4.3 ARRAYS

With the basic built-in Java data types, each identifier corresponds to a single variable. But when you want to handle sets of values of the same type, say, the first 1000 primes for example – you really don't want to have to name them individually. What you need is an **array**.

An array is a named set of variables of the same type. Each variable in the array is called an **array element**. To refer to a particular element in an array you use the array name combined with an integer value of type ***int***, called an **index**. You are already familiar with how to declare an array in C or C++. In Java, it is slightly different from C/C++. There are three steps to create an array in Java :

a) Declare a variable to hold array

b) Create memory locations

c) Put values into the memory locations

4.3.1 DECLARING ARRAY VARIABLES

You are not obliged to create the array itself when you declare the array variable. The array variable is distinct from the array itself. There are two ways to declare the array variables in Java:

data type arrayname[ ];

or

data type[ ] arrayname;

For example,

```
int sum[ ];
float percentage[ ];
int roll_number[ ];
int primes[ ];
```

The above declaration can be written as :

```
int[ ] sum;
float[ ] percentage;
int[ ] roll_number;
int[ ] primes;
```

Remember that in such types of declarations no memory has been allocated to store the array itself and the number of elements has not been defined.

4.3.2 CREATING AN ARRAY

Creating an array means allocating memory for it. Java allows us to create arrays using ***new*** operator. The syntax is given below :

arrayname = new type[size];

For example,

```
sum = new int[4];
percentage = new float[10];
roll_number = new float[15];
primes = new int[10];
```

The keyword **new** indicates that you are allocating new memory for the array, and int[4] specifies you want capacity for 4 variables of type int in the array. Since each element in the **sum** array is an integer variable requiring 4 bytes, the whole array will occupy 16 bytes, plus 4 bytes to store the reference to the array.

When an array is created like this, all the array elements are initialized to a default value automatically. Similarly, for the percentage, roll_number and primes array also a fixed size will be allocated. The above declared and created arrays are **one-dimensional array** since each of its elements is referenced using one index.

The following figure depicts the meaning of the declaring and creating an integer array named **sum**.

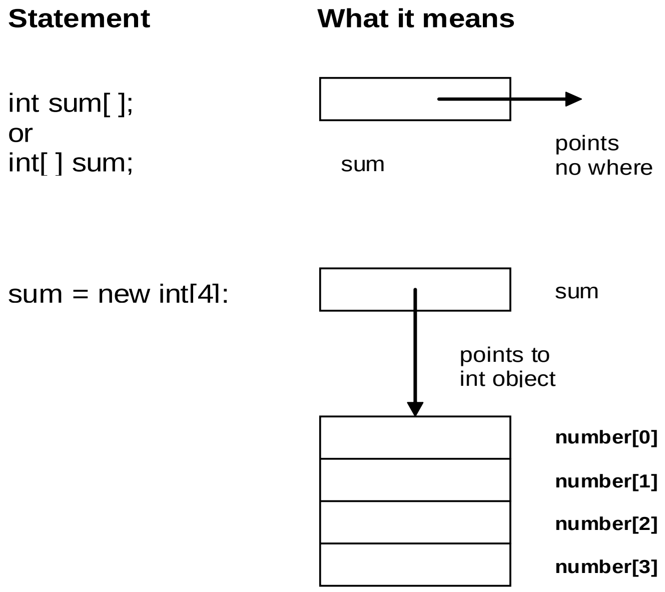

Fig. 4.1 Allocation of memory for an array

4.3.3 INITIALIZING ARRAYS

You can initialize an array with your own values when you declare it, and at the same time determine how many elements it will have. Following the declaration of the array variable, simply add an equal sign followed by the list of element values enclosed between braces.

For example,

```
int[ ] primes = {2, 3, 5, 7, 11, 13, 17};
```

Here, ***primes*** is an array of 7 elements. The array size is determined by the number of initial values; so no other statement is necessary to define the array.

If you specify initializing values for an array, you must include values for *all* the elements. If you only want to set some of the array elements to values explicitly, you should use an assignment statement for each element.

For example:

```
                int[ ] primes = new int[100];
                primes[0] = 2;
                primes[1] = 3;
```

The first statement declares and defines an integer array of 100 elements, all of which will be initialized to zero. The two assignment statements then set values for the first two array elements.

You can also initialize an array with an existing array. For example,

```
        int[ ] even = {4, 6, 8, 10};
        int[ ] number = even;
```

Here, both arrays refer to the same set of elements and you can access the elements of the array through either variable name – for example, even[2] refers to the same variable as number[2].

The following program demonstrates the initialization of one dimensional array :

Program 1: To find the average of given list of numbers.

```
class Average
   {
        public static void main(String args[])
              {
 double nums[] = {50.1, 51.2, 52.3, 53.4, 54.5, 55.6, 56.7, 57.8,
                           58.9, 59.0};
  double result = 0;
  int i;
  for(i=0; i<10; i++)
```

```
      result = result + nums[i];
      System.out.println("Average is " + result / 10);
 }
      }
```

Output :

```
Average is 54.95
```

Array Length

You can refer to the length of the array using **length**, a data member of the array object. In our array example i.e. the array **sum,** its length can e assigned to an another variable as follows:

int size = sum.length

The following program 2 & 3 demonstrates the use of length object for computing the largest and the smallest number from a given list of numbers.

Program 2: To find the smallest and largest number from a given list of number.

```
  class FindNumber
       {

    public static void main(String[] args)
      {

         //array of 10 numbers
     int numbers[] = new int[]{12,83,50,18,22,72,41,29,43,21};
         //assign first element of an array to largest and smallest
         int smallest = numbers[0];
         int largetst = numbers[0];
```

```
        for(int i=1; i< numbers.length; i++)
          {
                if(numbers[i] > largetst)
                        largetst = numbers[i];
                else if (numbers[i] < smallest)
                        smallest = numbers[i];


           }

        System.out.println("Largest Number is    : " + largetst);
        System.out.println("Smallest Number is : " + smallest);
         }
}
```

Output :

```
Largest Number is      : 83
Smallest Number is   : 12
```

Program 3 : To check whether a given number is palindrome or not.

```
class Palindrome
  {

      public static void main(String[] args)
        {

        //array of numbers
int numbers[] = new int[]{2332,42,11,223,24};

        //iterate through the numbers
for(int i=0; i < numbers.length; i++)
{
        int number = numbers[i];
        int reversedNumber  = 0;
        int temp=0;

        /*
        * If the number is equal to it's reversed number, then
        * the given number is a palindrome number.
        * 121 is a palindrome number while 12 is not.
                */

        //reverse the number
       while(number > 0)
         {
        temp = number % 10;
        number = number / 10;
        reversedNumber = reversedNumber * 10 + temp;
          }

       if(numbers[i] == reversedNumber)
 System.out.println(numbers[i] + " is a palindrome number");
       else
System.out.println(numbers[i] + " is not a palindrome number");
 }

       }
 }
```

Output :

```
 2332 is a palindrome number
 42 is not a palindrome number
 11 is a palindrome number
 223 is not a palindrome number
 24 is not a palindrome number
```

Reusing Array Variables :

you can use an array variable to reference different arrays at different points in your program. Suppose you have declared and defined the variable primes as before:

```
        int[ ] primes = new int[10];
                        // Allocate an array of 10 integer elements
```

This produces an array of 10 elements of type *int*. Perhaps you want the array variable primes to refer to a larger array, with 50 elements say. You would simply write :

```
        primes = new int[50];
                        // Allocate an array of 50 integer elements
```

Now the variable primes refer to a new array of values of type *int* that is entirely separate from the original. When this statement is executed, the previous array of 10 elements is discarded, along with all the data values you may have stored in it. The variable primes can now only be used to reference elements of the new array.

4.4 MULTIDIMENSIONAL ARRAYS

A table organized in rows and columns is a very effective means for communicating many different types of information. In Java, we represent table as two-dimensional arrays. The general syntax for allocating a two dimensional array is :

type [ ][ ] arrayname = new type[rows][cols];

For example,

```
int [ ][ ] matrix = new int [3][5];
```

This creates a two dimensional matrix having 3 rows and 5 columns that can store 15 integer values. We are already familiar with C or C++ how to access the elements of a two dimensional array. In Java also, the same techniques are used. For example, to refer to the elements at the second column of the third row we will indicate it as:

```
matrix [2][1]
```

We can also initialize a two dimensional array as shown below :

```
int [  ] [  ] matrix = {
                        { 10, 5,  -3, 9, 2 },
                        {  1 , 0, 14, 5, 6 },
                        { -1, 7,  4,  9,  2 }
                          };
```

There is no limit to the number of dimensions an array can have. We can declare three-dimensional, four-dimensional, and higher dimensional arrays. However, arrays with a dimension higher than 2 are not frequently used in object-oriented languages.

The following figure depicts the representation of the matrix array in memory :

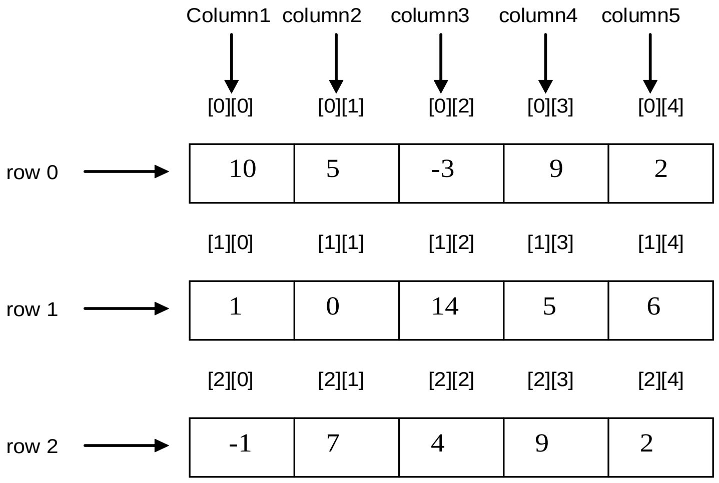

Fig. 4.2 Representation of matrix array

The following program demonstrate the use of two dimensional array in Java programming. This program will simply display the initialized list of elements as output.

Program 4 : To display the elements of a two dimensional array in matrix form.

```
class Matrix
    {
       public static void main(String args[])
        {
               int mat[][] = {
                                { 10, 5, -3, 9, 2 },
                                { 1, 0, 14, 5, 6 },
                                { -1, 7, 4, 9, 2 },
                                 };

               int i, j;
               for(i=0; i<3; i++)
               {
                  for(j=0; j<5; j++)
                  System.out.print(mat[i][j] + " ");
                   System.out.println();
               }
            }
        }
```

Output :

```
10,  5,  -3,  9, 2
 1,   0, 14,  5, 6
-1,   7,   4,  9, 2
```

Multidimensional arrays are actually arrays of arrays. It means we can create or allocate the sub arrays (columns). It is possible to create sub arrays of different lengths which will look like a triangle. For example, the following program creates a two dimensional array in which the sizes of the second dimension are unequal or the array looks like a triangle :

Program 5 : To display the elements of a two dimensional matrix in triangular form.

```
class TriangularArray
  {
    public static void main(String args[])
     {
       int arr[][] = new int[4][];
       int i;
       for(i=0; i<4; i++)        // Creates triangular array
       arr[i] = new int[i+1];
       int j, k = 0;
       for(i=0; i<4; i++)        // Assigns the elements
       for(j=0; j<i+1; j++)
         {
              arr[i][j] = k;
              k++;
          }
         for(i=0; i<4; i++)    // Display the elements
             {
              for(j=0; j<i+1; j++)
              System.out.print(arr[i][j] + " ");
              System.out.println();
              }
          }
    }
```

Output :

```
0
1 2
3 4 5
6 7 8 9
```