Lesson:

1D Arrays

Pre-Requisites:

- Basic Java syntax

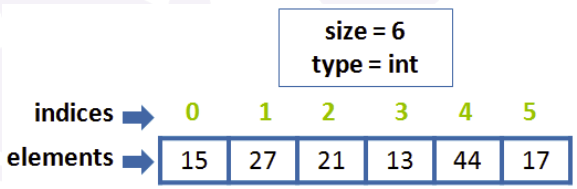
List of concepts involved:

- Introduction to Array Data Structure
- Array implementation
- Palindrome Array
- Reverse array
- Missing number in an array
- Duplicate numbers in an array

Introduction to Array Data Structure

- An array is defined as a collection of elements having the same data type.
- An array's primary purpose is to hold a group of the same sort of data. For instance, floating numbers, strings, integers, characters or even non- primitive (user-defined data types).
- Arrays are stored in contiguous memory (consecutive/ adjacent memory locations in the memory heap).
- Each array has a unique name which is used to clearly identify the purpose of the array.
- The data elements in the array are ordered, and array has an index beginning with 0. We can directly access the required elements using the index.
- Arrays class in java has a property called length which is used to get the size of the array.
- The array's size cannot be changed (once initialized).

Example:



Few points to note-

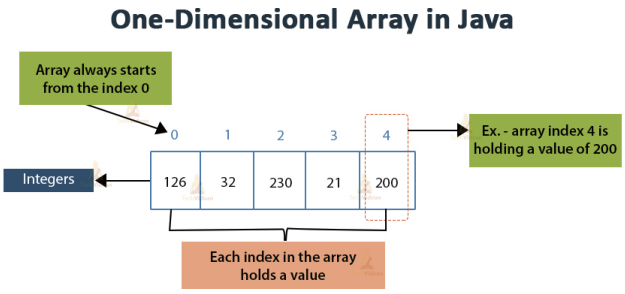
1. The sample array shown above is an integer type array i.e., all the elements are of type integer.
2. Size/ length of the array is defined as the number of elements present. Hence, length=6.
3. The first element i.e. 15 is indexed 0, second element is indexed 1 and so on. So, the last index of the array would be (length-1), i.e. 5.

Array implementation

There are two major components in array declaration- data type and name. Type determines the data type of each element present in the array.
The general form of a one-dimensional array declaration is:

type var-name[];
OR
type[] var-name;

Example 1:
int numberOfItems[];
or int[] numberOfItems;
In both of them, the type of array is 'int' and its name is 'numberOfItems'.



Syntax for declaring a single dimension array -

```
int[] <name_of_array> = {element_0, element_1, element_2, element_3, ...
elementN};
```

Foreach loop is another traversing loop like for in java. It doesn't require initializing the counters variable and the terminating loop conditions.

Syntax:

```
for (type var : array) {
    statements using var;
}
```

Let's consider an example,
LP_Example1.java

Output:

```
Delhi, Chandigarh, Pune,
```

Also, we can create arrays of different data types like char, float, long etc.

eg . long[] longArray;
char charArray[];

Property on Array

1. **length**

This is used to find the length of an array. How many elements an array is storing.

for eg. int[] arr = {1,2,3,4};
arr.length will give 4 as output.

Palindrome Array :

Q1. Given an array, Print true if the array is a palindrome or false otherwise. A palindrome array is an array that reads the same from front end and back end.

Eg. [1,2,3,2,1] Output : The given array is a palindrome.
Eg. [1,2,2,1,2] Output : The given array is not a palindrome.

Solution:

LP_Code1.java

Output :

```
The given array is a palindrome.
```

Approach :

- A palindrome array is an array that reads the same from front end and back end.
- Example : [1 , 2 , 3 , 2 , 1] this array is a palindrome because if read from left to right it is read 1 2 3 2 1 and from right to left also it is reading 1 2 3 2 1 therefore it is a palindrome array.
- Array like [1 , 2 , 3 , 5] is not a palindrome array.
- So we have noticed here one thing, that any ith element from the start and from the back must resemble in order to be a palindrome.
- So we applied the same check for every i and n-i-1 position where n is the size of the array.
- If, at any time, we found that those two elements do not resemble, we can clearly return a false that the given array is not a palindrome.
- We have iterated till half of the array to save some iterations only, because we are already checking for first and last element, second and second last element and so on.

Time complexity : O(n) where n = length of the array
Space complexity : O(1) since no extra space is used.

Reverse Array :

Q2 : Given an array, reverse the given array.
Input : arr[ ] = [1 , 2 ,3 , 4, 1]
Output: [1, 4 , 3 , 2, 1]

Solution :

LP_Code2.java

Output:

```
The reversed array is :
1  4  3  2  1
```

Approach :

- To reverse this array one method is to create a new array of the same size and fill the ith index of the new array with (n-i-1)th index of the old array.
- But this method will consume an extra space of O(n) because a new array is being created.
- Second method could be we can use the same array and interchange the values at i and (n-i-1)th indices.
- This will also result in a reversed array.
- We have implemented this method due to improved space complexity.

Time complexity : O(n) where n = size of the array
Space complexity : O(1) since no extra space is being utilized.

Missing number :

Q3 : Given an array arr[] of size N-1 with integers in the range of [1, N], the task is to find the missing number from the first N integers.

Note: There are no duplicates in the list.

Input: arr[] = {1, 2, 4, 6, 3, 7, 8}, N = 8
Output: 5
Explanation: The missing number between 1 to 8 is 5

Input: arr[] = {1, 2, 3, 5}, N = 5
Output: 4
Explanation: The missing number between 1 to 5 is 4

Solution :

LP_Code3.java

Output :

```
The missing number is : 4
```

Approach :

- Using summation of first N natural numbers : The idea behind the approach is to use the summation of the first N numbers.
- Find the sum of the numbers in the range [1, N] using the formula N * (N+1)/2. Now find the sum of all the elements in the array and subtract it from the sum of the first N natural numbers. This will give the value of the missing element.
- Calculate the sum of the first N natural numbers as sumtotal= N*(N+1)/2.
- Traverse the array from start to end.
- Find the sum of all the array elements.
- Print the missing number as SumTotal – sum of array.

Time Complexity: O(N) where N = length of the array
Auxiliary Space: O(1)

Modification for Overflow: The approach remains the same but there can be an overflow if N is large. In order to avoid integer overflow, pick one number from the range [1, N] and subtract a number from the given array (don't subtract the same number twice). This way there won't be any integer overflow.

LP_Code4.java

Create a variable total = 1 which will store the missing number and a counter variable i = 2.
Traverse the array from start to end.
Update the value of sum as total = total – array[i-2] and increment i by 1. This performs the task mentioned in the above idea.
Print the missing number as a total.

Duplicate number :

Q4. Write a program to print the duplicate elements of an array.Note: There would be only a single duplicate element present in the array for this question.
Input1 : arr[] = [1 4 2 2 5]
Output1 : 2

Input2 : arr[] = [1 2 3 6 8 5 4 8]
Output2 : 8

Solution :

LP_Code5.java

Approach :

- Duplicate elements can be found using two loops. The outer loop will iterate through the array from 0 to length of the array. The outer loop will select an element. The inner loop will be used to compare the selected element with the rest of the elements of the array.
- If a match is found which means the duplicate element is found then, display the element.

Time complexity : O(n^2) where n = length of the array.
Space complexity : O(1)