Core Java

Akshita Chanchlani

Contents

- Array
- Enum

Array Introduction

- Array, stack, queue, LinkedList are data structures.
- In Java, data structure is called collection and value stored inside collection is called element.
- Array is a sequential/linear container/collection which is used to store elements of same type in continuous memory location.

In C/C++

Static Memory allocation for array

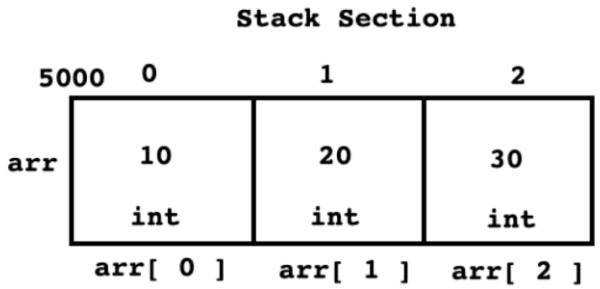
```
int arr1[ 3 ];    //OK

int size = 3;
int arr2[ size ];     //OK
```

In C/C++

Dynamic Memory allocation for array

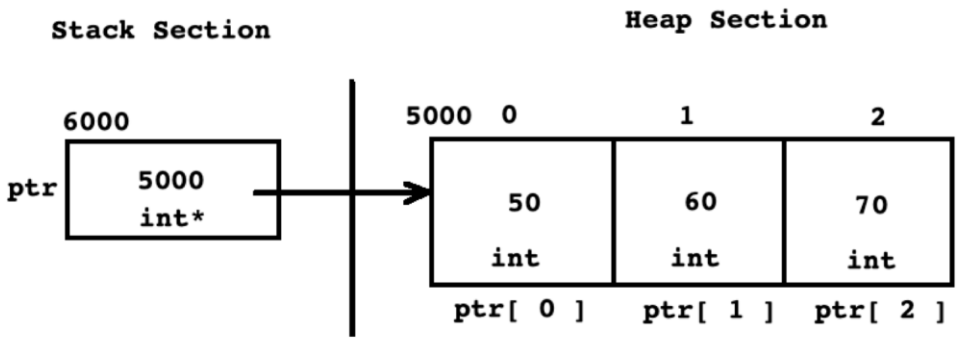
```
int *arr = ( int* )malloc( 3 * sizeof( int ));
//or
int *arr = ( int* )calloc( 3, sizeof( int ));
```

Static v/s Dynamic Memory Allocation In C/C++



[Static Memory Allocation]

```
arr[ 0 ]  =>  10
arr[ 1 ]  =>  20
arr[ 2 ]  =>  30
```



[Dynamic Memory Allocation]

```
ptr[ 0 ]  =>  50
ptr[ 1 ]  =>  60
ptr[ 2 ]  =>  70
```

Array Declaration and Initialization In C

- `int arr[ 3 ]; //OK : Declaration`
- `int arr[ 3 ] = { 10, 20, 30 }; //OK : Initialization`
- `int arr[ ] = { 10, 20, 30 }; //OK`
- `int arr[ 3 ] = { 10, 20 }; //OK : Partial Initialization`
- `int arr[ 3 ] = { }; //OK : Partial Initialization`
- `int arr[ 3 ] = { 10, 20, 30, 40, 50 }; //Not recommended`

Accessing Elements Of Array

- If we want to access elements of array then we should use integer index.
- Array index always begins with 0.

```
int arr[ 3 ] = { 10, 20, 30 };
printf("%d\n", arr[ 0 ] );
printf("%d\n", arr[ 1 ] );
printf("%d\n", arr[ 2 ] );
```

```
int arr[ 3 ] = { 10, 20, 30 };
int index;
for( index = 0; index < 3; ++ index )
    printf("%d\n", arr[ index ] );
```

Advantage and Disadvantages Of Array

- **Advantage Of Array**

  1. We can access elements of array randomly.

- Disadvantage Of Array

  1. We can not resize array at runtime.
  2. It requires continuous memory.
  3. Insertion and removal of element from array is a time consuming task.
  4. Using assignment operator, we can not copy array into another
  5. Compiler do not check array bounds(min and max indexarray).

Array In Java

- Array is a reference type in Java. In other words, to create instance of array, new operator is required. It means that array instance get space on heap.
- **There are 3 types of array in Java:**
    1. Single dimensional array
    2. Multi dimensional array
    3. Ragged array
- **Types of loop in Java:**
    1. do-while loop
    2. while loop
    3. for loop
    4. for-each loop
- **To perform operations on array we can use following classes:**
    1. java.util.Arrays

Methods Of java.util.Arrays Class

Following are the methods of java.util Arrays class.(try javap java.util.Arrays)

- public static <T> List<T> asList(T... a)
- public static int binarySearch(int[] a, int key) //Overloaded
- public static int binarySearch(Object[] a, Object key)
- public static int[] copyOf(int[] original, int newLength)
- public static <T> T[] copyOf(T[] original, int newLength)
- public static int[] copyOfRange(int[] original, int from, int to)
- public static <T> T[] copyOfRange(T[] original, int from, int to)
- public static void fill(int[] a, int val)
- public static void fill(Object[] a, Object val)
- public static void fill(Object[] a, int fromIndex, int toIndex, Object val)
- public static void sort(int[] a) //Overloaded
- public static void sort(Object[] a)
- public static void parallelSort(int[] a)
- public static <T extends Comparable<? super T>> void parallelSort(T[] a)
- public static String toString(Object[] a) //Overloaded
- public static String deepToString(Object[] a)
- public static IntStream stream(int[] array)   //Overloaded
- public static <T> Stream<T> stream(T[] array)

