Chapter 8

Array Data Types

Processing collections of objects

Outline

Mathematical sequences and subscript notation
Declaring and constructing arrays of primitive type
Sequential array processing
Declaring and constructing arrays of object type
String arrays and command-line arguments
Loan repayment and investment table examples
Arrays as method arguments and return values
Maximum array element algorithm
Linear search algorithm
Bubble sort algorithm
Efficient evaluation of a polynomial
A line graph using arrays
Two-dimensional arrays
Card deck application
8.1 Introduction

In this chapter we introduce the array data types to organize a sequence of data items so that they can be accessed using an index. The concept of an array is related to the subscript notation used for sequences in mathematics. Java arrays are objects so we first learn how to construct arrays of primitive types such as int and double and then we introduce arrays of object types (reference types). Arrays are designed to be processed using loops, especially the for-loop, so we discuss some array processing models and applications.

Some standard array processing algorithms are introduced: finding the maximum and minimum values in an array, searching for an element in an array using the linear search algorithm, sorting array elements in increasing order using the bubble sort algorithm, and the efficient evaluation of a polynomial.

Arrays of points or lines are also useful in graphics programs and as an example we write a class to draw a line graph given an array of points.

One-dimensional arrays can be generalized to $n$-dimensions and the important case of two-dimensional arrays is discussed using a matrix as an example.

The main problem with arrays in Java is that they are not dynamic. This means that the size of an array cannot be changed after it has been created so it is necessary to either know the size at compile-time or run-time. In later Chapters we will see how to use dynamic data types such as ArrayList.

8.2 Mathematical sequences and subscript notation

In programming languages the concept of an array derives from the similar concept of a subscripted variable in mathematics. A subscripted variable is useful when you want to define an ordered sequence of numbers or variables. For example, if $x_1, x_2, \ldots, x_n$ are $n$ real numbers or variables then $\langle x_1, x_2, \ldots, x_n \rangle$ denotes the ordered sequence formed from them. Each subscript $j$ with $1 \leq j \leq n$ is called an index. Sequences and subscripted variables are quite useful, as the following examples show.

**Example 8.1** (Summation notation for sequences) If $\langle x_1, x_2, \ldots, x_n \rangle$ is a sequence of real numbers then their average is defined by

$$\frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{1}{n} \sum_{k=1}^{n} x_k$$

where the summation symbol denotes the sum of the variables $x_k$ in the sequence. At the bottom of the summation symbol we put the index name and its initial value and at the top we put the final value of the index. Then $x_k$ is a typical term in the sum.

**Example 8.2** (Infinite sequences) Sequences can be infinite. In this case a rule defines each value in the sequence. The geometric sequence $\langle a_0, a_1, \ldots, a_n, \ldots \rangle$ is defined, for some number $r$ called the geometric ratio, by $a_k = ra_{k-1}$, for all integers $k \geq 1$. This defines each sequence value in terms of the previous one. $\langle 2, 6, 18, 54, \ldots \rangle$ is an example with $a_0 = 2$ and $r = 3$. 
### Example 8.3 (Sequences of vertices)

If the points \( v_k = (x_k, y_k), k = 0, \ldots, n - 1 \), are vertices of an \( n \)-sided polygon then the polygon can be defined as the vertex sequence \( \langle v_0, v_1, \ldots, v_{n-1} \rangle \).

### Example 8.4 (Matrix multiplication)

Consider the \( 3 \times 3 \) matrices

\[
A = \begin{bmatrix}
a_{11} & a_{12} & a_{13} \\
a_{21} & a_{22} & a_{23} \\
a_{31} & a_{32} & a_{33}
\end{bmatrix}, \quad B = \begin{bmatrix}
b_{11} & b_{12} & b_{13} \\
b_{21} & b_{22} & b_{23} \\
b_{31} & b_{32} & b_{33}
\end{bmatrix}, \quad C = \begin{bmatrix}
c_{11} & c_{12} & c_{13} \\
c_{21} & c_{22} & c_{23} \\
c_{31} & c_{32} & c_{33}
\end{bmatrix}.
\]

If \( C \) is the product of \( A \) and \( B \), denoted by \( C = AB \), then the matrix element \( c_{ij} \) in row \( i \) and column \( j \) of \( C \) is defined by

\[
c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j} = \sum_{k=1}^{3} a_{ik}b_{kj}, \quad i = 1, 2, 3, \ j = 1, 2, 3
\]

in terms of the matrix elements of \( A \) and \( B \). Here we have a double subscript notation with the first subscript identifying the row of the matrix and the second subscript identifying the column.

This rule for matrix multiplication can be generalized. Suppose that \( A \) is an \( m \times p \) matrix \((m \ \text{rows}, \ p \ \text{columns})\), and \( B \) is a \( p \times n \) matrix. Then

\[
c_{ij} = a_{i1}b_{1j} + \cdots + a_{ip}b_{pj} = \sum_{k=1}^{p} a_{ik}b_{kj}, \quad 1 \leq i \leq m, \ 1 \leq j \leq n,
\]

are the matrix elements of the \( m \times n \) product matrix \( C = AB \).

A finite sequence is represented in Java by an array. The mathematical subscript notation \( a_k \) is denoted using square brackets by \( a[k] \). Sequences like this are called one-dimensional arrays. Matrices are examples of two-dimensional arrays and the mathematical notation \( a_{ij} \) is denoted using two sets of square brackets by \( a[i][j] \). Array indices in Java always begin at the value 0, so if \( a \) is the name of an array with \( n \) elements then these elements are denoted by \( a[0], a[1], \ldots, a[n-1] \).

## 8.3 Declaring and constructing arrays

We have seen that there are two kinds of variables: variables of primitive type and variables of object type (reference type). The same is true for arrays. There are arrays of primitive type and arrays of object type. In either case arrays are objects but the elements can be of a primitive type such as \texttt{int} or \texttt{double} or references to objects such as BankAccount objects.
8.3.1 Arrays of primitive type

Declaring array types

Corresponding to each primitive type there is an associated array type obtained by putting [] after the primitive type name. This gives the array types int[], double[], char[], and boolean[] for an array of int, double, char, and boolean values, respectively, and similarly for the other primitive types. For example, the statement

\[
\text{int[]} \ \text{score};
\]

declares that the variable score is the name of an array of integers. The type of this array is int[] so score is a reference to an array of integers. We say that score is an array reference.

Constructing arrays

Once an array reference has been defined we can construct an array for it to reference. Since arrays are objects this is done using new. For example we can associate a 5 element array with score using the statement

\[
\text{score} = \text{new int[5]};
\]

The keyword new is followed by the type of the array elements and then brackets containing the number of array elements. A common mistake is to assume that the 5 represents the highest index. Since indices begin at 0, the highest index is always one smaller than the value specified after new. Therefore this statement allocates storage space for 5 integer variables denoted by

\[
\text{score}[0], \text{score}[1], \text{score}[2], \text{score}[3], \text{score}[4]
\]

It is possible to define both the array reference and the array in one statement using

\[
\text{int[]} \ \text{score} = \text{new int[5]};
\]

Assigning values to array elements

Once an array has been constructed values can be assigned to the elements using assignment statements or array initializers.

**Example 8.5** (Array assignment) The statements

\[
\begin{align*}
\text{int[]} \ \text{score} &= \text{new int[5];} \\
\text{score}[0] &= 1000; \\
\text{score}[1] &= 3250; \\
\text{score}[2] &= 2104; \\
\text{score}[3] &= 675; \\
\text{score}[4] &= 1454;
\end{align*}
\]

declare an array called score, construct the array, and assign values to the array elements using assignment statements.
A pictorial representation of this three-step process is shown in Figure 8.1 for the `score` array. Part (a) shows the array reference before the array has been created, part (b) shows the situation after the array has been created. The question marks indicate that at this stage no values have been assigned to the elements. Finally, part (c) shows the array after the five assignment statements in Example 8.5 have been executed.

### Using array initializers

Assignment statements give values to array elements at run-time. If you know what the values of the array elements are at compile-time (when you write the class), or if you have an array of constants, then it is more convenient to use an array initializer to assign values. An array initializer is a comma-separated list of values enclosed in braces. It can be used as the right side of an array declaration statement.

In the following three examples `new` is not used and the size of the array is not specified because the compiler automatically constructs the array and determines the appropriate size using the list of values in the initializer.

**Example 8.6 (Array initializers)** The following statement uses an array initializer

```java
int[] factorial = {1,1,2,6,24,120,720,5040,40320,362880,3628800,39916800,479001600};
```

Thus, `0!` is `factorial[0]` which is 1, and `12!` is `factorial[12]` which is `479001600`. In this example it would be better to use the `final` modifier to define the local array of constants

```java
final int[] FACTORIAL = {1,1,2,6,24,120,720,5040,40320,362880,3628800,39916800,479001600};
```
This informs the compiler that the values cannot be changed. We can also use

```java
private static final int[] FACTORIAL =
{1,1,2,6,24,120,720,5040,40320,362880,3628800,39916800,479001600};
```

which makes the constant array a static data field.

**EXAMPLE 8.7 (Array initializers)** If you need the number of days in each month the array

```java
final int[] DAYS_IN_MONTH = {31,28,31,30,31,30,31,31,30,31,30,31};
```

is useful. A leap year test can be used to obtain the correct value for February.

**EXAMPLE 8.8 (Array initializers)** If you need the day of the year, for a given month and day of the month, the array

```java
final int[] DAY_NUMBER = {0,31,59,90,120,151,181,212,243,273,304,334};
```

is useful. Each entry is the number of days preceding the first day of each month (0 days precede January 1, 31 days precede February 1, etc.). For a leap year 1 can be added for months beyond February.

### 8.3.2 Calculating the number of days in a month

As a simple example let us write a class to solve the following problem.

“Given the year and the month what is the number of days in the month.”

The only problem here is to account for the fact that February has 29 days in a leap year and 28 days otherwise. We have already written an `isLeapYear` method in Chapter 6, Example 6.30. The array in Example 8.7 can be used. This gives the statements

```java
int days = DAYS_IN_MONTH[month-1];
if (isLeapYear(year) && month == 2) days++;
```

to calculate `days`, the number of days in the month.

Notice that `month` has a value in the range 1 to 12 but array indices always begin at zero so we needed to use `month-1` as the array index. This is an excellent example showing how arrays can be used to “look up” values. Here is a complete class to test the calculations.

```java
Class DaysInMonthCalculator

package chapter8.simple_arrays; // remove this line if you're not using packages
/**
 * Compute the number of days in a month given the year and the month.
 */
public class DaysInMonthCalculator
```
Declaring and constructing arrays

```java
private static int[] DAYS_IN_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

/**
 * Calculate number of days in a month
 * @param year the year to use
 * @param month the month in the range 1 to 12
 */
public int daysInMonth(int year, int month)
{
    int days = DAYS_IN_MONTH[month-1];
    if (isLeapYear(year) && month == 2) days++;
    return days;
}

/**
 * Return true of given year is a leap year else false.
 * @param year the year to test
 * @return true if year is a leap year else false
 */
public boolean isLeapYear(int year)
{
    return (year % 4 == 0) && (year % 100 != 0) || (year % 400 == 0);
}
```

We have used the `static` modifier in the definition of the array because it is not associated with any object. A suitable runner class that can be used from the console is given by

```java
package chapter8.simple_arrays; // remove this line if you're not using packages
import java.util.Scanner;

/**
 * A runner class for DaysInMonthCalculator
 */
public class DaysInMonthRunner
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter year");
        int year = input.nextInt();
        input.nextLine();
        System.out.println("Enter month (1 to 12) ");
        int month = input.nextInt();
        input.nextLine();

        DaysInMonthCalculator calculator = new DaysInMonthCalculator();
```
if (calculator.isLeapYear(year))
    System.out.println(year + " is a leap year");

System.out.println("Number of days in month is " +
    calculator.daysInMonth(year, month));
}
}

Declaring the size of an array at run-time

So far the size of our arrays has been declared at compile time. For example, the score array has
size 5, the FACTORIAL array has size 13, and the DAYS_IN_MONTH array has size 12. The following
example shows how to to define the size at run-time.

\[ \text{Example 8.9} \text{(Run-time array size)} \]

Assuming that input is a Scanner object the statements

System.out.println("Enter number of vertices in polygon");
int size = input.nextInt();
int[] v = new int[size];

declare and construct an array v to hold size elements where the value of size isn’t known until
the program is running.

Once the size of an array has been specified, either at compile-time or run-time, it cannot be
changed. In this sense arrays in Java are not dynamic. However, there are situations where it is
useful to have dynamic arrays that can be re-dimensioned at any time (increased or decreased in
size). In later Chapters we will introduce classes that are dynamic.

The length of an array

The length, or size, of an array is the number of elements in the array. If score is an array then the
length can be determined using the special notation score.length. This looks like a method call
but there are no parentheses. You can think of length as a public instance data field for an array.
Then score.length is an example of a qualified name for this field. It is a common mistake to
use score.length() since this is the correct syntax for String objects.

8.3.3 Sequential array processing

Arrays can easily be processed sequentially using a for-loop. In fact, this is the primary use of a
for-loop. For example, if a is an array, the standard loop structure

for (int k = 0; k < a.length; k++)
{
    // process array element a[k] here
}
can be used to access the array elements $a[k]$ one element at a time, starting with $a[0]$ and ending with $a[a.length-1]$. It is an error to use an array index that is out of bounds (negative or greater than $a.length - 1$). The Java interpreter will throw an `ArrayIndexOutOfBoundsException` for an invalid array index.

**Example 8.10** (Sum and average of the elements of an array) Assuming that the `score` array has already been defined as an array of `double` numbers, the statements

```java
double sum = 0.0;
for (int k = 0; k < score.length; k++)
{
    sum = sum + score[k];
}
double average = sum / (double) score.length;
```

calculate the sum and average of its elements.

**Example 8.11** (Displaying an array) The for-loop

```java
for (int k = 0; k < FACTORIAL.length; k++)
{
    System.out.println(k + "! = " + FACTORIAL[k]);
}
```

displays the factorials using the array declared in Example 8.6.

