Securin assessment:

PART B:

```
import java.util.ArrayList;
import java.util.HashMap;


public class Main {


    public static void main(String[] args) {
        int[] Die_A = {1, 2, 3, 4, 5, 6};
        int[] Die_B = {1, 2, 3, 4, 5, 6};


        HashMap<Integer, Double> probabilities = calculateProbabilities(Die_A, Die_B);
        System.out.println("Probabilities: " + probabilities);


        ArrayList<int[]> newDiceA = new ArrayList<>();
        ArrayList<int[]> newDiceB = new ArrayList<>();
        undoomDice(Die_A, Die_B, probabilities, newDiceA, newDiceB);


        System.out.println("New Dice A:");
        for (int[] face : newDiceA) {
            for (int spot : face) {
                System.out.print(spot + " ");
```

```
            }
            System.out.println();
        }

        System.out.println("New Dice B:");
        for (int[] face : newDiceB) {
            for (int spot : face) {
                System.out.print(spot + " ");
            }
            System.out.println();
        }
    }

    private static HashMap<Integer, Double> calculateProbabilities(int[] Die_A, int[]
Die_B) {
        HashMap<Integer, Double> probabilities = new HashMap<>();
        for (int faceA : Die_A) {
            for (int faceB : Die_B) {
                int total = faceA + faceB;
                probabilities.put(total, probabilities.getOrDefault(total, 0.0) + 1.0);
            }
        }

        int totalRolls = Die_A.length * Die_B.length;
```

```
        for (int total : probabilities.keySet()) {
            probabilities.put(total, probabilities.get(total) / totalRolls);
        }

        return probabilities;
    }

    private static void undoomDice(int[] Die_A, int[] Die_B, HashMap<Integer, Double>
probabilities,
                                   ArrayList<int[]> newDiceA, ArrayList<int[]> newDiceB) {
        for (int spotsA = 1; spotsA <= 4; spotsA++) {
            for (int spotsB = 1; spotsB <= 6; spotsB++) {
                if (spotsA + spotsB > 8) {
                    continue;
                }
                for (int faceA = 0; faceA < Math.pow(6, spotsA); faceA++) {
                    for (int faceB = 0; faceB < Math.pow(6, spotsB); faceB++) {
                        boolean valid = true;
                        for (int rollA = 0; rollA < Math.pow(6, spotsA); rollA++) {
                            for (int rollB = 0; rollB < Math.pow(6, spotsB); rollB++) {
                                int total = calculateTotal(Die_A, Die_B, faceA, faceB, spotsA,
spotsB, rollA, rollB);
                                if (!probabilities.containsKey(total)) {
                                    valid = false;
```

```
                        break;
                    }
                }
                if (!valid) {
                    break;
                }
            }
            if (valid) {
                newDiceA.add(generateFace(Die_A, spotsA, faceA));
                newDiceB.add(generateFace(Die_B, spotsB, faceB));
            }
        }
    }
}

    private static int calculateTotal(int[] Die_A, int[] Die_B, int faceA, int faceB, int spotsA,
int spotsB,
                                      int rollA, int rollB) {
        int total = 0;
        for (int i = 0; i < spotsA; i++) {
            total += Die_A[(faceA / (int) Math.pow(6, i)) % 6];
        }
```

```
        for (int i = 0; i < spotsB; i++) {
            total += Die_B[(faceB / (int) Math.pow(6, i)) % 6];
        }
        return total;
    }

    private static int[] generateFace(int[] Die, int spots, int face) {
        int[] newFace = new int[spots];
        for (int i = 0; i < spots; i++) {
            newFace[i] = Die[(face / (int) Math.pow(6, i)) % 6];
        }
        return newFace;
    }
}
```

OUTPUT:

Probabilities: {2=0.027777777777777776, 3=0.05555555555555555,
4=0.08333333333333333, 5=0.1111111111111111, 6=0.1388888888888889,
7=0.16666666666666666, 8=0.1388888888888889, 9=0.1111111111111111,
10=0.08333333333333333, 11=0.05555555555555555, 12=0.027777777777777776}