Chapter 6

Arrays

Lecture slides for:

Java Actually: A Comprehensive Primer in Programming

Khalid Azim Mughal, Torill Hamre, Rolf W. Rasmussen

Cengage Learning, 2008.

ISBN: 978-1-844480-933-2

`http://www.ii.uib.no/~khalid/jac/`

Modified: 14/11/08

Overview

- Using arrays to organize a collection of values.
- Declaring array references, creating arrays and using arrays.
- Initializing an array.
- Iterating over an array.
- Multidimensional arrays.
- Common operations on arrays.
- Generating pseudo-random numbers

Arrays

- Arrays can be used to create a *collection* of data values.
  - An array is a collection with a finite number of *elements*.
  - Elements in an array can be one of the following:
    - either *values* of a particular primitive data type,
    - or *references* to objects.
  - All elements in an array have the *same element type.*
  - An array has a *length* which corresponds to the number of elements in it, and cannot be changed after the array is created.
- In Java arrays are *objects*.
  - Java provides language construction for declaration, creation, initialization and use of arrays.

Arrays (cont.)

Declaration:

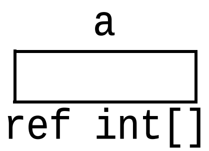
```
<elementType> <arrayName> [];
```

Alternative declaration:

```
<elementType>[] <arrayName>;
```

- `<elementType>` can be a *primitive datatype* or a *reference type* (for example, a class).
- Declaration alone creates a *reference* for the array.

`int[] a;` creates a reference to an array of *int*:



Arrays (cont.)

Creation:

- the array itself is created using the `new` operator.
- can combine the declaration with the creation:

```
<elementType>[] <arrayName> = new <elementType>[<numOfElements>];

double[] d = new double[5]; // creates an array object with 5
                            // elements of type double.
```

- can create an array and assign its reference value to a reference that is already declared:

```
int[] a;
a = new int[8];             // creates av array object with 8
                            // elements of type int
```

- Elements are always initialized to the default value for *`elementType`* when the array is created.
- Each array has a field, `length`, that indicates the number of elements in the array.
  - Value of `a.length` is equal to the number of elements array `a` can hold, i.e `8`.

Arrays (cont.)

Use:

`<arrayName> [<index>]`

- `index` is an integer expression that satisfies the following relation:
  `0 ≤ index < numOfElements`
  - `0` and (`numOfElements-1`) are lower and upper limits for `index`, respectively.
  - The value of the `index` is checked automatically during execution.
  - If `index ≥ numOfElements` or `index < 0`, an `ArrayIndexOutOfBoundsException` is thrown at runtime.
- Notation above can interpreted as variable:
  - `a[2]` indicates the 3rd element in the array `a`.

```
a[2] = 5;
a[2] = 3 * a[2];                // a[2] is assigned the value 15.
```

Example of an array

```
// Declaration and creation
int[] drawers = new int[10];
// indexed variable: drawers[0], drawers[1], ... ,
//                   drawers[9];
// Each indexed variable corresponds to a drawer.
// int drawers0, drawers1, drawers2, drawers3, drawers4,
//     drawers5, drawers6, drawers7, drawers8, drawers9;

// Explicit initialization
drawers[0] = 29;
```

