RPS DAY 13-14 Assignments

Assignment 6

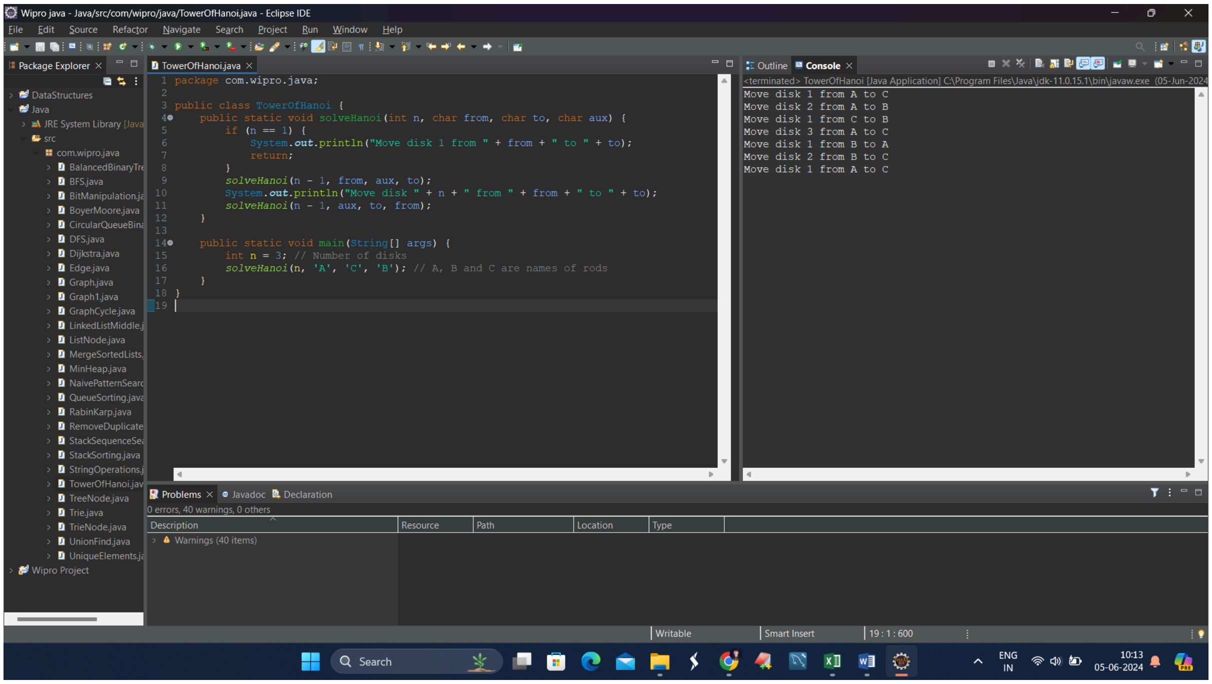
Name: Akshada Baad
Batch - CPPE

Day 13 and 14:

Task 1: Tower of Hanoi Solver

Create a program that solves the Tower of Hanoi puzzle for n disks. The solution should use recursion to move disks between three pegs (source, auxiliary, and destination) according to the game's rules. The program should print out each move required to solve the puzzle.

```
public class TowerOfHanoi {
    public static void solveHanoi(int n, char from, char to, char aux) {
        if (n == 1) {
            System.out.println("Move disk 1 from " + from + " to " + to);
            return;
        }
        solveHanoi(n - 1, from, aux, to);
        System.out.println("Move disk " + n + " from " + from + " to " + to);
        solveHanoi(n - 1, aux, to, from);
    }

    public static void main(String[] args) {
        int n = 3; // Number of disks
        solveHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
    }
}
```



```java
package com.wipro.java;

public class TowerOfHanoi {
    public static void solveHanoi(int n, char from, char to, char aux) {
        if (n == 1) {
            System.out.println("Move disk 1 from " + from + " to " + to);
            return;
        }
        solveHanoi(n - 1, from, aux, to);
        System.out.println("Move disk " + n + " from " + from + " to " + to);
        solveHanoi(n - 1, aux, to, from);
    }

    public static void main(String[] args) {
        int n = 3; // Number of disks
        solveHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
    }
}
```

```
<terminated> TowerOfHanoi [Java Application] C:\Program Files\Java\jdk-11.0.15.1\bin\javaw.exe  (05-Jun-2024
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

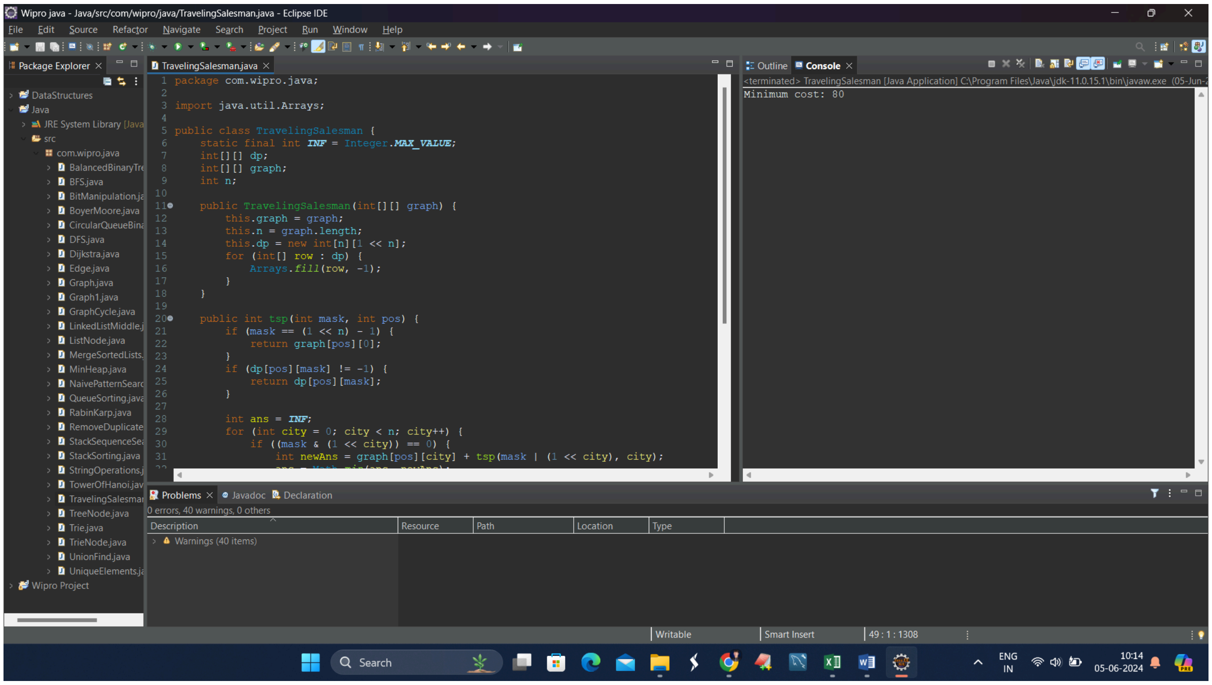
0 errors, 40 warnings, 0 others

Description	Resource	Path	Location	Type
Warnings (40 items)				

Task 2: Traveling Salesman Problem

Create a function int FindMinCost(int[,] graph) that takes a 2D array representing the graph where graph[i][j] is the cost to travel from city i to city j. The function should return the minimum cost to visit all cities and return to the starting city. Use dynamic programming for this solution.

