```
/* Program 7
Implement 0/1 Knapsack problem using Dynamic Programming.
*/

import java.util.Scanner;
public class KnapsackDP {
        static final int MAX = 20;   // max. no. of objects
        static int w[];              // weights 0 to n-1
        static int p[];              // profits 0 to n-1
        static int n;                // no. of objects
        static int M;                // capacity of Knapsack
        static int V[][];                    // DP solution process - table
        static int Keep[][];         // to get objects in optimal solution

        public static void main(String args[]) {
                w = new int[MAX];
                p = new int[MAX];
                V = new int [MAX][MAX];
                Keep = new int[MAX][MAX];
                int optsoln;
                System.out.println("*****KNAPSACK USING DYNAMIC PROGRAMMING*****");
                ReadObjects();
                for (int i = 0; i <= M; i++)
                        V[0][i] = 0;
                for (int i = 0; i <= n; i++)
                        V[i][0] = 0;
                optsoln = Knapsack();
                System.out.println("Optimal solution (Maximum Profit) = " + optsoln);
        }

        static int Knapsack() {
                int r;                                  // remaining Knapsack capacity
                for (int i = 1; i <= n; i++)
                        for (int j = 0; j <= M; j++)
                                if ((w[i] <= j) && (p[i] + V[i - 1][j - w[i]] > V[i - 1][j])) {
                                        V[i][j] = p[i] + V[i - 1][j - w[i]];
                                        Keep[i][j] = 1;
                                } else {
                                        V[i][j] = V[i - 1][j];
                                        Keep[i][j] = 0;
                                }

                // Find the objects included in the Knapsack
                r = M;
                System.out.println("Items selected are = ");
```

```
                for (int i = n; i > 0; i--) // start from Keep[n,M]
                        if (Keep[i][r] == 1) {
                                System.out.println(i + " ");
                                r = r - w[i];
                        }
                System.out.println();
                return V[n][M];
        }

          static void ReadObjects() {
                Scanner scanner = new Scanner(System.in);
                System.out.println("Enter number of objects:  ");
                n = scanner.nextInt();
                System.out.println("Enter the max capacity of knapsack: ");
                M = scanner.nextInt();
                System.out.println("Enter Weights: ");
                for (int i = 1; i <= n; i++)
                        w[i] = scanner.nextInt();
                System.out.println("Enter Profits: ");
                for (int i = 1; i <= n; i++)
                        p[i] = scanner.nextInt();
                scanner.close();
          }
}
```

OUTPUT:

```
*****KNAPSACK USING DYNAMIC PROGRAMMING*****
Enter number of objects:
4
Enter the max capacity of knapsack:
15
Enter Weights:
9 3 2 5
Enter Profits:
20 30 25 45
Items selected are =
4
3
2

Optimal solution (Maximum Profit) = 100
```