

## Task 1: Knapsack Problem

Write a function `int Knapsack(int W, int[] weights, int[] values)` in java that determines the maximum value of items that can fit into a knapsack with a capacity `W`.

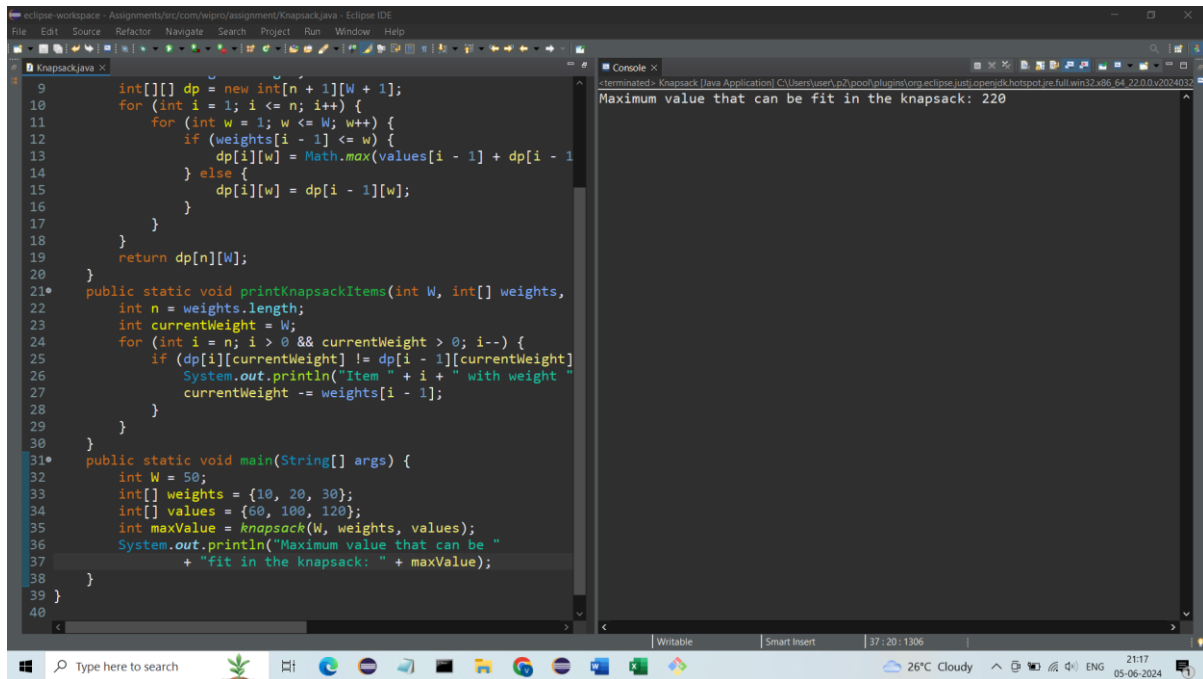
The function should handle up to 100 items. Find the optimal way to fill the knapsack with the given items to achieve the maximum total value. You must consider that you cannot break items, but have to include them whole.

### Function: -

```
public static int knapsack(int W, int[] weights, int[] values) {
    if (weights == null || values == null || weights.length != values.length || weights.length > 100) {
        throw new IllegalArgumentException("Invalid input arguments");
    }
    int n = weights.length;
    int[][] dp = new int[n + 1][W + 1];
    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= W; w++) {
            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];
            }
        }
    }
    return dp[n][W];
}
```

```
public static void printKnapsackItems(int W, int[] weights, int[] values, int[][] dp) {
    int n = weights.length;
    int currentWeight = W;
    for (int i = n; i > 0 && currentWeight > 0; i--) {
        if (dp[i][currentWeight] != dp[i - 1][currentWeight]) {
            System.out.println("Item " + i + " with weight " + weights[i - 1] + " and value " + values[i - 1] + " is picked.");
            currentWeight -= weights[i - 1];
        }
    }
}
```

