CODING PROGRAMMES

1. Reverse a String:

```
public class ReverseString {
    public static String reverse(String str) {
        char[] chars = str.toCharArray();
        String reversed = "";
        for (int i = chars.length - 1; i >= 0; i--) {
            reversed += chars[i];
        }
        return reversed;
    }

    public static void main(String[] args) {
        String str = "Hello";
        System.out.println(reverse(str)); // Output: olleH
    }
}
```

2. Check if a Number is Prime

```
public class PrimeCheck {
    public static boolean isPrime(int num) {
        if (num <= 1) return false;
        for (int i = 2; i <= num / 2; i++) {
            if (num % i == 0) return false;
        }
        return true;
```

```
    }

    public static void main(String[] args) {
        int num = 29;
        System.out.println(isPrime(num)); // Output: true
    }
}
```

3. Find the Factorial of a Number

```
public class Factorial {
    public static int factorial(int num) {
        int result = 1;
        for (int i = 1; i <= num; i++) {
            result *= i;
        }
        return result;
    }

    public static void main(String[] args) {
        int num = 5;
        System.out.println(factorial(num)); // Output: 120
    }
}
```

4. Find the Greatest Common Divisor (GCD):

```
public class GCD {
    public static int gcd(int a, int b) {
```

```
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    public static void main(String[] args) {
        int a = 56, b = 98;
        System.out.println(gcd(a, b)); // Output: 14
    }
}
```

5. Bubble Sort:

```
public class BubbleSort {
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - 1 - i; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
```

```
            }
        }
    }
}

public static void main(String[] args) {
    int[] arr = {64, 34, 25, 12, 22, 11, 90};
    bubbleSort(arr);
    for (int i : arr) {
        System.out.print(i + " ");
    }
    // Output: 11 12 22 25 34 64 90
}
}
```

5. Fibonacci Series

```
public class Fibonacci {
    public static void main(String[] args) {
        int count = 10;
        printFibonacci(count);
    }

    public static void printFibonacci(int count) {
        int num1 = 0, num2 = 1;
```

```
        System.out.print("Fibonacci Series: " + num1 + " " + num2);

        for (int i = 2; i < count; i++) {
            int num3 = num1 + num2;
            System.out.print(" " + num3);
            num1 = num2;
            num2 = num3;
        }
    }
}
```

6. Find the Sum of Digits of a Number

```
public class SumOfDigits {
    public static void main(String[] args) {
        int number = 1234;
        int sum = sumOfDigits(number);
        System.out.println("Sum of digits of " + number + " is: " + sum);
    }

    public static int sumOfDigits(int num) {
        int sum = 0;
        while (num != 0) {
            sum += num % 10;
            num /= 10;
```

```
        }
        return sum;
    }
}
```

7. Check if a String is a Palindrome

```
public class PalindromeCheck {
    public static void main(String[] args) {
        String input = "madam";
        boolean isPalindrome = isPalindrome(input);
        System.out.println("Is \"" + input + "\" a palindrome? " + isPalindrome);
    }

    public static boolean isPalindrome(String str) {
        int start = 0;
        int end = str.length() - 1;
        while (start < end) {
            if (str.charAt(start) != str.charAt(end)) {
                return false;
            }
            start++;
            end--;
        }
        return true;
```

```
    }
}
```

8. Find the Second Largest Number in an Array

```
public class SecondLargest {
    public static void main(String[] args) {
        int[] numbers = {3, 5, 7, 2, 8};
        int secondLargest = findSecondLargest(numbers);
        System.out.println("Second largest number is: " + secondLargest);
    }

    public static int findSecondLargest(int[] arr) {
        int first = Integer.MIN_VALUE;
        int second = Integer.MIN_VALUE;
        for (int num : arr) {
            if (num > first) {
                second = first;
                first = num;
            } else if (num > second && num != first) {
                second = num;
            }
        }
        return second;
    }
```

```
}
```

