Softcopy – Practical-4 (0/1 Knapsack)

Code

```
import java.util.*;

public class Code_A4 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] values = null;
        int[] weights = null;
        int n = 0;
        int choice;

        while (true) {
            System.out.println("\n=== 0-1 Knapsack Menu ===");
            System.out.println("1. Enter items (values & weights)");
            System.out.println("2. Solve 0-1 Knapsack");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter number of items: ");
                    n = sc.nextInt();
                    values = new int[n];
                    weights = new int[n];

                    for (int i = 0; i < n; i++) {
                        System.out.print("Item " + (i + 1) + " value: ");
                        values[i] = sc.nextInt();
                        System.out.print("Item " + (i + 1) + " weight:
");
                        weights[i] = sc.nextInt();
                    }
                    break;

                case 2:
                    if (values == null || weights == null || n == 0) {
                        System.out.println("Please enter items first!");
                        break;
                    }

                    System.out.print("Enter knapsack capacity: ");
                    int capacity = sc.nextInt();

                    int[][] dp = buildKnapsackDP(values, weights, n,
capacity);
                    System.out.println("\nDP Table:");
                    printDPTable(dp, n, capacity);

                    System.out.println("Maximum value that can be put in
knapsack: " + dp[n][capacity]);
                    break;

                case 3:
                    System.out.println("Exiting program. Goodbye!");
                    sc.close();
                    return;

                default:
                    System.out.println("Invalid choice!");
            }
        }
```

```
    }

    // Build DP table
    private static int[][] buildKnapsackDP(int[] val, int[] wt, int n,
int W) {
        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 (wt[i - 1] <= w)
                    dp[i][w] = Math.max(val[i - 1] + dp[i - 1][w - wt[i -
1]], dp[i - 1][w]);
                else
                    dp[i][w] = dp[i - 1][w];
            }
        }
        return dp;
    }

    // Print DP table
    private static void printDPTable(int[][] dp, int n, int W) {
        System.out.print("    ");
        for (int w = 0; w <= W; w++)
            System.out.printf("%4d", w);
        System.out.println();

        for (int i = 0; i <= n; i++) {
            System.out.printf("%2d ", i);
            for (int w = 0; w <= W; w++)
                System.out.printf("%4d", dp[i][w]);
            System.out.println();
        }
    }
}
```

Output

```
$ java Code_A4.java

=== 0-1 Knapsack Menu ===
1. Enter items (values & weights)
2. Solve 0-1 Knapsack
3. Exit
Enter your choice: 1
Enter number of items: 3
Item 1 value: 45
Item 1 weight: 32
Item 2 value: 6
Item 2 weight: 3
Item 3 value: 7
Item 3 weight: 3

=== 0-1 Knapsack Menu ===
1. Enter items (values & weights)
2. Solve 0-1 Knapsack
3. Exit
Enter your choice: 2
Enter knapsack capacity: 7

DP Table:
      0   1   2   3   4   5   6   7
 0    0   0   0   0   0   0   0   0
 1    0   0   0   0   0   0   0   0
 2    0   0   0   6   6   6   6   6
 3    0   0   0   7   7   7  13  13
Maximum value that can be put in knaps

=== 0-1 Knapsack Menu ===
1. Enter items (values & weights)
2. Solve 0-1 Knapsack
3. Exit
Enter your choice: 3
Exiting program. Goodbye!
```