### Output: -



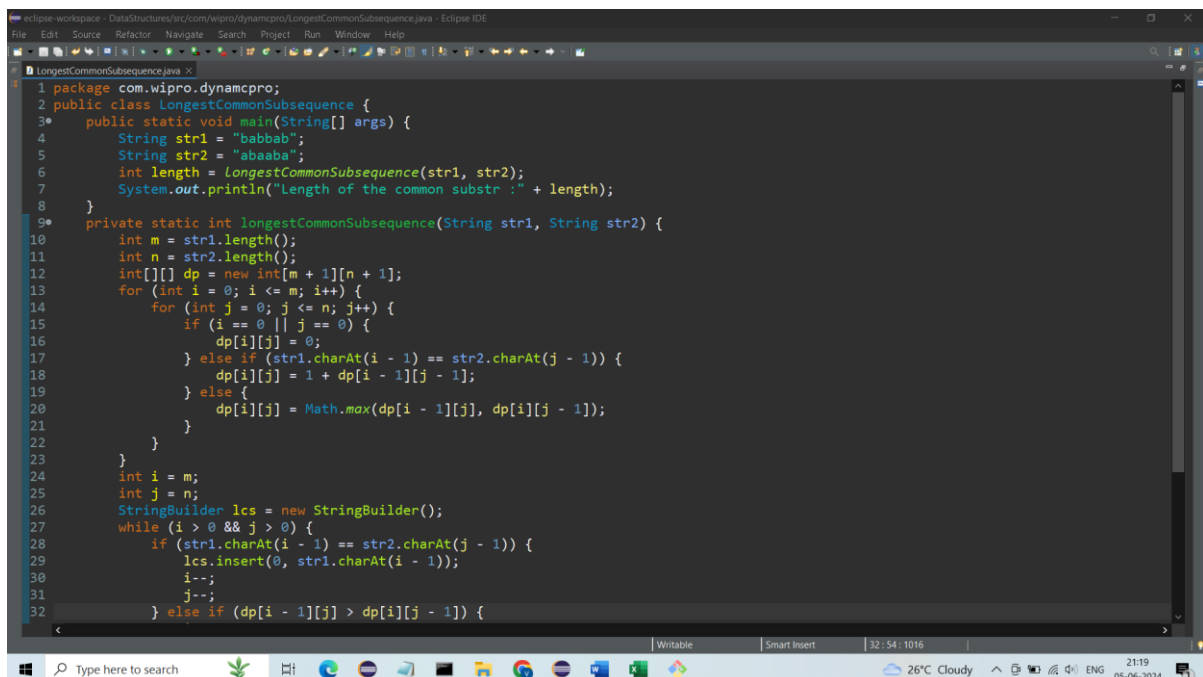
```
9 int[][] dp = new int[n + 1][W + 1];
10 for (int i = 1; i <= n; i++) {
11     for (int w = 1; w <= W; w++) {
12         if (weights[i - 1] <= w) {
13             dp[i][w] = Math.max(values[i - 1] + dp[i - 1][w], dp[i - 1][w]);
14         } else {
15             dp[i][w] = dp[i - 1][w];
16         }
17     }
18 }
19 return dp[n][W];
20 }
21 public static void printKnapsackItems(int W, int[] weights,
22 int n = weights.length;
23 int currentWeight = W;
24 for (int i = n; i > 0 && currentWeight > 0; i--) {
25     if (dp[i][currentWeight] != dp[i - 1][currentWeight]) {
26         System.out.println("Item " + i + " with weight " + weights[i - 1]);
27         currentWeight -= weights[i - 1];
28     }
29 }
30 }
31 public static void main(String[] args) {
32     int W = 50;
33     int[] weights = {10, 20, 30};
34     int[] values = {60, 100, 120};
35     int maxValue = knapsack(W, weights, values);
36     System.out.println("Maximum value that can be fit in the knapsack: " +
37         + "fit in the knapsack: " + maxValue);
38 }
39 }
40 }
```