Single Dimensional Array

Reference declaration

```
int arr[ ]; //OK
int [ arr ]; //NOT OK
int[ ] arr; //OK
```

Instantiation

```
int[ ] arr1 = new int[ 3 ];
//or
int size = 3;
int[ ] arr2 = new int[ size ];
```

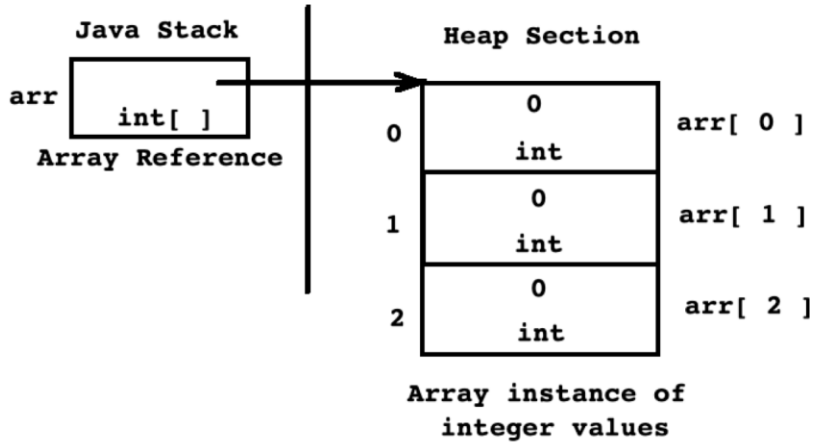
```
int[] arr1 = new int[ -3 ];  //NegativeArraySizeException
//or
int size = -3;
int[] arr2 = new int[ size ];    //NegativeArraySizeException
```

Initialization

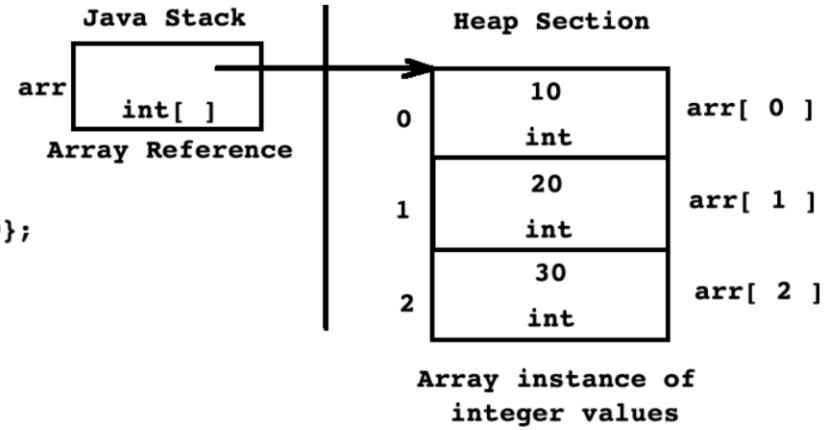
```
int[] arr = new int[ size ]{ 10, 20, 30 };  //Not OK
int[] arr = new int[  ]{ 10, 20, 30 };  //OK
int[] arr = { 10, 20, 30 };  //OK
```

Single Dimensional Array

```
int[] arr = new int[3];
```




```
int[] arr = new int[]{10,20,30};
```



Using length Field

```
public class Program {
    public static void printRecord( int[] arr ) {
        for( int index = 0; index < arr.length; ++ index )
            System.out.print( arr[ index ] +"    ");
        System.out.println();
    }
    public static void main(String[] args) {
        int[] arr1 = new int[   ] { 10, 20, 30 };
        Program.printRecord(arr1);

        int[] arr2 = new int[   ] { 10, 20, 30, 40, 50 };
        Program.printRecord(arr2);

        int[] arr3 = new int[   ] { 10, 20, 30, 40, 50, 60, 70   };
        Program.printRecord(arr3);
    }
}
```

Using toString( ) Method

```
public static void main(String[] args) {
    int[] arr = new int[   ] { 10, 20, 30, 40, 50 };
    System.out.println(arr.toString()); //[I@6d06d69c
}
public static void main(String[] args) {
    double[] arr = new double[   ] { 10.1, 20.2, 30.3, 40.4, 50.5 };
    System.out.println(arr.toString()); //[D@6d06d69c
}
//Check the documentation of getName() method of java.lang.Class.
public static void main(String[] args) {
    int[] arr = new int[   ] { 10, 20, 30, 40, 50 };
    System.out.println(Arrays.toString(arr));   //[10, 20, 30, 40, 50]
}
```

ArrayIndexOutOfBoundsException

- Using illegal index, if we try to access elements of array then JVM throws ArrayIndexOutOfBoundsException. Consider following code:

```
public static void main(String[] args) {
    int[] arr = new int[  ] { 10, 20, 30, 40, 50 };
    //int element = arr[ -1 ]; //ArrayIndexOutOfBoundsException
    //int element = arr[ arr.length ]; //ArrayIndexOutOfBoundsException
    //int element = arr[ 7 ]; //ArrayIndexOutOfBoundsException
}
```

ArrayStoreException

- If we try to store incorrect type of object into array then JVM throws ArrayStoreException.
- Consider the following code:

```
public class Program {
    public static void main(String[] args) {
        Object[] arr = new String[ 3 ];
        arr[ 0 ] = new String("DAC");    //OK
        arr[ 1 ] = "DMC";    //OK
        arr[ 2 ] = new Integer(123);    //Not OK : ArrayStoreException
    }
}
```

