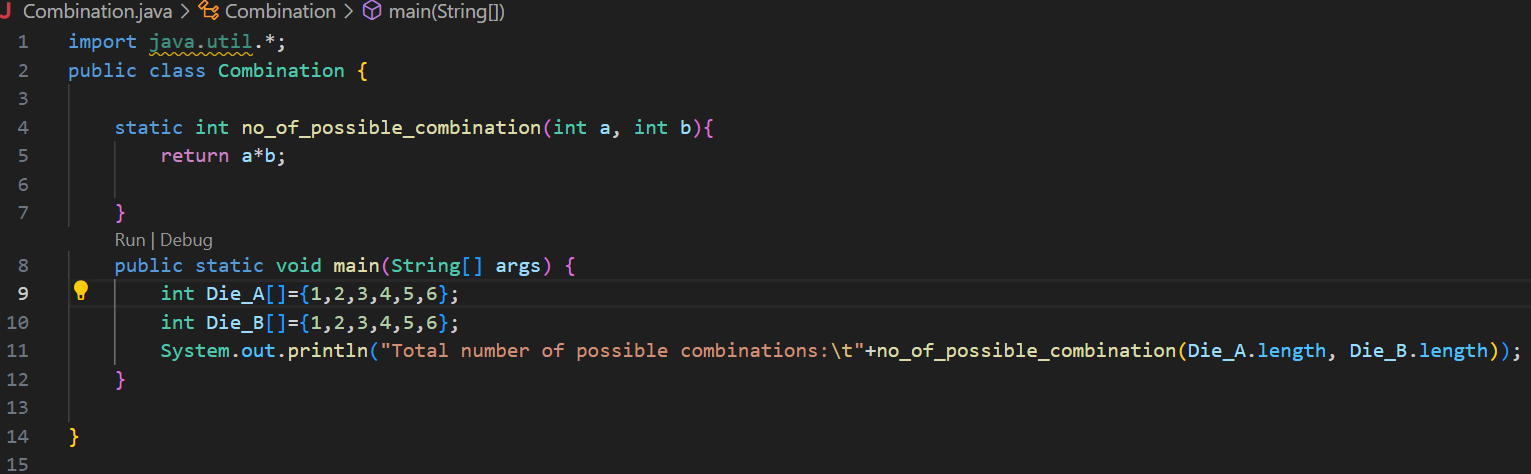
Securin – Project Submission (in Java)

Part A

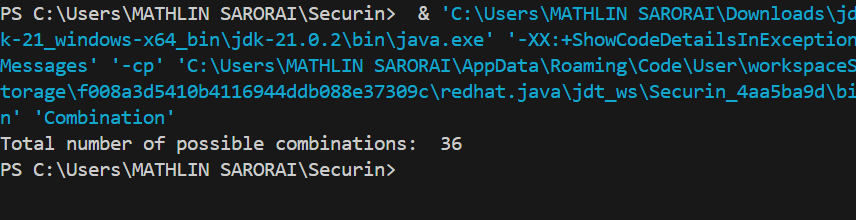
1. How many total combinations are possible? Show the math along with the code!

**Approach:**

To solve this programming challenge. My solution contains function that takes the 2 arrays. i.e Die\_A and Die\_B. And finds the product of the length of 2 arrays and return the same.

****

**Output:**

****

1. Calculate and display the distribution of all possible combinations that can be obtained when rolling both Die A and Die B together. Show the math along with

the code! Hint: A 6 x 6 Matrix.

**Approach:**

To solve this programming challenge. My solution contains a function that takes both the arrays. i.e. Die\_A array and Die\_B array that contains the faces of each of them.

I have created a 2d array that is sized to store all 36 combinations sum possible.

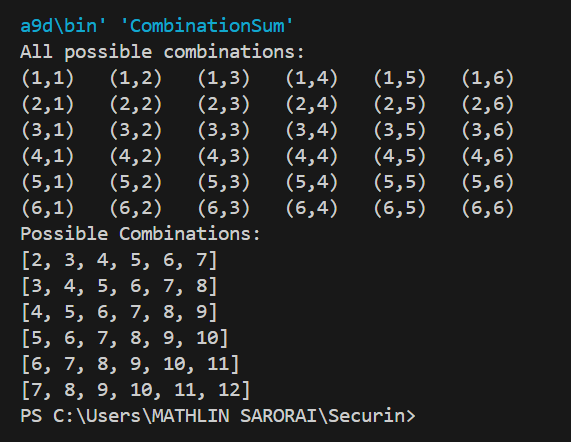
To find the combination sum. I used 2 loops to iterate through each element of array A and array B. which is added and stored in the 2D array.

Using the same loops without occupying excess time complexity I print the possible pair of dices.

After complete execution of the for loops. The print function is called that is programmed to print the values in screen.



