FULL STACK DEVELOPMENT – WORKSHEET B

Ans 1 – Compilation error on many levels.

Ans 2-

```
    class Vowel {
  public static void main(String args[]) {
    int count =0;       // initialising a variable count that will count the number of vowels in the string
    String str = new String("GFHYB");
  /* Using for loop to traverse through the String */
    for(int i=0; i<str.length(); i++) {
/*if the character at index 'i' in the String will have any of vowels, the count variable will increase by
1 every time a vowel comes during traversing of string*/
     if(str.charAt(i)== 'a' || str.charAt(i)== 'A' ||
     str.charAt(i)== 'e' || str.charAt(i)== 'E' ||
     str.charAt(i)== 'i' || str.charAt(i)== 'I' ||
     str.charAt(i)== 'o' || str.charAt(i)== 'O' ||
     str.charAt(i)== 'u' || str.charAt(i)== 'U'  ){
      count++;
      }
    }
//if count's value is greater than equals to 1 it we will print true
//else we will print false
 if(count>=1){System.out.println("True");}
    else{System.out.println("False");}
   }
 }
```

Ans 3 -

```
import java.util.*;
public class Practice{
   // Function to remove duplicates from an ArrayList
   public static  ArrayList<Integer> removeDuplicates(ArrayList<Integer> list)
   {
      // Create a new ArrayList
      ArrayList<Integer> newList = new ArrayList<Integer>();


      // Traverse through the first list
      for (int i : list) {


         // If this element is not present in newList
         // then add it
         if (!newList.contains(i)) {
            newList.add(i);
         }
      }


      // return the new list
      return newList;
   }
   public static void main(String args[])
   {
      // Get the ArrayList with duplicate values using Array.asList function to add them as list
      ArrayList<Integer>list = new ArrayList<>(Arrays.asList(1, 10, 1, 2, 2, 3, 3, 10, 3, 4, 5, 5));


      // Print the Arraylist
```

```
        System.out.println("ArrayList with duplicates: "
                + list);



        // Remove duplicates
        ArrayList<Integer>newList = removeDuplicates(list);



        // Print the ArrayList with duplicates removed
        System.out.println("ArrayList with duplicates removed: "
                + newList);
    }
}
```

Ans 4 -

```
class LinkedList {
   Node head; // head of list


   /* Linked list Node*/
   class Node {
      int data;
      Node next;
      Node(int d)
      {
         data = d;
         next = null;
      }
   }


   /* Function to get Union of 2 Linked Lists */
   void getUnion(Node head1, Node head2)
   {
      Node t1 = head1, t2 = head2;


      // insert all elements of list1 in the result
      while (t1 != null) {
         push(t1.data);
         t1 = t1.next;
      }


      // insert those elements of list2 that are not present
      while (t2 != null) {
        if (!isPresent(head, t2.data))
            push(t2.data);
        t2 = t2.next;
    }
}


void getIntersection(Node head1, Node head2)
{
    Node result = null;
    Node t1 = head1;


    // Traverse list1 and search each
    // element of it in list2.
    // If the element is present in
    // list 2, then insert the
    // element to result
    while (t1 != null) {
        if (isPresent(head2, t1.data))
            push(t1.data);
        t1 = t1.next;
    }
}


/* Utility function to print list */
void printList()
{
    Node temp = head;
    while (temp != null) {
        System.out.print(temp.data + " ");
        temp = temp.next;
    }
    System.out.println();
}

/* Inserts a node at start of linked list */
void push(int new_data)
{
    /* 1 & 2: Allocate the Node &
            Put in the data*/
    Node new_node = new Node(new_data);

    /* 3. Make next of new Node as head */
    new_node.next = head;

    /* 4. Move the head to point to new Node */
    head = new_node;
}

/* A utility function that returns true
if data is present in linked list
else return false */
boolean isPresent(Node head, int data)
{
    Node t = head;
    while (t != null) {
        if (t.data == data)
            return true;
        t = t.next;
    }
    return false;
}
public static void main(String args[])
{
    LinkedList llist1 = new LinkedList();
    LinkedList llist2 = new LinkedList();
    LinkedList unin = new LinkedList();
    LinkedList intersecn = new LinkedList();


    /*create a linked lists 10->15->4->20 */
    llist1.push(20);
    llist1.push(4);
    llist1.push(15);
    llist1.push(10);


    /*create a linked lists 8->4->2->10 */
    llist2.push(10);
    llist2.push(2);
    llist2.push(4);
    llist2.push(8);


    intersecn.getIntersection(llist1.head, llist2.head);
    unin.getUnion(llist1.head, llist2.head);


    System.out.println("First List is");
    llist1.printList();
```

```
        System.out.println("Second List is");
        llist2.printList();


        System.out.println("Intersection List is");
        intersecn.printList();


        System.out.println("Union List is");
        unin.printList();
    }
}
```

