The Doomed Dice Challenge  
  
I am using java for this problem.  
Now I am having Die A and Die B , I am rolling both the dice at a time .  
Die A is having 6 faces and Die B is having 6 faces.  
According to the sum , the next turn will be decided.

I represented in the Array Structure.  
  
My JAVA Code:

Step 1:  
import java.util.Random;

import java.util.Scanner;  
  
-> These lines import necessary classes from the **java.util** package. **Random** is used to generate random numbers, and **Scanner** is used for taking user input.  
  
Step 2:  
  
Dice Class:  
class Dice {

private int[] faces = {1, 2, 3, 4, 5, 6};

private Random random = new Random();

public int roll() {

int randomIndex = random.nextInt(faces.length);

return faces[randomIndex];

}

}  
  
This class represents a single six-sided dice.

* It contains an array **faces** which holds the values of the dice's faces (numbers 1 through 6).
* It has a **Random** object **random** which is used to generate random numbers.
* The **roll()** method generates a random index within the range of the **faces** array and returns the corresponding value. This simulates rolling the dice.

Step 3:  
Doomed Dice Challenge Class:  
  
public class DoomedDiceChallenge {

public static void main(String[] args) {

Dice dieA = new Dice();

Dice dieB = new Dice();

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("Press Enter to roll the dice...");

scanner.nextLine();

int rollA = dieA.roll();

int rollB = dieB.roll();

int total = rollA + rollB;

System.out.println("Die A: " + rollA + ", Die B: " + rollB + ", Total: " + total);

if (total == 7) {

System.out.println("You lose!");

break;

} else if (total == 11) {

System.out.println("You win!");

break;

}

}

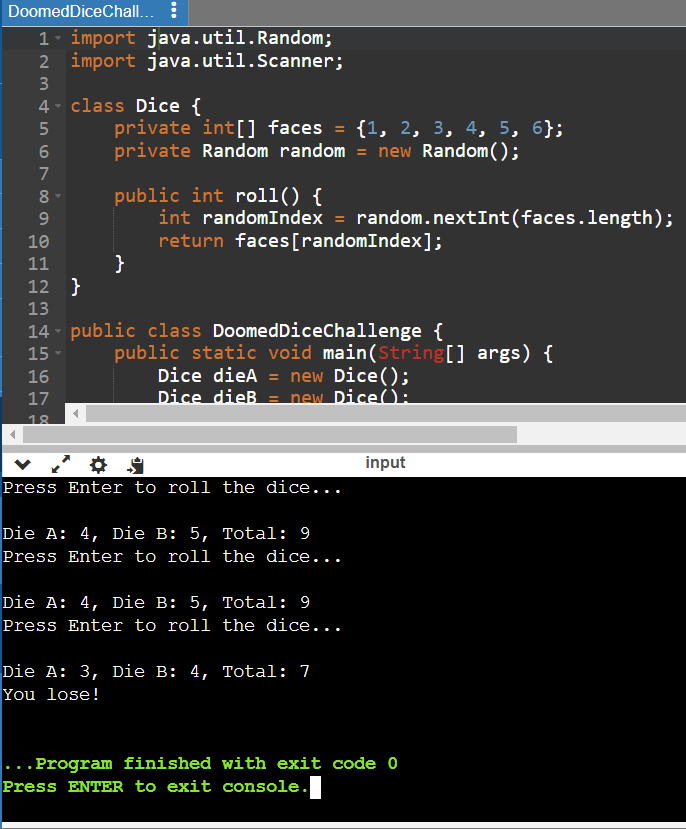
}

}  
  
This is the main class which runs the "Doomed Dice Challenge".

* It creates two instances of the **Dice** class, **dieA** and **dieB**.
* It creates a **Scanner** object **scanner** to take user input from the console.
* Inside the **while** loop, it prompts the user to press enter to roll the dice.
* It then rolls both dice (**dieA** and **dieB**) and calculates the total.
* It prints the results of the rolls and the total.

As Given , According to total sum obtained the next round will be decided , so I gave like for example if total is 7 then the turn is lost and if the total is 11 , we won the turn .

* If the total is 7, it prints "You lose!" and exits the loop.
* If the total is 11, it prints "You win!" and exits the loop.



