Worked Example 7.1

Rolling the Dice

Your task is to analyze whether a die is fair by counting how often the values 1, 2, …, 6 appear. Your input is a sequence of die toss values, and you should print a table with the frequencies of each die value.

Step 1 Decompose your task into steps.

Our first try at a decomposition simply echoes the problem statement:

```
Read the die values.
Count how often the values 1, 2, ..., 6 appear.
Print the counts.
```

But let's think about the task a little more. This decomposition suggests that we first read and store all die values. Do we really need to store them? After all, we only want to know how often each face value appears. If we keep an array or array list of counters, we can discard each input after incrementing the counter.

This refinement yields the following outline:

```
For each input value
    Increment the corresponding counter.
Print the counters.
```

Step 2 Choose between array lists and arrays.

There is no need to store the input values. We only need to decide how to store the counters. The number of counters is known in advance: we need one counter for each of the six face values. Because the number of elements is known in advance, we will use an array.

Actually, we will use an array of seven integers, "wasting" the element `counters[0]`. That trick makes it easy to update the counters. When reading an input value between 1 and 6, we simply execute

```
counters[value]++; // value is between 1 and 6
```

That is, we declare the array as

```
int[] counters = new int[sides + 1];
```

Why introduce a `sides` variable? Suppose you later changed your mind and wanted to investigate 12-sided dice.

Then the program can simply be changed by setting `sides` to 12.

Step 3 Determine which algorithm(s) you need.

We already discussed how the input should be processed: We read values and increment the corresponding counters.

The only remaining task is to print the counts. A typical output might look like this:

```
1: 3
2: 3
3: 2
4: 2
5: 2
6: 0
```

We haven't seen an algorithm for this exact output format. It is similar to the basic loop for printing all elements:

```
for (int element : counters)
{
   System.out.println(element);
}
```

However, that loop is not appropriate for two reasons. First, it displays the unused `0` entry. The "enhanced" for loop is no longer suitable if we want to skip that entry. We need a traditional `for` loop instead:

```
for (int i = 1; i < counters.length; i++)
{
   System.out.println(counters[i]);
}
```

This loop prints the counter values, but it doesn't quite match the sample output. We also want the corresponding face values:

```
for (int i = 1; i < counters.length; i++)
{
   System.out.printf("%2d: %4d\n", i, counters[i]);
}
```

Step 4 Use classes and methods to structure your program.

As in How To 6.1, we will use a class `DiceCounter` that collects the data and another class `DiceAnalyzer` that processes the user input. The `DiceCounter` class maintains the counts and has two methods:

- `void addValue(int value)`
- `void display()`

The `addValue` method increments the counter, and the `display` method prints all counter values, as discussed in the preceding section.

The main method of the `DiceAnalyzer` class reads the inputs and calls `addValue` for each of them. Then it calls `display`.

Step 5 Assemble and test the program.

The listing at the end of this section shows the complete program. There is one notable feature that we have not previously discussed. When updating a counter

```
counters[value]++;
```

we want to be sure that the user did not provide a wrong input which would cause an array bounds exception. Therefore, we reject inputs < 1 or $>$ `sides`.

The following table shows test cases and their expected output. To save space, we only show the counters in the output.

Test Case	Expected Output	Comment
`1 2 3 4 5 6`	`1 1 1 1 1 1`	Each number occurs once.
`1 2 3`	`1 1 1 0 0 0`	Numbers that don't appear should have counts of zero.
`1 2 3 1 2 3 4`	`2 2 2 1 0 0`	The counters should reflect how often each input occurs.
(No input)	`0 0 0 0 0 0`	This is a legal input; all counters are zero.
`0 1 2 3 4 5 6 7`	**Error**	Each input should be between 1 and 6.

Here's the complete program:

ch07/dice/DiceCounter.java

```
/**
   This class counts how often dice values occur.
*/
public class DiceCounter
{
   private int[] counters;

   /**
      Constructs a dice counter suitable for dice with a given number
      of sides.
      @param sides the number of sides of the dice to be counted
   */
   public DiceCounter(int sides)
   {
      counters = new int[sides + 1];
   }

   /**
      Adds a die value to this counter.
      @param value the value. Must be between 1 and the number of sides.
   */
   public void addValue(int value)
   {
      counters[value]++;
   }

   /**
      Displays all counts.
   */
   public void display()
   {
         for (int j = 1; j < counters.length; j++)
         {
            System.out.println(j + ": " + counters[j]);
         }
      }
   }
```

ch07/dice/DiceAnalyzer.java

```
import java.util.Scanner;

public class DiceAnalyzer
{
   public static void main(String[] args)
   {
      final int SIDES = 6;
      DiceCounter counter = new DiceCounter(SIDES);
      System.out.println("Please enter values, Q to quit:");
      Scanner in = new Scanner(System.in);
      while (in.hasNextInt())
      {
         int value = in.nextInt();
         if (1 <= value && value <= SIDES)
         {
            counter.addValue(value);
         }
         else
         {
            System.out.println(value + " is not a valid input.");
         }
      }
      counter.display();
   }
}
```

Program Run

```
Please enter values, Q to quit:
1 2 3 1 2 3 4 Q
1: 2
2: 2
3: 2
4: 1
5: 0
6: 0
```