OOP using Java

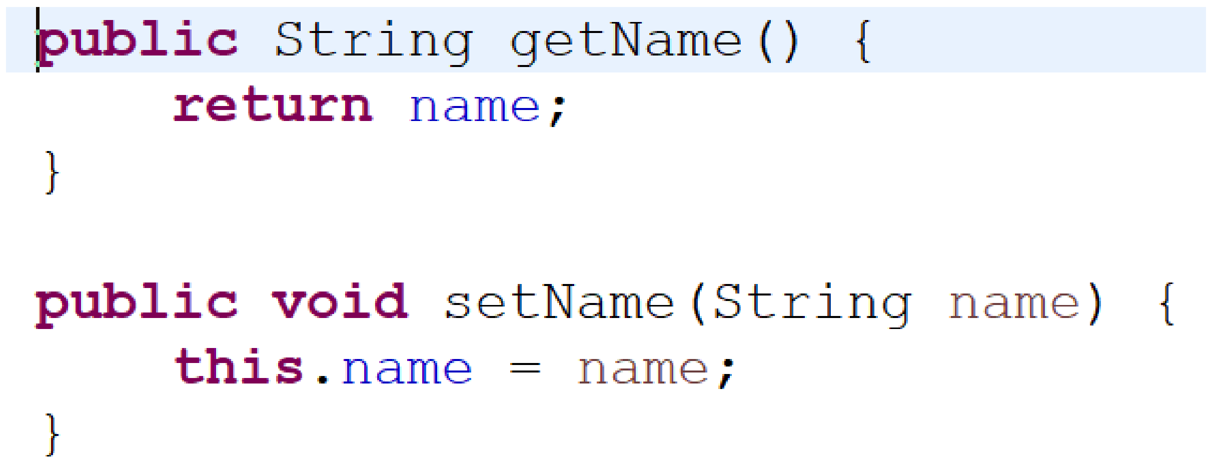
Trainer: Mr. Rohan Paramane

Agenda

- Setter Getter
- Literals
- Singleton
- Array
  - Single Dimensional Array

Setters and Getters

- To access private members of the class outside the class public methods should be used.
- If a value of single private field needs to be changed then the public method used for it is called as setter.
- If a value of single private field needs accessed then the public method used for it to access is called as getter.
- The syntax to write setter and getter is as below.



```
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}
```

Literals & null Literal

- Consider following literals in Java:
  - true : boolean
  - 'A' : char ch;
  - “Rohan" : String str;
  - 123 : int num1;
  - 72.93f : float num2
  - 3.142 : double num3
  - null : Used to initialize reference variable.

- null is a literal which is designed to initialize reference variable
  - int num = null ; //invalid
  - Integer num=null; // VALID
  - String str=null; // VALID
  - Employee emp=null; // VALID

Singleton Design Pattern

- The singleton design pattern allows only to create one instance of your class.
- This is achieved by making the constructor private.
- As the constructor becomes private its instance cannot be created outside the class.
- The instance is created inside the class only once and same instance is returned every time through the static getter method.
- Requirements for singleton design patter
  - Private constructor
  - Static field of same type as that of class
  - Static Getter method for the field to return its instance.

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

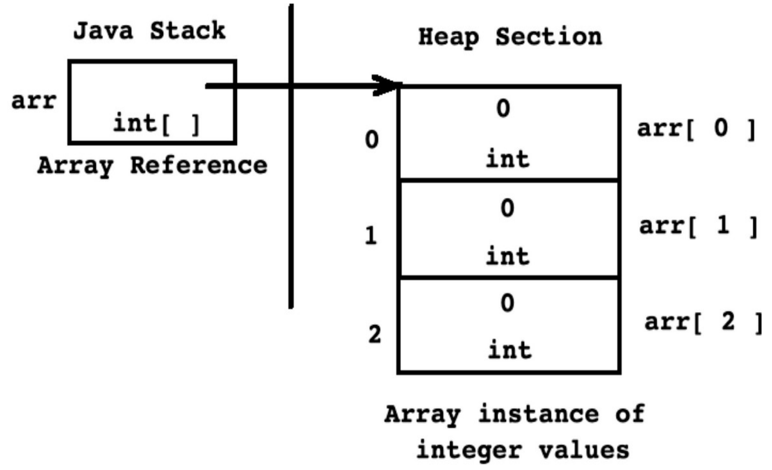
```
int[ ] arr1 = new int[ 3 ];
//or
int size = 3;
int[ ] arr2 = new int[ size ];
```