Ans 5-

```
public class Practice{


    // function to calculate the sum of the middle row of a matrix
    public static int sumOfMiddleRow(int [][] matrix, int n, int m){
      int totalSum =0; //variable to store the total sum value

      // Iterating over the middle column and picking the middle value
      for(int col = 0; col<m; col++){
          totalSum += matrix[n/2][col];
      }
      return totalSum;
    }

    // function to calculate the sum of the middle column of a matrix
    public static int sumOfMiddleColumn(int [][] matrix, int n, int m){
      int totalSum =0; //variable to store the total sum value

      // Iterating over all rows and picking the middle value
      for(int row = 0; row<n; row++){
          totalSum += matrix[row][m/2];
      }
      return totalSum;
    }

    public static void main(String[] args) {
      int n= 3; // number of rows
      int m = 3; // number of columns
      // Input matrix
      int [][]matrix = {{1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}};

    System.out.println("Sum of the middle row: " + sumOfMiddleRow(matrix,n,m));
    System.out.println("Sum of the middle column: "+ sumOfMiddleColumn(matrix,n,m));
  }
}
```

Ans 6-

```
         import java.util.*;
     // link list node
       class Node {
    int key;
    Node next;


    public Node(int key) {
        this.key = key;
        next = null;
    }
}


public class Main {
    // return a newnode
    public static Node newNode(int key) {
        return new Node(key);
    }




    public static void main(String[] args) {


        // Link List a: 2->8->15->30
        Node a = new Node(2);
        a.next = new Node(8);
        a.next.next = new Node(15);
        a.next.next.next = new Node(30);


        // Link list b: 5->7->20
    Node b = new Node(5);

    b.next = new Node(7);

    b.next.next = new Node(20);


    //create a new Arraylist to add nodes

    List<Integer> v = new ArrayList<>();

    //add values of Linked List a to arrayList v

    while (a != null) {

      v.add(a.key);

      a = a.next;

    }

    ////add values of Linked List a to arrayList v

    while (b != null) {

      v.add(b.key);

      b = b.next;

    }

    //sort the updated Arraylist v

    Collections.sort(v);

    Node result = new Node(-1);

    Node temp = result;

    for (int i = 0; i < v.size(); i++) {

      result.next = new Node(v.get(i));

      result = result.next;

    }

     temp = temp.next;

    System.out.print("Resultant Merge Linked List is : ");

    while (temp != null) {

      System.out.print(temp.key + " ");

      temp = temp.next;

    }

  }
```

```
}
```

Ans 7-

```
        // Java program to print Bottom View of Binary Tree
import java.io.*;
import java.lang.*;
import java.util.*;


class Practice{


// Tree node class
static class Node
{
  // Data of the node
  int data;


  // Horizontal distance of the node
  int hd;


  // Left and right references
  Node left, right;


  // Constructor of tree node
  public Node(int key)
  {
    data = key;
    hd = Integer.MAX_VALUE;
    left = right = null;
  }
}
```

```
static void printBottomViewUtil(Node root, int curr, int hd,TreeMap<Integer, int[]> m)
{
    // Base case if root is null so binary tree is empty
    if (root == null)
        return;


    // If node for a particular
    // horizontal distance is not
    // present, add to the map.
    if (!m.containsKey(hd))
    {
        m.put(hd, new int[]{ root.data, curr });
    }

    // Compare height for already present node at similar horizontal distance
    else
    {  int[] p = m.get(hd);
        if (p[1] <= curr)
        {
            p[1] = curr;
            p[0] = root.data;
        }
        m.put(hd, p);
    }


    // Recur for left subtree
    printBottomViewUtil(root.left, curr + 1,
                        hd - 1, m);
    // Recur for right subtree
    printBottomViewUtil(root.right, curr + 1,
                        hd + 1, m);
}


static void printBottomView(Node root)
{


    // Map to store Horizontal Distance,
    // Height and Data.
    TreeMap<Integer, int[]> m = new TreeMap<>();


    printBottomViewUtil(root, 0, 0, m);


    // Prints the values stored by printBottomViewUtil()
    for(int val[] : m.values())
    {
        System.out.print(val[0] + " ");
    }
}


public static void main(String[] args)
{  //input the binary tree
    Node root = new Node(23);
    root.left = new Node(11);
    root.right = new Node(25);
```

```
   root.left.left = new Node(28);

   root.left.right = new Node(6);

   root.right.left = new Node(7);

   root.right.right = new Node(28);

   root.left.right.left = new Node(13);

   root.left.right.right = new Node(17);



   System.out.println("Bottom view of the given binary tree:");

    printBottomView(root);

}

}
```

