**Securin Assessment Solution**

Name: Warish

Rollno. S20200010232

IIIT Sri City Chittor

Please find the github repository link of the assessment solution : [Link](https://github.com/mohdwarish483/Securin_Java_Challenge)

1. **Logic applied to solve the given assessment Problem**

**Part A: Calculations**

1. **calculateTotalCombinations(int diceASize, int diceBSize):**

Calculates the total number of possible combinations by multiplying the number of faces on each dice (6 \* 6 = 36).

**Possible Combinations of Dice A (diceAPossibleCombinations):**

This recursive function generates all possible combinations of Dice A. It takes an array of faces on Dice A, the desired length of combinations, a list to store the current combination, and a list to store all combinations. The base case is when the present list reaches the desired length.

**Possible Combinations of Dice B (diceBPossibleCombinations):**

Similar to Dice A, this recursive function generates all possible combinations of Dice B. It takes an array of faces on Dice B, the desired length of combinations, a starting index, a list to store the current combination, and a list to store all combinations.

1. **Combination Distribution (calculateCombinationDistribution):**

Creates a 6x6 matrix to represent all possible combinations of the two dice.

Iterates through each combination and calculates the sum of the corresponding faces on Dice A and Dice B.

Stores the sum in the corresponding cell of the matrix.

It then displays the distribution matrix (i, j) and the sum matrix.

1. **calculateProbabilityOfSums(diceA, diceB):**

Creates an array of size 13 to store the count of each possible sum (2-12).

Iterates through each combination and increments the count for the corresponding sum.

Calculates the probability of each sum by dividing its count by the total number of combinations.

Prints the probability of each sum.

**Part B : Undoom the Dice**

**undoomDice(List<Integer> diceA, List<Integer> diceB):**

This function undoes the transformation by generating all combinations of Dice A and Dice B and checking if the probability distribution matches the original distribution. If a match is found, it prints the transformed dice and probabilities.Ensuring Dice A does not have more than 4 spots on a face.Handles the condition where Die B can have more than 6 spots.

Validates that the overall probability distribution remains the same.

**Main Function:**

Initializes original dice values (Dice A and Dice B).

Prints the original dice values.