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

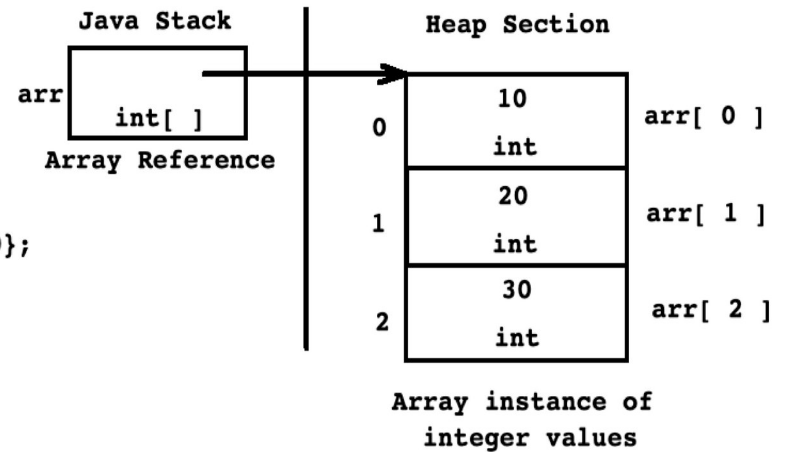
Initialization

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

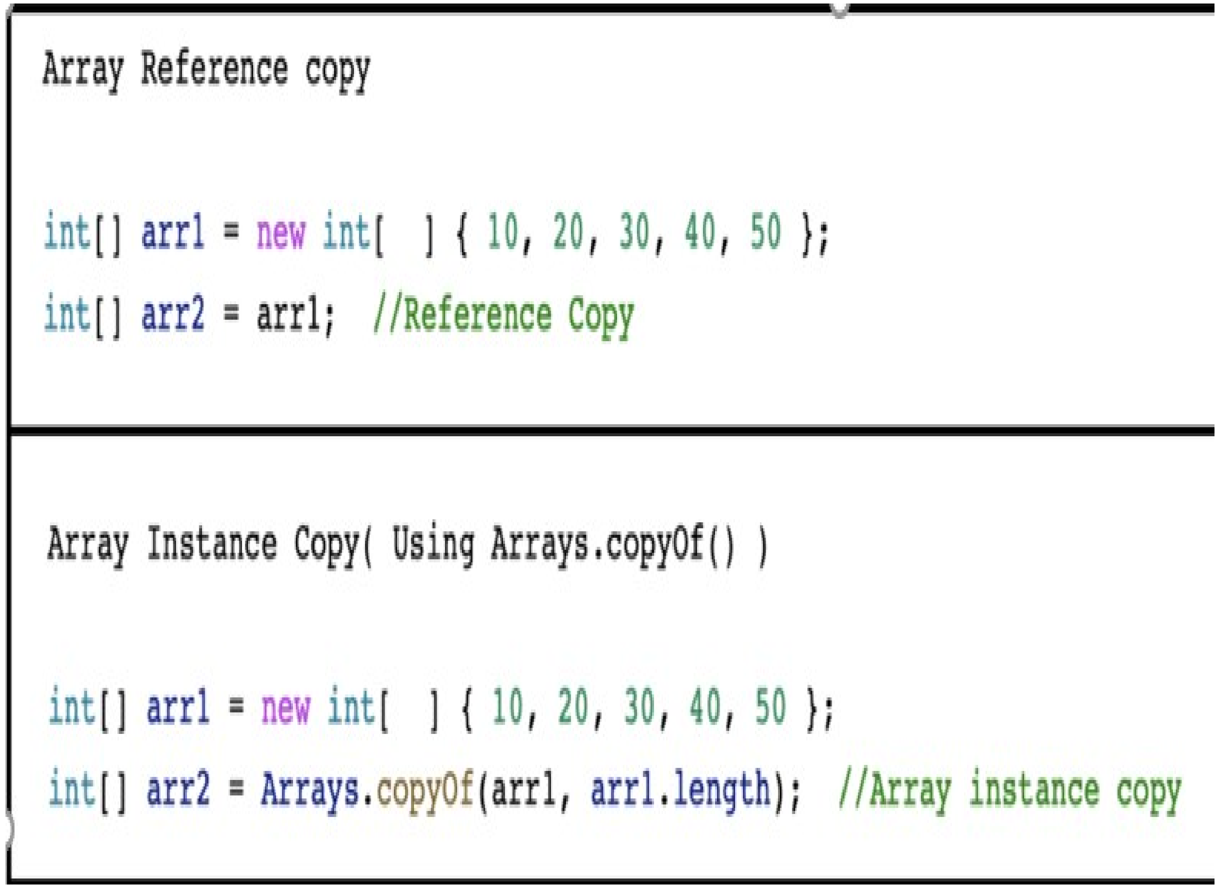
`int[] arr = new int[3];`



