OOP using Java

Trainer: Mr. Rohan Paramane

Static Field & its Initialization

- **Static Field**
  - Static field do not get space inside instance rather all the instances of same class share single copy of it.
  - Static Field is also called as class variable. It gets space once per class.
  - Static Field gets space once per class during class loading on method area .
  - Static Field can be accessed using object reference but it is designed to access using class name and dot operator.
- **Static Initializer block**
  - It is used to initialize the static fields
  - A static initialization block is a normal block of code enclosed in braces, { }, and preceded by the static keyword. Here is an example: static { // code to write }
  - A class can have any number of static initialization blocks, and they can appear anywhere in the class body.
  - The runtime system guarantees that static initialization blocks are called in the order that they appear in the source code.

Static Method

- To access non static members of the class, we should define non static method inside class.
- Non static method/instance method is designed to call on instance.
- To access static members of the class, we should define static method inside class.
- static method/class level method is designed to call on class name.
- static method do not get this reference:
- If we call, non static method on instance then method get this reference.
- Static method is designed to call on class name.
- Since static method is not designed to call on instance, it doesn't get this reference.

Static Import

- If static members belonging to the different class then use of type name and dot operator is mandatory.
- There are situations where you need frequent access to static final fields (constants) and static methods from one or two classes.
- Prefixing the name of these classes over and over can result in cluttered code.
- The static import statement gives you a way to import the constants and static methods that you want to use so that you do not need to prefix the name of their class.

Arrays

- Array is a sequential/linear container/collection which is used to store elements of same type in continuous memory location.
- If we want to access elements of array then we should use integer index.
- Array index always begins with 0.
- **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 job
4. Using assignment operator, we can not copy array into another array.
5. Compiler do not check array bounds(min and max index).

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. Multidimensional Array
  3. Ragged Array
- To perform operations on array we can use java.util.Arrays
- To display the array contents we can use the below ways
  - Use length field and for loop (arr.length)
  - Use Arrays.tostring(arr) method.
- Using illegal index, if we try to access elements of array then JVM throws ArrayIndexOutOfBoundsException.
- If we try to store incorrect type of object into array then JVM throws ArrayStoreException.
- If we try to negative value for array size then JVM throws NegativeArraySizeException.
- To sort the array we can use Arrays.sort(arr) method (sorting algorithm used is Dual-Pivot Quicksort)
- To copy the array we can use Arrays.copyOf(arr) method.

Single Dimensional Array

Reference declaration

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

Instantiation

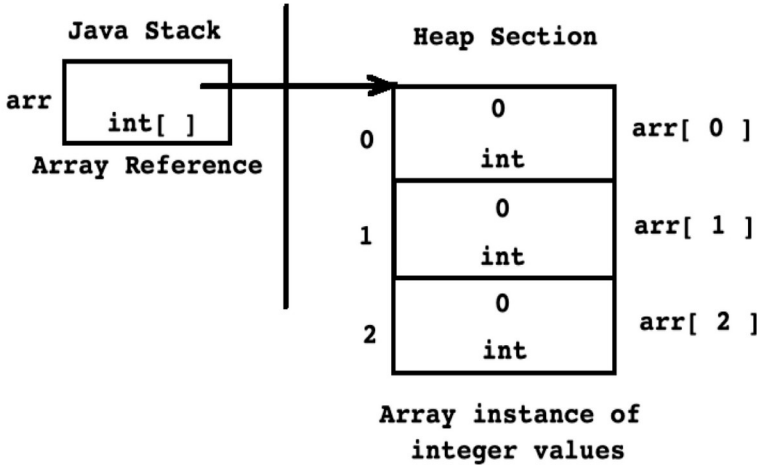
```
int[ ] arr1 = new int[ 3 ];
//or
int size = 3;
int[ ] arr2 = new int[ size ];
```

