Chapter 3
Arrays and methods

Methods

1 Introduction

Suppose that you need to find the sum of integers from 1 to 10, from 20 to 37, and from 35 to 49, respectively.

You may write the code as follows:

```
int sum = 0;
for (int i = 1; i <= 10; i++)
  sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 37; i++)
  sum += i;
System.out.println("Sum from 20 to 37 is " + sum);

sum = 0;
for (int i = 35; i <= 49; i++)
  sum += i;
System.out.println("Sum from 35 to 49 is " + sum);
```

very similar
except the starting and ending integers are different.

1 Introduction

Write the common code once and reuse it

```
1  public static int sum(int i1, int i2) {
2    int result = 0;
3    for (int i = i1; i <= i2; i++)
4      result += i;
5
6    return result;
7  }
8
9  public static void main(String[] args) {
10   System.out.println("Sum from 1 to 10 is " + sum(1, 10));
11   System.out.println("Sum from 20 to 37 is " + sum(20, 37));
12   System.out.println("Sum from 35 to 49 is " + sum(35, 49));
13 }
```

2 Defining a Method

```
modifier returnValueType methodName(list of parameters) {
    // Method body;
}
```

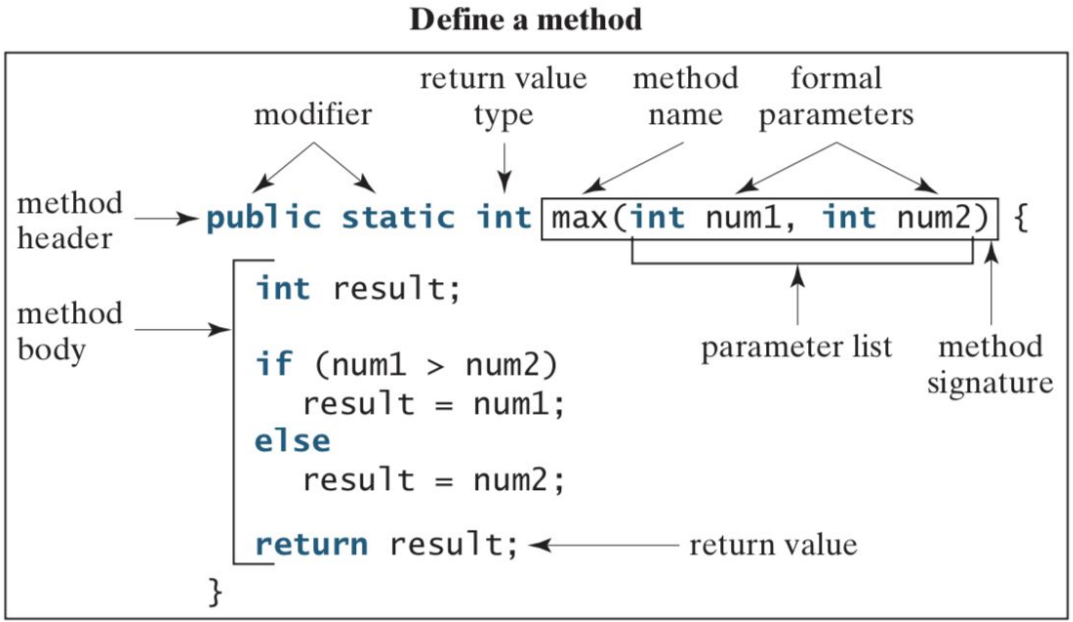


FIGURE 6.1 A method definition consists of a method header and a method body.

3 Calling a Method

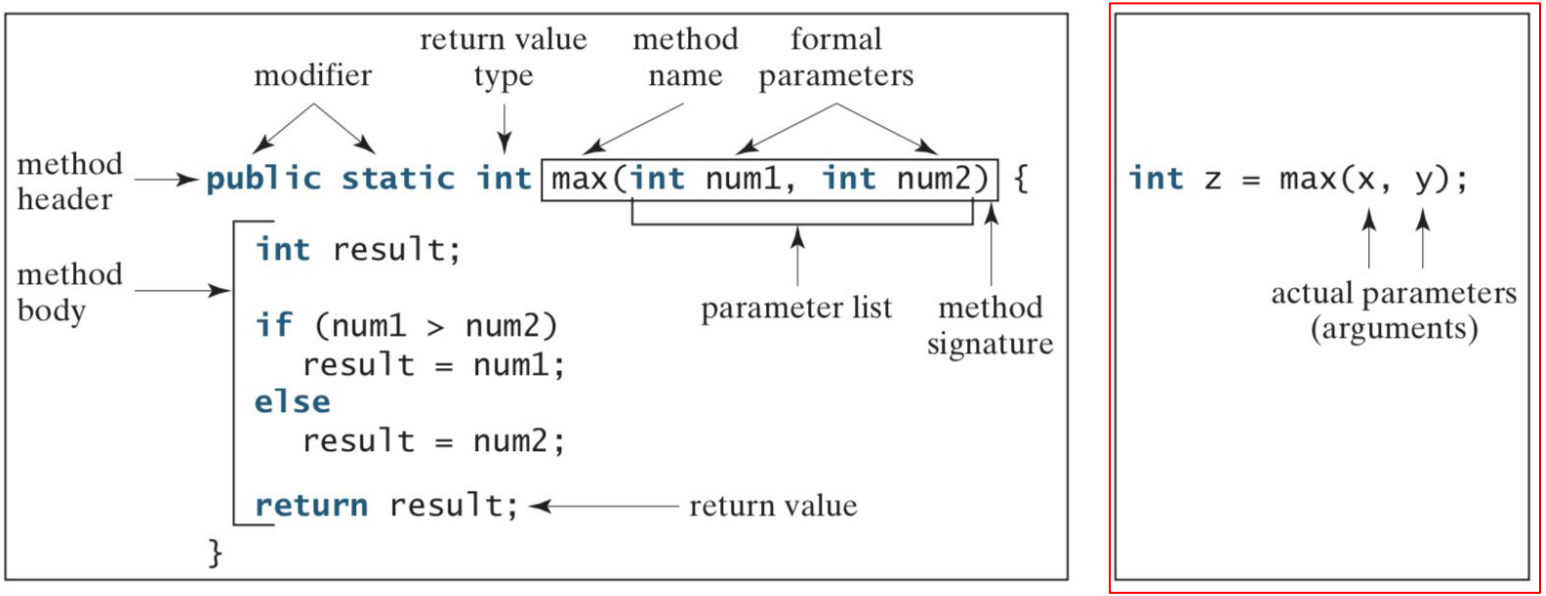


FIGURE 6.1 A method definition consists of a method header and a method body.

3 Calling a Method

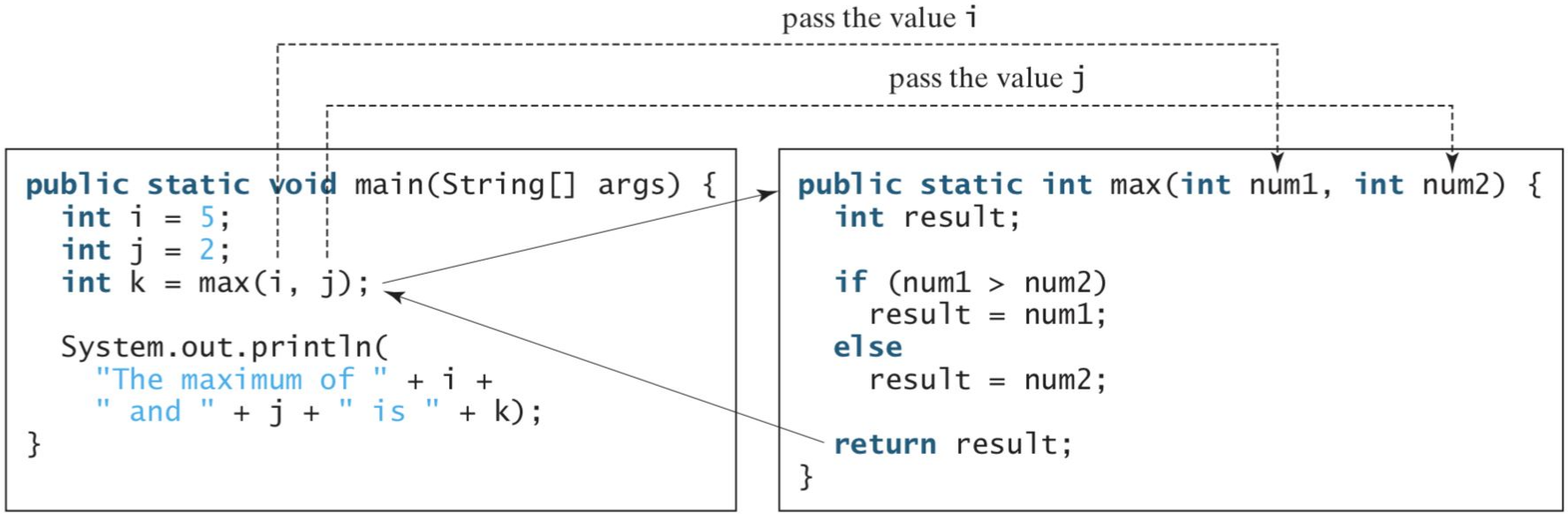
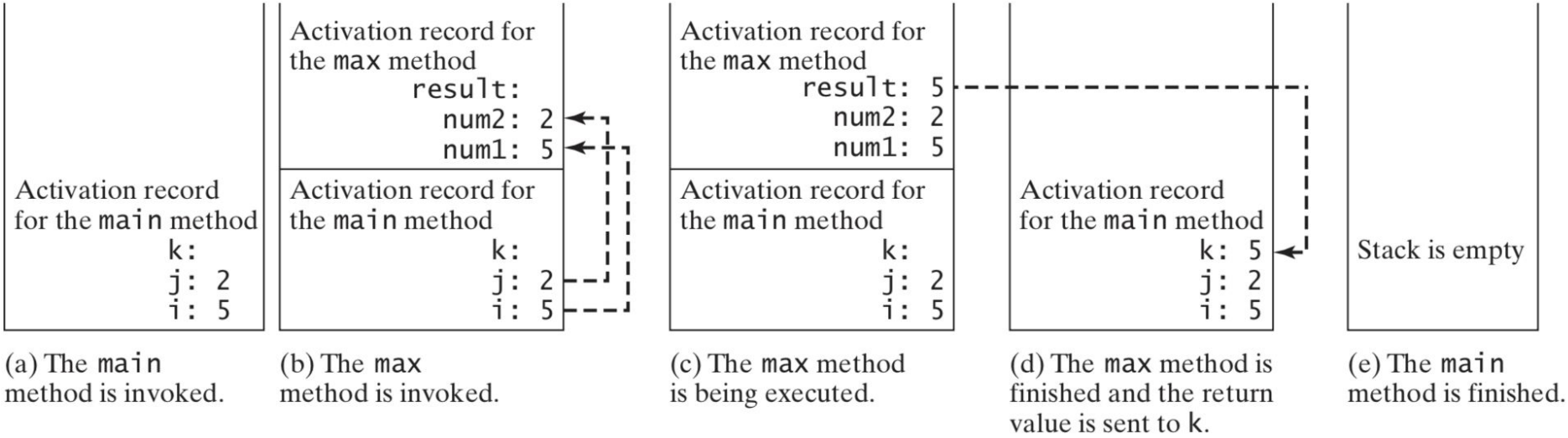


FIGURE 6.2 When the `max` method is invoked, the flow of control transfers to it. Once the `max` method is finished, it returns control back to the caller.

3 Calling a Method



(a) The `main` method is invoked.

(b) The `max` method is invoked.

(c) The `max` method is being executed.

(d) The `max` method is finished and the return value is sent to `k`.

(e) The `main` method is finished.

FIGURE 6.3 When the `max` method is invoked, the flow of control transfers to the `max` method. Once the `max` method is finished, it returns control back to the caller.

4 void Method Example

A void method does not return a value.

LISTING 6.2 TestVoidMethod.java

```
1  public class TestVoidMethod {
2    public static void main(String[] args) {
3      System.out.print("The grade is ");
4      printGrade(78.5);
5
6      System.out.print("The grade is ");
7      printGrade(59.5);
8    }
9
10   public static void printGrade(double score) {
11     if (score >= 90.0) {
12       System.out.println('A');
13     }
14     else if (score >= 80.0) {
15       System.out.println('B');
16     }
17     else if (score >= 70.0) {
18       System.out.println('C');
19     }
20     else if (score >= 60.0) {
21       System.out.println('D');
22     }
23     else {
24       System.out.println('F');
25     }
26   }
27 }
```

5 Passing Arguments by Values

The arguments are passed by value to parameters when invoking a method.

LISTING 6.5 TestPassByValue.java

```
1  public class TestPassByValue {
2    /** Main method */
3    public static void main(String[] args) {
4      // Declare and initialize variables
5      int num1 = 1;
6      int num2 = 2;
7
8      System.out.println("Before invoking the swap method, num1 is " +
9        num1 + " and num2 is " + num2);
10
11     // Invoke the swap method to attempt to swap two variables
12     swap(num1, num2);
13
14     System.out.println("After invoking the swap method, num1 is " +
15       num1 + " and num2 is " + num2);
16   }
17
18   /** Swap two variables */
19   public static void swap(int n1, int n2) {
20     System.out.println("\tInside the swap method");
21     System.out.println("\t\tBefore swapping, n1 is " + n1
22       + " and n2 is " + n2);
23
24     // Swap n1 with n2
25     int temp = n1;
26     n1 = n2;
27     n2 = temp;
28
29     System.out.println("\t\tAfter swapping, n1 is " + n1
30       + " and n2 is " + n2);
31   }
32 }
```

5 Passing Arguments by Values

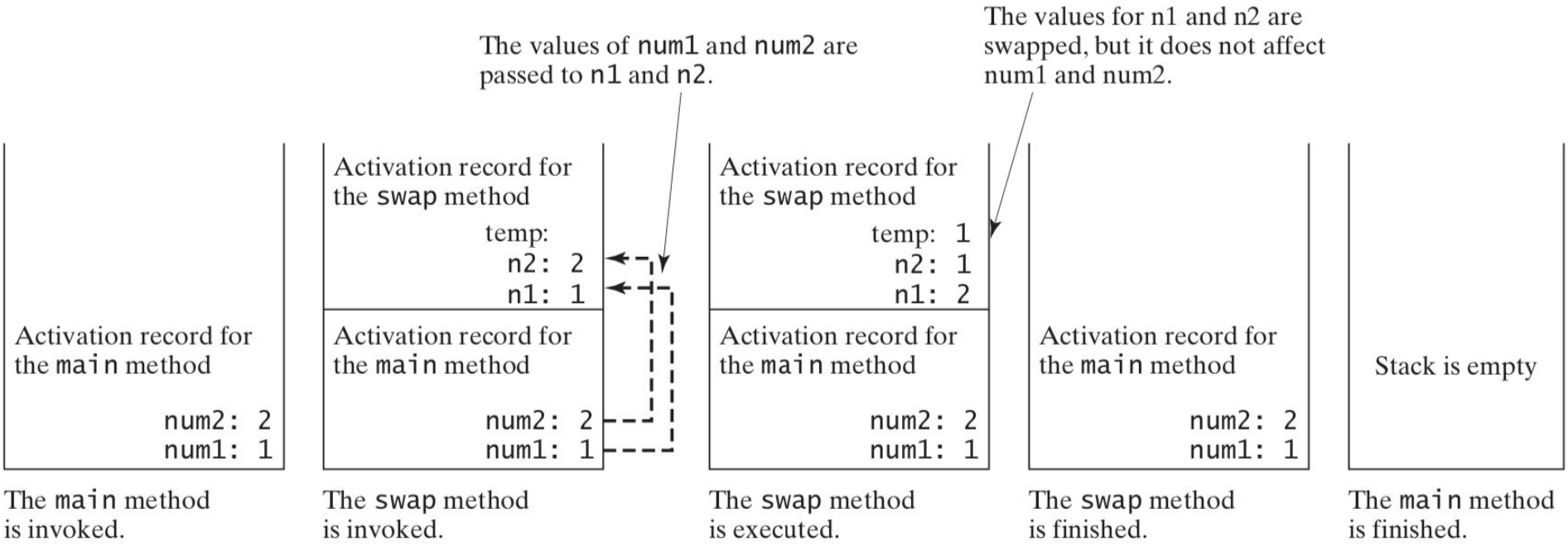


FIGURE 6.4 The values of the variables are passed to the method's parameters.

7 Overloading Methods

Overloading methods: the methods with the **same name** with **different signatures**.

LISTING 6.9 TestMethodOverloading.java

```
1  public class TestMethodOverloading {
2    /** Main method */
3    public static void main(String[] args) {
4      // Invoke the max method with int parameters
5      System.out.println("The maximum of 3 and 4 is "
6        + max(3, 4));
7
8      // Invoke the max method with the double parameters
9      System.out.println("The maximum of 3.0 and 5.4 is "
10       + max(3.0, 5.4));
11
12     // Invoke the max method with three double parameters
13     System.out.println("The maximum of 3.0, 5.4, and 10.14 is "
14       + max(3.0, 5.4, 10.14));
15   }
16
17   /** Return the max of two int values */
18   public static int max(int num1, int num2) {
19     if (num1 > num2)
20       return num1;
21     else
22       return num2;
23   }
24
25   /** Find the max of two double values */
26   public static double max(double num1, double num2) {
27     if (num1 > num2)
28       return num1;
29     else
30       return num2;
31   }
32
33   /** Return the max of three double values */
34   public static double max(double num1, double num2, double num3) {
35     return max(max(num1, num2), num3);
36   }
37 }
```

9 The Scope of Variables

The scope of a variable is the part of the program where the variable can be referenced.

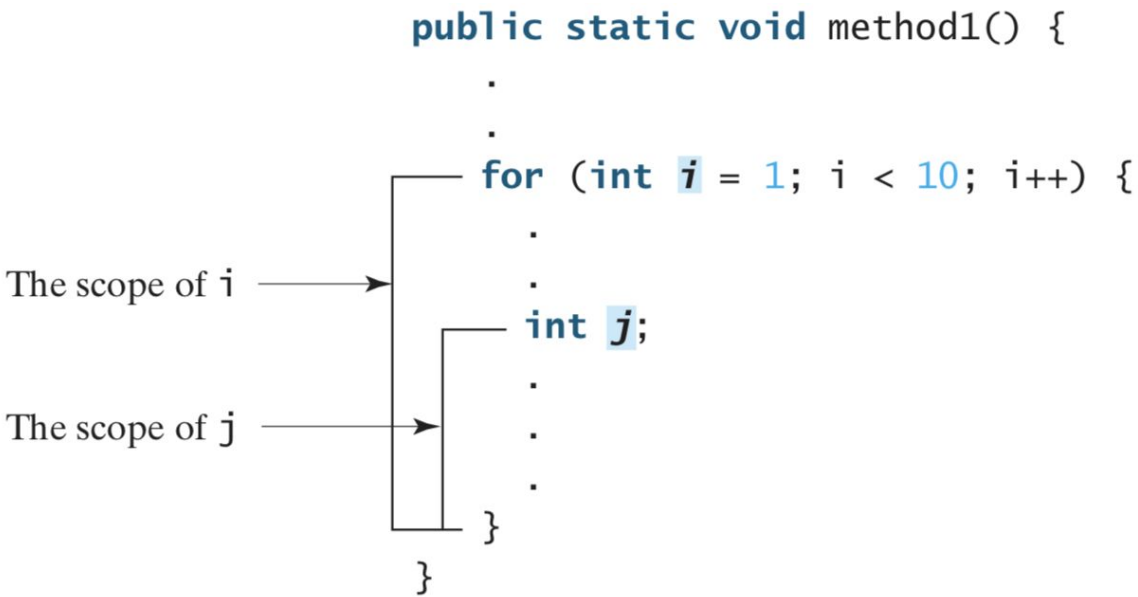
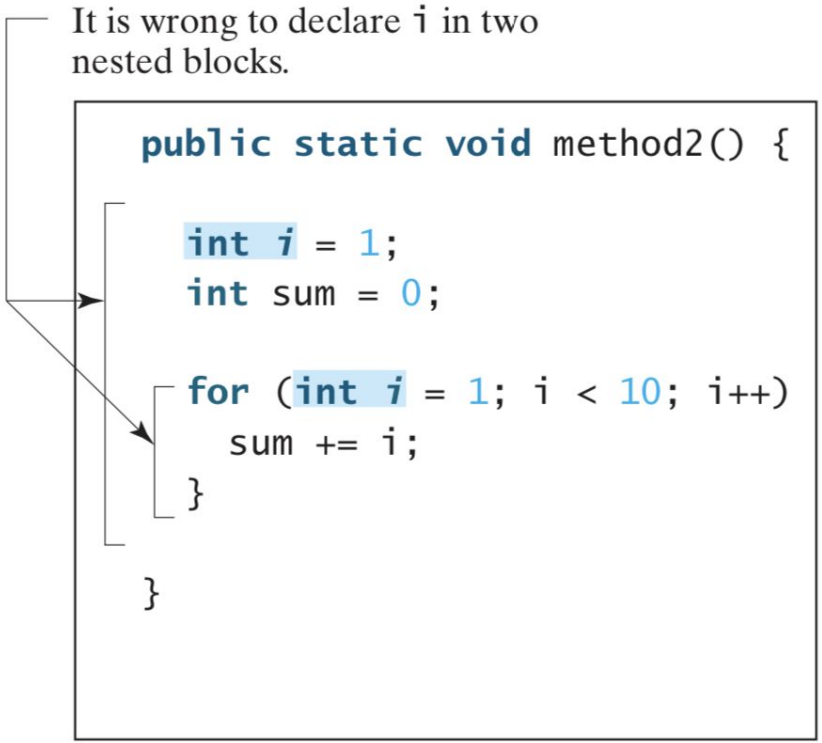


FIGURE 6.5 A variable declared in the initial action part of a `for`-loop header has its scope in the entire loop.

9 The Scope of Variables



SINGLE-DIMENSIONAL ARRAYS

1 Introduction

A single array variable can reference a large collection of data.

2 Array Basics

- Once an array is created, its **size** is **fixed**.
- An array reference variable is used to access the elements in an array using **an index**.
- Declaring Array Variables

elementType[] arrayRefVar;
OR
elementType arrayRefVar[];

double[] myList;

2 Array Basics

- Creating Arrays: use the new operator and assign its reference to the variable
  arrayRefVar = new elementType[arraySize];
  myList = new double[10];

- Declaring and creating arrays in one statement:
  double[] myList = new double[10];

2 Array Basics

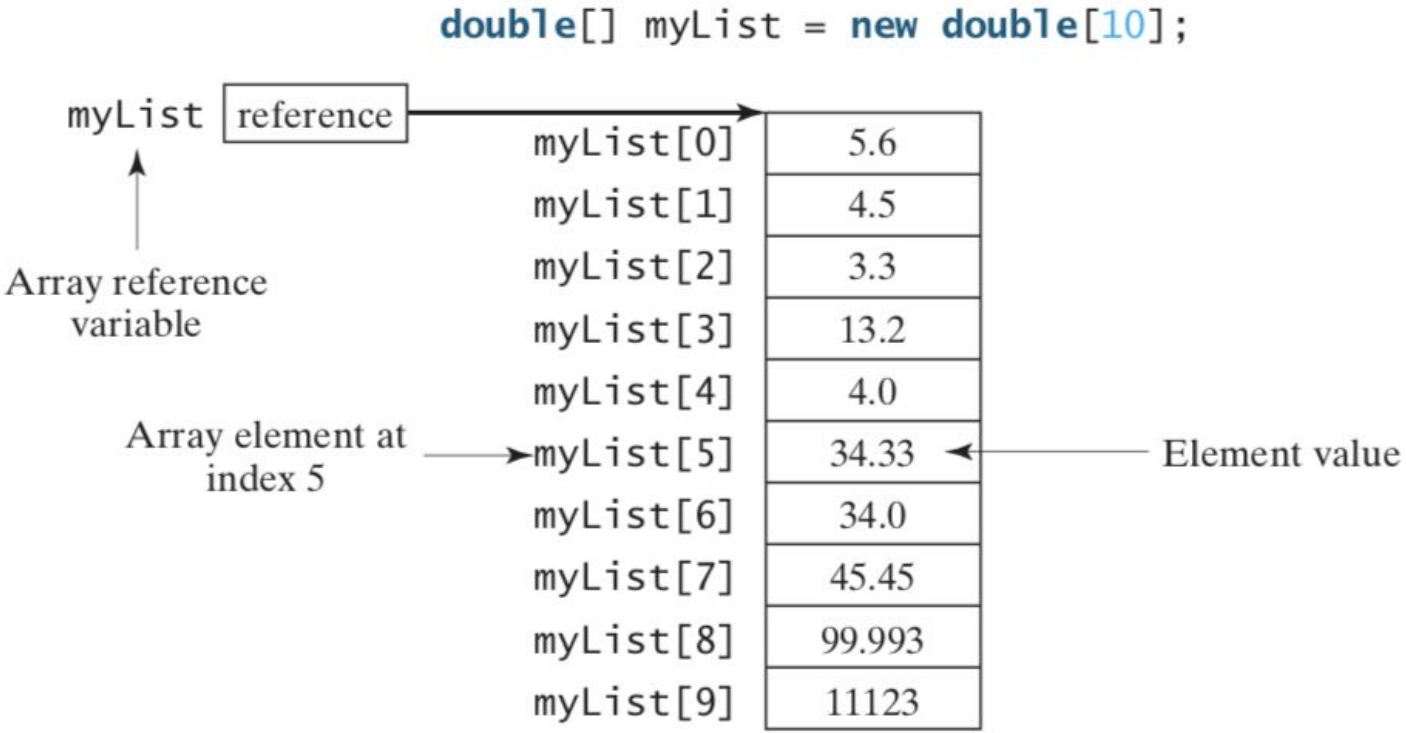


FIGURE 7.1 The array `myList` has ten elements of `double` type and `int` indices from `0` to `9`.

2 Array Basics

- Array Size and Default Values
  - The size of an array cannot be changed after the array is created.
  - Size can be obtained using arrayRefVar.length.
  - Default Values:
    - **0**: **numeric primitive data types**,
    - **\u0000**: **char** types
    - **false**: **boolean types**.

2 Array Basics

- Accessing Array Elements
  - The array elements are accessed through the index.
  - [0 arrayRefVar.length-1]
  - Syntax:
    - arrayRefVar[index];

2 Array Basics

Array Initializers: declare, create, and initialize the array all in one statement

```
elementType[] arrayRefVar = {value0, value1, ..., valuek};
```

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

Equivalent to

```
double[] myList = new double[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

2 Array Basics

Processing Arrays

Use "for" loop:

■ All of the elements in an array are of the same type. They are evenly processed in the same fashion repeatedly using a loop.

■ Since the size of the array is known, it is natural to use a for loop.

```
double[] myList = new double[10];
for (int i = 0; i < myList.length; i++) {
        myList[i] = Math.random() * 100;
}
```

2 Array Basics

Foreach Loops

```
for (elementType element: arrayRefVar) {
        // Process the element
}
```

```
for (double e: myList) {
        System.out.println(e);
}
```

3 Copying Arrays

Copy the contents of one array into another:

- Copy the array's individual elements into the other array.

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];
for (int i = 0; i < sourceArray.length; i++) {
   targetArray[i] = sourceArray[i];
 }
```

- System.arraycopy

```
System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);
```

4 Passing Arrays to Methods

When passing an array to a method, the reference of the array is passed to the method.

```
public class Test {
  public static void main(String[] args) {
    int x = 1; // x represents an int value
    int[] y = new int[10]; // y represents an array of int values

    m(x, y); // Invoke m with arguments x and y

    System.out.println("x is " + x);
    System.out.println("y[0] is " + y[0]);
  }

  public static void m(int number, int[] numbers) {
    number = 1001; // Assign a new value to number
    numbers[0] = 5555; // Assign a new value to numbers[0]
  }
}
```

x is 1
y[0] is 5555

5 Returning an Array from a Method

```
public static int[] reverse(int[] list) {
        int[] result = new int[list.length];
        for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
              result[j] = list[i];
        }
         return result;
}

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

6 Variable-Length Argument Lists

A variable number of arguments of the same type can be passed to a method and treated as an array.

LISTING 7.5 VarArgsDemo.java

```
1  public class VarArgsDemo {
2    public static void main(String[] args) {
3      printMax(34, 3, 3, 2, 56.5);
4      printMax(new double[]{1, 2, 3});
5    }
6
7    public static void printMax(double... numbers) {
8      if (numbers.length == 0) {
9        System.out.println("No argument passed");
10       return;
11     }
12
13     double result = numbers[0];
14
15     for (int i = 1; i < numbers.length; i++)
16       if (numbers[i] > result)
17         result = numbers[i];
18
19     System.out.println("The max value is " + result);
20   }
21 }
```

7 Searching Arrays

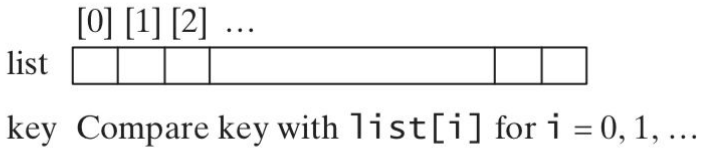
- Linear Search
- Binary Search

7 Searching Arrays

The Linear Search Approach

LISTING 7.6 LinearSearch.java

