CSW ASSIGNMENT 2

PART B

Q1.

```
import java.util.ArrayList;
import java.util.Scanner;

public class ArrayListOperations {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<Integer>();
        Scanner scanner = new Scanner(System.in);

        // Add some elements to the list
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40);
        list.add(50);

        // Display the list
        System.out.println("List: " + list);

        // Ask user to enter a number and search for it
        System.out.print("Enter a number to search: ");
        int num = scanner.nextInt();
        if (list.contains(num)) {
            System.out.println(num + " is present in the list.");
        } else {
            System.out.println(num + " is not present in the list.");
        }
```

```
        // Remove an element from an asked position
        System.out.print("Enter a position to remove an element: ");
        int pos = scanner.nextInt();
        if (pos >= 0 && pos < list.size()) {
            list.remove(pos);
            System.out.println("Element removed from position " + pos);
            System.out.println("Updated List: " + list);
        } else {
            System.out.println("Invalid position. Element cannot be removed.");
        }

        // Check if the ArrayList is empty or not
        if (list.isEmpty()) {
            System.out.println("The list is empty.");
        } else {
            System.out.println("The list is not empty.");
        }

        scanner.close();
    }
}
```

Q2.

```
import java.util.LinkedList;
import java.util.Scanner;

class Student {
    String name;
    int age;
    double mark;
```

```
    public Student(String name, int age, double mark) {
        this.name = name;
        this.age = age;
        this.mark = mark;
    }

    @Override
    public String toString() {
        return "Name: " + name + ", Age: " + age + ", Mark: " + mark;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj instanceof Student) {
            Student other = (Student) obj;
            if (this.name.equals(other.name) && this.age == other.age && this.mark == other.mark) {
                return true;
            }
        }
        return false;
    }
}

public class LinkedListOperations {

    public static void main(String[] args) {

        // Create an empty LinkedList of Student type
        LinkedList<Student> list = new LinkedList<>();

        // Add some Student objects to the list
```

```
list.add(new Student("John", 20, 80.5));
list.add(new Student("Marry", 19, 90.0));
list.add(new Student("Bob", 21, 70.0));
list.add(new Student("Alice", 18, 85.5));

// Display the list
System.out.println("List elements:");
for (Student s : list) {
   System.out.println(s);
}

// Ask the user to enter a student object and print the existence of the object
Scanner sc = new Scanner(System.in);
System.out.print("Enter the name of the student to search: ");
String name = sc.nextLine();
System.out.print("Enter the age of the student to search: ");
int age = sc.nextInt();
System.out.print("Enter the mark of the student to search: ");
double mark = sc.nextDouble();
Student studentToSearch = new Student(name, age, mark);
if (list.contains(studentToSearch)) {
   System.out.println(studentToSearch + " is present in the list");
} else {
   System.out.println(studentToSearch + " is not present in the list");
}

// Remove a specified student object
System.out.println("Removing " + studentToSearch + " from the list...");
list.remove(studentToSearch);
System.out.println("List elements after removing an element:");
for (Student s : list) {
```

```
            System.out.println(s);
        }

        // Count the number of objects present in the list
        int count = list.size();
        System.out.println("Number of objects present in the list: " + count);

        sc.close();
    }
}
```

Q3.

```
import java.util.Scanner;
import java.util.Stack;

public class DecimalToBinaryUsingStack {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a decimal number: ");
        int decimal = sc.nextInt();

        Stack<Integer> stack = new Stack<>();

        while (decimal > 0) {
            int remainder = decimal % 2;
            stack.push(remainder);
            decimal = decimal / 2;
        }

        System.out.print("Binary equivalent: ");
```

```
        while (!stack.isEmpty()) {
            System.out.print(stack.pop());
        }

        sc.close();
    }
}
```