9. Remove Duplicates from an Array

```
import java.util.Arrays;


public class RemoveDuplicates {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 2, 3, 4, 4, 5};
        int[] result = removeDuplicates(numbers);
        System.out.println("Array after removing duplicates: " + Arrays.toString(result));
    }


    public static int[] removeDuplicates(int[] arr) {
        int n = arr.length;
        if (n == 0 || n == 1) {
            return arr;
        }
        int[] temp = new int[n];
        int j = 0;
        for (int i = 0; i < n - 1; i++) {
            if (arr[i] != arr[i + 1]) {
                temp[j++] = arr[i];
            }
        }
```

```
        temp[j++] = arr[n - 1];
        int[] result = new int[j];
        for (int i = 0; i < j; i++) {
            result[i] = temp[i];
        }
        return result;
    }
}
```

10. Find the Length of the Longest Substring Without Repeating Characters

```
public class LongestSubstring {
    public static void main(String[] args) {
        String input = "abcabcbb";
        int length = lengthOfLongestSubstring(input);
        System.out.println("Length of the longest substring without repeating characters: " + length);
    }

    public static int lengthOfLongestSubstring(String s) {
        int n = s.length();
        int maxLength = 0;
        for (int i = 0; i < n; i++) {
            boolean[] visited = new boolean[256];
            for (int j = i; j < n; j++) {
```

```
            if (visited[s.charAt(j)]) {
                break;
            } else {
                maxLength = Math.max(maxLength, j - i + 1);
                visited[s.charAt(j)] = true;
            }
        }
    }
    return maxLength;
  }
}
```

11. Merge Two Sorted Arrays

```
import java.util.Arrays;

public class MergeSortedArrays {
  public static void main(String[] args) {
    int[] arr1 = {1, 3, 5, 7};
    int[] arr2 = {2, 4, 6, 8};
    int[] mergedArray = mergeArrays(arr1, arr2);
    System.out.println("Merged sorted array: " + Arrays.toString(mergedArray));
  }

  public static int[] mergeArrays(int[] arr1, int[] arr2) {
```

```
        int n1 = arr1.length;
        int n2 = arr2.length;
        int[] mergedArray = new int[n1 + n2];
        int i = 0, j = 0, k = 0;
        while (i < n1 && j < n2) {
            if (arr1[i] < arr2[j]) {
                mergedArray[k++] = arr1[i++];
            } else {
                mergedArray[k++] = arr2[j++];
            }
        }

        while (i < n1) {
            mergedArray[k++] = arr1[i++];
        }
        while (j < n2) {
            mergedArray[k++] = arr2[j++];
        }
        return mergedArray;
    }
}
```

12. Find the Missing Number in an Array

```
public class MissingNumber {
```

```
    public static void main(String[] args) {
        int[] numbers = {1, 2, 4, 5, 6};
        int missingNumber = findMissingNumber(numbers, 6);
        System.out.println("Missing number is: " + missingNumber);
    }

    public static int findMissingNumber(int[] arr, int n) {
        int totalSum = n * (n + 1) / 2;
        int arraySum = 0;
        for (int num : arr) {
            arraySum += num;
        }
        return totalSum - arraySum;
    }
}
```

13. Find the Maximum Subarray Sum

```
public class MaxSubarraySum {
    public static void main(String[] args) {
        int[] numbers = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
        int maxSum = maxSubarraySum(numbers);
        System.out.println("Maximum subarray sum is: " + maxSum);
    }
    public static int maxSubarraySum(int[] arr) {
```

```
        int maxSum = Integer.MIN_VALUE;
        int currentSum = 0;

        for (int num : arr) {
            currentSum += num;
            if (currentSum > maxSum) {
                maxSum = currentSum;
            }
            if (currentSum < 0) {
                currentSum = 0;
            }
        }
        return maxSum;
    }
}
```

