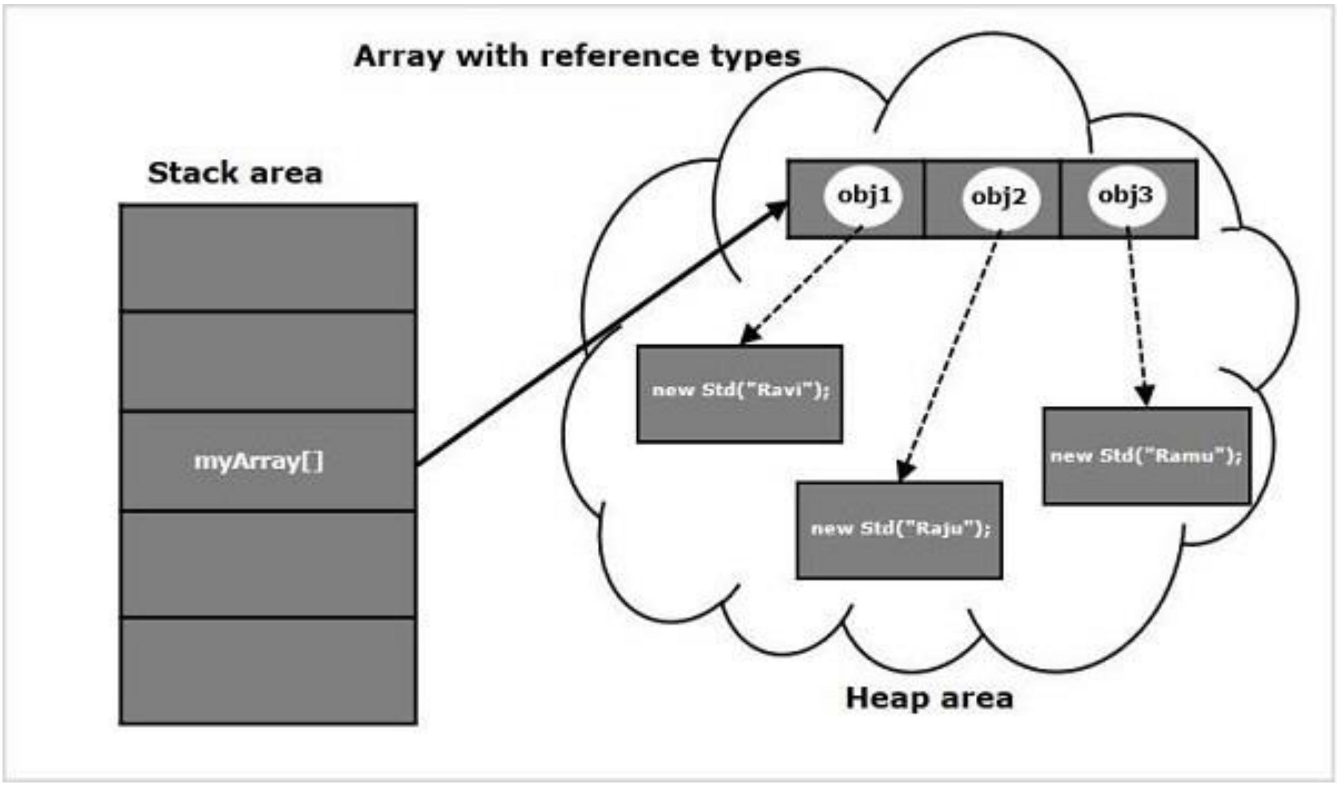
ARRAY STORAGE IN JAVA

The reference types in Java are stored in heap area. Since arrays are reference types (we can create them using the new keyword) these are also stored in heap area.

In addition to primitive datatypes arrays also store reference types: Another arrays (multi-dimensional), Objects. In this case the object array/multi-dimensional stores the references of the objects/arrays init, which points to the location of the objects/arrays.

```
class Std {
  private String name;
  public Std(String name){
    this.name = name;
  }
}
public class Sample {
  public static void main(String args[]) throws Exception {
    //Creating an array to store objects of type Std
    Std myArray[] = new Std[3];
    //Populating the array
    myArray [0] = new Std("Ravi");
    myArray [1] = new Std("Raju");
    myArray [2] = new Std("Ramu");
  }
}
```

