```
Arrays
======
Arrays are used to store a group of like typed values, whether they are
primitives or objects that can be referenced using one variable

Features of an Array
--------------------
- Arrays are objects in Java.
- In Java all arrays are allocated memory at run time.
- The direct super class of an array type is Object class.
- Arrays can store only homogeneous data, i.e. data of same type.
- Ex: An array of integer values can hold only integer values
- Ex: An array of Strings contain only Strings and so on.
- The values (aka elements) in an array are stored in contiguous memory
locations
- The elements of an array can be accessed by specifying the index of the
desired element within square [ ] brackets after the array name.
- Ex: marks[0] is the 1st element, marks[7] is the 8th element and so
on..

Types of Arrays
===============
- The arrays are identified as being of 'n' dimensions.
- There could any number of dimensions in an array.
- Increase in dimensions of an array increase the complexity of code.
- We generally use upto 3 dimensional arrays.
- So, we have 3 types of arrays:
      1) 1-D array
      2) 2-D array
      3) 3-D array

1-D Array
---------
Suppose we want to store the marks of 3 subjects of 1 student

Syntax 1:
      int [] marks = new int[3];
      marks[0] = 65;
      marks[1] = 67;
      marks[2] = 78;

Syntax 2
      int [] marks = {65, 67, 78};

Note: No of subjects : marks.length
      Print marks    : S.o.p(Arrays.toString(marks));

In Memory : Refer image: 01_1_d_array.png

2-D Array
---------
Suppose there are 3 students, each student has 4 subjects each

Syntax 1
```

```
	int [][] marks = new int[3][4];

	// Student 1 Subject 1
	marks[0][0] = 65

	// Student 1 Subject 2
	marks[0][1] = 65

	// Student 2 Subject 4
	marks[1][3] = 65

	// Student 3 Subject 2
	marks[2][1] = 65

Syntax 2
	int [][] marks = {
		{56, 67, 78, 89},
		{65, 76, 87, 98},
		{91, 81, 71, 61},
	}

Note: No of students : marks.length
	No of subjects of student 2 : marks[1].length
	Print marks    : S.o.p(Arrays.deepToString(marks));

In Memory : Refer image: 02_2_d_array.png

3-D Array
---------
Suppose there are 3 classes, each class has 4 students, each student has
5 subjects each

Syntax 1
	int[][][] marks = new int[3][4][5];

	// Class 1 Student 1 Subject 1
	marks[0][0][0] = 76;

	// Class 2 Student 3 Subject 4
	marks[1][2][3] = 65;

	// Class 3 Student 1 Subject 1
	marks[2][0][0] = 76;

Syntax 2
	int[][][] marks = {
		{
			{56, 67, 78, 89, 92},
			{65, 76, 87, 98, 73},
			{91, 81, 71, 61, 61},
			{87, 62, 78, 59, 65}
		},
		{
			{51, 62, 73, 84, 87},
```

```
                {60, 71, 82, 93, 68},
                {86, 76, 66, 67, 51},
                {82, 82, 98, 89, 45}
            },
            {
                {59, 70, 81, 92, 91},
                {65, 76, 87, 98, 79},
                {91, 91, 71, 89, 61},
                {87, 62, 54, 94, 65}
            }
      };

Note: No of classes : marks.length
      No of students in class 3 : marks[2].length
        No of subjects of student 2 in class 3 : marks[2][1].length
      Print marks    : S.o.p(Arrays.deepToString(marks));

In Memory : Refer image: 03_3_d_array.png

Jagged Array
============
- Jagged arrays (a.k.a Ragged Arrays) is array of arrays such that member
arrays can be of different sizes
- i.e., we can create a 2-D arrays but with variable number of columns in
each row.

For ex: To store the marks of 3 students each having 2, 1 and 3 subjects,
we create a Jagged array

Syntax 1
--------
int [][] marks = new int[3][];
marks[0] = new int[2]; // Student 1 has 2 subjects
marks[1] = new int[1]; // Student 2 has 1 subject
marks[2] = new int[3]; // Student 3 has 3 subjects

Syntax 2
--------
int [][] marks = {
      {56, 67},
      {65},
      {91, 81, 61},
}

Object Arrays
=============
- In the example above we stored integer values in the array.
- Similarly we can also store objects in an array.
- An object array is also known as as array of references

Features
--------
- An "array of objects" in Java contains elements where each element is
an instance of a specific class.
```

- This allows us to group multiple objects of the same type in a single array

1 dimensional array

Suppose we have a class Product with fields id, name, quantity and price
And we have a store that has all products of category Electronics

```
// Declare an array of Product objects of size 3
Product[] electronics = new Product[3];

// Initialize the array with Product objects
electronics[0] = new Product(1, "Laptop", 5, 899.99);
electronics[1] = new Product(2, "Phone", 10, 499.99);
electronics[2] = new Product(3, "Tablet", 7, 299.99);
```

2 dimensional array

Suppose we have a store that has 3 categories, each category having 3 products

```
Product[][] products = {
    // Electronics category
    {
        new Product(1, "Laptop", 5, 899.99),
        new Product(2, "Smartphone", 10, 499.99),
        new Product(3, "Tablet", 7, 299.99)
    },
    // Apparels category
    {
        new Product(4, "T-Shirt", 20, 19.99),
        new Product(5, "Jeans", 15, 39.99),
        new Product(6, "Jacket", 5, 79.99)
    },
    // Eatables category
    {
        new Product(7, "Apple", 100, 0.99),
        new Product(8, "Bread", 50, 1.99),
        new Product(9, "Milk", 30, 2.49)
    }
};
```

```
Note: No of categories : products.length
       No of products in category 2 : products[1].length
     Print products   : S.o.p(Arrays.deepToString(products));
```

The toString() of Product class should be defined to print values properly

In Memory : Refer image: 04_Objects_1d_array.png

Important methods
==========

Here's a list of useful methods in the Arrays class from the java.util package, along with brief descriptions:

1. Arrays.copyOf(array, newLength)
   Creates a new array with a specified length, copying elements from the original array.

2. Arrays.copyOfRange(array, from, to)
   Copies a range of elements from the original array into a new array.

3. Arrays.sort(array)
   Sorts the elements of the array in ascending order.

4. Arrays.sort(array, fromIndex, toIndex)
   Sorts the specified range of the array.

5. Arrays.binarySearch(array, key)
   Searches for the specified key in a sorted array using binary search.

6. Arrays.equals(array1, array2)
   Checks if two arrays are equal, i.e., they contain the same elements in the same order.

7. Arrays.toString(array)
   Returns a string representation of the array.

8. Arrays.deepToString(array)
   Returns a string representation of the array, handling nested arrays.

9. Arrays.fill(array, value)
   Fills the entire array with the specified value.

10. Arrays.fill(array, fromIndex, toIndex, value)
    Fills a specified range of the array with the specified value.

Arrays Parallel Sort
==========
- Java 8 introduced a new method called as parallelSort() in java.util.Arrays Class.
- It uses Parallel Sorting of array elements

Syntax: Arrays.parallelSort(numbers);
-----X-----X-----X-----X-----X-----X-----