CSE310: Programming in Java

Topic: Array and Enum

Outlines

- Introduction
- Array Creation
- Array Initialization
- Enumerations

Array

- Definition:

  An array is a finite 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).
- Array elements are stored in contiguous memory locations.

- Examples:
  - **C**ollection of numbers
  - **C**ollection of names

More points

- In Java all arrays are dynamically allocated.
- Since arrays are objects in Java, we can find their length using the object property *length*.
- The direct superclass of an array type is *Object.*
- If we try to access array outside of its index then *ArrayIndexOutOfBounds* Exception will be raised

One-Dimensional Arrays

- A one-dimensional array is a list of variables of same type.
- The general form of a one-dimensional array declaration is:

type [] arr_ref_var; ***OR***

type [] arr_ref_var= new type[size];

Example:

int [] num = new int [10];

It will create an array of 10 integers.

Syntax and Example

Declaration of array variable:

data-type variable-name[];

eg. *int marks[];*

This will declare an array named ‘marks’ of type ‘int’. But no memory is allocated to the array.

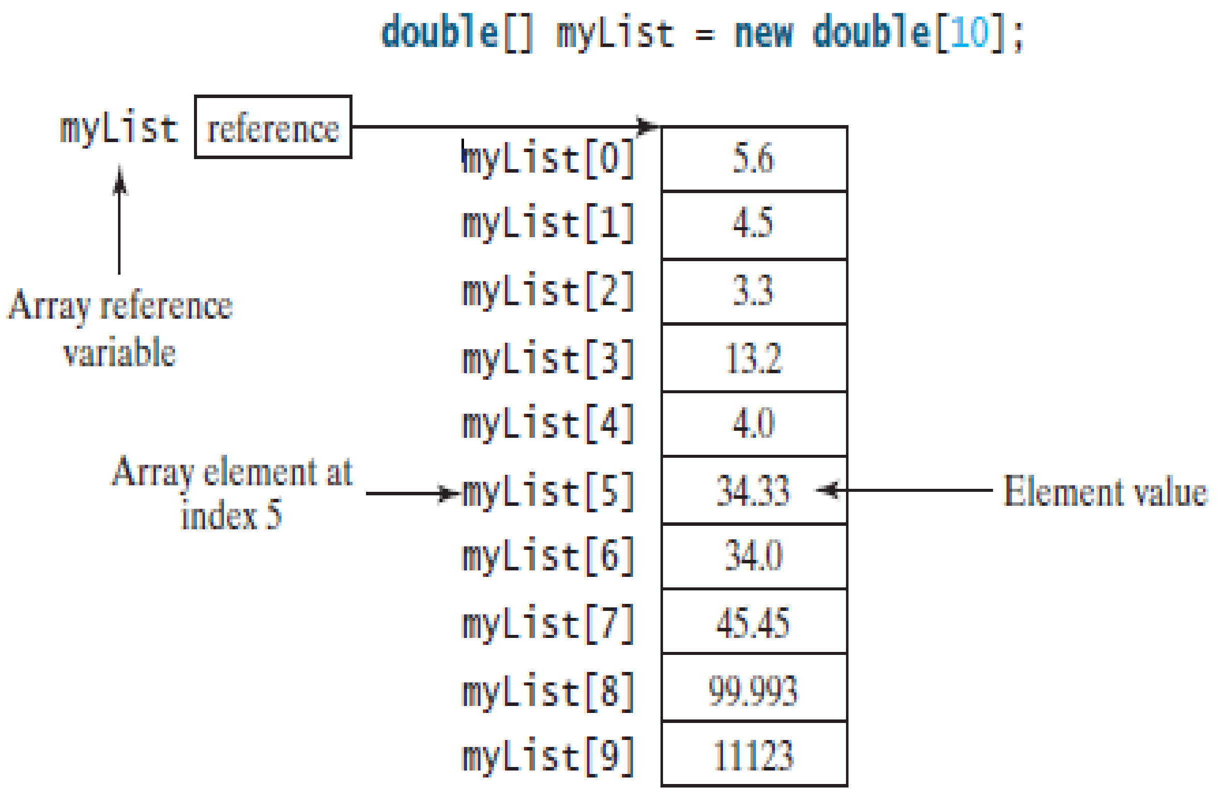
Allocation of memory:

variable-name = new data-type[size];