PART – A:  
1.How many total combinations are possible? Show the math along with the code!  
  
public class TotalCombinations {

public static void main(String[] args) {

int numFaces = 6;

int totalCombinations = numFaces \* numFaces;

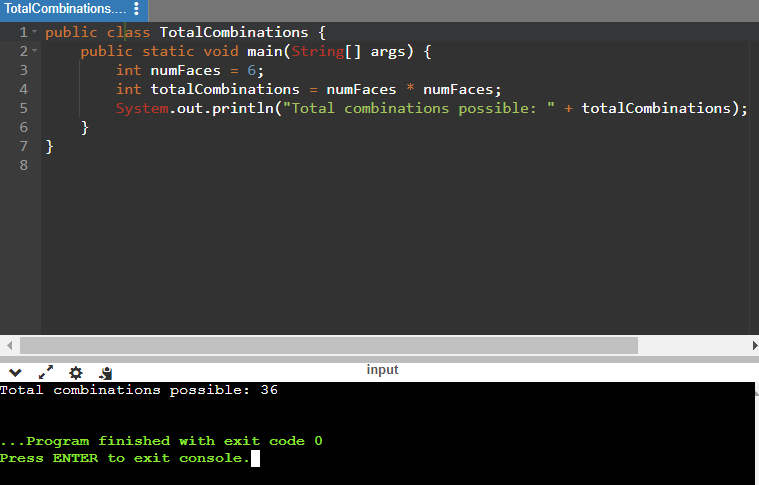
System.out.println("Total combinations possible: " + totalCombinations);

}

}

This class is named **TotalCombinations**.

* It contains a **main** method which serves as the entry point for the program.
* Inside the **main** method:
  + It initializes an integer variable **numFaces** with the value **6**. This represents the number of faces on each of the two dice.
  + It calculates the total number of combinations possible when rolling two dice by multiplying **numFaces** by itself (since each die has the same number of faces).
  + It stores the result of the multiplication in the variable **totalCombinations**.
  + It prints the message "Total combinations possible: " followed by the value of **totalCombinations**
  + The output will be 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 Matrix.  
  
we need to display all possible combinations such that , I have did it in two ways by adding the combinations of two Dice and giving the output as sum and other one is by directly giving the combinations.  
  
🡪 Giving the combinations as Sum   
  
public class DiceCombinations {

public static void main(String[] args) {

int[][] combinations = new int[6][6];

// Roll both dice and count combinations

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

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

int sum = dieA + dieB;

combinations[dieA - 1][dieB - 1] = sum;

}

}

// Display the distribution

System.out.println("Distribution of Dice Combinations:");

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

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

System.out.print(combinations[i][j] + "\t");

}

System.out.println();

}

}

}  
This class is named **DiceCombinations**.