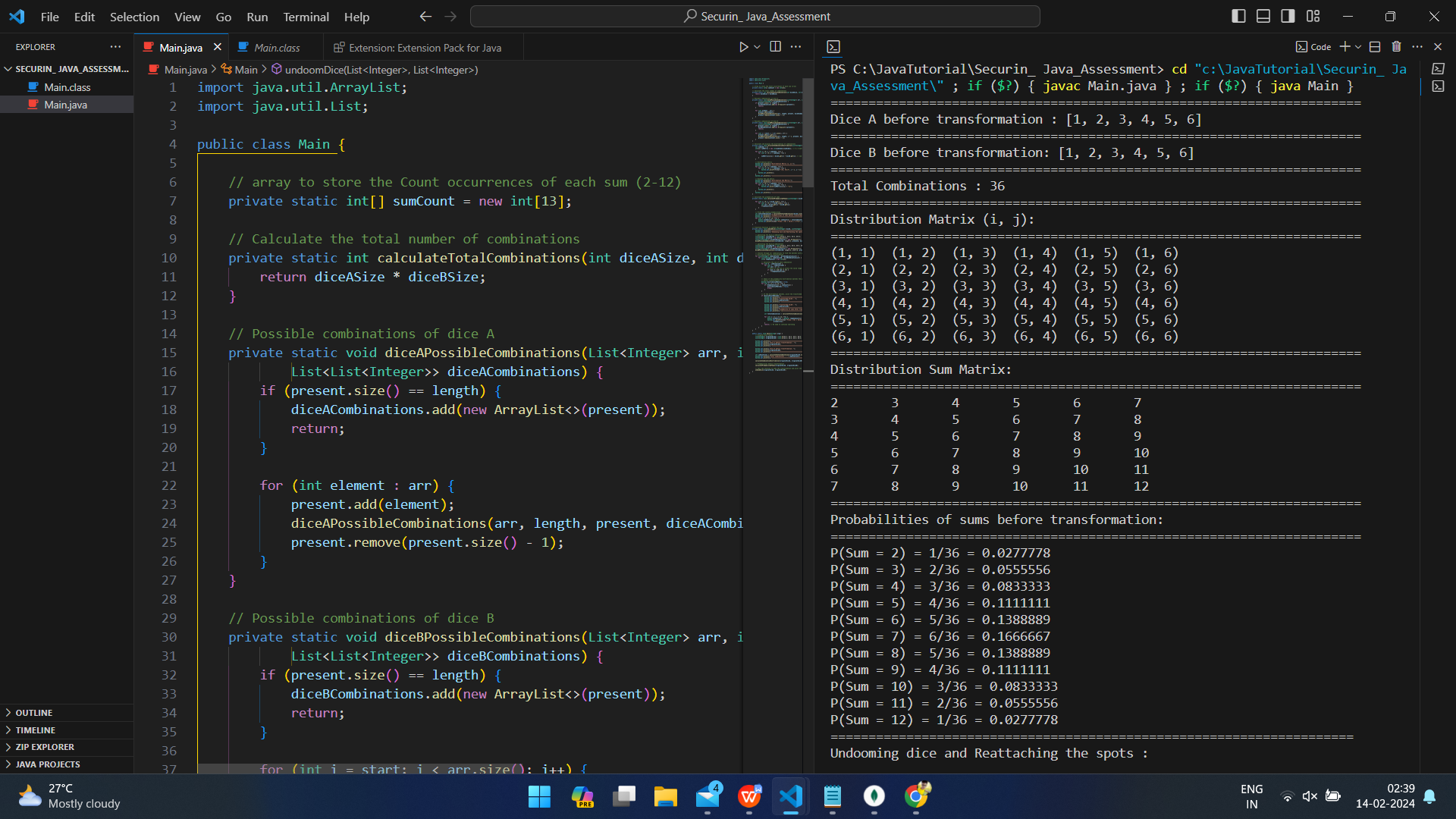
Calculates and prints the total combinations.