Q4.

```
import java.util.Scanner;
import java.util.Stack;

public class PostfixEvaluationUsingStack {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a postfix expression: ");
        String postfix = sc.nextLine();

        Stack<Integer> stack = new Stack<>();

        for (int i = 0; i < postfix.length(); i++) {
            char c = postfix.charAt(i);
            if (Character.isDigit(c)) {
                stack.push(c - '0');
            } else {
                int operand2 = stack.pop();
                int operand1 = stack.pop();
                int result = performOperation(operand1, operand2, c);
                stack.push(result);
            }
```

```
        }

        System.out.println("Result: " + stack.pop());

        sc.close();
    }

    public static int performOperation(int operand1, int operand2, char operator) {
        switch (operator) {
            case '+':
                return operand1 + operand2;
            case '-':
                return operand1 - operand2;
            case '*':
                return operand1 * operand2;
            case '/':
                return operand1 / operand2;
            case '^':
                return (int) Math.pow(operand1, operand2);
            default:
                return 0;
        }
    }
}
```

Q5.

```
import java.util.ArrayDeque;
import java.util.ArrayList;

public class BreadthFirstSearchUsingArrayDeque {

    static class Graph {
```

```
private final int vertices;
private final ArrayList<Integer>[] adjacencyList;

Graph(int vertices) {
    this.vertices = vertices;
    adjacencyList = new ArrayList[vertices];

    for (int i = 0; i < vertices; i++) {
        adjacencyList[i] = new ArrayList<>();
    }
}

void addEdge(int source, int destination) {
    adjacencyList[source].add(destination);
    adjacencyList[destination].add(source);
}

void bfs(int startVertex) {
    boolean[] visited = new boolean[vertices];
    ArrayDeque<Integer> queue = new ArrayDeque<>();

    visited[startVertex] = true;
    queue.offer(startVertex);

    while (!queue.isEmpty()) {
        int vertex = queue.poll();
        System.out.print(vertex + " ");

        for (int neighbor : adjacencyList[vertex]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
```

```
                queue.offer(neighbor);
            }
        }
    }
  }

  public static void main(String[] args) {
    Graph graph = new Graph(6);
    graph.addEdge(0, 1);
    graph.addEdge(0, 2);
    graph.addEdge(1, 3);
    graph.addEdge(1, 4);
    graph.addEdge(2, 4);
    graph.addEdge(3, 4);
    graph.addEdge(3, 5);

    System.out.println("BFS traversal starting from vertex 0:");
    graph.bfs(0);
  }
}
```

Q6.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Stack;

public class GraphDFSUsingStack {

  private int vertices;
  private ArrayList<ArrayList<Integer>> adjacencyList;
```

```
public GraphDFSUsingStack(int vertices) {
    this.vertices = vertices;
    adjacencyList = new ArrayList<>(vertices);

    for (int i = 0; i < vertices; i++) {
        adjacencyList.add(new ArrayList<>());
    }
}

public void addEdge(int source, int destination) {
    adjacencyList.get(source).add(destination);
}

public void dfs(int start) {
    boolean[] visited = new boolean[vertices];
    Arrays.fill(visited, false);

    Stack<Integer> stack = new Stack<>();
    visited[start] = true;
    stack.push(start);

    while (!stack.isEmpty()) {
        int vertex = stack.pop();
        System.out.print(vertex + " ");

        for (int adjVertex : adjacencyList.get(vertex)) {
            if (!visited[adjVertex]) {
                visited[adjVertex] = true;
                stack.push(adjVertex);
            }
        }
```

```
        }
    }

    public static void main(String[] args) {
        GraphDFSUsingStack graph = new GraphDFSUsingStack(6);

        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(1, 3);
        graph.addEdge(1, 4);
        graph.addEdge(2, 4);
        graph.addEdge(3, 4);
        graph.addEdge(3, 5);
        graph.addEdge(4, 5);

        System.out.println("DFS traversal starting from vertex 0:");
        graph.dfs(0);
    }
}
```