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



Reference Copy and Instance Copy

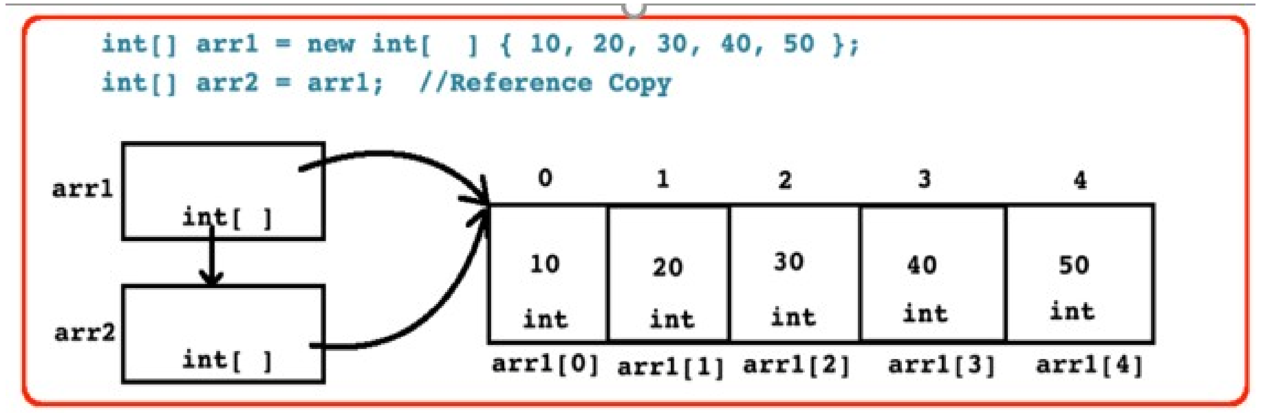


Array Reference copy

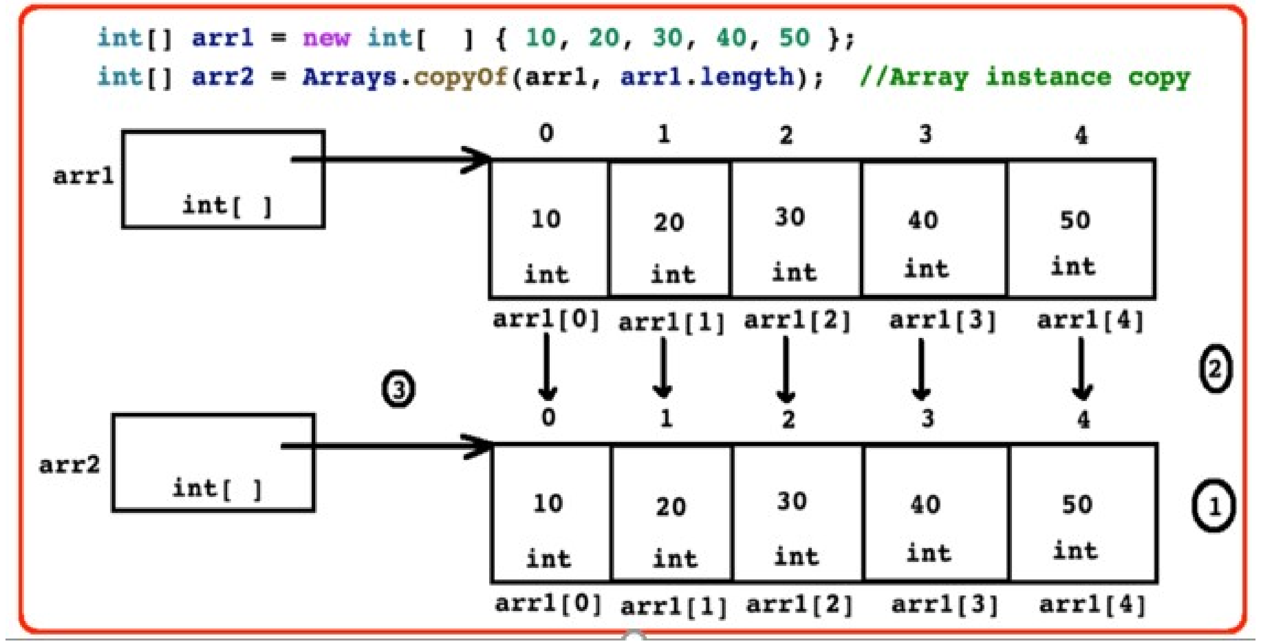
```
int[] arr1 = new int[  ] { 10, 20, 30, 40, 50 };
int[] arr2 = arr1;  //Reference Copy
```

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
```



```
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
```

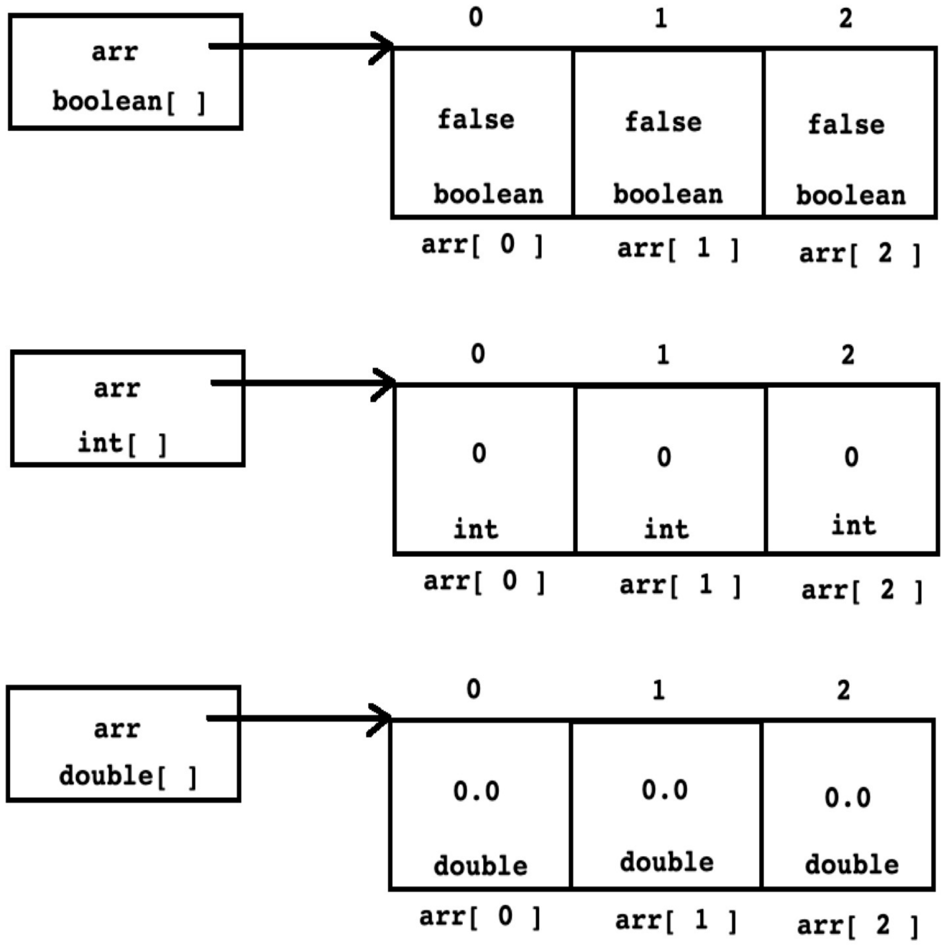
Array Of Primitive Values

