OOPS Concept

In this Lecture we Learn

- OOP Basic Concepts
- Arrays in Java

Object-oriented Programming (OOP)

introduction

- Java is a true OO language and therefore the underlying structure of all Java programs is classes.
- Anything we wish to represent in Java must be encapsulated in a class that defines the “state” and “behaviour” of the basic program components known as objects.
- Classes create objects and objects use methods to communicate between them. They provide a convenient method for packaging a group of logically related data items and functions that work on them.
- A class essentially serves as a template for an object and behaves like a basic data type “int”. It is therefore important to understand how the fields and methods are defined in a class and how they are used to build a Java program that incorporates the basic OO concepts such as encapsulation, inheritance, and polymorphism.

OOP

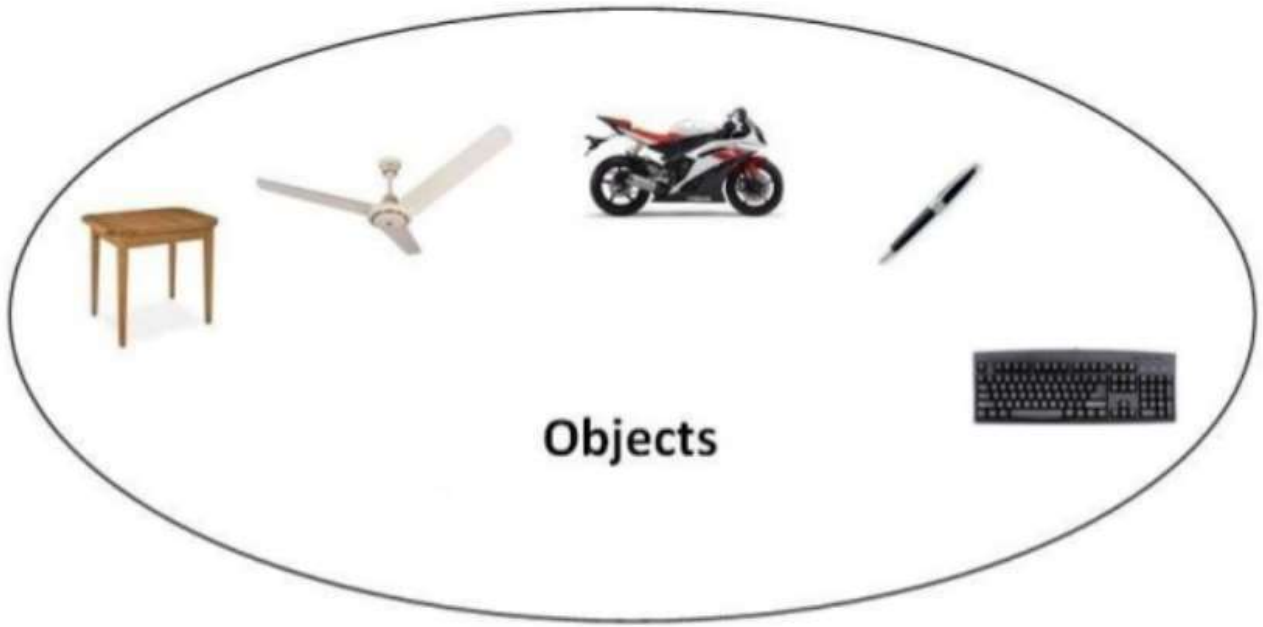
- Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts –
  - Classes
  - Objects
  - Instance
  - Method
  - Message Parsing
  - Polymorphism
  - Inheritance
  - Encapsulation
  - Abstraction

Object in Java

- An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of intangible object is banking system.
- An object has three characteristics:
  - **state:** represents data (value) of an object.
  - **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
  - **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

Object in Java

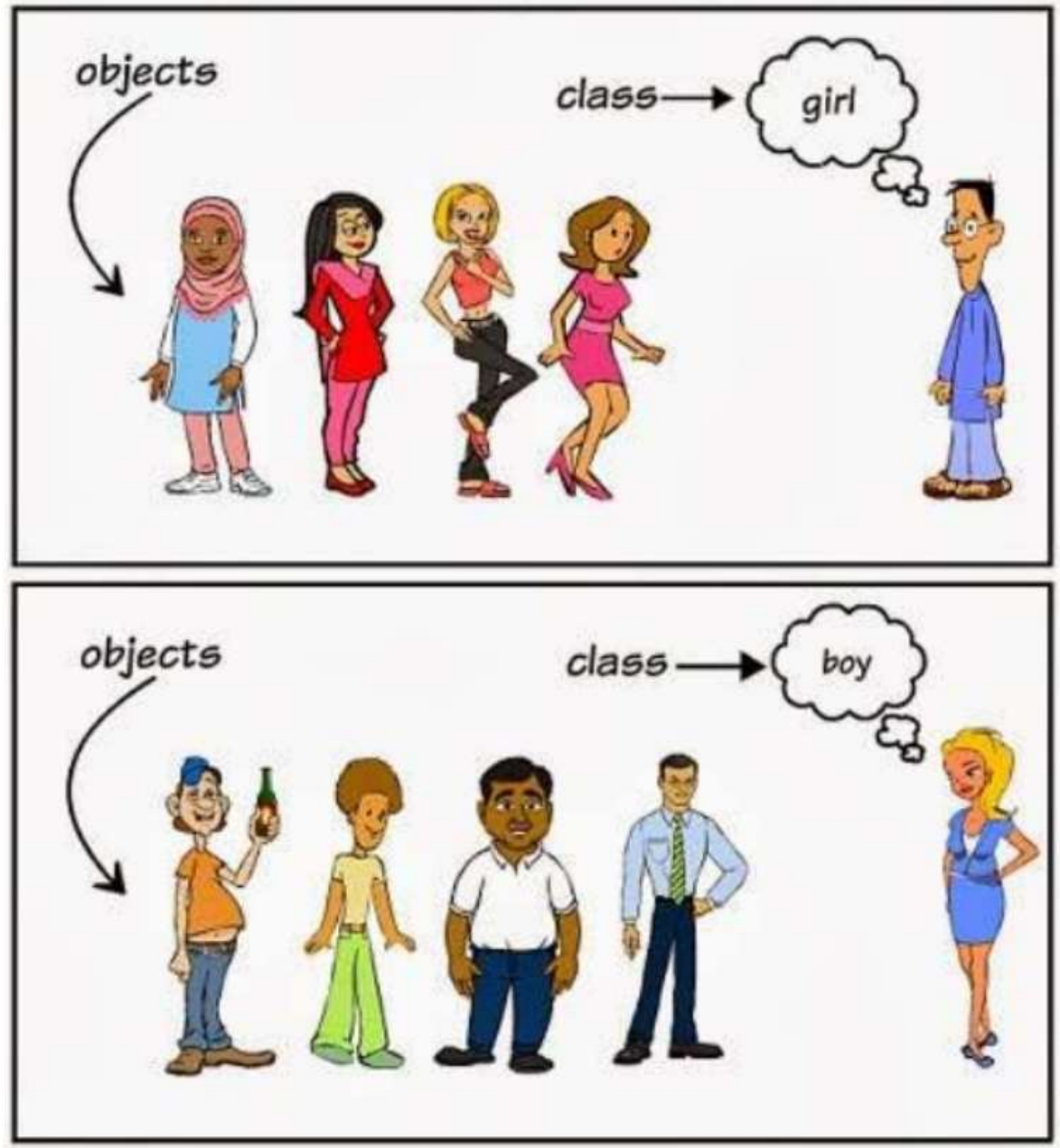
- For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.
- **Object is an instance of a class.** Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.



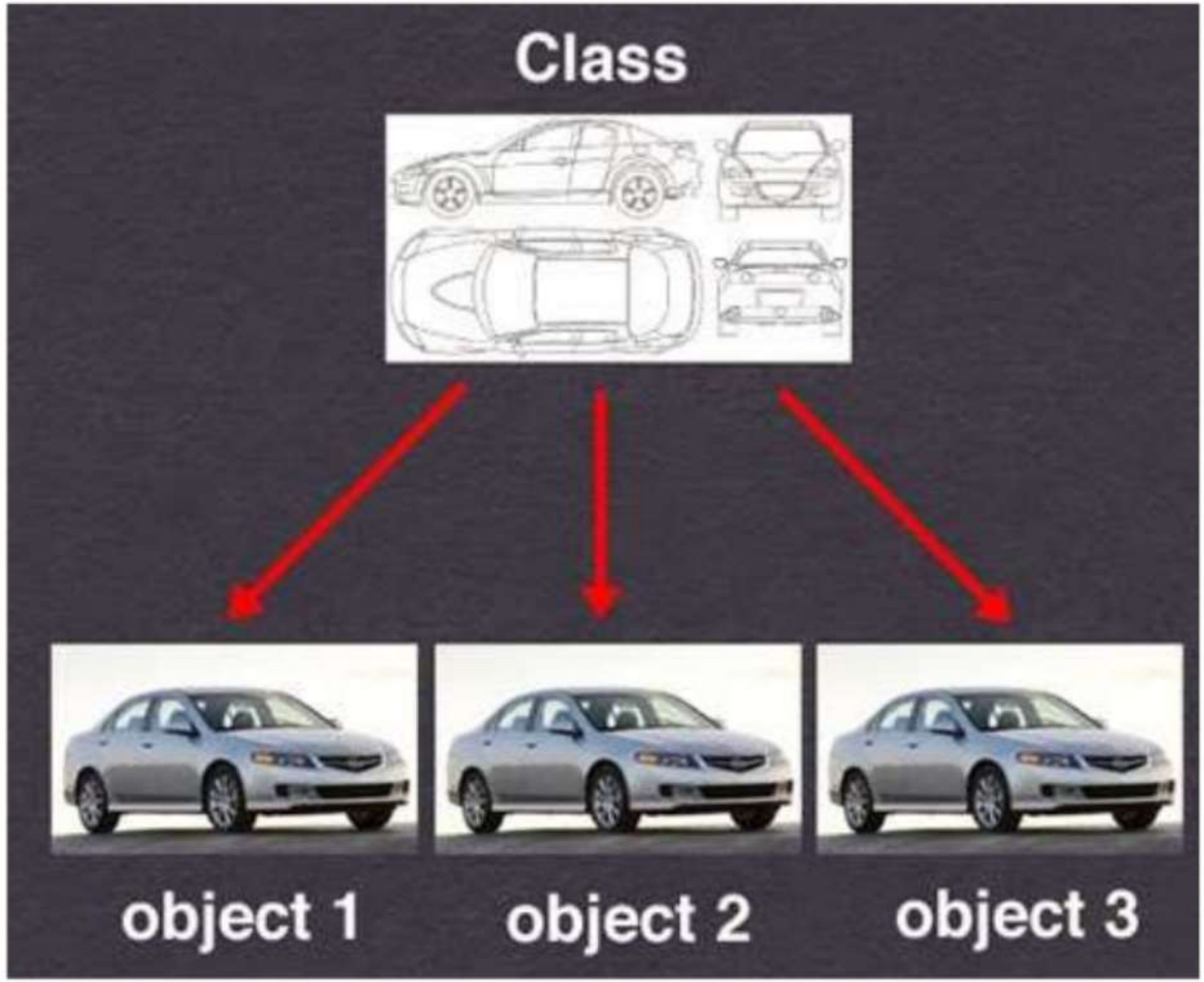
Class in Java

- A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.
- A class in java can contain:
  - **data member**
  - **method**
  - **constructor**
  - **block**
  - **class and interface**

