Day 15 and 16:
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.

```java
package com.assignment.day15_16;

public class KnapsackProblem {

    public static void Knapsack(int W, int[] weights,
int[] values) {
        int n = weights.length;
        int[][] K = 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)
                    K[i][w] = 0;
                else if (weights[i - 1] <= w)
                    K[i][w] = Math.max(values[i - 1] +
K[i - 1][w - weights[i - 1]], K[i - 1][w]);
                else
                    K[i][w] = K[i - 1][w];
            }
        }

        int result = K[n][W];
        System.out.println("Maximum value that can be put
in a knapsack of capacity " + W + ": " + result);


        int w = W;
        for (int i = n; i > 0; i--) {
            if (result != K[i - 1][w]) {
                System.out.println("Item " + i + "
(weight: " + weights[i - 1] + ", value: " + values[i - 1]
+ ")");
                result -= values[i - 1];
                w -= weights[i - 1];
```

```
            }
        }

        System.out.println("Total weight of items
included: " + (W - w));
        System.out.println("Total value of items
included: " + K[n][W]);
    }

    public static void main(String[] args) {
        int W = 50;
        int[] weights = {10, 20, 30};
        int[] values = {60, 100, 120};
        Knapsack(W, weights, values);
    }
}
```

Output:

```
Problems  Javadoc  Declaration  Coverage  Console ×
<terminated> KnapsackProblem [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (04-Jun-2024, 9:46:15 pm – 9:46:16 pm) [p
Maximum value that can be put in a knapsack of capacity 50: 220
Item 3 (weight: 30, value: 120)
Item 2 (weight: 20, value: 100)
Total weight of items included: 50
Total value of items included: 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.

```
package com.assignment.day15_16;

public class LongestCommonSubsequence {
    private static int[][] dp;

    public static void main(String[] args) {
        String str1 = "babbab";
        String str2 = "abaaba";
```

```
        int length = longestCommonSubsequence(str1,
str2);
        System.out.println("Length of the common substr
:" + length);
        String lcs=getlongestCommonSubsequence(length,
str2,str1);
        System.out.println("Longest common Sequence :" +
lcs);

    }

    private static String
getlongestCommonSubsequence(int length, String str1,
String str2) {

            int m = str1.length();
            int n = str2.length();
            char[] lcs = new char[length];
            int index = length - 1;
            int i = m, j = n;
            while (i > 0 && j > 0) {
                if (str1.charAt(i - 1) ==
str2.charAt(j - 1)) {
                    lcs[index--] = str1.charAt(i - 1);
                    i--;
                    j--;
                } else if (dp[i - 1][j] > dp[i][j - 1])
{
                    i--;
                } else {
                    j--;
                }
            }
            return String.valueOf(lcs);
        }

    private static int longestCommonSubsequence(String
str1, String str2) {
        int m = str1.length();
        int n = str2.length();
        dp = new int[m + 1][n + 1];

        for (int i = 0; i <= m; i++) {
```

```
                for (int j = 0; j <= n; j++) {
                    if (i == 0 || j == 0) {

                        dp[i][j] = 0;
                    } else if (str1.charAt(i - 1) ==
str2.charAt(j - 1)) {
                        dp[i][j] = 1 + dp[i - 1][j - 1];
                    } else {
                        dp[i][j] = Math.max(dp[i - 1][j],
dp[i][j - 1]);
                    }
                }
            }
            return dp[m][n];
        }

}
```

Output:

```
17⊖     private static String getlongestCommonSubsequence(int l
18
```

Problems @ Javadoc Declaration Coverage Console ×

```
<terminated> LongestCommonSubsequence (1) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (04-Jun-202
Length of the common substr :4
Longest common Sequence :baba
```