Sorting Array Elements

```
public class Program {
    public static void main(String[] args) {
        int[] arr = new int[] { 50, 10, 40, 20, 30 };
        System.out.println(Arrays.toString(arr));
        Arrays.sort(arr);   //The sorting algorithm is a Dual-Pivot Quicksort
        System.out.println(Arrays.toString(arr));
    }
}
```

Reference Copy and Instance Copy

Array Reference copy

```
int[] arr1 = new int[   ] { 10, 20, 30, 40, 50 };
int[] arr2 = arr1;  //Reference Copy
```

Array Instance Copy(Using Arrays.copyOf())

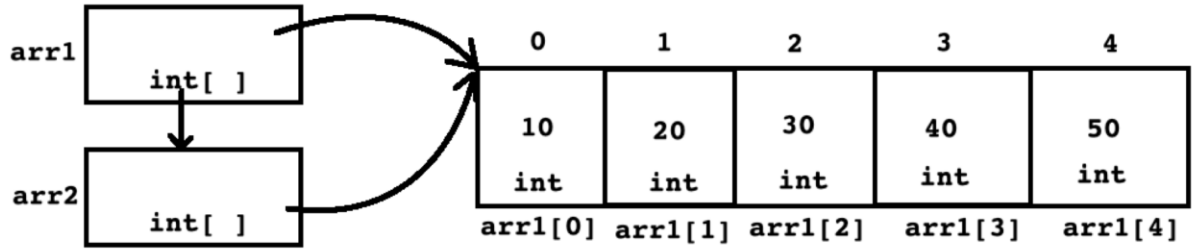
```
int[] arr1 = new int[   ] { 10, 20, 30, 40, 50 };
int[] arr2 = Arrays.copyOf(arr1, arr1.length);  //Array instance copy
```

Implementation of Arrays.copyOf method:

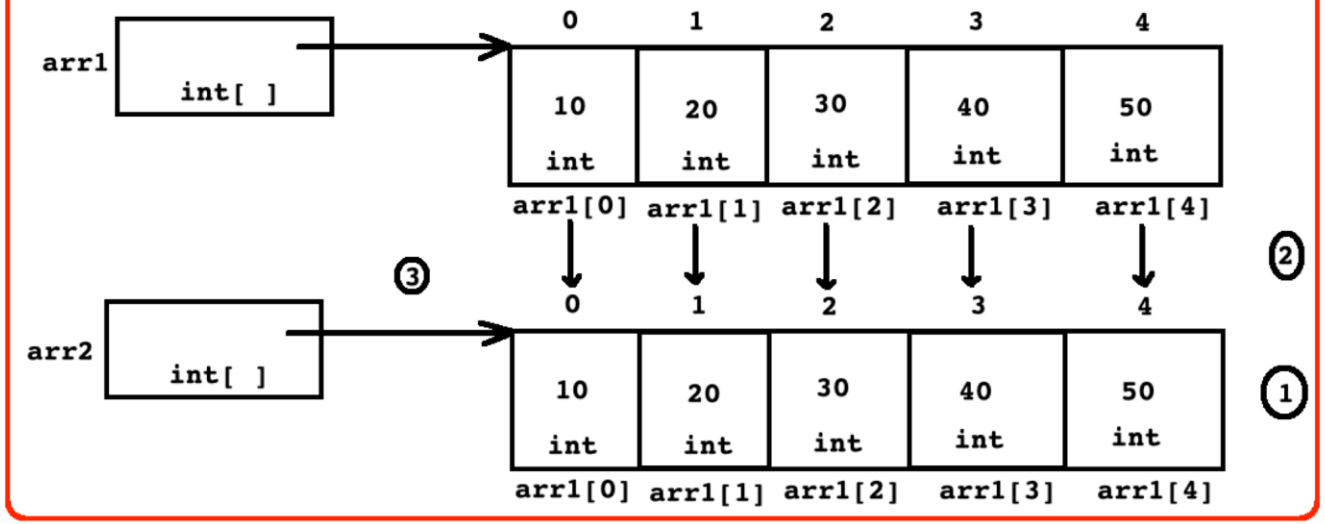
```
public static int[] copyOf(int[] original, int newLength) {
    int[] copy = new int[newLength];
    //public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length);
    System.arraycopy(original, 0, copy, 0, Math.min(original.length, newLength));
    return copy;
}
```

Reference Copy and Instance Copy

```
int[] arr1 = new int[  ] { 10, 20, 30, 40, 50 };
int[] arr2 = arr1;  //Reference Copy
```




```
int[] arr1 = new int[  ] { 10, 20, 30, 40, 50 };
int[] arr2 = Arrays.copyOf(arr1, arr1.length);  //Array instance copy
```

