Programming In Java

Array and ArrayList

Arrays

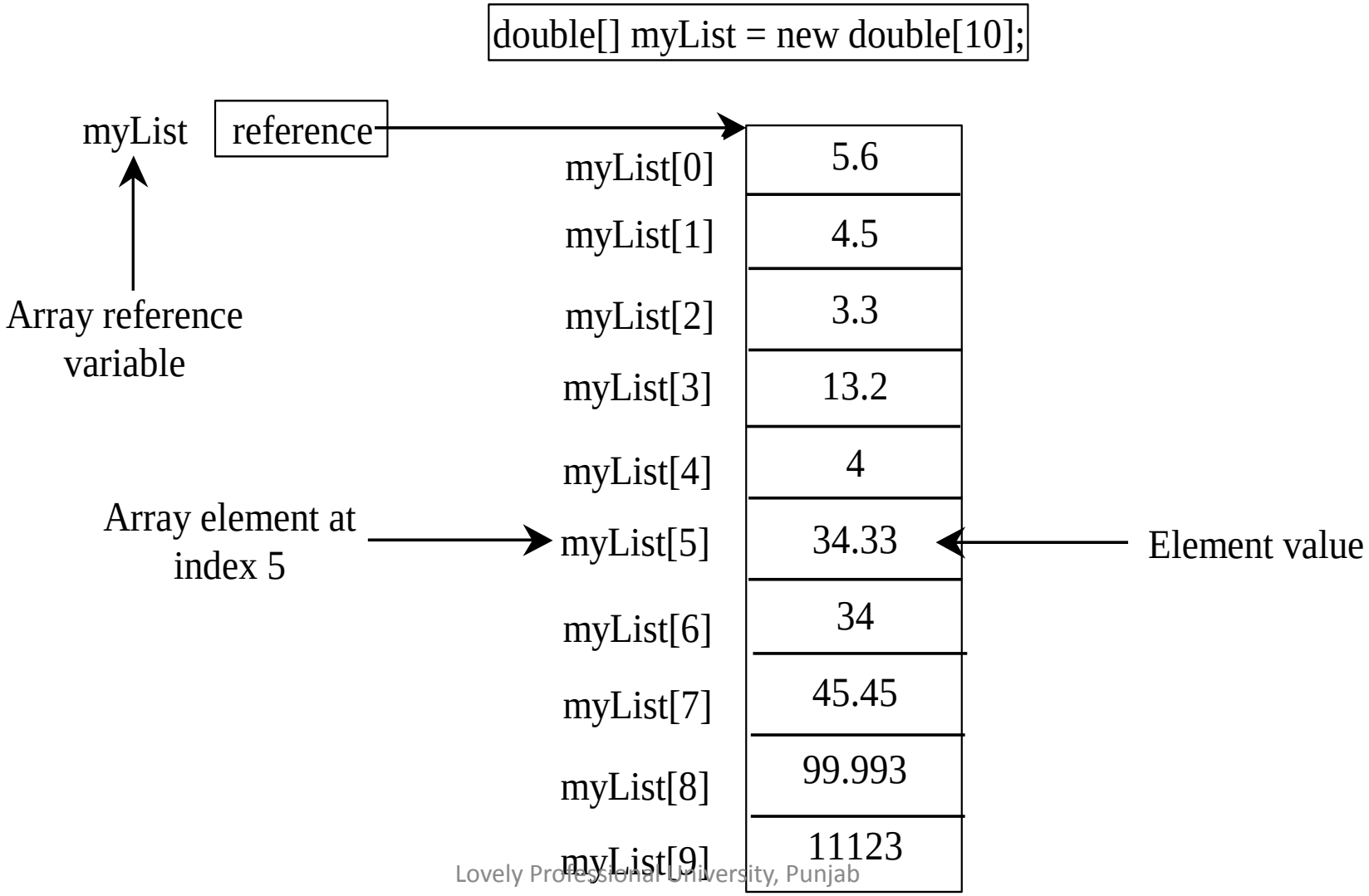
- Single dimensional,
- Multidimensional and
- Jagged Array
- Declaring and using ArrayList class