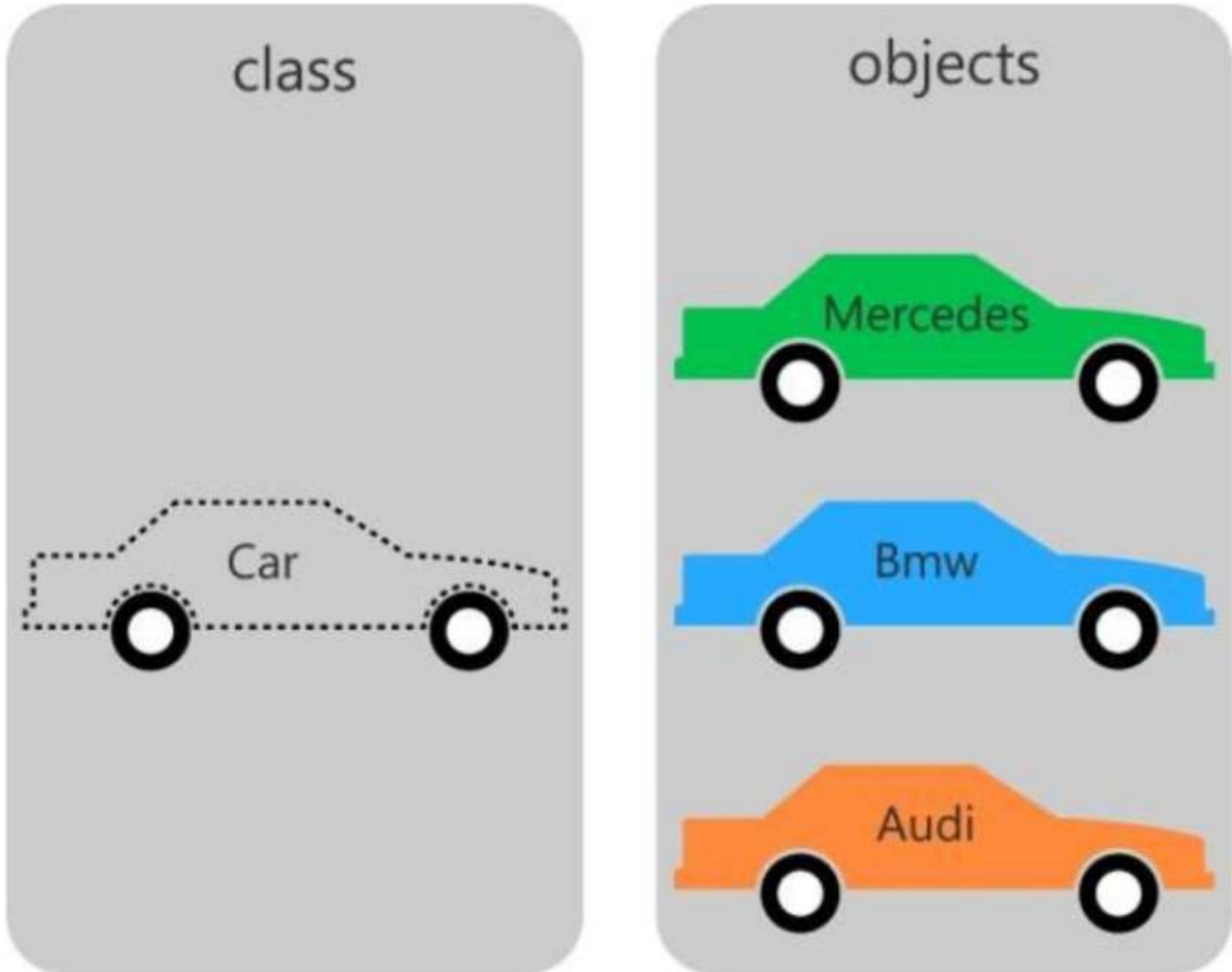
Object and Class in Java



Object and Class in Java

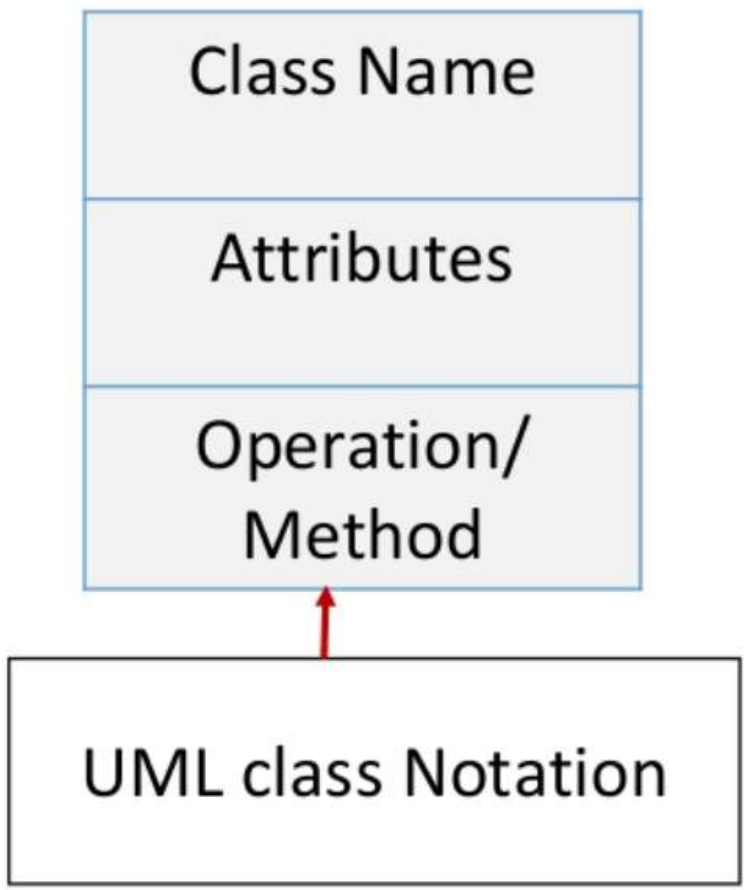


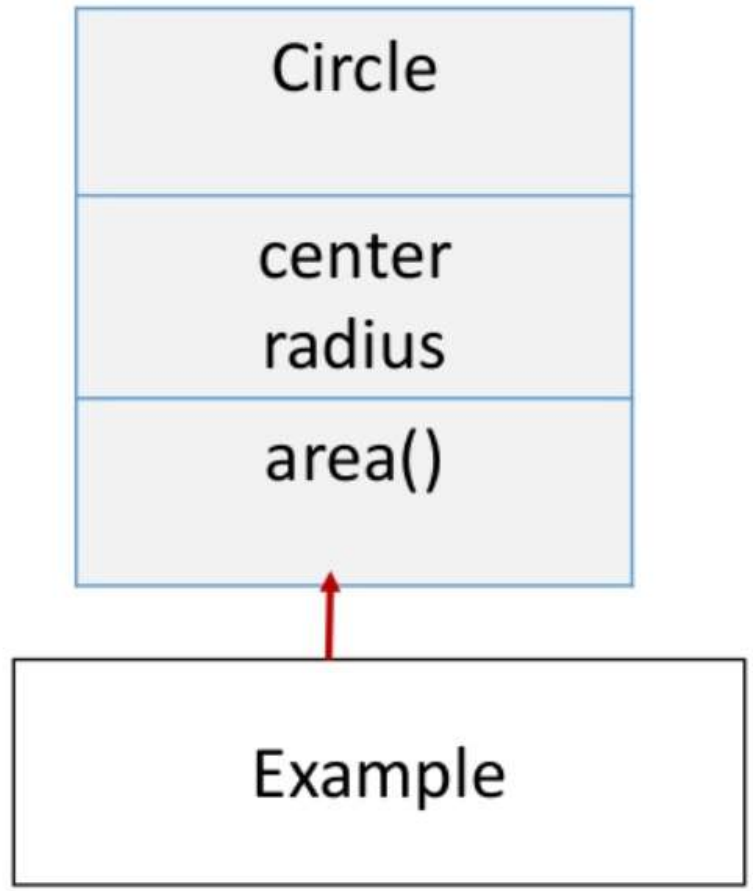
Object and Class in Java



Class in Java

- A *class* is a collection of *fields* (data) and *methods* (procedure or function) that operate on that data.





Class in Java

- A class is a user defined data type.
- The instance of the class are called **objects**.
- Class is a place where we can define the properties and functionalities of the objects.

Class Declaration in Java

- Declaration of class must start with the **keyword *class*** followed by the class name and class members are declared within braces.
- **Syntax of declaring class**

```
access specifier class class_name
  {
  // some data/some fields
  // some functions/methods
  }
```

Syntax of Declaring Class

```
access specifier class classname {
    type instance-variable1;
     // ...
    type instance-variableN;
    type methodname1(parameter-list) {
      // body of method
    }
       // ...
    type methodnameN(parameter-list) {
      // body of method
    }
}
```

Class Explanation Of Syntax

- **Class Name**

```
access specifier class classname
{
}
```

- **Access Specifier : Access** modifiers (or **access specifiers**) are keywords in object-oriented languages that set the accessibility of classes, methods, and other members.
- **class** is Keyword in Java used to create class in java.
- classname is Name of the User defined Class.

Class Explanation Of Syntax

Class Instance Variable

```
type instance-variable1;
type instance-variable2;
// ...
type instance-variableN;
```

- Instance Variables are Class Variables of the class.
- When a number of objects are created for the same class, **the same copy of instance variable is provided to all**.
- **Instance variables have different value** for different objects.
- Access Specifiers can be applied to instance variable i.e public, private.
- Instance Variable are also called as "**Fields**"

Class Explanation Of Syntax

Class Method

```
type methodname1(parameter-list) {
    // body of method
  }
```

- Above syntax is of **Class Methods**.
- These methods are equivalent to function in C Programming Language.
- Class methods **can be declared public or private**.
- These methods are meant for operating on class data i.e **Class Instance Variables**.
- Methods have **return type as well as parameter list**. We are going to

Example : Class Concept

```
public class Rectangle {
    // two fields
    public int breadth;
    public int length;

    // two methods
    public void setLength(int newValue) {
        length = newValue;
    }
    public void setBreadth(int newValue) {
        breadth = newValue;
    }
}
```

Example : Class Concept

```
public class Rectangle {
    // two fields
    public int breadth;
    public int length;

    // two methods
    public void setLength(int newValue) {
        length = newValue;
    }
    public void setBreadth(int newValue) {
        breadth = newValue;
    }
}
```

Class Name

Access Specifier

Method Parameter

Method return type

Access Specifiers in Java

- There can be three access specifiers in Java :
  - **Private** : This access specifier makes sure that the private members can be accessed from other members of the class only or from the friends of class (this is an advanced concept, will learn it later).
  - **Protected** : This access specifier makes sure that the protected members can be accessed from members of the same class ,from the friend classes and also from members of the derived classes.
  - **Public** : This access specifier makes sure that the public members can be accessed from anywhere through the object of the class.

Access Specifiers in Java

- Private, Protected, Public are also **called visibility labels**.
- The member that declare **private** can be accessed only from within class.
- **Public** member can be accessed anywhere through the object of the class.
- In Java data can be hidden by making private.

Constructors in Java

- When discussing about classes, one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

- Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Constructors in Java

- Constructor is a class member function with same name as the class.
- The main job of constructor is to allocate memory for class objects.
- Constructor must be in pubic area of the class
- Constructor is automatically called when object is created.

Constructors Example

```
public class Test {

   public Test() {
   //default constructor (without parameter)
   }

   public Test(String name) {
      // This constructor has one parameter, name.
   }
}
```

Constructor name and class are same

Different Ways To Create An Object In Java?

- There are many ways to create an object in java. They are:
  - By new keyword
  - By newInstance() method
  - By clone() method
  - By factory method etc.
- We will learn, these ways to create the object later.

Anonymous Object

- Anonymous simply means nameless. An object that have no reference is known as anonymous object.
- If you have to use an object only once, anonymous object is a good approach.
- The anonymous object is created and dies instantaneously. But, still with anonymous objects work can be extracted before it dies like calling a method using the anonymous object:
- Anonymous object instantiation:

```
new Test()
```

Anonymous Object Example

```
class Test{
void fact(int  n){
 int fact=1;
 for(int i=1;i<=n;i++){
  fact=fact*i;
 }
  System.out.println("factorial is "+fact);
}
public static void main(String[] args) {
   //calling method with anonymous object
   new Test().fact(5);
 }
}
```

Creating an Object

- As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.
- There are three steps when creating an object from a class −
  - **Declaration** − A variable declaration with a variable name with an object type.
  - **Instantiation** − The 'new' keyword is used to create the object.
  - **Initialization** − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Creating an Object Using new keyword

```
Test mytest = new Test();
```

- Actual Creation of Object.
- **Memory is allocated for an object** after executing this statement.
- This Statement will create instance of the class "**Test**" and name of instance is nothing but actual object "**mytest**".

