BridgeLabz

Employability Delivered

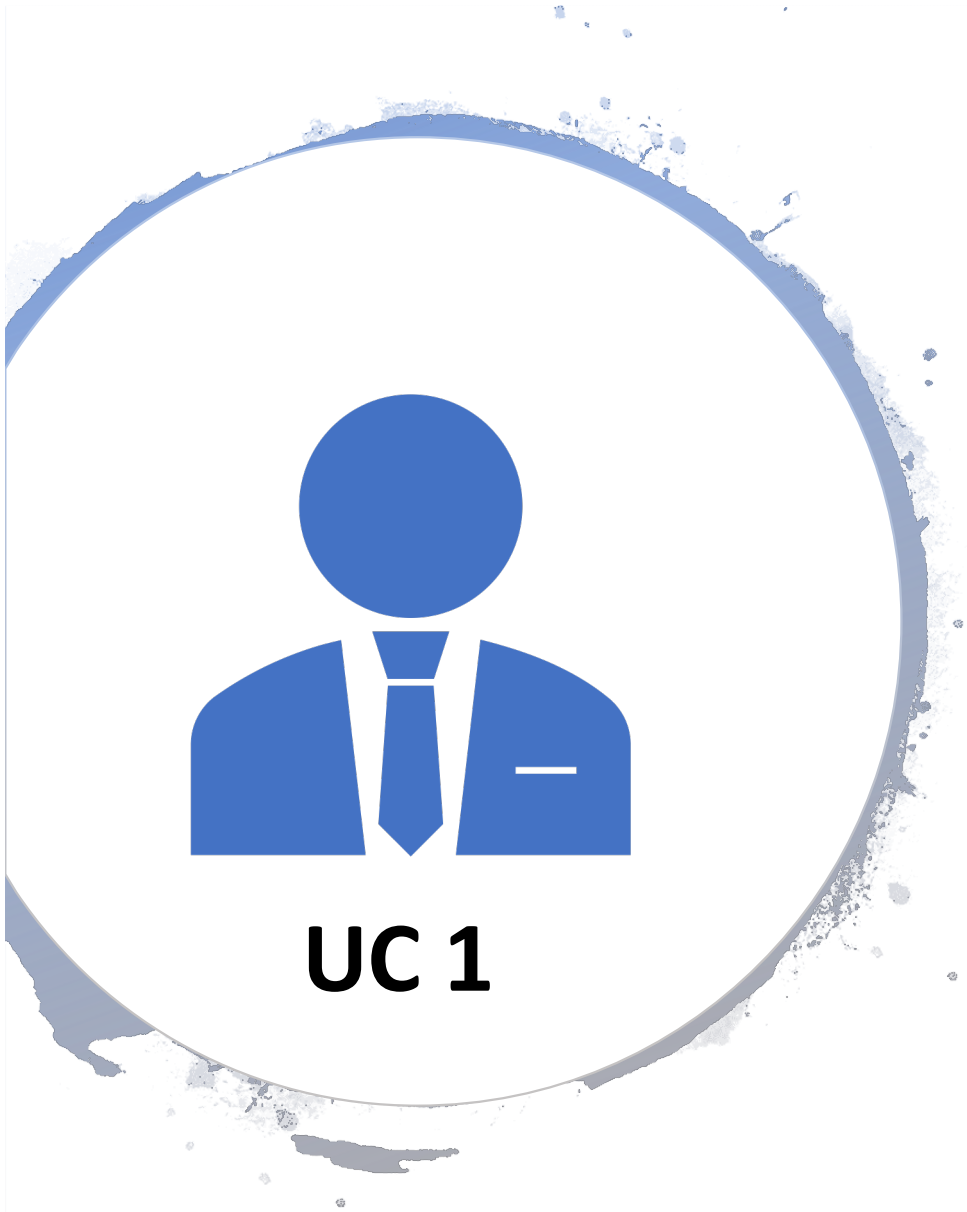
Java Core Concepts – Data Structures using Generics

Section 3: Hash Tables

Hash Tables

BridgeLabz
Employability Delivered

- **K, V Pair –** A Hash Table is a data structure that stores values which have keys associated with each of them i.e. simply a Key-Value Pair Data Structure.
- This is called Direct Addressing where in it uses the **one-to-one mapping** between the values and keys when storing in a table.