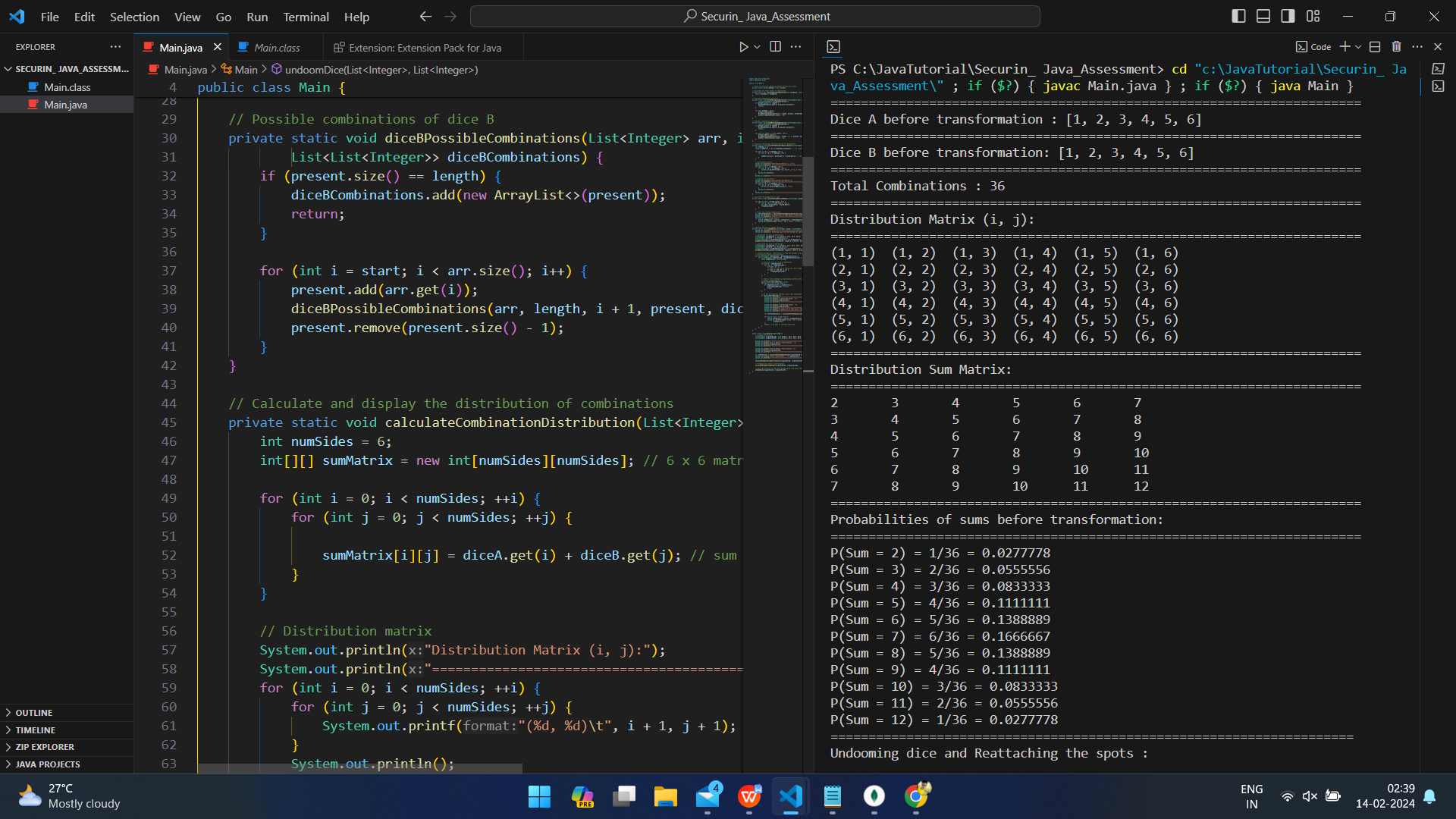
Calls the function to calculate and display the combination distribution.