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

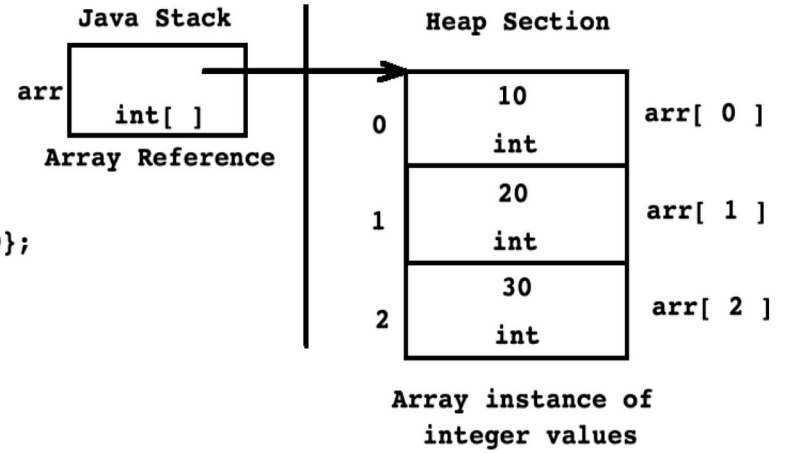
Initialization

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

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



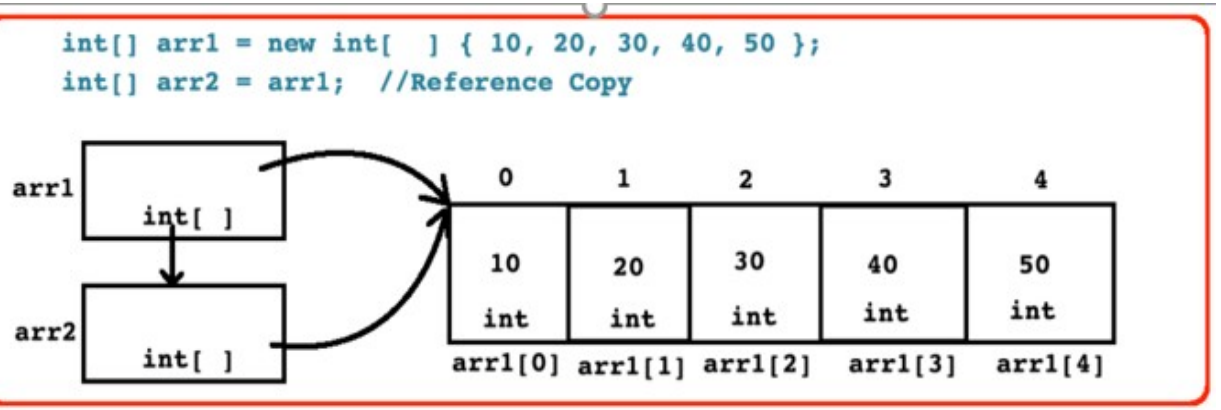

```
int[] arr = new int[]{10,20,30};
```