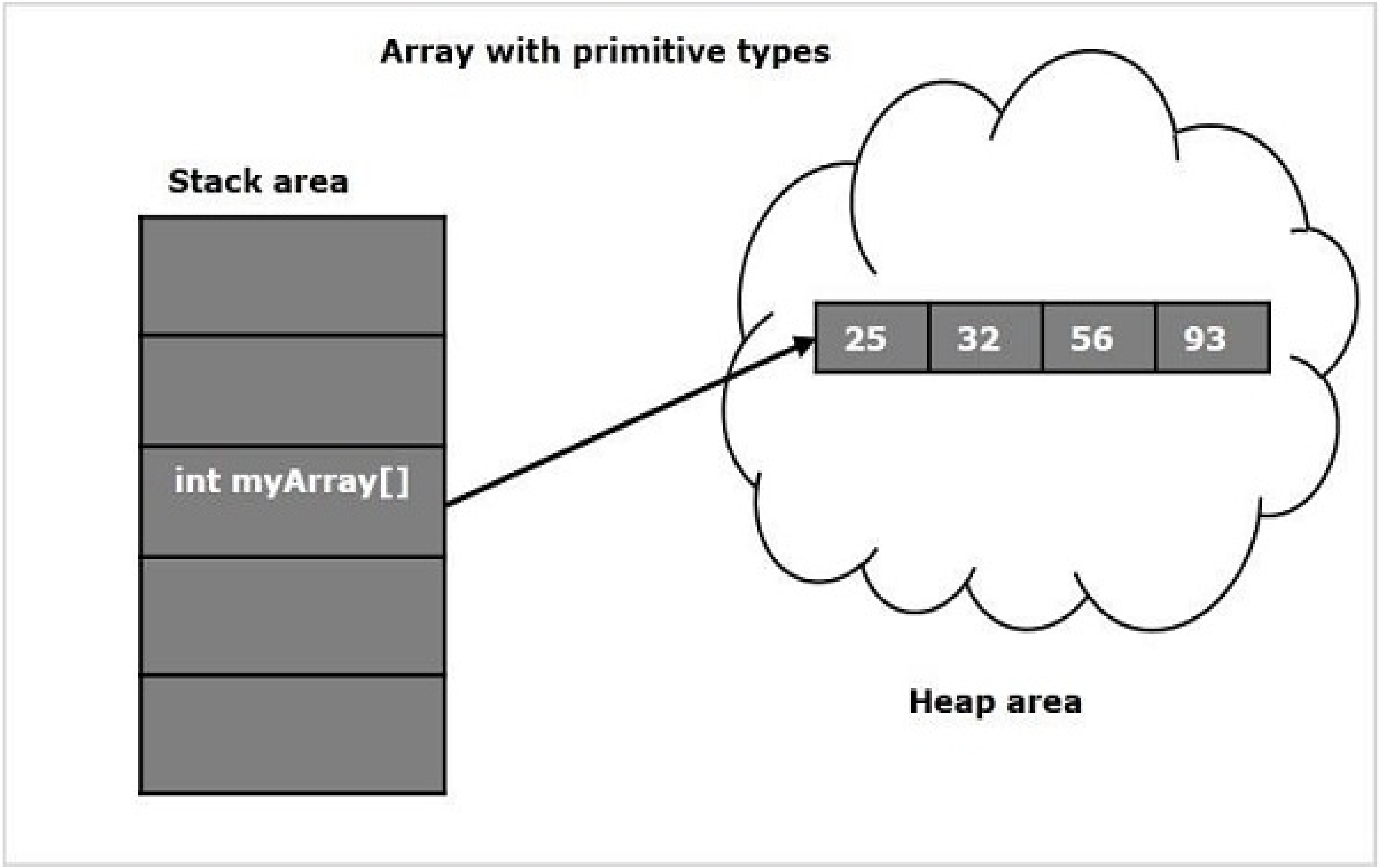
eg. *marks = new int[5];*

This will allocate memory of 5 integers to the array ‘marks’ and it can store upto 5 integers in it. ‘new’ is a special operator that allocates memory.

Another Example of array



Memory allocation



Accessing elements in the array:

• Specific element in the array is accessed by specifying name of the array followed the index of the element.

• All array indexes in Java start at zero.

variable-name[index] = value;

Example:

marks[0] = 10;

This will assign the value 10 to the 1st element in the array.

marks[2] = 863;

This will assign the value 863 to the 3rd element in the array.