**Example 8.12** (Reading arrays interactively) If `input` is a `Scanner` object, statements similar to

```java
System.out.println("Enter the number of array elements");
int size = input.nextInt();
input.nextLine();
double[] score = new double[size];
for (int k = 0; k < score.length; k++)
{
    System.out.println("Enter element " + k);
    score[k] = input.nextDouble();
    input.nextLine();
}
```

can be used to read the number of array elements, construct an array of this size, and use a for-loop to read values for the array elements.

### 8.3.4 Arrays of object type

So far we have considered only arrays whose elements are of a primitive type such as `int` or `double`. It is also common to define arrays of object type. Each array element contains a reference
to an object of a specified type. Thus the array is not really an array of objects, although this terminology is common, it is an array of references to objects. Constructing such an array is a three step process:

1. Declare an array reference variable.
2. Construct an array of references.
3. Construct some objects and assign their references to the array elements.

As for arrays of primitive type, steps 1 and 2 can be done separately, or they can be done together.

**Example 8.13 (BankAccount array)** The statements

```java
BankAccount[] b;
b = new BankAccount[3];
b[0] = new BankAccount(123, "Fred", 150.50);
b[1] = new BankAccount(345, "Mary", 375.00);
b[2] = new BankAccount(987, "Bill", 75.50);
```

illustrate these three steps. The first statement defines \( b \) as a reference to an array whose elements will be BankAccount object references. Next an assignment statement constructs the array of three references and the last three assignment statements construct BankAccount objects and associate them with the array of references. The new keyword is used in two ways here: first to construct the array of references, and then to construct objects for the array elements to reference. The single statement

```java
BankAccount[] b = new BankAccount[3];
```

can also be used to declare the array reference and construct the array.

The entire process is shown in Figure 8.2 for this example. Part (a) corresponds to

```java
BankAccount[] b;
```
and part (b) corresponds to

```java
b = new BankAccount[3];
```

At this stage we have the array of references indicated by the absence of arrows in part (b). Finally the assignment statements that construct three BankAccount objects result in the picture in part (c), showing three arrows referencing the three BankAccount objects.

**Example 8.14** *(Array initializers for object types)* The statements

```java
BankAccount[] b = { new BankAccount(123, "Fred", 150.50),
    new BankAccount(345, "Mary", 375.00),
    new BankAccount(987, "Bill", 75.50) };
```

use an array initializer. The result is the same as in Example 8.13.

**Example 8.15** *(Totaling bank balances)* If `b` is an array of BankAccount objects, as in the preceding example, the statements

```java
double totalBalance = 0.0;
for (int k = 0; k < b.length; k++)
{
    totalBalance = totalBalance + b[k].getBalance();
}
double averageBalance = totalBalance / b.length;
```

calculate the total of all balances and the average balance for accounts in the array.

**Point2D arrays**

In graphics programming it is common to use objects of type `Point2D.Double` and the following example shows how to define an array of points.

**Example 8.16** *(Array of Point2D objects)* The statements

```java
Point2D.Double[] p = new Point2D.Double[3];
p[0] = new Point2D.Double(0,0);
p[1] = new Point2D.Double(1,2);
p[2] = new Point2D.Double(2,4);
```

define an array for three Point2D.Double objects corresponding to the points (0,0), (1,2), and (2,4). The equivalent statement

```java
Point2D.Double[] p = { new Point2D.Double(0,0), new Point2D.Double(1,2),
       new Point2D.Double(2,4) };
```

uses an array initializer.

**Example 8.17** *(Connecting an array of points with lines)* Given the array `p` of points in Example 8.16 and a graphics context `g2D`, the for-loop
for (int k = 0; k < p.length - 1; k++)
{
    g2D.draw(new Line2D.Double(p[k], p[k+1]));
}
connects the points with lines. Note that the largest \( k \) value is \( p.length - 2 \) since \( n \) points define \( n - 1 \) line segments.

### 8.3.5 String arrays

String arrays are common. For example, suppose we want to convert a month number in the range 1 to 12 to a name such as "January" or "February". Without arrays this can be done with a large multiple if-statement. However, we can define a string array containing the month names and use the month number as an array index to "look-up" the name.

#### Example 8.18 (Array of month names)

The statement

```java
```

defines an array for the names of the 12 months. Now if \( \text{month} \) is a month number in the range 1 to 12, \( \text{MONTH_NAMES}[\text{month}-1] \) is the name of the month.

#### Example 8.19 (Array of day names)

Similarly, the statement

```java
```

defines an array of strings for the names of the days of the week.

### Command-line arguments

We can now explain the formal argument of the `main` method of a runner class, which we have not needed so far: it is an array of `String` objects. This array is used to store command-line arguments. When we write a main method such as

```java
public static void main(String[] args)
{
    // statements
}
```

we are specifying that `args` is the name of an array of strings. These strings are the command-line arguments. When you use the Java interpreter to run a class the command-line arguments are typed after the name of the class on the command line. Each argument is separated by one or more spaces. For example, consider the following short class:
8.3 Declaring and constructing arrays

```java
package chapter8.simple_arrays; // remove this line if you’re not using packages
/**
 * To show how command line arguments can be read into a String array that
 * is available to the program. The command line arguments are available
 * in the args array: args[0], args[1], ..., args[args.length - 1].
 */
public class CommandLineArguments
{
    public static void main(String[] args)
    {
        for (int k = 0; k < args.length; k++)
        {
            System.out.println("Argument "+ k + ": " + args[k]);
        }
    }
}
```

When you run this class from the command line the first command line argument after the class name will be `args[0]`, the second will be `args[1]`, and so on. The number of command-line arguments typed is given by `args.length`. The `println` statement simply displays these arguments.

Here is some sample output for two program runs:

```java
java CommandLineArguments zero one two
.Argument 0 is zero
.Argument 1 is one
.Argument 2 is two
```

```java
java CommandLineArguments "zero one two"
.Argument 0 is zero one two
```

The second example shows that spaces can be included in command-line arguments if the argument is enclosed in double quotes.

Command-line arguments can be quite useful for console programs that require only a few input items. For example, we can rewrite the `LoanRepaymentTableRunner` class from Chapter 7, page 348, so that it gets its four input values from the command line. Here is the new version of the class.

```java
package chapter8.loan_repayment; // remove this line if you’re not using packages
/**
 * Class for running LoanRepaymentTable from console using command-line args
 */
public class LoanRepaymentTableRunner
{
    public static void main(String[] args)
    {
```
if (args.length == 4)
{
    double a = Double.parseDouble(args[0]);  // loan amount
    int y = Integer.parseInt(args[1]);        // number of years
    int p = Integer.parseInt(args[2]);        // payments per year
    double r = Double.parseDouble(args[3]);  // annual rate in percent

    LoanRepaymentTable table = new LoanRepaymentTable(a, y, p, r);
    System.out.println(table);
}
else
{
    System.out.println("Args: amount years paymentsPerYear annualRate(percent)");
}
}

The main method checks to see if four command-line arguments were supplied, otherwise a mes-

sage is displayed indicating the format of the arguments. Therefore if you forget the meaning
of the arguments just run the class without any arguments. If four arguments are supplied the
command-line strings are converted to the four arguments needed by the constructor using the
static Integer.parseInt and Double.parseDouble methods in the wrapper classes introduced
in Section 7.2.1. The output shown in Chapter 7 can be produced using the command

    java LoanRepaymentTableRunner 10000 5 2 10

As another example we can rewrite InvestmentTableRunner from Chapter 7 (page 354), to
use command-line arguments to obtain the seven constructor arguments. Here is the new version
of the class.

```java
package chapter8.investment; // remove this line if you’re not using packages
import chapter7.investment.InvestmentTable; // remove this line if you’re not using packages

/**
 * Class for running InvestmentTable from console using command-line args
 */
public class InvestmentTableRunner
{
    public static void main(String[] args)
    {
        if (args.length == 7)
        {
            double minRate = Double.parseDouble(args[0]);
            double rateStep = Double.parseDouble(args[1]);
            double maxRate = Double.parseDouble(args[2]);
```
int minYears = Integer.parseInt(args[3]);
int yearStep = Integer.parseInt(args[4]);
int maxYears = Integer.parseInt(args[5]);

double amount = Double.parseDouble(args[6]);
InvestmentTable table =
    new InvestmentTable(minRate, rateStep, maxRate,
        minYears, yearStep, maxYears, amount);
System.out.println(table);

8.3.6 Using arrays as method arguments and return values

An array can be used as a method argument, as shown in the main method where the argument is an array of strings. When the method is called, a reference to the array becomes a local variable within the method. This means that it is sometimes possible to modify an array object from within the method.

For example, if the array is of primitive type then the array elements can be modified. If the array is of object type then the array references can be used to modify the associated objects if they are mutable.

**EXAMPLE 8.20** (Array sum method) In Example 8.10 we wrote statements to sum the elements of an array. It is easy to write a method called sum that takes a double array as a formal argument, calculates the sum of the elements and returns it. The method prototype is

```java
public double sum(double[] a)
```

The square brackets indicate that a is an array of double precision numbers. The complete method declaration is

```java
public double sum(double[] a)
{
    double s = 0.0;
    for (int k = 0; k < a.length; k++)
    {
        s = s + a[k];
    }
    return s;
}
```
The array reference `a` becomes a local variable inside the method.

**EXAMPLE 8.21** (Method that prints an array) The method

```java
public void printArray(int[] a) {
    System.out.print("<" + a[0]);
    for (int k = 1; k < a.length; k++)
    {
        System.out.print("," + a[k]);
    }
    System.out.print(">");
}
```

can be used to print an integer array in the format `<1,2,3>`.

**EXAMPLE 8.22** (Method to connect an array of points with lines) The method

```java
public void drawLines(Graphics2D g2D, Point2D.Double[] p)
{
    for (int k = 0; k < p.length - 1; k++)
    {
        g2D.draw(new Line2D.Double(p[k], p[k+1]));
    }
}
```

based on Example 8.17 can be used to connect the points with lines.

**EXAMPLE 8.23** (Modifying elements of array arguments) To show that array elements can be modified by a method, when the array is an argument, consider the following method

```java
public void timesTwo(int[] a)
{
    for (int k = 0; k < a.length; k++)
    {
        a[k] = 2 * a[k];
    }
}
```

which multiplies the elements of an integer array by 2. If you call this method using statements such as

```java
int[] myArray = {1,2,3,4,5};
timesTwo(myArray);
for (int k = 0; k < myArray.length; k++)
{
    System.out.println(myArray[k]);
}
```
then the numbers printed are 2, 4, 6, 8, and 10, indicating that the array elements in myArray were changed by the method. This is what is expected since inside the method a and myArray both reference the same array.

**EXAMPLE 8.24** (Reading an array interactively and returning it) The method

```java
public int[] readArray()
{
    Scanner input = new Scanner(System.in);
    System.out.print("Enter size of array: ");
    int size = input.nextInt();
    input.nextLine();
    int[] a = new int[size];
    for (int k = 0; k < a.length; k++)
    {
        System.out.print("Enter element "+k+": ");
        a[k] = input.nextInt();
        input.nextLine();
    }
    return a;
}
```

can be used to read an array using console input. The return type indicates that the method returns a reference to an array. To call the method from within the same class use a statement such as

```java
int[] testArray = readArray();
```

The method is responsible for reading the array size from the user, constructing the array, asking the user for the array elements, and returning a reference to the array. It is important to realize that a is a local variable, so it disappears when the method exits. However, the array is an object and it doesn’t disappear since we are returning a reference to it as the value of the method and assigning it to testArray for example.

**Testing an array processing method**

The following simple class can be used to test an average method for finding the average of the numbers in an array.

**Class Average**

```java
package chapter8.array_average; // remove this line if you're not using packages
/**
 * A simple class for testing the array average method
 */
public class Average
{

```
/**
 * Return the average of the elements of an array.
 * @param a the array to average
 * @return the average of the elements of array a
 */
public double average(double[] a)
{
    double s = 0;
    for (int k = 0; k < a.length; k++)
    {
        s = s + a[k];
    }
    return s / (double) a.length;
}

You can easily test this in BlueJ using the following steps.

1. Construct an Average object called avg.
2. From its method menu choose average.
3. Enter an array initializer such as \{1,2,3,4,5\} in the input text box and the average will be shown in the method result window.

To test the method from the command line we can use command-line arguments to enter the array as shown in the following class.

```java
package chapter8.array_average; // remove this line if you’re not using packages
/**
 * Command line tester for the Average class.
 * Get array as command-line args.
 */
public class AverageRunner
{
    public static void main(String[] args)
    {
        double[] a = new double[args.length]; // construct array
        for (int k = 0; k < args.length; k++) // get its elements
        {
            a[k] = Double.parseDouble(args[k]); // convert string to double
        }
        Average avg = new Average();
        System.out.println("Average is " + avg.average(a));
    }
}
```

Some typical output is

```
java AverageRunner 1 2 3 4 5
Average is 3.0
```
8.4 Some simple array algorithms

Many algorithms can be expressed using arrays. The simplest ones are for finding the maximum and minimum values in an array.

Searching and sorting are two of the most important processing operations performed by computers so it is important to have efficient algorithms. Here we consider only the array version of the simplest searching algorithm, called linear search, for finding a given element in an array.

Then we consider the array version of the simplest sorting algorithm called bubble sort. In a later Chapter we cover searching and sorting in more detail using more efficient algorithms.

8.4.1 Algorithm for the maximum array element

The maximum problem can be stated as follows:

“Given the array \(a_0, \ldots, a_{n-1}\), determine an index \(i\) such that \(0 \leq i \leq n-1\) and \(a_i \geq a_k\) for all \(k\) such that \(0 \leq k \leq n-1\). Then the maximum value is \(a_i\).”

The index \(i\) is not unique since the maximum value may occur more than once.

The algorithm begins by assuming the maximum value is at index 0, then a loop is used to process the remaining elements in the array. Each time a larger value is obtained the index is updated. The pseudo-code algorithm is given in Figure 8.3. The final value of \(i\) is such that \(a_i\) is the maximum value. This algorithm finds the first occurrence of the maximum. To find the last occurrence change the inequality to \(a_k \geq a_i\). We could have written the algorithm to directly return the maximum value instead of the index but this would be less general since the position of the maximum would be unknown.