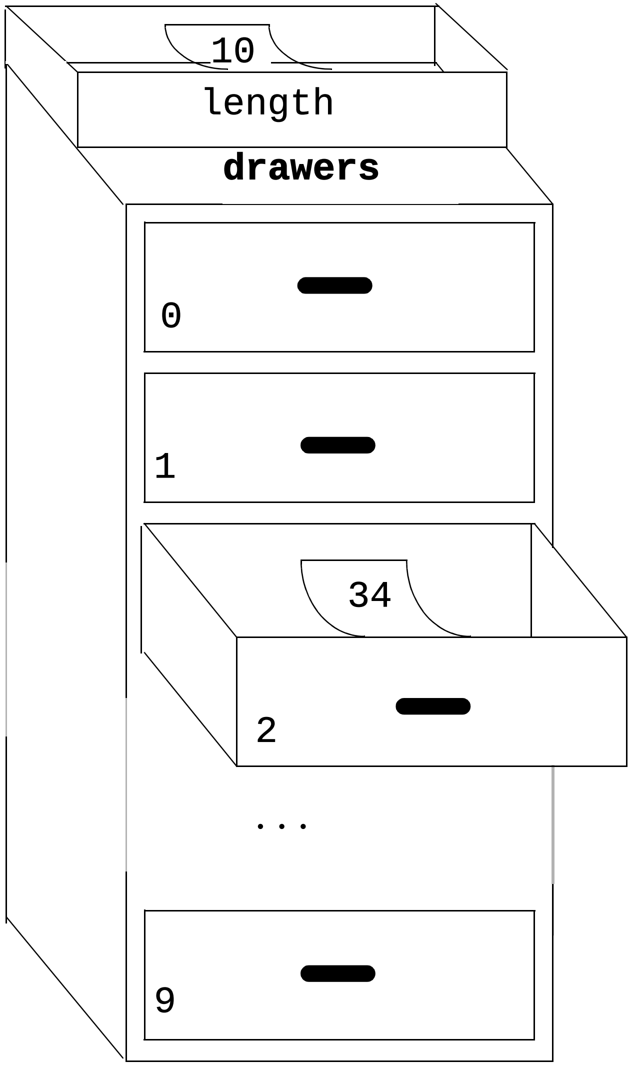
indexed variable

index

```
// use
drawers[2] = drawers[0]+5;
```

array navn

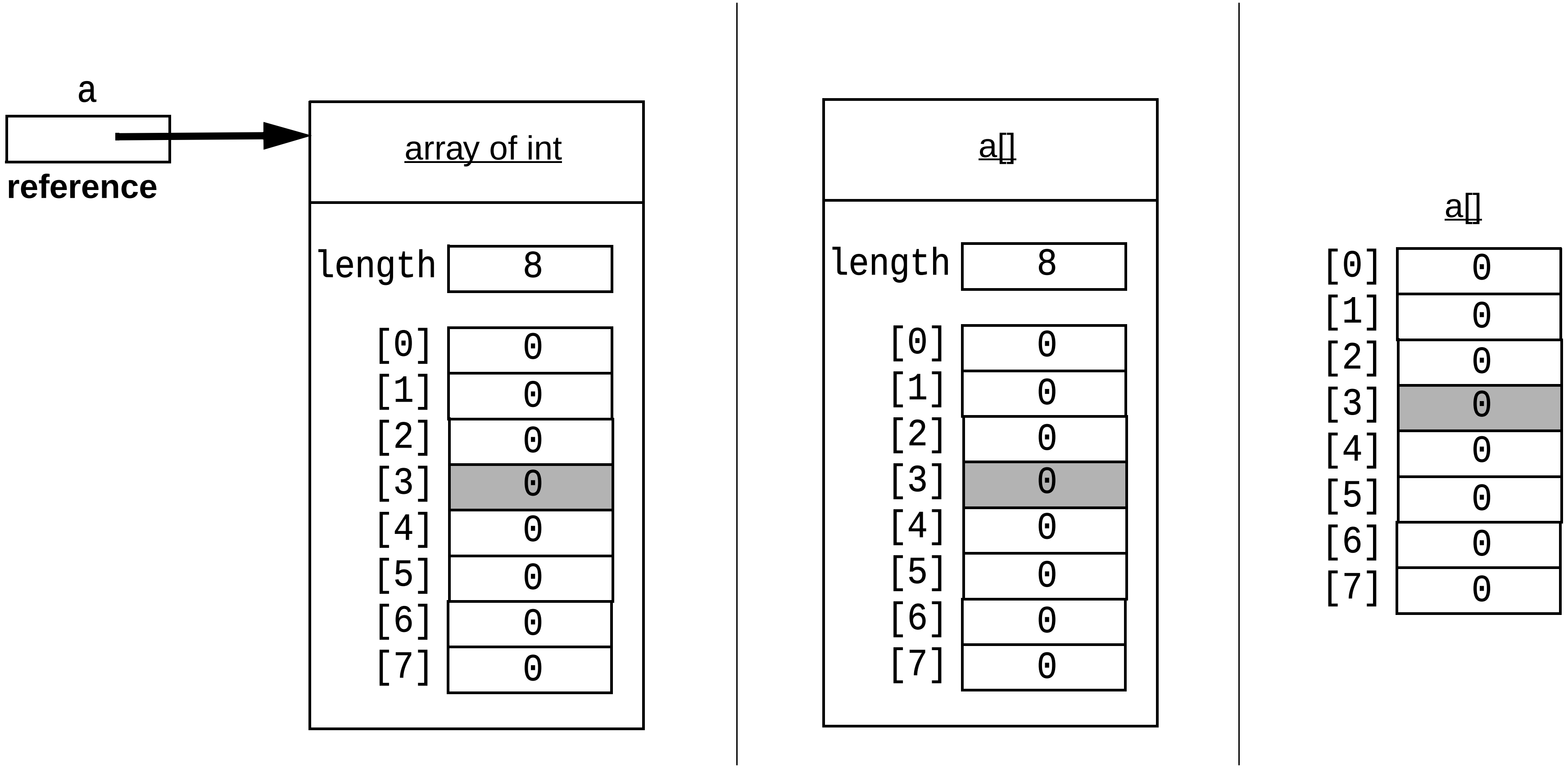
```
...
for (int i = 0; i < drawers.length; i++)
    drawers[i] = 1;

// drawers[10] does not exist!
```