Example

STEP 1 : (Declaration)

int marks[];

marks → null

STEP 2: (Memory Allocation)

marks = new int[5];

marks →

0	0	0	0	0
marks[0]	marks[1]	marks[2]	marks[3]	marks[4]

STEP 3: (Accessing Elements)

marks[0] = 10;

marks →

10	0	0	0	0
marks[0]	marks[1]	marks[2]	marks[3]	marks[4]

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

- Default values:
  - zero (0) for numeric data types,
  - \u0000 for chars and
  - false for Boolean types

- Length of an array can be obtained as:

  array_ref_var.length

Examples…

```
// to show the working of single dimension array
class Example
{
  public static void main(String args[])
   {
         int a[] = new int[5];
         a[0]=12;a[1]=34;a[2]=56;a[3]=78;a[4]=90;
         System.out.println("Length of the array is "+a.length);
         System.out.println("Printing the elements of array");
         for(int i=0;i<a.length;i++)
         System.out.println(a[i]);
   }
}
```

```
// to show the working of single dimension array
import java.util.Scanner;
class Example
{
  public static void main(String args[])
    {
     int n;// number of elements in array
     Scanner ob = new Scanner(System.in);
     System.out.println("Enter the number of elements in array");
     n=ob.nextInt();
     int a[] = new int[n];
     System.out.println("Enter"+n+"elements of array");
     for(int i=0;i<a.length;i++)
     a[i]=ob.nextInt();
     System.out.println("Printing the elements of array");
     for(int i=0;i<a.length;i++)
     System.out.println(a[i]);
    }
}
```

Note

- Arrays can store elements of the ***same data type***. Hence an ***int*** array CAN NOT store an element which is not an int.

- Though an element of a compatible type can be converted to int and stored into the int array.

eg. marks[2] = (int) 22.5;

This will convert '22.5' into the int part '22' and store it into the 3rd place in the int array 'marks'.

- Array indexes start from zero. Hence 'marks[index]' refers to the (index+1)th element in the array and 'marks[size-1]' refers to last element in the array.

- For an array of the char[] type, it can be printed using one print statement. For example, the following code displays Dallas:

```
char[] city = {'D', 'a', 'l', 'l', 'a', 's'};
System.out.println(city);
```

- Accessing an array out of bounds is a common programming error that throws a runtime ArrayIndexOutOfBoundsException. To avoid it, make sure that you do not use an index beyond arrayRefVar.length – 1.

Array Initialization

1. data Type [] array_ref_var = {value0, value1, …, value n};
2. data Type [] array_ref_var = new data Type [n];

   array_ref_var [0] = value 0;

   array_ref_var [1] = value 1;

   …

   array_ref_var [n-1] = value n;

3. data type [] array_ref_var = new int[] {value1,value 2..}

Array initialization: Example

```
class Example
{
   public static void main(String args[])
     {
         int [] a = new int [] {1,2,3,4,5};
            for(int i : a)
            System.out.println(i);


     }
}
```

Printing array elements using for each loop

We can also print the Java array using for-each loop. The Java for-each loop prints the array elements one by one. It holds an array element in a variable, then executes the body of the loop.

The syntax of the for-each loop is given below:

```
 for(data_type variable:array)
{
 //body of the loop
 }
```

Example 1

```
class Example
{
    public static void main(String args[])
     {
      int arr[]={33,3,4,5};
      //printing array using for-each loop
      for(int i:arr)
      System.out.println(i);
     }
}
```

Example 2

```
import java.util.Scanner;
class Example
{
   public static void main(String args[])
      {
        int i;
        String s[] = new String[5];
        Scanner ob = new Scanner(System.in);
        System.out.println("Enter the 5 strings");
        for(i=0;i<s.length;i++)
        s[i]=ob.nextLine();
        System.out.println("5 strings are");
        for(String x : s)
        System.out.println(x);
      }
}
```

Exercise

Write a program which prompts the user to enter the number of elements.Now read the marks of all the subjects from the user using Scanner class.Write a method which calculates the percentage of the user.

Multi-Dimensional Array

- Multidimensional arrays are arrays of arrays(2D,3D….)
- Two-Dimensional arrays are used to represent a table or a matrix.
- A two-dimensional array is actually an array in which each element is a one-dimensional array.
- Declaration:

  elementType[][] arrayRefVar; or elementType arrayRefVar[][];

  Example: int[][]a; or int a[][];

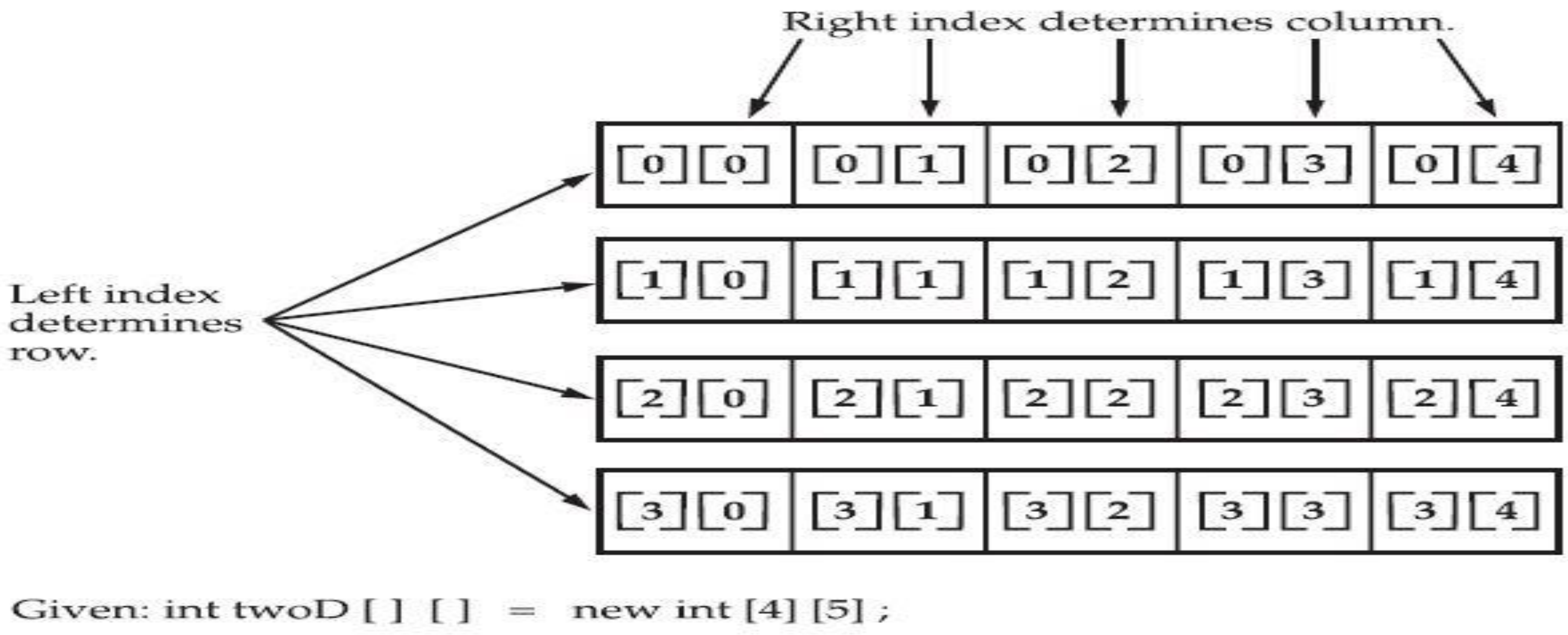
  Creating 2D array:

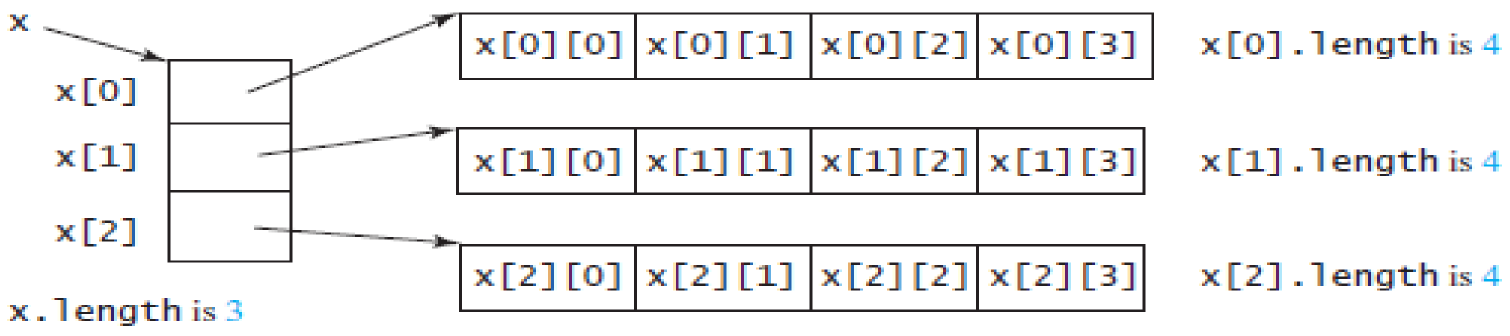
  elementType[][] arrayRefVar=new elementType[n][m];

  Example:

  int twoD[][] = new int[4][5];

Conceptual View of 2-Dimensional Array





x
x[0]
x[1]
x[2]
x.length is 3
x[0][0] x[0][1] x[0][2] x[0][3]
x[1][0] x[1][1] x[1][2] x[1][3]
x[2][0] x[2][1] x[2][2] x[2][3]
x[0].length is 4
x[1].length is 4
x[2].length is 4

A two-dimensional array is a one-dimensional array in which each element is another one-dimensional array.

```
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();
              }
          }
    }
```

Output:

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

- When we allocate memory for a multidimensional array, we need to only specify the memory for the first (leftmost) dimension.

```
int twoD[][] = new int[4][];
```

- The other dimensions can be assigned manually.

Initializing Multi-Dimensional Array

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

Syntax—Giving other dimensions manually

```
Syntax: data_type array_name[][] = new data_type[n][];  //n: no. of rows
          array_name[] = new data_type[n1] //n1= no. of colmuns in row-1
          array_name[] = new data_type[n2] //n2= no. of colmuns in row-2
          array_name[] = new data_type[n3] //n3= no. of colmuns in row-3
          ---------------------------------------------------------------------------------------
          array_name[] = new data_type[nk]  //nk=no. of colmuns in row-n
```

This type of array is also known as Jagged/ or ragged arrays

Program example-Jagged arrays

```
// Program to demonstrate 2-D jagged array in Java
class Main
{
	public static void main(String[] args)
	{
	  int arr[][] = new int[2][];
	  arr[0] = new int[3];
	  arr[1] = new int[2];
	  int count = 0;
	  for (int i=0; i<arr.length; i++)
		for(int j=0; j<arr[i].length; j++)
			arr[i][j] = count++;
	  System.out.println("Contents of 2D Jagged Array");
	  for (int i=0; i<arr.length; i++)
	  {
		for (int j=0; j<arr[i].length; j++)
			System.out.print(arr[i][j] + " ");
		System.out.println();
	  }
	}
}
```

Output:

```
Contents of 2D Jagged Array
0 1 2
3 4
```

Array Cloning

- To actually create another array with its own values, Java provides the **clone()** method.

- arr2 = arr1; (assignment)

  is not equivalent to

  arr2 = arr1.clone(); (cloning)

In first case, Only one array is created and two references arr1 and arr2 are pointing to the same array. While in second case two different arrays are created.

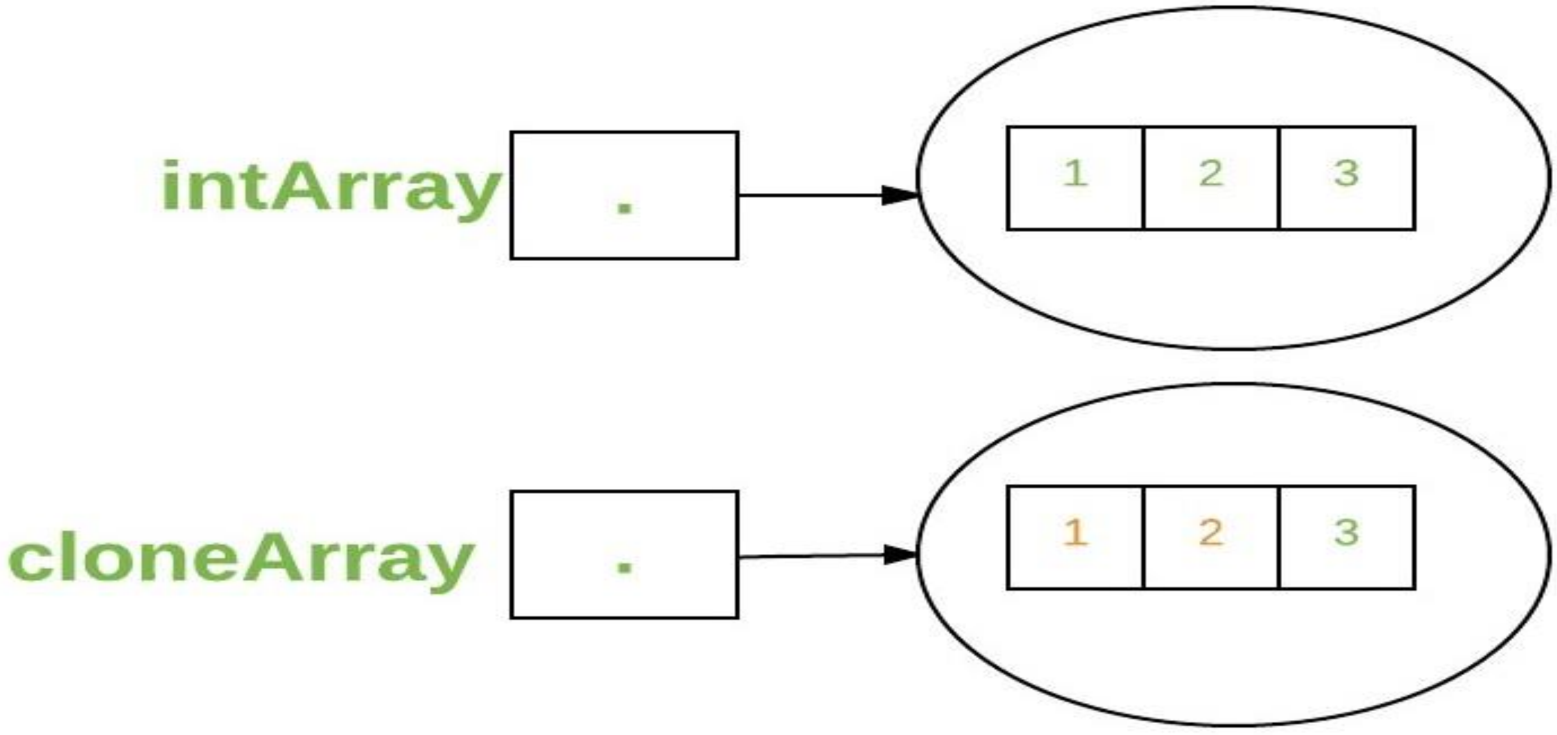
Cloning of 1D Array

```
// Java program to demonstrate
// cloning of one-dimensional arrays
class Test
{
    public static void main(String args[])
    {
      int intArray[] = {1,2,3};
      int cloneArray[] = intArray.clone();
      // will print false as deep copy is created
      // for one-dimensional array
      System.out.println(intArray == cloneArray);
      for (int i = 0; i < cloneArray.length; i++) {
            System.out.print(cloneArray[i]+" ");
      }
    }
}
```

