**Name:Vaishnavi Ingole**

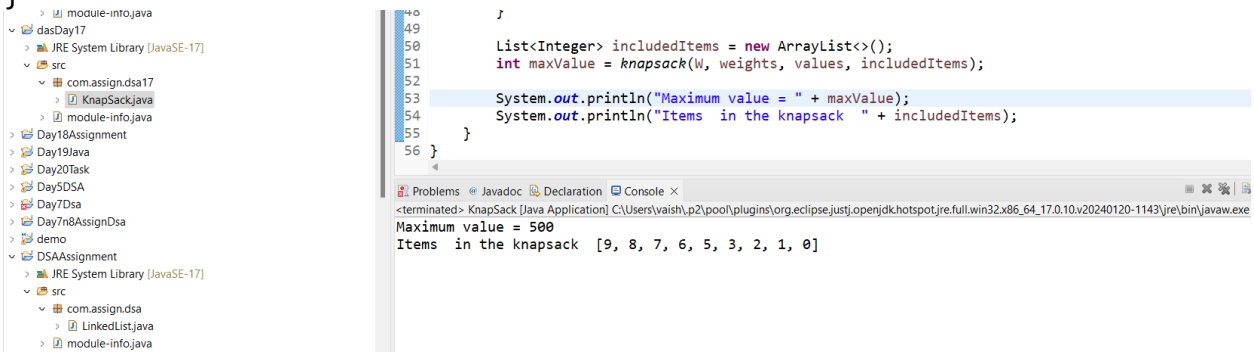
**Mail:vaishnavingole54@gmail.com**

**Day 15 and 16:**

**Task 1: Knapsack Problem** Write a function int Knapsack(int W, int[] weights, int[] values) in C# that determines the maximum value of items that can fit into a knapsack with a capacity W. The function should handle up to 100 items. Find the optimal way to fill the knapsack with the given items to achieve the maximum total value. You must consider that you cannot break items, but have to include them whole.

```
package com.assign.dsa17;

import java.util.ArrayList;
import java.util.List;

public class KnapSack {

    public static int knapsack(int W, int[] weights, int[] values,
List<Integer> includedItems) {
        int n = weights.length;
        int[][] dp = new int[n + 1][W + 1];


        for (int i = 0; i <= n; i++) {
            for (int w = 0; w <= W; w++) {
                if (i == 0 || w == 0) {
                    dp[i][w] = 0;
                } else if (weights[i - 1] <= w) {
                    dp[i][w] = Math.max(values[i - 1] + dp[i - 1][w -
weights[i - 1]], dp[i - 1][w]);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }


        int res = dp[n][W];
        int w = W;
        for (int i = n; i > 0 && res > 0; i--) {
            if (res != dp[i - 1][w]) {
                includedItems.add(i - 1);
```

```
                res -= values[i - 1];
                w -= weights[i - 1];
            }
        }

        return dp[n][W];
    }

    public static void main(String[] args) {
        int W = 50;
        int[] weights = new int[100];
        int[] values = new int[100];


        for (int i = 0; i < 100; i++) {
            weights[i] = i + 1;
            values[i] = (i + 1) * 10;
        }

        List<Integer> includedItems = new ArrayList<>();
        int maxValue = knapsack(W, weights, values, includedItems);

        System.out.println("Maximum value = " + maxValue);
        System.out.println("Items  in the knapsack  " +
includedItems);
    }
}
```