14. Rotate an Array

```
import java.util.Arrays;

public class RotateArray {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5, 6, 7};
        int steps = 3;
        rotateArray(numbers, steps);
```

```
        System.out.println("Array after rotation: " + Arrays.toString(numbers));
    }

    public static void rotateArray(int[] arr, int steps) {
        int n = arr.length;
        steps = steps % n;
        reverseArray(arr, 0, n - 1);
        reverseArray(arr, 0, steps - 1);
        reverseArray(arr, steps, n - 1);
    }

    public static void reverseArray(int[] arr, int start, int end) {
        while (start < end) {
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;
            start++;
            end--;
        }
    }
}
```

15. Find the Longest Palindromic Substring

```
public class LongestPalindromicSubstring {
```

```
public static void main(String[] args) {
    String input = "babad";
    String longestPalindrome = longestPalindromicSubstring(input);
    System.out.println("Longest palindromic substring is: " + longestPalindrome);
}

public static String longestPalindromicSubstring(String s) {
    int n = s.length();
    if (n == 0) return "";
    String longest = s.substring(0, 1);

    for (int i = 0; i < n - 1; i++) {
        String p1 = expandAroundCenter(s, i, i);
        if (p1.length() > longest.length()) {
            longest = p1;
        }
        String p2 = expandAroundCenter(s, i, i + 1);
        if (p2.length() > longest.length()) {
            longest = p2;
        }
    }

    return longest;
```

```
    }

    public static String expandAroundCenter(String s, int c1, int c2) {
        int l = c1, r = c2;
        int n = s.length();
        while (l >= 0 && r < n && s.charAt(l) == s.charAt(r)) {
            l--;
            r++;
        }
        return s.substring(l + 1, r);
    }
}
```

16. Find the Intersection of Two Arrays

```
import java.util.Arrays;

public class ArrayIntersection {
    public static void main(String[] args) {
        int[] arr1 = {1, 2, 2, 1};
        int[] arr2 = {2, 2};
        int[] intersection = findIntersection(arr1, arr2);
        System.out.println("Intersection of arrays: " + Arrays.toString(intersection));
    }
```

```
    public static int[] findIntersection(int[] arr1, int[] arr2) {
        int[] temp = new int[Math.min(arr1.length, arr2.length)];
        int k = 0;
        for (int i = 0; i < arr1.length; i++) {
            for (int j = 0; j < arr2.length; j++) {
                if (arr1[i] == arr2[j]) {
                    temp[k++] = arr1[i];
                    arr2[j] = Integer.MIN_VALUE; // Mark as visited
                    break;
                }
            }
        }
        return Arrays.copyOf(temp, k);
    }
}
```

17. Find the Majority Element

```
public class MajorityElement {
    public static void main(String[] args) {
        int[] numbers = {3, 3, 4, 2, 4, 4, 2, 4, 4};
        int majorityElement = findMajorityElement(numbers);
        System.out.println("Majority element is: " + majorityElement);
    }
```

```
    public static int findMajorityElement(int[] arr) {
        int count = 0, candidate = -1;
        for (int num : arr) {
            if (count == 0) {
                candidate = num;
                count = 1;
            } else if (num == candidate) {
                count++;
            } else {
                count--;
            }
        }
        return candidate;
    }
}
```

18. Implement Binary Search

```
public class BinarySearch {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        int target = 5;
        int index = binarySearch(numbers, target);
        System.out.println("Element " + target + " found at index: " + index);
    }
```

```
    public static int binarySearch(int[] arr, int target) {
        int left = 0, right = arr.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (arr[mid] == target) {
                return mid;
            } else if (arr[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return -1; // Element not found
    }
}
```

19. Implement a Stack

```
public class Stack {
    private int[] arr;
    private int top;
    private int capacity;

    public Stack(int size) {
```

```
    arr = new int[size];
    capacity = size;
    top = -1;
}

public void push(int x) {
    if (isFull()) {
        System.out.println("Stack Overflow");
        return;
    }
    arr[++top] = x;
}

public int pop() {
    if (isEmpty()) {
        System.out.println("Stack Underflow");
        return -1;
    }
    return arr[top--];
}

public int peek() {
    if (!isEmpty()) {
```

```
            return arr[top];
        }
        return -1;
    }

    public boolean isEmpty() {
        return top == -1;
    }
    public boolean isFull() {
        return top == capacity - 1;
    }

    public static void main(String[] args) {
        Stack stack = new Stack(5);
        stack.push(1);
        stack.push(2);
        stack.push(3);
        System.out.println("Top element is: " + stack.peek());
        System.out.println("Popped element is: " + stack.pop());
        System.out.println("Top element is: " + stack.peek());
    }
}
```

