**Problem Statement – The Doomed Dice Challenge**

**Overview:**

**Part – A**

**\*The problem has been solved using JAVA**

We are given two six-sided dice, Die A and Die B, each with faces numbered from 1 to 6. We can only roll both the dice together & our turn is guided by the obtained sum. Example: Die A = 6, Die B = 3. Sum = 6 + 3 = 9.

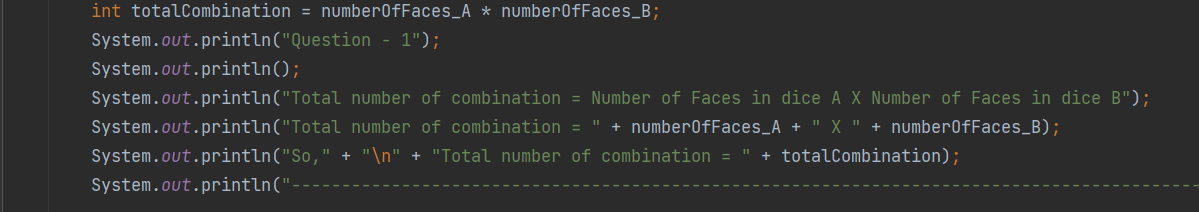
We may represent Dice as an Array or Array-like structure. Die A = [1, 2, 3, 4, 5, 6] where the indices represent the 6 faces of the die & the value on each face.

Import the necessary packages and create the main function in which the program starts. With the given details, we have set the number faces for Die A and Die B as 6.

**Question**

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

The question is asked about the total number of combinations that are possible with the two dices given, for which we have found the answer by multiplying the number of faces in each die.