Output:

```
false
1 2 3
```

Deep copy is created for ID array



Deep Copy is created for one-dimensional array by clone() method

Cloning of 2D Array

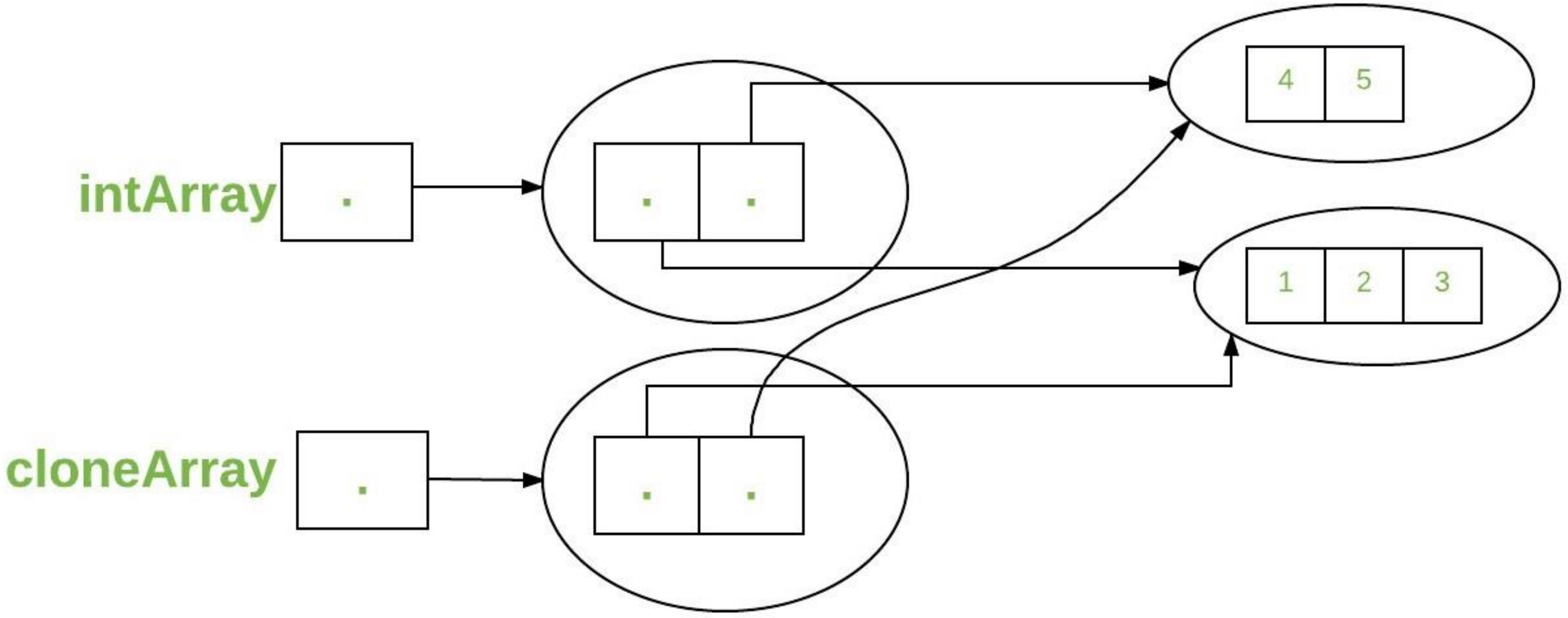
A clone of a multi-dimensional array (like Object[][]) is a “shallow copy” however, which is to say that it creates only a single new array with each element array a reference to an original element array, but subarrays are shared.

```
// Java program to demonstrate
// cloning of multi-dimensional arrays
class Test
{
    public static void main(String args[])
    {
      int intArray[][] = {{1,2,3},{4,5}};
      int cloneArray[][] = intArray.clone();
      // will print false
      System.out.println(intArray == cloneArray);
      // will print true as shallow copy is created
      // i.e. sub-arrays are shared
      System.out.println(intArray[0] == cloneArray[0]);
      System.out.println(intArray[1] == cloneArray[1]);
    }
}
```

