Foreach in Java :

for (char c : s)  
 System.*out*.print(c);

List<Integer> to int[]:

list.stream().mapToInt(i->i).toArray();

Use ArrayDeque<> (Deque is interface) instead of Stack

Queue<Pair<NaryNode[], Integer>> q = **new** LinkedList<Pair<NaryNode[], Integer>>(); - queue

**List**

List<Integer> results = **new** ArrayList<>();

results.add(object),

results.get(index)

results.size()

**array**[]

int[] memo = new int[]{1};

memo.length

memo[1] = 2 // add. var i=memo[1] //get

NaryNode[] array = n.children.toArray(**new** NaryNode[0]);//Convert List of nodes to array of nodes

int[] array2 = list.stream().mapToInt(i -> i).toArray(); // converts List of ints to array of ints

**2d Array**

int[][] multi = new int[2][10];

or

int[][] multi = new int[2][];

multi[0] = new int[10];

multi[1] = new int[10];

**StringBuilder**

var sb = **new** StringBuilder("01")

sb.append(sb.charAt(j) == '0' ? "01" : "10"); // append, char at. Ternary op

sb.setLength(4);

**Queue //Use linkedList**

Queue<TreeNode> queue = **new** LinkedList<>();

queue.add(root);

var node = queue.poll();

**//using Queue**

Deque<Pair<TreeNode, Integer>> q = **new** ArrayDeque<>();

q.push(obj)

var item = q.pop()

You cannot add nulls to Deque/ArrayDeque

You can add nulls to Queue/LinkedList

**Stack**

Stack<TreeNode> stack = **new** Stack<>();

stack.push(root);

var node = stack.pop()

**HashMap**

**Map<k,v> map = new HashMap<>();**

**Add**

**put**

**Pair //Tuple**

**Java.util.Map**

Stack<java.util.Entry<TreeNode, Integer>> stack = new Stack<>();

Stack.push<New java.util.AbstractMap. SimpleEntry<TreeNode, Integer>(root, 2)

item.getKey() // = object

item.getValue() // = int

Don’t use

**Binary**

current.getValue() << 1 | node.left.vall // doubles current (as its binary \*2) and adds val to it

**Custom Comparator**

**public** **int** closestValue(TreeNode root, **double** target) {

List<Integer> nums = **new** ArrayList();

inorder(root, nums);

**return** Collections.min(nums, **new** Comparator<Integer>() {

@Override

**public** **int** compare(Integer o1, Integer o2) {

**return** Math.*abs*(o1 - target) < Math.*abs*(o2 - target) ? -1 : 1;

}

});

}