Output:



1. Calculate the Probability of all Possible Sums occurring among the number of combinations from (2). Example: P(Sum = 2) = 1/X as there is only one combination possible to obtain Sum = 2. Die A = Die B = 1.

**Approach:**

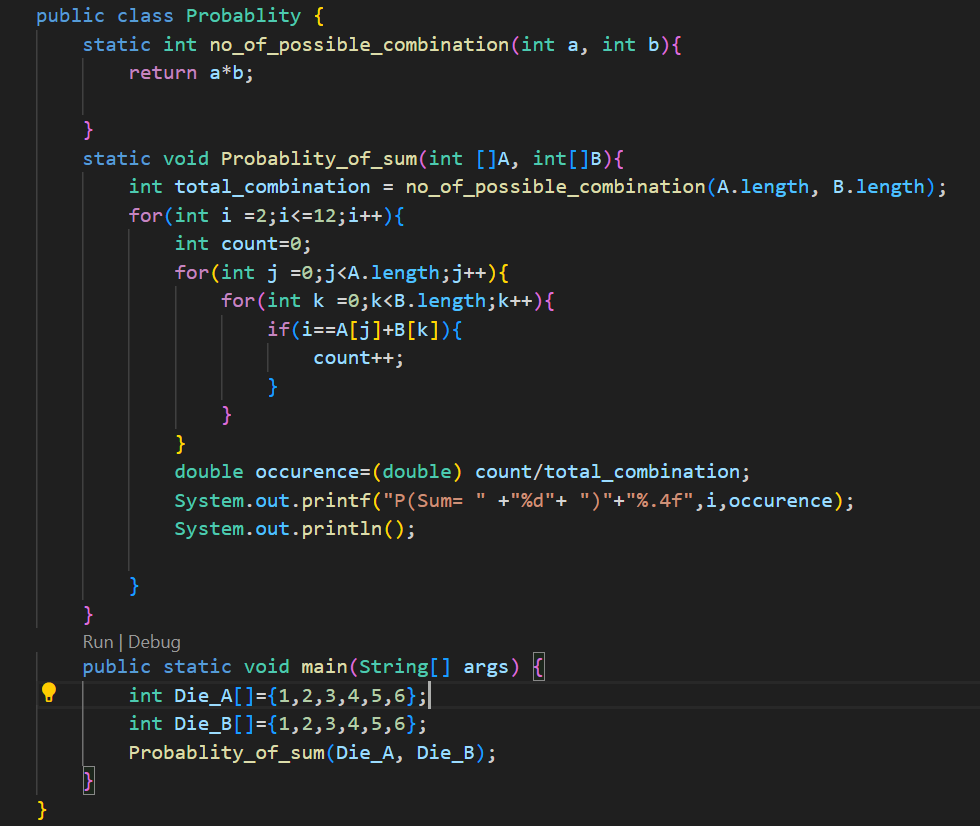
To solve this programming question my solution contains 2 functions. No\_of\_possible\_combinations are used to find the total possible combinations count. And the probability\_of\_sum function finds the probability.

Here as the range of the sum is known from 2 – 12 (both inclusive).

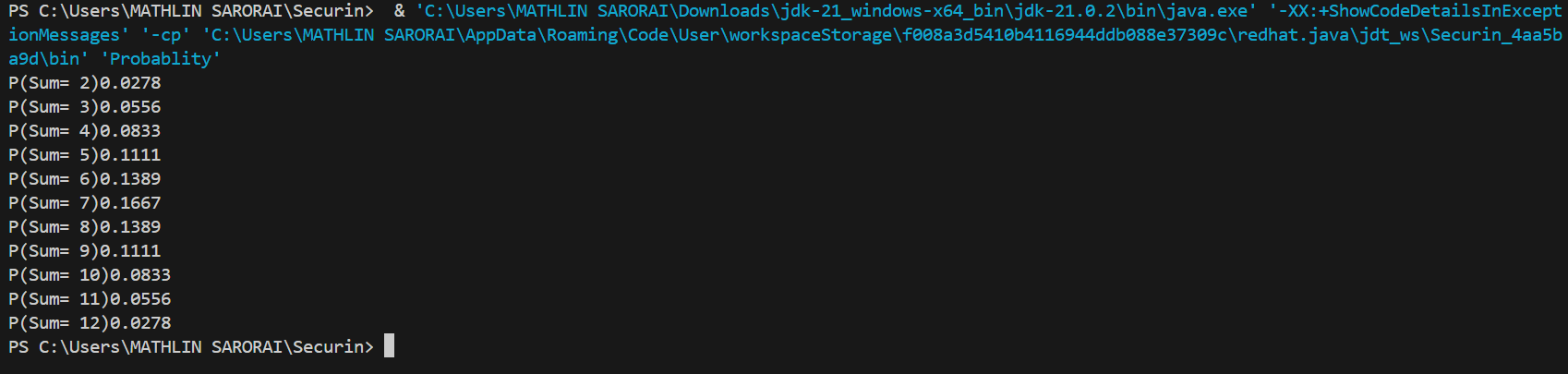
My solution uses nested loops to tackle the solution. Wherein the first loop iterates through the possible sums. The second and third loop iterates through the arrays of Die A and Die B and sums up each combination.

