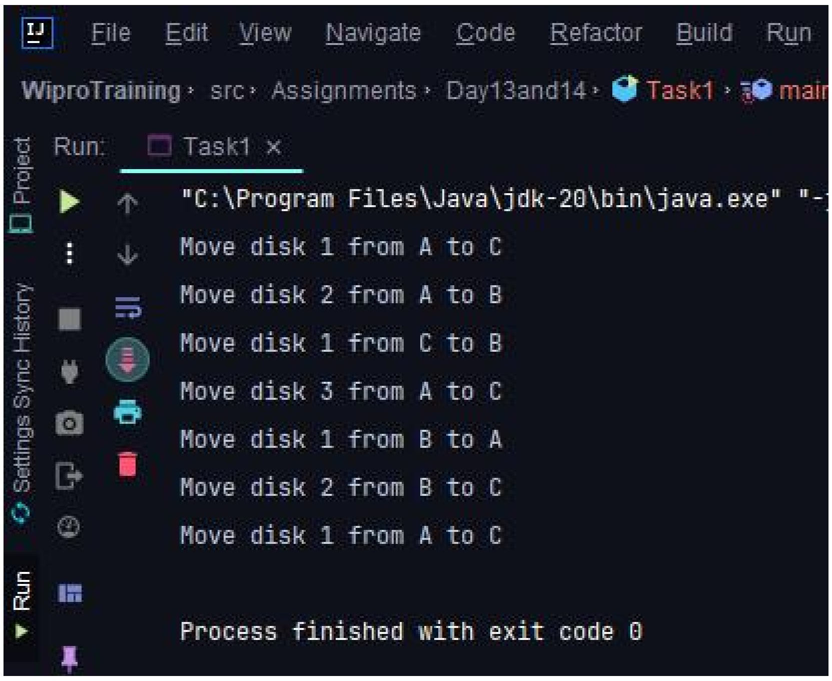
# Day 13&14

**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.

**Program:**

```
package Assignments.Day13and14;

public class Task1 {

   public static void solveHanoi(int n, char source, char auxiliary, char
destination) {
      if (n == 1) {
         System.out.println("Move disk 1 from " + source + " to " + destination);
         return;
      }

      solveHanoi(n - 1, source, destination, auxiliary);

      System.out.println("Move disk " + n + " from " + source + " to " +
destination);
      solveHanoi(n - 1, auxiliary, source, destination);
   }

   public static void main(String[] args) {
      int n = 3;
      char sourcePeg = 'A';
      char auxiliaryPeg = 'B';
      char destinationPeg = 'C';

      solveHanoi(n, sourcePeg, auxiliaryPeg, destinationPeg);
   }
}
```

**Output:**


File Edit View Navigate Code Refactor Build Run
WiproTraining › src › Assignments › Day13and14 › Task1 › main
Run: Task1

```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-
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

Process finished with exit code 0
```

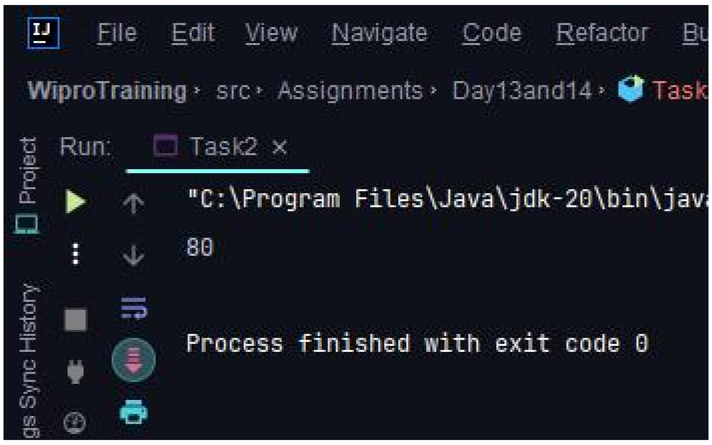
**Task 2: Traveling Salesman Problem**
Create a function int FindMinCost(int[,] graph) that takes a 2D array representing the graph where graph[i][j] is the cost to travel from city i to city j. The function should return the minimum cost to visit all cities and return to the starting city. Use dynamic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