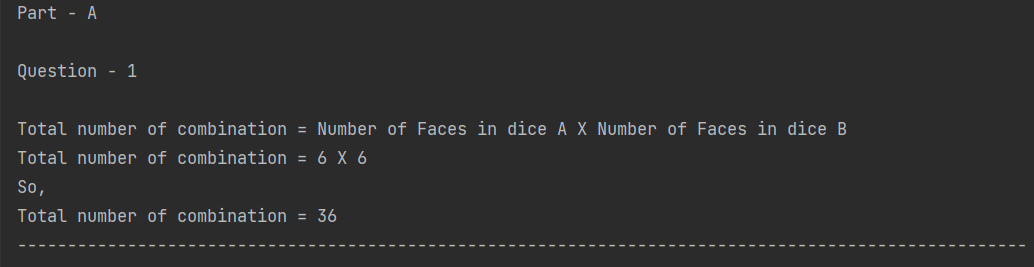
****

Total number of combination

= Number of faces of Die A X Number of faces of Die B

=36

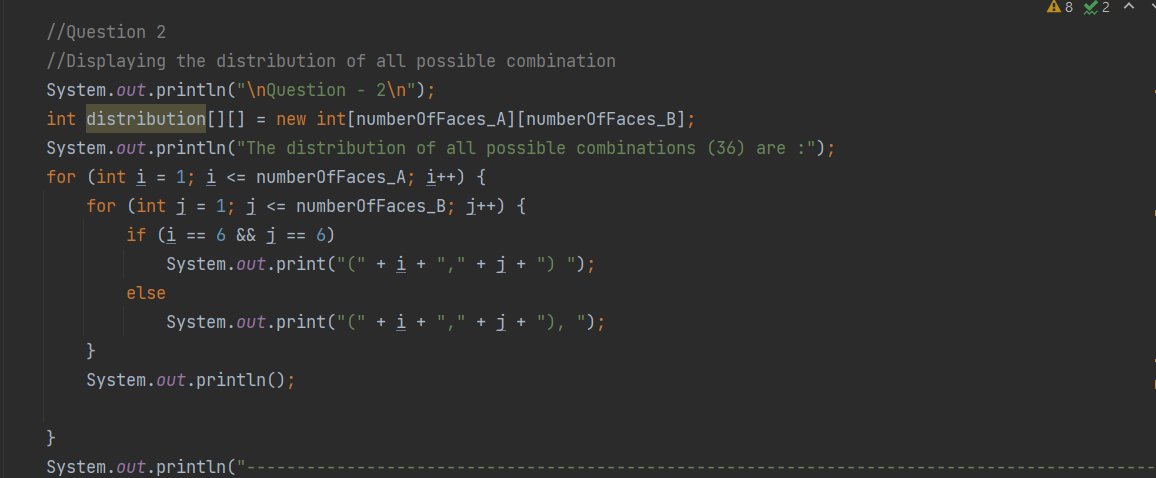
**OUTPUT :**

****

**Question**

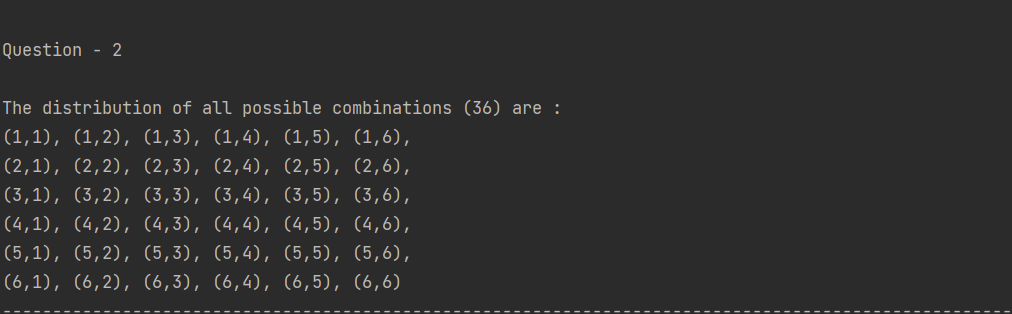
1. **Calculate and display the distribution of all possible combinations that can be obtained when rolling both Die A and Die B together. Show the math along with the code!**

They have asked us to display the distribution of of all the possible combinations. As we know there are (6 X 6) 36 total combinations.

****

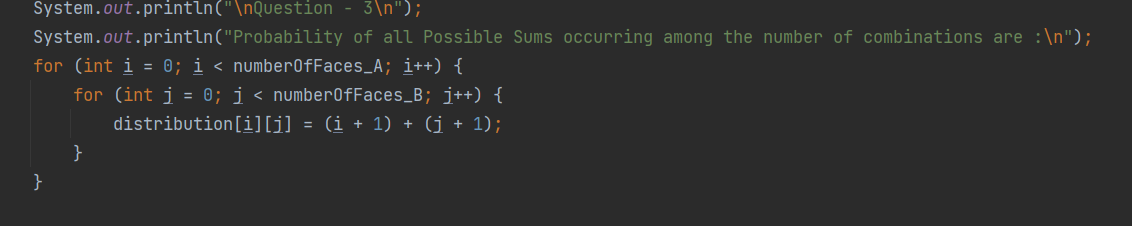
Now let us traverse this as a matrix as keeping Die A faces to be the limit of i for loop and Faces of Die B as the limit of j for loop. Within the loops, print i and j every time its being traversed . If we do so, for 36 times, the loop being traversed, we get the pairs like :

**OUTPUT :**



**Question**

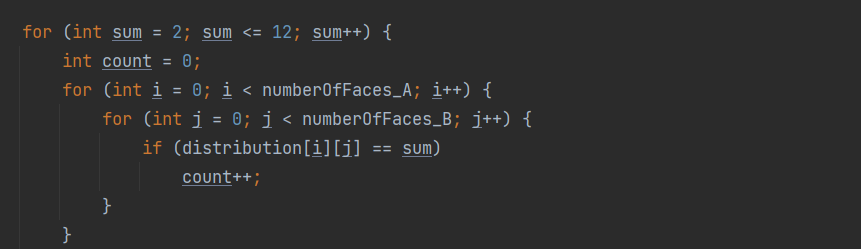
1. **Calculate the Probability of all Possible Sums occurring among the number of combinations from (2).**

****They have asked us to find about the probability of all possible sums occurring among the different combinations starting right from 2.

