ARRAYS AND VECTORS

Session Objectives

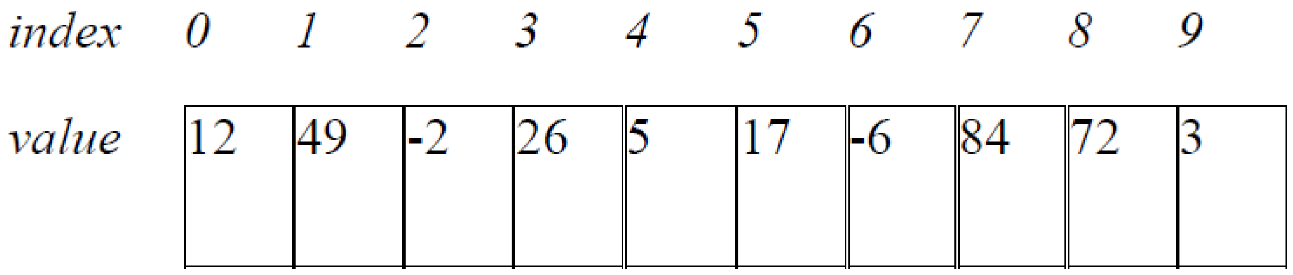
- Upon completing this session you should be able to:
  - Declare, populate and access arrays
  - Identify the different array types
  - Explain the importance of arrays
  - Explain the use of the vector class

Introduction to Arrays

- Program data is stored in the variables and takes the memory spaces randomly.
- However, when we need the data of the same type to store in the contiguous memory allocations, individual variables fall short.
- Thus, an array is a named memory location used to store a collection of data/ variables of the same type.
- Array contains the values which are implicitly referenced through the index values. So to access the stored values in an array we use indexes.
- Suppose an array contains "n" integers. The first element of this array will be indexed with the "0" value and the last integer will be referenced by "n-1" indexed value.
- We also need to store values (populate) and be able to access them from the array

Arrays introduction

- Element: One value in an array.
- Index: A 0-based integer, which is used to access an element from an array.
- We usually draw an array as a row or column of boxes.
- Example: an array of ten integers (scores)



- A particular value in an array is referenced using the array name followed by the index in brackets
- For example, the expression *scores[3]* refers to the value 26 (which is the 4th value in the array)
- That expression represents a place to store a single integer, and can be used wherever an integer variable can
- For example, it can be assigned a value, printed, or used in a calculation
- An array stores multiple values of the same type.
- That type can be primitive types or objects. Therefore, we can create an array of integers, or an array of characters, or an array of String objects, etc.
- In Java, the array itself is an object. Therefore the name of the array is a object reference variable, and the array itself is instantiated separately

Declaring Array Variables

- Syntax:
  - arraytype[] arrayname e.g.
- String[] args; or
- Int[] age;
- or
- arrayType arrayName[ ]; // works but not preferred way.
  - Example: int Marks[];

Creating Arrays

- You can create an array by using the ***new*** operator with the following syntax:

  arrayName = new dataType[arraySize];
- The above statement does two things:
  - It creates an array using new dataType[arraySize]
  - It assigns the reference of the newly created array to the variable arrayName.
- The length of the array – arraySize- is specified between [ ] brackets.
- Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

  dataType[] arrayName = new dataType[arraySize];

  eg int [] marks= new int[4];
- Alternatively you can create arrays as follows:

  dataType[] arrayName = {value0, value1, ..., valuek};

  eg int [] marks={45,30,50,40}

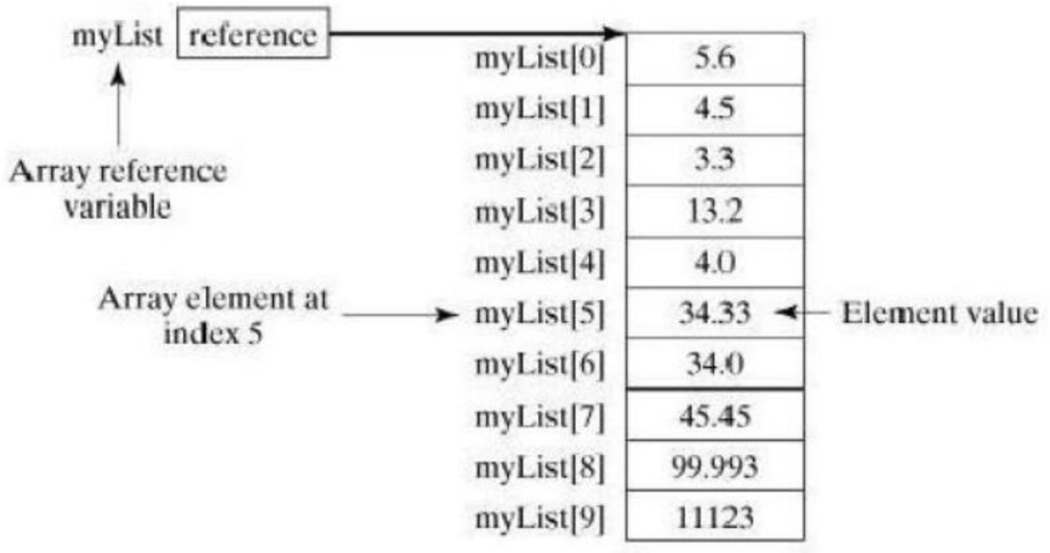
  char myArray={'A', 'B', 'C', 'D'};
- The array elements are accessed through the index. Array indices are 0-based; that is, they start from 0 to arrayName.length-1.

Example

- Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList:

  double[] myList = new double[10];

- Below diagram represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



Accessing array elements

- Assigning a value to an array element:

```
<array name> [ <index> ] = <value> ;
```

Example:

```
scores[0] = 27;
scores[3] = -6;
```

- Using an array element's value in an expression:

```
<array name> [ <index> ]
```

Example:

```
System.out.println(numbers[0]);
if (numbers[3] < 0) {
System.out.println("Element 3 is negative.");
}
```

Bounds Checking

- Once an array is created, it has a fixed size. An index used in an array reference must specify a valid element. That is, the index value must be in bounds (0 to N-1)
- The Java interpreter will throw an exception if an array index is out of bounds. This is called automatic bounds checking
- For example, if the array codes can hold 100 values, it can only be indexed using the numbers 0 to 99. If count has the value 100, then the following reference will cause an ArrayOutOfBoundsException:

```
System.out.println (codes[count]);
```

Example:

```
int[] data = new int[10];
System.out.println(data[0]); // okay
System.out.println(data[-1]); // exception
System.out.println(data[9]); // okay
System.out.println(data[10]); // exception
```

Initializer Lists

- An initializer list can be used to instantiate and initialize an array in one step
- The values are delimited by braces and separated by commas

Examples:

```
int[] units = {147, 323, 89, 933, 540, 269, 97, 114, 298,476};
char[] letterGrades = {'A', 'B', 'C', 'D', 'F'};
```

- Note that when an initializer list is used:
  - the new operator is not used
  - no size value is specified
- The size of the array is determined by the number of items in the initializer list;
- An initializer list can only be used in the declaration of an array

Arrays and for loops

- Arrays are very commonly used with for loops that pass over each element and process it in some way:
- Example (print each element of an array):

```
for (int i = 0; i < 10; i++) {
System.out.print(scores[i] + " ");
}
```

- Sometimes we assign each array element a value in a for loop.
- Example:

```
for (int i = 0; i < 10; i++) {
scores[i] = 2 * i;
}
```

The .length field

- Each array object has a public constant called *length* that stores the size of the array
- An array has a field named length that returns its total number of elements.
- **General syntax:**
- <array name> .length
- Notice that it doesn't have parentheses like a String's .length() .

- **Example**:

```
for (int i = 0; i < scores.length; i++) {
System.out.print(scores[i] + " ");
}
```

Example 1:

- //Program to store five numbers in an array and calculate the average of numbers

```
class Average {
public static void main(String s[]) {
double num[] = {10.1, 11.2, 13.3, 14.4, 15.5};
double result =0.0;
int i;
for (i = 0; i<num.length ; i++) {
result = result + num[i] ; // result += no[i];
}
System.out.println("Average is of the numbers is: " + result / 5) ;}
}
```

- **Exercise**:
- Rewrite the program such that the array elements are read through the keyboard, use Scanner.

```
//Program receives elements from key board and displays the items entered
import java.util.Scanner;
public class ArrayKeyboardInput{
public static void main(String[] args)   {
int num;
Scanner input=new Scanner(System.in);
System.out.print(“Please enter the number of elements you wish to store in the array: ");
//read the number of elements
num=input.nextInt();
//create an array in the memory of length 10
int[] Myarray = new int[10];
System.out.println("Enter the elements of the array: ");
for(int i=0; i<num; i++)  {
//reading array elements from the user
Myarray[i]=input.nextInt();  }
System.out.println(“The Array elements entered are: ");
// accessing array elements using the for loop
for (int i=0; i<num; i++)   {
System.out.println(Myarray[i]);  }
}
}
```

Example 2:

```
//Arrays of arrays of varying lengths
class ArrayOfArrays {
public static void main(String s[]) {
double myNumbers [][] = new double[2][]; //defines 2 elements, each is an array. 2 elements are //allocated, each of which can hold a I D array
myNumbers [0] = new double [2]; //1st array has 2 elements
myNumbers [1] = new double [3]; //2nd array has 3 //elements
myNumbers [0][0] = 10.0;
myNumbers [0][1] = 20.0 ;
myNumbers [1][0] = 100.5;
myNumbers [1][1] = 200.5;
myNumbers [1][2] = 300.5;
for(int i=0; i< myNumbers.length; i++)
for(int j=0; j< myNumbers.length; j++)
System.out.println("value is " + myNumbers [i][j]) ;
}
}
```

Example

- Program showing how to create, initialize, and process arrays:

```
public class TestArray {
public static void main(String[] args) {
double[] myList = {1.9, 2.9, 3.4, 3.5};
// Print all the array elements
for (int i = 0; i < myList.length; i++) {
System.out.println(myList[i] + " ");
}
// Summing all elements
double total = 0;
for (int i = 0; i < myList.length; i++) {
total += myList[i];
}
System.out.println("Total is " + total);
// Finding the largest element
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
if (myList[i] > max) max = myList[i];
}
System.out.println("Max is " + max);}}
```

Example 3:

- //Program to read numbers from user using input Dialogs and displays them in reverse

```
import java.io.*;
import javax.swing.JOptionPane;
class JReverse_Numbers{
public static void main(String args[]) throws IOException{
int numbers[] = new int[10];
System.out.println("The size of the array is:"+numbers.length);
System.out.println("Enter array elements");
for (int index = 0;index<numbers.length;index++){
System.out.println("Number "+ index + ":");
numbers[index] = Integer.parseInt(JOptionPane.showInputDialog("Array Elements"));
}
System.out.println("Numbers in reverse");
for (int index = numbers.length-1;index>=0;index--){
System.out.println("The numbers are:\t" +numbers[index]);
}
System.out.println ( );
}}
```

Two-Dimensional Arrays

- A one-dimensional array stores a simple list of values
- A two-dimensional array can be thought of as a table of values, with rows and columns
- A two-dimensional array element is referenced using two index values
- To be precise, a two-dimensional array in Java is an array of arrays

Multidimensional Arrays

- An array can have as many dimensions as needed, creating a multidimensional array
- Each dimension subdivides the previous one into the specified number of elements
- Each array dimension has its own length constant
- Because each dimension is an array of array references, the arrays within one dimension could be of different lengths
- A k-dimensional array can be created with either of the following methods:
  - Using the new operator
  - Using the k-dimensional initializer

Examples

- double mat1[ ][ ] = new double[4][5]; // This creates a 4 x 5 two-dimensional array
- int mat2[ ] [ ] = {{1, 2, 3}, {4, 5, 6}}; // This creates a 2 x 3 two-dimensional array.

- The elements are initialized as:

```
mat2[0][0]=1 mat2[1][0]= 4
mat2[0][1]=2 mat2[1][1]= 5
mat2[0][2]=3 mat2[1][2]= 6
```

- Multidimensional arrays are often processed using for statements.
- To process all the items in a two-dimensional array, a for statement needs to be nested inside another.