Array

- Definition:

  An array is a group/collection of variables of the same type that are referred to by a common name.
- Arrays of any type can be created and may have one or more dimensions.
- A specific element in an array is accessed by its index (subscript).
- Examples:
  - **C**ollection of numbers
  - **C**ollection of names

Introducing Arrays



Declaring Array Variables

- `datatype[] arrayRefVar;`

  Example:

  ```
  double[] myList;
  ```

- `datatype arrayRefVar[];` // This style is //allowed, but not preferred

  Example:

  ```
  double myList[];
  ```

Creating Arrays

```
arrayRefVar = new datatype[arraySize];
```

Example:

```
myList = new double[10];
```

`myList[0]` references the first element in the array.

`myList[9]` references the last element in the array.

Declaring and Creating in One Step

- ```
  datatype[] arrayRefVar = new
                           datatype[arraySize];
  ```

  ```
  double[] myList = new double[10];
  ```

- ```
  datatype arrayRefVar[] = new
                           datatype[arraySize];
  ```

  ```
  double myList[] = new double[10];
  ```

Example

STEP 1 : (Declaration)

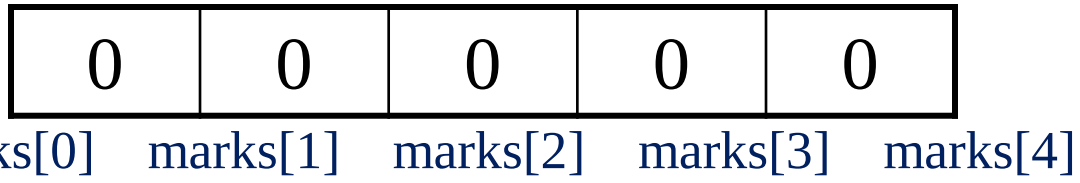
int marks[];

marks → null

STEP 2: (Memory Allocation)

marks = new int[5];

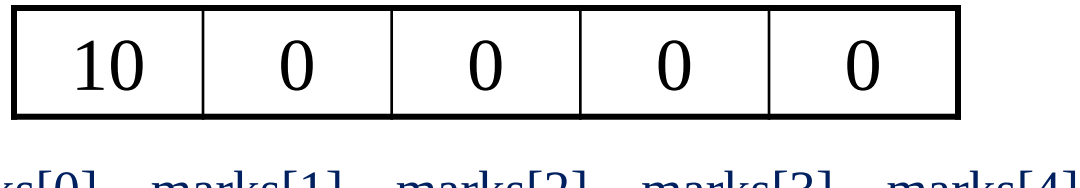
marks →



STEP 3: (Accessing Elements)

marks[0] = 10;

marks →



- Size of an array can’t be changed after the array is created.

- Default values:
  - zero (0) for numeric data types,
  - false for boolean types

- Length of an array can be obtained as:

  array_ref_var.length

- The array's length is available as a final instance variable length.

Example

```
class Demo_Array
    {
        public static void main(String args[])
                {
                        int marks[];
                        marks = new int[3];

                         marks[0] = 10;
                        marks[1] = 35;
                        marks[2] = 84;
    System.out.println("Marks of 2nd student=" + marks[1]);
                }
    }
```

The Length of an Array

Once an array is created, its size is fixed. It cannot be changed. You can find its size using

arrayRefVar.length

For example,

myList.length returns 10

Indexed Variables

- The array elements are accessed through the index. The array indices are *0-based*, i.e., it starts from 0 to arrayRefVar.length-1. In the example in Figure , myList holds ten double values and the indices are from 0 to 9.

- Each element in the array is represented using the following syntax, known as an *indexed variable*:

  arrayRefVar[index];

  For Example:

```
myList[2] = myList[0] + myList[1];
```

Array Initializers

- Declaring, creating, initializing in one step:

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

This shorthand syntax must be in one statement.