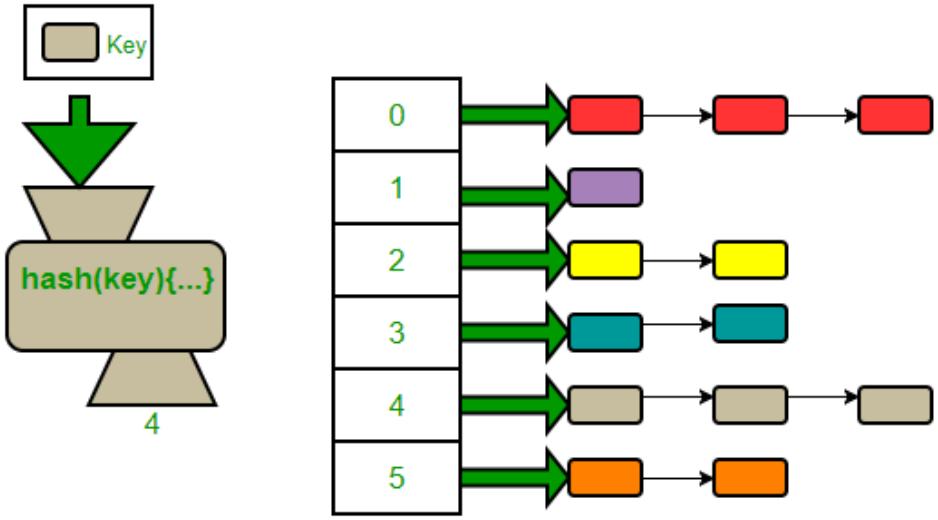
Ability to find frequency of words in a sentence like “To be or not to be”

- Use LinkedList to do the Hash Table Operation
- To do this we create MyMapNode with Key Value Pair and create LinkedList of MyMapNode

Hash Tables

BridgeLabz
Employability Delivered

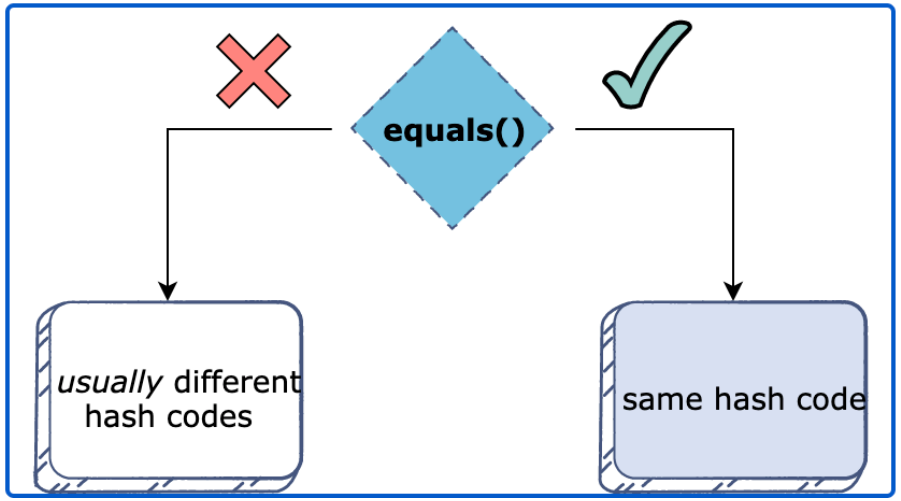
- There is a problem with Direct Addressing when there is a large number of key-value pairs.
- To avoid this we use Hash Function to find the index of the key-value pair in the hash table.
- **Components of Hash Table –**
  - **Hash Function:** Determines index of a key-value pair which is essentially a hash value**. h(k) = Hash Code of k % m**
  - **Array/List:** This holds all the key-value entries in the table. The size of the array/list should be set according to the amount of data expected.



What is Hash Code in Java???