Create an Object Using new keyword Example

```
public class Test{
    // This constructor has one parameter, name.
    public Test(String name) {
    System.out.println("Passed Name is : " + name );
   }
    public static void main(String[] args) {
   // Following statement would create an object myName
     Test myName = new Test("Adil");
  }
}
```

Output is:
Passed Name is : Adil

The **java.lang.Class.newInstance()** creates a new instance of the class represented by this Class object. The class is instantiated as if by a new expression with an empty argument list. The class is initialized if it has not already been initialized. .

```
import java.util.*;
import java.lang.*;

public class ClassDemo {

  public static void main(String[] args) {
    try {
      // date object
      Date d = new Date();
      Class cls = d.getClass();
      System.out.println("Time = " + d.toString());

      /* creates a new instance of the class represented by this
        Class object cls */
      Object obj = cls.newInstance();
      System.out.println("Time = " + obj);
    } catch(InstantiationException e) {
      System.out.println(e.toString());
    } catch(IllegalAccessException e) {
      System.out.println(e.toString());
    }
  }
}
```

Java Object clone()
The Java Object clone() method creates a shallow copy of the object.

Here, the shallow copy means it creates a new object and copies all the fields and methods associated with the object.

The syntax of the `clone()` method is:

```
object.clone()
```

clone() Return Values

- returns the copy of the object
- throws `CloneNotSupportedException` if the object's class does not implement the `Cloneable` interface

> **Note:** The `Object` class does not implement `Cloneable`. Hence, we cannot call the `clone()` method for the object of the `Object` class.

```
class Main implements Cloneable {
  // declare variables
  String name;
  int version;
  public static void main(String[] args) {

    // create an object of Main class
    Main obj1 = new Main();

    // initialize name and version using obj1
    obj1.name = "Java";
    obj1.version = 14;

    // print variable
    System.out.println(obj1.name);       // Java
    System.out.println(obj1.version);    // 14
    try {
      // create clone of obj1
      Main obj2 = (Main)obj1.clone();

      // print the variables using obj2
      System.out.println(obj2.name);      // Java
      System.out.println(obj2.version);   // 14
    }
    catch (Exception e) {
      System.out.println(e);
    }
  }
}
```

Variable Types

- Local variables
- Instance variables
- Class/Static variables
- We lean about variable type in detail in lecture No. 10 so stay with me

Variable Types

- **Local Variables**
  - Local variables are declared in methods, constructors, or blocks.
  - Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
  - Access modifiers cannot be used for local variables.
  - Local variables are visible only within the declared method, constructor, or block.
  - Local variables are implemented at stack level internally.
  - There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Local Variables Example

```
public class Test{
  public void age() {
    int age = 0 ;
    age = age + 7;
    System.out.println("Age is : " + age);
 }
public static void main(String[] args) {
   Test test = new Test();
   test.age();
  }
}
```

Here, *age* is a local variable. This is defined inside *age()* method and its scope is limited to only this method.

Output is:

Age is : 7

Inheritance in Java

- **Inheritance in java** is a mechanism in which one object acquires all the properties and behaviors of parent object.
- The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.
- Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.
- Why use inheritance in java
  - For Method Overriding (so runtime polymorphism can be achieved).
  - For Code Reusability.

Polymorphism in Java

- **Polymorphism in java** is a concept by which we can perform a *single action by different ways*. Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
- There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.
- If you overload static method in java, it is the example of compile time polymorphism.

Arrays in Java

Introduction

- Consider the following declarations:

```
int a, b, c, d, e, f, g, h, i, j;
```

Issues:

1. Variables of the same **data**_type, i.e. ***int***.

2. Individual operation (e.g., input) is needed for each variable

Introduction

Better Solution:

- Use Loops. (One variable will be enough for the 10 input values).

Issue:

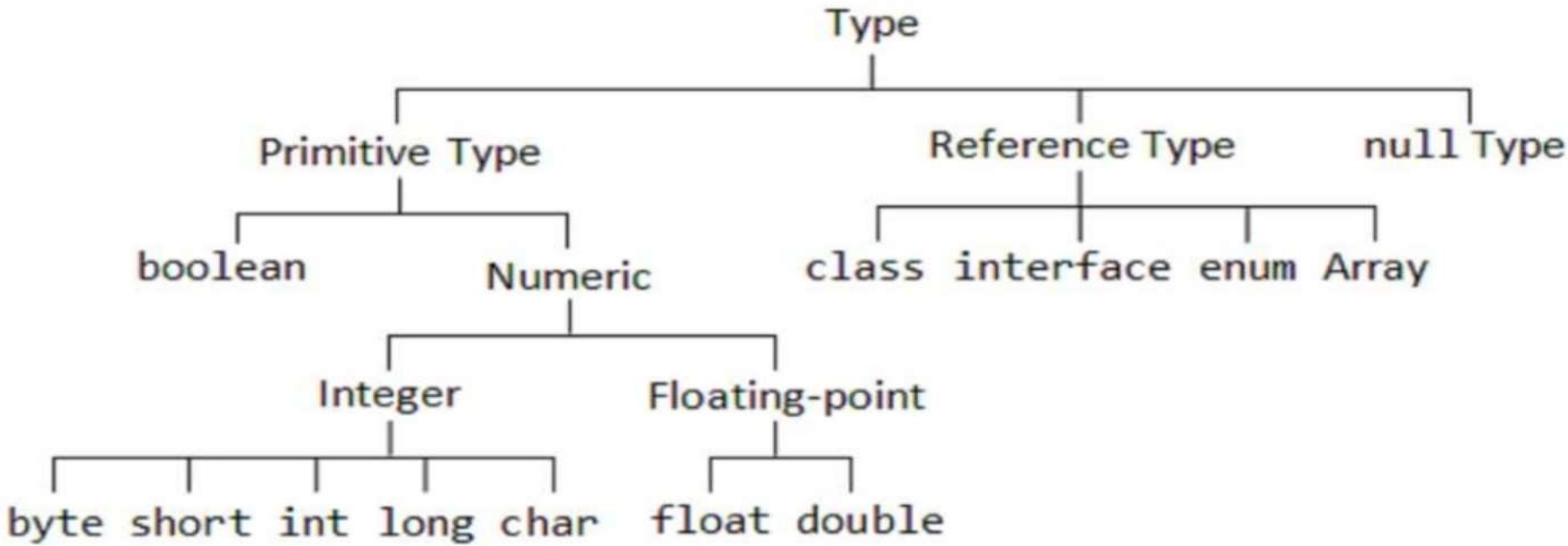
- Impossible to access or recall let say the 5th or even the 9th value after executing the needed operation.

Introduction

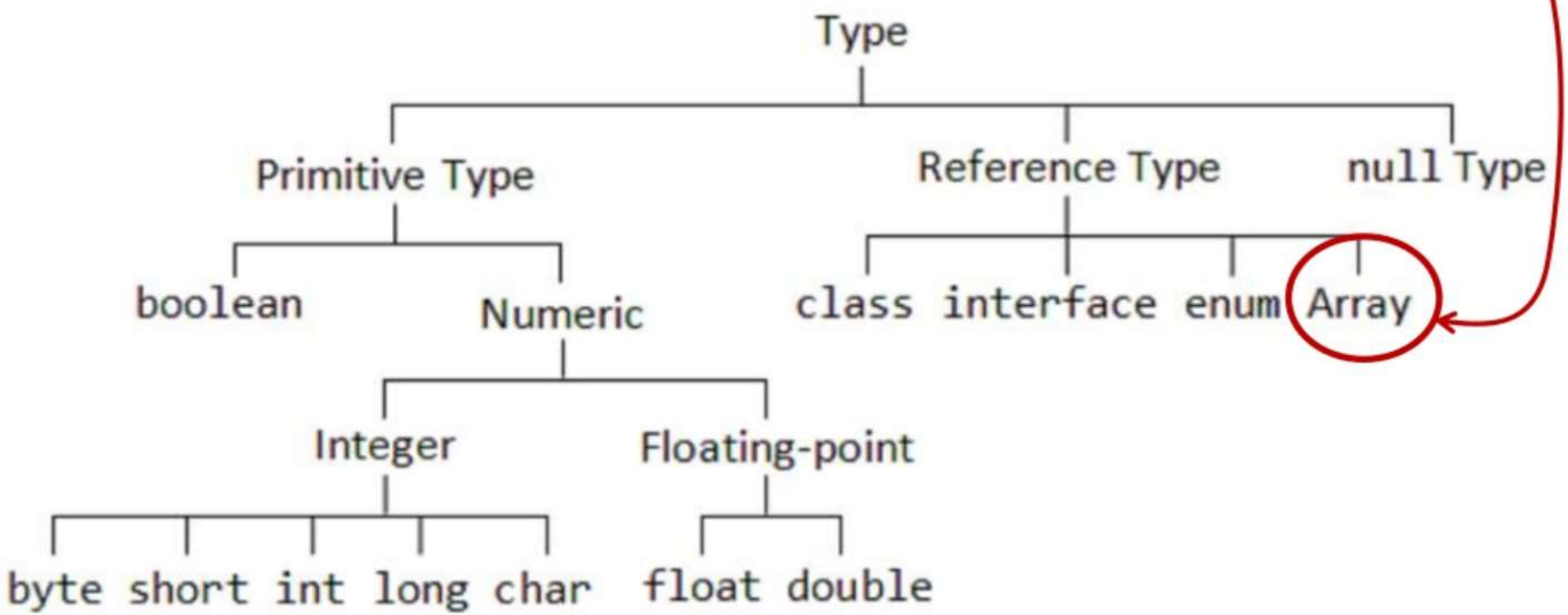
Best Solution:

- Declare an array variable that will allow us to store values in contiguous memory location, perform some operations, and still have the control over previously accessed values.

- This principle is what we call an ***array***.


Type
Primitive Type
Reference Type
null Type
boolean
Numeric
class
interface
enum
Array
Integer
Floating-point
byte
short
int
long
char
float
double

Array is a Reference type



Java Array

- Java provides a data structure, the **array**, which stores a **fixed-size sequential** collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a **collection of variables** of the same type.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

Java Array

- Arrays are **data structures** consisting of **related data** items of the **same type**.
- An array is a **consecutive** group of **memory locations**
- The memory locations in the array are known as **elements** of array.
- **Total number** of elements in the array is call the **length** of the array.

Java Array

- To refer to **a particular location** or element in the array, we **specify the name** of the array and the **position number** of the particular element in the array in square brackets ([]).
- The **position number** is more formally called a subscript or **index**.
- A **subscript** must be an **integer** or integer expression.