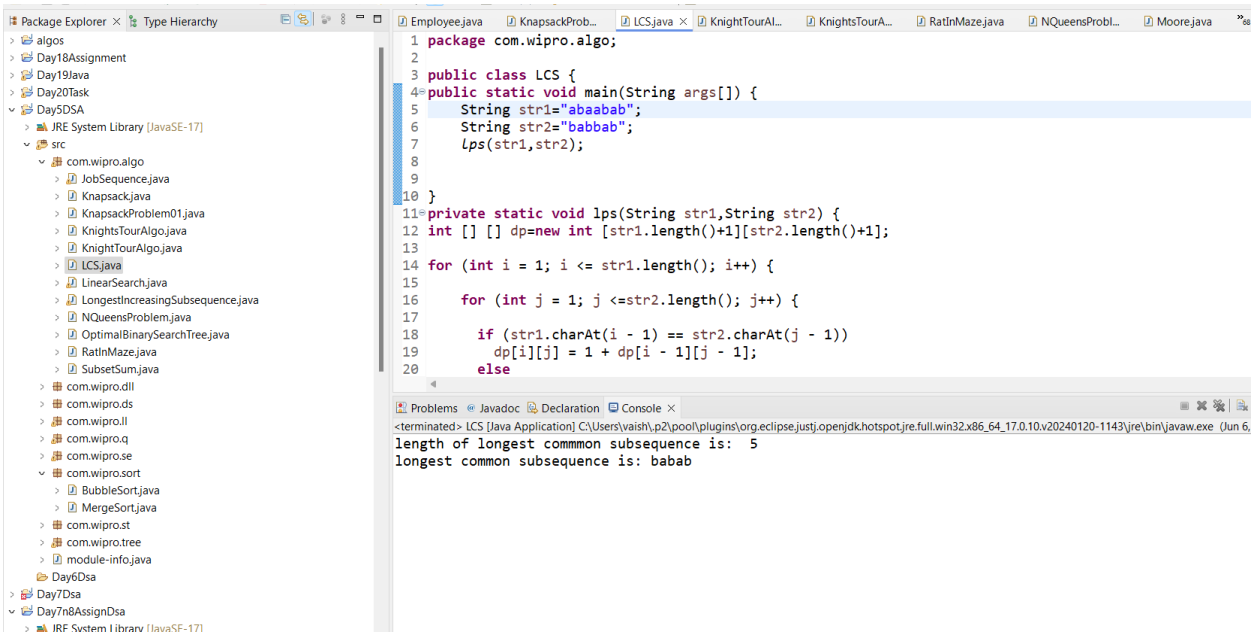



**Task 2: Longest Common Subsequence** Implement int LCS(string text1, string text2) to find the length of the longest common subsequence between two strings.

```
package com.wipro.algo;


import java.util.ArrayList;

import java.util.List;
```

```
public class KnapsackProblem01 {

	public static void main(String[] args) {

		int capacity =8;
		int[] values= {1,2,5,6};
		int[] weights = {2,3,4,5};
		int n = values.length;

		int maxValue= knapsack(capacity,weights,values, n);
		System.out.println("Maximum value that can be obtained :"+ maxValue);

	}

	private static int knapsack(int capacity, int[] weights, int[] profits, int n) {
		int[][] t =new int[n+1][capacity+1];

		for(int rownum =0 ;rownum<=n; rownum++) {
			for(int colnum =0; colnum <=capacity ; colnum++) {
				if(rownum ==0 || colnum ==0) {
					t[rownum][colnum] =0;
				}else if(weights[rownum-1] <= colnum) {
					t[rownum][colnum] =  Math.max(t[rownum-1][colnum],
profits[rownum -1] +
							t[rownum -1][colnum -weights[rownum-1]]);

				}else {
					t[rownum][colnum] = t[rownum-1][colnum];
```

```
                    }
                }
            }

            List<Integer> itemsIncluded  = findItemsIncluded(t,weights,profits,n,capacity);
            System.out.println("Items  in knapsack :" + itemsIncluded);
            return t[n][capacity];
    }

    private static List<Integer> findItemsIncluded(int[][] t, int[] weights, int[] profits, int n, int
capacity) {

        List<Integer> Items = new ArrayList<>();
        int i = n;
        int k = capacity;

        while (i > 0 && k > 0) {
            if (t[i][k] != t[i - 1][k]) {
                Items.add(i - 1);
                k -= weights[i - 1];
            }
            i--;
        }

        return Items;
    }

}
```


Package Explorer
Type Hierarchy
algos
Day18Assignment
Day19Java
Day20Task
Day5DSA
JRE System Library [JavaSE-17]
src
com.wipro.algo
JobSequence.java
Knapsack.java
KnapsackProblem01.java
KnightsTourAlgo.java
KnightTourAlgo.java
LCS.java
LinearSearch.java
LongestIncreasingSubsequence.java
NQueensProblem.java
OptimalBinarySearchTree.java
RatInMaze.java
SubsetSum.java
com.wipro.dll
com.wipro.ds
com.wipro.ll
com.wipro.q
com.wipro.se
com.wipro.sort
BubbleSort.java
MergeSort.java
com.wipro.st
com.wipro.tree
module-info.java
Day6Dsa
Day7Dsa
Day7n8AssignDsa
JRE System Library [JavaSE-17]
Employee.java
KnapsackProb...
LCS.java
KnightTourAl...
KnightsTourA...
RatInMaze.java
NQueensProbl...
Moore.java
1 package com.wipro.algo;
2
3 public class LCS {
4 public static void main(String args[]) {
5 String str1="abaabab";
6 String str2="babbab";
7 lps(str1,str2);
8
9
10 }
11 private static void lps(String str1,String str2) {
12 int [] [] dp=new int [str1.length()+1][str2.length()+1];
13
14 for (int i = 1; i <= str1.length(); i++) {
15
16 for (int j = 1; j <=str2.length(); j++) {
17
18 if (str1.charAt(i - 1) == str2.charAt(j - 1))
19 dp[i][j] = 1 + dp[i - 1][j - 1];
20 else
Problems
Javadoc
Declaration
Console
<terminated> LCS [Java Application] C:\Users\vaish\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (Jun 6,
length of longest commmon subsequence is: 5
longest common subsequence is: babab