This shorthand notation is equivalent to the following statements:

```
double[] myList = new double[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

- **An Array of Characters is Not a String**
- In the Java programming language, unlike C, an array of char is not a String, and neither a String nor an array of char is terminated by '\u0000' (the NUL character).

- A String object is immutable, that is, its contents never change, while an array of char has mutable elements.
- The method toCharArray in class String returns an array of characters containing the same character sequence as a String.
- The class StringBuffer implements useful methods on mutable arrays of characters.

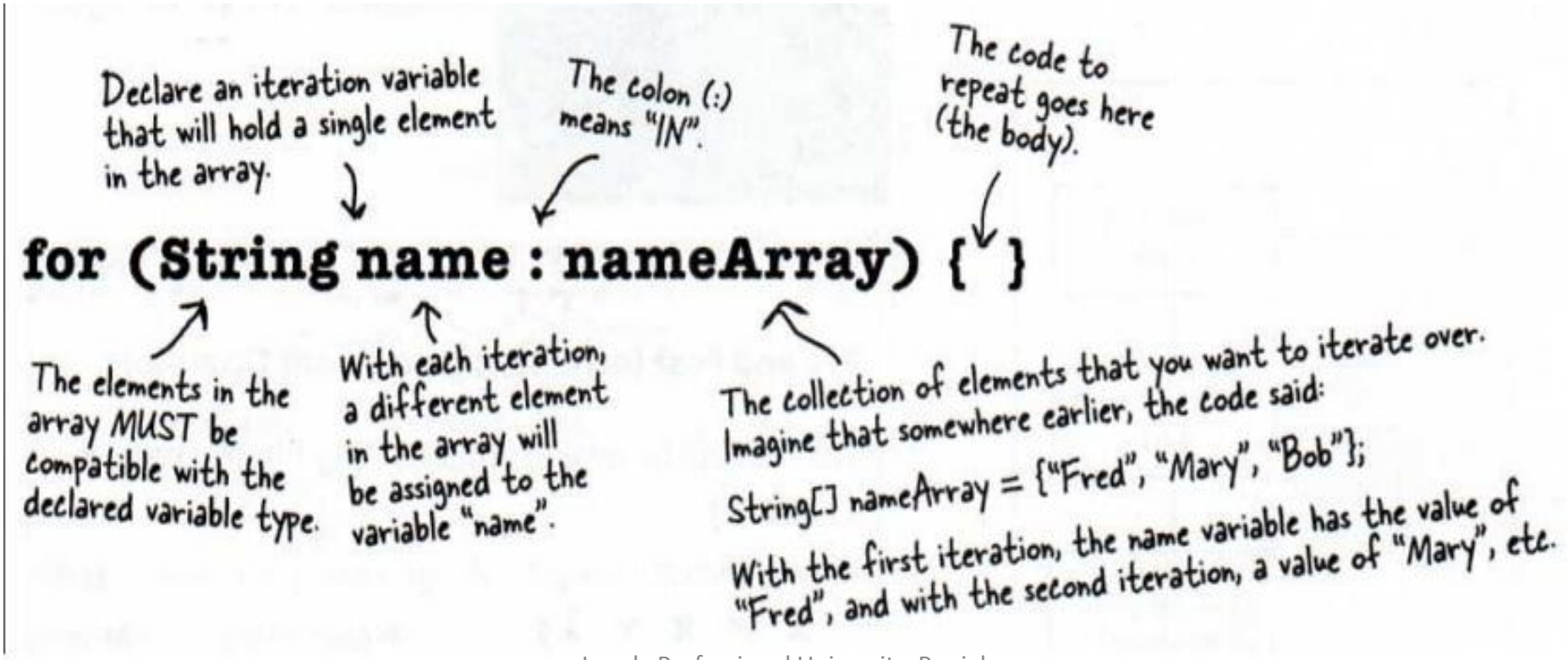
- The enhanced for loop

- iterates through the elements of an array or a collection without using a counter (thus avoiding the possibility of "stepping outside" the array).