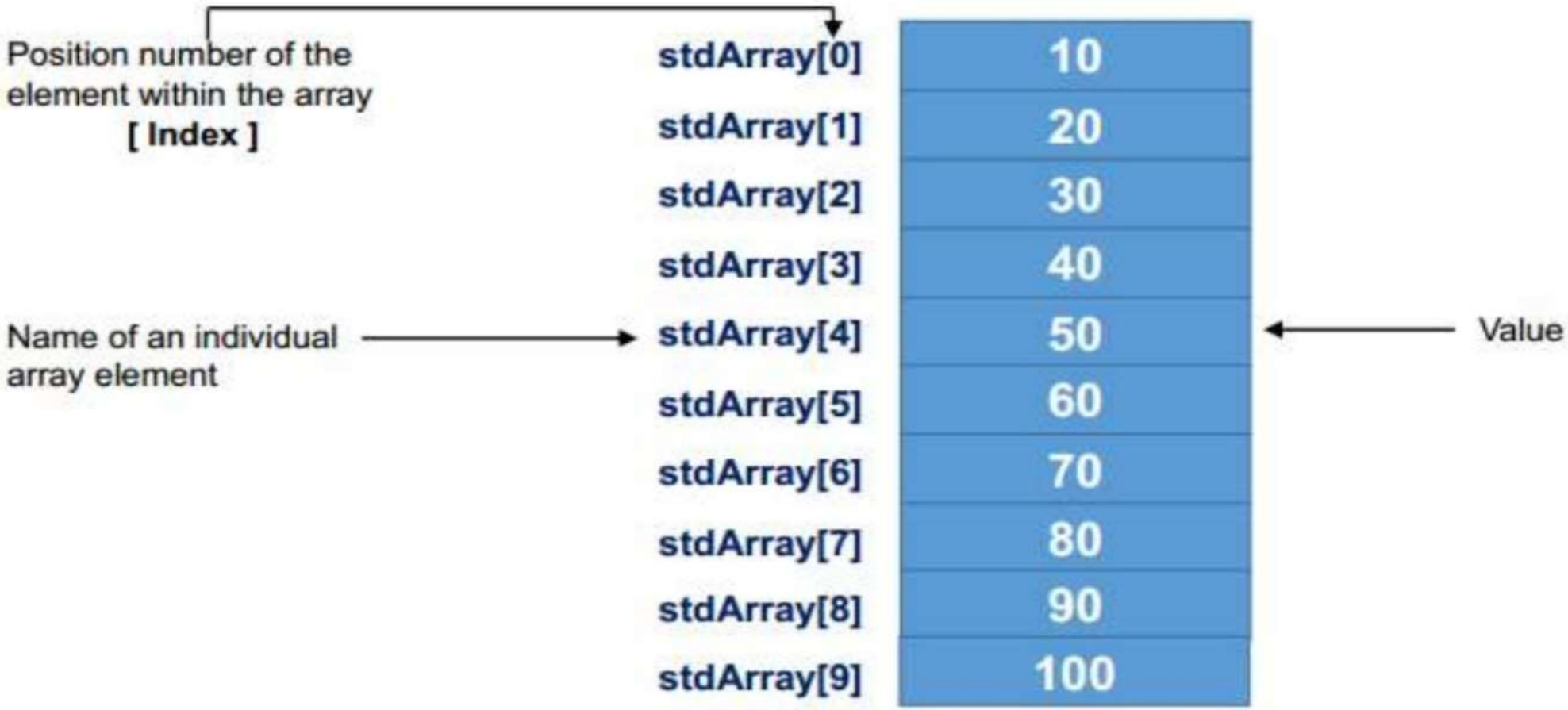
Java Array

- The rules of **variable naming** convention apply to **array names**
- In Java each array has:
  - Data type
  - Name
  - Size
- The arrays occupy the memory depending upon their size

Java Array

- The **first element** has subscript 0 (zero) and is sometimes called the **zeroth element**
- The highest subscript in array is 1 less than the number of elements in the array

Java Array


Position number of the element within the array
[Index]
Name of an individual array element
stdArray[0]
stdArray[1]
stdArray[2]
stdArray[3]
stdArray[4]
stdArray[5]
stdArray[6]
stdArray[7]
stdArray[8]
stdArray[9]
10
20
30
40
50
60
70
80
90
100
Value

Advantages of Arrays

- Array can **store a large** number of **values** with **single name**.
- The **values** store in the array can **sorted easily**.
- The **search process** can be **applied** on array **easily**.
- It can be used to **implement** other **data structures** like linked lists, stacks, queues, trees, graphs etc.
- **2D arrays** are used to represent **matrices**

Type of Arrays

- There are two types of array.
  - Single Dimensional Array
  - Multidimensional Array

Single Dimensional Array in java

- **Syntax to Declare an Array in java**

```
dataType[] identifier; (or)     // preferred way.
dataType [] identifier; (or) // works but not preferred way.
dataType identifier [];      // works but not preferred way.
```

Single Dimensional Array in java

- **Examples**

```
int[] arr;          //Array of Integer type
char[] arr;     //Array of Character
short[] arr;  //Array of Short type
long[] arr;  //Array of long type
```

Creating Arrays

- **new** operator is used to initialize an array.
- **Syntax**

```
arrayRefVar = 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 arrayRefVar.

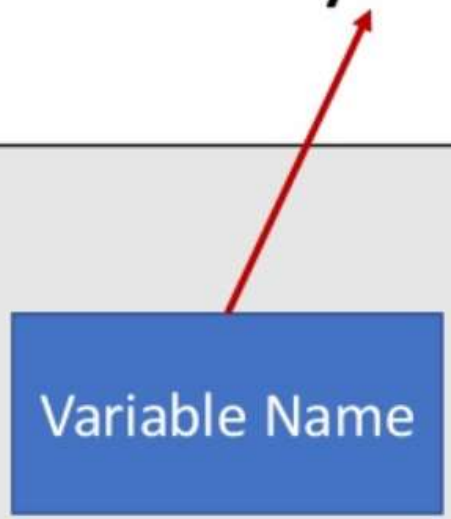
Creating Arrays

- Unlike declarations for primitive data type variables, the declaration of an array variable does not allocate any space in memory for the array.
- Only a storage location for the reference to an array is created. If a variable does not reference to an array, the value of the variable is null.
- We cannot assign elements to an array unless it has already been created. After an array variable is declared, you can create an array by using the **new** operator.

Creating Arrays

- **new** operator is used to initialize an array.
- Syntax

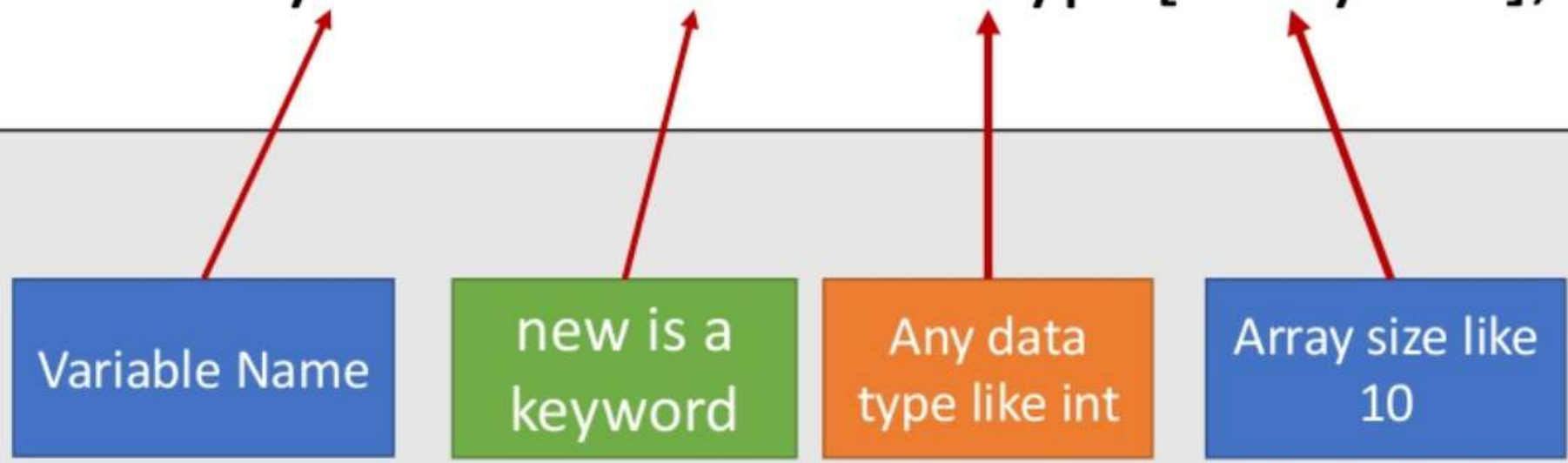
```
arrayRefVar = new dataType[arraySize];
```



Creating Arrays

- **new** operator is used to initialize an array.
- Syntax

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



Creating Arrays

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

- Alternatively you can create arrays as follows

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

Creating Arrays

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

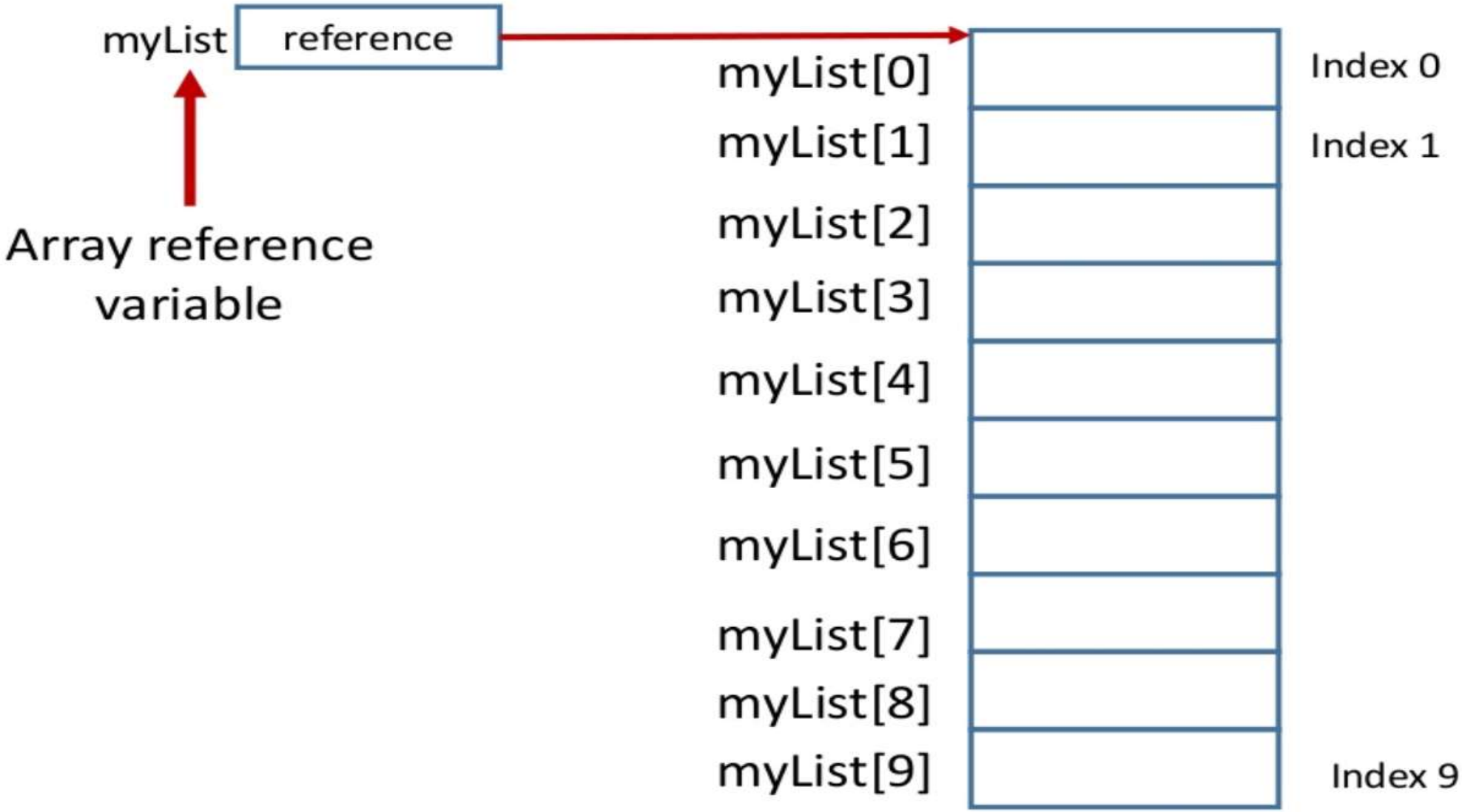
```
double[] myList = new double[10];
```

- The array elements are accessed through the **index**. Array indices are 0-based; that is, they start from 0 to **arrayRefVar.length-1**.

Creating Arrays

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

- The following creates an array of myList for a class with 10 elements. And all elements in the array are initialized to zero automatically.


myList
reference
Array reference variable
myList[0]
myList[1]
myList[2]
myList[3]
myList[4]
myList[5]
myList[6]
myList[7]
myList[8]
myList[9]
Index 0
Index 1
Index 9

Initialization of Array

- Assigning the values into Array, This process is known as initialization.
- This is done using the Array subscripts as show below:

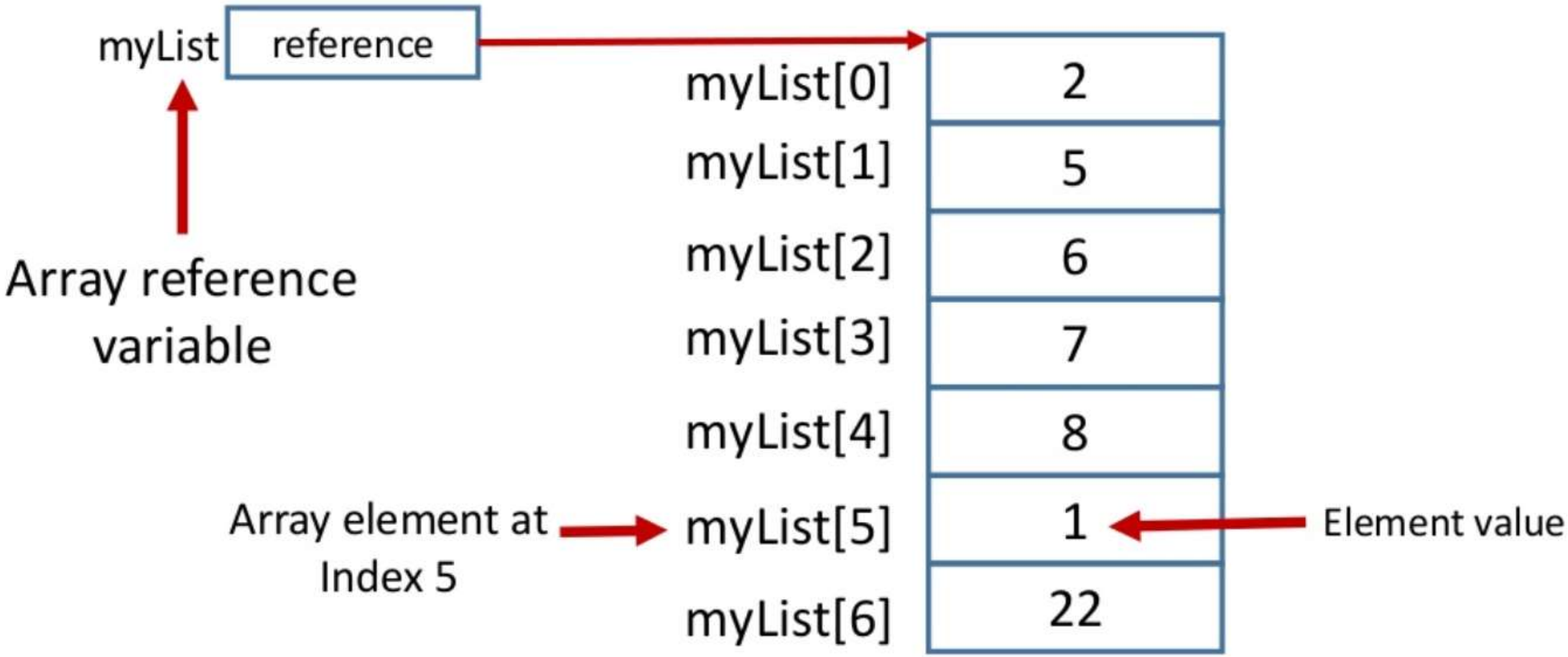
```
arrayname[subscript] = value;
```

- **Example:**

```
Number[0]=31;
Number[1]=34;
.....
Number[5]=20;
```

Initialization of Array

```
double[] myList = new double[]{2,5,6,7,8,1,22};
```



Declaration, Instantiation and Initialization of Java Array

- Syntax for default values:

```
int[] num = new int[5];
Or (less preferred)
int num[] = new int[5];
```

- Syntax with values initialization:

```
int[] num = {1,2,3,4,5};
Or (less preferred)
int num[] = {1, 2, 3, 4, 5};
```

Point to be Note

- We can assign an array a null value but we can't create an empty array by using a blank index.

```
int[] array = null;    // legal
int[] array = new int[];    // illegal initialization
```

Example of Single Dimensional java Array

```
13      public class Test {
14          public static void main(String[] args) {
15              int array[] = new int[5];
16               array[2] = 25;
17               System.out.println("Third Element = " +array[2]);
                 System.out.println("Fourth Element = " +array[3]);
19          }
20      }
```

javaapplication5.Test > main >

Output - JavaApplication5 (run)

```
run:
Third Element = 25
Fourth Element = 0
```

Example of Single Dimensional java Array

```
public class Test {
   public static void main(String[] args) {
      int array[] = new int[5];
       array[2] = 25;
       System.out.println("Third Element = " +array[2]);
       System.out.println("Fourth Element = " +array[3]);
   }
}
```

We not Assigned any value to Fourth Element of the array so the Original Default value of 0 is Printed

Output is:

Third Element = 25

Fourth Element = 0

Array Length in Java

- The length of the array can be obtained using the length variable in array. This variable will be useful when iterating through the values of an array.

```
int[] array = new int[12];
System.out.println("Array length is " + array.length);
```

This Information will be Useful in the Manipulation of Array when their sizes are not known.

Output is:
Array length is 12

Example of Single Dimensional ava Array

```
13    public class Test {
14        public static void main(String[] args) {
      int a[]=new int[]{10,12,13,14};  //declaration and instantiation
16    //printing array
17    for(int i=0;i<a.length;i++)//length is the property of array
18    System.out.println(a[i]);
19
20        }
21    }
```

javaapplication5.Test > main > a

Output - JavaApplication5 (run)

```
run:
10
12
13
14
```

Example of Single Dimensional ava Array

```
public class Test {
    public static void main(String[] args) {
        int a[]=new int[]{10,12,13,14};
        for(int i=0;i<a.length;i++)
        System.out.println(a[i]);
    }
}
```

Both Generate
Same Output

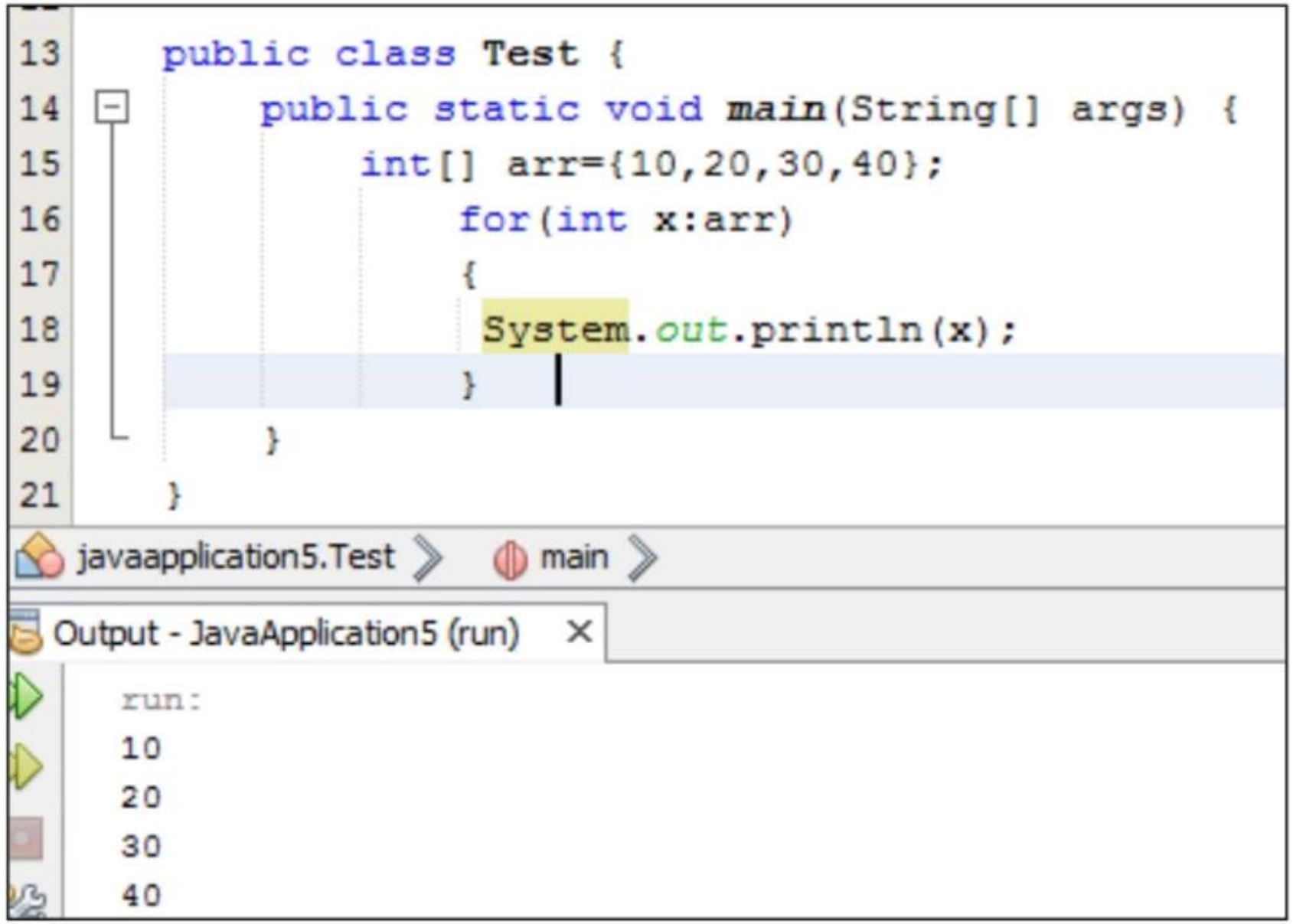
```
System.out.println(a[0])
System.out.println(a[1])
System.out.println(a[2])
System.out.println(a[3])
```

For-Each Loop/Enhanced For Loop

- The for-each loop introduced in Java5. It is mainly used to traverse array or collection elements. The advantage of for-each loop is that it eliminates the possibility of bugs and makes the code more readable.
- **Advantage of For-Each Loop:**
  - It makes the code more readable.
  - It eliminates the possibility of programming errors.
- **Syntax of For-Each Loop:**

```
for(data_type variable : array | collection){}
```

Example of Foreach Loop



The Enhanced For Loop Versus the Traditional For Loop

- When you need to access the values stored in an array, from the first element to the last element, the enhanced for loop is simpler to use than the traditional for loop.

- With the enhanced for loop we do not have to be concerned about the size of the array, and we do not have to create an “index” variable to hold subscripts.

The Enhanced for Loop Versus the Traditional for Loop

- Circumstances in which the enhanced for loop is not adequate. You cannot use the enhanced for loop if:
  - We need to change the contents of an array element
  - you need to work through the array elements in reverse order
  - We need to access some of the array elements, but not all of them
  - We need to simultaneously work with two or more arrays within the loop
  - you need to refer to the subscript number of a particular element
- In any of these circumstances, you should use the traditional for loop to process the array.

Taking input from User Using Array

```
12  import java.util.Scanner;
13  public class Test {
14      public static void main(String[] args) {
15      Scanner input = new Scanner(System.in);
16      double[] numbers = new double[3];
17      System.out.println("Please Enter Three Numbers: ");
18
19      for (int i = 0; i < numbers.length; i++){
20          numbers[i] = input.nextDouble();
21          }
22       System.out.println("You Enter the Following Numbers: ");
23      for(double i: numbers){       //for printing array
24          System.out.println(i);
25          }
26      }
27  }
```

test.Test > main >

Output - Test (run) × | Notifications ×

```
run:
Please Enter Three Numbers:
22
33
44
You Enter the Following Numbers:
22.0
33.0
44.0
```

Explanation of Previous Code

```
import java.util.Scanner;
public class Test {
   public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
   double[] numbers = new double[3];
   System.out.println("Please Enter Three Numbers: ");
   for (int i = 0; i < numbers.length; i++){
      numbers[i] = input.nextDouble();
      }
    System.out.println("You Enter the Following Numbers: ");
   for(double i: numbers){     //for printing array (for each loop)
      System.out.println(i);
      }
   }
}
```

Creating an Array of size 3

Problem Statement

- Write a Java program which calculates the sum of squares of 3 numbers and stored in an array.
- Note Talking input of three numbers from user.

Solution of the Previous Program

```
8  import java.util.Scanner;
9  public class Test {
10     public static void main(String[] args) {
11     Scanner input = new Scanner(System.in);
12     int sum=0;
13     int [] numbers = new int [3];
14     System.out.println("Please Enter Three Numbers: ");
15     for (int i = 0; i < numbers.length; i++){
16         numbers[i] = input.nextInt();
17         }
18     for(int i: numbers){       //for printing array
19         sum= sum+i * i;
20         System.out.println("Square of "+i+" is "+(i*i));
21         }
22     System.out.println("Sum of Square is: "+sum);
23     }
24 }
```

test.Test > main >

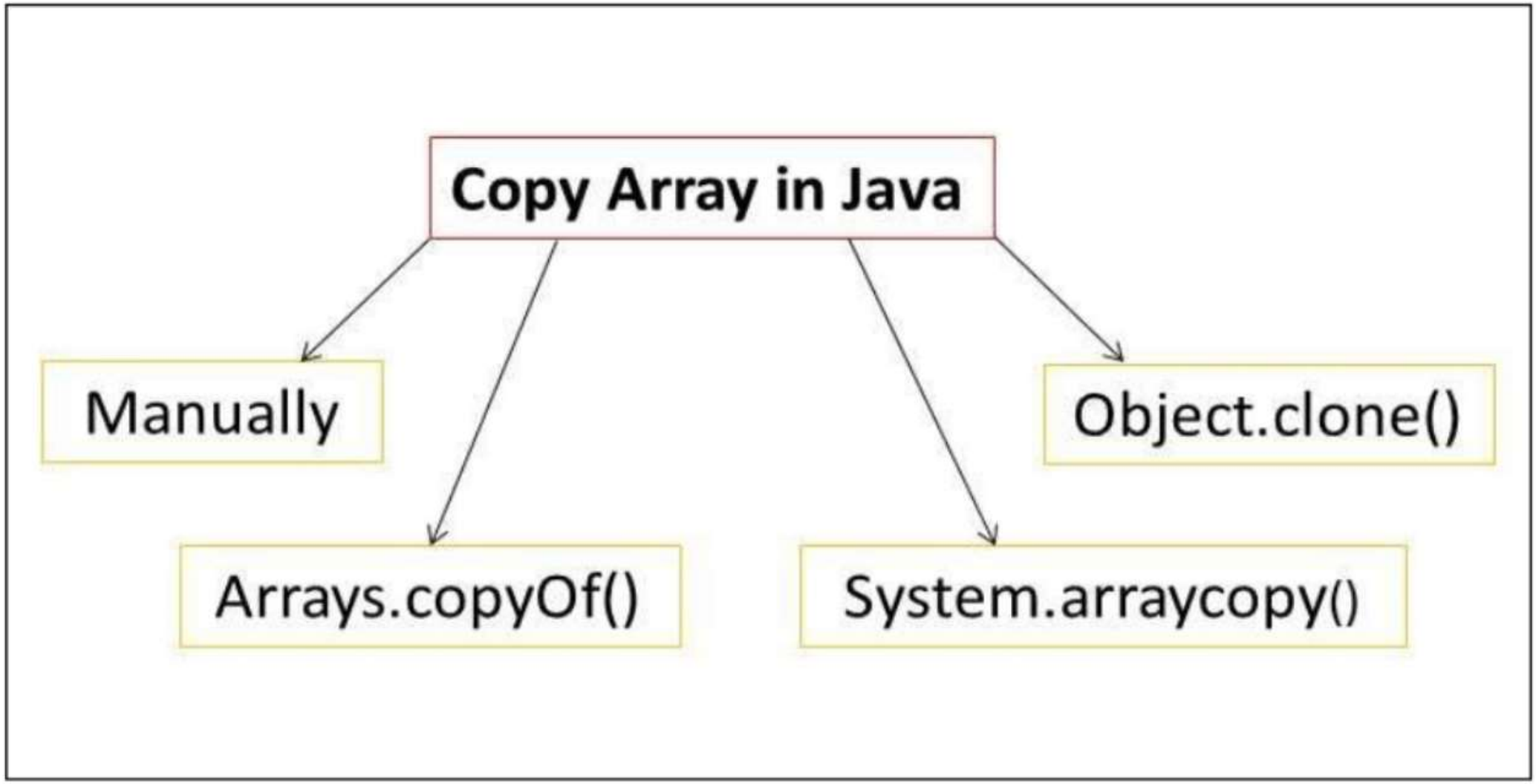
Output - Test (run) × | Notifications ×

```
run:
Please Enter Three Numbers:
2
3
4
Square of 2 is 4
Square of 3 is 9
Square of 4 is 16
Sum of Square is: 29
```

How to Copy One Array to Another in Java

- There are four ways to copy arrays
  - using a loop structure
  - using Arrays.copyOf()
  - using System.arraycopy()
  - using clone()

How to Copy One Array to Another in Java



Copying Arrays (using a loop structure**)**

- One of the most common things you can do with an array is to copy an array. We're going to copy an entire an array, and to copy part of an array.

```
int[] firstArray = new int[5];
int[] secondArray = new int[5];
```

Copying Arrays

- For copying data from one array to another both array need to be of:
  - Same data type
  - Same size

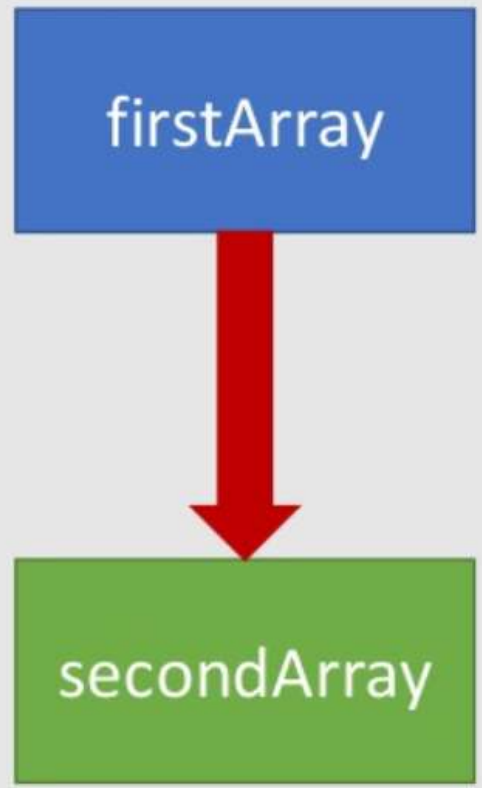
```
int[] firstArray = new int[5];
int[] secondArray = new int[5];
```

Copying Arrays (using a loop structure)

```
int[] firstArray = new int[5];
int[] secondArray = new int[5];
```

```
secondArray[0] = firstArray[0];
secondArray[1] = firstArray[1];
secondArray[2] = firstArray[2];
secondArray[3] = firstArray[3];
secondArray[4] = firstArray[4];
```

```
secondArray[i] = firstArray[i];
```



Copying Arrays (using a loop structure)

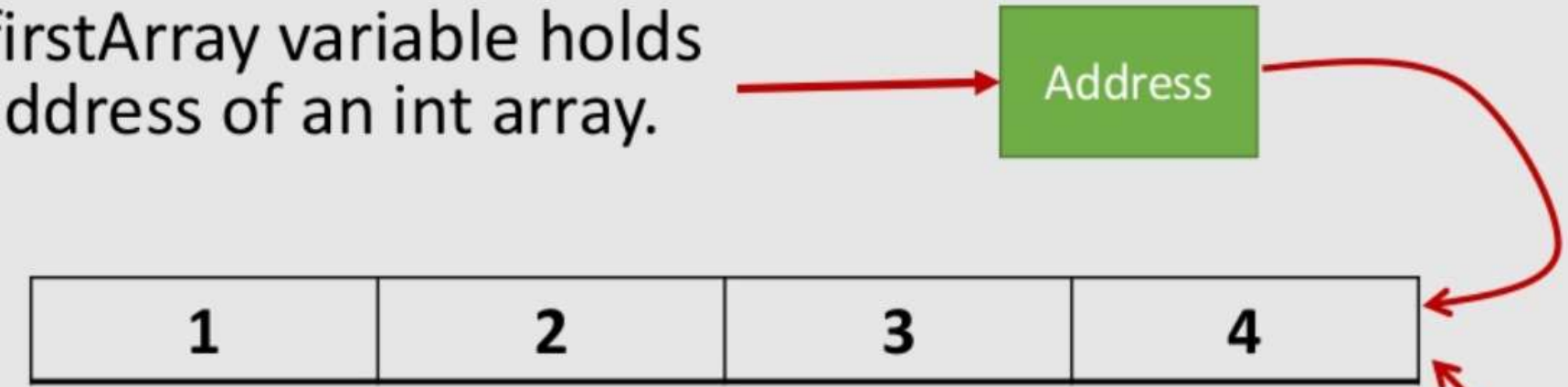
```
public class Test {
    public static void main(String[] args) {
    int firstArray[] = { 1, 2, 3, 4};
        int secondArray[] = new int[firstArray.length];


        for (int i = 0; i < a.length; i++) {
            secondArray[i] = firstArray[i];
        System.out.println(secondArray[i]);
        }
    }
}
```

Copying Arrays (using a loop structure)

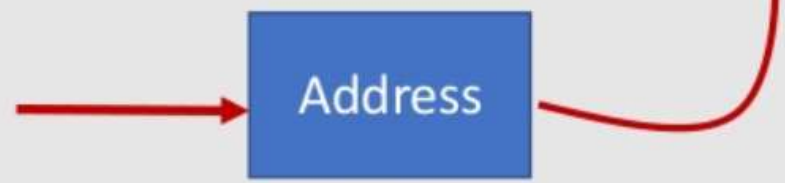
- Both firstArray and secondArray reference the same array

```
secondArray[i] = firstArray[i];
```

The firstArray variable holds the address of an int array.



The secondArray variable holds the address of an int array



Copying Arrays (using Arrays.copyOf()**)**

- **Syntax of Arrays.copyOf() Method**

```
public static int[] copyOf(int[] original,int newLength)
```

- The second parameter specifies the length of the new array, which could either less or equal or bigger than the original length.

Copying Arrays (using Arrays.copyOf())

```
import java.util.Arrays;
public class Test {
   public static void main(String[] args) {
   int a[]={10,20,30};
      //copying one array to another
      int b[]=Arrays.copyOf(a,a.length);

      //printing array
      for(int i=0;i<b.length;++i){
         System.out.println(b[i]+" ");
      }
   }
}
```

Copying Arrays (using System.arraycopy())

- Most efficient copying data between arrays is provided by System.arraycopy() method. The method requires five arguments.
- **Syntax of System.arraycopy() Method**

```
public static void arraycopy(Object source,
                             int srcIndex,
                             Object destination,
                             int destIndex,
                             int length)
```

- The method copies length elements from a source array starting with the index srcIndex to a new array destination at the index destIndex.

Copying Arrays (using System.arraycopy())

```
public static void arraycopy(Object source,
                             int srcIndex,
                             Object destination,
                             int destIndex,
                             int length)
```

- **source** -- This is the source array.
- **srcIndex** -- This is the starting position in the source array.
- **Object destination** -- This is the destination array.
- **destIndex** -- This is the starting position in the destination data.
- **length** -- This is the number of array elements to be copied.

Copying Arrays (using System.arraycopy())

```
public class Test {
   public static void main(String[] args) {
   int[] a = {1, 2, 3};
   int[] b = new int[a.length];
   System.arraycopy(a, 0, b, 0, 3);

      //printing array
      for(int i=0;i<b.length;++i){
         System.out.println(b[i]);
      }
   }
}
```

Copying Arrays (using clone())

```
public class Test {
    public static void main(String[] args) {
       int a[]={10,20,30};
        int b[]=new int[a.length];
        //copying one array to another
        b=a.clone();

        //printing array
        for(int i=0;i<b.length;++i){
            System.out.println(b[i]);
        }
    }
}
```

Output is:

10
20
30

Java Program to Multiply the Elements of an Array

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


     int[] array = {1,2,3};
    int mul = 1;
    //for each loop
    for( int num : array) {
       mul = mul*num;
    }
    System.out.println("Multiplication of array elements is: "+mul);
   }
}
```

Output is:

Multiplication of array elements is: 6

Java Program to Find the Largest Number in an Array

```
import java.util.Scanner;
public class Test {
   public static void main(String[] args) {
   int n, max;
      Scanner s = new Scanner(System.in);
      System.out.print("Enter number of elements in the array: ");
      n = s.nextInt();
      int a[] = new int[n];
      for(int i = 0; i < n; i++)
      {
         System.out.print("Enter element "+(i+1)+": ");
         a[i] = s.nextInt();
      }
```

Java Program to Find the Largest Number in an Array

```
max = a[0];
      for(int i = 0; i < n; i++)
      {
         if(max < a[i])
         {
            max = a[i];


         }
      }
      System.out.println("Maximum value:"+max);
   }

}
```

Your Task-1

- Write a Java program to find the sum and average of one dimensional integer array.

