Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Fall, Year: 2023), B.Sc. in CSE (Day)

Lab Report NO: 07

Course Title : Algorithm Lab
Course Code : CSE 204
Section : D - 9

Lab Experiment Name : Implement dynamic programming

Student Details

Name		ID
1.	Jahidul Islam	221002504

Lab Date : 04/12/2023
Submission Date : 12/12/2023
Course Teacher's Name : Md. Abu Rahman Refat

Lab Report Status

Marks: …… **Signature:.....**
Comments:..... **Date:.....**

1. TITLE:

Given a list of coins i.e 1 taka, 5 taka and 10 taka, can you determine the total number of combinations
of the coins in the given list to make up the number N taka?

2. OBJECTIVES/AIM:

Objective 1: Develop a dynamic programming solution for the coin change problem.

Objective 2: Take input From the user for the target amount.

Objective 3: Provide flexibility for users to input different coin denominations. By changing the array of coins.

Objective 4: Output the total number of combinations to make the target amount using the given coin denominations.

3. PROCEDURE:

3.1. User Input:

- The program begins by prompting the user to enter the target amount (N) they want to make change for.

3.2. Dynamic Programming Implementation:

- Utilizing a dynamic programming approach, the program calculates the total number of combinations to make change for the target amount.

```
int combinations = countCombinations(targetAmount, coins);
```

3.3. User-Friendly Coin Input:

- For simplicity, the program currently uses predefined coin denominations (coins = {1, 5, 10}). Users can customize this array as needed.

3.4. Output:

- Finally, the program prints the total number of combinations to the console.

3.5. Flexibility and Further Customization:

- Users can modify the program to use different coin denominations by updating the coins array.

- The program is designed to be flexible and can be integrated into larger systems requiring dynamic programming solutions for the coin change problem.

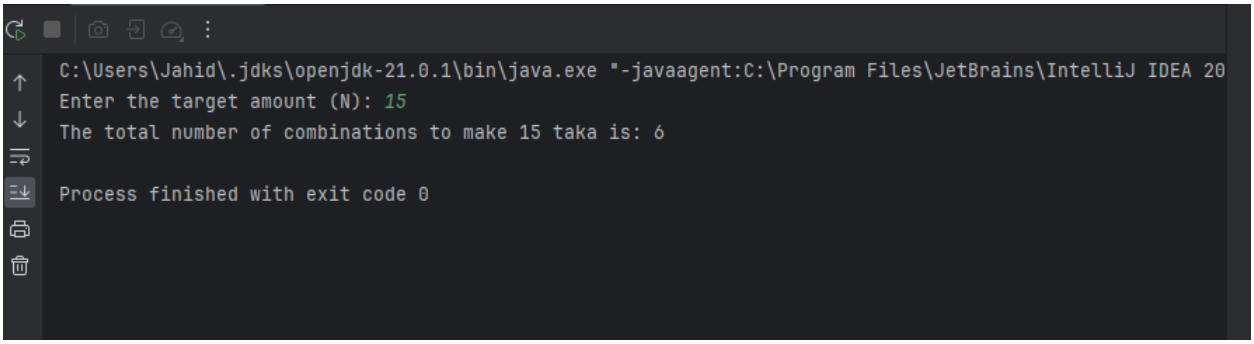
4. IMPLEMENTATION:

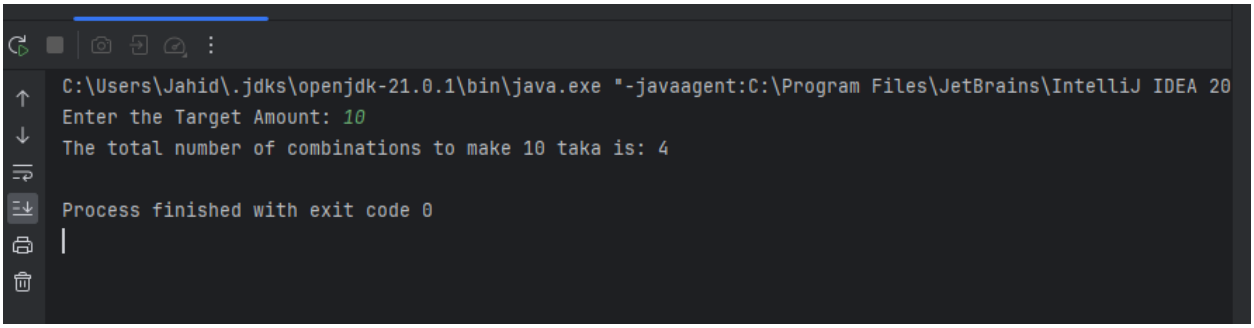
Code in Java:

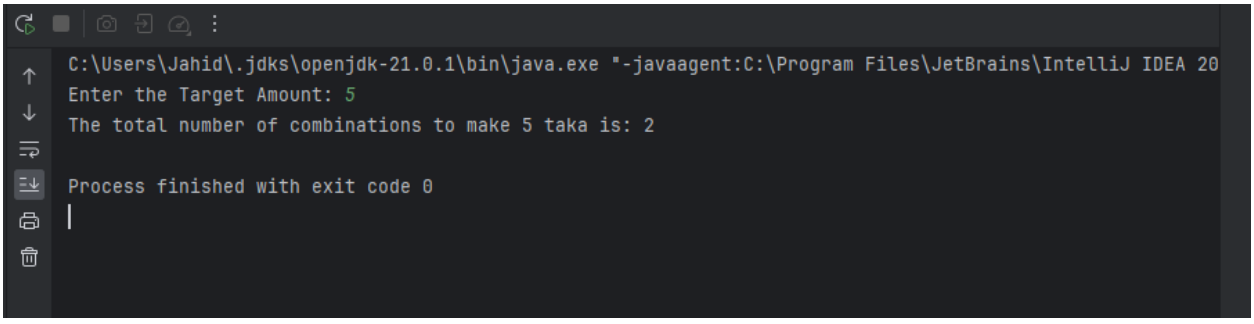
```
import java.util.*;
public class CoinChange {
    public static int countCombinations(int[] coins, int
target) {
        int[] dp = new int[target + 1];
        dp[0] = 1;

        for (int coin : coins) {
            for (int i = coin; i <= target; i++) {
                dp[i] += dp[i - coin];
            }
        }

        return dp[target];
    }

    public static void main(String[] args) {
        int[] coins = {1, 5, 10};

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the Target Amount: ");
        int targetAmount = sc.nextInt();
//        int targetAmount = 15;

        int combinations = countCombinations(coins,
targetAmount);
        System.out.println("The total number of combinations
to make " + targetAmount + " taka is: " + combinations);
    }
}
```

5. TEST RESULT / OUTPUT:

- Objective 1: CoinChange Combination







6. DISCUSSION AND ANALYSIS:

6.1. Dynamic Programming Solution:

The implementation employs a dynamic programming approach to efficiently calculate the total number of combinations for making change. This ensures optimal time complexity by avoiding redundant calculations and storing intermediate results in the dp array.

6.2. User Input Flexibility:

The program offers flexibility by allowing users to input the target amount and customize the coin denominations. This makes the solution adaptable to various scenarios where different denominations may be required.

6.3. Efficiency:

The algorithm's efficiency is notable, particularly for larger target amounts, as the dynamic programming technique optimizes the computation by breaking down the problem into smaller subproblems and reusing solutions.

6.4. Scalability:

The solution is scalable, and additional coin denominations can be easily incorporated. The program adapts to changes in the coin set without requiring extensive modifications.

7. CONCLUSION:

The program successfully addresses the coin change problem by employing dynamic programming. The flexibility of the solution makes it suitable for many applications where dynamic programming is necessary for efficient problem-solving.

Further improvements could involve adding additional features, such as handling different currency units or implementing a graphical user interface for enhanced user interaction.