Problem Statement : The Doomed Dice Challenge

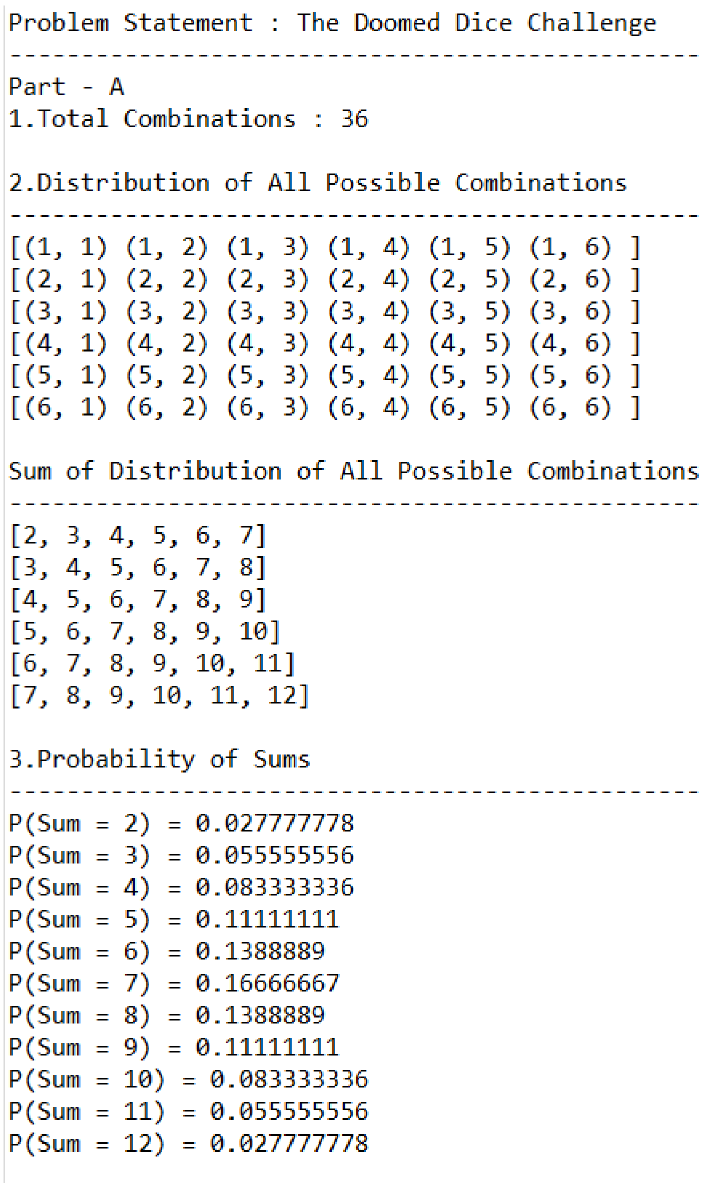
Github Link : https://github.com/Ahathiya/Securin-The-Doomed-Dice-Challenge.git

Part-A

```
package securinprogram;
import java.util.Arrays;

public class part_a
{
    public static void main(String[] args)
    {
        int[] originalDieA = {1, 2, 3, 4, 5, 6};
        int[] originalDieB = Arrays.copyOf(originalDieA,
originalDieA.length);
        System.out.println("Problem Statement : The Doomed Dice Challenge
\n-----------------------------------------------");

        // Part-A-1: Total Combinations
        int totalCombinations = originalDieA.length * originalDieB.length;
        System.out.println("Part - A \n1.Total Combinations : " +
totalCombinations);

        // Part-A-2: Combinations Distribution
        int[][] combinationsDistribution =
calculateCombinationsDistribution(originalDieA, originalDieB);
        System.out.println("\n2.Distribution of All Possible Combinations
\n----------------------------------------------- ");
        combinedDisplay(originalDieA,originalDieB);
        System.out.println("\nSum of Distribution of All Possible
Combinations \n----------------------------------------------- ");
        displayMatrix(combinationsDistribution);

        // Part-A-3: Probability of Sums
        System.out.println("\n3.Probability of Sums \n--------------------
----------------------------");
        calculateSumProbabilities(combinationsDistribution,
totalCombinations);
    }

    public static void combinedDisplay(int[] dieA, int[] dieB) {
        for (int a : dieA) {
            System.out.print("[");
            for (int b : dieB) {
                System.out.print("(" + a + ", " + b + ") ");
            }
            System.out.println("]");
        }
    }

    private static int[][] calculateCombinationsDistribution(int[] dieA,
int[] dieB)
```

```
    {
        int[][] distribution = new int[6][6];

        for (int i = 0; i < dieA.length; i++)
        {
            for (int j = 0; j < dieB.length; j++)
            {
                int sum = dieA[i] + dieB[j];
                distribution[dieA[i] - 1][dieB[j] - 1] = sum;
            }
        }
        return distribution;
    }

    private static int calculateSumProbabilities(int[][] distribution,
double totalCombinations)
    {
        for (int targetSum = 2; targetSum <= 12; targetSum++)
        {
            int count = 0;
            for (int i = 0; i < distribution.length; i++)
            {
                  for (int j = 0; j < distribution[i].length; j++)
                  {
                        if (distribution[i][j] == targetSum)
                        {
                              count++;
                        }
                  }
            }
            float probability = (float) ((float) count /
totalCombinations);
            System.out.println("P(Sum = " + targetSum + ") = " +
probability);
        }
            return 0;
    }

    private static void displayMatrix(int[][] matrix)
    {
        for (int[] row : matrix)
        {
            System.out.println(Arrays.toString(row));
        }
    }

}
```

