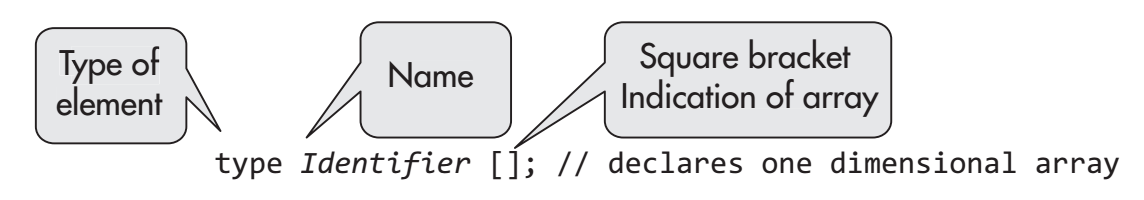
**Introduction**

In programming language, an array is a structure consisting of a group of elements of the same type. When a large number of data values of the same type are to be processed, it can be done efficiently by declaring an array of the data type and by making the data values elements of the array. The complete data gets represented by a single object with a single name in the computer memory, and at the same time, an individual array element is a variable of the type declared. Thus, an array is a sequence of objects of the same data type. The type of data that the array holds becomes the type of the array, which is also called base type of the array.

**Declaration and Initialization of Arrays**

The declaration starts with the type, followed by an identifier and a square bracket, and ends with a semicolon as illustrated below.



Graphical user interface, text, application

Description automatically generated

**Initialization**

**Diagram, text

Description automatically generated**

Alternative Array Declaration syntax

Text, application

Description automatically generated

**Storage of Array in Computer Memory**

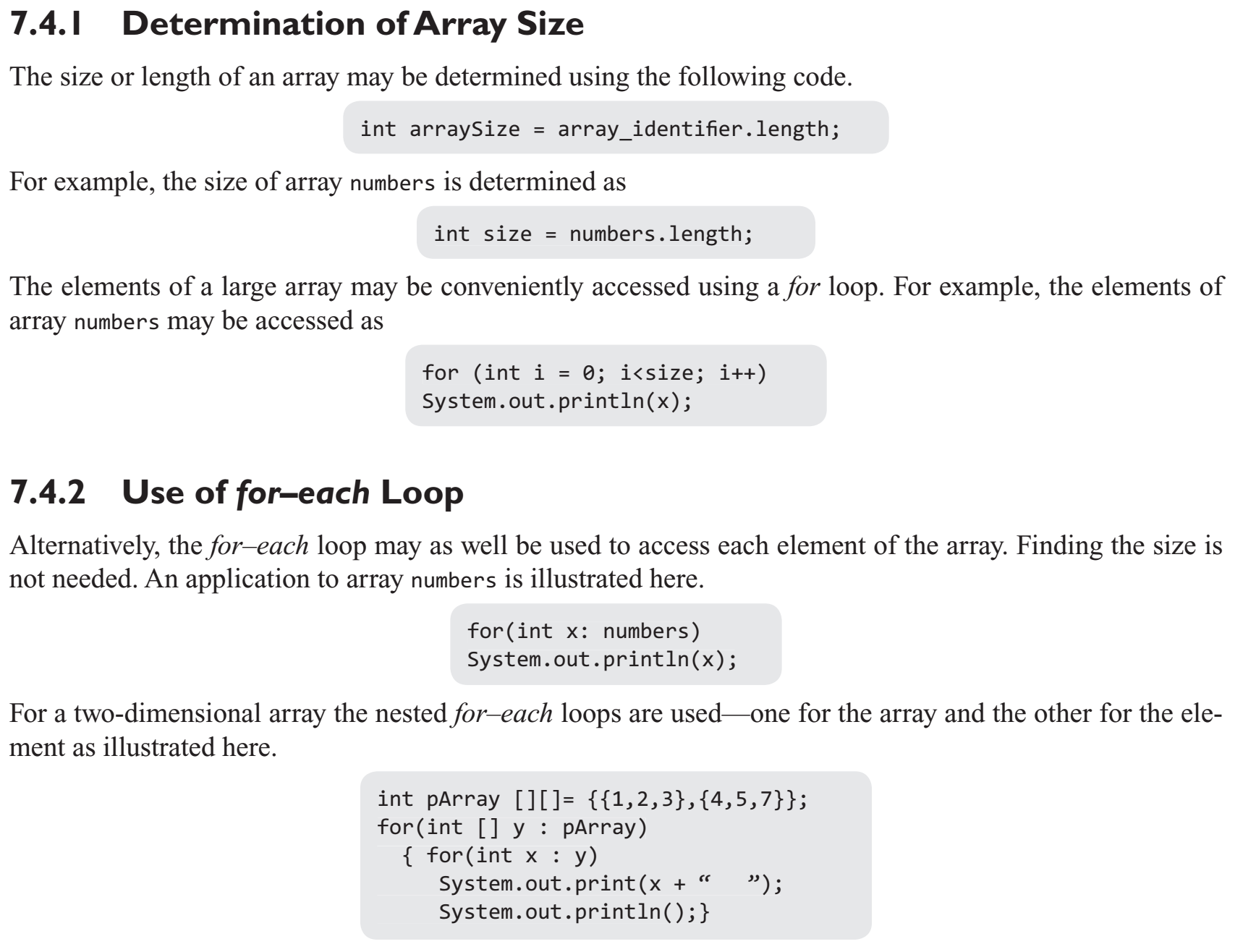
int[] numbers = new int[4];