To achieve this,

We know as there are 36 different combinations, there will 36 sums generated with all possible pairs, this includes the repeating sums also.  
We will run a matrix loop to take the values of I and j and add them up together to find the total sum. Here we use(i+1) and (j+1) as the loop here has started with 0 index (For die, there is no 0 face).

Now these sums could be stored in a matrix array distribution[][].



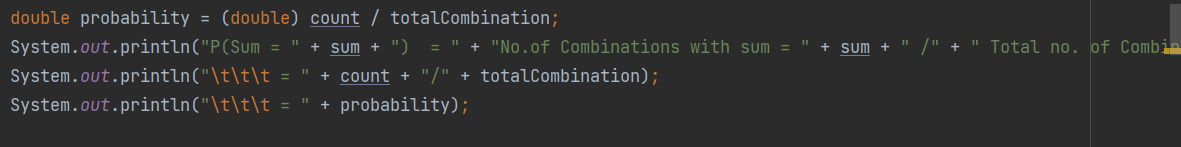
We know the total sum varies from 2 to 12 because the min possibility Is(1,1) and maximum possibility is (6,6).

Now we have to run again two loops to find how many time each sum has repeated. This will be used to easily find the probability distribution.

The condition we used here is:

If (distribution matrix elements is equal to the sum value)

If so then count value is incremented. For every value of sum, a separate count will be produced that says how many times the sum has been arrived by summing up the distribution pairs.



We know the total combinations are 36 outcomes. So for finding each sum value probability we have to divide count value(No.of combinations with each sum) by total combination (Total number of combinations).

i.e.,

Probability outcome

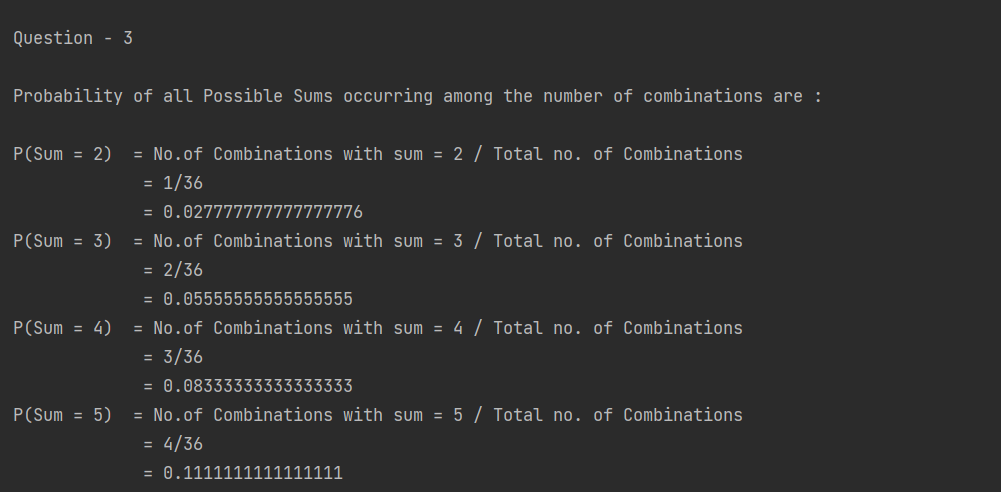
= No.of combinations with certain sum / Total combinations