A similar pseudo-code algorithm for finding the minimum can be written by changing the comparison \(a_k > a_i\) to \(a_k < a_i\).

Here is a simple tester class for the Java implementation of this algorithm
package chapter8.array_algorithms; // remove this line if you’re not using packages
/**
 * A simple class for testing the findMaximum method
 */
public class MaxFinder
{
    /**
     * Determines the maximum array element and returns its position.
     * @param a the array
     * @return position of the first occurrence of the maximum
     */
    public int findMaximum(double[] a)
    {
        int index = 0;
        for (int k = 1; k <= a.length - 1; k++)
        {
            if (a[k] > a[index])
                index = k;
        }
        return index;
    }
}

This class can easily be tested in BlueJ as described above for the Average class. The following example shows how to test it in BeanShell.

**EXAMPLE 8.25** (Testing findMaximum in BeanShell) Try the statements

```
bsh % addClassPath("c:/book-projects/chapter8/array_algorithms");
bsh % MaxFinder f = new MaxFinder();
bsh % double[] a = new double[] {1,2,5,4,3};
bsh % int maxIndex = f.findMaximum(a);
bsh % print(maxIndex);
  2
bsh % print(a[maxIndex]);
  5
```

The result returned is 2, the index of the maximum value 5. In BeanShell it is necessary to use new double[]{1,2,5,4,3} to specify the array argument.

To test the class from the command line we can use the simple runner class

**Class MaxFinderRunner**

```
package chapter8.array_algorithms; // remove this line if you’re not using packages
/**
 * Command line tester for the MaxFinder class.
 * Get array as command line args.
 */
```
public class MaxFinderRunner {
    public static void main(String[] args) {
        double[] a = new double[args.length]; // construct array
        for (int k = 0; k < args.length; k++) // get its elements
            a[k] = Double.parseDouble(args[k]); // convert string to double
        MaxFinder finder = new MaxFinder();
        int indexMax = finder.findMaximum(a);
        System.out.println("Index of maximum is "+indexMax);
        System.out.println("Maximum value is "+a[indexMax]);
    }
}

that uses command-line arguments.

We have used an array of double numbers but the algorithm is easily modified to use an array of any type for which the elements can be ordered and compared.

The following example gives the algorithm for an array of BankAccount objects where the ordering is defined by the account balance.

[EXAMPLE 8.26] (Maximum bank account balance method) The method

    public int findMaximum(BankAccount[] a) {
        int index = 0;
        for (int k = 1; k <= a.length - 1; k++)
            if (a[k].getBalance() > a[index].getBalance())
                index = k;
        return index;
    }

returns the array index of the reference to the bank account that has the maximum balance.

Similarly, the following example gives the algorithm for an array of strings.

[EXAMPLE 8.27] (String array example) The method

    public int findMaximum(String[] a) {
        int index = 0;
        for (int k = 1; k <= a.length - 1; k++)
            if (a[k].compareTo(a[index]) > 0)
                index = k;
        return index;
    }
returns the array index of the reference to the string that lexicographically follows all strings in the array using the `compareTo` method.

### 8.4.2 Linear search algorithm

In a linear search of an array we are looking for a given value \( x \) among the array elements. If we find \( x \) we can return its index. Otherwise we can return the invalid index \(-1\). The linear search problem can be stated as follows:

> “Given the array \( \langle a_0, \ldots, a_{n-1} \rangle \) and a value \( x \) to find, determine an index \( i \) such that \( a_i = x \) and \( 0 \leq i \leq n-1 \). If such an index cannot be found let the index be \(-1\).”

We cannot use a for-loop here since we do not know how many times the body of the loop will be executed. The loop continues as long as there are elements in the array left to examine and as long as we have not found the element we are looking for. Therefore, we use a while-loop.

The pseudo-code algorithm is given in Figure 8.4. There are two ways the while-loop can terminate. If \( \text{index} \leq n-1 \) is false we have “gone off the end” of the array and the entire boolean expression is false so the loop will exit. Because of short-circuit evaluation, the expression \( a_{\text{index}} \neq x \) will not be evaluated in this case. Otherwise the array index could be out of range. If the element \( x \) is found then the expression \( a_{\text{index}} \neq x \) will be false and the loop will exit. When the loop exits we can test \( \text{index} \) to see which exit was taken. If \( \text{index} > n-1 \) then we did not find \( x \) so \(-1\) is returned. Otherwise, \( x \) was found and \( \text{index} \) is returned.

It is easy to translate this pseudo-code algorithm into the following tester class for the corresponding Java method.

```java
package chapter8.array_algorithms; // remove this line if you’re not using packages

/*

Class LinearSearcher

-------------

returns the array index of the reference to the string that lexicographically follows all strings in the array using the `compareTo` method.

**

8.4.2 Linear search algorithm

In a linear search of an array we are looking for a given value \( x \) among the array elements. If we find \( x \) we can return its index. Otherwise we can return the invalid index \(-1\). The linear search problem can be stated as follows:

> “Given the array \( \langle a_0, \ldots, a_{n-1} \rangle \) and a value \( x \) to find, determine an index \( i \) such that \( a_i = x \) and \( 0 \leq i \leq n-1 \). If such an index cannot be found let the index be \(-1\).”

We cannot use a for-loop here since we do not know how many times the body of the loop will be executed. The loop continues as long as there are elements in the array left to examine and as long as we have not found the element we are looking for. Therefore, we use a while-loop.

The pseudo-code algorithm is given in Figure 8.4. There are two ways the while-loop can terminate. If \( \text{index} \leq n-1 \) is false we have “gone off the end” of the array and the entire boolean expression is false so the loop will exit. Because of short-circuit evaluation, the expression \( a_{\text{index}} \neq x \) will not be evaluated in this case. Otherwise the array index could be out of range. If the element \( x \) is found then the expression \( a_{\text{index}} \neq x \) will be false and the loop will exit. When the loop exits we can test \( \text{index} \) to see which exit was taken. If \( \text{index} > n-1 \) then we did not find \( x \) so \(-1\) is returned. Otherwise, \( x \) was found and \( \text{index} \) is returned.

It is easy to translate this pseudo-code algorithm into the following tester class for the corresponding Java method.

```
8.4 Some simple array algorithms

* A simple class for testing the linearSearch method
*/
public class LinearSearcher
{

/**
 * Find a given element in an array
 * @param a the array
 * @param x the element to search for
 * @return position of the first occurrence of x or -1
 * if x is not found.
 */
public int search(double[] a, double x)
{
    int index = 0;
    int n = a.length; // number of array elements
    while (index < n && a[index] != x)
    {
        index = index + 1;
    }
    if (index >= n)
        return -1;
    else
        return index;
}
}

This class can be tested using BlueJ and the following example shows how to test the class using BeanShell

**Example 8.28** (Testing linear search using BeanShell) Try the statements

```bash
bsh % addClassPath("c:/book-projects/chapter8/array_algorithms");
bsh % LinearSearcher searcher = new LinearSearcher();
bsh % int index = searcher.search( new double[]{1,2,3,4,5}, 1);
bsh % print(index); 0
bsh % int index = searcher.search( new double[]{1,2,3,4,5}, 2);
bsh % print(index); 1
bsh % int index = searcher.search( new double[]{1,2,3,4,5}, 7);
bsh % print(index); -1
```

to test the linear search algorithm. The final result shows that 7 was not found in the array.

A runner class for command-line testing is given by

**Class LinearSearcherRunner**

```

```

book-projects/chapter8/array_algorithms
8.4.3 Bubble sort algorithm

There are many sorting algorithms. The simplest is called bubble sort. It is not very efficient for large arrays but it is the easiest to understand. More efficient algorithms will be considered in a later Chapter.

For arrays the basic sorting problem is to rearrange the elements in increasing order, or in decreasing order. For example, the array ⟨5, 3, 8, 5, 4, 2, 2⟩ is not sorted. In increasing order the sorted array is ⟨2, 2, 3, 4, 5, 5, 8⟩. Similarly, the string array ⟨"one", "two", "three"⟩ is not sorted. In increasing lexicographic order the sorted array is ⟨"one", "three", "two"⟩.

Bubble sort is the easiest sorting algorithm to understand because of its intuitive nature. If the elements of ⟨a₀, ..., aₙ₋₁⟩ can be ordered the algorithm for increasing order is

- **Pass 1**: Process the array elements a₀ to aₙ₋₁ exchanging or swapping elements that are out of order: if a₀ > a₁, swap them, if a₁ > a₂ swap them, ..., if aₙ₋₂ > aₙ₋₁ swap them. After this first pass through the array the largest element will be in the last position, its correct position. In other words the largest element “bubbles to the top” so we don’t need to consider it again.

- **Pass 2**: For the second pass process the elements a₀ to aₙ₋₂, swapping elements that are out of order. At the end of this pass the elements aₙ₋₂ and aₙ₋₁ are in their correct positions.

- **Pass n − 1**: For the final pass a₂ to aₙ₋₁ are in their correct position so we need only consider a₀ and a₁: If a₀ > a₁ then swap these elements

If we denote the pass number by p then after pass 1 one element is in its correct position, after pass 2 two elements are in their correct position and after n − 1 passes n − 1 elements are in their
correct position. We can stop here since if \( n - 1 \) elements are in their correct position then the only remaining element, \( a_0 \) must be in its correct position as the smallest array element. This means that the outer loop over the pass number ranges from \( p = 1 \) to \( p = n - 1 \).

At the start of pass number \( p \) we have the \( p - 1 \) elements \( a_{n - 1 - (p - 1)} \) to \( a_{n - 1} \) in their correct positions so we need to compare the elements \( a_0 \) to \( a_{n - 1 - p} \). Thus, the inner loop goes from \( j = 0 \) to \( j = n - 1 - p \). This gives the top level pseudo-code algorithm

\[
\text{FOR } p \leftarrow 1 \text{ TO } n - 1 \text{ DO}
\]
\[
\begin{align*}
\text{Compare pairs at positions } (0, 1), (1, 2), \ldots, (n - 1 - p, n - p) & \\
\text{swapping elements that are out of order.}
\end{align*}
\]

\text{END FOR}

As an example, consider the array \( \langle a_0, \ldots, a_7 \rangle \) given by \( \langle 44, 55, 12, 42, 94, 18, 6, 67 \rangle \). The steps are shown in Table 8.1, where boldface elements are in their correct position. A pseudo-code algorithm

\[
\text{Algorithm bubbleSort(} \langle a_0, a_1, \ldots, a_{n-1} \rangle \text{)}
\]
\[
\text{FOR } p \leftarrow 1 \text{ TO } n - 1 \text{ DO}
\]
\[
\begin{align*}
\text{FOR } j \leftarrow 0 \text{ TO } n - 1 - p \text{ DO} & \\
\text{IF } a_j > a_{j+1} \text{ THEN} & \\
\text{swap}(a_j, a_{j+1}) & \\
\text{END IF} & \\
\text{END FOR} & \\
\text{END FOR} & \\
\text{END FOR}
\end{align*}
\]

\text{Figure 8.5: Pseudo-code bubble sort algorithm}

\begin{table}[h]
\centering
\begin{tabular}{l|cccccccc}
Pass & \text{\(a_0\)} & \text{\(a_1\)} & \text{\(a_2\)} & \text{\(a_3\)} & \text{\(a_4\)} & \text{\(a_5\)} & \text{\(a_6\)} & \text{\(a_7\)} \\
\hline
Start of pass 1 & 44 & 55 & 12 & 42 & 94 & 18 & 6 & 67 \\
Start of pass 2 & 44 & 12 & 42 & 55 & 18 & 6 & 67 & 94 \\
Start of pass 3 & 12 & 42 & 44 & 18 & 6 & 55 & 67 & 94 \\
Start of pass 4 & 42 & 12 & 18 & 6 & 44 & 55 & 67 & 94 \\
Start of pass 5 & 12 & 18 & 6 & 42 & 44 & 55 & 67 & 94 \\
Start of pass 6 & 12 & 6 & 18 & 42 & 44 & 55 & 67 & 94 \\
Start of pass 7 & 6 & 12 & 18 & 42 & 44 & 55 & 67 & 94 \\
End of pass 7 & 6 & 12 & 18 & 42 & 44 & 55 & 67 & 94 \\
\end{tabular}
\caption{Bubble sort example}
\end{table}

\text{Sorting an array of numbers}

It is easy to translate this pseudo-code algorithm into the following tester class for the corresponding Java method.
package chapter8.array_algorithms; // remove this line if you’re not using packages
/**
 * A simple class for testing the bubbleSort method.
 * We use the double[] return type instead of void
 * for the bubbleSort method so the method can be
 * directly tested with BlueJ.
 */
public class BubbleSorter
{
    /**
     * Sort an array in increasing order
     * @param a the array
     */
    public double[] bubbleSort(double[] a)
    {
        int n = a.length;
        for (int p = 1; p <= n - 1; p++) // loop over passes
        {
            for (int j = 0; j <= n - 1 - p; j++)
            {
                if (a[j] > a[j + 1])
                {
                    double temp = a[j];
                    a[j] = a[j + 1];
                    a[j + 1] = temp;
                }
            }
            return a;
        }
    }
}

Three assignment statements are needed to exchange (swap) two values, since it is necessary to use a temporary variable to save the first element of the pair being swapped.

Normally the return type of the bubbleSort method would be void but we have made it double[] here so that the class can be tested using BlueJ as follows

1. Construct a BubbleSorter object.
2. From its object menu select the bubbleSort method
3. In the resulting “method call” window enter an array such as {5,4,1,6,2} in the text box.
4. You will get a “Method Result” window showing the array as an <object reference> so click on it and then click the “Inspect” button to see the sorted array or click the “Get” button to put a reference to the sorted array on the workbench.

The following example shows how to test the class using BeanShell
8.4 Some simple array algorithms

Example 8.29 (Testing bubble sort using BeanShell)  Try the statements

```java
bsh % addClassPath("c:/book-projects/chapter8/array_algorithms");
bsh % BubbleSorter sorter = new BubbleSorter();
bsh % double[] a = {44,55,12,42,94,18,6,67};
bsh % sorter.bubbleSort(a);
bsh % print(a);
double[]: {6.0,12.0,18.0,42.0,44.0,55.0,67.0,94.0}
```

to test the bubble sort algorithm using the example given in Table 8.1

A runner class for command-line testing is given by

Class BubbleSorterRunner

```java
package chapter8.array_algorithms; // remove this line if you’re not using packages
/**
 * Command line tester for the BubbleSorter class.
 * Get array as command line args.
 */
public class BubbleSorterRunner
{
    public static void main(String[] args)
    {
        double[] a = new double[args.length]; // construct array
        for (int k = 0; k < args.length; k++) // get its elements
        {
            a[k] = Double.parseDouble(args[k]); // convert string to double
        }
        BubbleSorter sorter = new BubbleSorter();
        System.out.println("Array to sort is "+arrayToString(a));
        sorter.bubbleSort(a);
        System.out.println("Sorted array is "+arrayToString(a));
    }

    public static String arrayToString(double[] a)
    {
        String s = "<" + a[0];
        for (int k = 1; k < a.length; k++)
        {
            s = s + "," + a[k];
        }
        return s + ">";
    }
}
```

Here we have included an arrayToString method to make a string representation of an array which can be displayed. Some typical output is

```java
java BubbleSorterRunner 44 55 12 42 94 18 6 67
Array to sort is <44.0,55.0,12.0,42.0,94.0,18.0,6.0,67.0>
Sorted array is <6.0,12.0,18.0,42.0,44.0,55.0,67.0,94.0>
```
Sorting an array of strings

We have written the bubble sort method to sort an array of type `double[]` but it can easily be modified to sort arrays of other types. The following example gives a method for sorting a string array.

**Example 8.30** (Sorting an array of strings) The method

```java
public void bubbleSort(String[] a)
{
    int n = a.length;
    for (int p = 1; p <= n - 1; p++) // loop over passes
    {
        for (int j = 0; j <= n - 1 - p; j++)
        {
            if (a[j].compareTo(a[j+1]) > 0)
            {
                String temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
}
```

sorts an array of strings in increasing lexicographical order.

Recall that a string array is not an array of string objects, it is an array of references to string objects. Therefore the method does not swap string objects in memory. This would be very inefficient. Instead string references are swapped. The final result is a sorted array of string references such that the string referenced by `a[0]` precedes the string referenced by `a[1]`, and so on.

8.5 Efficient evaluation of a polynomial

Polynomials are simple functions that occur in many applications. They can be represented by arrays and we want to develop an efficient algorithm to evaluate them. A polynomial of degree $n$ is a function $p$ defined for each value of $x$ by

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

in terms of the array $\langle a_0, a_1, \ldots, a_n \rangle$ of $n + 1$ coefficients with $a_n \neq 0$. An “obvious” way to calculate $p(x)$, given $x$ and the coefficient array, is

\[
\begin{align*}
\text{sum} & \leftarrow a_0 \\
\text{FOR } k \leftarrow 1 \text{ TO } n \text{ DO} \\
\text{sum} & \leftarrow \text{sum} + a_k x^k \\
\text{END FOR}
\end{align*}
\]
8.5 Efficient evaluation of a polynomial

ALGORITHM EvaluatePolynomial((a₀, a₁, ..., aₙ), x)
  p ← aₙ
  FOR k ← n - 1 TO 0 BY -1 DO
    p ← aₖ + xp
  END FOR
  RETURN p

Figure 8.6: Pseudo-code polynomial evaluation algorithm

However, this is not a very efficient algorithm: n multiplications to compute \(aₙx^n\), \(n - 1\) multiplications to compute \(aₙ₋₁x^{n-1}\), and so on. The total number of multiplications is

\[
n + (n - 1) + \cdots + 1 = \frac{n(n + 1)}{2}.
\]

8.5.1 Horner’s algorithm

We can compute \(p(x)\) with only \(n\) multiplications using Horner’s algorithm. To derive it, write the polynomial in the nested form

\[
p(x) = a₀ + x(a₁ + a₂x + \cdots + aₙx^{n-1})
\]

\[
= a₀ + x(a₁ + x(a₂ + \cdots + aₙx^{n-2}))
\]

\[
= a₀ + x(a₁ + x(a₂ + \cdots + x(a_{n-2} + x(a_{n-1} + xaₙ))\cdots))
\]

and evaluate it from the “inside out” using the following scheme:

\[
p_n = a_n
\]

\[
p_{n-1} = a_{n-1} + xa_n = a_{n-1} + xp_n
\]

\[
p_{n-2} = a_{n-2} + x(a_{n-1} + xa_n) = a_{n-2} + xp_{n-1}
\]

\[
\vdots
\]

\[
p_k = a_k + xp_{k+1}
\]

\[
\vdots
\]

\[
p_1 = a_1 + xp_2
\]

\[
p_0 = a_0 + xp_1
\]

Then \(p_0\) is the value of \(p(x)\). In the loop the subscript \(k\) moves downward from \(n - 1\) to 0 with each iteration. However, we do not need to use subscripts on \(p\) since each value can be obtained from the previous one using \(p ← a_k + xp\). The elegant pseudo-code algorithm shown in Figure 8.6 evaluates the polynomial using only \(n\) multiplications (the for-loop executes \(n\) times and there is one multiplication each time).
8.5.2 A class for polynomials

We can think of a polynomial as an object from a class called Polynomial. Since each polynomial is defined by its coefficient array then a reference to this array can be the private data field so we need a constructor with an array argument. Also we include an eval method based on the pseudo-code algorithm for evaluating a polynomial:

```java
package chapter8.polynomial; // remove this line if you’re not using packages

/**
 * A class that represents a polynomial using an array and
 * shows how to use Horner’s rule to efficiently evaluate
 * a polynomial p. A reference to the user’s array is kept
 * as a data field and an eval method is
 * provided for evaluating p at a given value of x.
 */
public class Polynomial {
    private double[] a; // the coefficient array

    /**
     * Construct a polynomial p with a given coefficient array
     * @param coefficients the array of coefficients such that
     * coefficients[k] is the coefficient of xˆk
     */
    public Polynomial(double[] coefficients) {
        a = coefficients;
    }

    /**
     * Evaluate the polynomial p.
     * @param x the value to evaluate polynomial at.
     * @return p(x) the value of p at x
     */
    public double eval(double x) {
        int n = a.length - 1;
        double p = a[n];
        for (int k = n-1; k >= 0; k--)
            p = a[k] + x*p;
        return p;
    }

    /**
     * Return a string representation of a polynomial in format
     * a[0] + a[1]x + a[2]xˆ2 + ...
     */
    public String toString() {
        // String representation
    }
}
```
8.5 Efficient evaluation of a polynomial

```
*/
public String toString()
{
    String p = (a[0] == 0) ? "" : a[0] + "x";
    for (int k = 1; k < a.length; k++)
    {
        String term = (k == 1) ? "x" : "x^n" + k;
        if (p.equals("") && a[k] != 0)
        {
            p = a[k] + term;
        }
        else
        {
            if (a[k] > 0)
            {
                p = p + " + " + a[k] + term;
            }
            else if (a[k] < 0)
            {
                p = p + " - " + Math.abs(a[k]) + term;
            }
        }
    }
    if (p.equals("")) p = "0";
    return p;
}
```

The `toString` method returns a string representation of the polynomial using $x^n$ to represent $x^n$. It is a little complicated because there are several cases. For example, if there is a first power we want to display just $x$, not $x^1$.

To construct the polynomial $p$ defined by $p(x) = 1 + 2x + 3x^2$ we can use statements such as

```
double[] coeff = new double[] {1,2,3};
Polynomial p = new Polynomial(coeff);
```

or even the single statement

```
Polynomial p = new Polynomial(new double[] {1,2,3});
```

Now to evaluate $p(3.5)$ we can use a statement such as

```
double val = p.eval(3.5);
```

8.5.3 Testing the Polynomial class

To test the Polynomial class in BlueJ perform the following steps.

1. Construct a `Polynomial` object and enter an array such as `{1,0,3,0,5}` in the input box.
2. From its object menu select the eval method and enter 1.5.

3. In the resulting “Method Result” window the value 33.0625 is shown.

4. From the object menu select the toString method to see "1.0 + 3.0x^2 + 5.0x^4" in the “Method Result” window.

The following example shows how to test the class using BeanShell.

[EXAMPLE 8.31] (Evaluating polynomials using BeanShell) Try the statements

```bash
bsh % addClassPath("c:/book-projects/chapter8/polynomial");
bsh % Polynomial p = new Polynomial(new double[]{1,0,3,0,5});
bsh % print(p.eval(1.5));
33.0625
bsh % print(p);
1.0 + 3.0x^2 + 5.0x^4
bsh %
```

to test the Polynomial class for the example \( p(x) = 1 + 3x^2 + 5x^4 \). 

To test the Polynomial class here is a runner class that gets the coefficients of a polynomial followed by the value of \( x \) as command-line arguments, constructs the polynomial, and displays it and its value at \( x \).

[Class PolynomialRunner]  

```java
package chapter8.polynomial; // remove this line if you’re not using packages
/**
 * Class to test Polynomial using command-line arguments.
 * If there are \( n \) arguments the first \( n-1 \) define the coefficient
 * array and the last one gives the value of \( x \)
 */
public class PolynomialRunner {
    public static void main(String[] args) {
        int n = args.length;

        if (args.length >= 2) {
            // First \( n-1 \) args are the coefficients so construct
            // a polynomial using them
            double[] coefficients = new double[n-1];
            for (int k = 0; k < n-1; k++) {
                coefficients[k] = Double.parseDouble(args[k]);
            }
```
Here is some output for evaluating the polynomial \( p(x) = 1 + 3x^2 + 5x^4 \) at \( x = 1.5 \) and \( x = 3.5 \).

```
java PolynomialRunner 1 0 3 0 5 1.5
p(x) = 1.0 + 3.0x^2 + 5.0x^4
p(1.5) = 33.0625
```

```
java PolynomialRunner 1 0 3 0 5 3.5
p(x) = 1.0 + 3.0x^2 + 5.0x^4
p(3.5) = 788.0625
```

## 8.6 Line graph example using arrays

Many graphics programs require arrays to store points and lines. In this section we develop a `LineGraph` class for representing line graphs. The class has an array of points as an instance data field and draws line segments from one point to the next to obtain a line graph.

Another example is a bar graph class that stores an array of height values for the bars in a vertical bar graph and draws the bars (see Exercise 8.6). In each case the array data can be used to calculate the bounding box of the graph so that an appropriate coordinate system can be chosen.

### 8.6.1 Line graph class

We want to draw a line graph given an array \( \langle v_0, v_1, \ldots, v_{n-1} \rangle \) of \( n \) vertices \( v_k = (x_k, y_k) \) specified in order of increasing \( x \)-coordinate. Each pair of vertices is to be joined by a line segment and a small circle should appear at each vertex. An example for seven vertices is shown in Figure 8.7.

We won’t assume any particular coordinate system. For example, a line graph can represent a stock market’s closing average for several consecutive days or it might represent the daily high temperatures for the month of June.
Instead, to make the graph fit in the drawing window we will use the array data to compute the bounding box of the graph in the world coordinate system. Then we can use the worldTransform method from the BarGraph3 class (Chapter 5 page 239) to transform world coordinates to default user coordinates. The LineGraph class will have the following specification.

```java
public class LineGraph extends JPanel {
    private Point2D.Double[] v; // vertices of line graph
    private double xMin, xMax, yMin, yMax; // bounding box of graph

    /* Constructor for a given vertex array p */
    public LineGraph(Point2D.Double[] p) {...}

    public void paintComponent(Graphics g) {...}

    /* Compute xMin, xMax, yMin, yMax */
    private void computeBoundingBox() {...}

    /* Perform the affine transformation */
    private AffineTransform worldTransform(double xMin, double xMax,
                                           double yMin, double yMax, int w, int h) {...}
}
```

To make the class more reusable we do not input the vertices here. Instead we leave this to the user of the class. A LineGraph object receives a reference to the vertex array as the constructor argument. Each LineGraph object represents a line graph such as the one shown in Figure 8.8.

### Choosing a coordinate system

The private method computeBoundingBox needs to determine the smallest and largest of the x-coordinates in the vertex array and similarly for the y-coordinates. This can be done in one loop as follows

```java
private void computeBoundingBox() {
    xMin = v[0].getX();
    xMax = v[0].getX();
```
8.6 Line graph example using arrays

```java
double xMin = v[0].getX();
double yMin = v[0].getY();
for (int k = 1; k < v.length; k++)
{
    double x = v[k].getX();
double y = v[k].getY();
    if (x < xMin) xMin = x;
    if (x > xMax) xMax = x;
    if (y < yMin) yMin = y;
    if (y > yMax) yMax = y;
}
```

We want to define a coordinate system slightly larger than this bounding box to allow for a 5% border around the graph as shown in Figure 8.8. This is done in the `paintComponent` method as follows.

```java
int w = getWidth(); // JPanel width in pixels
int h = getHeight(); // JPanel height in pixels
double bx = (xMax - xMin) * 0.05;
double by = (yMax - yMin) * 0.05;
AffineTransform world =
    new worldTransform(xMin-bx, xMax+bx, yMin-by, yMax+by, w, h);
g2D.transform(world);
double pixelWidth = Math.abs(1 / world.getScaleX());
double pixelHeight = Math.abs(1 / world.getScaleY());
float thickness = (float) Math.min(pixelWidth, pixelHeight);
g2D.setStroke(new BasicStroke(thickness));
```

where we have also determined the dimensions of one pixel in the world coordinate system and set the line width to one pixel.

**Drawing the axes**

If $yMin \leq 0 \leq yMax$ the $x$-axis will be visible and similarly if $xMin \leq 0 \leq xMax$ the $y$-axis will be visible. Therefore to draw the axes we use

```java
g2D.setPaint(Color.blue);
if (yMin <= 0.0 && 0.0 <= yMax)
{
    g2D.draw(new Line2D.Double(xMin,0,xMax,0));
}
if (xMin <= 0.0 && 0.0 <= xMax)
{
    g2D.draw(new Line2D.Double(0,yMin,0,yMax));
}
```
Drawing the line segments

If there are \( n \) vertices \( v_0, \ldots, v_{n-1} \) then the first line segment joins \( v_0 \) to \( v_1 \) and the last joins \( v_{n-2} \) to \( v_{n-1} \). Therefore the following loop draws the line segments.

```java
g2D.setPaint(Color.black);
for (int k = 0; k < v.length - 1; k++)
{
    double x1 = v[k].getX();
    double y1 = v[k].getY();
    double x2 = v[k+1].getX();
    double y2 = v[k+1].getY();
    g2D.draw(new Line2D.Double(x1,y1,x2,y2));
}
```

Drawing the circles at each vertex

If \( (x,y) \) is a vertex we want to draw a filled circle with radius 3 pixels centered at this vertex. The following for-loop draws the circles.

```java
double xr = 3 * pixelWidth;
double yr = 3 * pixelHeight;
g2D.setPaint(Color.red);
for (int k = 0; k < v.length; k++)
{
    double x = v[k].getX();
    double y = v[k].getY();
    Ellipse2D.Double ellipse = new Ellipse2D.Double(x-xr, y-yr, 2*xr, 2*yr);
    g2D.fill(ellipse);
}
```

The bounding box of the ellipse has bottom left corner at \( (x-xr, y-yr) \) and its width and height are \( 2*xr \) and \( 2*yr \), respectively. Putting everything together we obtain the following class.

```java
package chapter8.line_graph; // remove this line if you’re not using packages
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

/**
* Draw a line graph given its vertices (x,y).
* The axes are drawn if they are visible. A small circle is also shown
* at each vertex. Affine transformations are used to map the world
* coordinate system of the line graph to the device coordinate system.
*/
```
8.6 Line graph example using arrays

public class LineGraph extends JPanel
{
    private Point2D.Double[] v; // vertices of line graph
    private double xMin, xMax, yMin, yMax; // bounding box of graph

    /* Construct a line graph for a specified array of points */
    public LineGraph(Point2D.Double[] p)
    {
        v = p;
        computeBoundingBox();
    }

    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        Graphics2D g2D = (Graphics2D) g;
        g2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                             RenderingHints.VALUE_ANTIALIAS_ON);

        int w = getWidth(); // JPanel width in pixels
        int h = getHeight(); // JPanel height in pixels

        // Make a world coordinate system with a 5 percent border
        double bx = (xMax - xMin) * 0.05;
        double by = (yMax - yMin) * 0.05;

        AffineTransform world = worldTransform(xMin-bx, xMax+bx, yMin-by, yMax+by, w, h);
        g2D.transform(world);

        // Width of a pixel in world space is (xMax - xMin + 2*bx) / (w-1)
        // Height of a pixel in world space is (yMax - yMin + 2*by) / (h-1)
        // But we can get these values from the affine transformation:

        double pixelWidth = Math.abs(1 / world.getScaleX()); // pixel width in world
        double pixelHeight = Math.abs(1 / world.getScaleY()); // pixel height in world

        // Now we can calculate a line thickness relative that is two pixels wide
        float thickness = (float) Math.min(pixelWidth, pixelHeight);
        g2D.setStroke(new BasicStroke(thickness));

        // Draw the x-axis in blue if it is visible
        g2D.setPaint(Color.blue);
        if (yMin <= 0.0 && 0.0 <= yMax)
        {
            g2D.draw(new Line2D.Double(xMin,0,xMax,0));
        }

        // Draw the y-axis in blue if it is visible
if (xMin <= 0.0 && 0.0 <= xMax)
{
g2D.draw(new Line2D.Double(0,yMin,0,yMax));
}

// draw the line segments connecting the vertices in black

g2D.setPaint(Color.black);
for (int k = 0; k < v.length - 1; k++)
{
    double x1 = v[k].getX();
    double y1 = v[k].getY();
    double x2 = v[k+1].getX();
    double y2 = v[k+1].getY();
    g2D.draw(new Line2D.Double(x1,y1,x2,y2));
}

// We need to draw a small red circle
// about each point that has a radius of 3 pixels.

double xr = 3 * pixelWidth; // ellipse radius in x direction
double yr = 3 * pixelHeight; // ellipse radius in y direction

g2D.setPaint(Color.red);
for (int k = 0; k < v.length; k++)
{
    double x = v[k].getX();
    double y = v[k].getY();
    Ellipse2D.Double ellipse = new Ellipse2D.Double(x-xr, y-yr, 2*xr, 2*yr);
    g2D.fill(ellipse);
}

/* Compute the bounding box of the line graph.
The x range will be xMin <= x <= xMax
The y range will be yMin <= y <= yMax
*/
private void computeBoundingBox()
{
    xMin = v[0].getX();
    xMax = v[0].getX();
    yMin = v[0].getY();
    yMax = v[0].getY();
    for (int k = 1; k < v.length; k++)
    {
        double x = v[k].getX();
        double y = v[k].getY();
        if (x < xMin) xMin = x;
        if (x > xMax) xMax = x;
        if (y < yMin) yMin = y;
        if (y > yMax) yMax = y;
8.6 Line graph example using arrays

Here is a simple class that uses the GraphicsFrame class to draw a graph of 8 vertices.

Class SimpleTester

```java
package chapter8.line_graph; // remove this line if you’re not using packages
import custom_classes.GraphicsFrame; // remove this line if you’re not using packages
import java.awt.geom.*;

/**
 * Runner class for testing the LineGraph class
 * by drawing a sample graph if 8 vertices.
 */
public class SimpleTester {
    public void runTest() {
        Point2D.Double[] v = new Point2D.Double[8];
        v[0] = new Point2D.Double(-4, -1);
        v[1] = new Point2D.Double(-3, 1);
        v[2] = new Point2D.Double(-2, 0.5);
        v[3] = new Point2D.Double(-1, 2);
        v[4] = new Point2D.Double(1, 0.5);
        v[5] = new Point2D.Double(3, 3);
        v[6] = new Point2D.Double(4, 0.75);
        v[7] = new Point2D.Double(6, 0.6);

        new GraphicsFrame("A Simple Line Graph", new LineGraph(v), 401, 301);
    }

    public static void main(String[] args) {
        new SimpleTester().runTest();
    }
}
```

The output window is shown in Figure 8.8. The line graph is black, the small circles at each vertex
are red and the coordinate axes $x = 0$ and $y = 0$ are both visible.

### 8.6.2 Drawing a random line graph

As another example we use the `LineGraph` class to generate a random array of points. The only problem here is that the $x$-coordinates must increase, $x_0 \leq x_1 \leq \cdots \leq x_{n-1}$, and this won’t be the case. However, we can use a version of the `bubbleSort` method (page 404) to sort the vertex array of randomly generated `Point2D.Double` objects in order of increasing $x$-coordinate before drawing the graph.

We want the number of points in the vertex array to be a random integer in the range 5 to 100, which can be done with the statements

```java
int numVertices = (int) (96.0 * Math.random()) + 5;
Point2D.Double[] v = new Point2D.Double[numVertices];
```

recalling that the `random` method generates random numbers $r$ such that $0 \leq r < 1$. Also let us randomly choose each $x$ or $y$ coordinate to be in the range -10 to 10. This can be done with the for-loop

```java
for (int k = 0; k < v.length; k++)
{
    double x = 20.0 * Math.random() - 10.0;
    double y = 20.0 * Math.random() - 10.0;
    v[k] = new Point2D.Double(x, y);
}
```

Now the vertex array can be used to construct a `LineGraph` object that is an argument to a `GraphicsFrame` object. Thus, we have the following class to generate the graph.
Class RandomTester

package chapter8.line_graph; // remove this line if you’re not using packages
package custom_classes.GraphicsFrame; // remove this line if you’re not using packages
import java.awt.geom.*;

/**
 * Test the LineGraph class by generating a random array of points sorted
 * in order of increasing x coordinate.
 */
public class RandomTester {
  public void runTest() {
    // random number of vertices in range 5 to 100
    int numVertices = (int) (96.0 * Math.random()) + 5;
    Point2D.Double[] v = new Point2D.Double[numVertices];

    // Generate random points with coordinates in range -10 to 10
    // and sort them on increasing x coordinate.
    for (int k = 0; k < v.length; k++) {
      double x = 20.0 * Math.random() - 10.0;
      double y = 20.0 * Math.random() - 10.0;
      v[k] = new Point2D.Double(x, y);
    }
    bubbleSort(v);
    new GraphicsFrame("A Random Line Graph", new LineGraph(v), 401, 301);
  }

  /* Sort an array of points in increasing order of x-coordinates. */
  private void bubbleSort(Point2D.Double[] a) {
    int n = a.length;
    for (int p = 1; p <= n - 1; p++) {
      for (int j = 0; j <= n - 1 - p; j++) {
        if (a[j].getX() > a[j + 1].getX()) {
          Point2D.Double temp = a[j];
          a[j] = a[j + 1];
          a[j + 1] = temp;
        }
      }
    }
  }
}
public static void main(String[] args) {
    new RandomTester().runTest();
}

A typical output window is shown in Figure 8.9

8.6.3 Converting arrays to GeneralPath objects

In Chapter 5 we used the GeneralPath class to construct custom shapes. We can use arrays of points to construct GeneralPath objects and then draw or fill them. This technique could have been used in the LineGraph class: rather than draw line segments one at a time we could have used the following statements

    GeneralPath p = new GeneralPath();
    p.moveTo((float) v[0].getX(), (float) v[0].getY());
    for (int k = 1; k < v.length; k++)
    {
        p.lineTo((float) v[k].getX(), (float) v[k].getY());
    }
    g2D.draw(p);

to define the path p as a Shape object and then draw it.

8.7 For-each loop

In some for loops that sequentially process the elements of an array the only purpose of the loop index is to successively reference the next element in the array. For example, the following for loop prints the elements of an array a, one per line:
for (int k = 0; k < a.length; k++)
{
    System.out.println(a[k]);
}

Here the loop index \( k \) is used only to refer to \( a[k] \). What we really need in this case is a loop which says “iterate over the elements of the array from beginning to end and display each array element”.

In Java 5 a special kind of for loop called the “for each” loop was introduced to deal with this situation. It has the syntax

    for ( Type name : collection )
    {
        Statements
    }

Here \( collection \) is the name of a collection of objects of the given \( Type \) and the variable \( name \) will successively take on the value of the next element in the collection (the only collection we know at this stage is the array). This kind of for loop is sometimes called an \texttt{iterator}. Read the colon as “in” so the loop says “for each name in the collection ...”

For example, the for loop that prints elements can be expressed for an array \( a \) of type \( \text{int}[] \) as

    for (int elem : a)
    {
        System.out.println(elem);
    }

This is much nicer. Here the first value of \( elem \) is \( a[0] \), the next value is \( a[1] \) and so on until the last value is \( a[a.length-1] \).

\textbf{Example 8.32} (Summing an integer array) The two methods

    public int arraySum(int[] a) {
        int sum = 0;
        for (int k = 0; k < a.length; k++)
        {
            sum = sum + a[k];
        }
        return sum;
    }

    public int arraySum(int[] a) {
        int sum = 0;
        for (int elem : a)
        {
            sum = sum + elem;
        }
        return sum;
    }

show how to sum the elements of an integer array using the ordinary for loop and the new one.

\textbf{Example 8.33} (Average balance in a \texttt{BankAccount} array) The method

    public double averageBalance(BankAccount[] b) {
        

Array Data Types

double sum = 0.0;
for (BankAccount account : b)
{
    sum = sum + account.getBalance();
}
return sum / b.length;

returns the average bank balance in the given array of bank accounts

8.8 Methods with a variable number of arguments

In Java 5 it is possible to have methods with a variable number of arguments. We have already
seen two such methods: String.format and printf (see Chapter 4.2.5 page 102). The syntax
for the method prototype is

    ReturnType methodName( initialArgumentList, Type ... name )

For example the String.format method has prototype

    public static format(String format, Object ... args)

In this case there is one initial argument to specify the format string. It is followed by a variable
number of arguments of type Object. Here args refers to the variable part of the argument list.
Within the method it can be accessed as an array using the standard for loop or the for each loop.

**[EXAMPLE 8.34]** (Min method with variable number of arguments) As an example here is a
method that takes a variable number of integer arguments and returns the minimum value among
the arguments.

    public int min(int ... args)
    {
        int minValue = Integer.MAX_VALUE;
        for (int k = 0; k < args.length; k++)
        {
            if (args[k] < minValue) minValue = args[k];
        }
        return minValue;
    }

Now we can call this method using expressions such as min(4,3), which uses two arguments, and
min(7,6,5,11), which uses four arguments.

The following version uses the for each loop
8.9 Two-dimensional arrays

So far we have considered only one-dimensional arrays. Two-dimensional arrays are also common. They correspond to doubly subscripted variables that occur, for example, in matrices (see Example 8.4). In mathematics the row and column indices normally start at 1 but we must take into account that indices always start at 0 in Java. As for one-dimensional arrays, a two-dimensional array can be initialized using assignment statements or an array initializer.

[Example 8.35](Defining a $2 \times 2$ matrix with assignment statements) The BeanShell statements

```java
bsh % double[][] a = new double[2][2]; // construct 2 by 2 array
bsh % a[0][0] = 1.0; // row 0, column 0
bsh % a[0][1] = 2.0; // row 0, column 1
bsh % a[1][0] = 3.0; // row 1, column 0
bsh % a[1][1] = 4.0; // row 1, column 1
```
declare a $2 \times 2$ matrix $a$ in Java and initialize it to the matrix $\begin{bmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \end{bmatrix}$. Here the first subscript is called the row index and the second is called the column index. BeanShell does not know how to print a 2-dimensional array but it can print the rows if you use `print(a[0])` for row 0 and `print(a[1])` for row 1.

[Example 8.36](Defining a $2 \times 2$ matrix with an initializer) The statement

```java
double[][] a = { {1.0, 2.0}, {3.0, 4.0} };
```
declares the matrix in the preceding example using an array initializer (Java or BeanShell). This is shorthand accepted by the compiler for

```java
double[][] a = new double[][] { {1.0, 2.0}, {3.0, 4.0} };
```
The initializer specifies the matrix one row at a time and it shows that a two-dimensional array is really an array of row arrays. For example, \(a[0]\) is a reference to row 0, with column entries \(a[0][0]\) and \(a[0][1]\); \(a[1]\) is a reference to row 1, with column entries \(a[1][0]\) and \(a[1][1]\). Therefore the matrix is a one-dimensional array of rows.

**Example 8.37** (One-dimensional array of rows) We can generalize and think of a two-dimensional array as a one-dimensional array of rows where each row is a one-dimensional array of elements. In fact, each row can have a different number of elements in it. An array with unequal row lengths is sometimes called a **ragged array**. If all rows have the same length the array is called a **rectangular matrix**. If the number of rows and columns are the same the array is called a **square matrix**.

For example, the statement

```java
int[][] a = { {1}, {3,4,5}, {6,7,8,9} };
```

defines a two-dimensional array such that row 0 has one element, row 1 has three elements, and row 2 has four elements. The number of rows is three and is given by \(a.length\). Row 0 is \(a[0]\) and its length is \(a[0].length\) (1 in this case), row 1 is \(a[1]\) and its length is \(a[1].length\) (3 in this case), and finally row 2 is \(a[2]\) and its length is \(a[2].length\) (4 in this case). Given a two-dimensional array \(a\) the loop

```java
for (int r = 0; r < a.length; r++)
{
    System.out.println("Row " + r + " has " + a[r].length + " elements");
}
```

displays the number of elements in each row using \(a.length\) as the number of rows and using \(a[r].length\) as the length of row \(r\).

Try it in BeanShell using “Capture System in/out/err”.

```bash
bsh % int[][] a = { {1}, {3,4,5}, {6,7,8,9} };
bsh % for(int r = 0; r < a.length; r++)
{
    System.out.println("Row " + r + " has " + a[r].length + " elements");
}
Row 0 has 1 elements
Row 1 has 3 elements
Row 2 has 4 elements
```

**Example 8.38** (One-dimensional array of rows) The array in Example 8.37 can also be constructed using

```java
int[][] a = new int[3][]; // array of references to three rows
a[0] = new int[] {1}; // construct row 0
a[1] = new int[] {3,4,5}; // construct row 1
a[2] = new int[] {6,7,8,9}; // construct row 2
```
which clearly shows that it is an array with three rows of various lengths. The statements

```java
int[][] a = new int[3][];
a[0] = new int[1];
a[1] = new int[3];
a[2] = new int[4];
```

construct this array but do not assign any values to the array elements.

There are four one-dimensional arrays here (*new* is used four times) as shown in Figure 8.10. The first is an array of three references to the rows (vertical array in the figure). The elements of this array are references to rows containing 1, 3, and 4 elements respectively.

The same picture can be used to represent elements of object type. In this case where the integers appear in the picture there would instead be arrows denoting references to the objects.

**8.9.1 Multiplying matrices**

As an example of two-dimensional array manipulation let us develop a class that multiplies an \( m \times p \) matrix \( A \) by an \( p \times n \) matrix \( B \) to get the \( m \times n \) matrix \( C = AB \) called the product matrix. Recall that the matrix multiplication \( AB \) is defined only if the number of columns of \( A \) is the same as the number of rows of \( B \) and we have denoted this number by \( p \).

The formulas for the matrix elements of \( C \) are given in Example 8.4 with indices beginning at 1. With indices beginning at zero the matrix elements \( c_{ij} \) are given by

\[
c_{ij} = a_{i0}b_{0j} + \cdots + a_{ip-1}b_{p-1,j} = \sum_{k=0}^{p-1} a_{ik}b_{kj}, \quad 0 \leq i \leq m-1, \quad 0 \leq j \leq n-1.
\]

We need three nested for-loops to calculate the matrix elements \( c_{ij} \). The outer loop index \( i \) goes from 0 to \( m-1 \) where \( m \) is the number of rows in \( A \). The next loop index \( j \) goes from 0 to \( n-1 \) where \( n \) is the number of columns in \( B \). This gives the loop structure

```java
int m = a.length; // rows in A
int n = b[0].length; // columns in B
for (int i = 0; i <= m-1; i++)
{
```
for (int j = 0; j <= n-1; j++)
{
    // calculate c[i][j] here
}
}

We have used a.length to get the number of rows in A and b[0].length to get the number of columns in B. This works since b is rectangular so b[k].length is the same for all row indices k.

To calculate c[i][j] we need to use another loop to compute the sum of p terms:

double sum = 0.0;
int p = b.length; // number of rows in B
for (int k = 0; k <= p-1; k++)
{
    sum = sum + a[i][k] * b[k][j];
}
c[i][j] = sum;

Putting this loop inside the outer two loops gives the following triply-nested loop structure for matrix multiplication:

int m = a.length; // rows in A
int n = b[0].length; // columns in B
int p = b.length; // number of rows in B
double[][] c = new double[m][n]; // create product matrix
for (int i = 0; i <= m-1; i++)
{
    for (int j = 0; j <= n-1; j++)
    {
        double sum = 0.0;
        for (int k = 0; k <= p-1; k++)
        {
            sum = sum + a[i][k] * b[k][j];
        }
        c[i][j] = sum;
    }
}

A method to do this matrix multiplication would have the prototype

    public double[][] multiply(double[][] a, double[][] b)

indicating that the arguments are matrices and the return value is a matrix. Here is a class containing the method which can be tested using BlueJ or BeanShell.

Class MatrixMultiplier
__________________________ book-projects/chapter8/two_d_arrays
package chapter8.two_d_arrays; // remove this line if you’re not using packages
/**
 * Class to illustrate matrix multiplication
 */
public class MatrixMultiplier
{
    public double[][] multiply(double[][] a, double[][] b)
    {
        int m = a.length; // rows in A
        int n = b[0].length; // columns in B
        int p = b.length; // number of rows in B
        double[][] c = new double[m][n]; // create product matrix

        for (int i = 0; i <= m-1; i++)
        {
            for (int j = 0; j <= n-1; j++)
            {
                double sum = 0.0;
                for (int k = 0; k <= p-1; k++)
                {
                    sum = sum + a[i][k] * b[k][j];
                }
                c[i][j] = sum;
            }
        }
        return c;
    }
}

Testing the method in BlueJ

It is interesting to test the multiply method in BlueJ. As a simple test let us compute the matrix product

\[ C = AB = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 4 & 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 23 & 23 & 23 & 23 \\ 53 & 56 & 59 & 62 \end{bmatrix} \]

Perform the following steps:

1. Create a MatrixMultiplier object called multiplier.
2. From its object menu select the multiply method.
3. In the input box for matrix a enter \{\{1,2,3\},\{4,5,6\}\}.
4. In the input box for matrix b enter \{\{1,2,3,4\},\{5,6,7,8\},\{4,3,2,1\}\} and click “OK”.
5. This gives the “Method Result” window showing <object-reference> for the return value of the method which is the product matrix. Select the object reference, click the “Get” button and give the object the name c. You will now have an array object on the workbench for the product matrix.
6. Right click on this object and choose “Inspect”. The resulting inspector window shows the information

```java
int length = 2
[0] = <object-reference>
[1] = <object-reference>
```

indicating that this object is an array of length 2 whose rows are also object references.

7. Click on the first object reference, select “Get”, and give this object the name `row0`. It will appear on the work bench and is the first row of the product matrix.

8. Similarly click on the second object reference, select “Get”, and give this object the name `row1`. It will appear on the work bench and is the second row of the product matrix.

9. Right click on the `row0` object and select “Inspect”. The resulting inspector window shows the information

```java
int length = 4
[0] = 23
[1] = 23
[2] = 23
[3] = 23
```

indicating that this object is an array of length 4 whose values are the first row of the product matrix.

10. Similarly, right click on the `row1` object and select “Inspect”. The resulting inspector window shows the information

```java
int length = 4
[0] = 53
[1] = 56
[2] = 59
[3] = 62
```

indicating that this object is an array of length 4 whose values are the second row of the product matrix.

### Testing the method in BeanShell

The following example shows another way to test the multiply method.

**Example 8.39** (Matrix multiplication using BeanShell) Try the following statements

```bash
bsh % addClassPath("c:/book-projects/chapter8/two_d_arrays");
bsh % MatrixMultiplier mult = new MatrixMultiplier();
bsh % double[][] a = {{1,2,3},{4,5,6}};
bsh % double[][] b = {{1,2,3,4},{5,6,7,8},{4,3,2,1}};
```
to test the matrix multiplication method.

Testing matrix multiplication

The following runner class can be used to test the matrix multiplication method.

```java
package chapter8.two_d_arrays; // remove this line if you’re not using packages
import java.util.Scanner;
/**
 * Class to test matrix multiplication.
 * Compute matrix product C = AB where A is an m by p
 * matrix, B is a p by n matrix and C is an m by n matrix.
 */
public class MatrixMultiplierRunner
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);

        // get matrix dimensions for A, B
        System.out.println("Enter number of rows in A");
        int m = input.nextInt();
        System.out.println("Enter number of columns in A");
        System.out.println("which is also the number of rows in B");
        System.out.println("Enter number of columns in B");
        int p = input.nextInt();
        System.out.println("Enter number of columns in B");
        int n = input.nextInt();

        double[][] a = new double[m][p];
        double[][] b = new double[p][n];

        // Get matrix elements of A one row at a time
        System.out.println("Enter rows of A");
        for (int i = 0; i < m; i++)
        {
            System.out.print("Enter row ");
            System.out.println(i);
            for (int j = 0; j < p; j++)
            {
                System.out.print("Enter Element ");
                System.out.println(j);
                a[i][j] = input.nextDouble();
            }
        }

        // Get matrix elements of B one row at a time
        System.out.println("Enter rows of B");
        for (int i = 0; i < p; i++)
        {
            System.out.print("Enter row ");
            System.out.println(i);
            for (int j = 0; j < n; j++)
            {
                System.out.print("Enter Element ");
                System.out.println(j);
                b[i][j] = input.nextDouble();
            }
        }

        // Compute matrix product C = AB
        double[][] c = new double[m][n];
        for (int i = 0; i < m; i++)
        {
            System.out.println("Enter row ");
            System.out.println(i);
            for (int j = 0; j < n; j++)
            {
                System.out.print("Enter Element ");
                System.out.println(j);
                c[i][j] = a[i][j] * b[j][i];
            }
        }

        // Print matrix C
        System.out.println("Matrix C");
        System.out.println("Enter row ");
        for (int i = 0; i < m; i++)
        {
            System.out.print("Enter row ");
            System.out.println(i);
            for (int j = 0; j < n; j++)
            {
                System.out.print("Enter Element ");
                System.out.println(j);
                System.out.println(c[i][j]);
            }
        }
    }
}
```
Array Data Types

```java
for (int j = 0; j < p; j++)
{
    a[i][j] = input.nextInt();
}

// Get matrix elements of B one row at a time
System.out.println("Enter rows of B");
for (int i = 0; i < p; i++)
{
    for (int j = 0; j < n; j++)
    {
        b[i][j] = input.nextInt();
    }
}

// Compute product matrix C and display all matrices
MatrixMultiplier mult = new MatrixMultiplier();
double[][] c = mult.multiply(a, b);
System.out.println("Matrix a:");
displayArray(a);
System.out.println("Matrix b:");
displayArray(b);
System.out.println("Matrix c:");
displayArray(c);
}

// Display a 2d-array one row at a time
private static void displayArray(double[][] a)
{
    for (int row = 0; row < a.length; row++)
    {
        System.out.print("Row " + row + ": ");
        for (int col = 0; col < a[0].length; col++)
        {
            System.out.print(a[row][col] + " ");
        }
        System.out.println();
    }
}
```

and here is some sample output.

```
Enter number of rows in A
2
Enter number of columns in A)
which is also the number of rows in B
3
Enter number of columns in B)
```
8.9 Two-dimensional arrays

4
Enter rows of A
1 2 3
4 5 6
Enter rows of B
1 2 3 4
5 6 7 8
4 3 2 1
Matrix a:
Row 0: 1.0 2.0 3.0
Row 1: 4.0 5.0 6.0
Matrix b:
Row 0: 1.0 2.0 3.0 4.0
Row 1: 5.0 6.0 7.0 8.0
Row 2: 4.0 3.0 2.0 1.0
Matrix c:
Row 0: 23.0 23.0 23.0 23.0
Row 1: 53.0 56.0 59.0 62.0

8.9.2 Board games

Board games provide another example of the usefulness of two-dimensional arrays. The board is a square or rectangular array of squares such as a chess board or a tic-tac-toe board. Each square can be unoccupied, or it can contain a game piece (chess piece for example, or an X or O in tic-tac-toe) for one player or the other.

As an example consider the three by three board for a game of tic-tac-toe. We can use a matrix of type int[][] to represent the board. A matrix element value of zero indicates that the corresponding square is unoccupied, a value of one indicates that an X is in the square and a value of two indicates that an O is in the square. We can use a two-dimensional array of the form

int[][] board = new int[3][3];

Then, the top row of squares is board[0][k], for k = 0,1,2, the middle row is board[1][k], for k=0,1,2, and the bottom row is board[2][k], for k=0,1,2.

To initialize the board before starting a new game the following nested loop can be used.

for (int row = 0; row < board.length; row++)
{
    for (int column = 0; column < board.length; column++)
    {
        board[row][column] = 0;
    }
}

There are many methods that can be written which would be useful in developing a complete program for the game. For example, when a user selects a square to mark with an X or an O the program should check to see if the attempted move is legal, a move being legal only if the chosen square is unoccupied. Thus, we could write the following method.
public boolean isLegalMove(int row, int column)
{
    return board[row][column] == 0;
}

which returns true only if the chosen square is unoccupied.