Your Task-2

- Write a Java program that asks the user to type 10 integers of an array. The program must compute and write how many integers are greater than or equal to 10

Passing Arrays to Methods

- Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an **int** array −

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

- You can invoke it by passing an array. For example, the following statement invokes the printArray method to display 3, 1, 2, 6, 4, and 2 −
- Example
- printArray(new int[]{3, 1, 2, 6, 4, 2});

Passing Arrays to Methods

```
12  public class Example {
13      public static void main(String[] args) {
14          int[] a = {1,2,3};
15          add(a);
16          for(int x:a)
17          System.out.println(x);
18      }
19      public static void add(int[] a)
20      {
21          for(int i=0;i<a.length;i++)
22              a[i]=a[i]+4;
23      }
24  }
```

Output - Example (run) | Notifications

```
run:
5
6
7
```

Another Example Passing Arrays to Methods

```
public static void main(String args[]) {
    int arr[] = { 56, 43, 12, 9, 39 };
    int pos = searchElement(arr, 9);   // search element '9'
    if (pos != -1) {
      System.out.println("Element is found at position : " + pos);
    }
    else {
      System.out.println("\nElement not found ... ");
    }
  }
}
```

Another Example Passing Arrays to Methods

```
public class Test {
    static int searchElement(int arr[], int val) {
    int pos = -1;
    for ( int i = 0; i < arr.length; i++ ) {
      if (arr[i] == val) { // compare each array element with 'var'
        pos = i;  // element found
        break;  // break out of loop
      }
    }
    return pos;
  }
```

Passing array to Method

Another Example of Returning an Array from a Method

```
9   public class Test {
10      static void display(int arr[]) {
11      for ( int element : arr ) {
12         System.out.print(element + " ");
13      }
14    }
15    static int[] getNewArray(int arr[]) {
16      int aux_array[] = new int[arr.length];
17      for ( int i = 0; i < arr.length; i++ ) {
18         /* multiply each element by 10 */
19         aux_array[i] = arr[i] * 10;
20      }
21      return aux_array;
22    }
23    public static void main(String args[]) {
24      int arr[] = { 1,2,3,4};
25      int new_arr[] = getNewArray(arr);
26      System.out.print("arr : ");
27      display(arr);
28      System.out.println();
29      System.out.print("new_arr : ");
30      display(new_arr);
31      System.out.println();
32    }
33  }
```

Output - Test (run) | Notifications

```
run:
arr : 1 2 3 4
new_arr : 10 20 30 40
```

Some useful Array Algorithms and Operations

• Comparing Arrays

```
int[] firstArray = { 5, 10, 15, 20, 25 };
int[] secondArray = { 5, 10, 15, 20, 25 };
if (firstArray == secondArray)     // This is a mistake.
System.out.println("The arrays are the same.");
else
System.out.println("The arrays are not the same.");
```

- When we use the == operator with reference variables, the operator compares the memory addresses that the variables contain, not the contents of the objects referenced by the variables. Because the two array variables in this code reference different objects in memory, they will contain different addresses. Therefore, the result of the boolean expression firstArray == secondArray is false, and the code reports that the arrays are not the same.

Comparing Arrays

```
import java.util.Arrays;
public class Test {
   public static void main(String args[]) {
    int arr1[] = {1, 2, 3};
    int arr2[] = {1, 2, 3};
        if (Arrays.equals(arr1, arr2))
            System.out.println("Arrays are Same");
        else
            System.out.println("Arrays are Not same");
   }
}
```

String Arrays

- **What are Strings:**
- The **collection** of the **alphabets** is called **string**.
- **For Example:-**
  - 1. “London”.
  - 2. “Austria”.
  - 3. “Football”.
  - 4. “Adil”
  - 5. “Tooth Paste”

Java String Array

- In this tutorial, we will show you how to create, initialize and print the **string arrays** in Java.
- The string arrays are used to store the string elements in arrays.
- Suppose, we want to store 50 States in our Java program in the string variables. What we can do is to create 50 string variables and store the State names in each variable, as below:
- *StrState1 = "**New York**"*
- *StrState2 = "**Washington**"*
- *StrState3 = "**Florida**"*
- *and so on.*

String Arrays

- String manipulation is the most common part of Java program. Strings represent a sequence of characters.
- The easiest way to represent a sequence of characters in Java is by using a character array.

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

String Arrays

- Although character arrays have the advantage of being able to query their length, they themselves are not good enough to support the range of operation we may like to perform on stings.

- **For Example**
- Copying one character array into another might require a lot of book keeping effort.
- Fortunately, Java is equipped to handle these situations more efficiently.

String Arrays

- In Java, strings are class objects and implemented using two classes, namely, **String** and **StringBuffer**.
- A Java string is an instantiated object of the **String** class. Java strings, as compared to C strings, are more reliable and predictable. This is basically due to C's lack bounds-checking.
- A Java string is not a character array and is not NULL terminated.

String Arrays

- Strings may be declared and created as follows:

```
String[] stringName;
stringName = new String[] {"string"};
```

- **Example**

```
Sting[] firstName;
firstName = new String[] {"Adil"};
```

- These two Statements may be combined as follows:

```
String[] firstName = new String[] {"Adil"};
```

String Arrays

- We can also create and use arrays that contain strings. The statement

```
String[] listArray = new String[4];
```

- Will create an listArray of size 4 to hold three sting constants. We can assign the strings to the listArray elements by elements using three different statements or more efficiently using a for loop.
- **Syntax with Values Initialization**

```
String listArray = {"Adil"}
```

String Arrays Creation and Initialization

- Java also allows you to create arrays of String objects. Here is a statement that creates an array of String objects initialized with values.

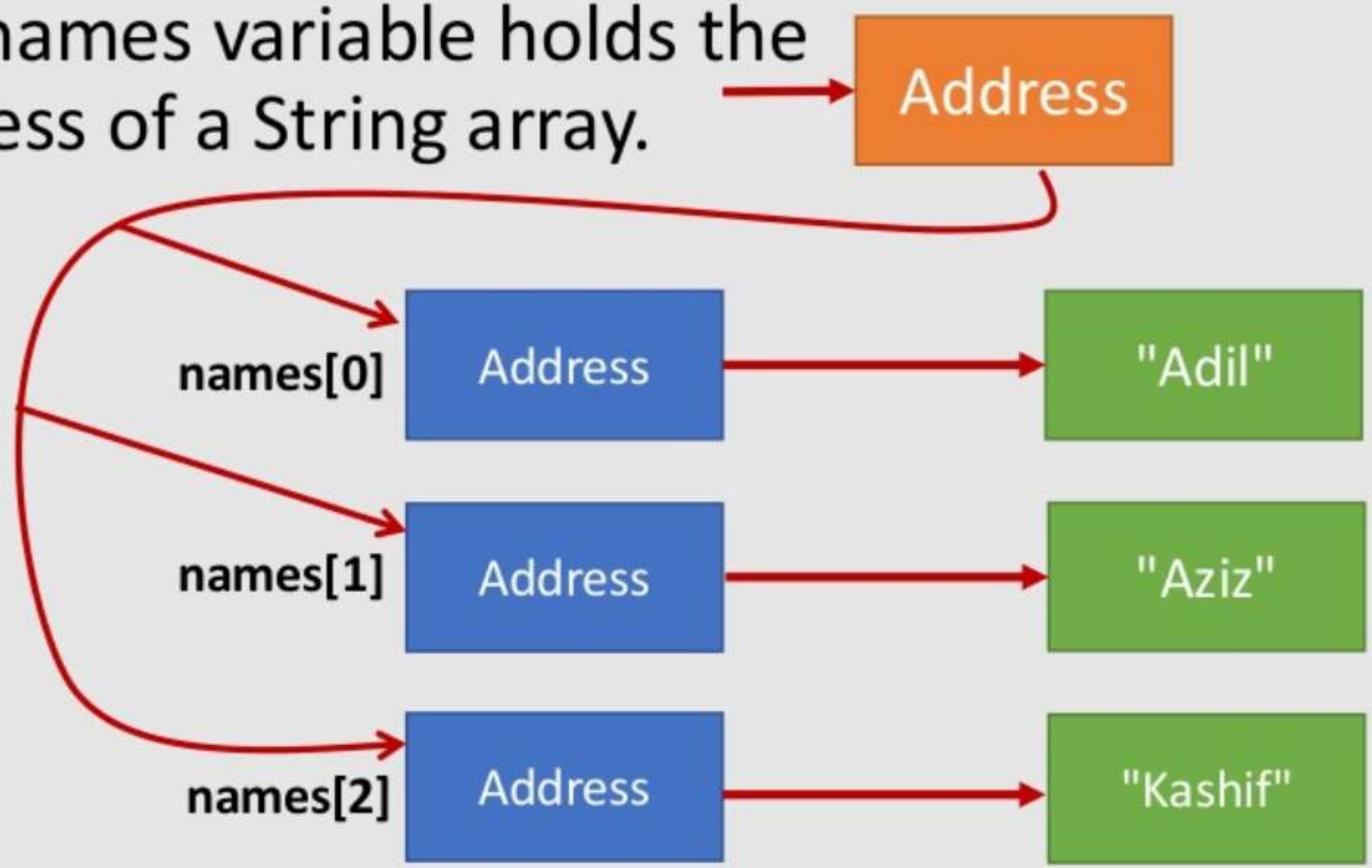
```
String[] names = {"Adil", "Aziz", "Kashif"};
```

- In memory, an array of String objects is arranged differently than an array of a primitive data type. To use a String object, you must have a reference to the String object. So, an array of String objects is really an array of references to String objects.

String Arrays Creation and Initialization

- A String array is an array of references to String objects.

The names variable holds the address of a String array.