Reference Copy and Instance Copy

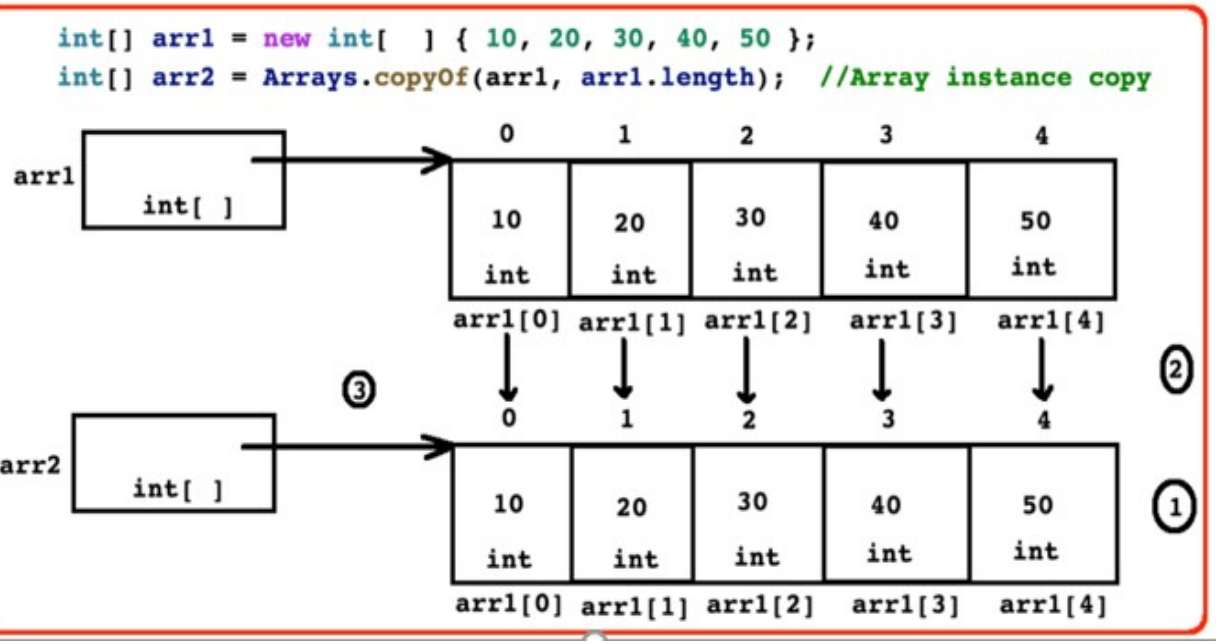
Array Reference copy

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



Array Instance Copy(Using Arrays.copyOf())

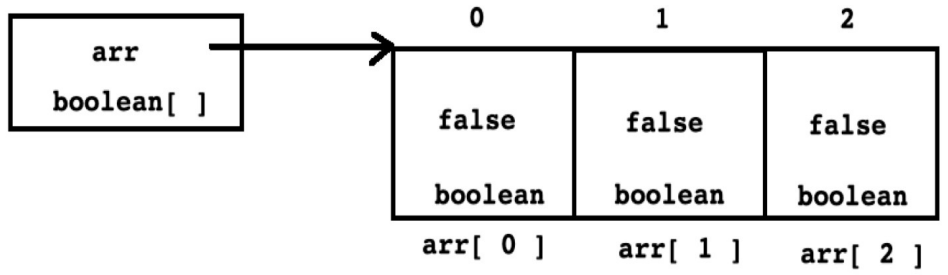
```
int[] arr1 = new int[  ] { 10, 20, 30, 40, 50 };
int[] arr2 = Arrays.copyOf(arr1, arr1.length);  //Array instance copy
```



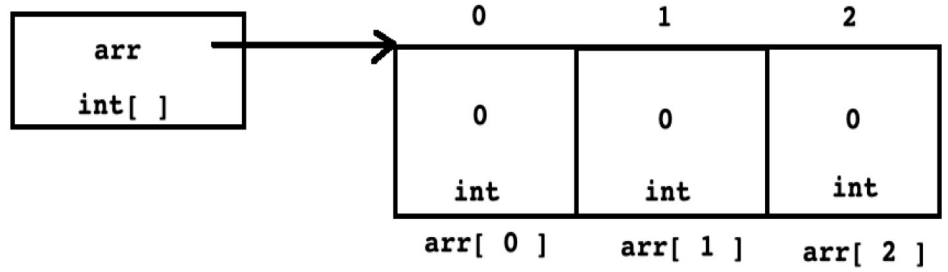
Array Of Primitive Values

```
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

    }

}
```