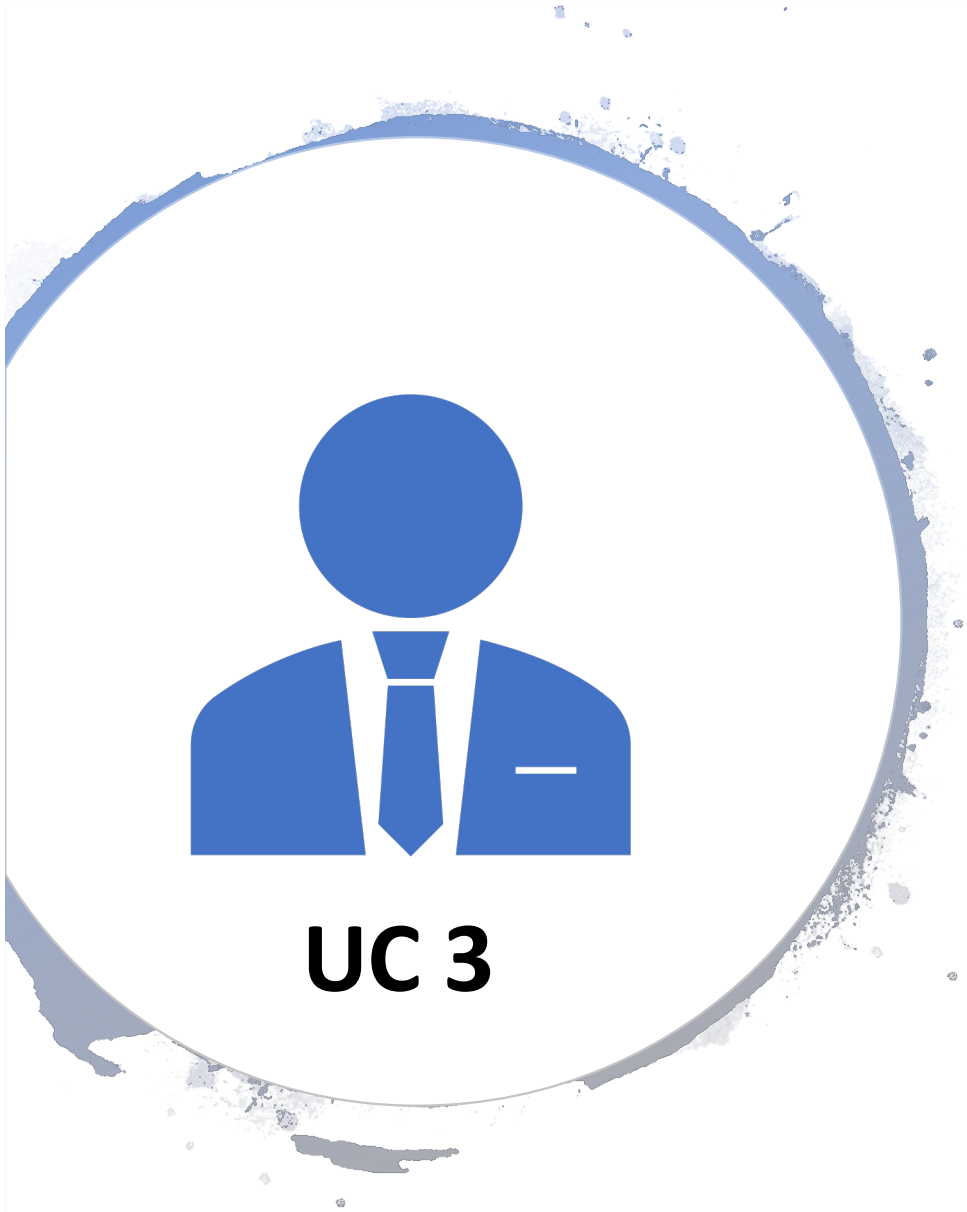
BridgeLabz
Employability Delivered

- A hash code is an integer value that is associated with each object in Java.
- It is obtained by calling the `hashCode()` method,
- Its main purpose is to facilitate hashing in hash tables.
- When `hashCode()` is called on two separate objects (which are equal according to the `equals()` method) it returns the same hash code value.
- However, if it is called on two unequal objects, it will typically return different integer values.