Arrays: Graphical Notation

Declaration, creation and default value initialization.

```
int[] a = new int[8];
```

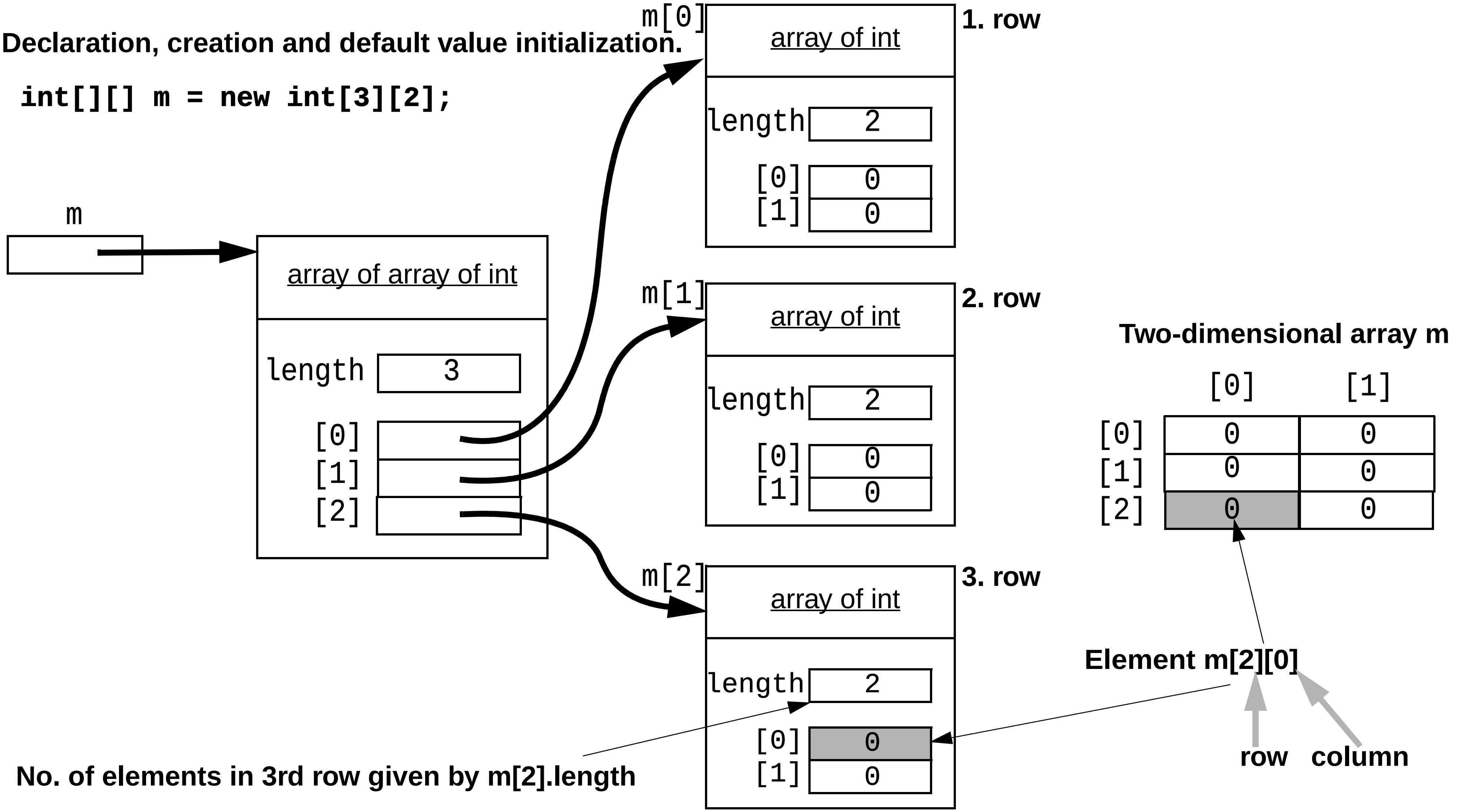


a[3] is the 4rth element in the array.

Multi-dimensional arrays: *arrays of arrays*

Declaration, creation and default value initialization.

```
int[][] m = new int[3][2];
```



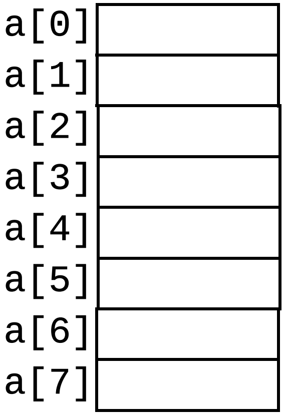
Two-dimensional array m

	[0]	[1]
[0]	0	0
[1]	0	0
[2]	0	0

No. of elements in 3rd row given by m[2].length

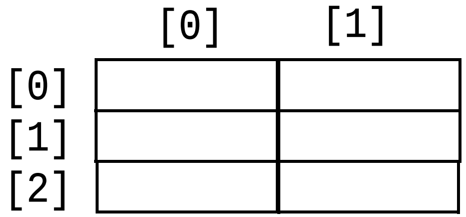
Storing of array elements

One-dimensional array a

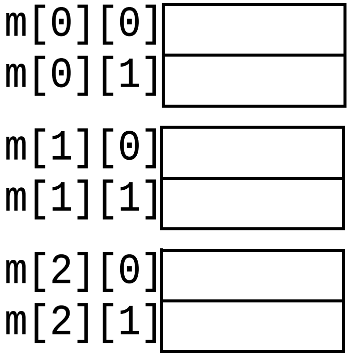


Elements of the array a are stored *sequentially*.

Two-dimensional array m



is stored as



Elements of the array m are stored *row-wise*.

Programming pattern – some simple algorithms

- Find the minimum value in an array:

```
int[] array = new int[N_ELEMENTS];
// Assume the array is initialized with values.
int minvalue = array[0];
for (int counter = 1; counter < array.length; ++counter) {
    if (array[counter] < minvalue) {
        minvalue = array[counter];
    }
}
```

- Find the minimum value in a 2-dimensional array:

```
int[][] matrix = new int[N_ELEMENTS][M_ELEMENTS]; // N x M matrix
// Assume the array is initialized with values.
int minvalue = matrix[0][0];
for (int counter1 = 0; counter1 < matrix.length; ++counter1) {
    // Find the minimum value in matrix[counter1];
    for (int counter2 = 0; counter2 < matrix[counter1].length; ++counter2) {
        if (matrix[counter1][counter2] < minvalue) {
            minvalue = matrix[counter1][counter2];
        }
    }
}
```

More on initializing of arrays

- Explicit initialization in the declaration of a one-dimensional array.

```
// integer array with 8 elements
int[] array = {1, 3, 49, 55, 58, 41, 52, 3146}; // declaration + initialization
array[] = {1, 3, 49, 55, 58, 41, 52, 3146};     // Compile-time error
array = {1, 3, 49, 55, 58, 41, 52, 3146};       // Compile-time error
```

- Explicit initialization in the declaration of a multi-dimensional array

```
double identityMatrix[][] = { // A 4 x 4 floating-point matrix:
    {1.0, 0.0, 0.0, 0.0 }, // 1. row initialization
    {0.0, 1.0, 0.0, 0.0 }, // 2. row initialization
    {0.0, 0.0, 1.0, 0.0 }, // 3. row initialization
    {0.0, 0.0, 0.0, 1.0 }  // 4. row initialization
};
// An array with 4 strings:
String[] pets = {"crocodiles", "elephants", "crocophants", "elediles"};
char[] charArray = {'a', 'h', 'a'}; // An array with 3 characters:
```

- Arrays in a multi-dimensional array can have different lengths:

```
double[][] matrix = new double[4][]; // left-most index always specified
for (int counter1 = 0; counter1 < matrix.length; counter1++) {
    matrix[counter1] = new double[counter1+1];
}
```

Histogram

Problem: Write a program to read floating-point numbers (that represent weights of children), groups them in weight groups and prints a histogram.

```
Type the weights (0-199). One weight per line. End with -1.
11.2
13
12.3
82.4
12.9
...
-1
Weight : Frequency
...
10   : ***
11   : *
12   : **
...
82   : *
83   :
...
199  : *****
```

Data structure:

- An array is used to count the weights.
- Index in the array represents a weight group.
- Value of an element in the array represents the number of children in this weight group, i.e. the frequency.

Algorithm:

- *Read the weights and place them in the correct weight group.*
- *Find the minimum and the maximum weight register.*
- *For each weight group (from minimum to maximum) print the number of stars that represent the frequency.*

Histogram (cont.)

```
import java.util.Scanner;
public class Histogram {
  public static void main(String[] args) {
    int N_ELEMENTS = 200;
    int[] histArray = new int[N_ELEMENTS];
    Scanner keyboard = new Scanner(System.in);

    // Read the weights
    System.out.println("Type the weights (0-199). One weight per line. End with -1.");
    int weight = (int) Math.round(keyboard.nextDouble());
    while (weight >= 0 && weight < histArray.length) {
      histArray[weight]++;
      weight = (int) Math.round(keyboard.nextDouble());
    }

    // Find the index of the element with minimun weight
    int minIndex;
    for (minIndex = 0;
         (minIndex < histArray.length) && (histArray[minIndex] == 0);
         ++minIndex);
    // and index of the element with the maximum weight.
    int maxIndex;
    for (maxIndex = histArray.length -1;
         (maxIndex >= 0) && (histArray[maxIndex] == 0);
         --maxIndex);

    // Print histogram
    System.out.println("Weight\t:\tFrequency");
    for (weight = minIndex; weight <= maxIndex; weight++) {
      System.out.print(weight + "\t:\t");

      for (int star = 1; star <= histArray[weight]; star++) {
        System.out.print("*");
      }

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

Enhanced `for` loop: `for(:)`

- If we are only interested in *reading* all the values in a collection one by one, we can use the enhanced `for` loop.
- The loop cannot be used to change the values in the collection.

Syntax:

```
for (loop variable declaration : collection)
  loop body
```

Example:

```
Integer[] ageArray = { 12, 65, 45, 60, 70, 45 };
int numOver60 = 0;
for (int age : ageArray) { // Type of loop variable is component type
  if (age >= 60) {
    numOver60++;
  }
}
System.out.println("Numbers over 60: " + numOver60);
```

Program output:

```
Numbers over 60: 3
```

Pseudo-random Number Generator

- To generate a random number we can use the class `java.util.Random`.
- Such number are called *pseudo-random numbers*, as they are not really random.

How to generate pseudo-random numbers in Java:

```
Random generator = new Random(); // Create an object of class Random
```

- Call the method `nextInt()` repeatedly on the `Random`-object:

```
int newNumber = generator.nextInt(); // random number in the interval
                                     // [-2^31, 2^31-1]
```

- We can specify an argument *n* in the call to the method `nextInt()` to return an integer number in the interval [0, *n*-1].

```
newNumber = generator.nextInt(11);      // random number in [0, 10]
newNumber = 2 + generator.nextInt(11);  // random number in [2, 12]
newNumber = 2 + 3*generator.nextInt(5); // random number in
                                        // {2, 5, 8, 11, 14}
```

Example: pseudo- number generator

```
import java.util.Scanner;
// Simulates dice throw using a pseudo-random number generator
// and computes the frequency of each dice value (1-6)
public class DiceStats2 {
  public static void main(String[] args) {

    // Array for counting the frequency of each dice value.
    int[] frequency = new int[6];

    Scanner keyboard = new Scanner(System.in);
    System.out.print("Enter the number of times to throw the dice: ");
    int noOfThrows = keyboard.nextInt();
    for (int i = 1; i <= noOfThrows; i++) {
      int diceValue = (int)(6.0*Math.random()) + 1;  // random numbers 1-6
      System.out.println(i + ". dice value: " + diceValue);
      // Increment the frequency counter for this throw.
      frequency[diceValue-1]++;
    }
    for (int diceValue = 1; diceValue <= 6; diceValue++) {
      System.out.println("No. of times the dice value is " + diceValue + ": "
                          + frequency[diceValue-1]);
    }
  }
}
```

Program out:

```
Enter the number of times to throw the dice: 3000
1. dice value: 4
2. dice value: 4
...
2999. dice value: 2
3000. dice value: 5
No. of times the dice value is 1: 493
No. of times the dice value is 2: 523
No. of times the dice value is 3: 511
No. of times the dice value is 4: 492
No. of times the dice value is 5: 515
No. of times the dice value is 6: 466
```