**Problem Statement : The Doomed Dice Challenge**

**Part – A**

1. How many total combinations are possible ?

**Logic to the problem :** Lets take a two dice with six sides . The probability formula is used to derive at the solution to this problem statement.

6^(no of dices) = total combinations here 6 represents the no of dots on a dice.

**Code : (Java)**

public class Main{

public static void main(String[] args){

int totalcombinations = 6\*6;

System.out.println("The possible total combinations are "+totalcombinations);

}

}

**Output :**



**Explanation** : For 2 dice , the formula is 6^2 which is equal to 36 that is the total number of combinations. If we use 3 dice, then find 6^3 which is the answer . Like wise we can use this formula to find the combinations for any no of dices.

**2. Calculate and display the distribution of all possible combinations that can be obtained when rolling both die A and die B together.**

**Logic to the problem** : Now we can find the distribution combination with the following logic. Take the first dice dots [1,2,3,4,5,6] and combine every dots with other dice dots [1,2,3,4,5,6]. Then find the 6x6 matrix that represents the possible sums obtained by the two dices.

**Code: (Java)**

public class Main{

public static void main(String args[]){

int[][] d\_m\_c = new int[6][6];

for(int i=1;i<=6;i++){

for(int j=1;j<=6;j++){

int sum = i+j;

d\_m\_c[i-1][j-1] = sum;

}

}

System.out.println("Possible combinations of Distribution:") ;

for(int[] r : d\_m\_c){

for (int i : r){

System.out.print(i+" ");

}

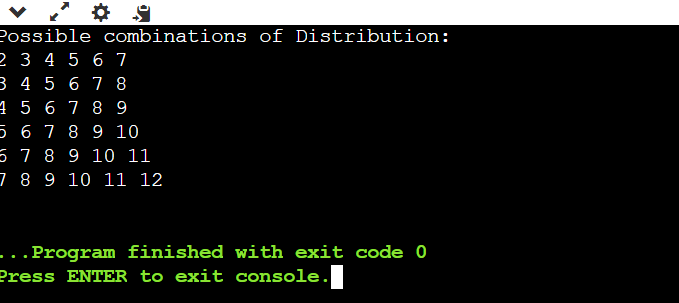
System.out.println();

}

}

}

**Output :**



**Explanation :** The two dimension matrix of size 6x6 is declared to store the result . Then two for loops are used to find the sum of the two dots on dice and store it in the matrix.

**3. Calculate the Probability of all possible sums occurring among the number of combinations from (2).**

**Logic to the problem :** Using the sum of the dots starting from 2 to 12 for 2 dices , we can find the probability for each sum by using the below formula.

P(sum=x) = no of times the x occurs/total combinations where x is the sum

**Code : (Java)**

public class Main {

public static void main(String[] args) {

int total\_combo = 6 \* 6;

int[] p = new int[11];

for (int i = 1; i <= 6; i++) {

for (int j = 1; j <= 6; j++) {

int v = i + j;

p[v - 2]++;

}

}

System.out.println("Probability of all possible sums:");

for (int i = 2; i <= 12; i++) {

int index = i - 2;

double prob = (double) p[index] / total\_combo;

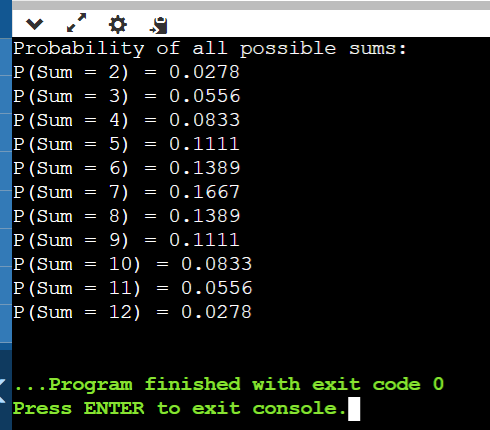
System.out.printf("P(Sum = %d) = %.4f%n", i, (prob == 0) ? 1.0 / total\_combo : prob);

}

}

}

**Output :**



**Explanation :**

* This method orchestrates the computation and presentation of the probabilities associated with each potential sum resulting from the combined roll of two six-sided dice.
* It iterates through possible sums ranging from 2 to 12, subsequently counting the occurrences of each sum within the combinationsMatrix.

The probability of each sum is calculated by dividing the count by the total number of combinations (totalCombinations).