Ability to find frequency of words in a large paragraph phrase “Paranoids are not paranoid because they are paranoid but because they keep putting themselves deliberately into paranoid avoidable situations”

- Use hashcode to find index of the words in the para
- Create LinkedList for each index and store the words and its frequency
- Use LinkedList to do the Hash Table Operation
- To do this create MyMapNode with Key Value Pair and create LinkedList of MyMapNode



Remove avoidable word from the phrase “Paranoids are not paranoid because they are paranoid but because they keep putting themselves deliberately into paranoid avoidable situations”

- Use LinkedList to do the Hash Table Operation like here the removal of word avoidable
- To do this create MyMapNode with Key Value Pair and create LinkedList of MyMapNode

HashTable API

The functions we plan to keep in our hash map are

- **get(K key)** : returns the value corresponding to the key if the key is present in **HT** (**H**ast **T**able)
- **getSize()** : return the size of the HT
- **add()** : adds new valid key, value pair to the HT, if already present updates the value
- **remove()** : removes the key, value pair
- **isEmpty()** : returns true if size is zero

MyHashMap Code Snippet

BridgeLabz
Employability Delivered

```java
public class MyMapNode<K, V> implements INode<K>{
    K key;
    V value;
    MyMapNode<K, V> next;

    public MyMapNode(K key, V value) {...}

    public K getKey() { return key; }

    public void setKey(K key) { this.key = key; }

    public INode<K> getNext() { return next; }

    public void setNext(INode<K> next) {
        this.next = (MyMapNode<K, V>) next;
    }

    public V getValue() { return this.value; }

    public void setValue(V value) { this.value = value; }

    @Override
    public String toString() {...}
}
```

```java
public class MyHashMap<K, V> {

    MyLinkedList<K> myLinkedList;

    public MyHashMap() {
        this.myLinkedList = new MyLinkedList<>();
    }

    public V get(K key) {
        MyMapNode<K,V> myMapNode = (MyMapNode<K,V>) this.myLinkedList.search(key);
        return (myMapNode == null) ? null : myMapNode.getValue();
    }

    public void add(K key, V value) {
        MyMapNode<K,V> myMapNode = (MyMapNode<K,V>) this.myLinkedList.search(key);
        if (myMapNode == null) {
            myMapNode = new MyMapNode<>(key, value);
            this.myLinkedList.append(myMapNode);
        } else {
            myMapNode.setValue(value);
        }
    }

    @Override
    public String toString() { return "MyHashMapNodes{" + myLinkedList + '}'; }
}
```

