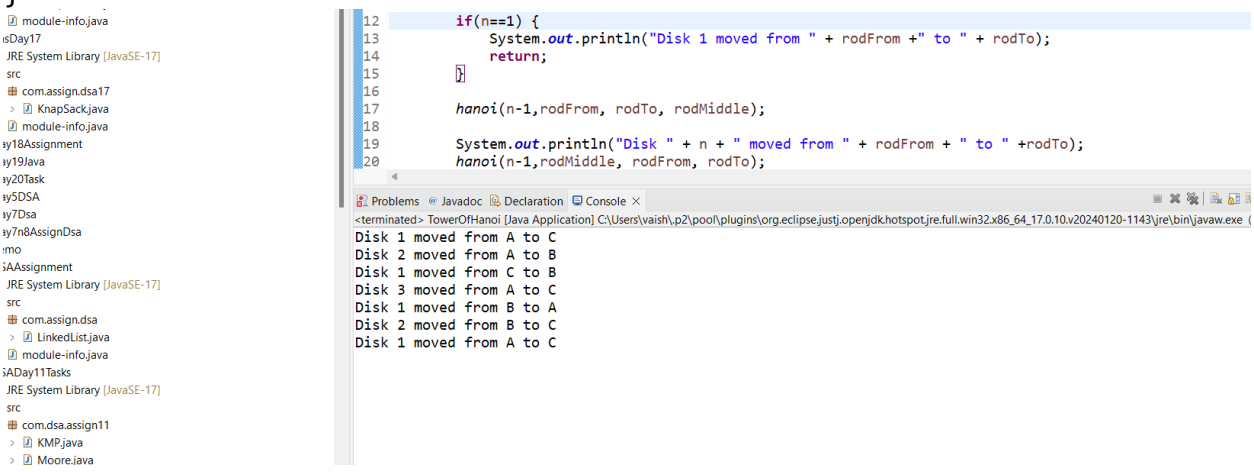
Day 13 and 14:

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.

```
package com.wipro.algos;

public class TowerN {
	public class TowerOfHanoi {

		public static void main(String[] args) {
			if (args.length != 1) {
				System.out.println("Usage: java TowerOfHanoi
<number_of_disks>");
				return;
			}

			try {
				int n = Integer.parseInt(args[0]);
				if (n <= 0) {
					System.out.println("The number of disks must be a
positive integer.");
					return;
				}
				hanoi(n, "A", "B", "C");
			} catch (NumberFormatException e) {
				System.out.println("Please enter a valid integer for
the number of disks.");
			}
		}

		private static void hanoi(int n, String rodFrom, String
rodMiddle, String rodTo) {
			if (n == 1) {
				System.out.println("Disk 1 moved from " + rodFrom + "
to " + rodTo);
				return;
			}

			hanoi(n - 1, rodFrom, rodTo, rodMiddle);
```