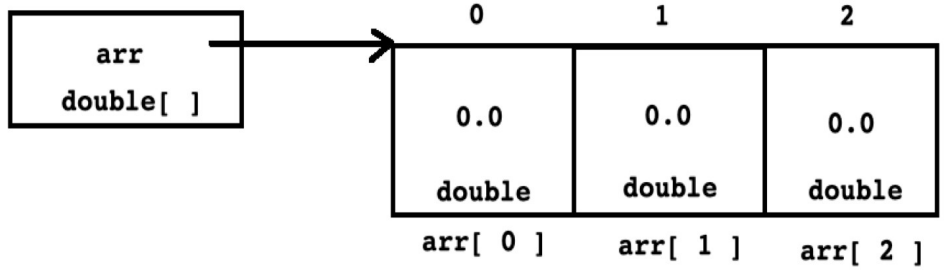
```
boolean[] arr = new boolean[3];
```




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



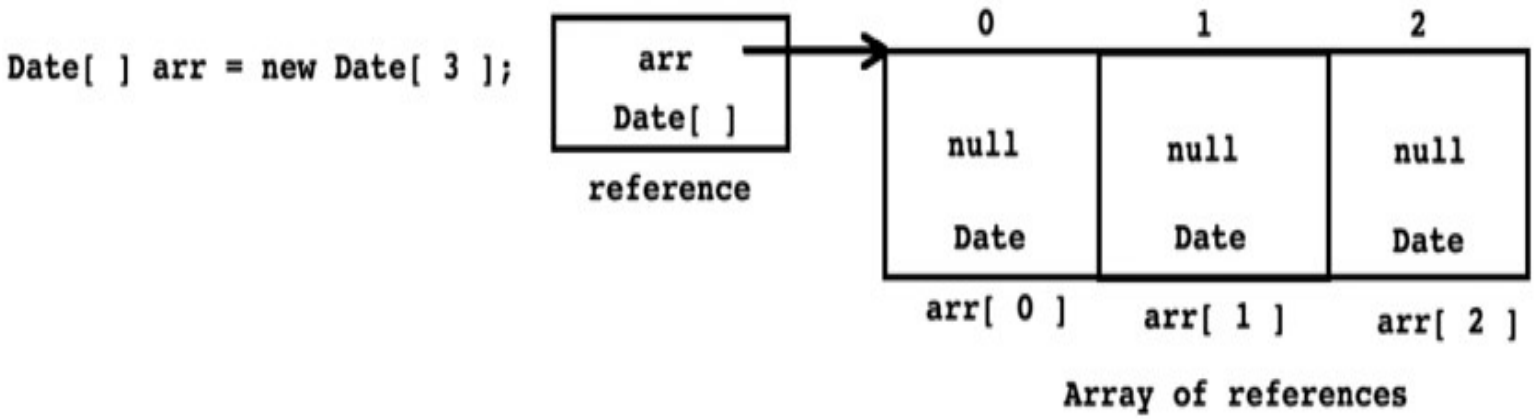

```
double[] arr = new double[3];
```



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

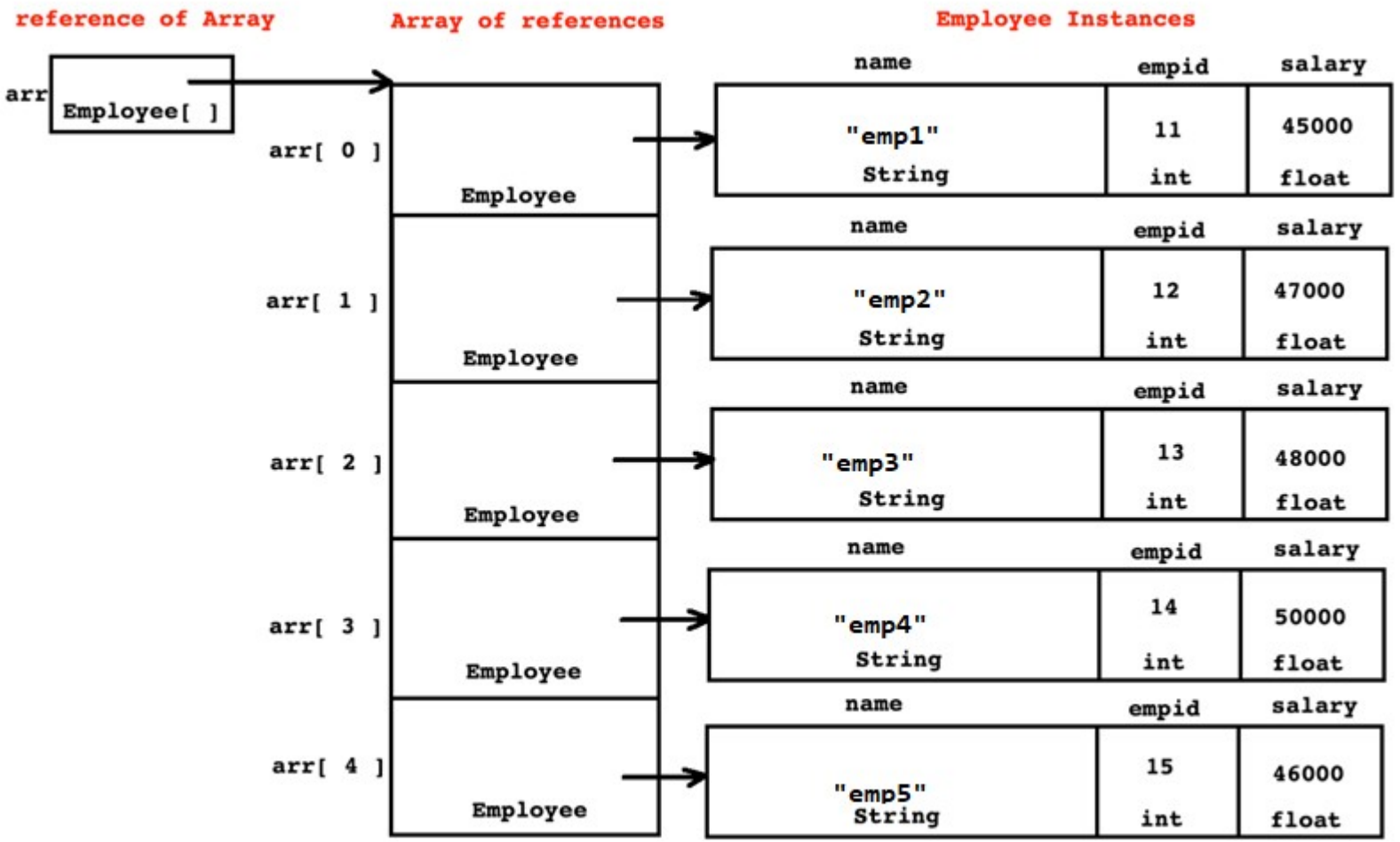
Array Of References