20. Implement a Queue

```
public class Queue {
    private int[] arr;
    private int front;
    private int rear;
    private int capacity;
    private int count;

    public Queue(int size) {
        arr = new int[size];
        capacity = size;
        front = 0;
        rear = -1;
        count = 0;
    }

    public void enqueue(int x) {
        if (isFull()) {
            System.out.println("Queue Overflow");
            return;
        }
        rear = (rear + 1) % capacity;
        arr[rear] = x;
        count++;
```

```
}

public int dequeue() {
    if (isEmpty()) {
        System.out.println("Queue Underflow");
        return -1;
    }
    int item = arr[front];
    front = (front + 1) % capacity;
    count--;
    return item;
}

public int peek() {
    if (!isEmpty()) {
        return arr[front];
    }
    return -1;
}

public boolean isEmpty() {
    return count == 0;
}
```

```
    public boolean isFull() {
        return count == capacity;
    }
    public static void main(String[] args) {
        Queue queue = new Queue(5);
        queue.enqueue(1);
        queue.enqueue(2);
        queue.enqueue(3);
        System.out.println("Front element is: " + queue.peek());
        System.out.println("Dequeued element is: " + queue.dequeue());
        System.out.println("Front element is: " + queue.peek());
    }
}
```

21. Find the First Non-Repeated Character in a String

```
public class FirstNonRepeatedCharacter {
    public static void main(String[] args) {
        String input = "swiss";
        char result = firstNonRepeatedCharacter(input);
        System.out.println("First non-repeated character is: " + result);
    }
    public static char firstNonRepeatedCharacter(String str) {
        int[] count = new int[256];
```

```
        for (int i = 0; i < str.length(); i++) {
            count[str.charAt(i)]++;
        }
        for (int i = 0; i < str.length(); i++) {
            if (count[str.charAt(i)] == 1) {
                return str.charAt(i);
            }
        }
        return '\0';
    }
}
```

22. Implement a Linked List

```
public class LinkedList {
    Node head;
    static class Node {
        int data;
        Node next;

        Node(int d) {
            data = d;
            next = null;
        }
    }
```

```
public void append(int new_data) {
    Node new_node = new Node(new_data);
    if (head == null) {
        head = new_node;
        return;
    }
    Node last = head;
    while (last.next != null) {
        last = last.next;
    }
    last.next = new_node;
}

public void printList() {
    Node n = head;
    while (n != null) {
        System.out.print(n.data + " ");
        n = n.next;
    }
}

public static void main(String[] args) {
    LinkedList list = new LinkedList();
```

```
        list.append(1);
        list.append(2);
        list.append(3);
        list.printList();
    }
}
```

23. Implement a Binary Tree

```
public class BinaryTree {
    Node root;

    static class Node {
        int data;
        Node left, right;

        Node(int item) {
            data = item;
            left = right = null;
        }
    }

    BinaryTree() {
        root = null;
    }
```

```
    void printInOrder(Node node) {
        if (node == null) {
            return;
        }
        printInOrder(node.left);
        System.out.print(node.data + " ");
        printInOrder(node.right);
    }

    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        System.out.print("Inorder traversal: ");
        tree.printInOrder(tree.root);
    }
}
```