Ans 8 -

```
     // A class to store a binary tree node
class Node
{
   int data;
   Node left = null, right = null;

   Node(int data) {
      this.data = data;
   }
}

class Main
{
   // Function to perform preorder traversal on a given binary tree
   public static void preorder(Node root)
   {
      if (root == null) {
         return;
      }

      System.out.print(root.data + " ");
      preorder(root.left);
      preorder(root.right);
   }

   // Utility function to swap left subtree with right subtree
   public static void swap(Node root)
   {
      if (root == null) {
         return;
```

```
    }

    Node temp = root.left;
    root.left = root.right;
    root.right = temp;
}

// Function to convert a given binary tree into its mirror
public static void convertToMirror(Node root)
{
    // base case: if the tree is empty
    if (root == null) {
        return;
    }

    // convert left subtree
    convertToMirror(root.left);

    // convert right subtree
    convertToMirror(root.right);

    // swap left subtree with right subtree
    swap(root);
}

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

```
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);

        convertToMirror(root);
        preorder(root);
    }
}
```

Ans 9-

```
 public class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode() {}
    TreeNode(int val) { this.val = val; }
    TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
 }


class Solution {
   //introducing method isSameTree with p as first binary tree's node and q as second
  public static boolean isSameTree(TreeNode p, TreeNode q) {
   //if p & q both are pointing towards null hence binary tree is empty or fully traversed, hence
equal.
     if(p==null && q==null) return true;

 /*if p is pointing towards another node but q is pointing towards
    null or vice versa we return false as they will not be considered as same/equal binary trees*/
     if(p!=null && q==null || p==null && q!=null) return false;

   /*if value in p nodes are not equal to values in q nodes we return false
     as they will not be considered as same/equal binary trees*/
     if(p.val!=q.val) return false;


   /*now we will use recursion for traversing through all the left nodes and right nodes of both trees
```

```
    also we have used && operator as it will only be considered same if both left nodes and right
nodes
    of the tree have same structure and values */
    return isSameTree(p.left,q.left) && isSameTree(p.right,q.right);
  }
  public static void main(String[] args) {
   //initialize Tree p
   TreeNode p = new TreeNode(1);
   p.left = new TreeNode(2);
   p.left.right = new TreeNode(3);
   p.left.left = new TreeNode(4);
   p.right = new TreeNode(5);
   p.right.left = new TreeNode(6);
   p.right.right = new TreeNode(7);


   //Initialize Tree q
   TreeNode q = new TreeNode(1);
   q.left = new TreeNode(2);
   q.left.right = new TreeNode(3);
   q.left.left = new TreeNode(4);
   q.right = new TreeNode(5);
   q.right.left = new TreeNode(6);
   q.right.right = new TreeNode(7);


   //call boolean funtion isSameTree
   if (isSameTree(p, q)) {
     System.out.println("Both Trees are Identical");
   }
   else{System.out.println("Both Trees are not Identical");}
  }
}
```

Ans 10-

```
public class Practice {
   //create a boolean function that will return true if the number is a power of true
   public static boolean powerOfTwo(int n)
      {   //divide n by 2 until it gives remainder 0
         while(n%2==0)
             {n=n/2;}
         //after dividing n by 2 until it is giving remainde 0 check if n becomes 1
         //if n becomes 1 it means n fully divisible by two and is a power of 2, we return true
          if(n==1)
           {return true; }
         else
           {return false;}
      }
      public static void main(String[] args) {
         //initialise n and print function
         int n = 1023;
        System.out.println( powerOfTwo(n));
      }
                }
```