An if condition is used to check the individual count of occurrence of each sum. And the probability is found by dividing the number of occurrences(count) by total combination possible (36).

Likewise, it found for all individual sums existing and its printed.

****

Output:



Part B

You were happily spending a lazy afternoon playing your board game with your dice when suddenly the mischievous Norse God Loki ( You love Thor too much & Loki didn’t like that much ) appeared. Loki dooms your dice for his fun removing all the “Spots” off the dice. No problem! You have the tools to re-attach the “Spots” back on the Dice.

However, Loki has doomed your dice with the following conditions:

● Die A cannot have more than 4 Spots on a face.

● Die A may have multiple faces with the same number of spots.

● Die B can have as many spots on a face as necessary i.e. even more than 6.

But to play your game, the probability of obtaining the Sums must remain the

same!

So if you could only roll P(Sum = 2) = 1/X, the new dice must have the spots reattached

such that those probabilities are not changed.

Input:

● Die\_A = [1, 2, 3, 4, 5, 6] & Die B = Die\_A = [1, 2, 3, 4, 5, 6]

Output:

● A Transform Function undoom\_dice that takes (Die\_A, Die\_B) as input &

outputs New\_Die\_A = [?, ?, ?, ?, ?, ?],New\_Die\_B = [?, ?,

?, ?, ?, ?] where,

● No New\_Die A[x] > 4

**Approach:**

**To solve this question, I have used Python as my codebase as java being compiler dependent fetches different outputs on different platform.**

The solution uses 3 User-defined function.

Firstly, function to find the probabilities of occurrences of all possible sums when rolling two dice.

My solution uses itertools.product and counts the occurrences of each sum using a dictionary. Later normalization is done by dividing each of them with total number of rolls.

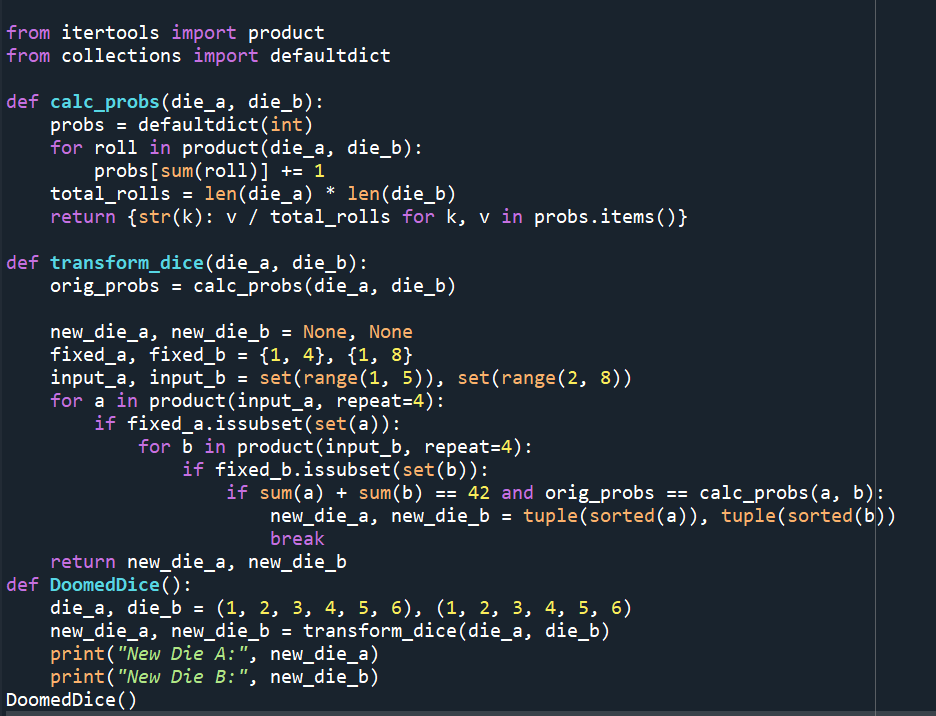
The majority party of code lies in transform\_dice()user-defined function. It first calls the calculate\_probs() function then initializes variables to store new dice configurations.

Transform\_dice() function finds the new configurations while maintaining same sum probabilities.

It calculates original probabilities by calling calc\_probs() and defines few fixed values. Alsso assigns the input values of die\_a and die\_b.

Iterates through all possible combinations of values of die\_a and die\_b.

For each combination it checks if the values are included in the combination and if the sum of the new configuration is equal to 42. And checks the probabilities comparing with original probabilities. If all conditions are set then the new configuration match the original probabilities.

****

Output:

