WORKED EXAMPLE 7.1 Rolling the Dice

Problem Statement 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.

© ktsimage/iStockphoto.

Step 1 Decompose your task into steps.

Our first try at 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 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 Determine which algorithm(s) you need.

We don't have a ready-made algorithm for reading inputs and incrementing a counter, but it is straightforward to develop one. Suppose we read an input into `value`. This is an integer between 1 and 6. If we have an array `counters` of length 6, then we simply call

```
counters[value - 1]++;
```

Alternatively, we can use an array of seven integers, "wasting" the element `counters[0]`. That trick makes it easier to update the counters. When reading an input value, we simply execute

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

That is, we create the array as

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

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

© Ryan Ruffatti/iStockphoto.

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

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 3 Use methods to structure your program.

We will provide a method for each step:

- `void countInputs()`
- `void printCounters()`

The `main` method calls these methods:

```
public class DiceAnalyzer
{
   public static void main(String[] args)
   {
      final int SIDES = 6;
      Dice dice = new Dice(SIDES);
      dice.countInputs();
      dice.printCounters();
   }
}
```

The `countInputs` method reads all inputs and increments the matching counters. The `printCounters` method prints the value of the faces and counters, as already described.

Step 4 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 error. 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:

worked_example_1/Dice.java

```java
import java.util.Scanner;

/**
   This program reads a sequence of die toss values and prints how many times
   each value occurred.
*/
public class Dice
{
   private int[] counters;

   public Dice(int sides)
   {
      counters = new int[SIDES + 1]; // counters[0] is not used
   }

   public void countInputs()
   {
      System.out.println("Please enter values, Q to quit:");
      Scanner in = new Scanner(System.in);
      while (in.hasNextInt())
      {
         int value = in.nextInt();

         // Increment the counter for the input value

         if (1 <= value && value <= counters.length)
         {
            counters[value]++;
         }
         else
         {
            System.out.println(value + " is not a valid input.");
         }
      }
   }

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

worked_example_1/DiceAnalyzer

```
public class DiceAnalyzer
{
   public static void main(String[] args)
   {
      final int SIDES = 6;
      Dice dice = new Dice(SIDES);
      dice.countInputs();
      dice.printcounters();
   }
}
```

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
```