Q.1 :

Sol:

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

public class GenerateParentheses {
   public static List<String> generateParenthesis(int n) {
      List<String> result = new ArrayList<>();
      generateParenthesisHelper(n, 0, 0, "", result);
      return result;
   }

   private static void generateParenthesisHelper(int n, int open, int close, String current, List<String>
result) {
      // Base case: if the length of the current combination is 2n, add it to the result
      if (current.length() == 2 * n) {
         result.add(current);
         return;
      }

      // If we can add an open parenthesis, do so
      if (open < n) {
         generateParenthesisHelper(n, open + 1, close, current + "(", result);
      }

      // If we can add a close parenthesis without violating the well-formed condition, do so
      if (close < open) {
         generateParenthesisHelper(n, open, close + 1, current + ")", result);
      }
```

```java
    }

    public static void main(String[] args) {
        int n = 3; // You can change the value of n as needed
        List<String> combinations = generateParenthesis(n);

        System.out.println("All combinations of well-formed parentheses for n=" + n + ":");
        for (String combination : combinations) {
            System.out.println(combination);
        }
    }
}
```

Q.2:

Sol:

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

public class Combinations {
    public static List<List<Integer>> combine(int n, int k) {
        List<List<Integer>> result = new ArrayList<>();
        combineHelper(n, k, 1, new ArrayList<>(), result);
        return result;
    }

    private static void combineHelper(int n, int k, int start, List<Integer> current, List<List<Integer>> result)
{
        // Base case: if we have selected k elements, add the combination to the result
        if (current.size() == k) {
            result.add(new ArrayList<>(current));
```

```
            return;
        }

        // Try adding numbers to the current combination
        for (int i = start; i <= n; i++) {
            current.add(i);
            combineHelper(n, k, i + 1, current, result);
            current.remove(current.size() - 1);
        }
    }

    public static void main(String[] args) {
        int n = 4; // You can change the value of n as needed
        int k = 2; // You can change the value of k as needed

        List<List<Integer>> combinations = combine(n, k);

        System.out.println("All combinations of " + k + " numbers chosen from [1, " + n + "]:");
        for (List<Integer> combination : combinations) {
            System.out.println(combination);
        }
    }
}
```

Q.3:

Sol:

```
class TreeNode {
    int val;
```

```java
    TreeNode left;
    TreeNode right;

    TreeNode(int x) {
        val = x;
    }
}

public class SumRootToLeafNumbers {
    public int sumNumbers(TreeNode root) {
        return sumNumbersHelper(root, 0);
    }

    private int sumNumbersHelper(TreeNode node, int currentSum) {
        if (node == null) {
            return 0;
        }

        // Calculate the current sum by adding the current node's value
        currentSum = currentSum * 10 + node.val;

        // If the current node is a leaf, return the current sum
        if (node.left == null && node.right == null) {
            return currentSum;
        }

        // Recursively calculate the sum for left and right subtrees
        int leftSum = sumNumbersHelper(node.left, currentSum);
        int rightSum = sumNumbersHelper(node.right, currentSum);
```

```
        // Return the sum of both subtrees
        return leftSum + rightSum;
    }

    public static void main(String[] args) {
        // Example usage:
        // Construct a binary tree
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);

        // Call the sumNumbers function
        SumRootToLeafNumbers solution = new SumRootToLeafNumbers();
        int totalSum = solution.sumNumbers(root);

        System.out.println("Total sum of root-to-leaf numbers: " + totalSum);
    }
}
```

Q.4:

Sol:

```
import java.util.Stack;

public class EvaluateRPN {
    public int evalRPN(String[] tokens) {
        Stack<Integer> stack = new Stack<>();

        for (String token : tokens) {
            if (isOperator(token)) {
```

```
            // Pop the top two numbers from the stack
            int operand2 = stack.pop();
            int operand1 = stack.pop();

            // Perform the operation and push the result back onto the stack
            int result = performOperation(operand1, operand2, token);
            stack.push(result);
        } else {
            // If it's a number, push it onto the stack
            stack.push(Integer.parseInt(token));
        }
    }

    // The final result will be on the top of the stack
    return stack.pop();
}

private boolean isOperator(String token) {
    return token.equals("+") || token.equals("-") || token.equals("*") || token.equals("/");
}

private int performOperation(int operand1, int operand2, String operator) {
    switch (operator) {
        case "+":
            return operand1 + operand2;
        case "-":
            return operand1 - operand2;
        case "*":
            return operand1 * operand2;
```

```java
            case "/":
                return operand1 / operand2;
            default:
                throw new IllegalArgumentException("Invalid operator: " + operator);
        }
    }

    public static void main(String[] args) {
        // Example usage:
        String[] tokens = {"2", "1", "+", "3", "*"};
        EvaluateRPN solution = new EvaluateRPN();
        int result = solution.evalRPN(tokens);

        System.out.println("Result of the expression: " + result);
    }
}
```

Q.5:

Sol:

```java
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class WordBreak {
    public boolean wordBreak(String s, List<String> wordDict) {
        Set<String> wordSet = new HashSet<>(wordDict);
        int n = s.length();

        // dp[i] is true if the substring s[0...i-1] can be segmented into words in the dictionary
```

```java
        boolean[] dp = new boolean[n + 1];
        dp[0] = true; // An empty string can always be segmented

        for (int i = 1; i <= n; i++) {
            for (int j = 0; j < i; j++) {
                if (dp[j] && wordSet.contains(s.substring(j, i))) {
                    dp[i] = true;
                    break;
                }
            }
        }

        return dp[n];
    }

    public static void main(String[] args) {
        WordBreak solution = new WordBreak();
        String s = "leetcode";
        List<String> wordDict = List.of("leet", "code");
        boolean canBreak = solution.wordBreak(s, wordDict);

        System.out.println("Can break into words: " + canBreak);
    }
}
```