```
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

    }

}
```

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

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

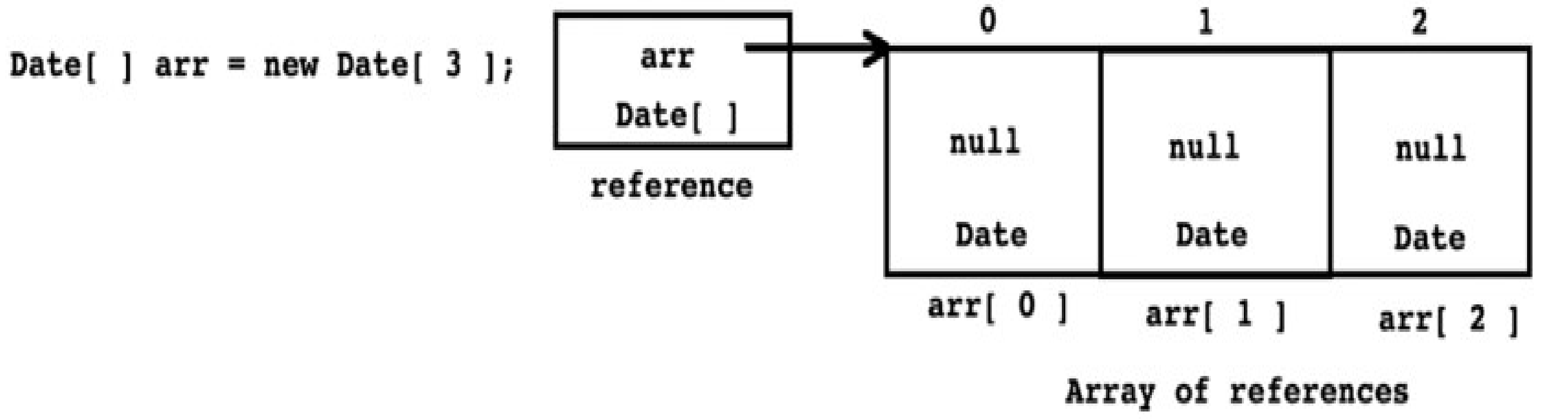
```
double[] arr = new double[3];