Example : P(sum=2) = no of times 2 occur/total combinations

= 1/36 = 0.0278

**PART-B**

**Key Constraints:**

1. Two dice, dieA and dieB, have specific faces with values.

2. DieA has faces {0, 1, 2, 3, 4}.

3. DieB has faces {1, 2, 3, 4, 5, 6, 7, 8}.

4. The sum of faces for each die combination should match predefined probabilities.

5. The predefined probabilities for the sum of faces are stored in `originalSumProbabilities`.

6. The total number of spots on the dice should be 42.

**Logic Overview:**

1. The program aims to find valid combinations of faces for dieA and dieB such that their sums match the predefined probabilities.

2. It generates all possible combinations of faces for dieA and dieB and checks if their sum probabilities match the predefined ones.

3. If a valid combination is found, it prints the combination; otherwise, it indicates that no valid combination exists.

**Step-by-Step Explanation:**

1. Initialization:

- `totalSpots`: Total number of spots on both dice combined.

- `possibleDieFacesA`: Faces for dieA.

- `possibleDieFacesB`: Faces for dieB.

- `originalSumProbabilities`: Predefined probabilities for the sum of faces.

2. Main Method (`public static void main(String[] args)`):

- Initializes two arrays for dieA and dieB.

- Calls `undoomDice` method with dieA and dieB.

3. `undoomDice` Method:

- Generates all combinations for dieA using `dieACombinations` method.

- For each combination of dieA, generates combinations for dieB using `dieBCombinations`.

- Checks if the combination is valid using `isValidDice`.

- Prints the valid combination or indicates that no valid combination exists.

4. `dieACombinations` Method:

- Recursively generates all combinations of faces for dieA.

5. `dieBCombinations` Method:

- Recursively generates all combinations of faces for dieB based on the previous face index and current spot count.

6. `calculateDiceProbabilities` Method:

- Calculates the probabilities for the sum of faces for a given combination of dieA and dieB.

7. `isValidDice` Method:

- Checks if the calculated probabilities match the predefined probabilities.

8. `sumList` Method:

- Calculates the sum of a list of integers.

**Understanding the Problem:**

The problem is to find valid combinations of faces for two dice, dieA and dieB, such that their sum probabilities match predefined values.