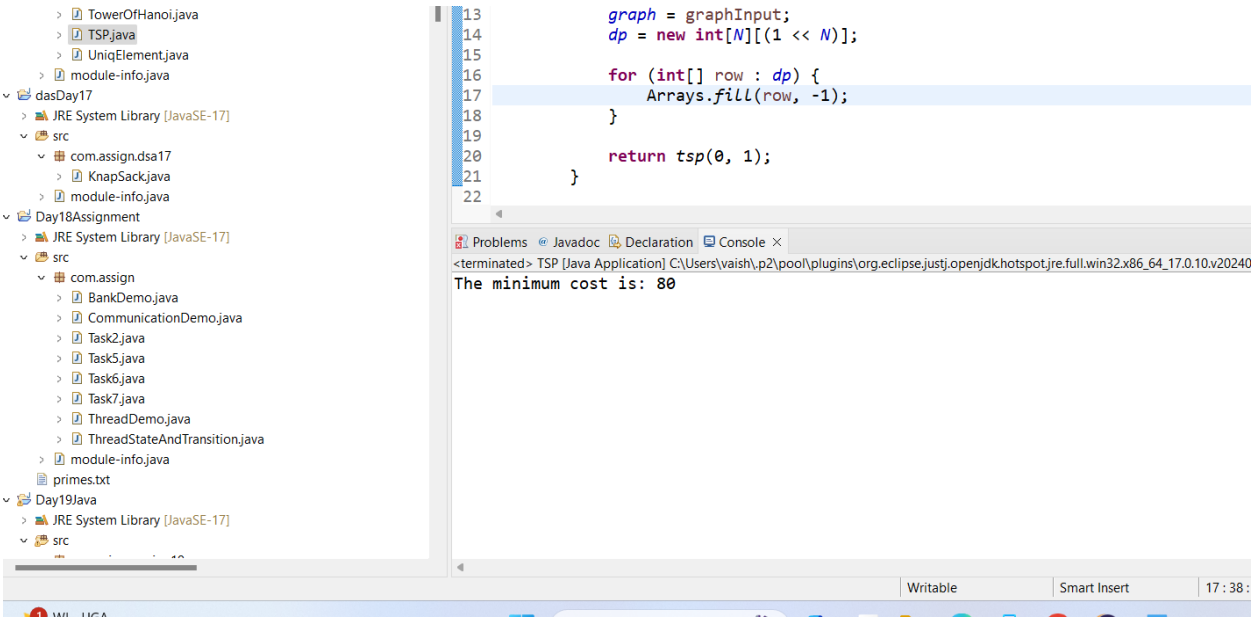
```
            System.out.println("Disk " + n + " moved from " + rodFrom
+ " to " + rodTo);
            hanoi(n - 1, rodMiddle, rodFrom, rodTo);
        }
    }
}
```



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.

```
package com.wipro.algos;

import java.util.Arrays;

public class TSP {
	private static int N;
        private static int[][] dp;
        private static int[][] graph;
        private static final int INF = Integer.MAX_VALUE / 2; // A
large value representing infinity

        public static int FindMinCost(int[][] graphInput) {
            N = graphInput.length;
            graph = graphInput;
            dp = new int[N][(1 << N)];

            for (int[] row : dp) {
                Arrays.fill(row, -1);
            }

            return tsp(0, 1);
```

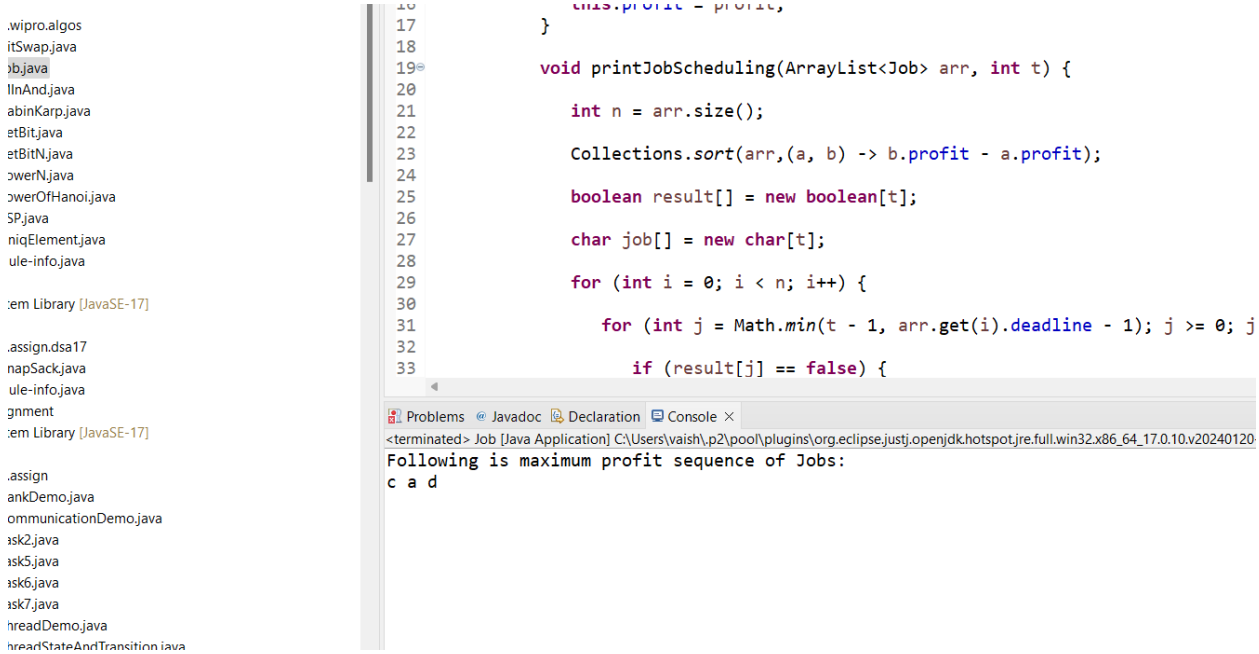
```
        }

        private static int tsp(int pos, int mask) {
            if (mask == (1 << N) - 1) {
                return graph[pos][0]; // Return to starting city
            }

            if (dp[pos][mask] != -1) {
                return dp[pos][mask];
            }

            int minCost = INF;
            for (int city = 0; city < N; city++) {
                if ((mask & (1 << city)) == 0) {
                    int newCost = graph[pos][city] + tsp(city, mask |
(1 << city));
                    minCost = Math.min(minCost, newCost);
                }
            }

            dp[pos][mask] = minCost;
            return minCost;
        }

        public static void main(String[] args) {
            int[][] graph = {
                {0, 10, 15, 20},
                {10, 0, 35, 25},
                {15, 35, 0, 30},
                {20, 25, 30, 0}
            };

            System.out.println("The minimum cost is: " +
FindMinCost(graph));
        }
}
```