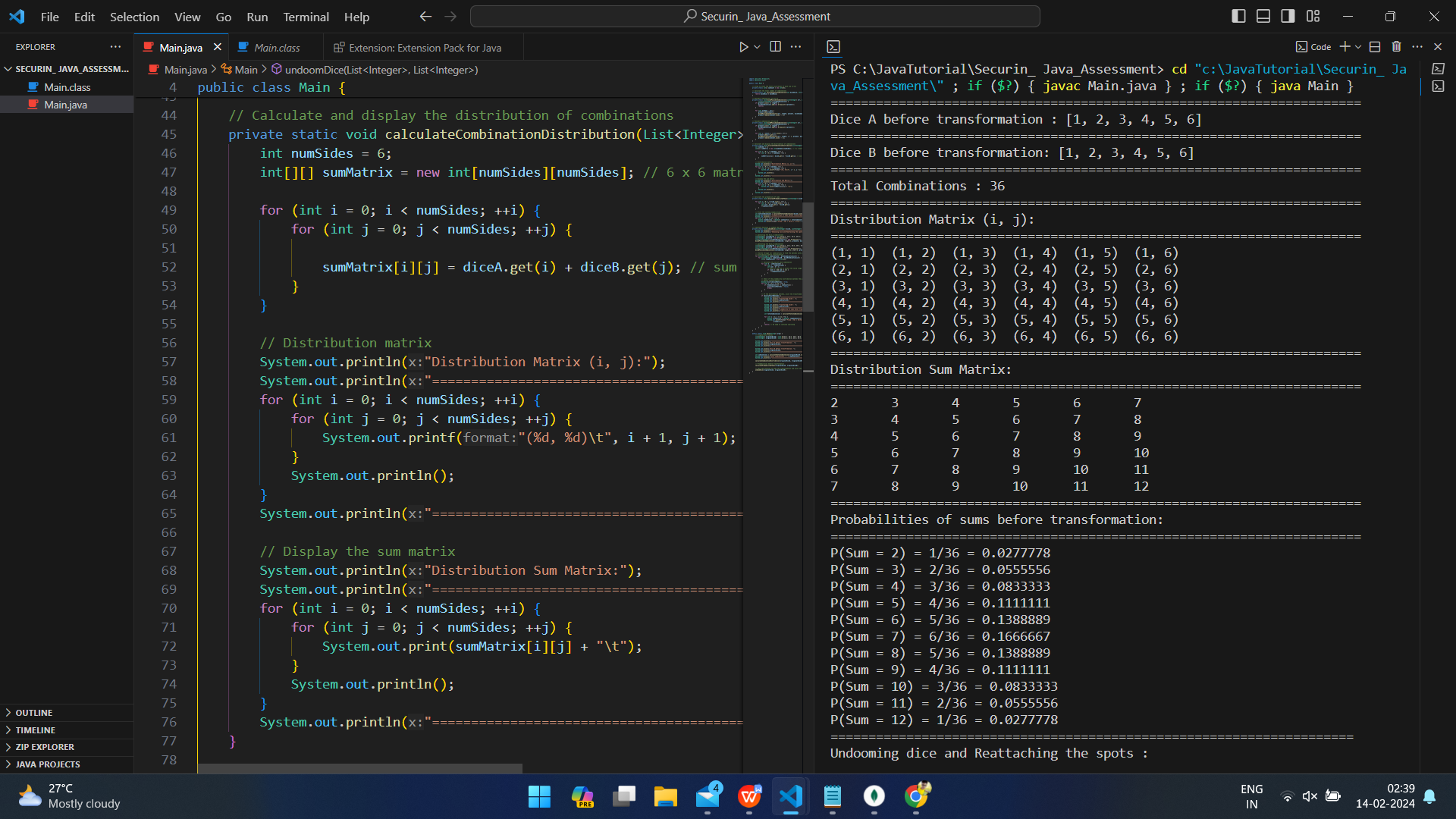
Calls the function to calculate and display the probabilities before transformation.