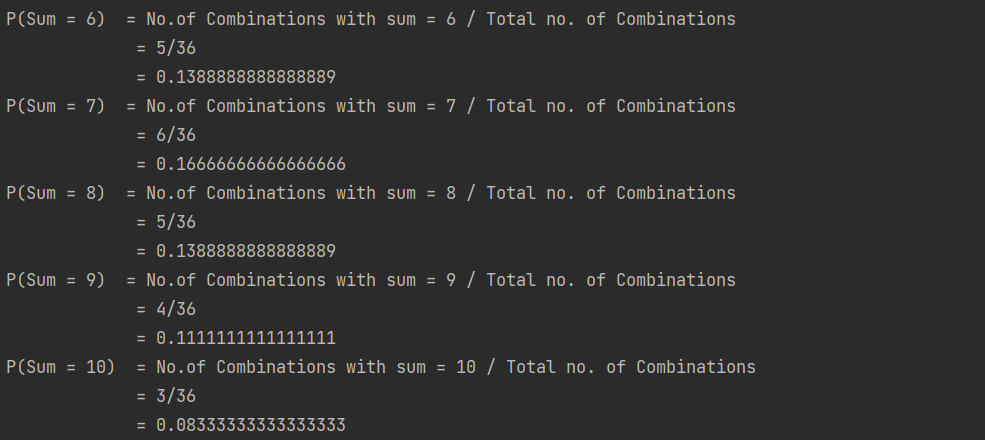
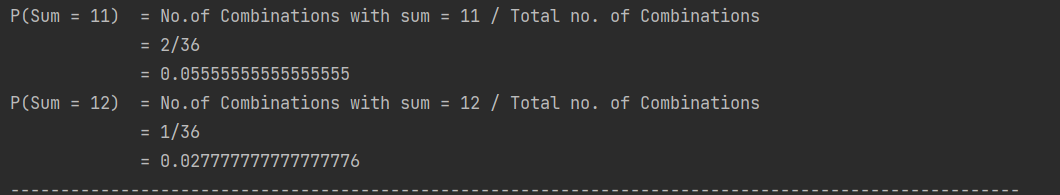
= Count / Total combinations

Now this is stored in a double value as decimal points will be generated.

This will provide output as:

**OUTPUT :**





Part A has been completed.

**Part – B**

Our dice have been doomed, so the spots goes missing. We are provided with 3 rules following which we should re-attach the spots on the dices.

The 3 rules are:

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

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

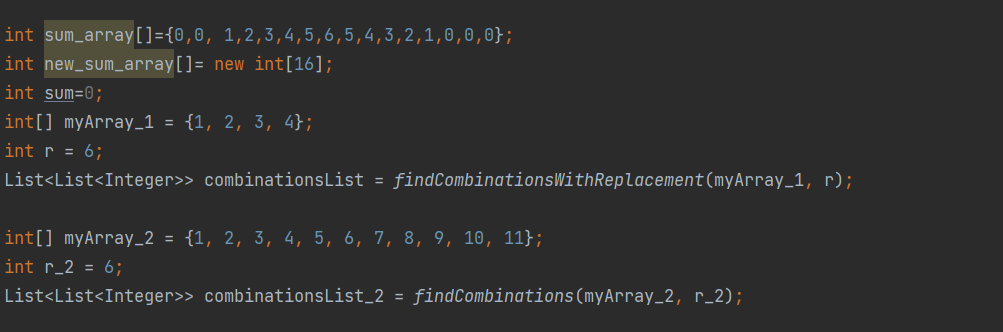
● Die B can have as many spots on a face as necessary i.e. even more than 6.

And importantly, the probability of obtaining the Sums must remain the

Same!

i.e., the new dices faces’ combinations must also give the same probability distribution as such as of our old dices.

For this part of question to get worked we should create a function undoom\_dice that takes the old dice faces and generate a new dice faces.



Let us first create sum\_array that contains how many times have a particular sum value has occurred in the older dice combinations.

sum\_array[]={0,0,1,2,3,4,5,6,5,4,3,2,1,0,0,0}

So, this says the number of pairs in the distribution contributing to 2(sum value) is just 1(i.e (1,1) pair) and sum values of the number of pairs in the distribution contributing to 3(sum value) is just 2 (i.e (1,2) and(2,1) ) and so on…

Similarly for the new combination that are about to be made, let us make a sum value pair array as :

int new\_sum\_array[]

Here we find both the arrays have the sizes as 16 because the maximum sum value that can be made in new combination is 15[4+11, denotes the pair(4,11)].