OUTPUT :



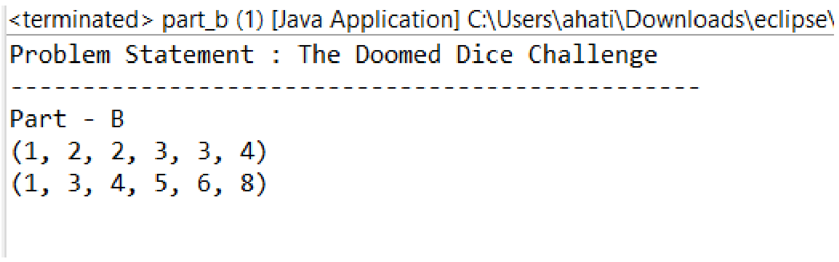
```
Problem Statement : The Doomed Dice Challenge
------------------------------------------------
Part - A
1.Total Combinations : 36

2.Distribution of All Possible Combinations
------------------------------------------------
[(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) ]

Sum of Distribution of All Possible Combinations
------------------------------------------------
[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]

3.Probability of Sums
------------------------------------------------
P(Sum = 2) = 0.027777778
P(Sum = 3) = 0.055555556
P(Sum = 4) = 0.083333336
P(Sum = 5) = 0.11111111
P(Sum = 6) = 0.1388889
P(Sum = 7) = 0.16666667
P(Sum = 8) = 0.1388889
P(Sum = 9) = 0.11111111
P(Sum = 10) = 0.083333336
P(Sum = 11) = 0.055555556
P(Sum = 12) = 0.027777778
```