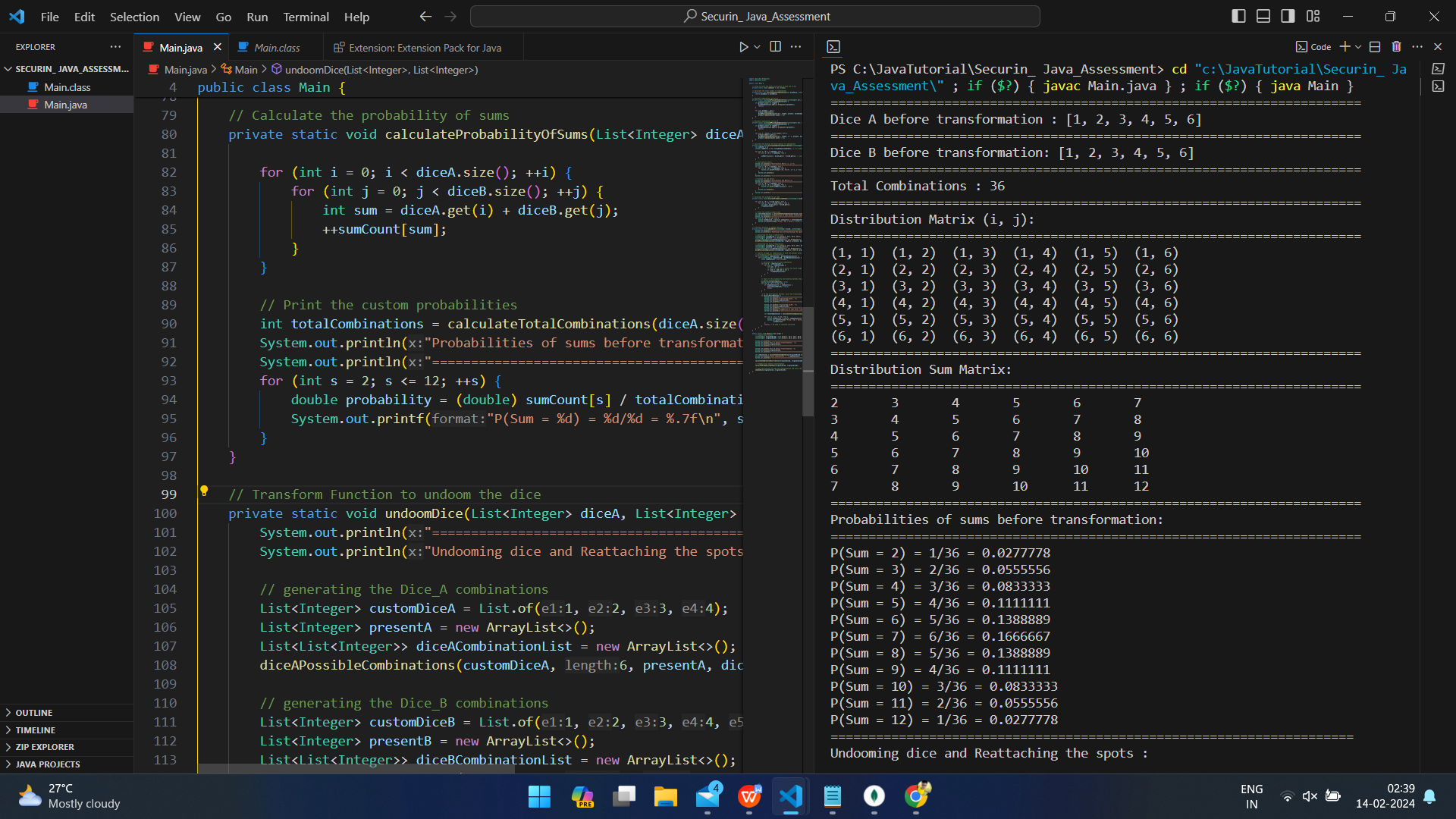
Calls the function to undoom the dice and prints the results.