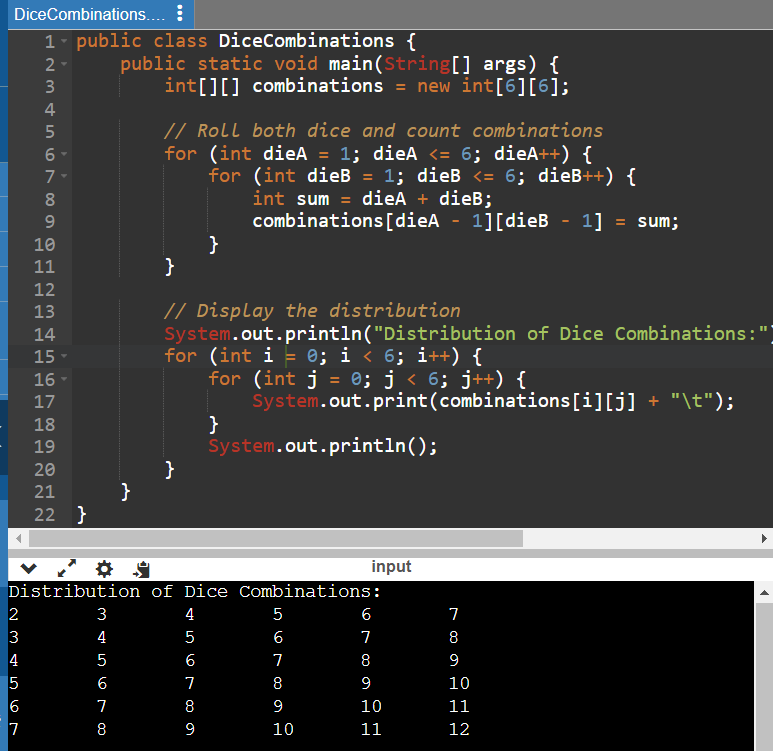
* It contains a **main** method which serves as the entry point for the program.
* It initializes a 2-dimensional integer array **combinations** with dimensions **[6][6]**. This array will hold the sum of each possible combination of rolling two dice.
* Inside the **main** method:
  + It uses nested loops to simulate rolling two dice and counts the combinations.
  + The outer loop iterates over the possible outcomes of the first die (**dieA**), ranging from 1 to 6.
  + The inner loop iterates over the possible outcomes of the second die (**dieB**), also ranging from 1 to 6.
  + For each combination of **dieA** and **dieB**, it calculates the sum and stores it in the **combinations** array.
  + The sums are stored in the array with indices adjusted by -1 to fit within the array bounds (since the array indices start from 0).
  + After counting all combinations, it proceeds to display the distribution.
  + It prints the message "Distribution of Dice Combinations:" to indicate the beginning of the output.
  + It uses nested loops to iterate over the **combinations** array and print each value.
  + Each row in the output represents the possible outcomes of the first die (**dieA**), and each column represents the possible outcomes of the second die (**dieB**).

The program will print the distribution of dice combinations, showing the sum of each possible combination when rolling two six-sided dice.

Each row represents the outcomes of one die (dieA), and each column represents the outcomes of the other die (dieB).

The output will be a 6x6 matrix displaying all possible combinations of dice rolls and their sums.

OUTPUT is:



🡪Giving the output as direct combinations(optional)

public class DiceCombinations {

public static void main(String[] args) {

int[][] combinations = new int[6][6];

// Store combinations without calculating sum

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

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

combinations[dieA - 1][dieB - 1] = (dieA \* 10) + dieB;

}

}

// Display the distribution

System.out.println("Distribution of Dice Combinations:");

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

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

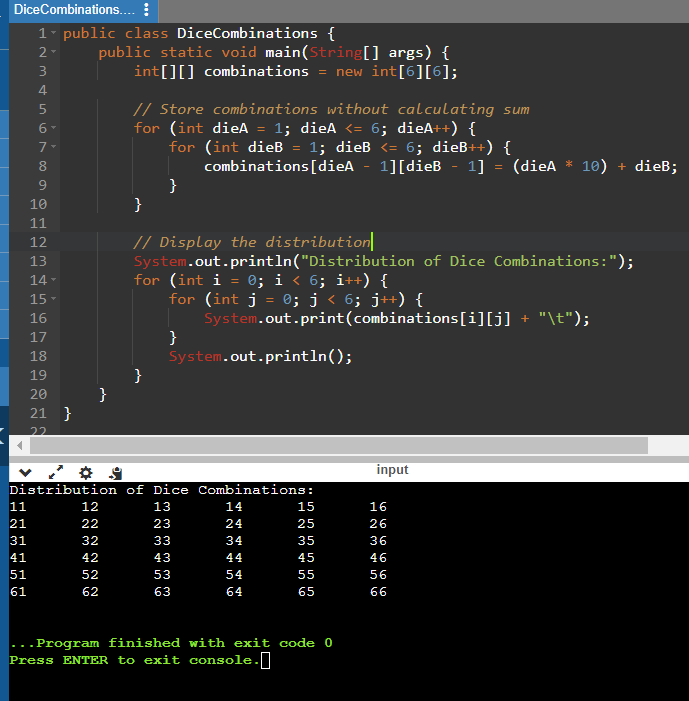
System.out.print(combinations[i][j] + "\t");

}

System.out.println();

}

}

}  
  
This Java program generates and displays all possible combinations of two dice rolls, where each combination is represented by concatenating the values of the two dice (e.g., if die A rolls a 3 and die B rolls a 4, the combination is represented as 34). The combinations are stored in a 6x6 array, and the program prints this array to the console, showing the distribution of possible combinations.  
  