The syntax of an enhanced for statement is:

for(type identifier : arrayName) { }

- Ex.

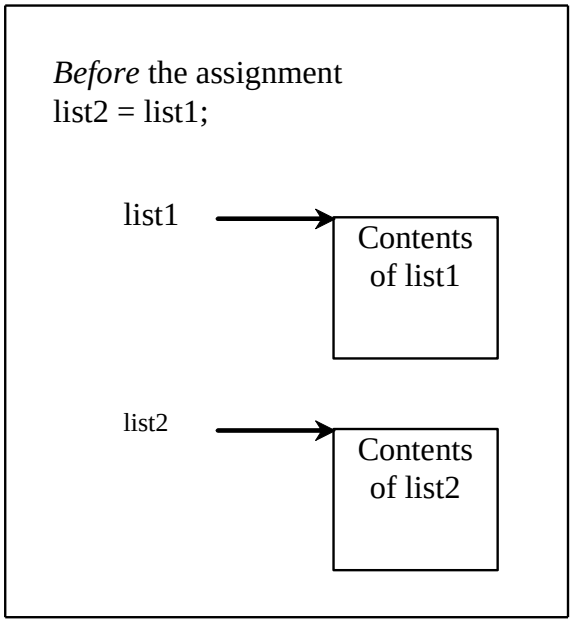


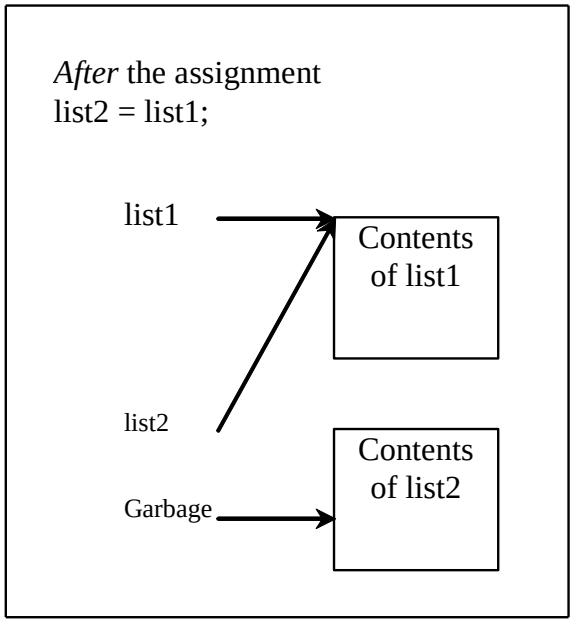
```
class Myarray
{
 public static void main(String [] ar)
  {
    int a[]= {1,2,3,4,5,6,7,8,9};
    int total = 0;
        for(int i : a)
     total +=i;
   System.out.println("sum of the elements " +total);
}
}
```

Copying Arrays

Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

list2 = list1;





Copying Arrays Cont…

- There are three ways to copy arrays:
  - Use a loop to copy individual elements one by one.
  - Use the static arraycopy method in the System class.
  - Use the clone method to copy arrays;

Copying Arrays Cont…

Using a loop:

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new
  int[sourceArray.length];

for (int i = 0; i < sourceArrays.length; i++)
   targetArray[i] = sourceArray[i];
```