```
1 public class LinearSearch {
2   /** The method for finding a key in the list */
3   public static int linearSearch(int[] list, int key) {
4     for (int i = 0; i < list.length; i++) {
5       if (key == list[i])
6         return i;
7     }
8     return -1;
9   }
10 }
```



7 Searching Arrays

The Binary Search Approach

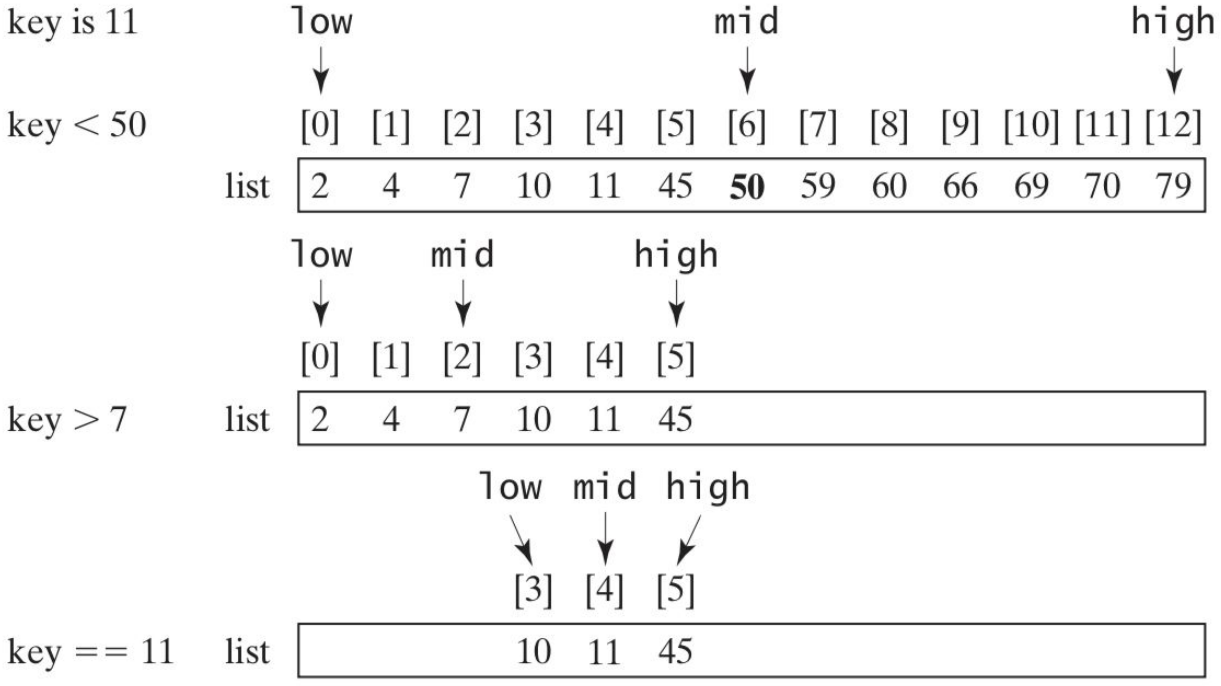


FIGURE 7.9 Binary search eliminates half of the list from further consideration after each comparison.

7 Searching Arrays

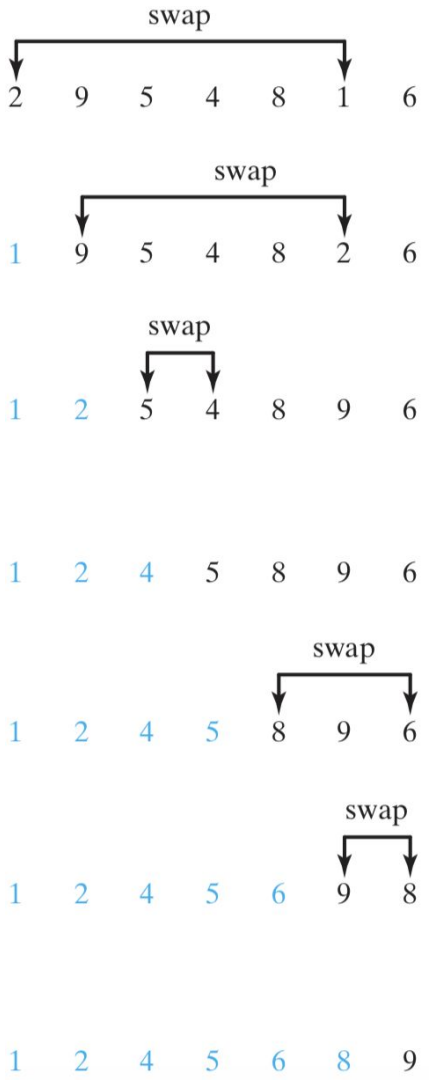
The Binary Search Approach

LISTING 7.7 BinarySearch.java

```
1  public class BinarySearch {
2    /** Use binary search to find the key in the list */
3    public static int binarySearch(int[] list, int key) {
4      int low = 0;
5      int high = list.length - 1;
6
7      while (high >= low) {
8        int mid = (low + high) / 2;
9        if (key < list[mid])
10         high = mid - 1;
11       else if (key == list[mid])
12         return mid;
13       else
14         low = mid + 1;
15     }
16
17     return -low - 1; // Now high < low, key not found
18   }
19 }
```

8 Sorting Arrays

- Selection sort finds the smallest number in the list and swaps it with the first element.
- It then finds the smallest number remaining and swaps it with the second element,
- and so on, until only a single number remains.



8 Sorting Arrays

LISTING 7.8 SelectionSort.java

