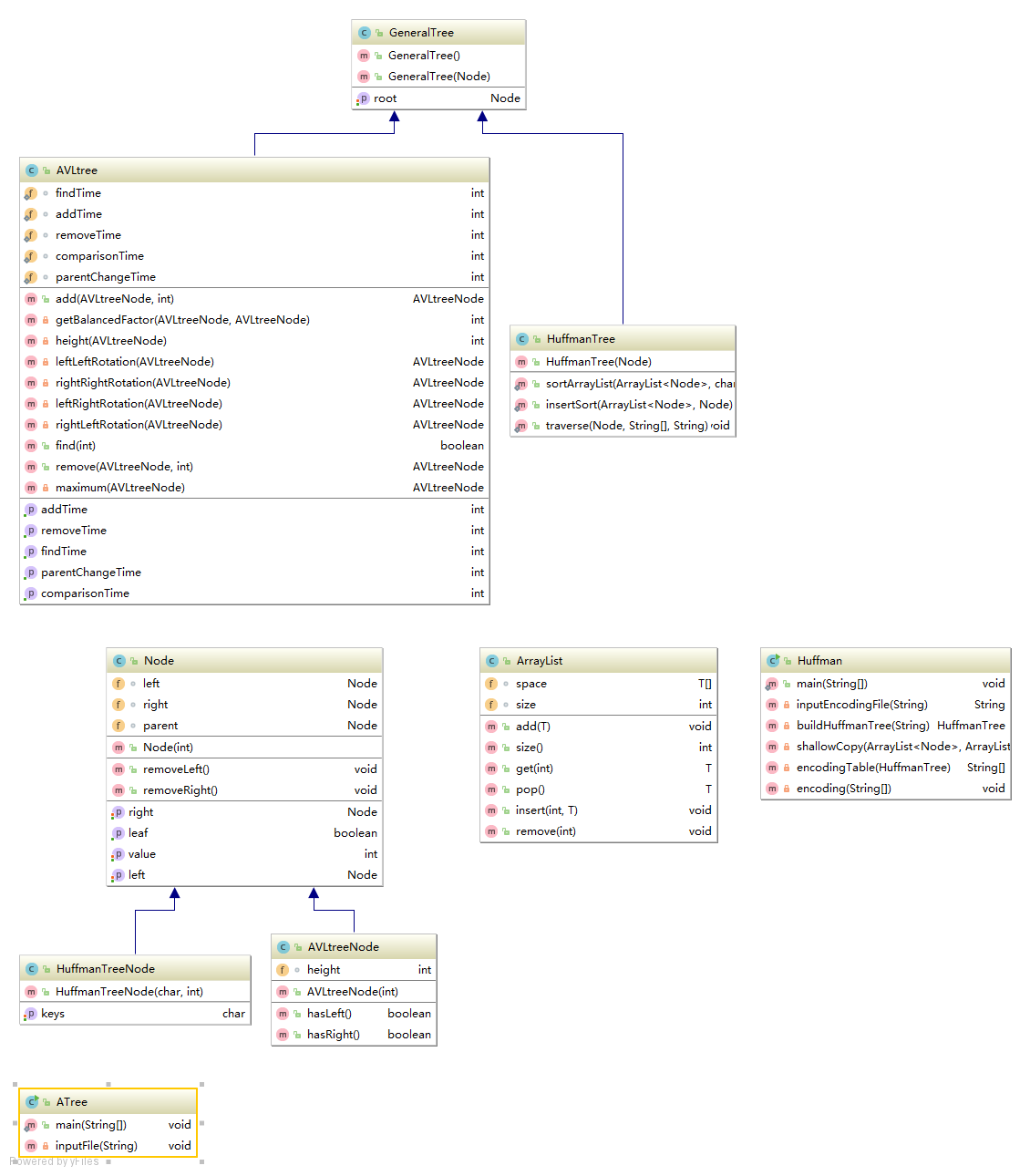
Part1 (b), (d) and (f):



Part 1 (c):

I used a class called Node to build the general tree, it has left node, right node, a value to store in and its parent as its properties and the root is a Node type in general tree.

Part 1 (e):

HuffmanTree inherits the GenenralTree class as well as its node-type HuffmanTreeNode from Node class. HuffmanTree class has the access to its father with more of its own methods. Node class only has a value property, and Huffman tree needs one more, which is the character key to store in.

Part 1(g):

AVLtree inherits the GenenralTree class as well as its node-type AVLtreeNode from Node class. Beside methods are inherited from GenenralTree, AVLtree also has more its own methods and properties to help build AVL Tree. Similarly AVLtreeNode has its father’s properties and methods also with its own, like height, which is needed for AVL tree.

Part 1(h):

I chose AVL tree for this design, because it is easier to add, find, remove and its worst time complexity is O (Log N) for those operations in theory. Also, AVL has the shallowest depth from its root to leafs, so my computer could handle it.

Part 4(a):

(2)

40038183 my rave waits above the noticed dance.

Result:

01010101000001010101110101100111000010010111110111011111100001111001101100101100111000101110001110001101001101011110111010011110110000011000011111010101001111011001011

Note: this result is generated from ‘m’ to ‘.’ including spacing.

(3)

“my rave waits above the noticed dance” has 38 characters ,so fixed length ASCII is 38\*8 = 304 bits

Result generated from Huffman tree has 167 bits. So 45% fewer bits were used.

(4)

From the file we used to build Huffman tree, most frequent word is ‘ ’, so I test it to get 00.Then, least frequent word ’x’ is 1100101010, which is 10 bits larger than fixed. From these results I can see this Huffman tree was built successfully, since it has Huffman tree’s features, so the test string should be alright.

Part4 (b):

(1)

Output:

The total number of comparisons is 51209

The total number of times a node’s parent changes is 1396

The total number of find operations is 385

The total number of add operations is 1000

The total number of remove operations is 178

(2)

From debugged results in IDE, I can see this AVL tree’s nodes are balanced since a node always has balanced left and right tree. Also, parent changing did not affect BST structure, since a node’s left was always less or equal than this node and right was always larger than it. My AVL tree is single linked from root, so it is ok for simple requirements. Double linked would be better for future design. Also, we only had less than 3000 words test case, so maybe S-play tree would be better design since it does not need to rebalance each time adding or removing ,so we do no need to keep track of nodes’ height changing.