**Program:**

```
package Assignments.Day13and14;

import java.util.*;

class Task2{

   static int V = 4;

   static int travellingSalesmanProblem(int [][] graph, int s) {

      ArrayList<Integer> vertex = new ArrayList<Integer>();

      for (int i = 0; i < V; i++)
         if (i != s)
            vertex.add(i);

      int min_path = Integer.MAX_VALUE;
```

```
        do {

            int current_pathweight = 0;
            int k = s;

            for (Integer integer : vertex) {
                current_pathweight += graph[k][integer];
                k = integer;
            }
            current_pathweight += graph[k][s];

            min_path = Math.min(min_path, current_pathweight);
        }
        while (findNextPermutation(vertex));
        return min_path;
    }

    public static ArrayList<Integer> swap(ArrayList<Integer> data, int left, int
right) {
        int temp = data.get(left);
        data.set(left, data.get(right));
        data.set(right, temp);
        return data;
    }

    public static ArrayList<Integer> reverse(ArrayList<Integer> data, int left, int
right) {
        while (left < right) {
            int temp = data.get(left);
            data.set(left++, data.get(right));
            data.set(right--, temp);
        }
        return data;
    }

    public static boolean findNextPermutation(ArrayList<Integer> data) {

        if (data.size() <= 1)
            return false;

        int last = data.size() - 2;

        while (last >= 0) {
            if (data.get(last) < data.get(last + 1)) {
                break;
            }
            last--;
        }

        if (last < 0)
            return false;

        int nextGreater = data.size() - 1;

        for (int i = data.size() - 1; i > last; i--) {
            if (data.get(i) > data.get(last)) {
                nextGreater = i;
                break;
            }
        }

        data = swap(data, nextGreater, last);
        data = reverse(data, last + 1, data.size() - 1);

        return true;
    }
    public static void main(String [] args) {
        int [][] graph = {{0, 10, 15, 20},
                          {10, 0, 35, 25},
                          {15, 35, 0, 30},
                          {20, 25, 30, 0}};
        int s = 0;
        System.out.println(travellingSalesmanProblem(graph, s));
    }
}
```

**Output:**



```
"C:\Program Files\Java\jdk-20\bin\jav
80

Process finished with exit code 0
```

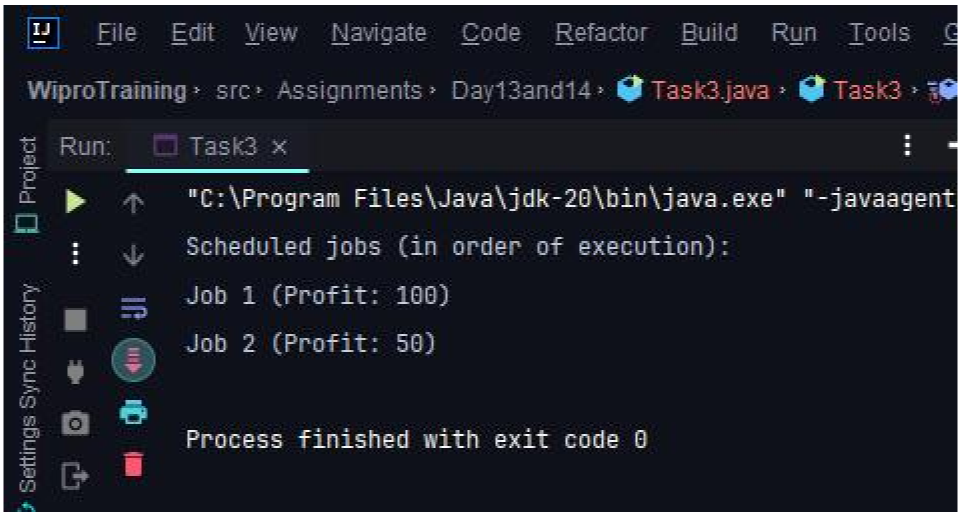
**Task 3: Job Sequencing Problem**
Define a class Job with properties int Id, int Deadline, and int Profit. Then implement a function List<Job> JobSequencing(List<Job> jobs) that takes a list of jobs and returns the maximum profit sequence of jobs that can be done before the deadlines. Use the greedy method to solve this problem.

**Program:**

```
package Assignments.Day13and14;

import java.util.*;

class Job {
   int id;
   int deadline;
   int profit;

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

public class Task3 {

   public static List<Job> jobSequencing(List<Job> jobs) {

      jobs.sort((a, b) -> b.profit - a.profit);

      int maxDeadline = jobs.stream().mapToInt(job -> 
job.deadline).max().orElse(0);
      boolean[] slot = new boolean[maxDeadline + 1];
      List<Job> result = new ArrayList<>();

      for (Job job : jobs) {
         for (int d = job.deadline; d > 0; d--) {
            if (!slot[d]) {
               slot[d] = true;
               result.add(job);
               break;
            }
         }
      }
```

```
        return result;
    }

    public static void main(String[] args) {
        List<Job> jobs = new ArrayList<>();
        jobs.add(new Job(1, 2, 100));
        jobs.add(new Job(2, 1, 50));
        jobs.add(new Job(3, 2, 10));
        jobs.add(new Job(4, 1, 20));

        List<Job> scheduledJobs = jobSequencing(jobs);
        System.out.println("Scheduled jobs (in order of execution):");
        for (Job job : scheduledJobs) {
            System.out.println("Job " + job.id + " (Profit: " + job.profit + ")");
        }
    }
}
```

**Output:**


File Edit View Navigate Code Refactor Build Run Tools
WiproTraining › src › Assignments › Day13and14 › Task3.java › Task3
Run: Task3
Project
Settings Sync History

```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent
Scheduled jobs (in order of execution):
Job 1 (Profit: 100)
Job 2 (Profit: 50)

Process finished with exit code 0
```