Another useful method would be one that determines whether the board is full, indicating the game is a draw if there is no winner:

public boolean isBoardFull()
{
    for (int row = 0; row < board.length; row++)
    {
        for (int column = 0; column < board.length; column++)
        {
            if (board[row][column] == 0) return false; // found empty square
        }
    }
    return true; // didn’t find any empty squares
}

This method returns false if there is at least one unoccupied square on the board.

Another important method would be one with prototype

public boolean isWinner(int player)

such that isWinner(1) has the value true if the player using X won the game and isWinner(2) has the value true if the player using Y won the game (see Exercise 8.5).

### 8.10 Card shuffling and dealing application

An interesting application of arrays is to represent playing cards and decks of cards as objects that can be used in a card game. We first show how to represent a card as an object from a Card class and then we show how to represent a deck of playing cards as an object from a CardDeck class using an array of Card objects.

We consider a standard deck of 52 cards arranged into four suits (Clubs, Diamonds, Hearts, and Spades) each containing 13 cards (Ace, Two, ..., Jack, Queen, King) as shown in Figure 8.2. Each row gives the 13 cards in a suit. The numbers 0 to 3 at the right give the suit number. The numbers 0 to 12 at the top give the rank number of a card within a suit, and the numbers 0 to 51 at the left give the card index that will correspond to the index in the array representation of a card deck.

#### 8.10.1 Card class

According to Table 8.2 a card can be represented either by its index in the range 0 to 51 or by a pair of numbers \((rank, suit)\) where \(0 \leq rank \leq 12\) and \(0 \leq suit \leq 3\). To go from one representation to the other we can use modular arithmetic as follows.
Card shuffling and dealing application

Table 8.2: A deck of playing cards

- Given index in the range $0 \leq index \leq 51$, the rank and suit can be computed using the formulas $rank = index \mod 13$ and $suit = index \div 13$.

- Given rank and suit then index is given by $index = 13 \times suit + rank$.

For example, $2\spadesuit$ has $index = 40$ so $rank = 40 \mod 13 = 1$ and $suit = 40 \div 13 = 3$. Conversely, given the rank and suit then $index = 13 \times 3 + 1 = 40$.

Card class design

For the Card class we choose the specification

```java
public class Card {
    public Card(int rank, int suit) {...}
    public Card(int index) {...}
    public String getRankName() {...}
    public String getSuitName() {...}
    public String getCardName() {...}
    public String toString() {...}
}
```

Here we have two constructors, one for each of the two ways to represent a card. Also there are “get methods” to return various string representations. The getRankName method will return a string such as "Ace" or "King", the getSuitName method will return a string such as "Clubs" or "Spades", the getCardName method returns a compact string representation such as "A-C" for $A\spadesuit$ or "K-S" for $K\spadesuit$, and the standard toString method returns the same string as getCardName.

Card class implementation

We now have enough information to write the class:

```java
package chapter8.card_deck; // remove this line if you’re not using packages
/**
 */
```
A Card object encapsulates a standard playing card in terms of 4 suits and 13 ranks. Each card can also be described in terms of an index in the range 0 to 51:

```
index 0 to 12:  A-C, 2-C, 3-C, ..., K-C
index 13 to 25: A-D, 2-D, 3-D, ..., K-D
index 26 to 38: A-H, 2-H, 3-H, ..., K-H
index 39 to 51: A-S, 2-S, 3-S, ..., K-S
```

The index, rank (0-12) and suit (0-3) are related by

```
index = 13*suit + rank
suit = index / 13
rank = index % 13
```

```java
public class Card {
    private static String[] ranks = {
        "Ace", "Two", "Three", "Four", "Five", "Six", "Seven",
        "Eight", "Nine", "Ten", "Jack", "Queen", "King"};

    private static String rankLetter = "A23456789TJQK";

    private static String[] suits = {
        "Clubs", "Diamonds", "Hearts", "Spades"};

    private static String suitLetter = "CDHS";

    private int rank; // 0 to 12
    private int suit; // 0 to 3
    private int index; // 0 to 51
    private String rankName; // "Ace", "Two", ..., "King"
    private String suitName; // "Clubs", ..., "Spades"
    private String cardName; // e.g. A-C for ace of clubs

    /**
     * Construct a card given its rank and suit.
     * @param rank card rank in range 0 (Ace) to 12 (King)
     * @param suit card suit in range 0 (Clubs) to 3 (Spades)
     */
    public Card(int rank, int suit) {
        this.rank = rank;
        this.suit = suit;
        index = 13*suit + rank;
        rankName = ranks[rank];
        suitName = suits[suit];
        cardName = rankLetter.substring(rank, rank+1) + "+=
                   + suitLetter.substring(suit, suit+1);
    }

    /**
     */
```
8.10 Card shuffling and dealing application

* Construct a card given its index.
* @param index card index in the range 0 (A-C) to 51 (K-S)
* /
public Card(int index)
{
    this.index = index;
    rank = this.index % 13;
    suit = this.index / 13;
    rankName = ranks[rank];
    suitName = suits[suit];
    cardName = rankLetter.substring(rank,rank+1) + "-
    + suitLetter.substring(suit,suit+1);
}

/**
 * Return the rank name of the card.
 * @return rank name of card: "Ace" to "King"
 */
public String getRankName()
{
    return rankName;
}

/**
 * Return the suit name of the card.
 * @return suit name of card: "Clubs" to "Spades"
 */
public String getSuitName()
{
    return suitName;
}

/**
 * Return the card name.
 * @return the card name in format A-C to K-S
 */
public String getCardName()
{
    return cardName;
}

/**
 * Return a string representation of a card.
 * @return a string representation of a card.
 */
public String toString()
{
    return cardName;
}

Here we use two constant arrays for the names of the ranks and the names of the suits. The numeric rank and suit values are used as indices into these arrays to determine rankName and suitName.
Similarly two strings are used to obtain single letter representations (using substring) of the rank and suit which are used to determine cardName.

This class can now be tested either with BlueJ or with BeanShell. The following example shows how to test it with BeanShell using Table 8.2 to check results.

**EXAMPLE 8.40 (Testing the Card class)** Try statements such as

```bash
bsh % addClassPath("c:/book-projects/chapter8/card_deck");
bsh % Card c = new Card(35);
bsh % print(c.getCardName());
T-H
bsh % print(c.getRankName());
Ten
bsh % print(c.getSuitName());
Hearts
bsh % print(c);
T-H
```

to test the Card class.

**8.10.2 CardDeck class**

A CardDeck object represents a deck of 52 Card objects.

**CardDeck class design**

The most fundamental operations for a deck of cards is to put the deck in some standard order (done by the constructor), shuffle the deck, and deal a card. This gives the following specification.

```java
public class CardDeck
{
    public CardDeck() {...}
    public void shuffle() {...}
    public Card deal() {...}
    public int cardsInDeck() {...}
    public boolean empty() {...}
    public String toString() {...}
}
```

Here the constructor creates a deck in the standard order given in Table 8.2, the shuffle method shuffles the deck into some random order, the deal method returns the top card and removes it from the deck, the cardsInDeck method returns a number in the range 0 to 52 indicating how many cards remain to be dealt, the empty method returns true if there are no more cards in the deck, and the toString method returns a string representation of the cards remaining in the deck.
CardDeck implementation

To implement the class we need an array of Card objects as a private data field and an index that keeps track of the array index of the current top card:

```java
private static final int DECK_SIZE = 52;
private Card[] deck;
private int topCardIndex;
```

Here for a full deck topCardIndex will be zero. Each time a card is dealt this index is incremented to refer to the new top card. When there are no cards remaining the index has the value 52 which is one more than the largest array index.

The constructor implementation

```java
public CardDeck()
{
  deck = new Card[DECK_SIZE];
  initialize();
}
```

constructs the deck and calls a private initialize method which constructs 52 cards in standard order:

```java
private void initialize()
{
  topCardIndex = 0;
  for (int k = 0; k < DECK_SIZE; k++)
  {
    deck[k] = new Card(k);
  }
}
```

The only tricky method is shuffle. We use the Random class in java.util to generate random integers:

```java
Random rand = new Random();
```

Then rand.nextInt(n) returns a random integer in the range 0 to n-1. The following algorithm can be used to shuffle the 52 cards.

- Step 0: Choose a random position from 0 to 51 and exchange the element at that index with the element at position 0. This gives a random card in position 0.
- Step 1: Choose a random position from 1 to 51 and exchange the element at that index with the element at position 1. This gives a random card in position 1.
- Step k: Choose a random position from k to 51 and exchange the element at that index with the element at position k. This gives a random card in position k.
• Step 50: Choose a random position from 50 to 51 and exchange the element at that index with the element at position 50. This gives a random card in position 50. This is the last step since the last card at position 51 will be random.

We can generate a random integer index in the range \( k \) to 51 using

\[
\text{int index} = \text{rand.nextInt(}DECK\_\text{SIZE} - k\text{)} + k;
\]

The \text{rand} operation here produces a random integer in the range 0 to \( 52 - k - 1 \) and we add \( k \) to get a random integer in the range \( k \) to 51. This gives the following implementation of \text{shuffle}.

```java
public void shuffle()
{
    Random rand = new Random();
    initialize();
    for (int k = 0; k <= DECK\_SIZE - 2; k++)
    {
        int index = rand.nextInt(DECK\_SIZE - k) + k;
        Card temp = deck[k];
        deck[k] = deck[index];
        deck[index] = temp;
    }
}
```

To deal a card from the deck we need to return a reference to the \text{Card} object whose index is \text{topCardIndex} and then we need to increment \text{topCardIndex} so that it references the next card. If the \text{deal} method is called on an empty deck of cards it returns \text{null}:

```java
public Card deal()
{
    if (topCardIndex == DECK\_SIZE) return null;
    Card topCard = deck[topCardIndex];
    topCardIndex++;
    return topCard;
}
```

The other methods are easily implemented and we obtain the class

```java
package chapter8.card_deck; // remove this line if you’re not using packages
import java.util.Random;
/**
 * A CardDeck object represents a deck of 52 Card objects.
 */
public class CardDeck
{
```
private static final int DECK_SIZE = 52;
private Card[] deck;
private int topCardIndex;

/**
 * Construct a deck of cards initialized in the standard order 0 to 51
 */
public CardDeck()
{
    deck = new Card[DECK_SIZE];
    initialize();
}

/**
 * Return all the cards to the deck in standard order.
 */
public void initialize()
{
    topCardIndex = 0;
    for (int k = 0; k < DECK_SIZE; k++)
    {
        deck[k] = new Card(k);
    }
}

/**
 * Return all the cards in the deck to standard order and
 * then shuffle them into a random order.
 */
public void shuffle()
{
    Random rand = new Random();
    initialize();
    for (int k = 0; k <= DECK_SIZE - 2; k++)
    {
        int index = rand.nextInt(DECK_SIZE - k) + k;
        Card temp = deck[k];
        deck[k] = deck[index];
        deck[index] = temp;
    }
}

/**
 * Deal a card from the deck.
 * If there are no more cards null is returned.
 * @return the card or null if there are no more cards.
 */
public Card deal()
{
    if (topCardIndex == DECK_SIZE) return null;
    Card topCard = deck[topCardIndex];
    topCardIndex++;
}
return topCard;
}

/**
 * Return the number of cards remaining in the deck.
 * @return the number of cards remaining in the deck.
 */
public int cardsInDeck()
{
    return DECK_SIZE - topCardIndex;
}

/**
 * Return true if deck is empty, false otherwise.
 * @return true if deck is empty, false otherwise.
 */
public boolean empty()
{
    return topCardIndex == DECK_SIZE;
}

/**
 * Return a string representation of the cards remaining in the deck.
 */
public String toString()
{
    StringBuilder b = new StringBuilder(215);
    for (int k = topCardIndex; k < DECK_SIZE; k++)
    {
        Card card = deck[k];
        b.append(card.getCardName());
        if ( (k+1) % 13 == 0)
            b.append("\n");
        else
            b.append( " ");
    }
    return b.toString();
}

The toString method uses the StringBuilder class introduced in Section 7.10.2 to produce a string representation of the cards remaining in the deck, 13 cards per line.

Testing the class in BlueJ

Try steps such as the following to test the class in BlueJ.


2. From its object menu select shuffle then use the toString method to see the first few cards of the deck (the newlines in the string are replaced by spaces in BlueJ).
3. From the object menu select the deal method. From the “Method Result” window select the object reference and use “Get” to put the Card object on the workbench.

4. Use the Card object’s methods to see the card’s properties.

5. From the CardDeck object menu select the cardsInDeck method to verify that there are now 51 cards in the deck.

**Testing the class in BeanShell**

**Example 8.41** (Testing the CardDeck class) Try statements such as

```
bsh % addClassPath("c:/book-projects/chapter8/card_deck");
bsh % CardDeck deck = new CardDeck();
bsh % deck.shuffle();
bsh % print(deck);
T-S 7-S A-S Q-D 5-C 9-H 7-C 5-D 6-C 4-H J-C 5-S T-H
9-D K-H J-S 4-D K-D 3-C 3-H K-S T-D 8-D Q-H A-C 3-S
J-D A-D 8-H 9-S 7-H 4-S 5-H 8-C 2-H K-C Q-S T-C 6-D
8-S 3-D 7-D J-H 6-H 6-S 4-C 2-S 2-C 9-C 2-D Q-C A-H
bsh % Card top = deck.deal();
bsh % print(top);
T-S
bsh % print(deck.cardsInDeck());
51
```

to test the CardDeck class using BeanShell.

**Testing the class from the command line**

The following simple class can be used to test the class from the command line.

**Class CardDeckTester**

```java
package chapter8.card_deck; // remove this line if you’re not using packages
/**
 * A simple test class for CardDeck class
 */
public class CardDeckTester {
    public void run() {
        CardDeck deck = new CardDeck();
        System.out.println("Initialized deck:");
        System.out.println(deck);
        System.out.println("Shuffled deck:");
```
deck.shuffle();
System.out.println(deck);

// Deal a few cards
Card card;
card = deck.deal();
System.out.println("dealing " + card.getCardName());
card = deck.deal();
System.out.println("dealing " + card.getCardName());

int count = 0;
while ( ! deck.empty() )
{
    Card c = deck.deal();
    count++;
    System.out.print(c.getCardName());
    if (count == 13)
    {
        System.out.println();
        count = 0;
    }
    else
    {
        System.out.print(" ");
    }
    if (count != 0) System.out.println();
}

public static void main(String[] args)
{
    new CardDeckTester().run();
}

A while-loop is used to deal the cards and display them as they are dealt, 13 cards per line. Here is some typical output.