```
1  public class SelectionSort {
2    /** The method for sorting the numbers */
3    public static void selectionSort(double[] list) {
4      for (int i = 0; i < list.length - 1; i++) {
5        // Find the minimum in the list[i..list.length-1]
6        double currentMin = list[i];
7        int currentMinIndex = i;
8
9        for (int j = i + 1; j < list.length; j++) {
10         if (currentMin > list[j]) {
11           currentMin = list[j];
12           currentMinIndex = j;
13         }
14       }
15
16       // Swap list[i] with list[currentMinIndex] if necessary
17       if (currentMinIndex != i) {
18         list[currentMinIndex] = list[i];
19         list[i] = currentMin;
20       }
21     }
22   }
23 }
```

9 The Arrays Class

The java.util.Arrays class contains useful methods for common array operations such as sorting and searching.

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers); // Sort the whole array
System.out.println("1. Index is " + java.util.Arrays.binarySearch(numbers, 11));
java.util.Arrays.fill(numbers, 5); // Fill 5 to the whole array
System.out.println(Arrays.toString(numbers));
```

10 Command-Line Arguments

The main method can receive string arguments from the command line.

```
public class TestMain {
	public static void main(String[] args) {
		for (int i = 0; i < args.length; i++)
			System.out.println(args[i]);
	}
}
```

```
java TestMain "First num" alpha 53
```

MULTIDIMENSIONAL ARRAYS

1 Introduction

Data in a table or a matrix
can be represented
using a two-dimensional array.

```
double[][] distances = {
{0, 983, 787, 714, 1375, 967, 1087},
{983, 0, 214, 1102, 1763, 1723, 1842},
{787, 214, 0, 888, 1549, 1548, 1627},
{714, 1102, 888, 0, 661, 781, 810}, {
1375, 1763, 1549, 661, 0, 1426, 1187},
{967, 1723, 1548, 781, 1426, 0, 239},
{1087, 1842, 1627, 810, 1187, 239, 0},
};
```

Distance Table (in miles)

	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1763	1723	1842
New York	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1763	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0

2 Two-Dimensional Array Basics

Declaring Variables of Two-Dimensional Arrays and Creating Two-Dimensional Arrays

```
elementType[][] arrayRefVar;
```

OR

```
elementType arrayRefVar[][]; // Allowed, but not preferred
```

```
int[][] matrix;
```

OR

```
int matrix[][]; // This style is allowed, but not preferred
```

```
matrix = new int[5][5];
```

2 Two-Dimensional Array Basics

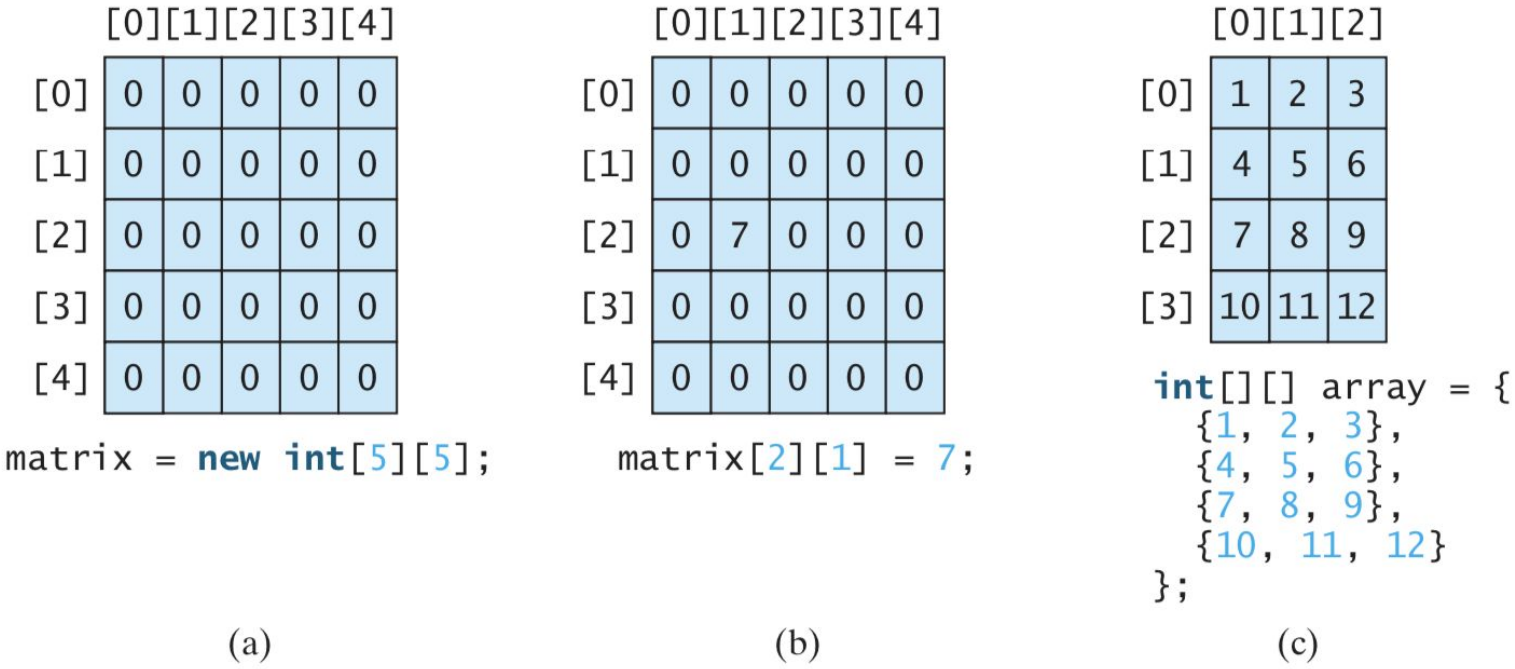


FIGURE 8.1 The index of each subscript of a two-dimensional array is an `int` value, starting from `0`.

