

DAY-7&8 JAVA ASSIGNMENT

Day 7 and 8:

Task 1: Balanced Binary Tree Check

Write a function to check if a given binary tree is balanced. A balanced tree is one where the height of two subtrees of any node never differs by more than one.

```
1 package tree;
2
3 public class BalancedBinaryTree {
4
5     // TreeNode class definition
6     static class TreeNode {
7         int val;
8         TreeNode left, right;
9     }
10    TreeNode(int item) {
11        val = item;
12        left = right = null;
13    }
14 }
15
16 // Method to check if the tree is balanced
17 public boolean isBalanced(TreeNode root) {
18     return checkHeight(root) != -1;
19 }
20
21 // Helper method to check the height of the tree
22 private int checkHeight(TreeNode node) {
23     if (node == null) {
24         return 0;
25     }
26
27     int leftHeight = checkHeight(node.left);
28     if (leftHeight == -1) return -1;
29
30     int rightHeight = checkHeight(node.right);
31     if (rightHeight == -1) return -1;
32
33     if (Math.abs(leftHeight - rightHeight) > 1) {
34         return -1;
35     }
36
37     return Math.max(leftHeight, rightHeight) + 1;
38 }
39
40 // Driver code to test the functionality
41 public static void main(String[] args) {
42     BalancedBinaryTree tree = new BalancedBinaryTree();
43     TreeNode root = new TreeNode(1);
44     root.left = new TreeNode(2);
45     root.right = new TreeNode(3);
46     root.left.left = new TreeNode(4);
47     root.left.right = new TreeNode(5);
48     root.right.right = new TreeNode(6);
49     root.left.left.left = new TreeNode(7);
50
51     System.out.println(tree.isBalanced(root));
52 }
53 }
54
55
```

Console: <terminated> Balance true

Task 2: Trie for Prefix Checking

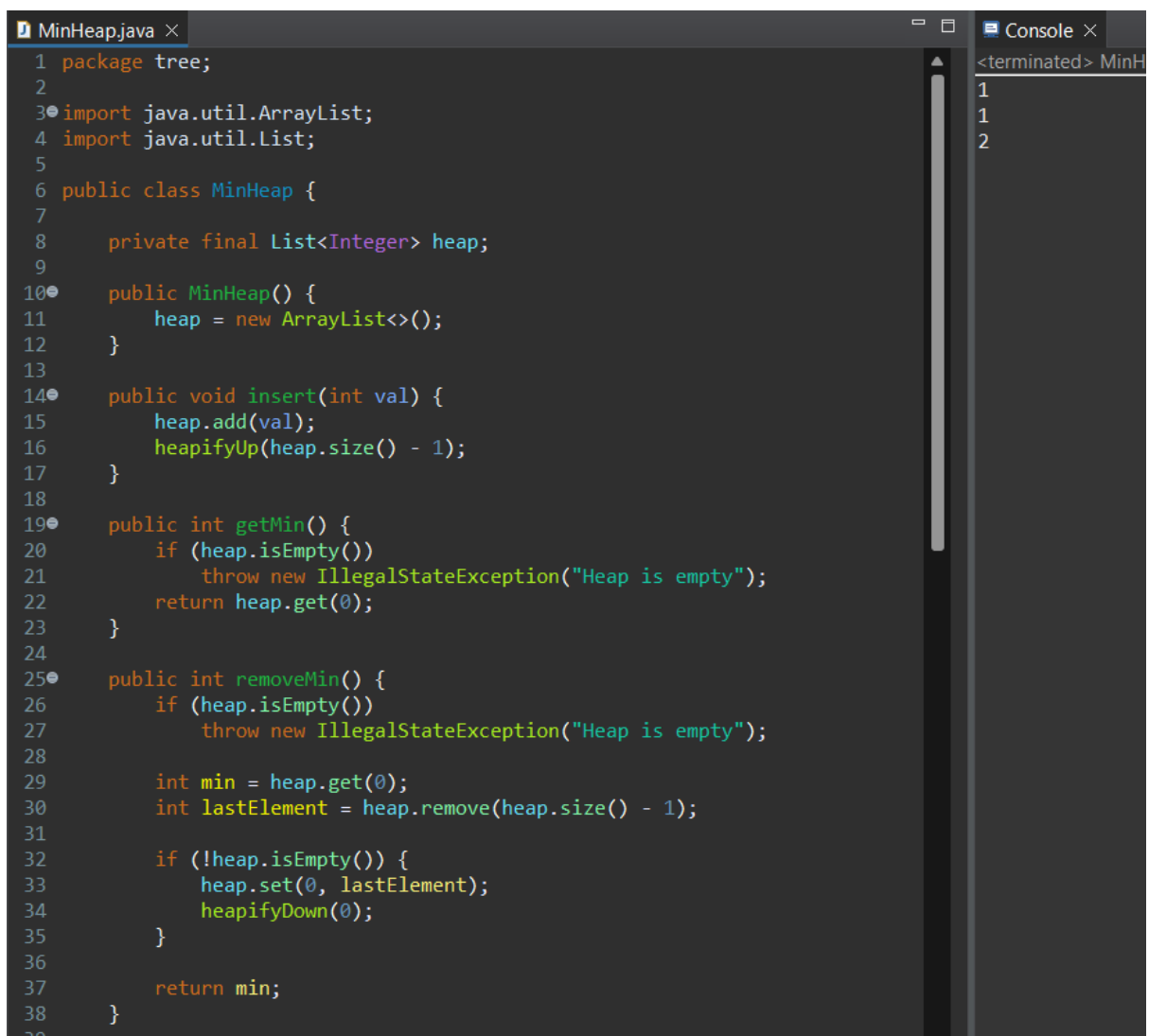
Implement a trie data structure in JAVA that supports insertion of strings and provides a method to check if a given string is a prefix of any word in the trie.

```
Trie.java x Console x
1 package tree;
2
3 import java.util.HashMap;
4 import java.util.Map;
5
6 public class Trie {
7
8     static class TrieNode {
9         Map<Character, TrieNode> children;
10         boolean isEndOfWord;
11
12         TrieNode() {
13             children = new HashMap<>();
14             isEndOfWord = false;
15         }
16     }
17
18     private final TrieNode root;
19
20     public Trie() {
21         root = new TrieNode();
22     }
23
24     public void insert(String word) {
25         TrieNode node = root;
26         for (char ch : word.toCharArray()) {
27             node = node.children.computeIfAbsent(ch, k -> new TrieNode());
28         }
29         node.isEndOfWord = true;
30     }
31
32     public boolean isPrefix(String prefix) {
33         TrieNode node = root;
34         for (char ch : prefix.toCharArray()) {
35             node = node.children.get(ch);
36             if (node == null) {
37                 return false;
38             }
39         }
40         return true;
41     }
42
43     // Driver code to test the Trie
44     public static void main(String[] args) {
45         Trie trie = new Trie();
46         trie.insert("apple");
47         trie.insert("app");
48         trie.insert("banana");
49
50         System.out.println(trie.isPrefix("app")); // Output: true
51         System.out.println(trie.isPrefix("ban")); // Output: true
52         System.out.println(trie.isPrefix("bat")); // Output: false
53     }
54 }
55
```