24. Implement Depth-First Search (DFS) for a Graph

```
import java.util.*;

public class Graph {
    private int V;
    private LinkedList<Integer> adj[];

    Graph(int v) {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i) {
            adj[i] = new LinkedList();
        }
    }

    void addEdge(int v, int w) {
        adj[v].add(w);
    }
    void DFSUtil(int v, boolean visited[]) {
        visited[v] = true;
        System.out.print(v + " ");

        Iterator<Integer> i = adj[v].listIterator();
        while (i.hasNext()) {
```

```
        int n = i.next();
        if (!visited[n]) {
            DFSUtil(n, visited);
        }
    }
}

void DFS(int v) {
    boolean visited[] = new boolean[V];
    DFSUtil(v, visited);
}

public static void main(String args[]) {
    Graph g = new Graph(4);

    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);
    System.out.println("Depth First Traversal starting from vertex 2:");
    g.DFS(2);
```

```
    }
}
```

25. Implement Breadth-First Search (BFS) for a Graph

```
import java.util.*;


public class GraphBFS {
    private int V;
    private LinkedList<Integer> adj[];
    GraphBFS(int v) {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i) {
            adj[i] = new LinkedList();
        }
    }
    void addEdge(int v, int w) {
        adj[v].add(w);
    }


    void BFS(int s) {
        boolean visited[] = new boolean[V];
        LinkedList<Integer> queue = new LinkedList<Integer>();
```

```
    visited[s] = true;
    queue.add(s);

    while (queue.size() != 0) {
        s = queue.poll();
        System.out.print(s + " ");
        Iterator<Integer> i = adj[s].listIterator();
        while (i.hasNext()) {
            int n = i.next();
            if (!visited[n]) {
                visited[n] = true;
                queue.add(n);
            }
        }
    }
}

public static void main(String args[]) {
    GraphBFS g = new GraphBFS(4);

    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
```

```
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);
        System.out.println("Breadth First Traversal starting from vertex 2:");
        g.BFS(2);
    }
}
```

26. Find the Maximum Depth of a Binary Tree

```
public class MaxDepthBinaryTree {
    static class Node {
        int data;
        Node left, right;
        Node(int item) {
            data = item;
            left = right = null;
        }
    }
    Node root;
    int maxDepth(Node node) {
        if (node == null) {
            return 0;
        } else {
            int leftDepth = maxDepth(node.left);
```

```
            int rightDepth = maxDepth(node.right);


            return Math.max(leftDepth, rightDepth) + 1;
        }
    }


    public static void main(String[] args) {
        MaxDepthBinaryTree tree = new MaxDepthBinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);


        System.out.println("Maximum depth of the binary tree is: " +
tree.maxDepth(tree.root));
    }
}
```

27. Find the Lowest Common Ancestor (LCA) of Two Nodes in a Binary Tree

```
public class LCABinaryTree {
    static class Node {
        int data;
        Node left, right;
```

```
    Node(int item) {
        data = item;
        left = right = null;
    }
}
Node root;
Node findLCA(Node node, int n1, int n2) {
    if (node == null) {
        return null;
    }
    if (node.data == n1 || node.data == n2) {
        return node;
    }

    Node leftLCA = findLCA(node.left, n1, n2);
    Node rightLCA = findLCA(node.right, n1, n2);

    if (leftLCA != null && rightLCA != null) {
        return node;
    }

    return (leftLCA != null) ? leftLCA : rightLCA;
```

```
    }

    public static void main(String[] args) {
        LCABinaryTree tree = new LCABinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);

        Node lca = tree.findLCA(tree.root, 4, 5);
        System.out.println("LCA of 4 and 5 is: " + lca.data);
    }
}
```

