

## Day 15 and 16: Manisha Assignment

### 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.

#### Explanation:

##### 1. Knapsack Class:

- Contains the `knapsack` method to solve the knapsack problem.
- Contains the `main` method to test the `knapsack` method.

##### 2. knapsack Method:

- Takes three parameters: `W` (the capacity of the knapsack), `weights` (an array of item weights), and `values` (an array of item values).
- Uses dynamic programming to fill a 2D array `dp` where `dp[i][w]` represents the maximum value of items that can be obtained with the first `i` items and a knapsack capacity of `w`.
- Iterates through each item and each possible knapsack capacity, updating the `dp` array based on whether the current item can be included in the knapsack.
- Returns the maximum value that can be obtained with the given items and knapsack capacity.

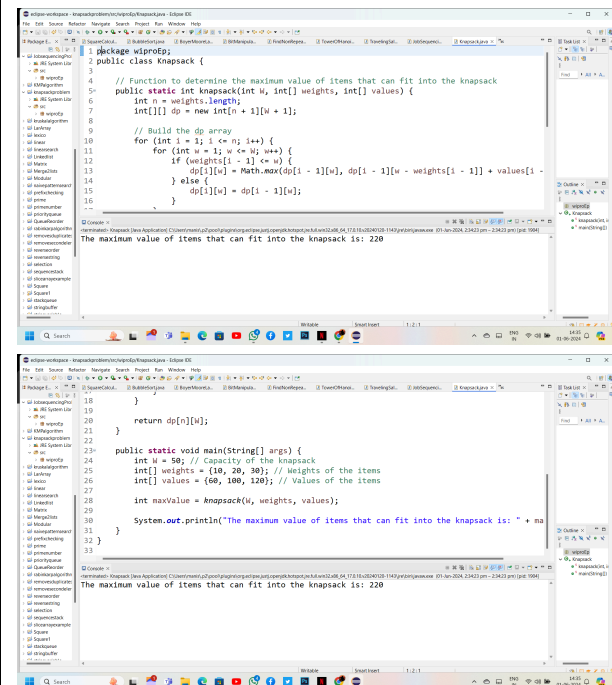
##### 3. main Method:

- Defines the capacity of the knapsack and arrays of item weights and values.
- Calls the `knapsack` method with the defined inputs.
- Prints the maximum value of items that can fit into the knapsack.

#### Here Output as Follows: Explanation of Output:

- The output shows the maximum value of items that can be included in the knapsack with a given capacity.
- In this case, the maximum value is 220, achieved by including items with weights 20 and 30 (values 100 and 120).

This code can be run in Eclipse or any other Java development environment. It demonstrates the dynamic programming approach to solving the knapsack problem efficiently.



```
1 package wipro;
2 public class Knapsack {
3     // Function to determine the maximum value of items that can fit into the knapsack
4     public static int knapsack(int W, int[] weights, int[] values) {
5         int n = weights.length;
6         int[][] dp = new int[n + 1][W + 1];
7
8         // Build the dp array
9         for (int i = 1; i <= n; i++) {
10             for (int w = 1; w <= W; w++) {
11                 if (weights[i - 1] <= w) {
12                     dp[i][w] = Math.max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1]);
13                 } else {
14                     dp[i][w] = dp[i - 1][w];
15                 }
16             }
17         }
18         return dp[n][W];
19     }
20
21     public static void main(String[] args) {
22         int W = 50; // Capacity of the knapsack
23         int[] weights = {10, 20, 30}; // Weights of the items
24         int[] values = {60, 100, 120}; // Values of the items
25         int maxvalue = knapsack(W, weights, values);
26         System.out.println("The maximum value of items that can fit into the knapsack is: " + maxvalue);
27     }
28 }
```

The maximum value of items that can fit into the knapsack is: 220

## Task 2: Longest Common Subsequence

Implement `int LCS(string text1, string text2)` to find the length of the longest common subsequence between two strings.

### Explanation:

#### 1. LongestCommonSubsequence Class:

- Contains the `LCS` method to solve the longest common subsequence problem.
- Contains the `main` method to test the `LCS` method.

#### 2. LCS Method:

- Takes two parameters: `text1` and `text2`, which are the two strings to be compared.

- Uses dynamic programming to fill a 2D array `dp` where `dp[i][j]` represents the length of the LCS of the first `i` characters of `text1` and the first `j` characters of `text2`.

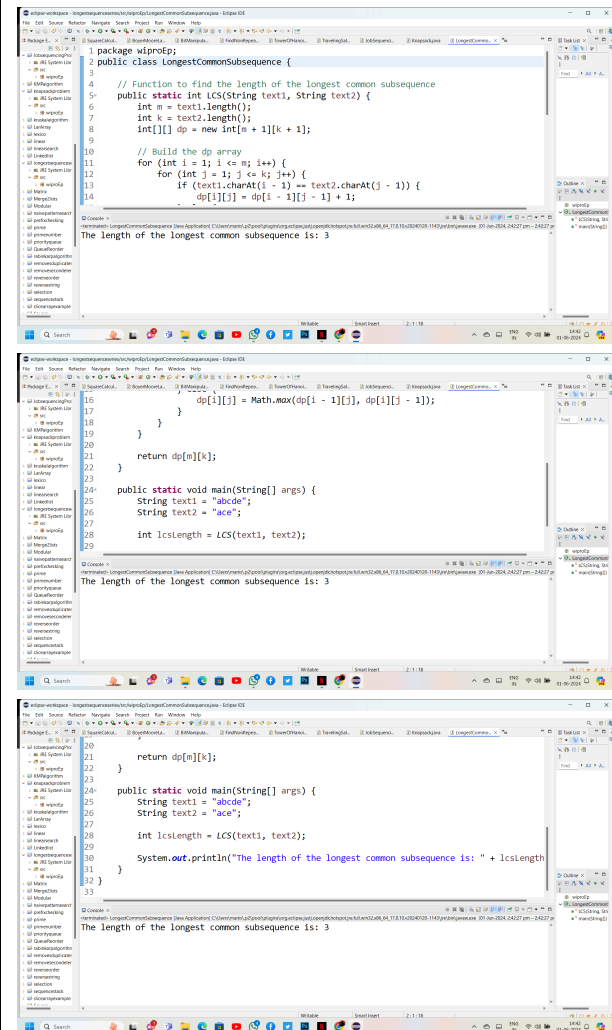
- Iterates through each character of both strings, updating the `dp` array based on whether the current characters of `text1` and `text2` match.

- Returns the length of the longest common subsequence.

#### 3. main Method:

- Defines two sample strings, `text1` and `text2`.
- Calls the `LCS` method with the sample strings.
- Prints the length of the longest common subsequence.

### Here output as Follows:



```
1 package wipro;
2 public class LongestCommonSubsequence {
3
4     // Function to find the length of the longest common subsequence
5     public static int LCS(String text1, String text2) {
6         int m = text1.length();
7         int k = text2.length();
8         int[][] dp = new int[m + 1][k + 1];
9
10        // Build the dp array
11        for (int i = 1; i <= m; i++) {
12            for (int j = 1; j <= k; j++) {
13                if (text1.charAt(i - 1) == text2.charAt(j - 1)) {
14                    dp[i][j] = dp[i - 1][j - 1] + 1;
15                } else {
16                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
17                }
18            }
19        }
20        return dp[m][k];
21    }
22
23    public static void main(String[] args) {
24        String text1 = "abcde";
25        String text2 = "ace";
26
27        int lcsLength = LCS(text1, text2);
28
29        System.out.println("The length of the longest common subsequence is: " + lcsLength);
30    }
31 }
32 }
```

The length of the longest common subsequence is: 3