2 Two-Dimensional Array Basics

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

Equivalent

```
int[][] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

2 Two-Dimensional Array Basics

Obtaining the Lengths of Two-Dimensional Arrays

x[0].length

x[1].length

. . . ,

x[x.length-1].length

x = new int[3][4]

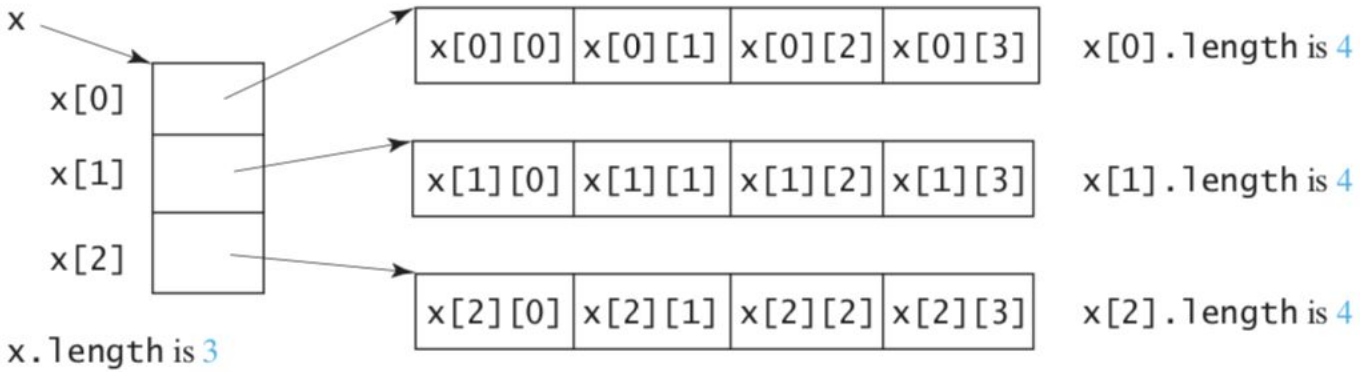
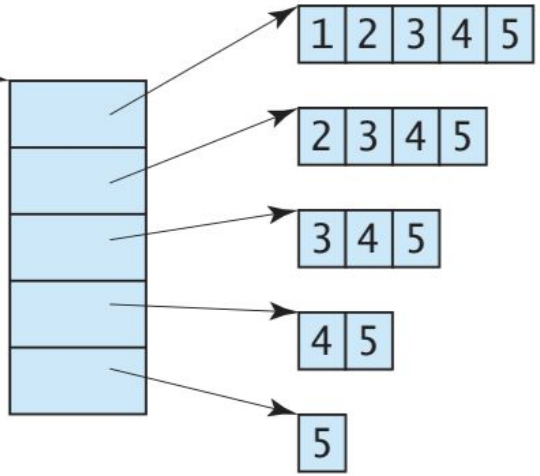


FIGURE 8.2 A two-dimensional array is a one-dimensional array in which each element is another one-dimensional array.

2 Two-Dimensional Array Basics

Ragged Arrays: array with rows whose have different lengths

```
int[][] triangleArray = {
   {1, 2, 3, 4, 5},
   {2, 3, 4, 5},
   {3, 4, 5},
   {4, 5},
   {5}
};
```



3 Processing Two-Dimensional Arrays

Nested for loops are often used to process a two-dimensional array.

```
int[][] matrix = new int[10][10];
for (int row = 0; row < matrix.length; row++) {
        for (int column = 0; column < matrix[row].length; column++) {
                matrix[row][column] = (int)(Math.random() * 100); }
}
```

4 Passing Two-Dimensional Arrays to Methods

The reference of the array is passed to the method.

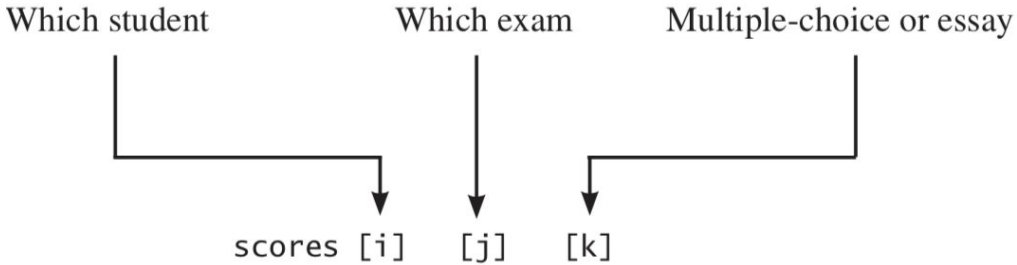
LISTING 8.1 PassTwoDimensionalArray.java

```
1  import java.util.Scanner;
2
3  public class PassTwoDimensionalArray {
4    public static void main(String[] args) {
5      int[][] m = getArray(); // Get an array
6
7      // Display sum of elements
8      System.out.println("\nSum of all elements is " + sum(m));
9    }
10
11   public static int[][] getArray() {
12     // Create a Scanner
13     Scanner input = new Scanner(System.in);
14
15     // Enter array values
16     int[][] m = new int[3][4];
17     System.out.println("Enter " + m.length + " rows and "
18       + m[0].length + " columns: ");
19     for (int i = 0; i < m.length; i++)
20       for (int j = 0; j < m[i].length; j++)
21         m[i][j] = input.nextInt();
22
23     return m;
24   }
25
26   public static int sum(int[][] m) {
27     int total = 0;
28     for (int row = 0; row < m.length; row++) {
29       for (int column = 0; column < m[row].length; column++) {
30         total += m[row][column];
31       }
32     }
33
34     return total;
35   }
36 }
```

5 Multidimensional Arrays

A two-dimensional array consists of an array of one-dimensional arrays
A three- dimensional array consists of an array of two-dimensional arrays

Eg. use a three-dimensional array to store exam scores for a class of six students with five exams, and each exam has two parts (multiple-choice and essay)




```
double[][][] scores = {
{{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
{{4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
{{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
{{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
{{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
{{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}};
```