**PART-A**

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

Logic:

The main objective is to find the total number of combinations possible when rolling both dice.

Die\_A = {1,2,3,4,5,6}

Die\_B = {1,2,3,4,5,6}

6\*6 = 36

Approach:

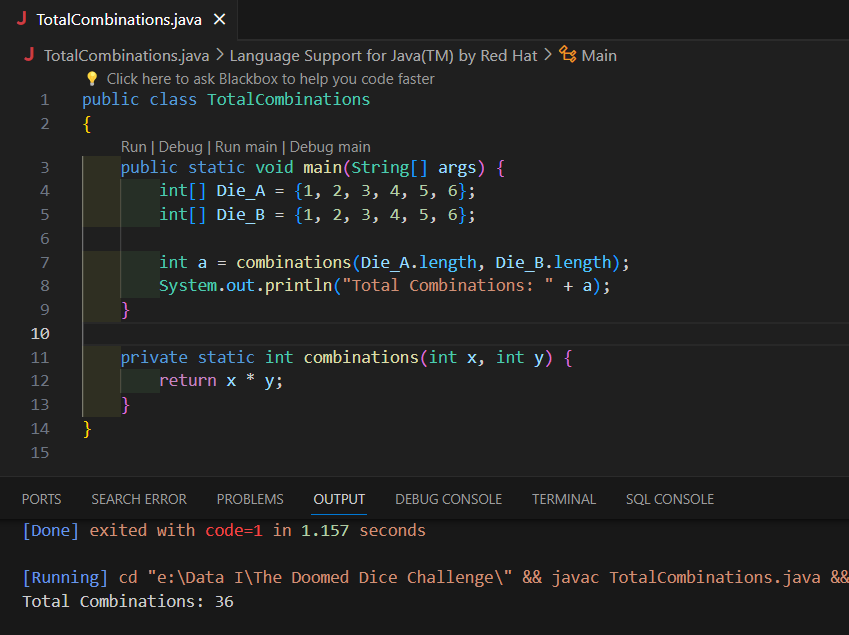
There are 1 to 6 faces in each of the two dice. The total possible combinations:

calculate(Die\_A.length, Die\_B.length) —> Die\_A=6, Die\_B=6

Die\_A \* Die\_B = 36

Code:

<https://github.com/Haritha-Sivasankaran/Doomed-Dice-Challenge/blob/main/The%20Doomed%20Dice%20Challenge/TotalCombinations.java>



**2. 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.**

Logic:

The main objective is to find the Possible Probability Distribution(Combinations) in the two dice

Die\_A = {1,2,3,4,5,6}

Die\_B = {1,2,3,4,5,6}

i j

(1,1) (1,2) (1,3) (1,4) (1,5) (1,6)

(2,1) (2,2) (2,3) (2,4) (2,5) (2,6)

(3,1) (3,2) (3,3) (3,4) (3,5) (3,6)

(4,1) (4,2) (4,3) (4,4) (4,5) (4,6)

(5,1) (5,2) (5,3) (5,4) (5,5) (5,6)

(6,1) (6,2) (6,3) (6,4) (6,5) (6,6)

We have to sum up all the possible combinations in each row

Approach:

Row 1 (Die\_A[0] + Die\_B[j] for all j): [1+1, 1+2, 1+3, 1+4, 1+5, 1+6] = [2, 3, 4, 5, 6, 7]

Row 2 (Die\_A[1] + Die\_B[j] for all j): [2+1, 2+2, 2+3, 2+4, 2+5, 2+6] = [3, 4, 5, 6, 7, 8]

Row 3 (Die\_A[2] + Die\_B[j] for all j): [3+1, 3+2, 3+3, 3+4, 3+5, 3+6] = [4, 5, 6, 7, 8, 9]

Row 4 (Die\_A[3] + Die\_B[j] for all j): [4+1, 4+2, 4+3, 4+4, 4+5, 4+6] = [5, 6, 7, 8, 9, 10]

Row 5 (Die\_A[4] + Die\_B[j] for all j): [5+1, 5+2, 5+3, 5+4, 5+5, 5+6] = [6, 7, 8, 9, 10, 11]

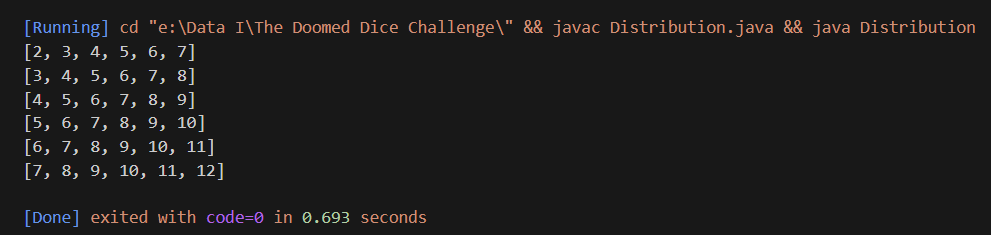
Row 6 (Die\_A[5] + Die\_B[j] for all j): [6+1, 6+2, 6+3, 6+4, 6+5, 6+6] = [7, 8, 9, 10, 11, 12]

It will be processed in Nested loops where the main condition comes inside the loop is

Die\_A[i] + Die\_B[j] which is broken down into both ith and jth rows and adds up the possible combinations.