The `arraycopy` Utility

```
arraycopy(sourceArray, src_pos,
  targetArray, tar_pos, length);
```

length -- This is the number of array elements to be copied

Example:

```
System.arraycopy(sourceArray,
  0, targetArray, 0,
  sourceArray.length);
```

```
public class arr
{
    public static void main(String[] args) {

        int arr1[] = { 1, 2, 3, 4, 5 };
        int arr2[] = new int[arr1.length];
        System.arraycopy(arr1, 0, arr2, 1, 3);

        System.out.print(arr2[0] + " ");
        System.out.print(arr2[1] + " ");
        System.out.print(arr2[2] + " ");
        System.out.print(arr2[3] + " ");
        System.out.print(arr2[4] + " ");
  }
}
```

The clone method

- In Object class
- `int[] sourceArray = {2, 3, 1, 5, 10};`
- `int[] targetArray = sourceArray.clone();`

Passing Arrays to Methods

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

Invoke the method

```
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
```

Invoke the method

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Anonymous array

- Java uses pass-by-value to pass arguments to a method. There are important differences between passing the values of variables of primitive data types and passing arrays.
- For an argument of a primitive type, the argument's value is passed.
- For an argument of an array type, the value of the argument is a reference to an array; this reference value is passed to the method. Semantically, it can be best described as pass-by-sharing, i.e., the array in the method is the same as the array being passed. So if you change the array in the method, you will see the change outside the method.

```
public classTest
{   public static void main(String[] args)
    {     int x = 1;                          // x represents an int value
          int[] y = new int[10];        // y represents an array of int values
          m(x,y) ;
       System.out.println("x is "+ x);
       System.out.println("y[0] is "+ y[0]);
      }
public static void m (int number , int [] numbers)
     {
        number = 1001;// Assign a new value to number
        numbers[0] = 5555;// Assign a new value to numbers[0]
      }
}
```

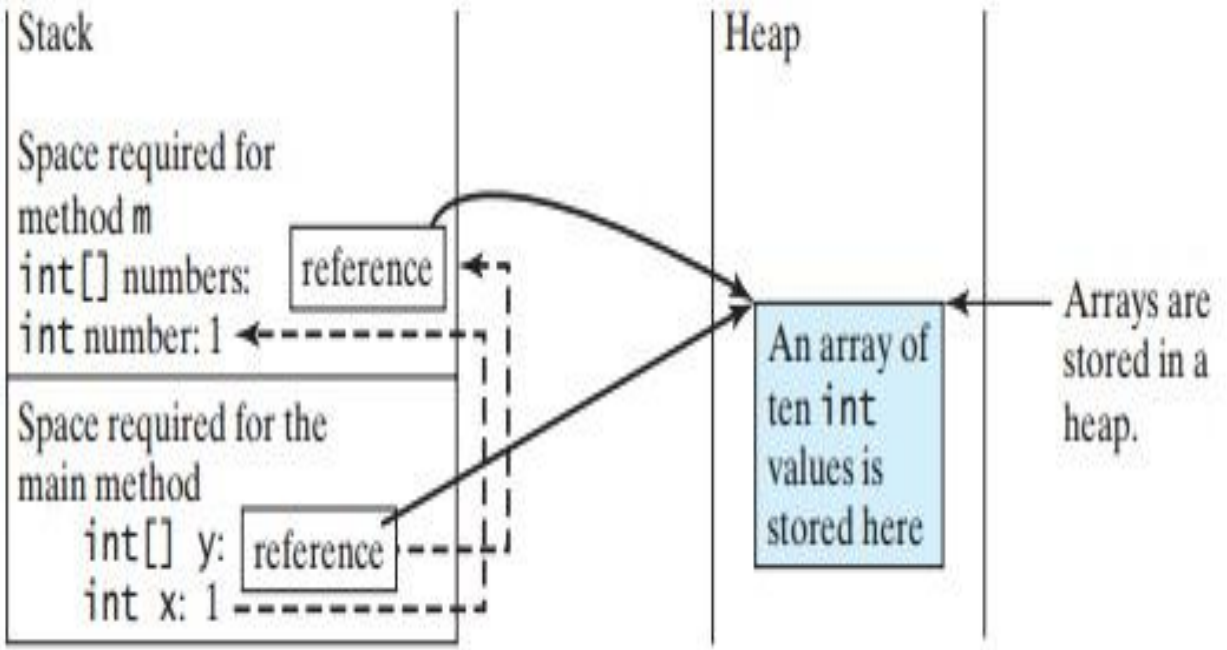


FIGURE The primitive type value in `x` is passed to `number`, and the reference value in `y` is passed to `numbers`.