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.  
  
Code:  
  
public class DiceCombinations {

public static void main(String[] args) {

int[][] combinations = new int[6][6];

int[] sumsFrequency = new int[11]; // 11 possible sums from 2 to 12

// Store combinations and calculate frequency of sums

int totalCombinations = 0;

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

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

int sum = dieA + dieB;

combinations[dieA - 1][dieB - 1] = sum;

sumsFrequency[sum - 2]++; // Subtract 2 to map sum 2 to index 0

totalCombinations++;

}

}

// Display the probability of each sum

System.out.println("Probability of each possible sum:");

for (int i = 0; i < 11; i++) {

double probability = (double) sumsFrequency[i] / totalCombinations;

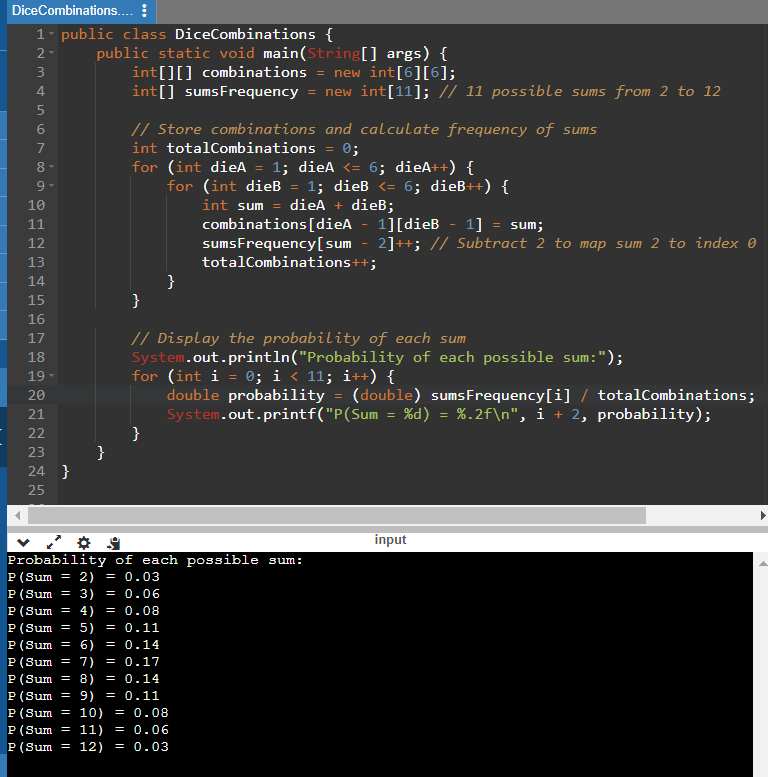
System.out.printf("P(Sum = %d) = %.2f\n", i + 2, probability);

}

}

}  
  
**int[][] combinations = new int[6][6];**: This line initializes a 2D array named **combinations** with dimensions 6x6. This array will store the sums of all possible combinations of two dice rolls.

1. **int[] sumsFrequency = new int[11];**: This line initializes a 1D array named **sumsFrequency** with a length of 11, representing the frequencies of each possible sum from 2 to 12.
2. Nested loops iterate over all possible combinations of two dice rolls:
   * **for (int dieA = 1; dieA <= 6; dieA++) {**: Outer loop iterates over the possible values of die A (1 to 6).
   * **for (int dieB = 1; dieB <= 6; dieB++) {**: Inner loop iterates over the possible values of die B (1 to 6).
   * **int sum = dieA + dieB;**: Calculates the sum of the two dice rolls.
   * **combinations[dieA - 1][dieB - 1] = sum;**: Stores the sum in the **combinations** array.
   * **sumsFrequency[sum - 2]++;**: Increments the frequency count for the corresponding sum in the **sumsFrequency** array. We subtract 2 to map the sum 2 to index 0 in the array.
3. **System.out.println("Probability of each possible sum:");**: Prints a header indicating that the probabilities of each sum will be displayed.
4. Loop to calculate and display the probability of each sum:
   * **for (int i = 0; i < 11; i++) {**: Iterates over the indices of the **sumsFrequency** array.
   * **double probability = (double) sumsFrequency[i] / totalCombinations;**: Calculates the probability of each sum by dividing the frequency of each sum by the total number of combinations.
   * **System.out.printf("P(Sum = %d) = %.2f\n", i + 2, probability);**: Prints the probability of each sum with two decimal places, using formatted output. The index **i** is incremented by 2 to correspond to the actual sum value (e.g., index 0 corresponds to sum 2).