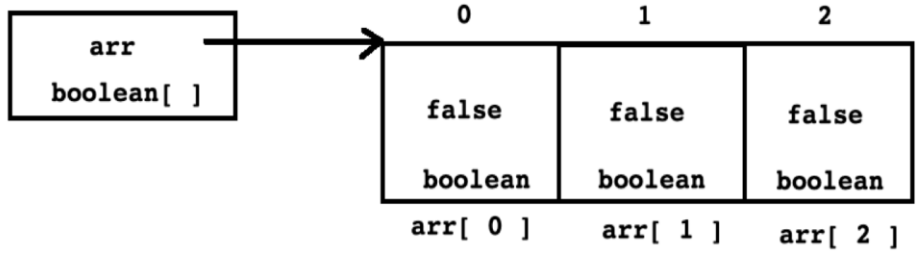


Array Of Primitive Values

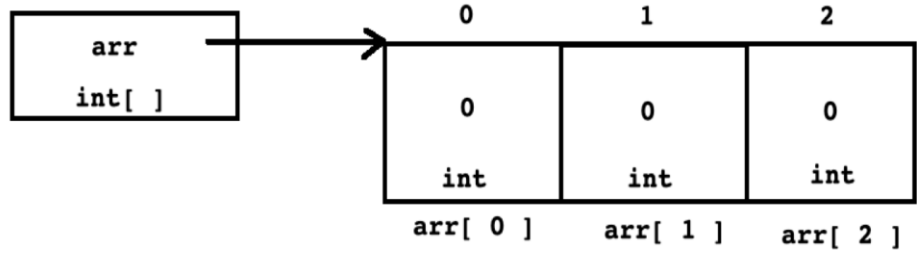
```
public class Program {
    public static void main(String[] args) {
        boolean[] arr = new boolean[ 3 ];   //contains all false
        int[] arr = new int[ 3 ];   //contains all 0
        double[] arr = new double[ 3 ]; //contains all 0.0
    }
}
```

Array Of Primitive Values

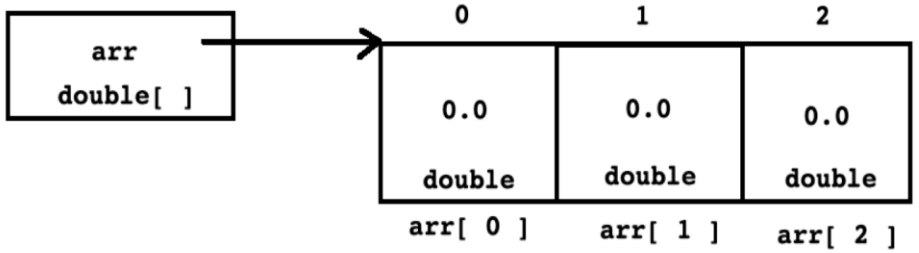
```
boolean[] arr = new boolean[3];
```




```
int[] arr = new int[3];
```




```
double[] arr = new double[3];
```



If we create array of primitive values then it's default value depends of default value of data type.

Array Of References

```
public class Program {
    public static void main(String[] args) {
        Date[] arr = new Date[ 3 ]; //Contains all null
    }
}
```

Array Of References and Instances

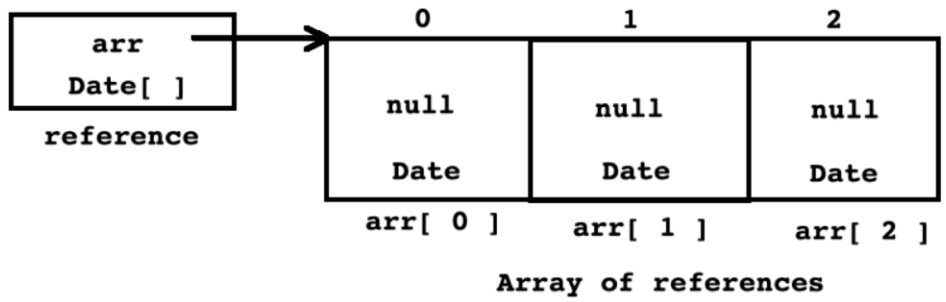
```
public class Program {
    public static void main(String[] args) {
        Date[] arr = new Date[ 3 ]; //Contains all null
    }
}
```

- Let us see how to create array of instances of non primitive type

```
public class Program {
    public static void main(String[] args) {
        Date[] arr = new Date[ 3 ];
        arr[ 0 ] = new Date( );
        arr[ 1 ] = new Date( );
        arr[ 2 ] = new Date( );
    }
    //or
    public static void main(String[] args) {
        Date[] arr = new Date[ 3 ];
        for( int index = 0; index < arr.length; ++ index )
        arr[ index ] = new Date( );
    }
}
```

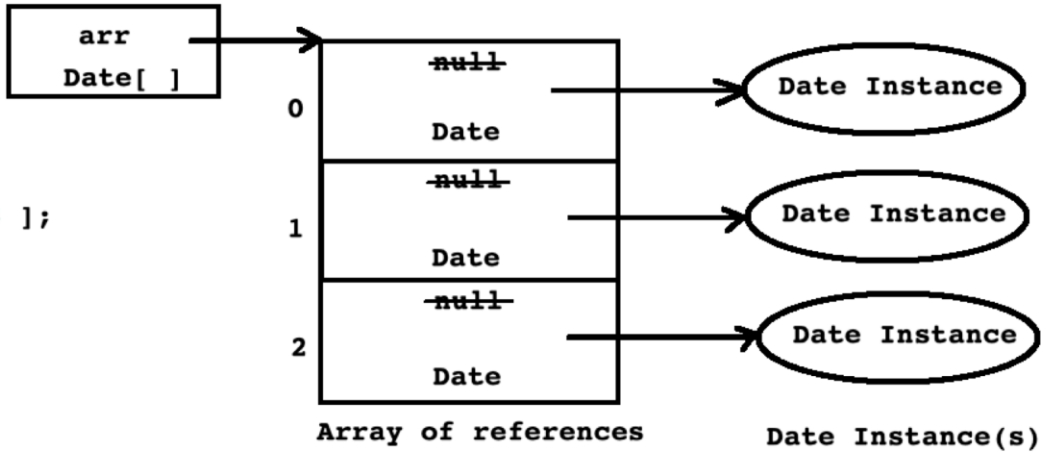
Array Of References and Instances

```
Date[ ] arr = new Date[ 3 ];
```



If we create an array of references then by default it contains null.

```
Date[] arr = new Date[ 3 ];
arr[ 0 ] = new Date( );
arr[ 1 ] = new Date( );
arr[ 2 ] = new Date( );
```



[Array Of Instances]

Multi Dimensional Array

- Array of elements where each element is array of same column size is called as multi dimensional array.

Reference declaration:

```
int arr[ ][ ];   //OK
int [ ]arr[ ]    //OK
int[ ][ ] arr;   //OK
```

Array Creation:

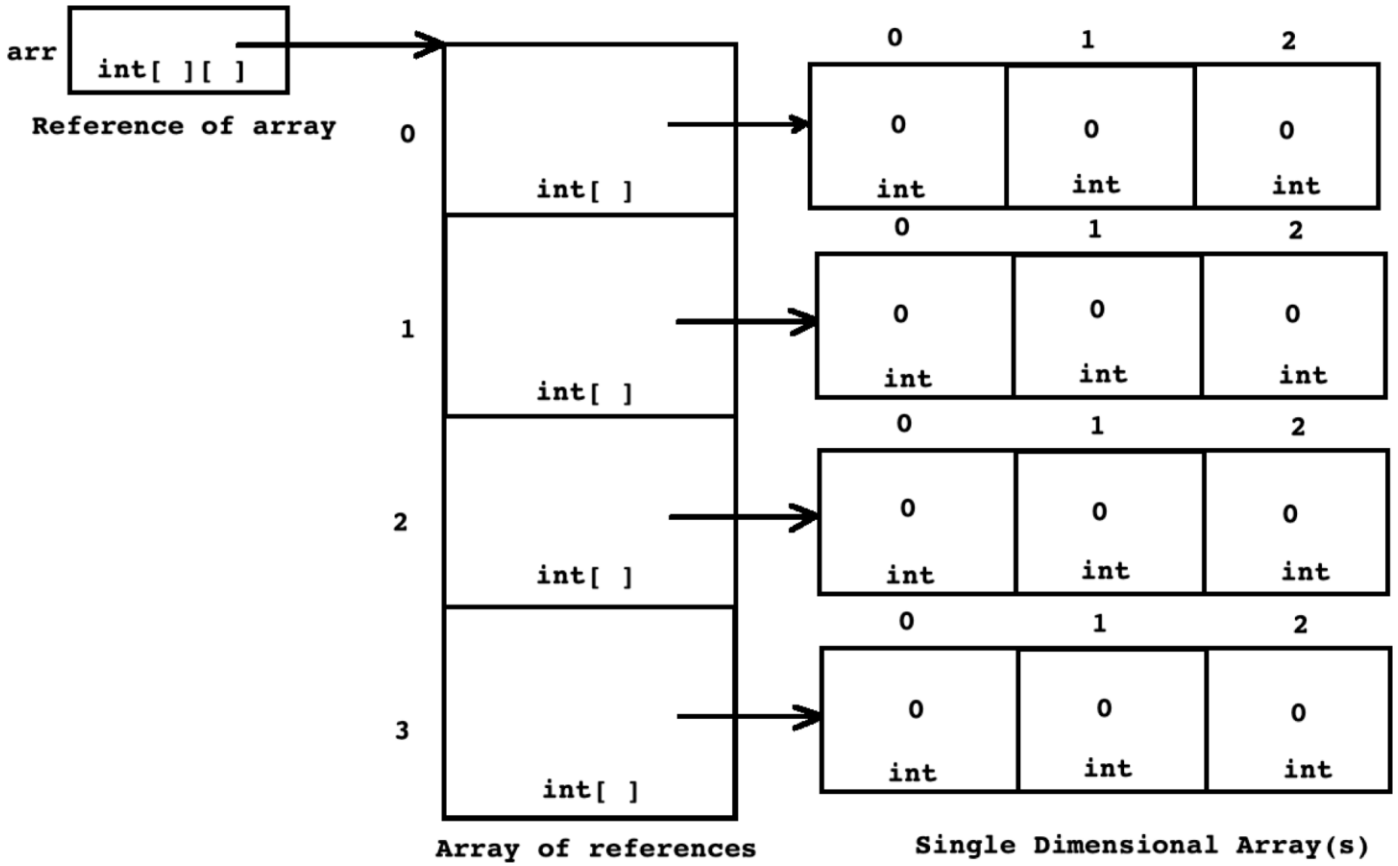
```
int[][] arr = new int[ 2 ][ 3 ];
```

Initialization

```
int[][] arr = new int[ ][ ]{{10,20,30},{40,50,60}}; //OK
int[][] arr = { {10,20,30}, {40,50,60} };    //OK
```

Multi Dimensional Array

+ Multi Dimensional Array



Ragged Array

- A multidimensional array where column size of every array is different.

Reference declaration

```
int arr[][];
int []arr[];
int[][] arr;
```

Array creation

```
int[][] arr = new int[3][];
arr[ 0 ] = new int[ 2 ];
arr[ 1 ] = new int[ 3 ];
arr[ 2 ] = new int[ 5 ];
```

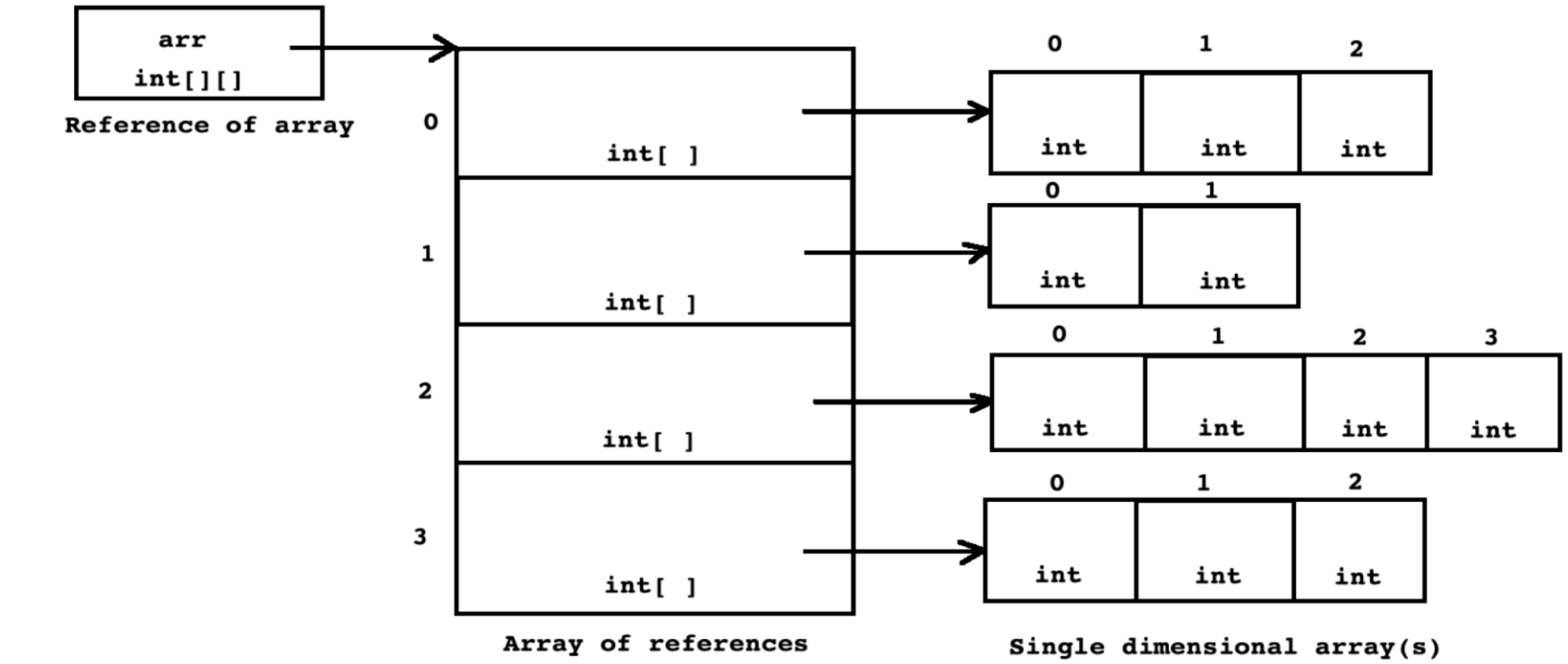
Array Initialization

```
int[][] arr = new int[3][];
arr[ 0 ] = new int[  ]{ 10, 20 };
arr[ 1 ] = new int[  ]{ 10, 20, 30 };
arr[ 2 ] = new int[  ]{ 10, 20, 30, 40, 50 };
```

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

Ragged Array

+ Ragged Array



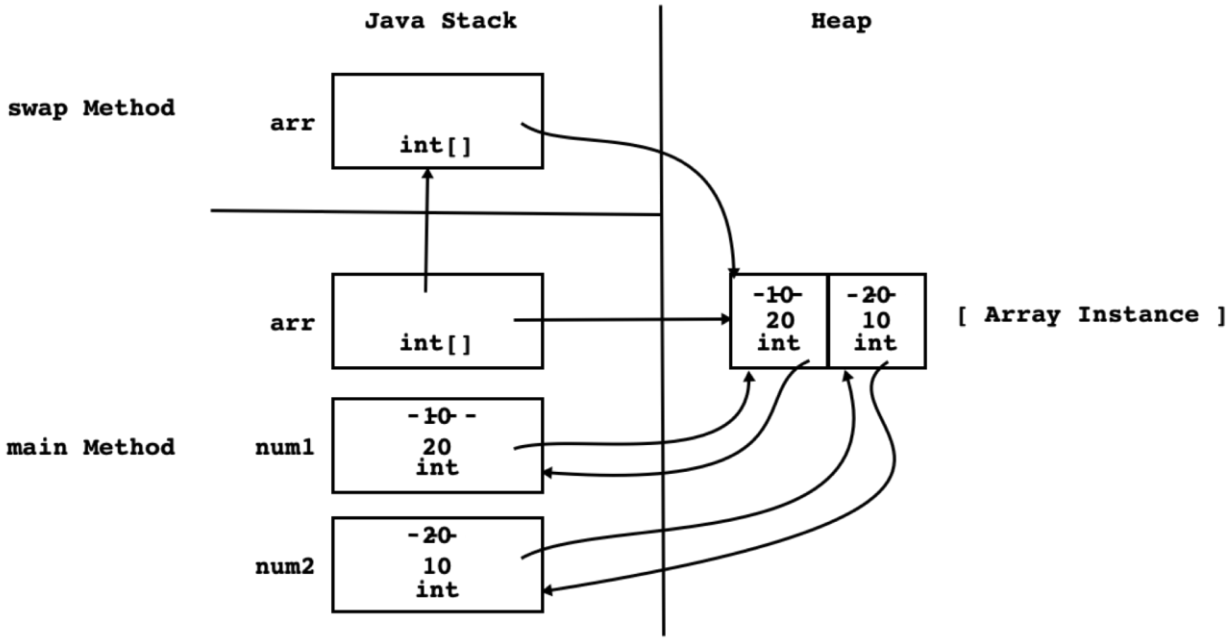
Argument Passing Methods

- In C programming language, we can pass argument to the function using 2 ways:
  1. By value.
  2. By address
- In C++ programming language, we can pass argument to the function using 3 ways:
  1. By value.
  2. By address
  3. By reference
- In Java programming language, we can pass argument to the method using a way:
  1. By value.
     - In other word, every variable of primitive type/non primitive type is pass to the method by value only.