<terminated> Trie [Java
true
true
false

Task 3: Implementing Heap Operations

Code a min-heap in JAVA with methods for insertion, deletion, and fetching the minimum element. Ensure that the heap property is maintained after each operation.



```
1 package tree;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class MinHeap {
7
8     private final List<Integer> heap;
9
10    public MinHeap() {
11        heap = new ArrayList<>();
12    }
13
14    public void insert(int val) {
15        heap.add(val);
16        heapifyUp(heap.size() - 1);
17    }
18
19    public int getMin() {
20        if (heap.isEmpty())
21            throw new IllegalStateException("Heap is empty");
22        return heap.get(0);
23    }
24
25    public int removeMin() {
26        if (heap.isEmpty())
27            throw new IllegalStateException("Heap is empty");
28
29        int min = heap.get(0);
30        int lastElement = heap.remove(heap.size() - 1);
31
32        if (!heap.isEmpty()) {
33            heap.set(0, lastElement);
34            heapifyDown(0);
35        }
36
37        return min;
38    }
39 }
```

Console output:

```
<terminated> MinH
1
1
2
```

```

39
40● private void heapifyUp(int index) {
41     while (index > 0) {
42         int parentIndex = (index - 1) / 2;
43         if (heap.get(index) >= heap.get(parentIndex))
44             break;
45
46         swap(index, parentIndex);
47         index = parentIndex;
48     }
49 }
50
51● private void heapifyDown(int index) {
52     int size = heap.size();
53     while (index < size) {
54         int leftChildIndex = 2 * index + 1;
55         int rightChildIndex = 2 * index + 2;
56         int smallestIndex = index;
57
58         if (leftChildIndex < size && heap.get(leftChildIndex) < heap.get(smallestIndex)) {
59             smallestIndex = leftChildIndex;
60         }
61
62         if (rightChildIndex < size && heap.get(rightChildIndex) < heap.get(smallestIndex)) {
63             smallestIndex = rightChildIndex;
64         }
65
66         if (smallestIndex == index)
67             break;
68
69         swap(index, smallestIndex);
70         index = smallestIndex;
71     }
72 }
73
74● private void swap(int index1, int index2) {
75     int temp = heap.get(index1);
76     heap.set(index1, heap.get(index2));
77     heap.set(index2, temp);
78 }
79
80 // Driver code to test the MinHeap
81● public static void main(String[] args) {
82     MinHeap minHeap = new MinHeap();
83     minHeap.insert(3);
84     minHeap.insert(1);
85     minHeap.insert(6);
86     minHeap.insert(5);
87     minHeap.insert(2);
88     minHeap.insert(4);
89
90     System.out.println(minHeap.getMin()); // Output: 1
91     System.out.println(minHeap.removeMin()); // Output: 1
92     System.out.println(minHeap.getMin()); // Output: 2
93 }
94 }
95

```