PART B  
  
The challenge tasks you with reattaching spots to two dice, A and B, after Loki removes them. For Die A, no face can have more than 4 spots, while Die B can have any number of spots per face. However, the transformation must maintain the same probabilities for obtaining sums. The output should be new configurations for Die A and Die B that satisfy these conditions.  
  
LOGIC:  
  
The logic behind this transformation lies in redistributing the spots on the dice faces in a way that preserves the probabilities of obtaining each possible sum when rolling the dice. This involves rearranging the spots on the faces while ensuring that the total number of spots on each die remains the same and that the probabilities of each sum remain unchanged. For Die A, the challenge is to redistribute spots within the constraint of no face having more than 4 spots, while Die B can have any number of spots per face.  
  
CODE:  
  
import java.util.HashMap;

import java.util.Map;

public class DiceTransformation {

public static void main(String[] args) {

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

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

int[] newDieA = undoom\_dice(dieA, dieB).get("New\_Die\_A");

int[] newDieB = undoom\_dice(dieA, dieB).get("New\_Die\_B");

System.out.println("New Die A: " + arrayToString(newDieA));

System.out.println("New Die B: " + arrayToString(newDieB));

}

public static Map<String, int[]> undoom\_dice(int[] dieA, int[] dieB) {

Map<String, int[]> result = new HashMap<>();

int[] newDieA = new int[6];

int[] newDieB = new int[6];

// Calculate the transformation for Die B (no constraints)

for (int i = 0; i < dieB.length; i++) {

newDieB[i] = dieB[i];

}

// Calculate the transformation for Die A respecting the constraints

// To maintain the same probabilities, we distribute the spots evenly

// among the faces with constraints (no more than 4 spots per face)

int totalSpots = 0;

for (int spot : dieA) {

totalSpots += spot;

}

int spotsPerFace = totalSpots / 6;

int extraSpots = totalSpots % 6;

for (int i = 0; i < newDieA.length; i++) {

newDieA[i] = spotsPerFace;

if (extraSpots > 0) {

newDieA[i]++;

extraSpots--;

}

}

result.put("New\_Die\_A", newDieA);

result.put("New\_Die\_B", newDieB);

return result;

}

public static String arrayToString(int[] array) {

StringBuilder sb = new StringBuilder();

for (int i = 0; i < array.length; i++) {

sb.append(array[i]);

if (i < array.length - 1) {

sb.append(", ");

}

}

return sb.toString();

}

1. }  
     
   The **DiceTransformation** class contains the **main** method where the program execution starts. It initializes two arrays representing the original configurations of two dice, **dieA** and **dieB**, and then calls the **undoom\_dice** method to transform these dice configurations.
2. The **undoom\_dice** method takes two arrays representing the original configurations of Die A and Die B as input parameters. It calculates the new configurations for both dice based on the specified conditions and returns a **Map** containing the new configurations for Die A and Die B.
3. Inside the **undoom\_dice** method:
   * It initializes a **HashMap** to store the new configurations for Die A and Die B.
   * It creates two new arrays, **newDieA** and **newDieB**, to store the new configurations for Die A and Die B respectively.
   * For Die B, it simply copies the original configuration since there are no constraints on the number of spots per face.
   * For Die A, it calculates the new configuration while respecting the constraint that no face can have more than 4 spots. To maintain the same probabilities of obtaining each sum, it distributes the spots evenly among the faces, ensuring that each face has at most 4 spots. Any remaining spots are distributed among the faces as evenly as possible.
   * Finally, it puts the new configurations for Die A and Die B into the **result** map and returns it.
4. The **arrayToString** method is a utility method that converts an integer array to a string for easier printing.

OUTPUT:  
  