MyLinkedHashMap Code Snippet

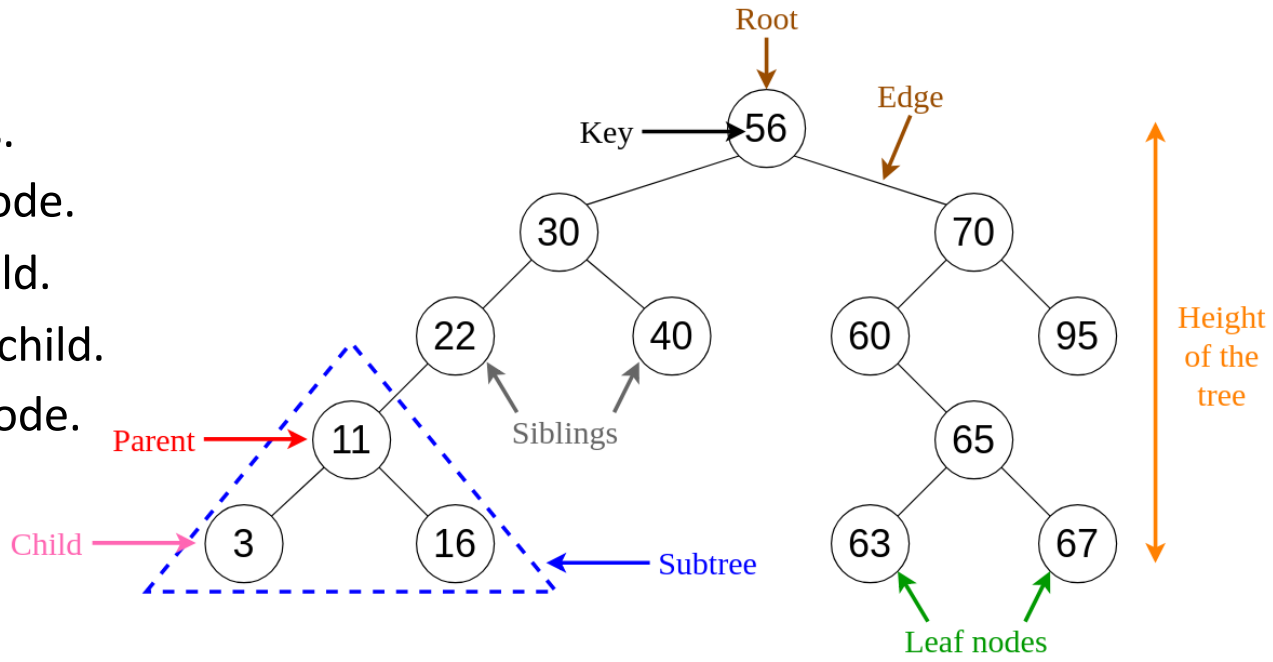

```java
public class MyLinkedHashMap<K, V> {
    private final int numBuckets;
    ArrayList<MyLinkedList<K>> myBucketArray;

    public MyLinkedHashMap() {
        this.numBuckets = 10;
        this.myBucketArray = new ArrayList<>(numBuckets);
        // Create empty LinkedLists
        for (int i = 0; i < numBuckets; i++)
            this.myBucketArray.add(null);
    }

    // This implements hash function to find index
    // for a key
    private int getBucketIndex(K key)
    {
        int hashCode = Math.abs(key.hashCode());
        int index = hashCode % numBuckets;
        return index;
    }

    public V get(K key) {
        int index = this.getBucketIndex(key);
        MyLinkedList<K> myLinkedList = this.myBucketArray.get(index);
        if(myLinkedList == null) return null;
        MyMapNode<K,V> myMapNode = (MyMapNode<K,V>) myLinkedList.search(key);
        return (myMapNode == null) ? null : myMapNode.getValue();
    }

    public void add(K key, V value) {
        int index = this.getBucketIndex(key);
        MyLinkedList<K> myLinkedList = this.myBucketArray.get(index);
        if(myLinkedList == null) {
            myLinkedList = new MyLinkedList<>();
            this.myBucketArray.set(index, myLinkedList);
        }
        MyMapNode<K,V> myMapNode = (MyMapNode<K,V>) myLinkedList.search(key);
        if (myMapNode == null) {
            myMapNode = new MyMapNode<>(key, value);
            myLinkedList.append(myMapNode);
        } else {
            myMapNode.setValue(value);
        }
    }

    @Override
    public String toString() {
        return "MyLinkedHashMap List{" + myBucketArray + '}';
    }
}
```

Section 4: Binary Tree

Trees and Binary Search Trees (BST)

BridgeLabz
Employability Delivered

- **Tree –** A tree is a **Hierarchical Nonlinear** structure where data is organized hierarchically and are linked together.
- **Binary Search Trees –** This is one such tree as the name suggests, is a binary tree where data is stored in a sorted order in the node itself or in the left or right sub tree.
- Each **Node** in a binary search tree comprises the following attributes.
  - **key –** The value stored in the node.
  - **left –** The pointer to the left child.
  - **right –** The pointer to the right child.
  - **p –** The pointer to the parent node.