Code:

<https://github.com/Haritha-Sivasankaran/Doomed-Dice-Challenge/blob/main/The%20Doomed%20Dice%20Challenge/Distribution.java>



**3. 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.**

Logic:

Possible combination Distributions:

[2, 3, 4, 5, 6, 7]

[3, 4, 5, 6, 7, 8]

[4, 5, 6, 7, 8, 9]

[5, 6, 7, 8, 9, 10]

[6, 7, 8, 9, 10, 11]

[7, 8, 9, 10, 11, 12]

Total Combinations = 36

The above matrix is the combination of all possible rows in the distribution of two dice

Now we have to find the sum of all possible combinations from the above matrix

P(Sum=X) = Possible pairs/total combinations

P(Sum=2) = 1/36

P(Sum=3) = 1/18

P(Sum=4) = 1/12

P(Sum=5) = 1/9

P(Sum=6) = 5/36

P(Sum=7) = 1/6

P(Sum=8) = 5/36

P(Sum=9) = 1/9

P(Sum=10) = 1/12

P(Sum=11) = 1/18

P(Sum=12) = 1/36

Approach:

The calculation of the probability of the sum of all possible combinations can be done in the nested loop in an optimized way

prob[Matrix[i][j] - 2] += 1.0 / totalcombinations

This was the main part of the code which helps to calculate the probability of the sum of all combinations possible where 1.0/total combinations are used to calculate the probability and -2 is used for to start the index of array[i][j] from 0th index.

P(Sum=2) = 0.0278

P(Sum=3) = 0.0556

P(Sum=4) = 0.0833

P(Sum=5) = 0.1111

P(Sum=6) = 0.1389

P(Sum=7) = 0.1667

P(Sum=8) = 0.1389

P(Sum=9) = 0.1111

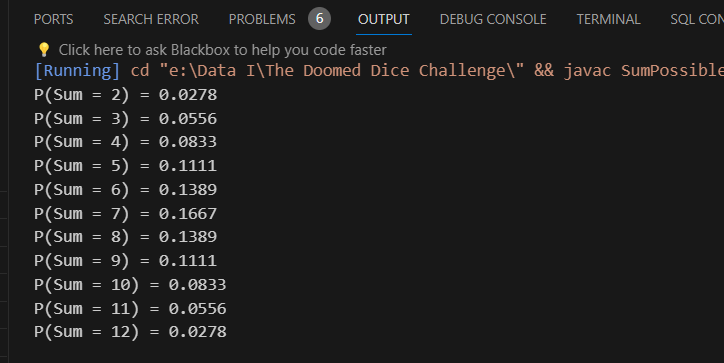
P(Sum=10) = 0.0833

P(Sum=11) = 0.0556

P(Sum=12) = 0.0278

Code:

<https://github.com/Haritha-Sivasankaran/Doomed-Dice-Challenge/blob/main/The%20Doomed%20Dice%20Challenge/SumPossibleCombinations.java>



**PART-B**

**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 in order 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**

Logic:

The task is to write a function undoom\_dice that takes the original Die A and Die B as input and outputs the new Die A and Die B satisfying the above conditions. The function should return the new configuration of the dice that maintains the same sum probabilities as the original dice.

The logic of the problem statement lies in understanding the probabilities of the sums of two dice and finding a way to distribute the spots on the new dice such that these probabilities remain unchanged, while also adhering to Loki’s conditions.

This involves concepts from probability theory and combinatorics.

Approach:

The key part of the problem is to maintain the same sum probabilities. This means that the probability of obtaining a certain sum when rolling the two dice should remain the same before and after reattaching the spots.   
  
So first we have to transform the die\_A :

The method iterates over the faces of Die A. For each face, it assigns the minimum of the original number of spots and 4 to the corresponding face on newDieA. This ensures that newDieA satisfies the condition that no face has more than 4 spots.

for (int i = 0; i < 6; i++) {

newDieA[i] = Math.min(4, dieA[i]);

}

newDie\_A = {1,2,3,4,4,4}

Now we arrange the total spots in Die\_B:  
 It calculates the total number of spots on Die B and a scale factor, which is the ratio of the total spots on Die B to 15 (the sum of spots on Die A if each face has at most 4 spots).

int totalSpotsB = Arrays.stream(dieB).sum();

double scaleFactor = (double) totalSpotsB / 15;  
  
We have to transform the die\_B:

The method iterates over the faces of Die B. For each face, it assigns the rounded result of the original number of spots divided by the scale factor to the corresponding face on newDieB. This attempts to maintain the same total number of spots on newDieB as on Die B.

for (int i = 0; i < 6; i++) {

newDieB[i] = (int) Math.round(dieB[i] / scaleFactor);

}

newDie\_B= [1, 1, 2, 3, 4, 4]

Code:

<https://github.com/Haritha-Sivasankaran/Doomed-Dice-Challenge/blob/main/The%20Doomed%20Dice%20Challenge/Lokidoomdice.java>