Example of String Array

```
public class Test {
   public static void main(String args[]) {
      String str[] = new String[4];
      str[0]="A";
      str[1]="d";
      str[2]="i";
      str[3]="l";
      for (String str1 : str) {
         System.out.print(str1);
      }
   }
}
```

Output is:
Adil

Example of String Array (Alternative)

```
public class Test {
   public static void main(String args[]) {
      String str[] = {"Adil"};
      for (String str1 : str) {
         System.out.println(str1);
      }
   }
}
```

Calling String Methods from an Array Element

```
public class StringArrayMethods
{
public static void main(String[] args)
{
   // Create an array of Strings.
   String[] names = { "Adil", "Aslam", "Hina", "Ameen" };

   // Display each string in the names array
   // in uppercase.
   for (int index = 0; index < names.length; index++)
      System.out.println(names[index].toUpperCase());
 }
}
```

Calling String Methods from an Array Element

```
public class StringArrayMethods
{
public static void main(String[] args)
{
   // Create an array of Strings.
   String[] names = { "Adil", "Aslam", "Hina", "Ameen" };

   // Display each string in the names array
   // in uppercase.
   for (int index = 0; index < names.length; index++)
      System.out.println(names[index].toUpperCase());
 }
}
```

Output is:
ADIL
ASLAM
HINA
AMEEN

toUpperCase is a Method of String

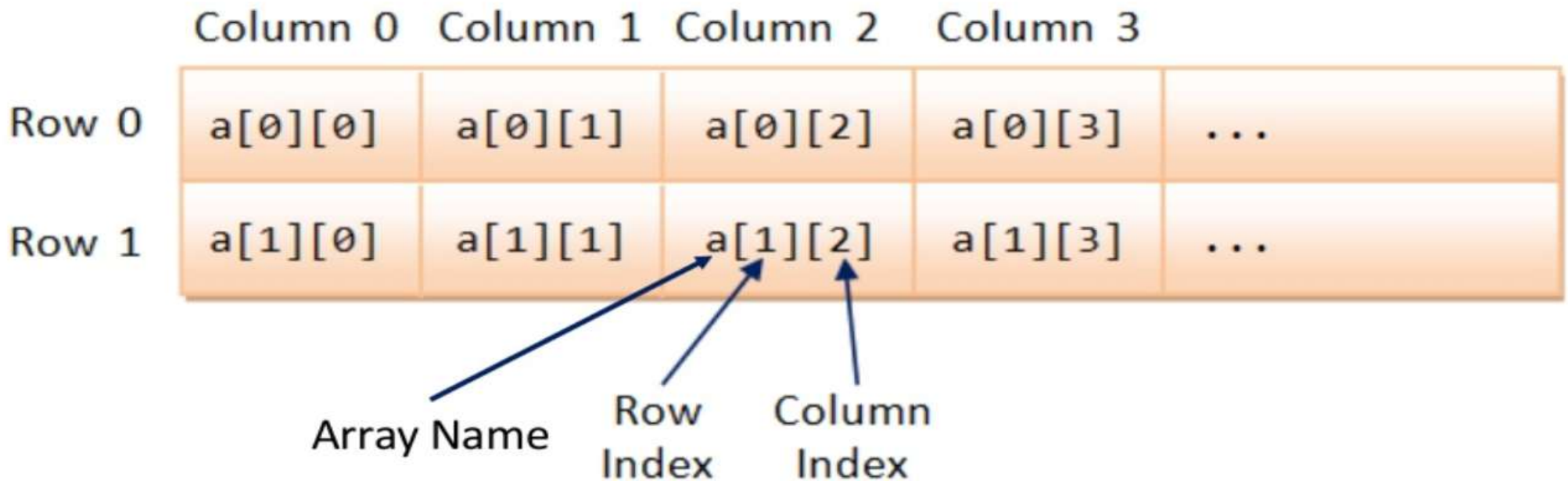
Calling String Methods from an Array Element

```
public class StringArrayMethods  {
 public static void main(String[] args)
 {
    // Create an array of Strings
    String[] names = { "Adil", "Aslam", "Hina", "Ameen" };

     // Display each string in the names array
     // in uppercase
     for (String name : names) {
        System.out.println(name.toUpperCase());
     }
  }
}
```

Multidimensional Array

- Arrays with two dimensions (i.e. subscripts) often represent tables of values consisting of information arranged in rows and columns.
- By convention, the first identifies the element's row and the second identifies the element's column
- Arrays that require two subscripts to identify a particular element are called two-dimensional arrays or 2-D arrays
- Arrays with two or more dimensions are known as multidimensional arrays

Two Dimensions Arrays



Example to Create two Dimensional Array in java

Create two Dimensional Array as:

```
//3 row and 3 column
int[][] arr = new int[3][3];

//3 row and 3 column
String[][] arr = new String[3]3];
```

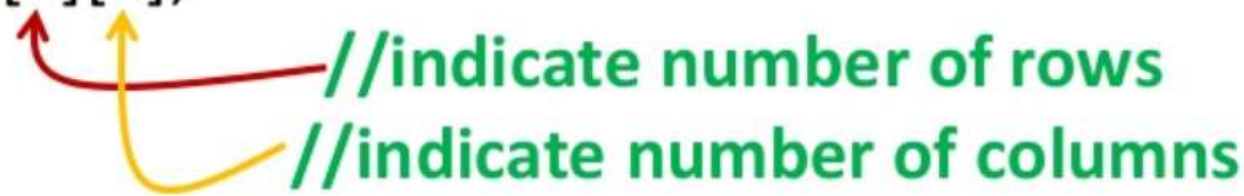
Example to initialize two Dimensional Array in java

```
int[][] arr = new int[2][2];
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
```

//indicate number of rows

Example to initialize two Dimensional Array in java

```
int[][] arr = new int[2][2];
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
```



Please Note that

- Like the one dimensional arrays, two dimensional arrays may me initialized by the following their declaration with list of initial values enclose in braces. For Example,

```
int arr[2][3] = {0,0,0,1,1,1} ,
```

- Initializes the elements of the first row to zero and second row to one. The initialization is done row by row. The above statement can be equivalently written as :

```
int arr[2][3] = { {0,0,0} , {1,1,1} },
```

- By surrounding the elements of each row by braces.
- We can also initialize a two dimensional array in the form of a matrix as shown below :

```
int arr[2][3] = {
                    {0,0,0} ,  //show one row
                    {1,1,1}
```

Example of two Dimensional Array in java

- **int**[][] arr = **new int**[2][3];
- Above statement will create 2*3 element array of an integer values that can by accessed by arr.

arr[0][0]	**arr[0][1]**	**arr[0][2]**
arr[1][0]	**arr[1][1]**	**arr[1][2]**

- **Initializing Values**
- **int** arr = {{1,2,3},{4,5,6}};

Two Dimensional Array in java

- By the way when you initially declare a two dimensional array, **we must remember to specify first dimension**, for example following array declaration is illegal in Java.

```
int[][] wrong = new int[][]; // not OK, you must specify 1st dimension
 int[][] right = new int[2][];   // OK
```

- The first expression will throw "Variable must provide either dimension expressions or an array initializer" error at compile time. On the other hand second dimension is optional and even if you don't specify compiler will not complain

Example of two Dimensional Array in java

```
public class Example {
   public static void main(String[] args) {
            int arr[][]= {{1,2},{3,4}};
      for (int i = 0; i < arr.length; i = i + 1)
      {
         for(int j=0; j < arr[i].length; j = j + 1)
         {
          System.out.print(arr[i][j]+"   ");
         }
         System.out.println();
      }

   }
}
```

Same Program as Previous Program with some Modification

Note the Output in Matrix form

Output is:

1 2

3 4

Another Example of 2D Array

```
public class Test {
    public static void main(String[] args) {
        String[][] salutation = {
            {"Mr. ", "Mrs. ", "Ms. "},
            {"Adil"}
        };
        // Mr. Adil
        System.out.println(salutation[0][0] + salutation[1][0]);
        // Mrs. Adil
        System.out.println(salutation[0][1] + salutation[1][0]);
    }
}
```

Output is:

Mr. Adil

Mrs. Adil

Passing Two-Dimensional Arrays to Methods

- When a two-dimensional array is passed to a method, the parameter must be declared as a reference to a two-dimensional array.
- The following method header shows an example.

```
private static void showArray(int[][] array)
```

- This method's parameter, array, is declared as a reference to a two-dimensional int array. Any two-dimensional int array can be passed as an argument to the method.

Passing Two-Dimensional Arrays to Methods-1

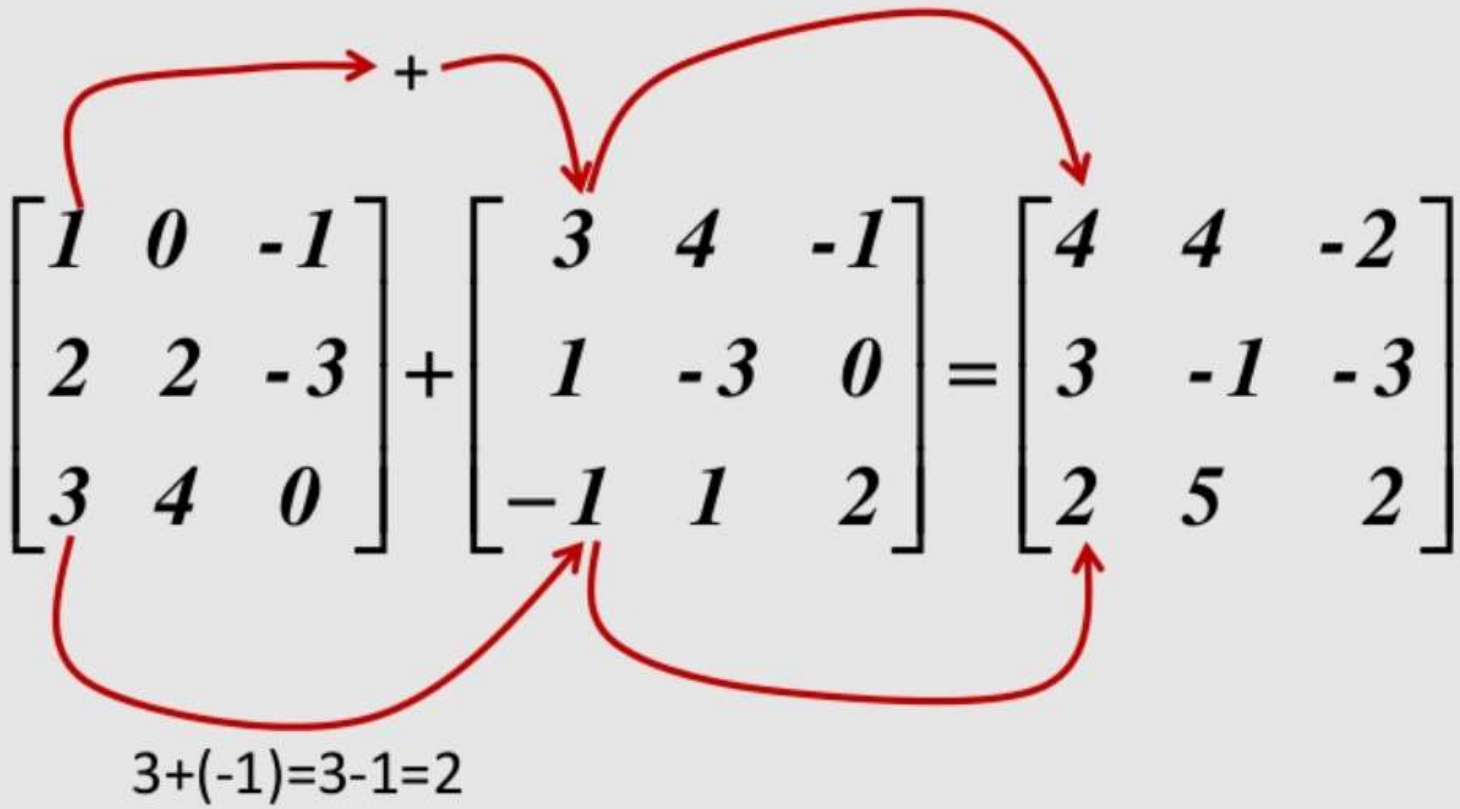
```
public class Pass2Darray {
public static void main(String[] args) {
   // Create a 2D array of integers
   int[][] numbers = { { 1, 2, 3, 4 },
                       { 5, 6, 7, 8 },
                       { 9, 10, 11, 12 } };
   System.out.println("Here are the Values in the Array");
   showArray(numbers);
   System.out.println("The Sum of the Values is "+
   arraySum(numbers));
}
```

Passing Two-Dimensional Arrays to Methods-2

```
private static void showArray(int[][] array)
{
   for (int[] array1 : array)
    {
      for (int col = 0; col < array1.length; col++)
       {
          System.out.print(array1[col] + " ");
       }
       System.out.println();
   }
}
```

Addition of Two Matrices

- In mathematics addition of two matrices is:



Addition of 2 Matrices in java

```
12  public class Test {
13      public static void main(String[] args) {
14       //creating two matrices
15      int a[][]={{1,2,3},{4,5,6}};
        int b[][]={{1,2,3},{4,5,6}};
17      //creating another matrix to store the sum of two matrices
18      int c[][]=new int[2][3];
19      //adding and printing addition of 2 matrices
20      for(int i=0;i<2;i++){
21          for(int j=0;j<3;j++){
22              c[i][j]=a[i][j]+b[i][j];
23              System.out.print(c[i][j]+" ");
24          }
25          System.out.println();//new line
26          }
27
28      }
29  }
```

Output - Test (run)

```
run:
2 4 6
8 10 12
```