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



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

Array of reference and instance

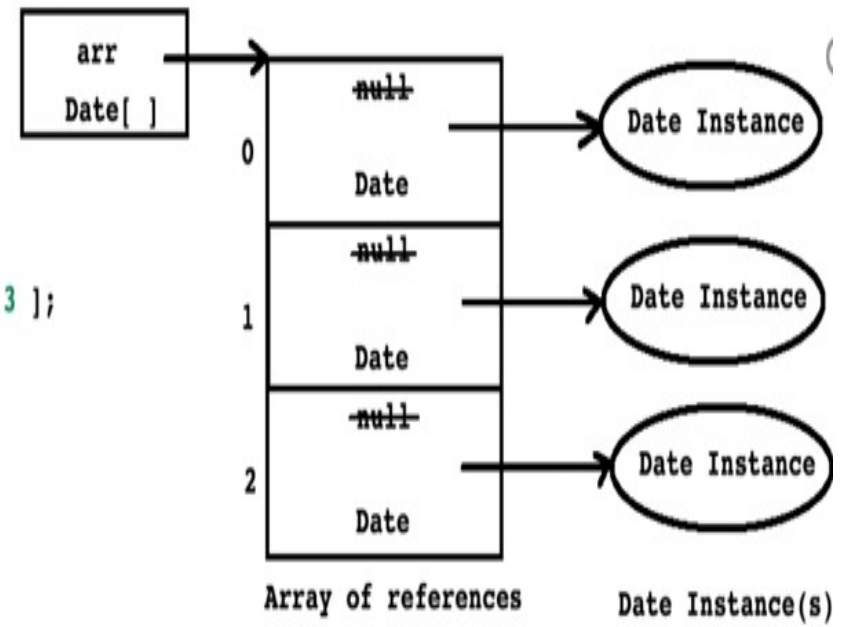


Array Of Instances

- 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( );
    }
}
```

```
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:

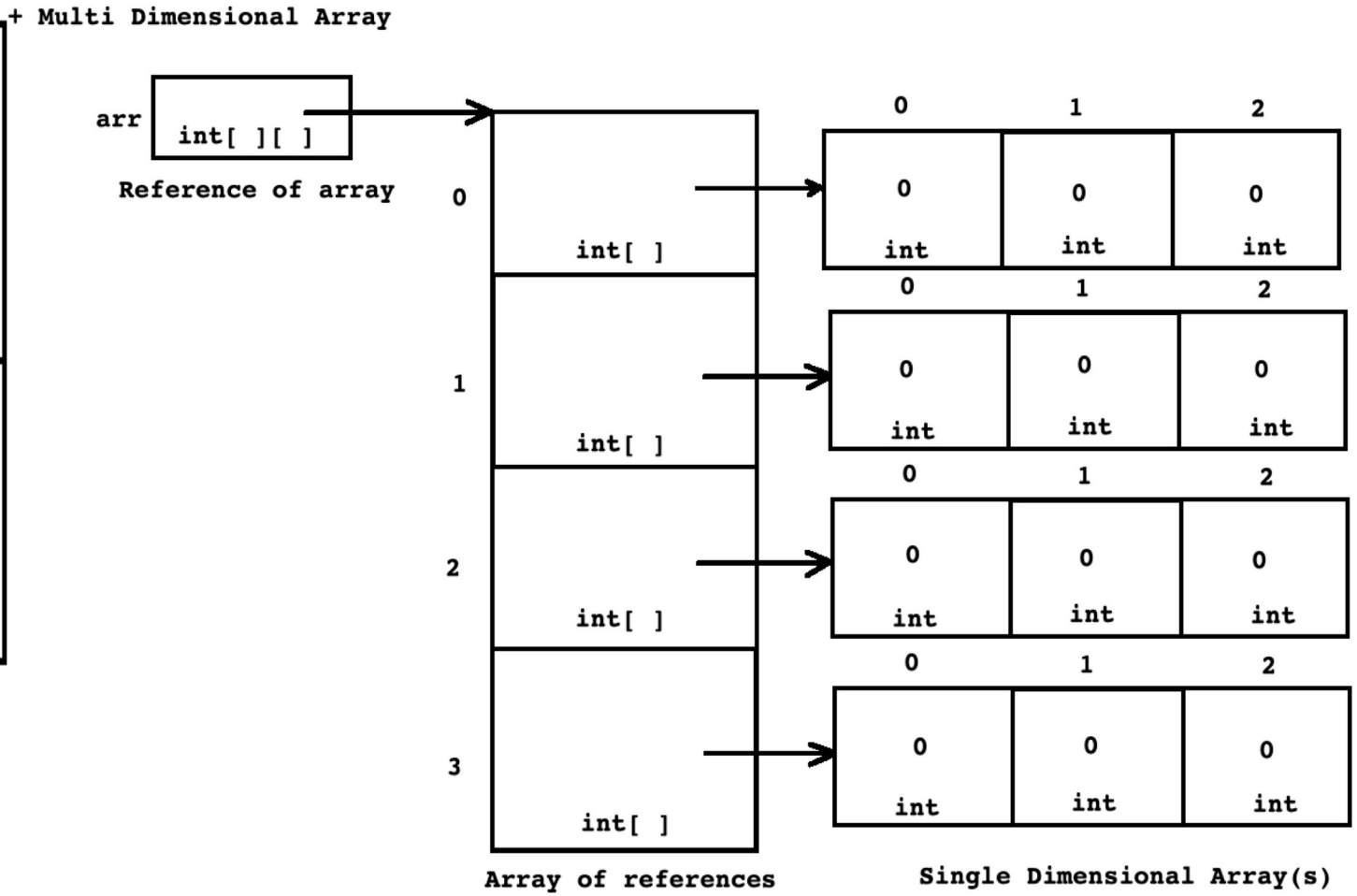
```
int arr[ ][ ];  //OK
int [ ]arr[ ]   //OK
int[ ][ ] arr;  //OK
```