28. Implement a Min-Heap

```
public class MinHeap {
    private int[] heap;
    private int size;
    private int maxSize;

    public MinHeap(int maxSize) {
```

```
    this.maxSize = maxSize;
    this.size = 0;
    heap = new int[this.maxSize + 1];
    heap[0] = Integer.MIN_VALUE;
}

private int parent(int pos) {
    return pos / 2;
}

private int leftChild(int pos) {
    return 2 * pos;
}

private int rightChild(int pos) {
    return 2 * pos + 1;
}

private boolean isLeaf(int pos) {
    return pos > (size / 2) && pos <= size;
}

private void swap(int fpos, int spos) {
```

```
    int tmp;
    tmp = heap[fpos];
    heap[fpos] = heap[spos];
    heap[spos] = tmp;
}
private void minHeapify(int pos) {
    if (!isLeaf(pos)) {
        if (heap[pos] > heap[leftChild(pos)] || heap[pos] > heap[rightChild(pos)]) {
            if (heap[leftChild(pos)] < heap[rightChild(pos)]) {
                swap(pos, leftChild(pos));
                minHeapify(leftChild(pos));
            } else {
                swap(pos, rightChild(pos));
                minHeapify(rightChild(pos));
            }
        }
    }
}

public void insert(int element) {
    if (size >= maxSize) {
        return;
    }
```

```
    heap[++size] = element;
    int current = size;

    while (heap[current] < heap[parent(current)]) {
        swap(current, parent(current));
        current = parent(current);
    }
}

public int remove() {
    int popped = heap[1];
    heap[1] = heap[size--];
    minHeapify(1);
    return popped;
}

public static void main(String[] arg) {
    MinHeap minHeap = new MinHeap(15);
    minHeap.insert(5);
    minHeap.insert(3);
    minHeap.insert(17);
    minHeap.insert(10);
    minHeap.insert(84);
```

```
        minHeap.insert(19);
        minHeap.insert(6);
        minHeap.insert(22);
        minHeap.insert(9);

        System.out.println("The Min val is " + minHeap.remove());
    }
}
```

29. Implement a Trie (Prefix Tree)

```
public class Trie {
    static final int ALPHABET_SIZE = 26;

    static class TrieNode {
        TrieNode[] children = new TrieNode[ALPHABET_SIZE];
        boolean isEndOfWord;

        TrieNode() {
            isEndOfWord = false;
            for (int i = 0; i < ALPHABET_SIZE; i++) {
                children[i] = null;
            }
        }
    }
```

```
static TrieNode root;

static void insert(String key) {
    int level;
    int length = key.length();
    int index;

    TrieNode pCrawl = root;

    for (level = 0; level < length; level++) {
        index = key.charAt(level) - 'a';
        if (pCrawl.children[index] == null) {
            pCrawl.children[index] = new TrieNode();
        }
        pCrawl = pCrawl.children[index];
    }

    pCrawl.isEndOfWord = true;
}

static boolean search(String key) {
    int level;
```

```
    int length = key.length();
    int index;
    TrieNode pCrawl = root;

    for (level = 0; level < length; level++) {
        index = key.charAt(level) - 'a';
        if (pCrawl.children[index] == null) {
            return false;
        }
        pCrawl = pCrawl.children[index];
    }

    return (pCrawl != null && pCrawl.isEndOfWord);
}

public static void main(String args[]) {
    String keys[] = {"the", "a", "there", "answer", "any", "by", "bye", "their"};

    root = new TrieNode();

    for (int i = 0; i < keys.length; i++) {
        insert(keys[i]);
    }
```

```
        if (search("the")) {
            System.out.println("the --- Present in trie");
        } else {
            System.out.println("the --- Not present in trie");
        }

        if (search("these")) {
            System.out.println("these --- Present in trie");
        } else {
            System.out.println("these --- Not present in trie");
        }
    }
}
```

30. Implement a HashMap

```
import java.util.ArrayList;
import java.util.List;

class HashMap<K, V> {
    private class Entry<K, V> {
        K key;
        V value;
        Entry<K, V> next;
```

```
    Entry(K key, V value) {
        this.key = key;
        this.value = value;
    }
}

private final int SIZE = 16;
private List<Entry<K, V>> buckets;

public HashMap() {
    buckets = new ArrayList<>(SIZE);
    for (int i = 0; i < SIZE; i++) {
        buckets.add(null);
    }
}

private int getBucketIndex(K key) {
    return key.hashCode() % SIZE;
}

public void put(K key, V value) {
    int bucketIndex = getBucketIndex(key);
```

```
    Entry<K, V> existing = buckets.get(bucketIndex);
    if (existing == null) {
        buckets.set(bucketIndex, new Entry<>(key, value));
    } else {
        while (existing.next != null) {
            if (existing.key.equals(key)) {
                existing.value = value;
                return;
            }
            existing = existing.next;
        }
        if (existing.key.equals(key)) {
            existing.value = value;
        } else {
            existing.next = new Entry<>(key, value);
        }
    }
}

public V get(K key) {
    int bucketIndex = getBucketIndex(key);
    Entry<K, V> entry = buckets.get(bucketIndex);
    while (entry != null) {
```