The memory of the array myArray might be like –



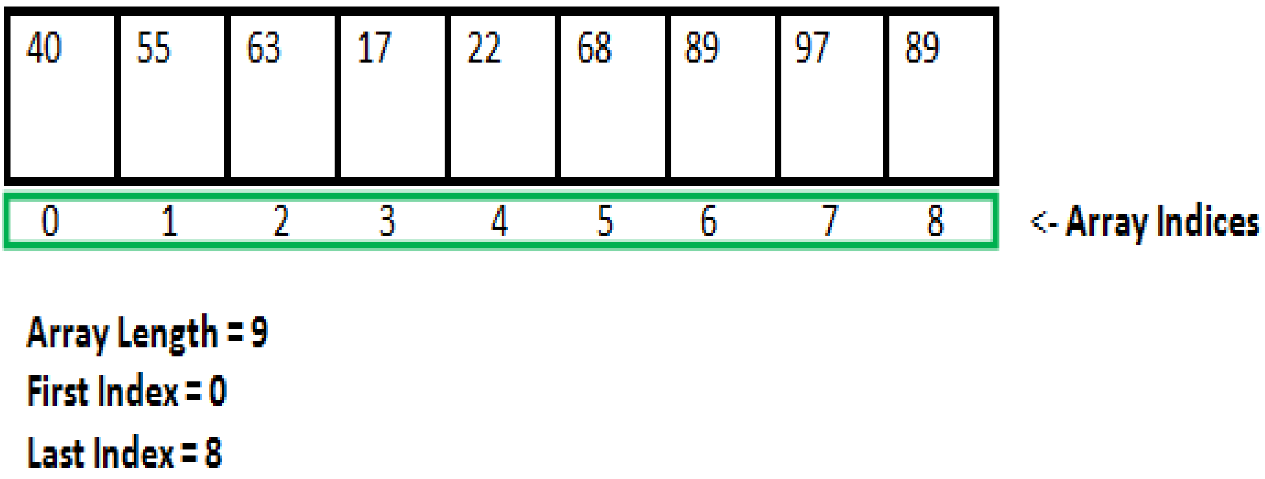
A stack and a heap are used for memory allocation in Java. However, the stack is used for primitive data types, temporary variables, object addresses etc. The heap is used for storing objects in memory.

STACK IN JAVA

Stacks are used to store temporary variables, primitive data types etc. A block in the stack exists for a variable only as long as the variable exists. After that, the block data is erased and it can be used for storing another variable.

ARRAY INDEXING IN JAVA

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on. As is clearly depicted below:



Array indexing has its advantages. For example, it:

- Makes the code optimized, we can retrieve or sort the data efficiently.
- Helps us get any data located at an index position.

ARRAY METHODS IN JAVA

The **Arrays** class in **java.util package** is a part of the **Java Collection Framework**. This class provides static methods to dynamically create and access **Java arrays**. It consists of only static methods and the methods of Object class. The methods of this class can be used by the class name itself.

Here Arrays class provides several static methods that can be used to perform these tasks directly without the use of loops, hence forth making our code super short and optimized.

Methods in Java Array Class

The Arrays class of the java.util package contains several static methods that can be used to fill, sort, search, etc in arrays. Now let us discuss the methods of this class.

1. asList()	Returns a fixed-size list backed by the specified Arrays

2. binarySearch()	Searches for the specified element in the array with the help of the Binary Search Algorithm
3. binarySearch(array, fromIndex, toIndex, key, Comparator)	Searches a range of the specified array for the specified object using the Binary Search Algorithm
4. compare(array 1, array 2)	Compares two arrays passed as parameters lexicographically.
5. copyOf(originalArray, newLength)	Copies the specified array, truncating or padding with the default value (if necessary) so the copy has the specified length.
6. copyOfRange(originalArray, fromIndex, endIndex)	Copies the specified range of the specified array into a new Arrays.
7. deepEquals(Object[] a1, Object[] a2)	Returns true if the two specified arrays are deeply equal to one another.
8. deepHashCode(Object[] a)	Returns a hash code based on the “deep contents” of the specified Arrays.
9. deepToString(Object[] a)	Returns a string representation of the “deep contents” of the specified Arrays.
10. equals(array1, array2)	Checks if both the arrays are equal or not.
11. fill(originalArray, fillValue)	Assigns this fill value to each index of this arrays.
12. hashCode(originalArray)	Returns an integer hashCode of this array instance.

13. mismatch(array1, array2)	Finds and returns the index of the first unmatched element between the two specified arrays.
14. parallelPrefix(originalArray, fromIndex, endIndex, functionalOperator)	Performs parallelPrefix for the given range of the array with the specified functional operator.
15. parallelPrefix(originalArray, operator)	Performs parallelPrefix for complete array with the specified functional operator.
16. parallelSetAll(originalArray, functionalGenerator)	Sets all the elements of this array in parallel, using the provided generator function.
17. setAll(originalArray, functionalGenerator)	Sets all the elements of the specified array using the generator function provided.
18. spliterator(originalArray)	Returns a Spliterator covering all of the specified Arrays.
19. spliterator(originalArray, fromIndex, endIndex)	Returns a Spliterator of the type of the array covering the specified range of the specified arrays.