Simulation Of Pass By Reference in Java

```
public class Program {
    private static void swap(int[] arr) {
        int temp = arr[0];
        arr[0] = arr[1];
        arr[1] = temp;
    }

    public static void main(String[] args) {
        int num1 = 10, num2 = 20;
        int[] arr = new int[] { num1, num2 };
        Program.swap(arr);  //passing arr as a argument to the method by value.
        num1 = arr[0];  num2 = arr[1];
        System.out.println("Num1    :    " + num1);  //20
        System.out.println("Num2    :    " + num2);  //10
    }
}
```

Simulation Of Pass By Reference in Java



Variable Arity/Argument Method

```
private static sum( int... arguments ){
    int result = 0;
    for( int element : arguments )
        result = result + element;
    return result;
}
public static void main(String[] args) {
    int result = 0;
    result = Program.sum( );     //OK
    result = Program.sum( 10, 20, 30 );     //OK
    result = Program.sum( 10, 20, 30, 40, 50  );     //OK
    result = Program.sum( 10, 20, 30, 40, 50, 60, 70, 80, 90, 100  );     //OK
}
```

- Consider Examples from Java API:
    1. public PrintStream printf(String format, Object... args);
    2. public static String format(String format, Object... args);
    3. public Object invoke(Object obj, Object... args);

Enum In C/C++ Programming language.

- According ANSI C standard, if we want to assign name to the integer constant then we should use enum.
- Enum helps developer to improve readability of source code.
- enum is keyword in C. Let us consider syntax of enum:

```
enum Identifier
{
    //enumerator-list
};
```

```
enum Color
{
    RED, GREEN, BLUE
    //RED = 0, GREEN = 1, BLUE = 2
};
```

Enum In C/C++ Programming language.

- By default, the first enumeration-constant is associated with the value 0. The next enumeration-constant in the list is associated with the value of (constant-expression + 1), unless you explicitly associate it with another value.

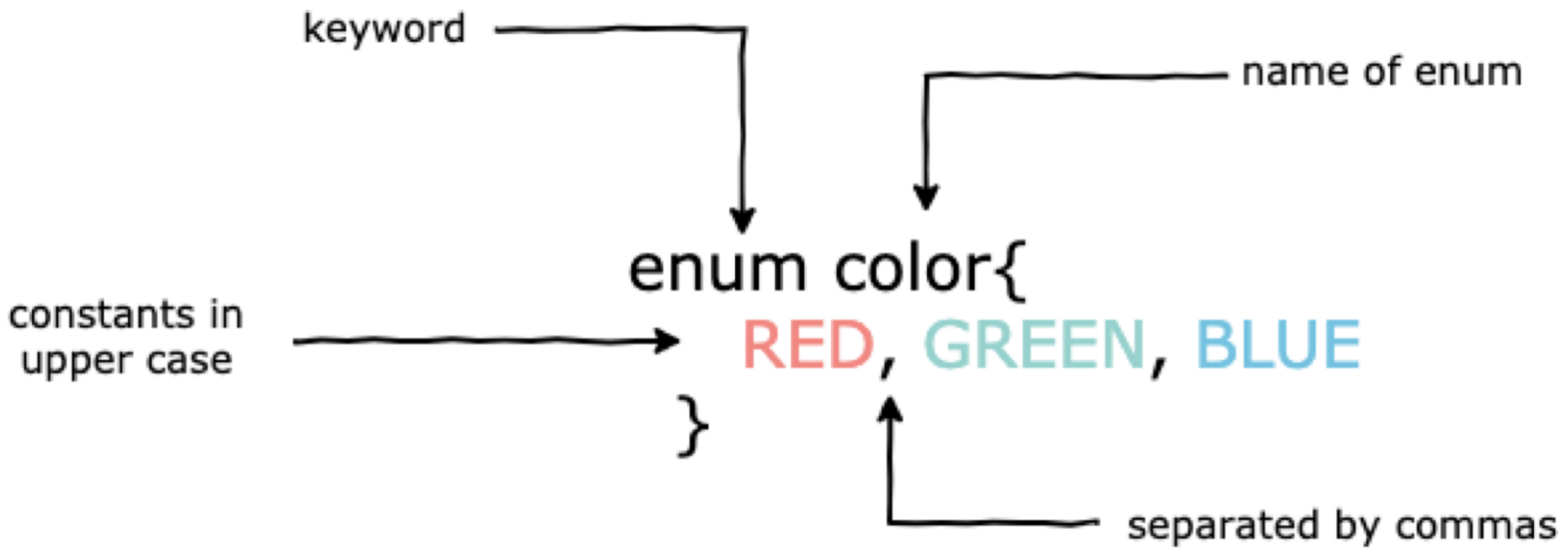
```
enum Channel
{
    FOX = 11,
    CNN = 25,
    ESPN = 15,
    HBO = 22,
    MAX = 30,
    NBC = 32
};
```

```
enum Suit { Diamonds = 1, Hearts, Clubs, Spades };
```