Table

Description automatically generated

**Accessing Elements of Arrays**

The individual member of an array may be accessed by its index value. The index value represents the place of element in the array. Fig. 7.2 shows the array int [] numbers as defined earlier. The array resides on four memory blocks, each of which has 4 bytes. The first element space is represented by numbers [0], that is, its index value is 0. The memory space of the second element is represented by numbers [1], the third by numbers [2] and the fourth by numbers [3]. Here 0, 1, 2, and 3 are the index values. Note that the value of an array element is different from its index value.



**Operations on Array Elements**

Diagram

Description automatically generated

public class ArrayOperations

{

public static void main(String[] args)

{

int []array1 = new int [] {1,2,3,4,5};

int [] array2; array2 = new int [5];

for(int k = 0; k<5; k++)

array2[k] = 2\*k; //assigning values to elements

System.out.println("The elements of array1 are as below.");

display(array1);

System.out.println("The elements of array2 are as below.");

display(array2);

for (int j = 0; j<5; j++)

{

array1[j] += 3;// 3 is added to elements of array1

array2[j] \*=2;

}// elements of array2 are multiplied by 2

System.out.println("New elements of Array1 after adding 3 are as below.");

display(array1);

System.out.println("New elements of array2 after multiplying by 2 are as below.");

display(array2);

} // method definition for display of array

public static void display(int[] array)

{

for (int z: array)

System.out.print(z + " ");

System.out.println();

}

}

**Assigning Array to Another Array**

Unlike in C and C++ languages, in Java, an array may be assigned as a whole to another array of same data type. In this process, the second array identifier, in fact, becomes the reference to the assigned array. The second array is not a new array, instead only a second reference is created. This is illustrated in Program 7.7. In this program, array1 is assigned to array2. Then, array1 is modified. array2 also gets modified, which shows it is not an independent array.

class AssigningArraytoAnother

{

public static void main (String args[])

{

int [] array1, array2;

array1 = new int [5];

array2 = new int [5];

for(int k = 0; k<5; k++)

array1[k] = 2\*k;

array2 = array1;// assigning array1 to array2

System.out.println("The elements of array1 are as below.");

display(array1);

System.out.println("The elements of array2 are as below.");

display(array2);

for (int j = 0; j<5; j++)

array1[j] += 3;// Adding 3 to all elements of array1

System.out.println("New elements of array1 are as below.");

display(array1);

System.out.println("New elements of array2 are as below.");

display(array2);

} // display method definition

public static void display( int [] array)

{

for (int i = 0; i < array.length; i++)

System.out.print(array[i]+" ");

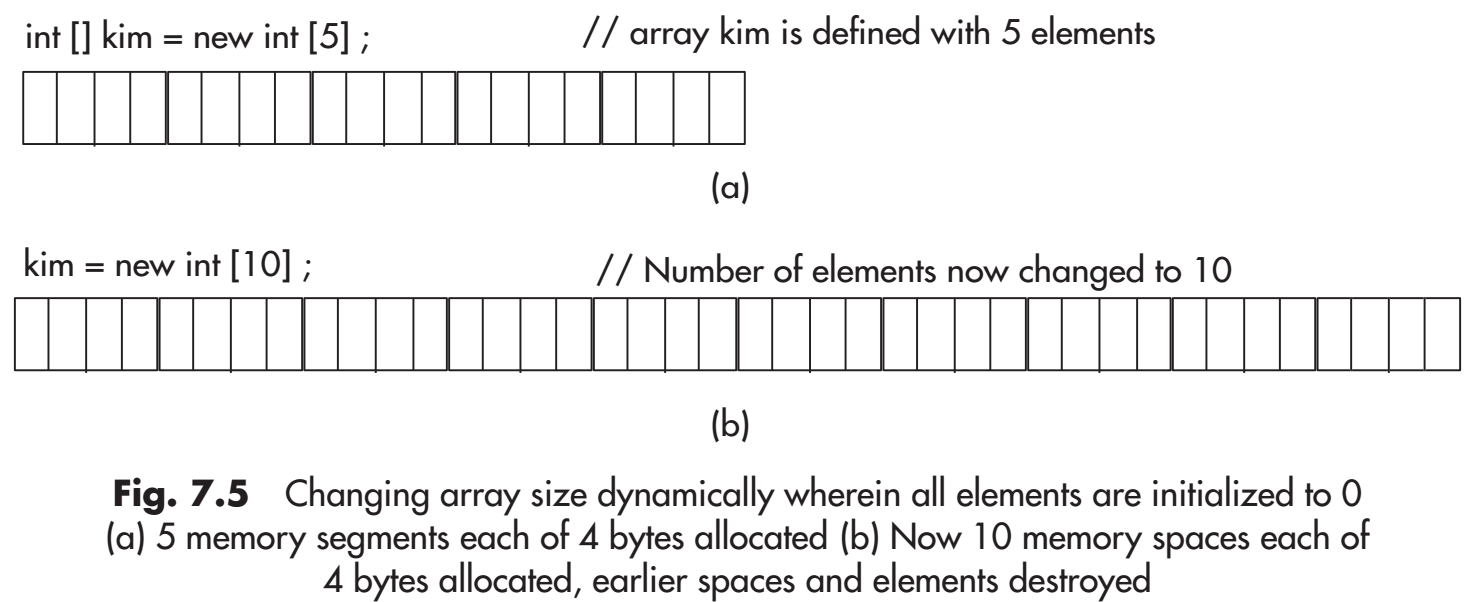
System.out.println();

}

}

**Dynamic Change of Array Size**

The number of elements (size) of the array may change during the execution of the program. This feature is unlike C and C++ wherein the array once declared is of fixed size, that is, the number of elements cannot be changed. In Java, however, you may change the number of elements by dynamically retaining the array name. In this process, the old array is destroyed along with the values of elements. This is illustrated in Fig. 7.5, wherein array kim is first defined to have 5 elements. It is again defined to have 10 elements with the same name. The identifier (name) of array is not tied to the array once declared. The new array with 10 elements is created.



class DynamicArray

{

public static void main (String args[])

{

int [] kim = new int [5];

int i,k;

for (i = 0;i<5; i++)

kim[i] = 1 + i;

for (k = 0;k<5; k++)

System.out.print(kim[k] + " ");

System.out.println();

kim = new int [10];

for (k = 0;k<10; k++)

System.out.print(kim[k] + " ");

System.out.println();

}

}

**Class Arrays**

The package java.util defines the class Arrays with static methods for general processes that are carried out on arrays such as sorting an array for full length of the array or for part of an array, binary search of an array for the full array or part of an array, for comparing two arrays if they are equal or not, for filling a part or the full array with elements having a specified value, and for copying an array to another array. The sort method of Arrays class is based on quicksort technique. The methods are applicable to all primitive types as well as to class objects.

Methods of Class Arrays

1.sort

2.binaySearch

3.equals

4.fill

5.copyOf

6.toString

7.deepToString

8.hashCode

Example Code 1

import java.util.Arrays;

public class ArraysExample1

{

public static void main(String[] args)

{

int [] intArray = new int[]{5,7,2,1,8,6,8,9};

char [] charArray = new char [] {'u','y', 't','a', 'm','f'};

float [] fArray = new float[] {3.4f,2.5f,8.5f,6.4f,9.5f,3.2f};

Arrays.sort(charArray);

for(char ch: charArray)

System.out.print(ch + " ");

System.out.println();

Arrays.sort(intArray, 2,5);// sort from index 2 to 5

for(int x: intArray)

System.out.print(x + " ");

System.out.println();

Arrays.fill(fArray, 1,4, 2.2f);

for(float y: fArray)

System.out.print(y + " ");

System.out.println();

}

}

Example Code 2

import java.util.Arrays;

public class ArraysExample2

{

public static void main(String[] args)

{

int [] intArray = new int[]{5,7,2,1,8,6,8,9};

int [] Array = new int[]{5,7,2,1,8,6,8,9};

System.out.println("Are the intArray and Array equal? "+ Arrays.equals(intArray, Array));

char [] charArray = new char [] {'u','y', 't','a', 'm','f'};

float [] fArray =new float[]{3.4f,2.5f,8.5f,6.4f,9.5f};

