

5

Java Class Libraries and Arrays

5.1 Package statements

The `package` statement defines that the classes within a Java file are part of a given package. The full name of a class is:

package.classFilename

The fully qualified name for a method is:

package.classFilename.method_name()

Each class file with the same package name is stored in the same directory. For example, the `java.applet` package contains several files, such as:

```
applet.java          appletcontent.java
appletstub.java     audioclip.java
```

Each has a first line of:

```
package java.applet;
```

and the fully classified names of the class files are:

```
java.applet.applet      java.applet.appletcontent
java.applet.appletstub  java.applet.audioclip
```

These can be interpreted as in the `java/applet` directory. An example listing from the class library given in Sample run 5.1.

Normally when a Java class is being developed it is not part of a package as it is contained in the current directory.



Sample run 5.1

```
java/
java/lang/
java/lang/Object.class
java/lang/Exception.class
java/lang/Integer.class
```

The main packages are:

java.applet	java.awt	java.awt.datatransfer
java.awt.event	java.awt.image	java.awt.peer
java.beans	java.io	java.lang
java.lang	java.lang.reflect	java.math
java.net	java.rmi	java.rmi.dgc
java.rmi.registry	java.rmi.server	java.security
java.security.acl	java.security.interfaces	java.sql
java.text	java.util	java.util.zip

5.2 Import statements

The `import` statement allows previously written code to be included in the applet. This code is stored in class libraries (or packages), which are compiled Java code. For the JDK tools, the Java source code for these libraries is stored in the `src/java` directory.

For example a Java program which uses maths methods will begin with:

```
import java.lang.Math;
```

This includes the `math` class libraries (which is in the `java.lang` package). The default Java class libraries are stored in the `classes.zip` file in the `lib` directory. This file is in a compressed form and should not be unzipped before it is used. The following is an outline of the file.

```
Searching ZIP: CLASSES.ZIP
Testing: java/
Testing: java/lang/
Testing: java/lang/Object.class
Testing: java/lang/Exception.class
Testing: java/lang/Integer.class
::      ::
Testing: java/lang/Win32Process.class
Testing: java/io/
Testing: java/io/FilterOutputStream.class
Testing: java/io/OutputStream.class
::      ::
Testing: java/io/StreamTokenizer.class
Testing: java/util/
Testing: java/util/Hashtable.class
Testing: java/util/Enumeration.class
::      ::
Testing: java/util/Stack.class
Testing: java/awt/
Testing: java/awt/Toolkit.class
Testing: java/awt/peer/
Testing: java/awt/peer/WindowPeer.class
::      ::
Testing: java/awt/peer/DialogPeer.class
Testing: java/awt/Image.class
Testing: java/awt/MenuItem.class
```

```

Testing: java/awt/MenuComponent.class
Testing: java/awt/image/
      ::      ::
      ::      ::
Testing: java/awt/ImageMediaEntry.class
Testing: java/awt/AWTException.class
Testing: java/net/
Testing: java/net/URL.class
Testing: java/net/URLStreamHandlerFactory.class
      ::      ::
Testing: java/net/URLConnection.class
Testing: java/applet/
Testing: java/applet/Applet.class
Testing: java/applet/AppletContext.class
Testing: java/applet/ AudioClip.class
Testing: java/applet/AppletStub.class

```

The other form of the `import` statement is:

```
import package.*;
```

which will import all the classes within the specified package. Table 5.1 lists the main class libraries and some sample libraries.

It can be seen that upgrading the Java compiler is simple, as all that is required is to replace the class libraries with new ones. For example, if the basic language is upgraded then `java.lang.*` files is simply replaced with a new version. The user can also easily add new class libraries to the standard ones. A complete listing of the classes is given in Appendix D.

Table 5.1 Class libraries

<i>Class libraries</i>	<i>Description</i>	<i>Example libraries</i>
<code>java.lang.*</code>	Java language	<code>java.lang.Class</code> <code>java.lang.Number</code> <code>java.lang.Process</code> <code>java.lang.String</code>
<code>java.io.*</code>	I/O routines	<code>java.io.InputStream</code> <code>java.io.OutputStream</code>
<code>java.util.*</code>	Utilities	<code>java.util.BitSet</code> <code>java.util.Dictionary</code>
<code>java.awt.*</code>	Windows, menus and graphics	<code>java.awt.Point</code> <code>java.awt.Polygon</code> <code>java.awt.MenuComponent</code> <code>java.awt.MenuBar</code> <code>java.awt.MenuItem</code>
<code>java.net.*</code>	Networking (such as sockets, URLs, ftp, telnet and HTTP)	<code>java.net.ServerSocket</code> <code>java.net.Socket</code> <code>java.net.SocketImpl</code>
<code>java.applet.*</code>	Code required to run an applet	<code>java.applet.AppletContext</code> <code>java.applet.AppletStub</code> <code>java.applet.AudioClip</code>

5.3 Mathematical operations

Java has a basic set of mathematics methods which are defined in the `java.lang.Math` class library. Table 5.2 outlines these methods. An example of a method in this library is `abs()` which can be used to return the absolute value of either a `double`, an `int` or a `long` value. Java automatically picks the required format and the return data type will be of the same data type of the value to be operated on.

As the functions are part of the `Math` class they are preceded with the `Math.` class method. For example:

```
val2=Math.sqrt(val1);
val3=Math.abs(val2);
z=Math.min(x,y);
```

Java stand-alone program 5.1 shows a few examples of mathematical operations and Sample run 5.2 shows a sample compilation and run session.

Table 5.2 Methods defined in `java.lang.Math`

<i>Method</i>	<i>Description</i>
<code>double abs(double a)</code>	Returns the absolute double value of a.
<code>float abs(float a)</code>	Returns the absolute float value of a.
<code>int abs(int a)</code>	Returns the absolute integer value of a.
<code>long abs(long a)</code>	Returns the absolute long value of a.
<code>double acos(double a)</code>	Returns the arc cosine of a, in the range of 0.0 through π .
<code>double asin(double a)</code>	Returns the arc sine of a, in the range of $\pi/2$ through $\pi/2$.
<code>double atan(double a)</code>	Returns the arc tangent of a, in the range of $-\pi/2$ through $\pi/2$.
<code>double atan2(double a, double b)</code>	Converts rectangular co-ordinates (a, b) to polar (r, theta).
<code>double ceil(double a)</code>	Returns the 'ceiling' or smallest whole number greater than or equal to a.
<code>double cos(double a)</code>	Returns the trigonometric cosine of an angle.
<code>double exp(double a)</code>	Returns the exponential number e (2.718...) raised to the power of a.

Table 5.2 Methods defined in `java.lang.Math` (continued)

<code>double floor(double a)</code>	Returns the ‘floor’ or largest whole number less than or equal to a.
<code>double IEEERemainder(double f1, double f2)</code>	Returns the remainder of f1 divided by f2 as defined by IEEE 754.
<code>double log(double a)</code>	Returns the natural logarithm (base e) of a.
<code>double max(double a, double b)</code>	Takes two double values, a and b, and returns the greater.
<code>double max(float a, float b)</code>	Takes two float values, a and b, and returns the greater number of the two.
<code>int max(int a, int b)</code>	Takes two int values, a and b, and returns the greater number.
<code>long max(long a, long b)</code>	Takes two long values, a and b, and returns the greater.
<code>double min(double a, double b)</code>	Takes two double values, a and b, and returns the smallest.
<code>float min(float a, float b)</code>	Takes two float values, a and b, and returns the smallest.
<code>int min(int a, int b)</code>	Takes two integer values, a and b, and returns the smallest number.
<code>long min(long a, long b)</code>	Takes two long values, a and b, and returns the smallest number of the two.
<code>double pow(double a, double b)</code>	Returns the number a raised to the power of b.
<code>double random()</code>	Generates a random number between 0.0 and 1.0.
<code>double rint(double b)</code>	Converts a double value into an integral value in double format.
<code>long round(double a)</code>	Rounds off a double value by first adding 0.5 to it and then returning the largest integer that is less than or equal to this new value.
<code>int round(float a)</code>	Rounds off a float value by first adding 0.5 to it and then returning the largest integer that is less than or equal to this new value.
<code>double sin(double a)</code>	Returns the trigonometric sine of a.
<code>double sqrt(double a)</code>	Returns the square root of a.
<code>double tan(double a)</code>	Returns the trigonometric tangent of an angle.



Java program 5.1

```
import java.lang.Math;
public class chap5_01
{
    public static void main(String[] args)
    {
        double x,y,z;
        int i;
        i=10;
        y=Math.log(10.0);
        x=Math.pow(3.0,4.0);
        z=Math.random(); // random number from 0 to 1
        System.out.println("Value of i is " + i);
        System.out.println("Value of log(10) is " + y);
        System.out.println("Value of 3^4 is " + x);
        System.out.println("A random number is " + z);
        System.out.println("Square root of 2 is " + Math.sqrt(2));
    }
}
```



Sample run 5.2

```
C:\DOCS\notes\INTER\java>javac chap05_1.java
C:\DOCS\notes\INTER\java>java chap05_1
Value of i is 10
Value of log(10) is 2.30259
Value of 3^4 is 81
A random number is 0.0810851
Square root of 2 is 1.41421
```

Java has also two predefined mathematical constants. These are:

- π is equivalent to 3.14159265358979323846
- e is equivalent to 2.7182818284590452354

5.4 Arrays

An array stores more than one value, of a common data type, under a collective name. Each value has a unique slot and is referenced using an indexing technique. For example a circuit with five resistor components could be declared within a program with five simple float declarations. If these resistor variables were required to be passed into a method then all five values would have to be passed through the parameter list. A neater way uses arrays to store all of the values under a common name (in this case R). Then a single array variable can then be passed into any method that uses it.

The declaration of an array specifies the data type, the array name and the number of elements in the array in brackets (`[]`). The following gives the standard format for an array declaration.

```
data_type array_name[];
```

The array is then created using the `new` keyword. For example, to declare an integer array named `new_arr` with 200 elements then the following is used:

```
int new_arr[];
new_arr=new int[200];
```

or, in a single statement, with:

```
int new_arr[]=new int[200];
```

Java program 5.2 gives an example of this type of declaration where an array (`arr`) is filled with 20 random numbers.

Figure 5.1 shows that the first element of the array is indexed 0 and the last element as `size-1`. The compiler allocates memory for the first element `array_name[0]` to the last array element `array_name[size-1]`. The number of bytes allocated in memory will be the number of elements in the array multiplied by the number of bytes used to store the data type of the array.

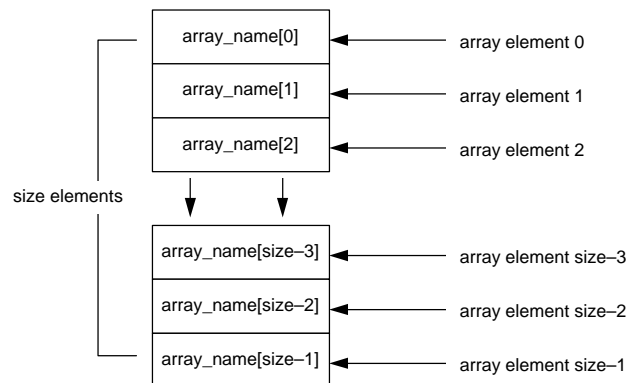


Figure 5.1 Array elements



Java program 5.2

```
public class chap5_02
{
    public static void main(String[] args)
    {
        double arr[]=new double[20];
        int i;

        for (i=0;i<20;i++) arr[i]=Math.random();
        for (i=0;i<20;i++) System.out.println(arr[i]);
    }
}
```



Sample run 5.3

```
C:\java\src\chap5>java chap5_02
0.6075765411193292
0.7524300612559963
0.8100796233691735
0.45045015538577704
0.32390753542869755
0.34033464565015836
0.5079716192482706
0.6426253967106341
0.7691175624480434
0.6475110502592946
0.1416366173783874
0.21181433233783153
0.21758072702009412
0.24203490620407764
0.7587570097412505
0.4470154908107362
0.19823448357551965
0.7340429664182364
0.7402367706819387
0.8975606689180567
```

Another way to create and initialise an array is to define the elements within the array within curly brackets (`{}`). Each element in the array is separated by a comma. The size of the array is then equal to the number of elements in the array. For example:

```
int    arr1[]={-3, 4, 10, 100, 30, 22};
String menus[]{"File", "Edit", "View", "Insert", "Help"};
```

A particular problem in most programming languages (such as C and Pascal) exists when accessing array elements which do not exist, especially by accessing an array element which is greater than the maximum size of the array. Java overcomes this by being able to determine the size of the array. This is done with the `length` field. For example, the previous example can be modified with:

```
for (i=0;i<arr.length;i++)    arr[i]=Math.random();
for (i=0;i<arr.length;i++)    System.out.println(arr[i]);
```

Java program 5.3 gives an example of an array of strings. In this case the array contains the names of playing cards. When run the program displays five random playing cards. Sample run 5.4 shows a sample run.



Java program 5.3

```
public class chap5_03
{
    public static void main(String[] args)
    {
        int cards,pick;
        String  Card[]{"Ace","King","Queen","Jack","10",
```

```

        "9", "8", "7", "6", "5", "4", "3", "2"};

    for (cards=0;cards<5;cards++)
    {
        pick=(int)Math.round((Card.length)*Math.random());
        System.out.print(Card[pick] + " ");
    }
}

```

 **Sample run 5.4**

Ace 10 King 2 3


Multi-dimensional arrays are declared in a similar manner. For example an array with 3 rows and 4 columns is declared with either of the following:

```
int arr[][]=new int[3][4];
```

or if the initial values are known with:

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

where `arr[0][0]` is equal to 1, `arr[1][0]` is equal to 5, `arr[2][3]` is equal to 12, and so on. This is proved with Java program 5.4 and Sample run 5.5.

 **Java program 5.4**

```

public class chap5_04
{
    public static void main(String[] args)
    {
        int row,col;
        int arr[][]={ {1,2,3,4},{5,6,7,8}, {9,10,11,12} };

        for (row=0;row<3;row++)
            for (col=0;col<4;col++)
                System.out.println("Arr["+row+""]["+col+""]="+arr[row][col]);
    }
}

```

 **Sample run 5.5**

```

C:\java\src\chap5>java chap5_05
Arr[0][0]=1
Arr[0][1]=2
Arr[0][2]=3
Arr[0][3]=4
Arr[1][0]=5
Arr[1][1]=6
Arr[1][2]=7
Arr[1][3]=8
Arr[2][0]=9
Arr[2][1]=10
Arr[2][2]=11
Arr[2][3]=12

```

5.5 Exercises

5.5.1 The `classes.zip` file contains all of the standard Java class libraries (a partial listing is shown in Sample run 5.1). Locate this file and determine the following:

- (i) The location of the file.
- (ii) The size of the file.
- (iii) The class directories within it.
- (iv) Some of the classes within it.

5.5.2 Using the methods in Java program 4.9 and the method developed in Exercise 4.8.5, and the standard sine and cosine library methods, write a program which determines the error between the standard library methods and the developed methods. From this, complete Table 5.3.

Table 5.3 Sine and cosine results

<i>Value</i>	<i>Standard cosine function</i>	<i>Developed cosine function</i>	<i>Standard sine function</i>	<i>Developed sine function</i>
2	-0.41614683		0.9092974	
-0.5				
1				
-1				

5.5.3 Compare the result from the exponential method developed in Exercise 4.8.6 with the standard `Math.exp()` library method.

5.5.4 Write a program, using arrays, with a method that returns the smallest value in an array. Refer to Program 4.6 for an outline of the program. Use method overloading to create one for an integer array and another for a double array.

5.5.5 Java program 5.5 can be used to arrange the values in an array in descending order and Sample run 5.6 shows a sample run. Explain its operation.



Java program 5.5

```
public class chap5_05
{
    public static void main(String[] args)
    {
        int array[]={1,4,7,3,-1,4};
        Array Arr = new Array();

        Arr.a=array;
        Arr.show();
    }
}
```

```

        Arr.descend();
        Arr.show();
    }
}
class Array
{
int a[];

    public void show()
    {
        int i;
        for (i=0;i<a.length;i++)
            System.out.println("Val "+i+" "+a[i]);
    }
    public void descend()
    {
        int i,j,temp;
        for (i=0;i<a.length-1;i++)
            for (j=i+1;j<a.length;j++)
                if (a[i]<a[j]) // swap values in array
                {
                    temp=a[i];
                    a[i]=a[j];
                    a[j]=temp;
                }
    }
}
}

```



Sample run 5.6

```

C:\java\src\chap4>java chap04_13
Val 0 1
Val 1 4
Val 2 7
Val 3 3
Val 4 -1
Val 5 4
Val 0 7
Val 1 4
Val 2 4
Val 3 3
Val 4 1
Val 5 -1

```

5.5.6 Write a method that arranges an array in ascending values.

5.5.7 Write a program which fills a two-dimensional array with five rows and four columns. Each row should be filled with the row number, the row number squared, the row number cubed and the row number to the fourth power. Thus the array will contain:

1	1	1	1
2	4	8	16
3	8	27	81
4	16	64	256
5	25	125	625

Rewrite the program using a method to fill the array and another one to display its contents.

5.6 Project

The results in soccer matches between four nations are:

Scotland	3	France	3
Germany	2	Spain	0
Spain	1	Scotland	1
France	1	Germany	1
Scotland	1	Germany	2
Spain	2	France	0

The resulting league table (assuming 3 points for a win and 1 point for a draw) is:

	Played	Won	Draw	Lost	GoalsFor	GoalsAgainst	Pts
Germany	3	2	1	0	5	2	7
Spain	3	1	1	1	3	3	4
Scotland	3	0	1	2	5	6	2
France	3	1	1	2	4	6	2

Write a Java program which stores the team names, their associated points (Pts), their goals for and against, the games they have won, drawn and lost, and the number of games they have played. The program will then sort the teams into their respective league positions. Java program 5.6 shows how the league can be set up and how it can be sorted using the points values. Unfortunately it does not take into account goal difference (which is used to sort the position when the points are the same). The rules for the sort should be:

1. The team with the highest number of points has the highest position.
2. If two teams have the same number of points then the team with the higher goal difference (goals for minus goal against) has the higher position.
3. If two teams have the same number of points and they have the same goal difference then the team with the higher number of goals scored has the higher position.
4. If two teams have identical points, goals scored and goals against then the teams are arranged alphabetically.



Java program 5.6

```
public class chap5_06
{
```

```

public static void main(String[] args)
{
    String names[]={"France","Scotland", "Germany","Spain"};
    int pts[]={2,2,7,4};

    League league = new League();

    league.name=names;
    league.pts=pts;

    league.show();
    league.sort();
    league.show();
}
}
class League
{
String    name[];
int      pts[];

    public void show()
    {
        int i;
        System.out.println("Name  Pts");

        for (i=0;i<pts.length;i++)
        {
            System.out.println(name[i]+" "+pts[i]);
        }
    }

    public void sort()
    {
        int    i,j,temp;
        String tempstr;

        for (i=0;i<pts.length-1;i++)
            for (j=i+1;j<pts.length;j++)
                if (pts[i]<pts[j])
                {
                    temp=pts[i];
                    pts[i]=pts[j];
                    pts[j]=temp;
                    tempstr=name[i];
                    name[i]=name[j];
                    name[j]=tempstr;
                }
    }
}

```

```

C:\java\src\chap4>java chap04_05

```

```

Name      Pts
France    2
Scotland  2
Germany   7
Spain     4
Name      Pts
Germany   7
Spain     4
France    2
Scotland  2

```