Part-B

```java
package securinprogram;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class part_b {
    public static void main(String[] args) {
      System.out.println("Problem Statement : The Doomed Dice Challenge
\n-----------------------------------------------\nPart - B");
```

```
      int[] Die_A = {1,2,3,4,5,6};
        int[] Die_B = {1,2,3,4,5,6};

        List<List<Integer>> new_die_possibilities = transform(Die_A,
Die_B);
        for (List<Integer> possibility : new_die_possibilities) {
            
printOrderly(possibility.stream().mapToInt(Integer::intValue).toArray());
        }
    }

    public static void printOrderly(int[] array) {
    System.out.print("(");
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i]);
        if (i < array.length - 1) {
            System.out.print(", ");
        }
    }
    System.out.println(")");
}
    public static int sumOfArray(int[] array) {
        int ans = 0;
        for (int element : array) {
            ans = ans + element;
        }
        return ans;
    }

    public static int[][] sum_Distribution(int[] Die_A, int[] Die_B) {
        int[][] sum_Array = new int[Die_A.length][Die_B.length];
        for (int a = 0; a < Die_A.length; a++) {
            for (int b = 0; b < Die_B.length; b++) {
                sum_Array[a][b] = Die_A[a] + Die_B[b];
            }
        }
        return sum_Array;
    }

    public static double single_Probability(int[] Die_A, int[] Die_B, int
sum) {
        int[][] sum_Array = sum_Distribution(Die_A, Die_B);
        int count = 0;
        for (int[] rows : sum_Array) {
            for (int column : rows) {
                if (column == sum) {
                    count = count + 1;
                }
            }
        }
        double probability = (double) count / 36;
        return probability;
    }
    public static Map<Integer, Double> all_Probability(int[] Die_A, int[]
Die_B) {
        Map<Integer, Double> prob_dict = new HashMap<>();
        for (int i = 2; i <= 12; i++) {
            double probability = single_Probability(Die_A, Die_B, i);
            prob_dict.put(i, Math.round(probability * 10000.0) / 10000.0);
        }
        return prob_dict;
    }

    public static Set<List<Integer>> posibilities_Calc(List<Integer> curr,
int free_space, List<Integer> input_values, Set<List<Integer>>
posibilities, List<Integer> fixed_values, boolean repetition) {
        if (free_space == 0) {
            curr.sort(null);
            posibilities.add(new ArrayList<>(curr));
            return posibilities;
        }
        if (repetition) {
            for (int input_value : input_values) {
                posibilities_Calc(new
ArrayList<>(curr){{add(input_value);}}, free_space - 1, input_values,
posibilities, fixed_values, true);
            }
        } else {
            for (int i = 0; i < input_values.size(); i++) {
                List<Integer> remaining_input_values = new
ArrayList<>(input_values);
                remaining_input_values.remove(i);
                final int inputValue = input_values.get(i);
                posibilities_Calc(new
ArrayList<>(curr){{add(inputValue);}}, free_space - 1,
remaining_input_values, posibilities, fixed_values, false);
            }
        }
        return posibilities;
    }

    public static List<List<Integer>> transform(int[] die_a, int[] die_b)
{
        Map<Integer, Double> original_possibilities =
all_Probability(die_a, die_b);
        List<Integer> fixed_values_a = List.of(1, 4);
        List<Integer> input_values_a = List.of(1, 2, 3, 4);
        int free_space_a = 4;
        Set<List<Integer>> posibilities_a = new HashSet<>();
        Set<List<Integer>> new_die_a_possibility = posibilities_Calc(new
ArrayList<>(fixed_values_a), free_space_a, input_values_a, posibilities_a,
fixed_values_a, true);

        List<Integer> fixed_values_b = List.of(1, 8);
        List<Integer> input_values_b = List.of(2, 3, 4, 5, 6, 7);
        int free_space_b = 4;
        Set<List<Integer>> posibilities_b = new HashSet<>();
        Set<List<Integer>> new_die_b_possibility = posibilities_Calc(new
ArrayList<>(fixed_values_b), free_space_b, input_values_b, posibilities_b,
fixed_values_b, false);

        List<List<Integer>> new_die_possibilities = new ArrayList<>();
        for (List<Integer> a : new_die_a_possibility) {
            for (List<Integer> b : new_die_b_possibility) {
                if
((sumOfArray(a.stream().mapToInt(Integer::intValue).toArray()) +
sumOfArray(b.stream().mapToInt(Integer::intValue).toArray())) == 42) {
                    Map<Integer, Double> new_sum_possibility =
all_Probability(a.stream().mapToInt(Integer::intValue).toArray(),
b.stream().mapToInt(Integer::intValue).toArray());
                    if
(original_possibilities.equals(new_sum_possibility)) {
                        new_die_possibilities.add(a);
                        new_die_possibilities.add(b);
                    }
                }
            }
        }
        return new_die_possibilities;
    }
}
```

OUTPUT :



```
<terminated> part_b (1) [Java Application] C:\Users\ahati\Downloads\eclipse\
Problem Statement : The Doomed Dice Challenge
------------------------------------------------
Part - B
(1, 2, 2, 3, 3, 4)
(1, 3, 4, 5, 6, 8)
```

Problem Statement :

You are given two six-sided dice, Die A and Die B, each with faces numbered from 1 to 6. You can only roll both the dice together & your turn is guided by the obtained sum.

Example: Die A = 6, Die B = 3. Sum = 6 + 3 = 9 You 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.

<u>Solution :</u>

This Java program, named **" DiceTransformation "**, performs several tasks related to two six-sided dice. Let's break down the code section by section :

<u>Part – A : Original Dice Combinations</u>

1. *How many total combinations are possible? Show the math along with the code!*
   Ans :
   ```
   Total Combinations : 36
   (by multiplying the size of the both dice 6*6 = 36)
   ```

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
   Ans :
   ```
   Distribution of All Possible Combinations
   ------------------------------------------------
   [(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) ]

   Sum of Distribution of All Possible Combinations
   ------------------------------------------------
   [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]
   ```

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.