Arrays.sort(charArray);

for(char ch: charArray)

System.out.print(ch + " ");

System.out.println();

int index = Arrays.binarySearch(charArray, 't');

if(index>=0)

System.out.println(" The key is at index value: " + index);

else

System.out.println("The key value is not in the array.");

for( float flo: fArray)

System.out.print(flo + " ");

System.out.println();

float floatArray[] = Arrays.copyOf(fArray,8);

for(float y: floatArray)

System.out.print(y + " ");

System.out.println();

}

}

Example Code 3

import java.util.Arrays;

public class ArraysExample3

{

public static void main(String[] args)

{

int [] intArray = new int[]{10,20,30,80,90,70,40,36};

System.out.println(Arrays.toString(intArray));

char [] charArray = new char [] {'u','y', 't','a', 'm','f'};

System.out.println(Arrays.hashCode(charArray));

float [] fArray = new float[]{3.4f, 2.5f, 8.5f, 6.4f, 9.5f, 3.2f};

System.out.println(Arrays.toString(fArray));

int array [] = Arrays.copyOfRange( intArray, 2,6);

for (int x: array)

System.out.print(x +" ");

System.out.println();

}

}

Example code 4

import java.util.Arrays;

public class ArraysExample4

{

public static void main(String[]args)

{

String s; int [][]number = {{1,2,3}, {4,5,6}};

s = Arrays.deepToString(number);

System.out.println(s);

}

}

**Two-Dimensional Array**

In Java, a two-dimensional array is treated as an array of arrays, and each of these arrays may have a different number of elements. This is elaborated in the next section. However, there is no difference in normal operations on arrays with this concept.

class Array2D {

public static void main (String args [])

{

double darray [][]= {{1,1,1,1},{2,2,2,2},{3,3,3,3}};

int i,j;

System.out.println("Elements of darray are as below.");

for (i = 0; i<3; i++)

{

for (j = 0; j<4; j++)

System.out.print(darray [i][j] + " ");

System.out.println();}

System.out.println("Length of darray = " + darray.length);

System.out.println("Length of darray[1] = " + darray[1]. length);

System.out.println("Length of darray[2] = " + darray[2]. length);

System.out.println("Elements of darray after squaring are as below.");

for (i = 0; i<3; i++)

{

for (j = 0; j<4; j++) // squaring each element of array before output

System.out.print(Math.pow(darray[i][j], 2.0) + " ");

System.out.println();

}

}

}

**Arrays of Varying Lengths**

In Java, a two-dimensional array is treated as an array whose elements are one-dimensional arrays, which may have different sizes, that is, different number of elements. This is not possible in C or C++. Thus, in Java, we may declare a two-dimensional array as

Graphical user interface, text, application

Description automatically generated

public class Array2D2

{

public static void main(String[] args)

{

int [][] array = new int [3][];

array[0] = new int[]{ 6, 7};

array[1] = new int []{10,11,12,14};

array[2] = new int [] {2,3,4};

for (int i =0; i<3;i++)

{

for (int j = 0; j<array[i].length; j++)

System.out.print(array[i][j] + " ");

System.out.println();