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
```



Ragged Array

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

Reference declaration

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

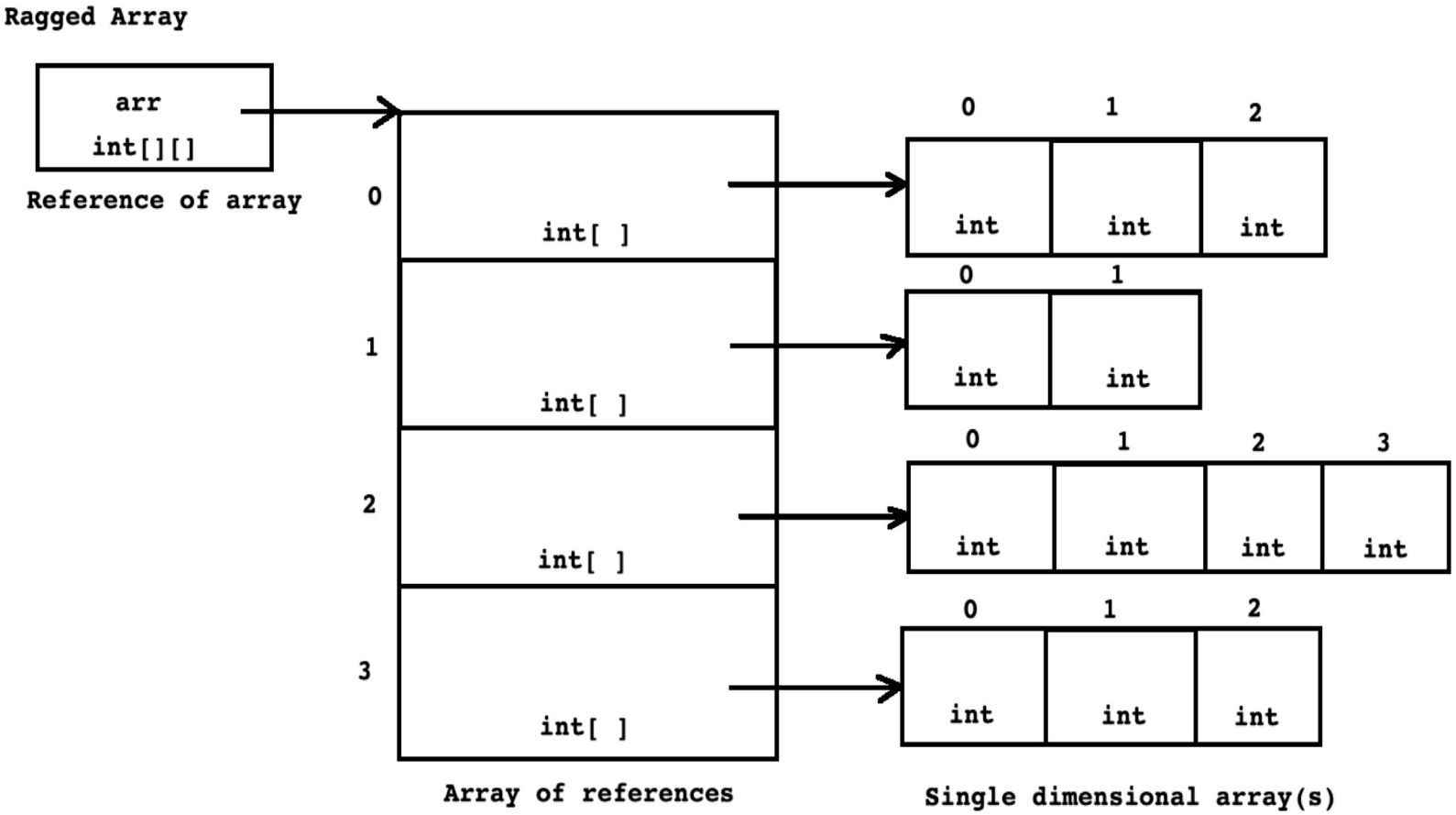
Array creation

```
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}};
```



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
}
```

Thank you!

Rohan Paramane

rohan.paramane@sunbeaminfo.com