LAB # 04

ARRAYS IN JAVA

OBJECTIVE: To understand arrays and its memory allocation.

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location.

- It is a data structure where we store similar elements.
- We can store only a fixed set of elements in a Java array.
- 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.

Example

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

```
//Java Program to illustrate how to declare, instantiate, initialize
//and traverse the Java array.
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Where is the memory allocated for Arrays in Java?

Each time an array is declared in the program, contiguous memory is allocated to it.

The address of the first array element is called the array base address. Each element will occupy the memory space required to accommodate the values for its type, i.e.; depending on the data type of the elements, 1, 4, or 8 bytes of memory are allocated for each element.

The next memory address is assigned to the next element in the array. This memory allocation process continues until the number of array elements is exceeded.

First, we should understand a little bit of memory. Memory has two parts.

1. **Stack**
2. **Heap**

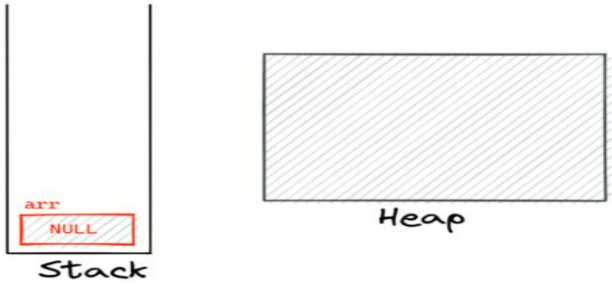
In Java, reference types are stored in the Heap area. As arrays are also reference types, they can be created using the "new" keyword they are also stored in the Heap area.

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

When we just write this

```
int arr[];
```

It only creates a variable in the stack and has value null by default.



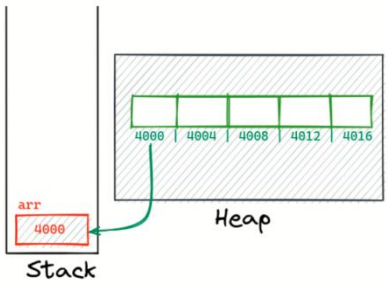
But after that when we allocate memory like this,

```
arr=new int[5];
```

Then to store 5 ints, a memory gets allocated in the heap. They will have their own address.

For the time being let's assume the addresses start from 4000. Now, these allocated memory units are contiguous, hence they will be spaced out by the same distance. So, the next memory location will have 4004 (assuming 4 bytes for int storage) and then 4008, 4012, and so on.

It doesn't stop there, now arr will store the address of the first memory block/unit among these contiguous memory blocks/units i.e. 4000.

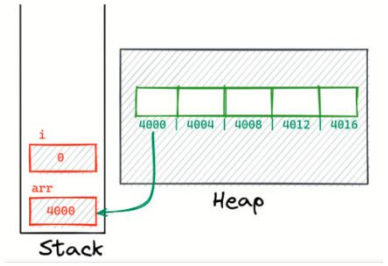


But now you might ask what happens if we do:

```
int i=0;
```

Here the memory will be allocated directly from the stack. The reason being these are primitive data types. An array on the other hand is an example of a non-primitive data type.

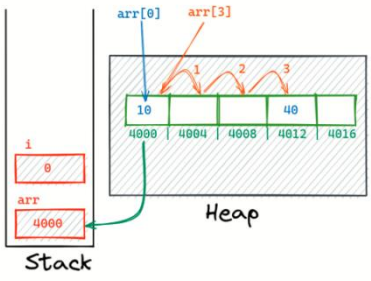
For primitive data types, both the name and value are stored in the stack. But for non-primitive, the name is stored in the stack, but the actual memory is allocated inside the heap, only the first address is stored under the name in the stack.



Now when we do operations like:

```
arr[0]=10;
arr[3]=40;
```

Java sees arr as 4000 because that is the value that is stored in the stack, now when we use the [3] operator Java looks for the 3rd position, i.e 3*4 = 12 bytes away from 4000 i.e 4012. So arr[3] represents the 4012 memory block.



Sample Program#1

```
class Main {
 public static void main(String[] args) {

   // create an array
   int[] age = {12, 4, 5, 2, 5};

   // access each array elements
   System.out.println("Accessing Elements of Array:");
   System.out.println("First Element: " + age[0]);
   System.out.println("Second Element: " + age[1]);
   System.out.println("Third Element: " + age[2]);
   System.out.println("Fourth Element: " + age[3]);
   System.out.println("Fifth Element: " + age[4]);
 }
}
```

Output:

```
Accessing Elements of Array:
First Element: 12
Second Element: 4
Third Element: 5
Fourth Element: 2
Fifth Element: 5
```

LAB TASKS

1. Write a program that takes two arrays of size 4 and swap the elements of those arrays.

2. Add a method in the class that takes array and merge it with the existing one.

3. In a JAVA program, take an array of type string and then check whether the strings are palindrome or not.

4. Given an array of integers, count how many numbers are even and how many are odd.

5. Given two integer arrays, merge them and remove any duplicate values from the resulting array.

HOME TASKS

1. Write a program that takes an array of Real numbers having size 7 and calculate the sum and mean of all the elements. Also depict the memory management of this task.

2. Add a method in the same class that splits the existing array into two. The method should search a key in array and if found splits the array from that index of the key.

3. Given an array of distinct integers and a target integer, return all unique combinations of numbers that add up to the target. Each number can be used only once in the combination.

4. You are given an array containing n distinct numbers taken from 0, 1, 2, ..., n. Write a program to find the one number that is missing from the array.

5. You are given an array of integers. Write a program to sort the array such that it follows a zigzag pattern: the first element is less than the second, the second is greater than the third, and so on.