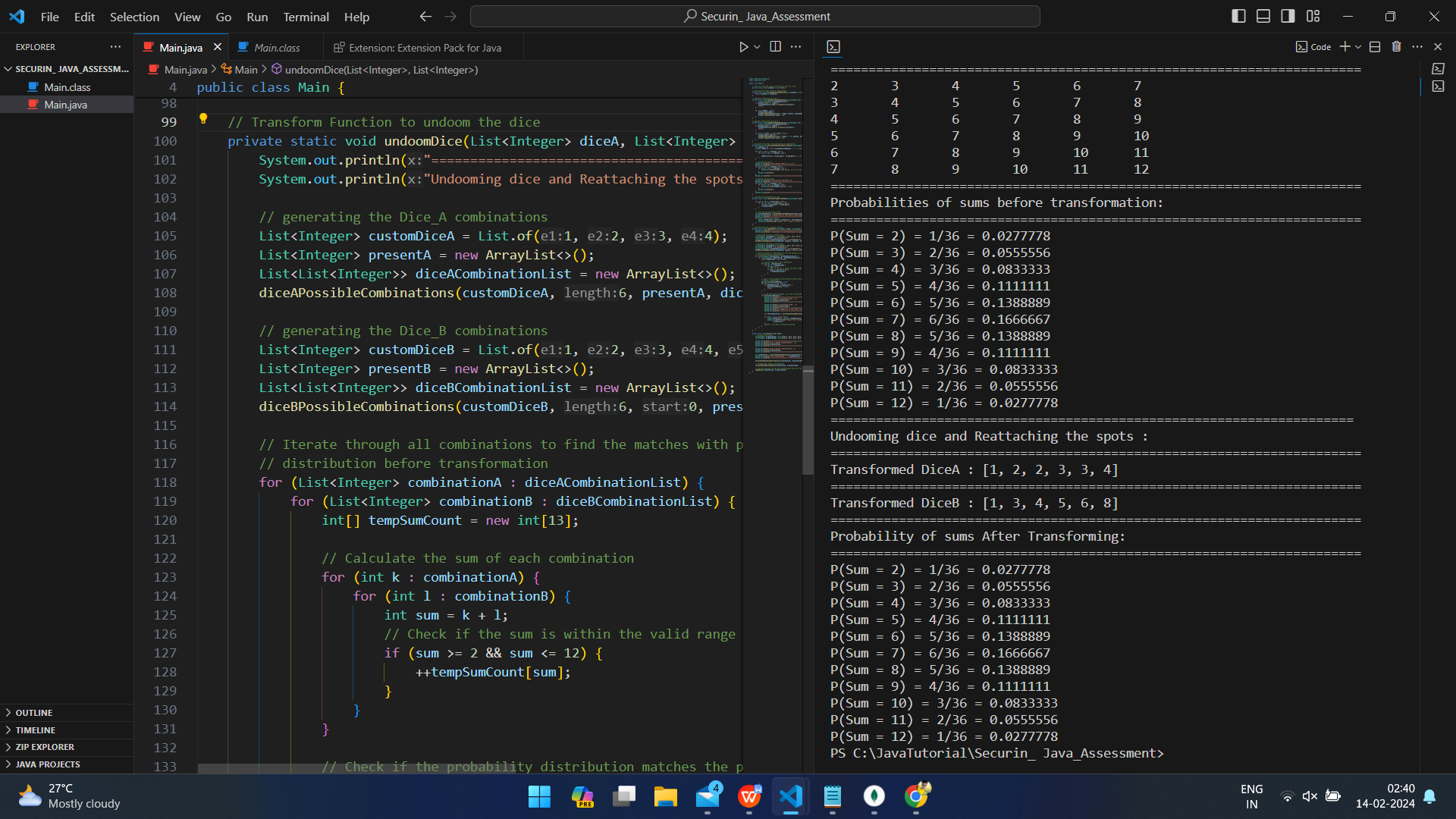
1. **Code and Output screenshot**

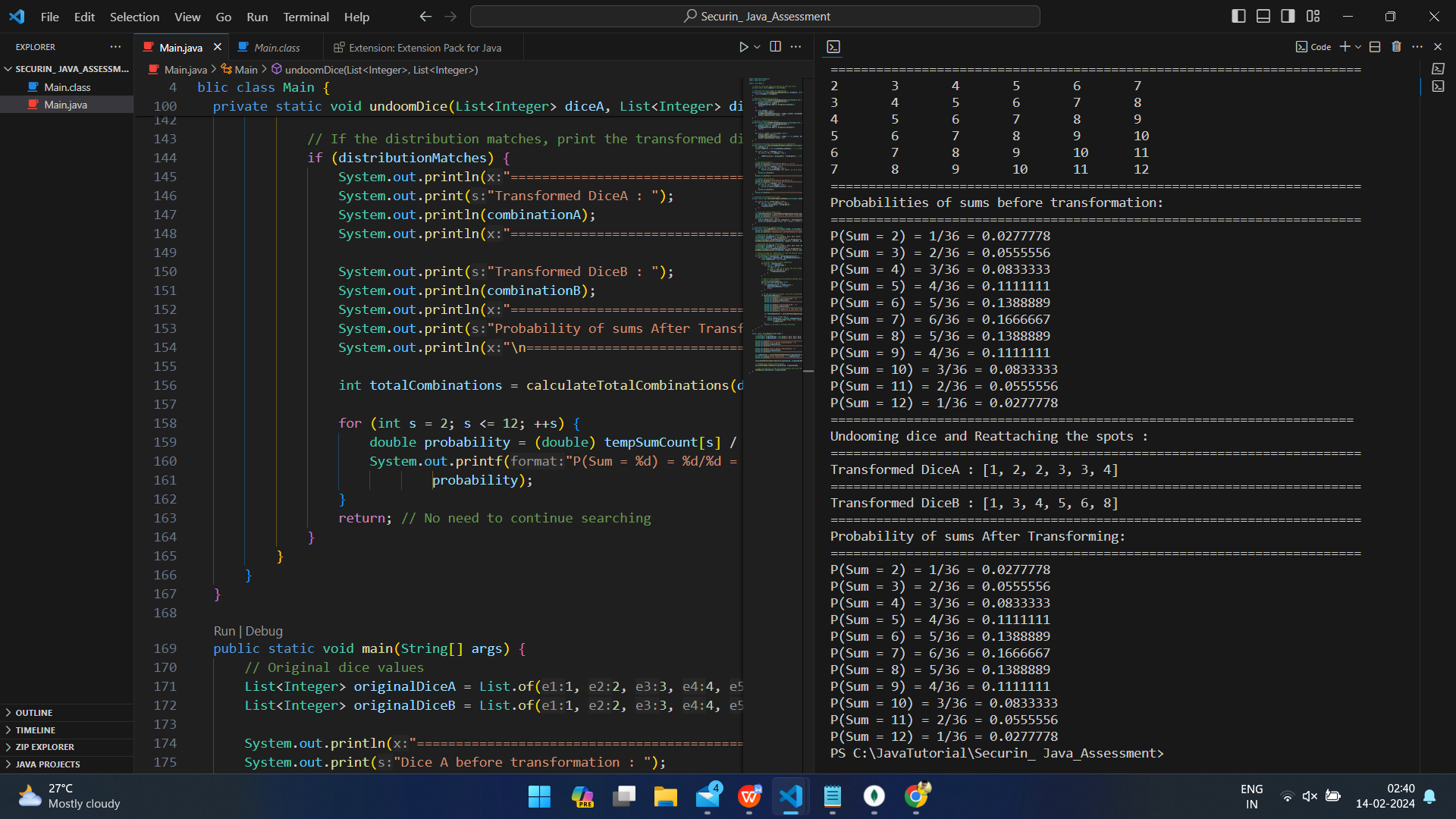
****

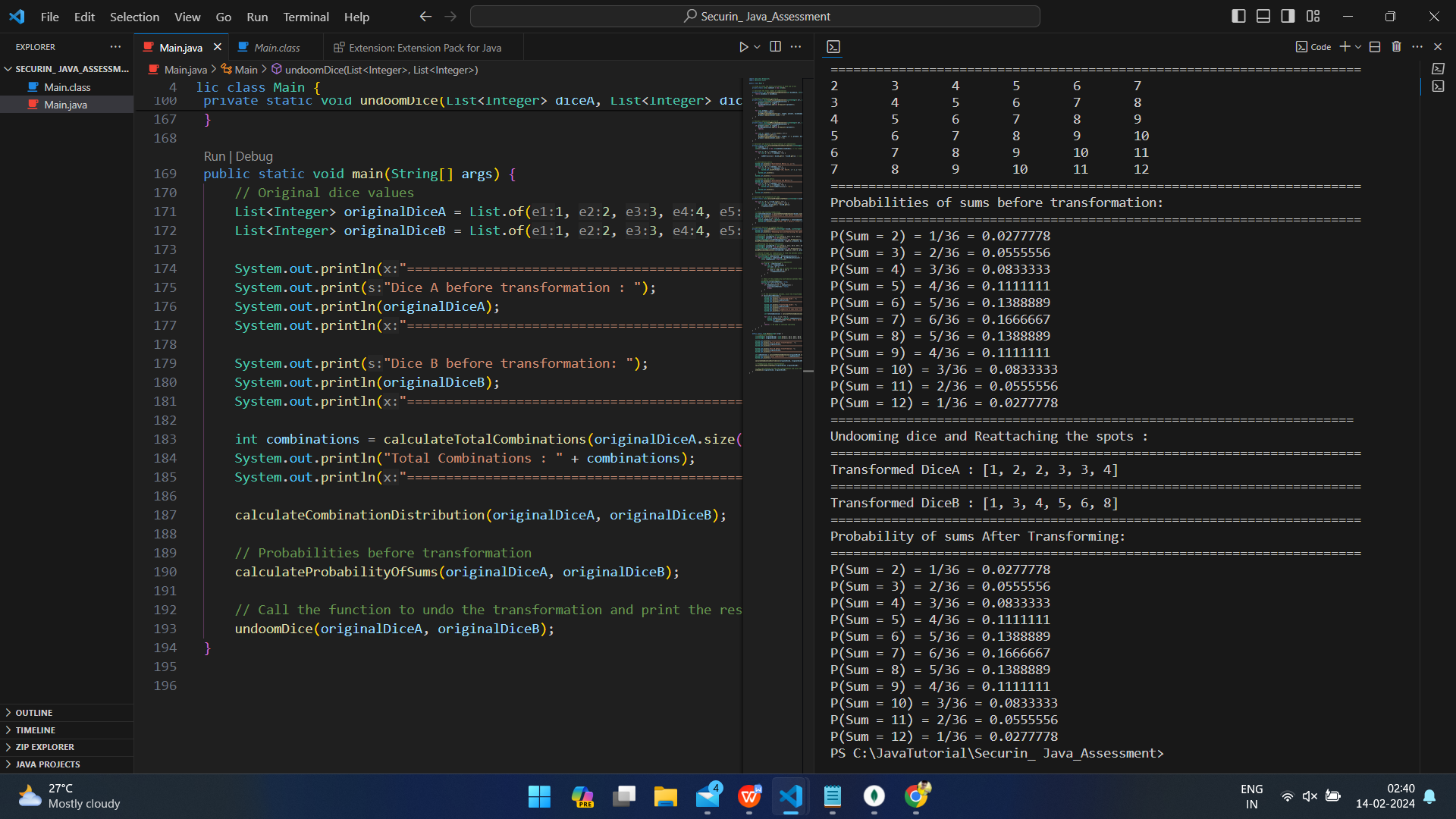
****

****

****

****

****

****

1. **Approach to the Solution:**

Given the Doomed Dice Challenge where Loki has tampered with the dice, removing all spots, the task is to reattach spots while adhering to specific constraints and maintaining the original probability distribution**.**

**The approach that I used to find the solution is as follows:**

1. **Generate Dice Combinations:**

Implement recursive functions (diceAPossibleCombinations and diceBPossibleCombinations) to generate all possible combinations for Die A and Die B, considering specific spot values for each die.

1. **Calculate Combination Distribution:**

Create a 6x6 matrix (sumMatrix) representing the sum of each combination of Die A and Die B. Display this distribution matrix to visualize all possible combinations.