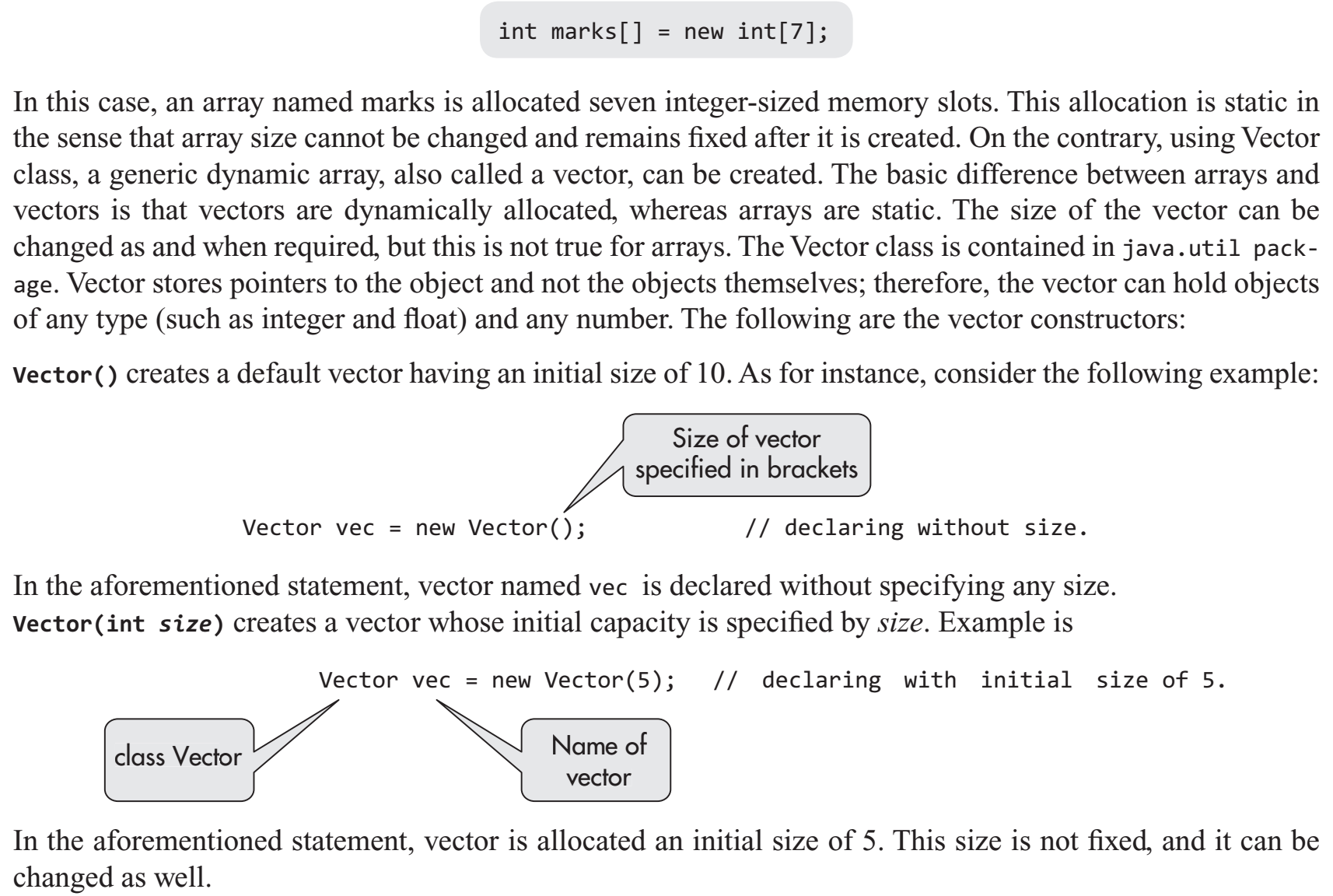
}

}

}

**Arrays as Vectors**

Similar to Arrays, vectors are another kind of data structure that is used for storing information. Using vector, we can implement a dynamic array. As we know, an array can be declared in the following way:



Vector(int size, int incr) creates a vector with initial capacity specified by size and increment is specified by incr. The increment is the number of elements added in each reallocation cycle. Its example includes

Vector(int size, int incr) creates a vector with initial capacity specified by size and increment is specified by incr. The increment is the number of elements added in each reallocation cycle. Its example includes

Vector vec = new Vector(5,2);

The statement declares a vector of size 5 and its space will increase by 2 when more than 5 elements are added. During the reallocation of space, the amount of extra space allocated depends on the increment that is specified when this vector is created. If this increment is not specified, the vector size is doubled in each allocation cycle. Vector(Collection c) creates a vector that contains elements of collection c.

Vectors have a number of advantages over arrays. Some of these include the following:

1. Vectors are dynamically allocated, and therefore, they provide efficient memory allocation.

2. Size of the vector can be changed as and when required.

3. They can store dynamic list of objects.

4. The objects can be added or deleted from the list as per the requirement.

Vectors are useful if the size of array is not known in advance

Graphical user interface, text, application

Description automatically generated

import java.util.\*;

public class VectorList

{

public static void main(String args[])

{

Vector vec = new Vector(7,2);// declaring vector

Integer [] anArray = new Integer[7];

System.out.println("Initial size:" + vec.size());

System.out.println("Initial capacity:" + vec.capacity());

vec.addElement(new Integer(7));// adding the integer value

vec.addElement(new Integer(10));

vec.addElement(new Integer(3));

vec.addElement(new Float(5.6));// adding the float value

vec.addElement(new Float(10.8));

vec.addElement(new Integer(23));

System.out.println("Enhanced capacity after addition:" + vec.capacity()); // displaying the capacity after addition

vec.addElement(new Integer(5));

System.out.println("current capacity is:" + vec.capacity());

System.out.println("First element:" + (Integer)vec.firstElement());

System.out.println("last element:" + (Integer)vec.lastElement());

System.out.println("Element at fourth position is:" + vec.get(4));

}

}