```



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

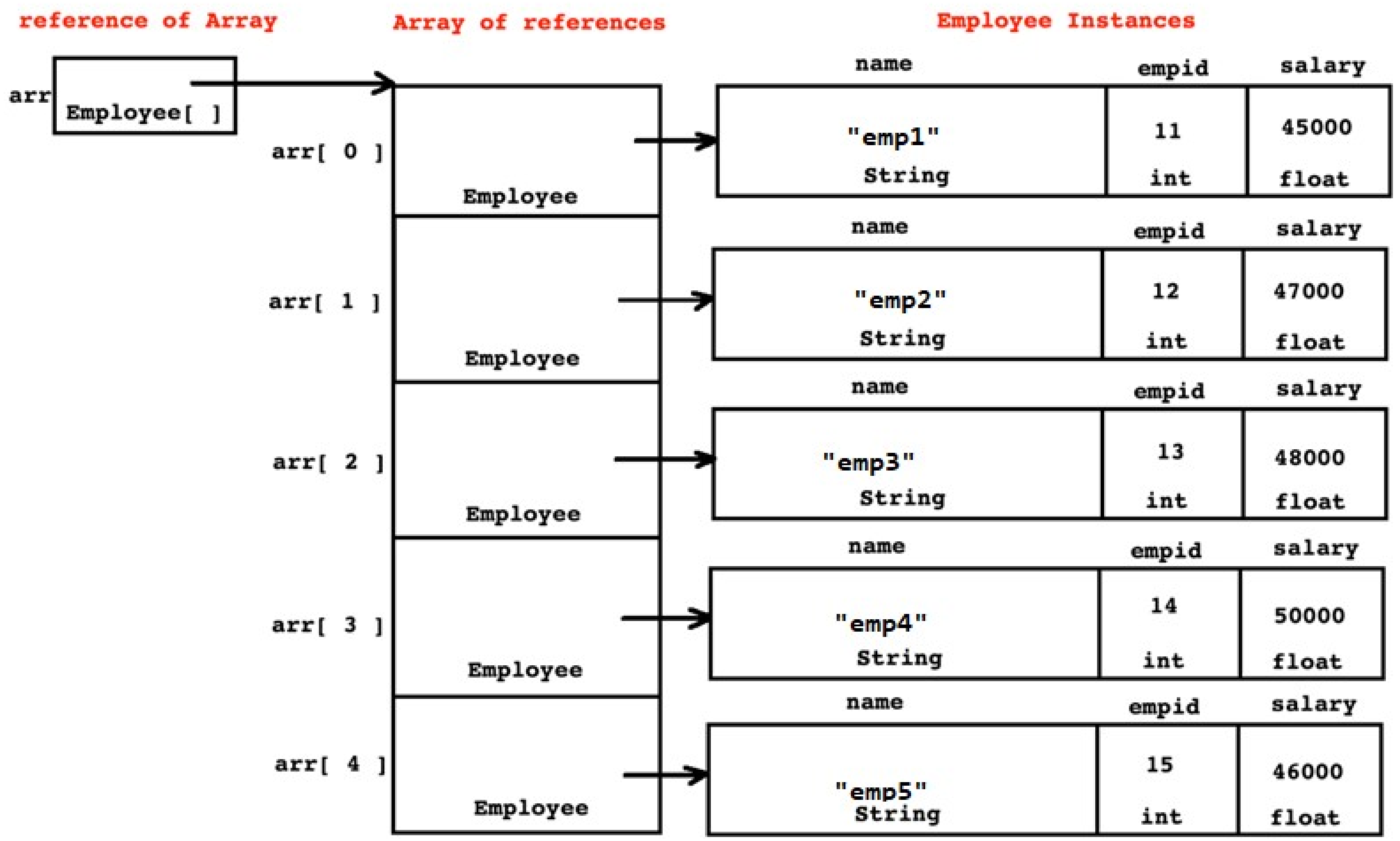
Array Of References

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

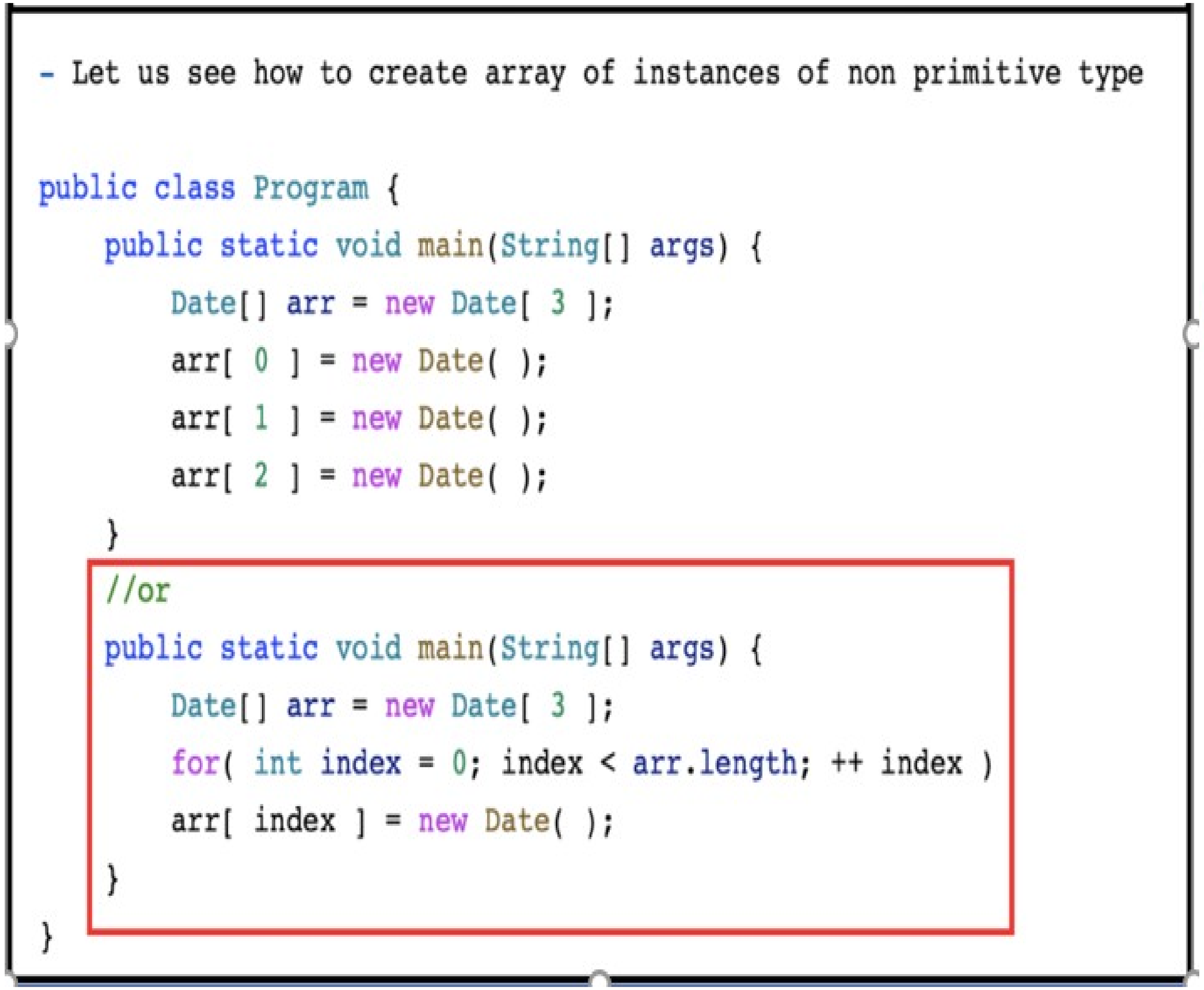