1. **Calculate Probability of Sums:**

Iterate through all combinations of Die A and Die B, updating the sumCount array to store the occurrences of each sum. Print the probabilities of sums before any transformation.

1. **Undoom Dice Function:**

Create the undoomDice function to find new combinations of Die A and Die B that match the original probability distribution.

Generate combinations for custom dice A and B (customDiceA and customDiceB), considering constraints.

1. **Iterate through Combinations:**

For each combination of Die A and Die B, calculate the sum for each possible pair of faces.

Check if the probability distribution of sums matches the original distribution before any transformation.

1. **Output Results:**

Print the transformed Dice A and Dice B, ensuring that the probability distribution after reattachment of spots remains consistent with the original distribution.

1. **Validation:**

Validate the solution iteratively, checking each combination for matching probability distributions. If a match is found, output the transformed dice.

1. **Key Logic and Considerations:**
2. **Preservation of Probability:**

The focus is on preserving the probability distribution of sums, ensuring that the reattachment of spots does not alter the fairness of the game.

1. **Iterative Validation:**

The solution iteratively checks each combination, and if a match is found, it stops searching, providing the transformed dice and updated probabilities.

1. **Testing and Validation:**

Tested and validated for various combinations to ensure that the probability distribution of sums remains unchanged after the reattachment of spots.

**Thank You**