```java
import java.util.Arrays;


public class TravelingSalesman {
    static final int INF = Integer.MAX_VALUE;
    int[][] dp;
    int[][] graph;
    int n;


    public TravelingSalesman(int[][] graph) {
        this.graph = graph;
        this.n = graph.length;
        this.dp = new int[n][1 << n];
        for (int[] row : dp) {
            Arrays.fill(row, -1);
        }
    }

    public int tsp(int mask, int pos) {
        if (mask == (1 << n) - 1) {
            return graph[pos][0];
        }
        if (dp[pos][mask] != -1) {
            return dp[pos][mask];
        }

        int ans = INF;
        for (int city = 0; city < n; city++) {
            if ((mask & (1 << city)) == 0) {
                int newAns = graph[pos][city] + tsp(mask | (1 << city), city);
                ans = Math.min(ans, newAns);
            }
        }
        return dp[pos][mask] = ans;
    }

    public static void main(String[] args) {
        int[][] graph = {
            {0, 10, 15, 20},
            {10, 0, 35, 25},
            {15, 35, 0, 30},
            {20, 25, 30, 0}
        };
        TravelingSalesman tsp = new TravelingSalesman(graph);
        System.out.println("Minimum cost: " + tsp.tsp(1, 0)); // Expected output: 80
    }
}
```



```
package com.wipro.java;

import java.util.Arrays;

public class TravelingSalesman {
    static final int INF = Integer.MAX_VALUE;
    int[][] dp;
    int[][] graph;
    int n;

    public TravelingSalesman(int[][] graph) {
        this.graph = graph;
        this.n = graph.length;
        this.dp = new int[n][1 << n];
        for (int[] row : dp) {
            Arrays.fill(row, -1);
        }
    }

    public int tsp(int mask, int pos) {
        if (mask == (1 << n) - 1) {
            return graph[pos][0];
        }
        if (dp[pos][mask] != -1) {
            return dp[pos][mask];
        }

        int ans = INF;
        for (int city = 0; city < n; city++) {
            if ((mask & (1 << city)) == 0) {
                int newAns = graph[pos][city] + tsp(mask | (1 << city), city);
```

```
<terminated> TravelingSalesman [Java Application] C:\Program Files\Java\jdk-11.0.15.1\bin\javaw.exe (05-Jun-
Minimum cost: 80
```

0 errors, 40 warnings, 0 others

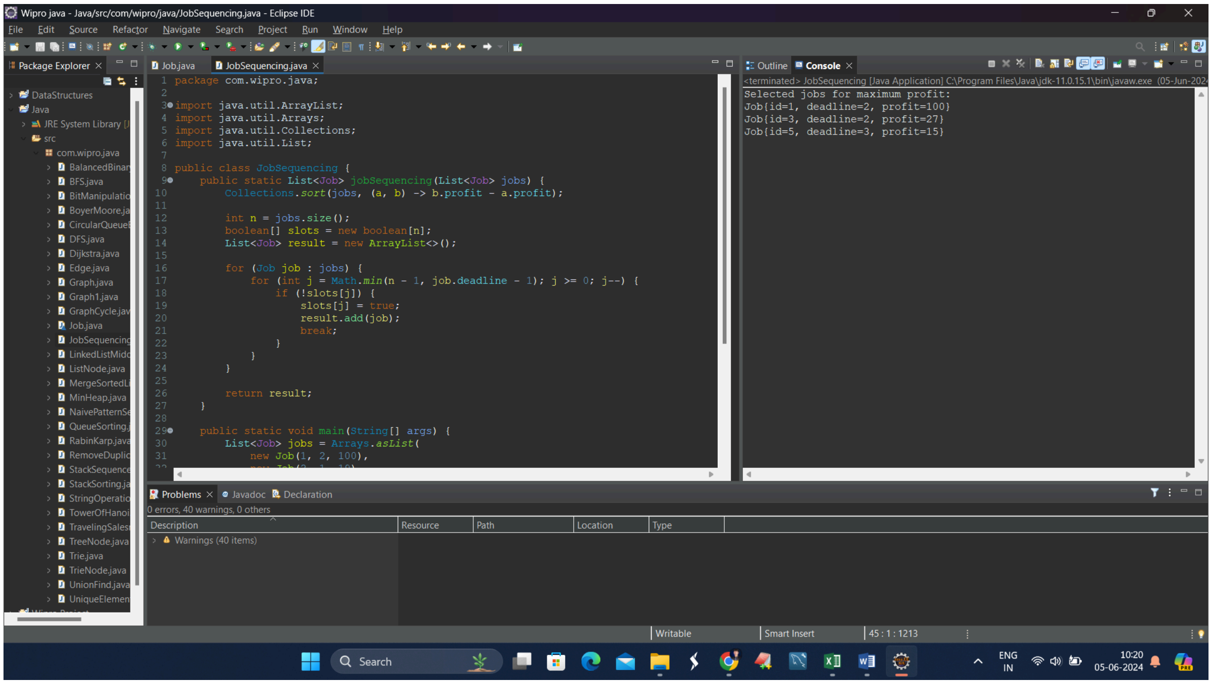
Description	Resource	Path	Location	Type
Warnings (40 items)				

Task 3: Job Sequencing Problem

Define a class Job with properties int Id, int Deadline, and int Profit. Then implement a function List<Job> JobSequencing(List<Job> jobs) that takes a list of jobs and returns the maximum profit sequence of jobs that can be done before the deadlines. Use the greedy method to solve this problem.