Output:

```
false
true
true
```

Shallow copy created for 2D array



Shallow Copy is created for multi-dimensional array by clone() method

Assignment for Practice

- WAP in which prompt the user to enter the number of subjects and number of CA in each subject. Read the marks of each CA and store in a two dimensional array.

Q1

Which of these is the correct syntax for array creation?

a) int arr[] = new arr[5]

b) int [5] arr = new int[]

c) int arr[5] = new int[]

d) int arr[] = new int [5]

Q2

Which of these is an incorrect Statement?
a) It is necessary to use new operator to initialize an array
b) Array can be initialized using comma separated expressions surrounded by curly braces
c) Array can be initialized when they are declared
d) None of the mentioned

Q3

What will be the output of the following Java code?

```
class array_output
{
    public static void main(String args[])
    {
        int array_variable [] = new int[10];
        for (int i = 0; i < 10; ++i)
        {
            array_variable[i] = i;
            System.out.print(array_variable[i] + " ");
            i++;
        }
    }
}
```

a) 0 2 4 6 8

b) 1 3 5 7 9

c) 0 1 2 3 4 5 6 7 8 9

d) 1 2 3 4 5 6 7 8 9 10

Q4

What will be the output of the following Java code?

```
class evaluate
{
    public static void main(String args[])
        {
          int arr[] = new int[] {0 , 1, 2, 3, 4, 5, 6, 7, 8, 9};
          int n = 6;
            n = arr[arr[n] / 2];
          System.out.println(arr[n] / 2);
        }
}
```

a) 3

b) 0

c) 6

d) 1

Q5

What will be the output of the following Java code?

```
class array_output
{
    public static void main(String args[])
    {
        int array_variable[][] = {{ 1, 2, 3}, { 4 , 5, 6}, { 7, 8, 9}};
        int sum = 0;
        for (int i = 0; i < 3; ++i)
            for (int j = 0; j <  3 ; ++j)
                sum = sum + array_variable[i][j];
        System.out.print(sum / 5);
    }
}
```

a) 8

b) 9

c) 10

d) 11

Q6(Output??)

```
public class Main
{
    public static void main(String[] args) {
     int a[][]=new int[2][2];
     System.out.println(a[0][1]);
    }
}
```

A. 0

B. 1

C. Compile time error

D. Run time error

Java Enum

Introduction

- Enum in java is a data type that contains fixed set of constants.
- It can be thought of as a class having fixed set of constants.
- The java enum constants are static and final implicitly. It is available from JDK 1.5.
- It can be used to declare days of the week, Months in a Year etc.

Advantages of Enum

- enum improves type safety
- enum can be easily used in switch
- enum can be traversed
- enum can have fields, constructors and methods

Important

- Enum can not be instantiated using new keyword because it contains private constructors only.

- The enum can be defined within or outside of the class because it is similar to a class.