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.

```
package com.wipro.algos;

import java.util.ArrayList;
import java.util.Collections;

public class Job {


        char id;
        int deadline, profit;

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

        void printJobScheduling(ArrayList<Job> arr, int t) {

           int n = arr.size();
```

```
                Collections.sort(arr,(a, b) -> b.profit - a.profit);

                boolean result[] = new boolean[t];

                char job[] = new char[t];

                for (int i = 0; i < n; i++) {

                   for (int j = Math.min(t - 1, arr.get(i).deadline -
1); j >= 0; j--) {

                      if (result[j] == false) {
                         result[j] = true;
                         job[j] = arr.get(i).id;
                         break;
                      }
                   }
                }

                for (char jb : job)
                System.out.print(jb + " ");
                System.out.println();
             }

             public static void main(String args[]) {
                ArrayList<Job> arr = new ArrayList<Job>();
                arr.add(new Job('a', 2, 100));
                arr.add(new Job('b', 2, 20));
                arr.add(new Job('c', 1, 40));
                arr.add(new Job('d', 3, 35));
                arr.add(new Job('e', 1, 25));

                System.out.println("Following is maximum profit
sequence of Jobs: ");
                Job job = new Job();

                job.printJobScheduling(arr, 3);
             }
          }
```


.wipro.algos
itSwap.java
ob.java
lInAnd.java
abinKarp.java
etBit.java
etBitN.java
owerN.java
owerOfHanoi.java
SP.java
niqElement.java
ule-info.java
em Library [JavaSE-17]
.assign.dsa17
napSack.java
ule-info.java
gnment
em Library [JavaSE-17]
.assign
ankDemo.java
ommunicationDemo.java
sk2.java
sk5.java
sk6.java
sk7.java
hreadDemo.java
hreadStateAndTransition.java
this.profit = profit;
}
void printJobScheduling(ArrayList<Job> arr, int t) {
int n = arr.size();
Collections.sort(arr,(a, b) -> b.profit - a.profit);
boolean result[] = new boolean[t];
char job[] = new char[t];
for (int i = 0; i < n; i++) {
for (int j = Math.min(t - 1, arr.get(i).deadline - 1); j >= 0; j
if (result[j] == false) {
Problems
Javadoc
Declaration
Console
<terminated> Job [Java Application] C:\Users\vaish\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120
Following is maximum profit sequence of Jobs:
c a d