Chapter-6
Arrays Strings and Vectors

Arrays:

- An array is a group of contiguous or related data items that share a common name.
- A particular value is indicated by writing a number called **index number or subscript** in brackets after the array name.

Example:

- salary[10] represents the salary of the 10th employee.
- The complete set of values is referred to as an **array**, the individual values are called **elements**.
- Arrays can be of any variable type.

One-dimensional arrays :

- A list of items can be given one variable name using only one subscript and such a variable is called a **single-subscripted variable** or a **one-dimensional array.**
- The subscript can begin with number 0. That is x[0] is allowed.
- we may create the variable number as follows int number[ ] = new int[5]; and the computer reserves five storage location

The values to the array elements can be assigned as follows:

```
number[0] = 35;
number[1] = 40;
number[2] = 20;
number[3] = 57;
number[4] = 19;
```

These elements may be used in programs just like any other Java variable. For example, the following are valid statements:

```
aNumber   =  number[0] + 10;
number[4] =  number[0] + number[2];
number[2] =  x[5] + y[10];
value[6]  =  number[i] * 3;
```

The subscript of an array can be integer constants, integer variables like i , or expressions that yield integers.

Creating an array

Creation of an array involves three steps:

1. Declare the array
2. Create memory locations
3. Put values into the memory locations.

Declaration of Arrays

Arrays in Java may be declared in two forms:

Form1 type arrayname[ ];

Form2 type [ ] arrayname;

Examples:

```
int       number[ ];
float     average[ ];
int[ ]    counter;
float[ ]  marks;
```

We do not enter the size of the arrays in the declaration.

Creation of Arrays :

- After declaring an array, we need to create it in the memory.
- Java allows us to create arrays using **new** operator only.

Syntax:

```
arrayname = new type[size];
```

Example:

```
number      = new int[5];
average     = new float[10];
```

Initialization of Arrays :

- The final step is to put values into the array created. This process is known as initialization .
- This is done using the **array subscripts** .

arrayname[subscript]=value;

Example:

```
number[0]  = 35;
number[1]  = 40;
```

Java creates arrays starting with a subscript of 0 and ends with a value one less than the size specified

Initialization can also be done as follows:

Syntax:

```
type arrayname[ ] = {list of values};
```

Example:

int number [ ] = {35, 40, 20, 57, 19};

Array Length:

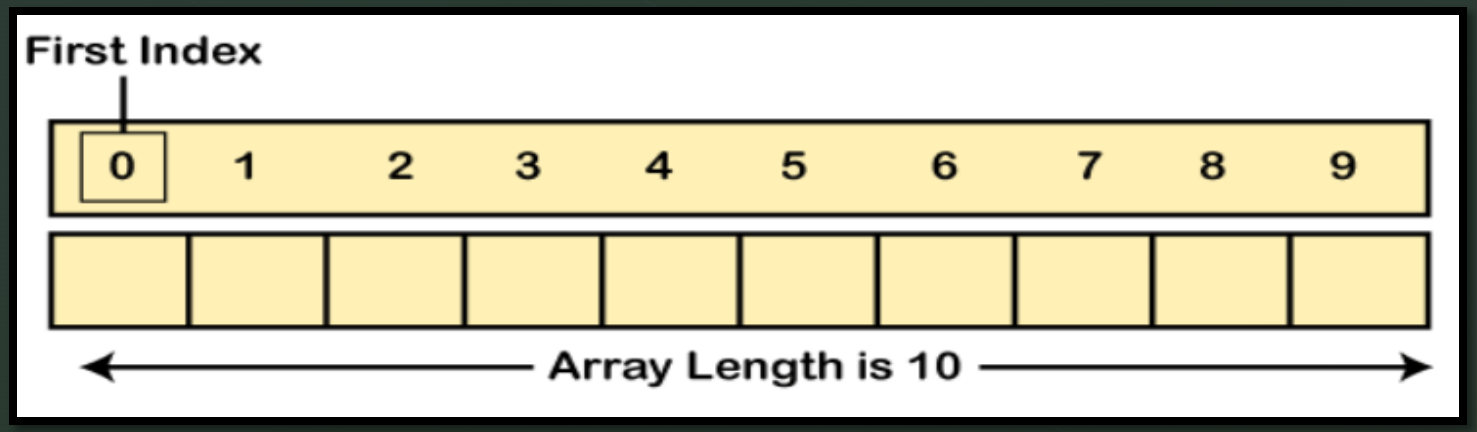
In **Java,** the array length is the number of elements that an array can holds.

The length property can be invoked by using the **dot (.) operator** followed by the array name. We can find the length of int[], double[], String[], etc.

For example:

1.int[] arr=**new int**[5];

2.int arrayLength=arr.length



Two Dimensional Array:

- 2D array can be defined as an **array of arrays**.
- The 2D array is organized as matrices which can be represented as the collection of rows and columns.

Creating two-dimensional array :

```
int myArray[ ][ ];
myArray = new int[3][4];
Or
int myArray[ ][ ] = new int[3][4];
```

This creates a table that can store 12 integer values, four across and three down.

Strings:

Strings represent a sequence of characters. The easiest way to represent a sequence of characters in Java is by using a character array.

Example:

```
char charArray[ ] = new char[4];
charArray[0] = 'J';
charArray[1] = 'a';
charArray[2] = 'v';
charArray[3] = 'a';
```

- strings are class objects and implemented using two classes **String and StringBuffer**.
- A Java string is an instantiated object of the String class.

Strings may be declared and created as follows:

```
String  stringName;
StringName = new String ("string");
```

Example:

String firstName;

firstName = new String("Anil");

The above statements may be combined as follows:

String firstName = new String("Anil");

Like arrays, it is possible to get the length of string using the **length method** of the String class.

int m = firstName.length( ) ;

Java strings can be concatenated using the **+ operator**.

Example:

```
String fullName  =  name1 + name2;
String city1     =  "New" + "Delhi";
```

String Arrays :

String itemArray[ ] = new String[3]; We can assign the strings to the itemArray element by element using three different statements or more efficiently using a for loop

String Methods

The String class defines a number of methods that allow us to accomplish a variety of string manipulation tasks. Some Most Commonly Used String Methods

Method Call	Task performed
`s2 = s1.toLowerCase;`	Converts the string s1 to all lowercase
`s2 = s1.toUpperCase;`	Converts the string s1 to all Uppercase
`s2 = s1.replace('x','y');`	Replace all appearances of x with y
`s2 = s1.trim();`	Remove white spaces at the beginning and end of the string s1
`s1.equals(s2)`	Returns 'true'if s1 is equal to s2
`s1.equalsIgnoreCase(s2)`	Returns 'true'if s1 = s2, ignoring the case of characters
`s1.length()`	Gives the length of s1
`s1.ChartAt(n)`	Gives nth character of s1
`s1.compareTo(s2)`	Returns negative if s1< s2, positive if s1 > s2, and zero if s1 is equal s2
`s1.concat(s2)`	Concatenates s1 and s2
`s1.substring(n)`	Gives substring starting from n^{th} character
`s1.substring(n, m)`	Gives substring starting from n^{th} character up to m^{th} (not including m^{th})
`String.ValueOf(p)`	Creates a string object of the parameter p (simple type or object)
`p.toString()`	Creates a string representation of the object p
`s1.indexOf('x')`	Gives the position of the first occurrence of 'x' in the string s1
`s1.indexOf('x',n)`	Gives the position of 'x' that occurs after nth position in the string s1
`String.ValueOf(Variable)`	Converts the parameter value to string representation

String Buffer Class:

- StringBuffer is a peer class of String.
- String creates strings of fixed_length, StringBuffer creates strings of flexible length that can be modified in terms of both length and content.
- Characters and substrings can be inserted in the middle of a string, or append another string to the end.

s1.indexOf('x')	Gives the position of the first occurrence of 'x' in the string s1
s1.indexOf('x',n)	Gives the position of 'x' that occurs after nth position in the string s1
s1.substring(n)	Gives substring starting from n^{th} character
s1.substring(n, m)	Gives substring starting from n^{th} character up to m^{th} (not including m^{th})
s1.insert(n, s2)	Inserts the string s2 at the position n of the string s1
s1.setChartAt(n, 'x')	Modifies the nth character to x

Vectors:

- Java does not support the concept of variable arguments to a function and this can be achieved in Java through the use of the Vector class contained in the **java.util package**.
- This class can be used to create a generic dynamic array known as vector that can hold objects of any type and any number.
- The objects do not have to be homogenous.
- Arrays can be easily implemented as vectors.

Vectors are created like arrays as follows:

```
Vector intVect = new Vector( );   // declaring without size
Vector list   =  new  Vector(3);  //  declaring with size
```

Advantages of vectors over arrays.

1. It is convenient to use vectors **to store objects**.
2. A vector can be used to **store a list of objects that may vary in size**.
3. We can **add and delete objects from the list** as and when required.

Important Vector Methods

Method Call	Task performed
`list.addElement(item)`	Adds the item specified to the list at the end
`list.elementAt(10)`	Gives the name of the 10th object
`list.size()`	Gives the number of objects present
`list.removeElement(item)`	Removes the specified item from the list
`list.removeElementAt(n)`	Removes the item stored in the nth position of the list
`list.removeAllElements( )`	Removes all the elements in the list
`list.copyInto(array)`	Copies all items from list to array
`list.insertElementAt (item, n)`	Inserts the item at nth position

Wrapper classes

- Vectors cannot handle primitive data types like int, float, long, char, and double.
- Primitive data types may be converted into object types by using the wrapper classes contained in the java. Lang package.

Wrapper Classes for Converting Simple Types

Simple Type	Wrapper Class
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long

- The wrapper classes have a number of unique methods for handling primitive data types and objects.

Converting Primitive Numbers to Object Numbers Using Constructor Methods

Constructor Calling	Conversion Action
`Integer IntVal = new Integer(i);`	Primitive integer to Integer object
`Float FloatVal = new Float(f);`	Primitive float to Float object
`Double DoubleVal = new Double(d);`	Primitive double to Double object
`Long LongVal = new Long(l);`	Primitive long to Long object

- I, f, d, and l are primitive data Values denoting int, float, double and long data types. They may be constants or Variables.

Converting Object Numbers to Primitive Numbers Using typeValue() method

Method Calling	Conversion Action
`int i = IntVal.intValue( );`	Object to primitive integer
`float f = FloatVal.floatValue( );`	Object to primitive float
`long l = LongVal.longValue( );`	Object to primitive long
`double d = DoubleVal.doubleValue( );`	Object to primitive double

Converting Numbers to Strings Using to String() Method

Method Calling	Conversion Action
`str = Integer.toString(i)`	Primitive integer to string
`str = Float.toString(f);`	Primitive float to string
`str = Double.toString(d);`	Primitive double to string
`str = Long.toString(l);`	Primitive long to string

Converting String Objects to Numeric Objects Using the Static Method ValueOf( )

Method Calling	Conversion Action
`DoubleVal = Double.ValueOf(str);`	Converts string to Double object
`FloatVal = Float.ValueOf(str);`	Converts string to Float object
`IntVal = Integer.ValueOf(str);`	Converts string to Integer object
`LongVal = Long.ValueOf(str);`	Converts string to Long object

Note: *These numeric objects may be converted to primitive numbers using the typeValue()*

Converting Numeric Strings to Primitive Numbers Using Parsing Methods

Method Calling	Conversion Action
`int i = Integer.parseInt(str);`	Converts string to primitive integer
`long l = Long.parseLong(str);`	Converts string to primitive long

Note: *parseInt() and parseLong( ) methods throw a NumberFormatException if the value of the str does not represent an integer*