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

- The following loop stores 0 into each location in two dimensional array A :

```
for (int row = 0; row < 3; row++) {
for (int column = 0; column < 4; column++) {
A [row][column] = 0;
}
```

Example

- Using a mathematical function in the Java Math class and handling of two–dimensional arrays, the following example calculates the max – min value.

```
public class MaxMin{
public static void main(String[]args) {
double mat[ ][ ]= { {2.3, 5.1, 9.9},{8.3, 4.5, 7.7},{ 5.2, 6.1, 2.8}} ;
int n = mat.length;
int m= mat[0].length;
double maxmin = 0.0;
for (int j = 0; j < m; j++){
double min = mat[j][0];
for (int i = 1; i <n ; i ++){
min = Math.min(min,mat[i][j]);
}
if (j==0){
maxmin = min;
}
else{
maxmin = Math.max(maxmin, min);
}
}
System.out.println("The max-min value is" + maxmin);} }
```

Arrays as Parameters

- An entire array can be passed to a method as a parameter
- Like any other object, the reference to the array is passed, making the formal and actual parameters aliases of each other
- Changing an array element in the method changes the original
- An array element can be passed to a method as well, and will follow the parameter
- passing rules of that element's type
  - Use square brackets [ ] in the parameter declaration:

```
public static double sum (double [] array) {
for (int i=0; i < array.length; i++) .. .
}
```

- This works for any size array
  - Use the .length attri

Arrays of Objects

- The elements of an array can be object references
- When you construct an array of objects (such as Strings), each element initially stores a special reference value called null.
- The following declaration reserves space to store 25 references to String objects
- String[] words = new String[5];

index	*0*	*1*	*2*	*3*	*4*
value	null	null	null	null	null

- null means 'no object'. It does NOT create the String objects themselves.
- Each object stored in an array must be instantiated separately
- The array elements should be initialized somehow:

```
for (int i = 0; i < words.length; i++) {
words[i] = "this is string #" + (i + 1);
}
words[0] = words[0].toUpperCase();
```

Example

- Program creates an array of strings and adds three names to the array. It then converts the names to lower case letters using the toLowerCase()); method.

```
class ArrayOfStrings{
public static void main(String args[]){
String names [] = {"CYNTHIA", "DERICK", "PAUL"};
for (int i = 0; i < names.length; i++)
System.out.println(names[i].toLowerCase());
}
}
```

Array example -Dialogs

```
import javax.swing.JOptionPane;
public class InputDialogExample {
public static void main(String[] args) {
Integer[] options = {2, 3, 5, 7, 9, 11};
int n = (Integer)JOptionPane.showInputDialog(null, "Pick a number
that is not prime:",
"Prime numbers", JOptionPane.QUESTION_MESSAGE, null, options,
options[2]);
System.out.println(n);
}
}
```

Why are arrays useful?

- We can use arrays to store a large amount of data without declaring many variables.
  - Example: Read in a file of 1000 numbers, then print out the numbers in reverse order.
- Arrays can help us to group related data into elements.
  - Example: For a given school exam, open a file full of exam scores and count up how many students got each score from 0 through 100.
- Arrays let us hold on to data and access it in random order.
  - Example: Read a file full of babies' names, store each name's data as an element in a large array, and then examine it later to find many names that the user types.

The Vector Class

- Arrays are fixed in size at declaration time -cannot grow or shrink dynamically during run time. This leads to wastage of memory as well as creating difficulty in case you wanted to add more elements to the program at run time.
- Java provides a Vector class which can grow or shrink during the run of the program
- An object of class Vector is similar to an array in that it stores multiple values. However, a vector only stores objects and does not have the indexing syntax that arrays have
- The methods of the Vector class are used to interact with the elements of a vector
- The Vector class is part of the java.util package
- An important difference between an array and a vector is that a vector can be thought of as dynamic, able to change its size as needed

- Each vector initially has a certain amount of memory space reserved for storing elements. If an element is added that doesn't fit in the existing space, more room is automatically acquired.
- The Vector class is implemented using an array. Whenever new space is required, a new, larger array is created, and the values are copied from the original to the new array.
- To insert an element, existing elements are first copied, one by one, to another position in the array. Therefore, the implementation of Vector in the API is not very efficient for inserting elements