- # Returning an Array from a Method

- You can pass arrays when invoking a method. A method may also return an array.
- For example,
- public static int[] reverse(int[] list)
- { int[] result = new int[list.length];
- for(inti = 0, j = result.length - 1; 5 i < list.length; i++, j--)
- {
- result[j] = list[i];
- }

return result;

}

For example, the following statement returns a new array list2 with elements 6,5,4,3,2,1.

int [] list1 = {1,2,3,4,5,6};

int[] list2 = reverse(list1);

- # Variable-Length Argument List
- We can pass variable number of argument of the same type of method. The parameter in the method is declared as follows:

  type name… parameter name
- Java treats variable length parameter as array.
- We can pass an array or a variable number of argument to a variable-length parameter.

```
public class arr
{
   public static void main(String[] args) {
     display();
     display(1,2,3,4,5,6,7);
     display(new int [] {1,2,3});
   }
  public static void display(int ... a)
   {
     //System.out.println(a);

    if(a.length == 0)
        System.out.println("no argument is passed");
    else
        System.out.println("no of argument is "+a.length );
   }
}
```

Multidimensional Arrays

- Declare/Create Two-dimensional Arrays

```
// Declare array ref var
dataType[][] refVar;

// Create array and assign its reference to
  variable
refVar = new dataType[10][10];

// Combine declaration and creation in one
  statement
dataType[][] refVar = new dataType[10][10];

// Alternative syntax
dataType refVar[][] = new dataType[10][10];
```

Declaring Variables of Two-dimensional Arrays and Creating Two-dimensional Arrays

```
int[][] matrix = new int[10][10];
```

or

```
int matrix[][] = new int[10][10];
matrix[0][0] = 3;

for (int i = 0; i < matrix.length; i++){
  for (int j = 0; j < matrix[i].length; j++){
    matrix[i][j] = (int)(Math.random() * 1000);
     }
  }
```

Declaring, Creating, and Initializing Using Shorthand Notations

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

```
int[][] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

Lengths of Two-dimensional Arrays

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

array.length

array[0].length

array[1].length

array[2].length

array[3].length

array[4].length ArrayIndexOutOfBoundsException

```
class TwoDimArr
    {
      public static void main(String args[])
            {
              int twoD[][]= new int[4][5];
              int i, j, k = 0;
              for(i=0; i<4; i++)
                    for(j=0; j<5; j++)
                        {
                          twoD[i][j] = k;
                          k++;
                        }
               for(i=0; i<4; i++)
                        {
                    for(j=0; j<5; j++)
                          System.out.print(twoD[i][j] + " ");
                    System.out.println();
                        }
             }
    }
```

Jagged Arrays

- Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as *a jagged array*. For example,

```
int[][] matrix = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```

matrix.length is 5
matrix[0].length is 5
matrix[1].length is 4
matrix[2].length is 3
matrix[3].length is 2
matrix[4].length is 1

- If we don’t know the values in a ragged array in advance, but know the sizes, we can create a ragged array using the syntax that follows:
- int[][] matrix = new int[5][] ;
- matrix[0] = new int[5];
- matrix[1] = new int[4];
- matrix[2] = new int[3];
- matrix[3] = new int[2];
- matrix[4] = new int[1];
- You can now assign values to the array. For example,
- matrix[0][3] = 50;
- matrix[4][0] = 45;

The ArrayList Class

We can create an array to store objects. But the array's size is fixed once the array is created. Java provides the ArrayList class that can be used to store an unlimited number of objects.

java.util.ArrayList	
+ArrayList()	Creates an empty list.
+add(o: Object) : void	Appends a new element o at the end of this list.
+add(index: int, o: Object) : void	Adds a new element o at the specified index in this list.
+clear(): void	Removes all the elements from this list.
+contains(o: Object): boolean	Returns true if this list contains the element o.
+get(index: int) : Object	Returns the element from this list at the specified index.
+indexOf(o: Object) : int	Returns the index of the first matching element in this list.
+isEmpty(): boolean	Returns true if this list contains no elements.
+lastIndexOf(o: Object) : int	Returns the index of the last matching element in this list.
+remove(o: Object): boolean	Removes the element o from this list.
+size(): int	Returns the number of elements in this list.
+remove(index: int) : Object	Removes the element at the specified index.
+set(index: int, o: Object) : Object	Sets the element at the specified index.