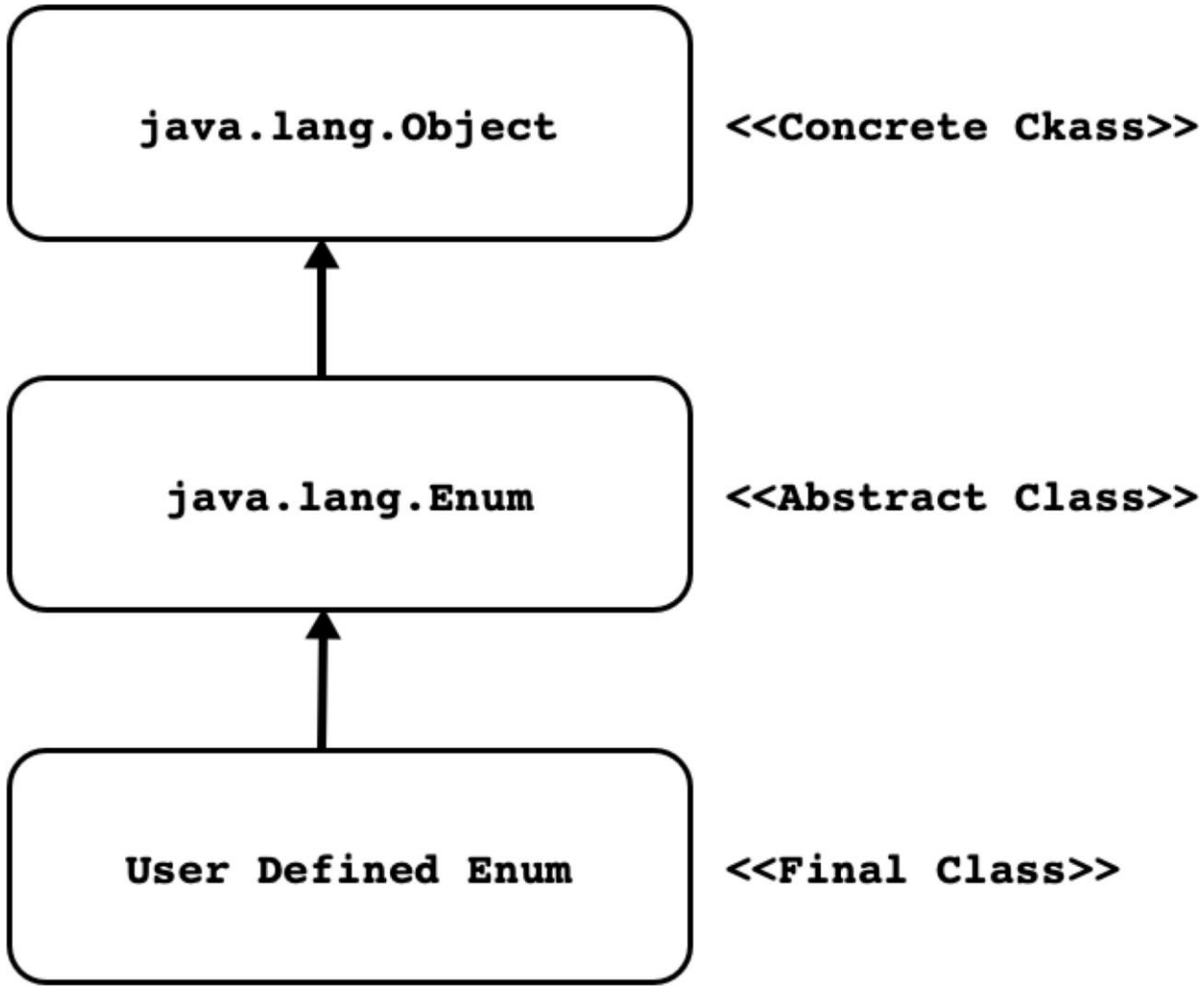
- constant-expression must have int type and can be negative.

Enum In Java Programming language.

- An enum is a class that represents a group of constants.
- **Enum keyword** is used to create an enum. The constants declared inside are separated by a comma and should be in upper case.



Enum Class Hierarchy



Enum API

- **Following are the methods declared in java.lang.Enum class:**

String	name()Returns the name of this enum constant, exactly as declared in its enum declaration.
int	ordinal()Returns the ordinal of this enumeration constant (its position in its enum declaration, where the initial constant is assigned an ordinal of zero).
String	toString()Returns the name of this enum constant, as contained in the declaration.
static <T extends Enum<T>> T	valueOf(Class<T> enumType, String name)Returns the enum constant of the specified enum type with the specified name.

Sole constructor : Programmers cannot invoke this constructor. It is for use by code emitted by the compiler in response to enum type declarations.

Enum for the compiler

Java Source Code

```
enum Color{
    RED, GREEN, BLUE
}
class Program{
    public static void main(String[] args) {
        Color color = Color.GREEN;
    }
}
```

Compiled Code

```
final class Color extends Enum<Color> {
    public static final Color RED;

    public static final Color GREEN;

    public static final Color BLUE;

    public static Color[] values();

    public static Color valueOf(String name);
}
```

Properties of enum

1. Similar to a class, an enum can have objects and methods. The only difference is that enum constants are public, static and final by default. Since it is final, we can't extend enums
2. It cannot extend other classes since it already extends the java.lang.Enum class.
3. It can implement interfaces.
4. The enum objects cannot be created explicitly and hence the enum constructor cannot be invoked directly.
5. It can only contain concrete methods and no abstract methods.

Application of enum

1. enum is used for values that are not going to change e.g. names of days, colors in a rainbow, number of cards in a deck etc.
2. enum is commonly used in switch statements and below is an example of it:

```
class Program {
    enum color {
        RED, GREEN, BLUE
    }
    public static void main(String[] args) {
        color x = color.GREEN; // storing value
        switch(x) {
        case RED:
            System.out.println("x has RED color");
            break;
        case GREEN:
            System.out.println("x has GREEN color");
            break;
        case BLUE:
            System.out.println("x has BLUE color");
            break;
        }
    }
}
```

Thank You.