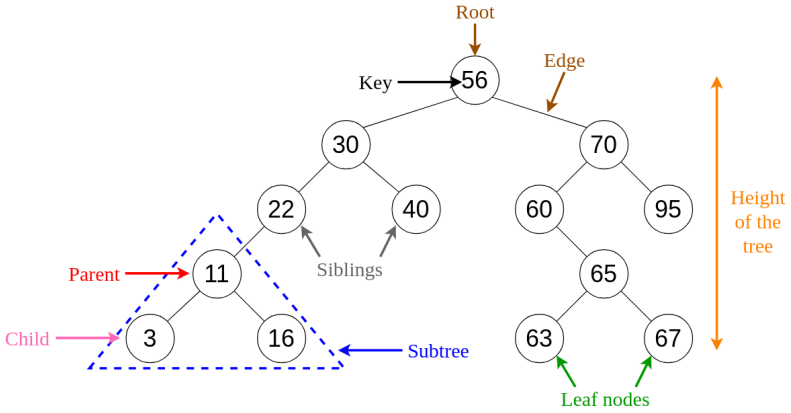
Ability to create a BST by adding 56 and then adding 30 & 70

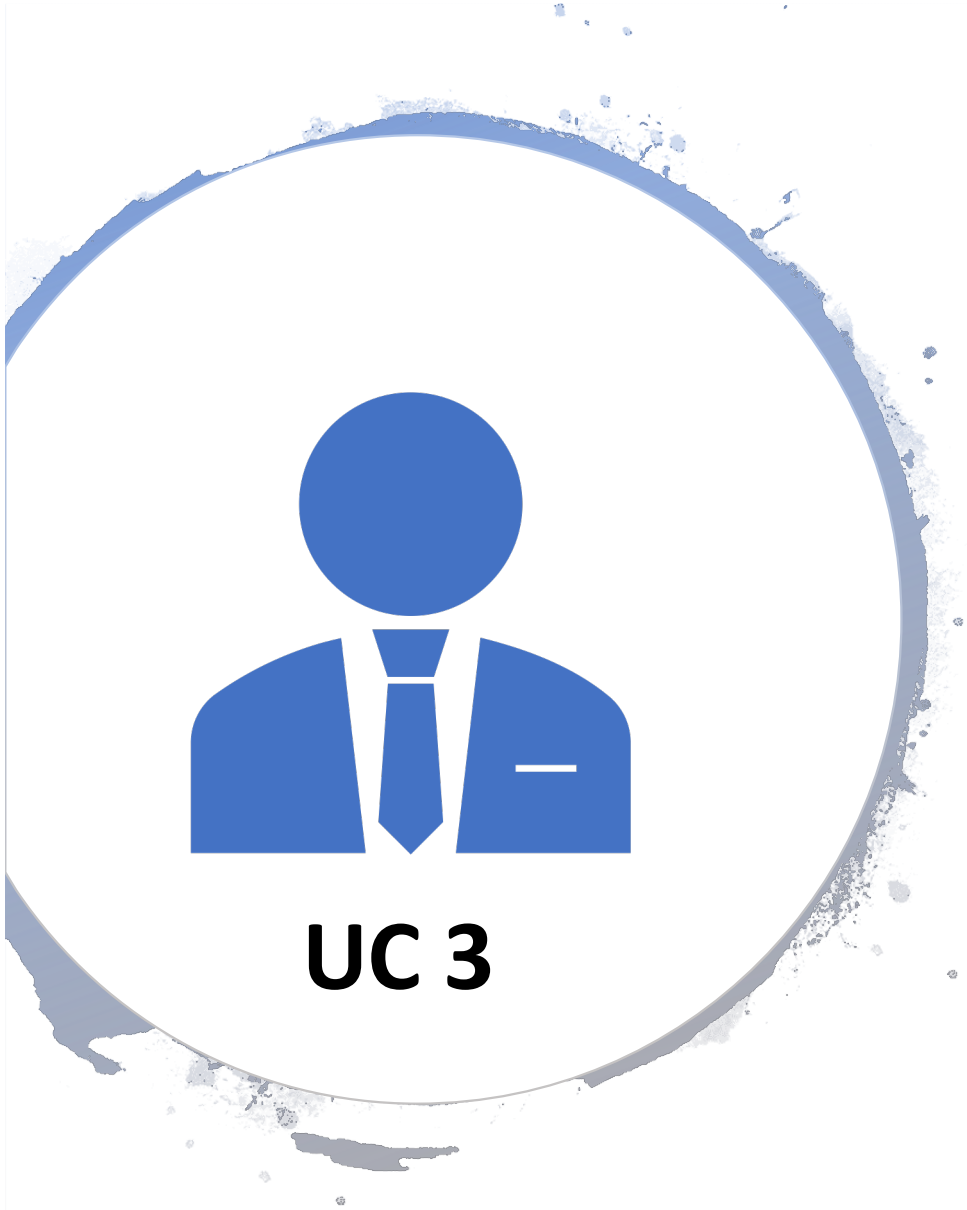
- Use INode to create My Binary Node
- Note the key has to extend comparable to compare and determine left or right node
- First add 56 as root node so 30 will be added to left and 70 to right