```
import java.util.*;

class Job {
    int id, deadline, profit;

    public Job(int id, int deadline, int profit) {
        this.id = id;
        this.deadline = deadline;
        this.profit = profit;
    }

    @Override
    public String toString() {
        return "Job{" + "id=" + id + ", deadline=" + deadline + ", profit=" + profit + '}';
    }
}

public class JobSequencing {
    public static List<Job> jobSequencing(List<Job> jobs) {
        Collections.sort(jobs, (a, b) -> b.profit - a.profit);

        int n = jobs.size();
        boolean[] slots = new boolean[n];
        List<Job> result = new ArrayList<>();

        for (Job job : jobs) {
            for (int j = Math.min(n - 1, job.deadline - 1); j >= 0; j--) {
                if (!slots[j]) {
                    slots[j] = true;
                    result.add(job);
                    break;
                }
            }
        }

        return result;
```

```java
    }

    public static void main(String[] args) {
        List<Job> jobs = Arrays.asList(
            new Job(1, 2, 100),
            new Job(2, 1, 19),
            new Job(3, 2, 27),
            new Job(4, 1, 25),
            new Job(5, 3, 15)
        );

        List<Job> result = jobSequencing(jobs);
        System.out.println("Selected jobs for maximum profit:");
        for (Job job : result) {
            System.out.println(job);
        }
    }
}
```



```java
package com.wipro.java;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class JobSequencing {
    public static List<Job> jobSequencing(List<Job> jobs) {
        Collections.sort(jobs, (a, b) -> b.profit - a.profit);

        int n = jobs.size();
        boolean[] slots = new boolean[n];
        List<Job> result = new ArrayList<>();

        for (Job job : jobs) {
            for (int j = Math.min(n - 1, job.deadline - 1); j >= 0; j--) {
                if (!slots[j]) {
                    slots[j] = true;
                    result.add(job);
                    break;
                }
            }
        }

        return result;
    }

    public static void main(String[] args) {
        List<Job> jobs = Arrays.asList(
            new Job(1, 2, 100),
```

```
<terminated> JobSequencing [Java Application] C:\Program Files\Java\jdk-11.0.15.1\bin\javaw.exe  (05-Jun-2024
Selected jobs for maximum profit:
Job{id=1, deadline=2, profit=100}
Job{id=3, deadline=2, profit=27}
Job{id=5, deadline=3, profit=15}
```

0 errors, 40 warnings, 0 others

Warnings (40 items)