**Java Code :-**

**import java.util.ArrayList;**

**import java.util.HashMap;**

**import java.util.List;**

**import java.util.Map;**

**public class DiceCombinations {**

**private static final int totalSpots = 42;**

**private static final int[] possibleDieFacesA = {0, 1, 2, 3, 4};**

**private static final int[] possibleDieFacesB = {1, 2, 3, 4, 5, 6, 7, 8};**

**private static final Map<Integer, Integer> originalSumProbabilities;**

**static {**

**originalSumProbabilities = new HashMap<>();**

**originalSumProbabilities.put(2, 1);**

**originalSumProbabilities.put(4, 3);**

**originalSumProbabilities.put(5, 4);**

**originalSumProbabilities.put(6, 5);**

**originalSumProbabilities.put(7, 6);**

**originalSumProbabilities.put(9, 4);**

**originalSumProbabilities.put(3, 2);**

**originalSumProbabilities.put(8, 5);**

**originalSumProbabilities.put(10, 3);**

**originalSumProbabilities.put(11, 2);**

**originalSumProbabilities.put(12, 1);**

**}**

**public static void main(String[] args) {**

**int[] dieA = {1, 2, 3, 4, 5, 6};**

**int[] dieB = {1, 2, 3, 4, 5, 6};**

**undoomDice(dieA, dieB);**

**}**

**private static void dieACombinations(List<Integer> currentDie, List<List<Integer>> validCombosDieA) {**

**if (currentDie.size() == 6) {**

**validCombosDieA.add(new ArrayList<>(currentDie));**

**return;**

**}**

**for (int face : possibleDieFacesA) {**

**currentDie.add(face);**

**dieACombinations(currentDie, validCombosDieA);**

**currentDie.remove(currentDie.size() - 1);**

**}**

**}**

**private static void dieBCombinations(int previousFaceIndex, List<Integer> currentDie,**

**int currentSpotCount, List<List<Integer>> validCombosDieB) {**

**if (currentSpotCount > totalSpots) {**

**return;**

**}**

**if (currentDie.size() == 6) {**

**if (currentSpotCount == totalSpots) {**

**validCombosDieB.add(new ArrayList<>(currentDie));**

**}**

**return;**

**}**

**for (int i = previousFaceIndex + 1; i < possibleDieFacesB.length; i++) {**

**currentDie.add(possibleDieFacesB[i]);**

**dieBCombinations(i, currentDie, currentSpotCount + possibleDieFacesB[i], validCombosDieB);**

**currentDie.remove(currentDie.size() - 1);**

**}**

**}**

**private static Map<Integer, Integer> calculateDiceProbabilities(List<Integer> dieA, List<Integer> dieB) {**

**Map<Integer, Integer> prob = new HashMap<>();**

**for (int dieFaceA : dieA) {**

**for (int dieFaceB : dieB) {**

**int sum = dieFaceA + dieFaceB;**

**prob.put(sum, prob.getOrDefault(sum, 0) + 1);**

**}**

**}**

**return prob;**

**}**

**private static boolean isValidDice(List<Integer> dieA, List<Integer> dieB) {**

**Map<Integer, Integer> newProbabilities = calculateDiceProbabilities(dieA, dieB);**

**return newProbabilities.equals(originalSumProbabilities);**

**}**

**private static void undoomDice(int[] dieA, int[] dieB) {**

**List<List<Integer>> validCombosDieA = new ArrayList<>();**

**dieACombinations(new ArrayList<>(), validCombosDieA);**

**for (List<Integer> dieCombinationA : validCombosDieA) {**

**List<List<Integer>> validCombosDieB = new ArrayList<>();**

**dieBCombinations(-1, new ArrayList<>(), sumList(dieCombinationA), validCombosDieB);**

**for (List<Integer> dieCombinationB : validCombosDieB) {**

**if (isValidDice(dieCombinationA, dieCombinationB)) {**

**System.out.println("New\_Die\_A = " + dieCombinationA);**

**System.out.println("New\_Die\_B = " + dieCombinationB);**

**return;**

**}**

**}**

**}**

**System.out.println("No valid dice combination exists");**

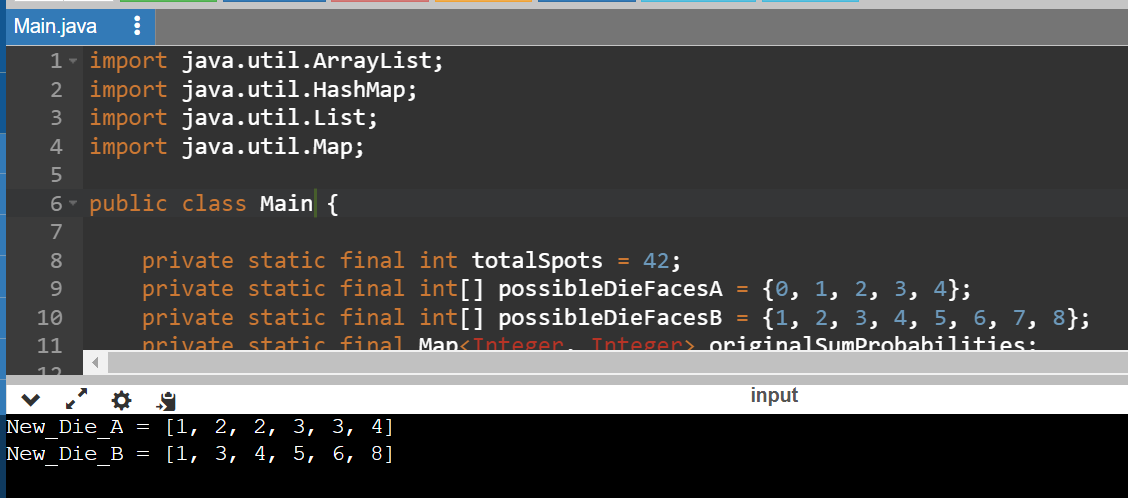
**}**

**private static int sumList(List<Integer> list) {**

**return list.stream().mapToInt(Integer::intValue).sum();**

**}**

**}**

****

**Designing the Solution:**

1. Generate all combinations for dieA and dieB.

2. Check each combination's validity using predefined probabilities.

3. Print the valid combination if found; otherwise, indicate that no valid combination exists.

**Example Usage:**

1. Define the faces for dieA and dieB.

2. Run the program to find valid combinations.

**Result:**

The program either prints the valid combination of dieA and dieB or indicates that no valid combination exists.