With the given condition we can come to a pre conclusion that

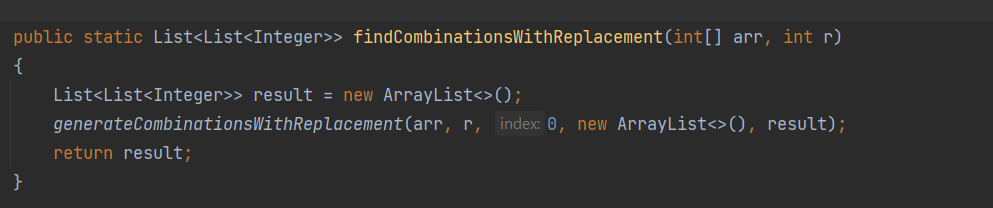
The new die A can have faces only with combinations [1,2,3,4] and with some repetitions but within this value.

And, The new die B can have values only within the range [1 to 11]. This is because the maximum sum obtained in our old combination is 12.

So here to get that combination the maximum possibility is (1,11)

i.e., 1 from die A and 11 from die B . Anything more than 11 in die B can generate wrong sum values during combinations.

With these conditions we are introducing new functions for generating the combinations.



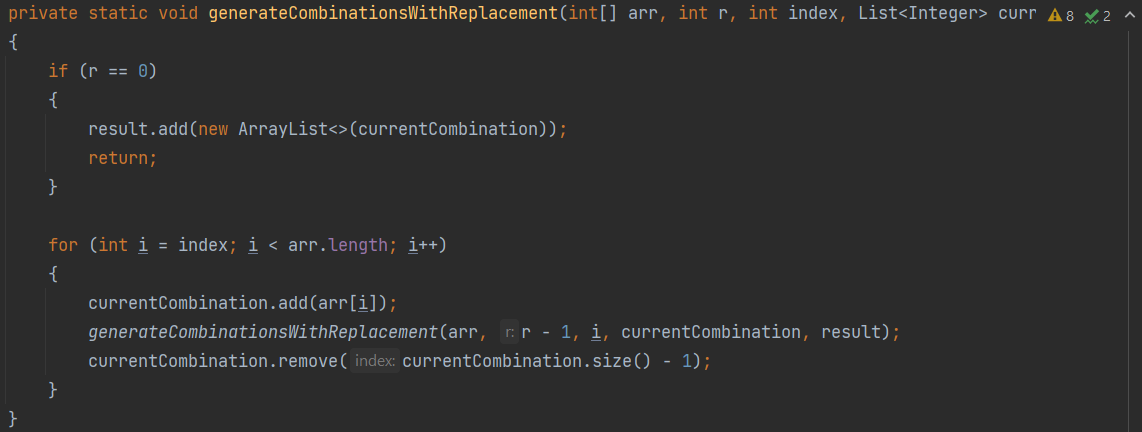
findCombinationsWithReplacement() is a function that we are using here to generate all combinations with a given set of constraints.

The constraint we use here is that

The array can take values within [1,2,3,4] and should generate number of possibilities with combination of 6 numbers   
i.e., r = 6

The actual generating of the combinations are done in another function.

Here we introduce List result to add all the combinations within which is used later. From this function   
generateCombinationsWith Replacement() is called



It is a recursive function, that is used to generate all the possible combinations.

Initially, the index is set as 0 and it is used in a for loop with the condition of I less than array length which is 4 here.

Here for each element every combinations are explored.

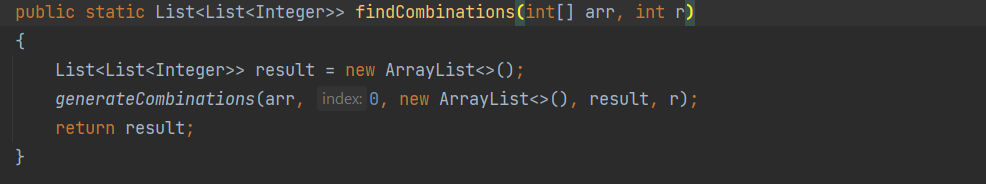
currentCombination.add(arr[i]) adds each element of the array to the combination.

generateCombinationsWith Replacement()is recursively called to generate the next set of elements.

currentCombination.remove(currentCombination.size-1) is used to backtrack by removing last element added to the current list.

