, indented by say two spaces (and this is applied recursively). For the example tree we would get:

```
food
  meat
    chicken
    beef
    fish
      salmon
      cod
      tuna
      shark
  fruit
  vegetable
    cabbage
```

Implement a method `toIndentedString()` that performs this representation.

Reflection and extension

- Did you use recursive methods for your implementations? Why or why not? Think about other ways to do it – what advantages and disadvantages do they have?
- Perhaps a more useful function than `toIndentedString()` would be a method of reading such an indented string (e.g., from an input file) and constructing a `Tree` from it (most likely a `Tree<String>`). How might one implement such a function?
- For testing purposes and building trees one might like to generate trees randomly in some way. How could one do that? One issue would be to ensure that we keep the size within reasonable bounds while allowing “interesting” branching and depth.