Ans :

```
Probability of Sums
-----------------------------
P(Sum = 2) = 0.027777778
```

```
P(Sum = 3) = 0.055555556
P(Sum = 4) = 0.083333336
P(Sum = 5) = 0.11111111
P(Sum = 6) = 0.1388889
P(Sum = 7) = 0.16666667
P(Sum = 8) = 0.1388889
P(Sum = 9) = 0.11111111
P(Sum = 10) = 0.083333336
P(Sum = 11) = 0.055555556
P(Sum = 12) = 0.027777778
```

<u>Code Logic :</u>

1.Initialization of Original Dice Arrays :

- Two arrays, *"originalDieA"* and *"originalDieB"*, are created and initialized with values representing the faces of two six-sided dice.

2.Total Combinations :

- *"totalCombinations"* variable calculates the total number of combinations possible when rolling both dice.

3. Combinations Distribution :

- *"combinedDisplay"* method to show the over all combinations of the faces when the two dice are rolled.

- *"calculateCombinationsDistribution"* method creates a 2D array *"combinationsDistribution "* to store the sums of the faces when the two dice are rolled. It populates this array by iterating over all combinations.

4.Probability of Sums :

- *"calculateSumProbabilities"* method calculates and prints the probability of each possible sum from 2 to 12 based on the distribution obtained earlier.

<u>Helper Methods :</u>

1. ***displayMatrix*** - A utility method to print a 2D matrix.

2. ***calculateSameSumProbabilities*** - Calculates the probability distribution of the sums for the given dice arrays.

<u>Explanation of Dice Transformation Logic :</u>

- ✓ In the "*calculateSameSumProbabilities"* method, the array length of probabilities is correctly set to 13, accounting for possible sums from 2 to 12.
- ✓ The "*calculateSumProbabilities"* method calculates the probability distribution based on the distribution array obtained from rolling two dice.

Part – B : Dice Transformation

Now comes the real challenge. 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 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 : Transform Function undoom_dice that takes (Die_A, Die_B) as input output New_Die_A = [?, ?, ?, ?, ?, ?],New_Die_B = [?, ?, ?, ?, ?, ?] where No New_Die A[x] > 4

Ans :

```
Part - B - Transform Dice
--------------------------------
Original Die A : [1, 2, 3, 4, 5, 6]
Original Die B : [1, 2, 3, 4, 5, 6]

New Die A      : [1, 2, 2, 3, 3, 4]
New Die B      : [1, 3, 4, 5, 6, 8]
```

Code Logic :

1. ***"sum_Distribution"*** Method :

- This method calculates the sum distribution of two dice arrays, ***Die_A*** and ***Die_B.***

- It creates a 2D array, ***"sum_Array",*** where ***"sum_Array[a][b]"*** stores the sum of ***Die_A[a]*** and ***Die_B[b].***

2. ***"single_Probability"*** Method :

- Given ***Die_A*** and ***Die_B*** arrays and a target ***sum***, this method calculates the probability of obtaining the target sum when rolling both dice.

- It uses the ***"sum_Distribution"*** method to get the sum array and counts occurrences of the target sum.

3. ***"all_Probability"*** Method :

- This method computes the probabilities of getting sums from 2 to 12 by calling ***"single_Probability***" for each sum.

4. "***posibilities_Calc"*** Method:**

- This recursive method generates all possible combinations of values with repetition or without repetition.

- It takes parameters like ***curr*** (current combination), ***free_space*** (remaining free slots), ***input_values, posibilities*** (set to store unique combinations), ***fixed_values*** *and* ***repetition***.

5. ***"transform"*** Method :

- This method transforms two dice arrays, ***die_a*** and ***die_b***, based on specific rules.

- It calculates the original probabilities using ***all_Probability***.

- It generates all possible combinations for each die using ***posibilities_Calc***.

- It checks for combinations where the sum is 42 and their probabilities match the original probabilities.

6. ***"printOrderly"*** Method :

- This method prints an array in the format (1, 2, 3, 4, 5, 6).

The code essentially simulates the transformation of two dice arrays while ensuring that certain values are fixed and specific rules are followed. The result is a set of transformed dice combinations with their sums having the same probabilities as the original dice.