Q.6:

Sol:

```
grep -P '^\s*(\(\d{3}\) \d{3}-\d{4}|\d{3}-\d{3}-\d{4})\s*$' file.txt
```

Q.7:

Sol: import java.util.ArrayList;

```
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

class TreeNode {
   int val;
   TreeNode left, right;

   public TreeNode(int val) {
      this.val = val;
   }
}

public class RightSideView {
   public List<Integer> rightSideView(TreeNode root) {
      List<Integer> result = new ArrayList<>();
      if (root == null) {
         return result;
      }

      Queue<TreeNode> queue = new LinkedList<>();
      queue.offer(root);

      while (!queue.isEmpty()) {
         int levelSize = queue.size();

         for (int i = 0; i < levelSize; i++) {
            TreeNode current = queue.poll();
```

```java
            // The last node encountered at each level is added to the result
            if (i == levelSize - 1) {
                result.add(current.val);
            }

            if (current.left != null) {
                queue.offer(current.left);
            }

            if (current.right != null) {
                queue.offer(current.right);
            }
        }
    }

    return result;
}

public static void main(String[] args) {
    // Example usage:
    TreeNode root = new TreeNode(1);
    root.left = new TreeNode(2);
    root.right = new TreeNode(3);
    root.left.right = new TreeNode(5);
    root.right.right = new TreeNode(4);

    RightSideView solution = new RightSideView();
    List<Integer> result = solution.rightSideView(root);
```

```
        System.out.println("Right side view of the binary tree: " + result);
    }
}
```

Q.8:

Sol:

```
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
    }
}

public class ReverseLinkedList {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        ListNode current = head;

        while (current != null) {
            ListNode nextNode = current.next;
            current.next = prev;
            prev = current;
            current = nextNode;
        }

        return prev;
    }
```

```java
// Helper method to print the linked list
private void printList(ListNode head) {
    ListNode current = head;
    while (current != null) {
        System.out.print(current.val + " ");
        current = current.next;
    }
    System.out.println();
}

public static void main(String[] args) {
    // Example usage:
    ListNode head = new ListNode(1);
    head.next = new ListNode(2);
    head.next.next = new ListNode(3);
    head.next.next.next = new ListNode(4);
    head.next.next.next.next = new ListNode(5);

    ReverseLinkedList solution = new ReverseLinkedList();
    System.out.println("Original Linked List:");
    solution.printList(head);

    // Reverse the linked list
    ListNode reversedHead = solution.reverseList(head);

    System.out.println("Reversed Linked List:");
    solution.printList(reversedHead);
}
```

```
}
```

Q.9:

Sol:

```
public class PowerOfTwo {
    public boolean isPowerOfTwo(int n) {
        // Check if n is positive and has only one bit set to 1
        return n > 0 && (n & (n - 1)) == 0;
    }

    public static void main(String[] args) {
        // Example usage:
        PowerOfTwo solution = new PowerOfTwo();

        int num1 = 16;
        System.out.println(num1 + " is a power of two: " + solution.isPowerOfTwo(num1));

        int num2 = 5;
        System.out.println(num2 + " is a power of two: " + solution.isPowerOfTwo(num2));
    }
}
```

Q.10:

Sol:

```
public class CountDigitOne {
    public int countDigitOne(int n) {
        int count = 0;

        for (int i = 1; i <= n; i++) {
```

```java
            count += countOnesInNumber(i);
        }

        return count;
    }

    private int countOnesInNumber(int num) {
        int count = 0;

        while (num > 0) {
            if (num % 10 == 1) {
                count++;
            }
            num /= 10;
        }

        return count;
    }

    public static void main(String[] args) {
        // Example usage:
        CountDigitOne solution = new CountDigitOne();
        int n = 13;
        System.out.println("Total number of digit 1 from 1 to " + n + ": " + solution.countDigitOne(n));
    }
}
```

Q.11:

Sol: import java.util.ArrayList;

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

```
class TreeNode {
   int val;
   TreeNode left, right;

   TreeNode(int x) {
      val = x;
   }
}

public class BinaryTreePaths {
   public List<String> binaryTreePaths(TreeNode root) {
      List<String> paths = new ArrayList<>();
      if (root != null) {
         binaryTreePathsHelper(root, "", paths);
      }
      return paths;
   }

   private void binaryTreePathsHelper(TreeNode node, String currentPath, List<String> paths) {
      // Append the current node's value to the current path
      currentPath += node.val;

      // If it's a leaf node, add the path to the result
      if (node.left == null && node.right == null) {
         paths.add(currentPath);
         return;
      }
```

```
        // If it's not a leaf node, continue the path with "->"
        currentPath += "->";

        // Recursively traverse the left and right subtrees
        if (node.left != null) {
            binaryTreePathsHelper(node.left, currentPath, paths);
        }

        if (node.right != null) {
            binaryTreePathsHelper(node.right, currentPath, paths);
        }
    }

    public static void main(String[] args) {
        // Example usage:
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.right = new TreeNode(5);

        BinaryTreePaths solution = new BinaryTreePaths();
        List<String> paths = solution.binaryTreePaths(root);

        System.out.println("Root-to-leaf paths:");
        for (String path : paths) {
            System.out.println(path);
        }
    }
}
```

Q.12:

Sol:

```java
public class AddDigits {
    public int addDigits(int num) {
        if (num == 0) {
            return 0;
        } else {
            return 1 + (num - 1) % 9;
        }
    }

    public static void main(String[] args) {
        // Example usage:
        AddDigits solution = new AddDigits();
        int num = 38;
        int result = solution.addDigits(num);

        System.out.println("Digital root of " + num + ": " + result);
    }
}
```