UC 2

Ability to create the binary tree shown in the figure

- Check if all are added with using size method in Binary Tree





Ability to search 63 in the Binary Tree

- Implement Search method and recursively search left or right nodes to find 63

MyBinaryTree Code Snippet


```java
public class MyBinaryNode<K extends Comparable<K>> {
    K key;
    MyBinaryNode<K> left;
    MyBinaryNode<K> right;

    public MyBinaryNode(K key) {
        this.key = key;
        this.left = null;
        this.right = null;
    }
}
```

```java
public class MyBinaryTreeTest {
    @Test
    public void given3NumbersWhenAddedToBSTShouldReturnSize3() {
        MyBinaryTree<Integer> myBinaryTree = new MyBinaryTree<>();
        myBinaryTree.add(56);
        myBinaryTree.add(30);
        myBinaryTree.add(70);
        int size = myBinaryTree.getSize();
        Assert.assertEquals( expected: 3, size);
    }
}
```

```java
public class MyBinaryTree<K extends Comparable<K>> {

    private MyBinaryNode<K> root;

    public void add(K key) {
        this.root = this.addRecursively(root, key);
    }

    private MyBinaryNode<K> addRecursively(MyBinaryNode<K> current, K key) {
        if (current == null)
            return new MyBinaryNode<>(key);
        int compareResult = key.compareTo(current.key);
        if (compareResult == 0) return current;
        if (compareResult < 0) {
            current.left = addRecursively(current.left, key);
        } else {
            current.right = addRecursively(current.right, key);
        }
        return current;
    }

    public int getSize() {
        return this.getSizeRecursive(root);
    }

    private int getSizeRecursive(MyBinaryNode<K> current) {
        return current == null ? 0 : 1 + this.getSizeRecursive(current.left)
                                       + this.getSizeRecursive(current.right);
    }
}
```

Java Collections

BridgeLabz
Employability Delivered

- The Collection in Java is a **framework** that provides **readymade** classes using interfaces to store and manipulate group of objects.
- Java Collections can achieve all the operations that you perform on a data such as **searching, sorting, insertion, manipulation, and deletion**.
- Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (**ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet**).Java **Generics**

Java Maps

BridgeLabz
Employability Delivered

- **Map –** Contains values on the basis of key, i.e. **key and value pair**. Each key and value pair is known as an **entry**. A Map contains unique **keys**. Map is like a **Dictionary**.
- A Map is useful if you have to search, update or delete elements on the basis of a key.
- There are two interfaces for implementing Map in java: **Map** and **SortedMap**, and three classes: **HashMap**, **LinkedHashMap**, and **TreeMap**.
  - HashMap – does not maintain any order,
  - LinkedHashMap – It inherits HashMap class. It maintains insertion order. and
  - TreeMap – It is the implementation of Map and SortedMap. It maintains ascending order...

References

BridgeLabz
Employability Delivered

- 8 Common Data Structures
- More about LinkedList
- Implement Singly Linked List
- What is hash code in java
- What is a hash table
- Linked HashMap Implementation
- Binary Tree Implementation
- https://www.educative.io/courses/ds-and-algorithms-in-python/g7OWWX38jM9

BridgeLabz

Employability Delivered

Thank You