If we create an array of references then by default it contains null.

Array of reference and instance

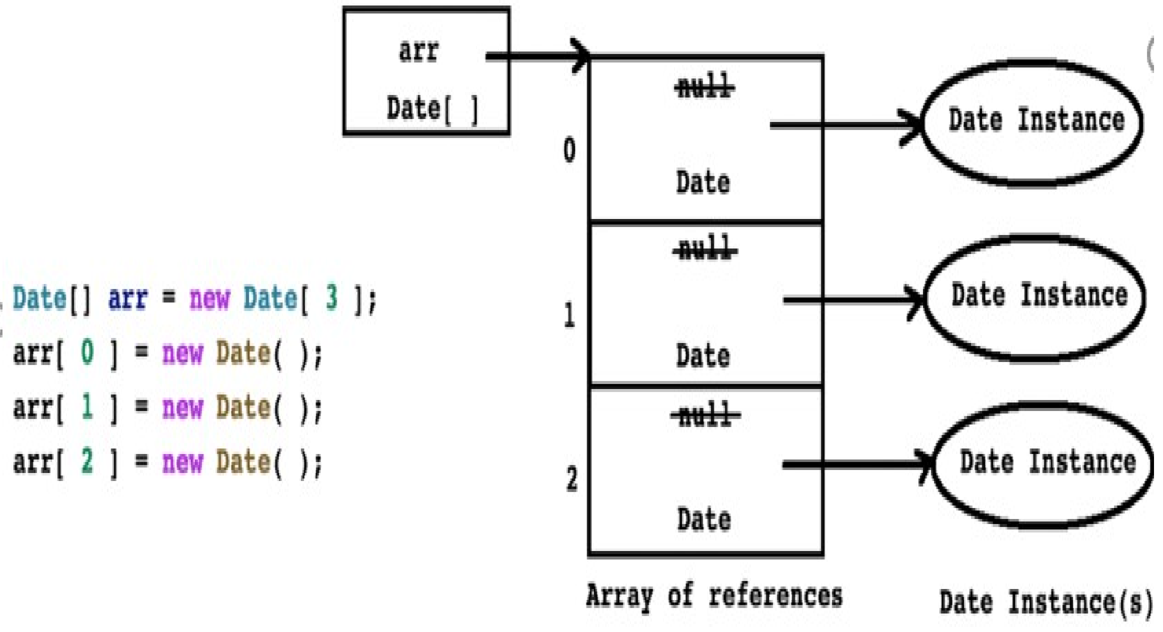


Array Of Instances



- 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]

Thank you!

Rohan Paramane

rohan.paramane@sunbeaminfo.com