- Every enum constant is always implicitly **public static final**. Since it is **static**, we can access it by using enum Name.

Examples...

```
// A simple enum example where enum is declared  outside any class (Note
    enum keyword instead of  class keyword)

enum Color

{

    RED, GREEN, BLUE;

}

public class Test

{

    public static void main(String[] args)

    {

     Color c1 = Color.RED;

     System.out.println(c1);

    }

}
```

values(),valueOf() and ordinal() method

- The java compiler internally adds the values(),valueOf() and ordinal() methods when it creates an enum.

- The values() method returns an array containing all the values of the enum.

- **valueOf() method** returns the enum constant of the specified string value, if exists

- By using **ordinal() method**, each enum constant index can be found, just like array index.

Examples….

```
class Example
    {
        public static void main(String args[])
        {
        enum session {WINTER,SUMMER,FALL}
        for(session s : session.values())
        System.out.println(s);
        }
    }
```

Examples….

```
enum session {WINTER,SUMMER,FALL}
class Example
    {
        public static void main(String args[])
        {
        for(session s : session.values())
        System.out.println(s);
        }
    }
```

// Working of values(), valueOf() and ordinal() method

```
class Example
    {
     public enum Season { WINTER, SPRING, SUMMER, FALL }
     public static void main(String[] args) {

     for (Season s : Season.values()){
          System.out.println(s);
          }
     System.out.println("Value of WINTER is: "+Season.valueOf("WINTER"));
     System.out.println("Index of WINTER is: "+Season.valueOf("WINTER").ordinal());
     System.out.println("Index of SUMMER is: "+Season.valueOf("SUMMER").ordinal());
    }
}
```

Enum with switch

```
// An enumeration of apple varieties.
enum Apple {
Jonathan, GoldenDel, RedDel, Winesap, Cortland
}
public class Main {
public static void main(String args[])
{
Apple ap;
ap = Apple.RedDel;
// Use an enum to control a switch statement.
switch(ap) {
case Jonathan:
System.out.println("Jonathan is red.");
break;
case GoldenDel:
System.out.println("Golden Delicious is yellow.");
break;
case RedDel:
System.out.println("Red Delicious is red.");
break;
case Winesap:
System.out.println("Winesap is red.");
break;
}
}
}
```

Java Enumeration are class types(We can give them constructors, add instance variables and methods etc)

```
// Use an enum constructor, instance variable, and method.
enum Apple {
Jonathan(10), GoldenDel(9), RedDel(12), Winesap(15), Cortland(8);
private int price; // price of each apple
// Constructor
Apple(int p) { price = p; }
int getPrice() { return price; }
}
public class Main {
public static void main(String args[])
{
Apple ap;
// Display price of Winesap.
System.out.println("Winesap costs " +Apple.Winesap.getPrice() +" cents.\n");
// Display all apples and prices.
System.out.println("All apple prices:");
for(Apple a : Apple.values())
System.out.println(a + " costs " + a.getPrice() +" cents.");
}
}
```

Key points....

- enum can contain constructor and it is executed separately for each enum constant at the time of enum class loading.
- We can't create enum objects explicitly and hence we can't invoke enum constructor directly.
- Every enum constant represents an **object** of type enum.

Q1(Output??)

```
enum Season
{
   WINTER,SUMMER,SPRING;
}
public class Main
{
      public static void main(String[] args) {
      Season var;
      var=SPRING;
      System.out.println(var);
      }
}
```

A. 0

B. Compile time error

C. -1

D. SPRING

Q2(Output??)

```
enum Flowers
{
   SUNFLOWER,JASMINE,LOTUS;
}
public class Main
{
     public static void main(String[] args) {
     Flowers var[]=Flowers.values();
     for(int i=1;i<2;i++)
     System.out.println(var[i]);
     }
}
```

A. JASMINE

B. LOTUS

C. SUNFLOWER

D. 1

Q3(Output??)

```
enum Colours
{
   WHITE,GREEN,RED,YELLOW
}
public class Main
{
     public static void main(String[] args) {
     System.out.println(Colours.valueOf("YELLOW").ordinal());
}
}
```

A. 0

B. 1

C. 2

D. 3

Q4(Output??)

```
enum Colours
{
  WHITE(23),GREEN(78),RED(7),YELLOW(100);
   int colour_code;
   Colours(int code){
      colour_code=code;
   }
   int get_code(){
      return colour_code;
   }
}
public class Main
{
       public static void main(String[] args) {
       System.out.println(Colours.RED.get_code());
}
}
```

A. 7

B. 0

C. 100

D. 23