Differences and Similarity between Arrays and ArrayList

	Array	ArrayList
Creating an array/ArrayList	`Object[] a = new Object[10]`	`ArrayList list = new ArrayList()`
Accessing an element	`a [index]`	`list.get(index)`
Updating an element	`a [index] = "London";`	`list.set(index, "London");`
Returning size	`a.length`	`list.size()`
Adding a new element		`list.add("London")`
Inserting a new element		`list.add(index, "London")`
Removing an element		`list.remove(index)`
Removing an element		`list.remove(Object)`
Removing all elements		`list.clear()`

Differences and Similarity between Arrays and ArrayList Cont…

- Once an array is created, its size is fixed.
  - We can access an array element using the square bracket notation (e.g., a[index]).
- When an ArrayList is created, its size is 0.
  - We cannot use the get and set method if the element is not in the list.
- It is easy to add, insert, and remove elements in a list, but it is rather complex to add, insert, and remove elements in an array.
- We have to write the code to manipulate the array in order to perform these operations

```
import java.util.ArrayList;
class Box {
        private double width;
        private double height;
        private double depth;
        Box(Box ob) {                   // pass object to constructor
                width = ob.width;
                height = ob.height;
                depth = ob.depth;
                }
        Box(double w, double h, double d) {
                width = w;
                height = h;
                depth = d;
                }
        Box() {
                width = -1; // use -1 to indicate
                height = -1; // an uninitialized
                depth = -1; // box
            }
        Box(double len) {
                width = height = depth = len;
            }
        double volume() {
                return width * height * depth;
              }
            }
```

```
public class TestArrayList
        {
public static void main(String[] args)
       {
 // Create a list to store cities
            java.util.ArrayList cityList = new java.util.ArrayList();

 // Add some cities in the list
            cityList.add("London");
            // cityList now contains [London]
            cityList.add("Denver");
 // cityList now contains [London, Denver]
            cityList.add("Paris");
 // cityList now contains [London, Denver, Paris]
            cityList.add("Miami");
 // cityList now contains [London, Denver, Paris, Miami]
            cityList.add("Seoul");
 // contains [London, Denver, Paris, Miami, Seoul]
            cityList.add("Tokyo");
 // contains [London, Denver, Paris, Miami, Seoul, Tokyo]

```

```
System.out.println("List size? "+ cityList.size());
System.out.println("Is Miami in the list? " +cityList.contains("Miami") );
System.out.println("The location of Denver in the list?"  + cityList.indexOf("Denver"));
System.out.println("The location of Uk in the list?"  + cityList.indexOf("Uk"));
System.out.println("Is the list empty? " +cityList.isEmpty() ); // Print false

// Insert a new city at index 2
cityList.add(2, "Xian");
// contains [London, Denver, Xian, Paris, Miami, Seoul, Tokyo]

// Remove a city from the list
cityList.remove("Miami");
// contains [London, Denver, Xian, Paris, Seoul, Tokyo]
```

```
// Remove a city at index 1
        cityList.remove(1);
// contains [London, Xian, Paris, Seoul, Tokyo]

// Display the contents in the list
        System.out.println(cityList.toString());        /*for(String name : cityList)
                                                          {  System.out.println(name);
                                                           }*/
// Display the contents in the list in reverse order
        for (int i = cityList.size() - 1; i >= 0; i--)
        System.out.print(cityList.get(i) + " ");
        System.out.println();

// Create a list to store two box
        java.util.ArrayList list = new java.util.ArrayList();

// Add two box
        list.add(new Box(2));
        list.add(new Box(3));

// Display the area of the first box in the list
        System.out.println("The Volume of the Box? " + ((Box)list.get(0)).volume());
 }
}
```

Thank You