31. Calculate age from dob and date should in proper date formate.

```
import java.time.LocalDate;

import java.time.Period;

import java.time.format.DateTimeFormatter;

import java.util.Scanner;


public class CalculateAge {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-
yyyy");



        System.out.print("Enter your date of birth (dd-MM-yyyy): ");

        String dobInput = scanner.nextLine();

        LocalDate dob = LocalDate.parse(dobInput, formatter);



        LocalDate currentDate = LocalDate.now();

        int age = calculateAge(dob, currentDate);



        System.out.println("Your age is: " + age + " years");

    }



    public static int calculateAge(LocalDate dob, LocalDate currentDate) {

        if ((dob != null) && (currentDate != null)) {
```

```
            return Period.between(dob, currentDate).getYears();
        } else {
            return 0;
        }
    }
}

import java.time.LocalDate;
import java.time.Period;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;

public class CalculateAge {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-
yyyy");

        System.out.print("Enter your date of birth (dd-MM-yyyy): ");
        String dobInput = scanner.nextLine();
        LocalDate dob = LocalDate.parse(dobInput, formatter);

        LocalDate currentDate = LocalDate.now();
        int age = calculateAge(dob, currentDate);

        System.out.println("Your age is: " + age + " years");
```

```
    }

    public static int calculateAge(LocalDate dob, LocalDate currentDate) {
        if ((dob != null) && (currentDate != null)) {
            return Period.between(dob, currentDate).getYears();
        } else {
            return 0;
        }
    }
}
```

32. Validation using Regex

1. Email Validation

```
import java.util.regex.*;

public class EmailValidation {
    public static void main(String[] args) {
        String email = "example@example.com";
        System.out.println("Is valid email: " + isValidEmail(email));
    }

    public static boolean isValidEmail(String email) {
        String emailRegex = "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,6}$";
        Pattern pattern = Pattern.compile(emailRegex);
        Matcher matcher = pattern.matcher(email);
        return matcher.matches();
    }
```

```
}
```

2. PAN Card Validation

```
import java.util.regex.*;


public class PANCardValidation {
    public static void main(String[] args) {
        String panCard = "ABCDE1234F";
        System.out.println("Is valid PAN card: " + isValidPANCard(panCard));
    }


    public static boolean isValidPANCard(String panCard) {
        String panCardRegex = "[A-Z]{5}[0-9]{4}[A-Z]{1}";
        Pattern pattern = Pattern.compile(panCardRegex);
        Matcher matcher = pattern.matcher(panCard);
        return matcher.matches();
    }
}
```

3. Pincode Validation

```
import java.util.regex.*;


public class PincodeValidation {
    public static void main(String[] args) {
        String pincode = "560001";
        System.out.println("Is valid pincode: " + isValidPincode(pincode));
    }
```

```
    public static boolean isValidPincode(String pincode) {
        String pincodeRegex = "^[1-9][0-9]{2}\\s{0,1}[0-9]{3}$";
        Pattern pattern = Pattern.compile(pincodeRegex);
        Matcher matcher = pattern.matcher(pincode);
        return matcher.matches();
    }
}
```

4. Mobile Number Validation

```
import java.util.regex.*;

public class MobileValidation {
    public static void main(String[] args) {
        String mobile = "9876543210";
        System.out.println("Is valid mobile number: " + isValidMobile(mobile));
    }

    public static boolean isValidMobile(String mobile) {
        String mobileRegex = "^[6-9][0-9]{9}$";
        Pattern pattern = Pattern.compile(mobileRegex);
        Matcher matcher = pattern.matcher(mobile);
        return matcher.matches();
    }
}
```