It indicates that all the combination for current element has been explored and it moves to next element.

This function in this way will produce all possible combinations.

Similarly we use another two functions.

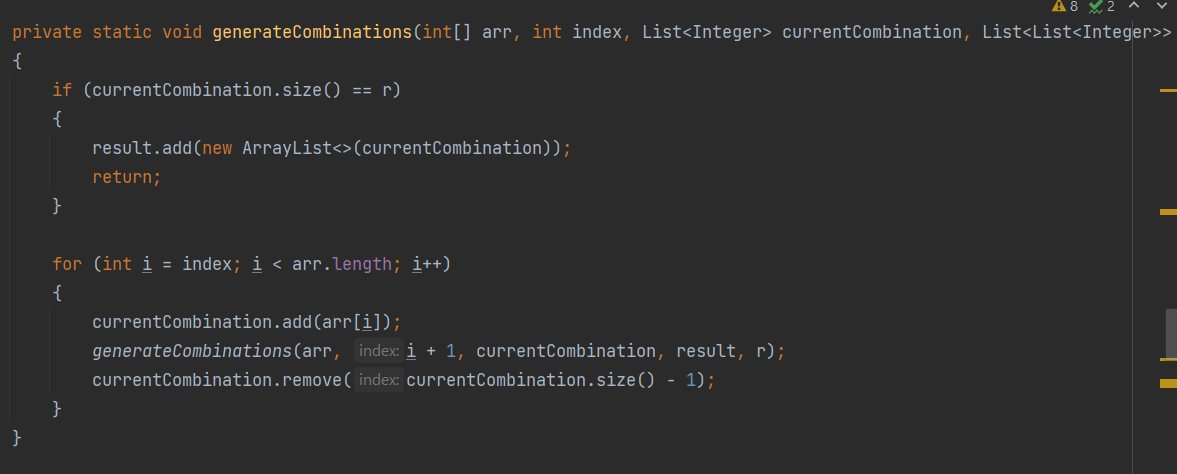
This is similar to the function findCombinationsWithReplacement()

But the only difference is repetition is not allowed

Here the constraint is

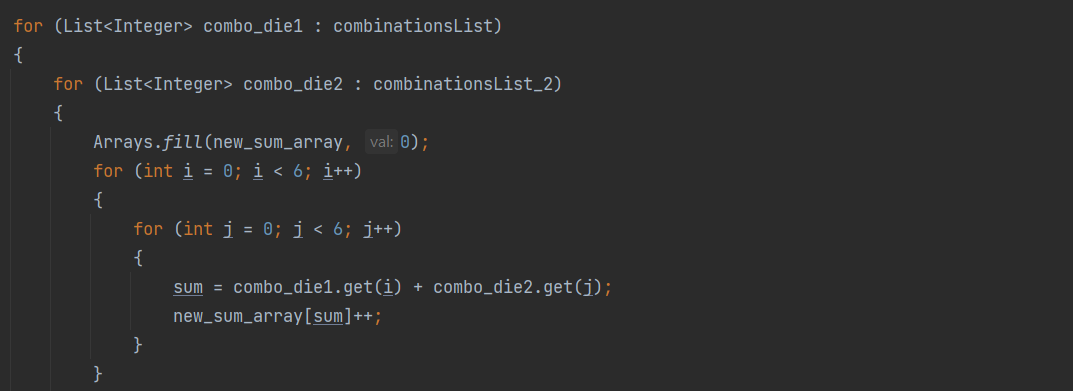
The array can take values within [1,2,3,4,5,6,7,8,9,10,11] and should generate number of possibilities with combination of 6 numbers   
i.e., r = 6

This will call the function generateCombinations()



This is similar to generateCombinationsWithReplacement() function but also here the repetitions are not allowed.

Now let us move to the actual functioning part of undoom\_dice():



Now we take each combination from combinationsList (possibiltes of new die A) and run another for loop for mapping it with every combination form combinationsList\_2 (possibilities of new die B).

Now run another 2 for loops to get each value for I and j from 1 to 6 that will denote each face of cube.