Maximum value that can be fit in the knapsack: 220

## Task 2: Longest Common Subsequence

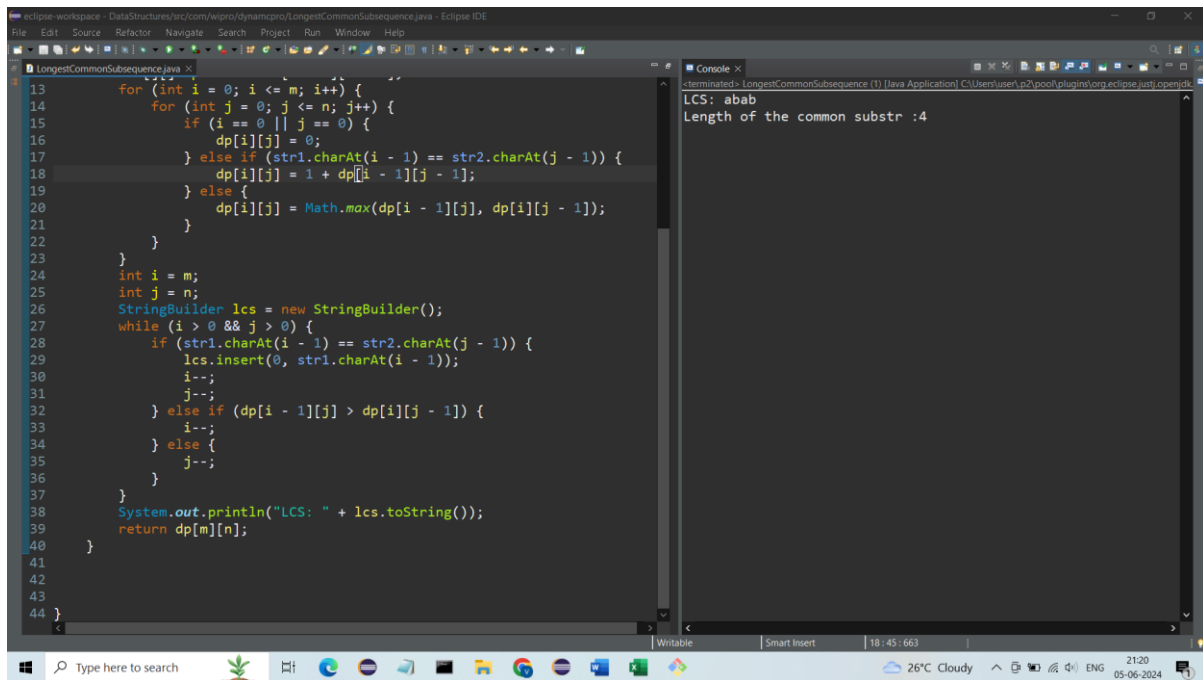
Implement int LCS(string text1, string text2) to find the length of the longest common subsequence between two strings.

Program: -



```
1 package com.wipro.dynamicpro;
2 public class LongestCommonSubsequence {
3     public static void main(String[] args) {
4         String str1 = "babbab";
5         String str2 = "abaaba";
6         int length = LongestCommonSubsequence(str1, str2);
7         System.out.println("Length of the common substr : " + length);
8     }
9     private static int longestCommonSubsequence(String str1, String str2) {
10         int m = str1.length();
11         int n = str2.length();
12         int[][] dp = new int[m + 1][n + 1];
13         for (int i = 0; i <= m; i++) {
14             for (int j = 0; j <= n; j++) {
15                 if (i == 0 || j == 0) {
16                     dp[i][j] = 0;
17                 } else if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
18                     dp[i][j] = 1 + dp[i - 1][j - 1];
19                 } else {
20                     dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
21                 }
22             }
23         }
24         int i = m;
25         int j = n;
26         StringBuilder lcs = new StringBuilder();
27         while (i > 0 && j > 0) {
28             if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
29                 lcs.insert(0, str1.charAt(i - 1));
30                 i--;
31                 j--;
32             } else if (dp[i - 1][j] > dp[i][j - 1]) {
```

## Output: -



```
13     for (int i = 0; i <= m; i++) {
14         for (int j = 0; j <= n; j++) {
15             if (i == 0 || j == 0) {
16                 dp[i][j] = 0;
17             } else if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
18                 dp[i][j] = 1 + dp[i - 1][j - 1];
19             } else {
20                 dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
21             }
22         }
23     }
24     int i = m;
25     int j = n;
26     StringBuilder lcs = new StringBuilder();
27     while (i > 0 && j > 0) {
28         if (str1.charAt(i - 1) == str2.charAt(j - 1)) {
29             lcs.insert(0, str1.charAt(i - 1));
30             i--;
31             j--;
32         } else if (dp[i - 1][j] > dp[i][j - 1]) {
33             i--;
34         } else {
35             j--;
36         }
37     }
38     System.out.println("LCS: " + lcs.toString());
39     return dp[m][n];
40 }
41
42
43
44 }
```

Console Output:

```
<terminated> LongestCommonSubsequence (1) [Java Application] C:\Users\user\p2\pool\plugins\org.eclipse.justi.openjdk
LCS: abab
Length of the common substr :4
```