Initialized deck:
A-C 2-C 3-C 4-C 5-C 6-C 7-C 8-C 9-C T-C J-C Q-C K-C
A-D 2-D 3-D 4-D 5-D 6-D 7-D 8-D 9-D T-D J-D Q-D K-D
A-S 2-S 3-S 4-S 5-S 6-S 7-S 8-S 9-S T-S J-S Q-S K-S

Shuffled deck:
9-S 7-D J-S 4-S 3-S 8-H 7-S T-D 4-C 9-C Q-C 6-H 2-S
5-C 2-C 8-C J-C 3-H 6-C Q-D Q-H 8-S 5-H 8-D K-D 3-C
T-C Q-S T-S J-D A-S 9-H 4-D 4-H K-C T-H 2-H 9-D A-C
5-S 2-D J-H A-D 7-C K-H 5-D K-S 3-D 7-H 6-S A-H 6-D
dealing 9-S
dealing 7-D
8.11 Review exercises

► **Review Exercise 8.1** Draw pictures of an array of three `BankAccount` objects that shows the three steps in the creation process.

► **Review Exercise 8.2** Draw a picture that shows that a 2-dimensional array is an array of rows.

► **Review Exercise 8.3** Write a pseudo-code algorithm that adds corresponding elements of the given arrays \(\langle a_0, \ldots, a_{n-1}\rangle\) and \(\langle b_0, \ldots, b_{n-1}\rangle\) to produce the array \(\langle c_0, \ldots, c_{n-1}\rangle\), using the addition formula \(c_k = a_k + b_k\).

► **Review Exercise 8.4** Write a pseudo-code algorithm called `max` that finds the maximum element in an \(m \times n\) rectangular array \(\langle a_{00}, \ldots, a_{mn}\rangle\) of numbers.

8.12 BeanShell exercises

The following BeanShell exercises can be done using the Workspace Editor. First run BeanShell, then choose “Workspace Editor” from the “File” menu to open the editor. If you want to use `System.out.println` then it is also necessary to choose “Capture System in/out/err” from the “File” menu.

Now you can type statements into the editor and they won’t be executed as they are entered. When you have finished entering statements choose “Evaluate in Workspace” from the “Evaluate” menu. Now the statements will be executed. You can edit the statements and evaluate them again, and so on.

This is useful for testing static methods. Type in the method, evaluate it then test it interactively using the workspace.

► **BeanShell Exercise 8.1** Write some statements to calculate \(2^0\) to \(2^{15}\) in a for-loop and store them in an array called `powerOfTwo` such that \(2^k\) is stored in `powerOfTwo[k]`.

► **BeanShell Exercise 8.2** Write some statements to calculate \(n!\) for \(n = 0\) to \(n = 12\) in a for-loop and store them in an array called `fact` such that \(n!\) is stored in `fact[n]`.

► **BeanShell Exercise 8.3** Write some statements to create an array of 3 `BankAccount` objects and write a for-loop to compute the total of their balances.

► **BeanShell Exercise 8.4** Translate the algorithm in Review Exercise 8.3 to a method with prototype

\[
\text{public double[]} \ \text{add(double[]} \ a, \ \text{double[]} \ b)\]
The method should construct the array for the sum and return a reference to it. Test the method using BeanShell.

**BeanShell Exercise 8.5** Write a Java method called `max` with prototype

```java
public double max(double[][] a)
```

that finds the maximum element in an \( m \times n \) rectangular array \( \langle a_{0,0}, \ldots, a_{mn} \rangle \) of numbers. Test the method using BeanShell.

**BeanShell Exercise 8.6** Do a more general version of the `max` method in BeanShell Exercise 8.5 that does not assume that array is rectangular. In other words each row can have a different number of elements. Recall that the number of rows in a 2-dimensional matrix `a` is `a.length` and the number of elements in row `k` is `a[k].length`.

**BeanShell Exercise 8.7** Write a method with prototype

```java
public boolean same(double[][] a)
```

that returns true if all the elements in the array `a` are the same and false otherwise.

**BeanShell Exercise 8.8** Write a method with prototype

```java
public Polynomial add(Polynomial p1, Polynomial p2)
```

that adds two `Polynomial` objects and returns the sum as a `Polynomial` object. (If polynomial `p_1` has degree `m` and polynomial `p_2` has degree `n`, then their sum has degree \( s = \max(m, n) \)).

### 8.13 Programming exercises

**Exercise 8.1** *(Reversing the elements of an array)*

Write a method with prototype

```java
public void reverse(int[] a)
```

that reverses the elements of the array `a` in place (without creating another array). Write a class to test the method using the `IntArrayIO` class for input and output. Hint: swap elements from the end with elements from the beginning in a for-loop.

**Exercise 8.2** *(A student mark histogram)*

Write a method with prototype

```java
public int[] markHistogram(int[] marks)
```

where `marks` is an array of marks in the range 0 to 100. The method creates a new array `h` of length 6 such that `h_0` is the number of marks `m` with `0 \leq m < 50`, `h_1` is the number of marks `m` with `50 \leq m < 59`, `h_2` is the number of marks `m` with `60 \leq m < 69`, and so on until `h_5` is the number of marks `m` with `90 \leq m \leq 100`. This array is returned as the value of the method. Write a class to test the method by reading an array of marks and displaying the number of marks in each range.
Exercise 8.3 (Sorting bank accounts)
Write a program called SortBankAccounts that

1. asks the user how many bank accounts will be entered,
2. reads the data for that many bank accounts and stores them in an array,
3. sorts the array in decreasing order by balance and displays the sorted array,
4. sorts the array in increasing lexicographical order according to the owner name and displays the sorted array.

Exercise 8.4 (Pascal’s triangle)
Write a program called PascalTriangleMaker that reads an integer value \( n \) with \( n \geq 0 \). computes the binomial coefficients

\[
\binom{n}{k} = C(n,k) = \frac{n!}{k!(n-k)!}, \quad n \geq 0, \quad k = 0, \ldots, n,
\]

the number of \( k \)-element subsets of an \( n \)-element set, using the recurrence relation

\[
C(n,k) = C(n-1,k) + C(n-1,k-1), \quad \text{with } C(n,0) = 1, C(n,n) = 1,
\]

which expresses a value in row \( n \) in terms of two neighboring values in the preceding row \( n-1 \).

Store the calculated coefficients in a two-dimensional ragged array: row \( n \) should have \( n+1 \) elements in it to store \( C(n,0), C(n,1), \ldots, C(n,n) \).

Display the array in the triangular form

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

How large can \( n \) be before integer overflow occurs?

Exercise 8.5 (An isWinner method for tic-tac-toe)
We have represented a tic-tac-toe board by a two-dimensional array called \( \text{board} \). Write a method called \( \text{isWinner} \) with prototype

\[
\text{public boolean isWinner(int player)}
\]

If the value of \( \text{player} \) is 1, corresponding to X, the method should return true if this player has won the game with three X’s in a row, or in a column, or in one of the two diagonals. Similarly, if the value of \( \text{player} \) is 2, corresponding to O, the method should return true if this player has won the game.

Test your method by including it in a class called TicTacToeTester that gets the 9 square values, one row at a time, as command-line arguments and displays either that the player using X has won, the player using Y has won, or the game is a draw (board is full but there is no winner). For example, here is the output for three program runs.
java TicTacToeTester 2 2 1 2 1 0 1 0 0
Player using X is the winner
java TicTacToeTester 1 1 2 1 2 0 2 0 0
Player using O is the winner
java TicTacToeTester 2 2 1 1 1 2 2 1 2
The game is a draw

◮ Exercise 8.6 (A bar graph class)
Using the LineGraph class as a model write a similar BarGraph class for a graph of vertical bars filled with random colors and outlined in black. The bar heights are stored in an array \( h = \langle h_0, h_1, \ldots, h_{n-1} \rangle \). In the world coordinate system the width of each bar is 1 unit, so the top left corner of bar \( i \) has coordinates \((i, h_i)\).

Write a runner class called BarGraphRunner that uses command-line arguments to test the BarGraph class. For the command

java BarGraphRunner 1 2 3 4 5 4 3 2 1
A typical output window is shown in Figure 5.20

◮ Exercise 8.7 (Drawing a pentagon using arrays)
Rewrite the DrawPentagon program from Chapter 5 using arrays of points and for-loops. An array for the five vertices of the pentagon can be declared using

private Point2D.Double[] v = new Point2D.Double[5];

◮ Exercise 8.8 (Drawing a pentagonal star)
Write a program called PentagonalStar to draw a five pointed star, similar to the one shown in Figure 8.11 as follows. Define the pentagonal angles and vertices of the inner pentagon by

\[
\alpha_k = \frac{\pi}{180} (72k + 54), \quad v_k = (r_1 \cos \alpha_k, r_1 \sin \alpha_k), k = 0, \ldots, 4
\]

Define the pentagonal star angles and vertices by

\[
\beta_k = \frac{\pi}{180} (72k + 90), \quad w_k = (r_2 \cos \beta_k, r_2 \sin \beta_k), k = 0, \ldots, 4
\]

The two radii, \( r_1 \) and \( r_2 \), are connected by the formula

\[
r_1 = \left( \frac{\sin a}{\cos b} \right) r_2, \text{ where } a = \left( \frac{\pi}{180} \right) 18, \text{ and } b = \left( \frac{\pi}{180} \right) 36
\]

The radius \( r_2 \) is the radius of the circumscribed circle for the star, and the smaller radius \( r_1 \) is the radius of the pentagon inside the star (formed from the 5 vertices of the star that are closest to the center).

Choose an appropriate coordinate system and value for \( r_2 \) and draw the pentagonal star as 10 lines between the following 10 pairs of vertices:

\( v_0w_0, w_0v_1, v_1w_1, w_1v_2, v_2w_2, w_2v_3, v_3w_3, w_3v_4, v_4w_4, w_4v_0 \)
8.13 Programming exercises

Figure 8.11: Output of the PentagonalStar program

Exercise 8.9 (Drawing a duck as an array of lines)
The following class

```java
public class Duck
{
    public static double minX = 0.0;
    public static double maxX = 29.0;
    public static double minY = 0.7;
    public static double maxY = 17.0;

    public static double[] x = {
        0.0, 2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.4, 17.0,
        17.3, 17.8, 18.5, 20.0, 22.0, 24.0, 26.0, 28.0, 29.0, 28.8,
        27.2, 25.0, 23.0, 21.5, 21.1, 21.5, 22.8, 24.1, 25.1, 25.2,
        24.2, 22.1, 20.8, 18.0, 16.0, 14.0, 12.0, 10.0, 8.0, 6.1,
        4.2, 3.0, 1.3, 0.0
    };

    public static double[] y = {
        8.8, 7.6, 7.1, 7.4, 8.0, 8.9, 9.6, 9.9, 9.4, 9.7,
        12.0, 14.0, 16.1, 17.0, 17.0, 16.0, 13.9, 13.1, 13.2, 12.3,
        11.5, 11.5, 11.5, 11.2, 10.5, 9.0, 8.0, 7.0, 5.1, 3.6,
        1.9, 1.1, 0.9, 0.7, 0.8, 0.9, 1.0, 1.2, 1.5, 2.1,
        2.9, 4.1, 6.0, 8.8
    };
}
```
defines a duck as a two arrays of $x$ and $y$ coordinates. Write a class called \texttt{DrawDuck} that draws the duck using a transformed coordinate system. Draw the duck as a \texttt{GeneralPath} object, outline it in black and fill it with green. The output window is shown in Figure 8.12

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure8_12.png}
\caption{A duck using arrays of points}
\end{figure}

\textbf{Exercise 8.10 (Returning multiple values from a method)}

If we want to find both the maximum and minimum values in an array of integers we could use the \texttt{findMinimum} and \texttt{findMaximum} methods in class \texttt{IntArrayMaxMin}. This is inefficient since both values can be found using one for-loop. We would like to write a method called \texttt{findMaxMin} that returns both values but a Java method can only return one value. To do this we invent a small class

\begin{verbatim}
public class IntRange
{
    public int min;
    public int max;
}
\end{verbatim}

with two public data fields and no methods. Now the \texttt{findMaxMin} method can be written with prototype

\begin{verbatim}
public IntRange findMaxMin(int[] a)
\end{verbatim}

so that it returns an \texttt{IntRange} object. The method can be called using a statement such as

\begin{verbatim}
IntRange r = findMaxMin(a);
\end{verbatim}

Then the minimum and maximum values can be obtained as $r$.\texttt{min}$ and $r$.\texttt{max}$ since public data fields can be accessed directly using the dot notation without the need for get methods. Write this method and test it: you could put it in \texttt{IntArrayMaxMin} as a \texttt{static} method and modify \texttt{MaxMinCalculator} to test it.
Exercise 8.11 (Counting the number of times the maximum occurs)
Write a pseudo-code algorithm that takes an array \(a_0,\ldots,a_{n-1}\) as input and returns the number of times the maximum array element occurs in the array. The obvious algorithm would use one for-loop to find the maximum value followed by another for-loop to count how many times the maximum occurs. Instead, write your algorithm using only a single for-loop, translate it into a method with prototype

\[
\text{public int countMax(double[]} a)\
\]
and write a tester class for it.

Exercise 8.12 Write a class called PolynomialAdder that uses the add method from BeanShell Exercise 8.8 to add two polynomials.

Exercise 8.13 Write a class that can produce monthly calendars of the form

July 2003
S M T W T F S
1  2  3  4  5
6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 28
27 28 29 30 31

Internally store a calendar as a 2-dimensional array with 6 rows and 7 columns. Store 0’s in positions that should be blank and store the day (1 to 31) in other positions. Use this array to print the calendar. You can use the CalendarMonth class in Chapter 4 (page 133) to determine on which day is the first of the month and how many days are in the month.
Array Data Types