For each face we have to check the possibility sum of the new sides of the cube.

Combo\_die1 is one of the possibility combination of new die A and so

Combo\_die2 is one of the possibility combination of new die B

Their sum is stored in a variable sum

Where

Sum = combo\_die1.get(i) + combo\_die2.get(j)

Now this sum should be ranging from 2 to 15 as per our condition

For each sum value let us increment its value in the new\_sum\_array[].

Initially,

New\_sum\_array will be =[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0].

The variable sum will give the necessary sum values from 2 to 12 as per satisfying our old dice sum range

So if (1,1) is taken then (1+1) will give sum as 2

This will increment the new\_sum\_array as

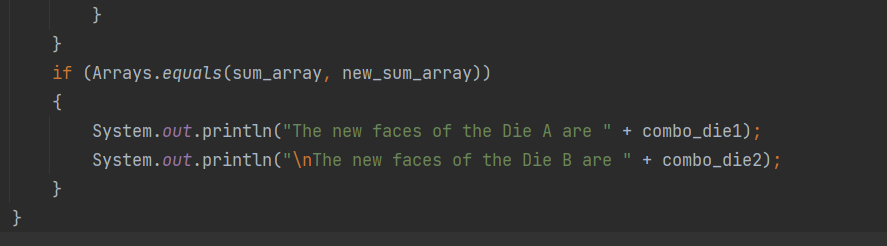
[0,0,1,0,0,0,0,0,0,0,0,0,0,0,0].

Here the position 2 has been incremented to 1.

If we keep on incrementing this process at one point of time our new\_sum\_array will be

[0,0,1,2,3,4,5,6,5,4,3,2,1,0,0,0]

Which is nothing but sum\_array[](The old dice pair sum ).



So for this we have used a condition :

If (sum\_array is equal to new\_sum array )

In short this means

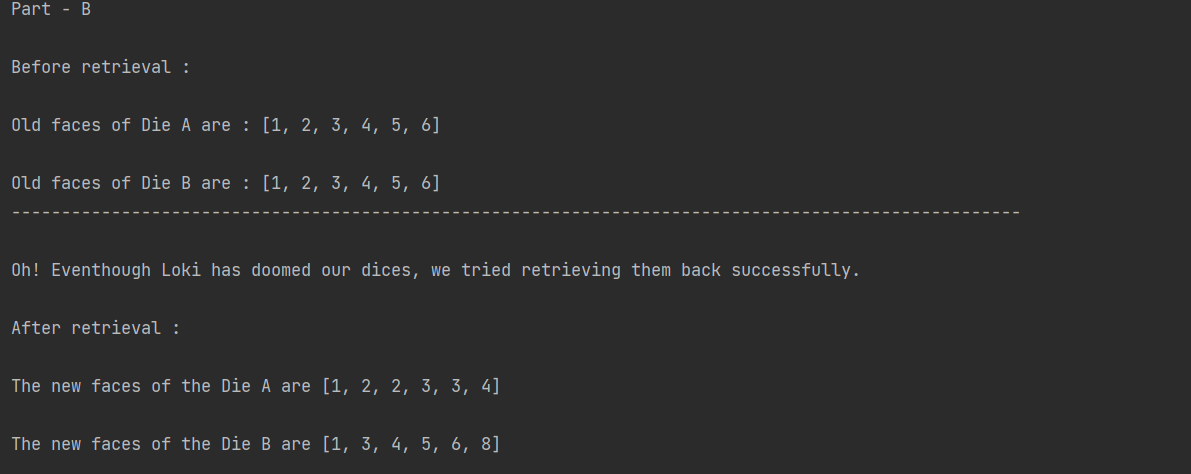
sum\_array[]= new\_sum\_array[]=[0,0,1,2,3,4,5,6,5,4,3,2,1,0,0,0]

If such a condition arises then the certain combination from the new die A combinations and new die B combination has to be printed.

Such a condition will arise only for one combination .

i.e.,

**OUTPUT :**



**CONCLUSION :**

Part A and Part B have been successfully completed using Java language .