Arrays class in Java

Last Updated: 24-04-2019

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.

Class Hierarchy:

```
java.lang.Object
 ↳ java.util.Arrays
```

Class Declaration:

```
public class Arrays
    extends Object
```

Syntax to use Array:

```
Arrays.<function name>;
```

Need for the Java-Arrays Class:

There are often times when **loops** are used to do some tasks on an array like:

- Fill an array with a particular value.

Arrays class provides several static methods that can be used to perform these tasks directly without the use of loops.

Methods in Java Array:

The Arrays class of the java.util package contains several static methods that can be used to fill, sort, search, etc in arrays. These are:

1. **static <T> List<T> asList(T... a)**: This method returns a fixed-size list backed by the specified Arrays.

```
// Java program to demonstrate
// Arrays.asList() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To convert the elements as List
        System.out.println("Integer Array as List: "
                           + Arrays.asList(intArr));
    }
}
```

Output:

```
Integer Array as List: [[I@232204a1]
```

2. **static int binarySearch(elementToBeSearched):** These methods searches for the specified element in the array with the help of Binary Search algorithm.

```
// Java program to demonstrate
// Arrays.binarySearch() method

import java.util.Arrays;

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

```
        int intArr[] = { 10, 20, 15, 22, 35 };
```

```
        System.out.println(intKey
                           + " found at index = "
                           + Arrays
                                 .binarySearch(intArr, intKey));
    }
}
```

Output:

```
 22 found at index = 3
```

3. **static <T> int binarySearch(T[] a, int fromIndex, int toIndex, T key, Comparator<T> c)**: This method searches a range of the specified array for the specified object using the binary search algorithm.

```
// Java program to demonstrate
// Arrays.binarySearch() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        Arrays.sort(intArr);
```



```
        int intKey = 22;

        System.out.println(
            intKey
            + " found at index = "
            + Arrays
                  .binarySearch(intArr, 1, 3, intKey));
    }
}
```

Output:

```
 22 found at index = -4
```

4. **compare(array 1, array 2)**: This method compares two arrays passed as parameters

```
// Java program to demonstrate
// Arrays.compare() method
```

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // Get the second Array
        int intArr1[] = { 10, 15, 22 };

        // To compare both arrays
        System.out.println("Integer Arrays on comparison: "
                           + Arrays.compare(intArr, intArr1));
    }
}
```

Output:

```
Integer Arrays on comparison: 1
```

5. **compareUnsigned(array 1, array 2)**: This method compares two arrays lexicographically, numerically treating elements as unsigned.

```
// Java program to demonstrate
// Arrays.compareUnsigned() method

import java.util.Arrays;

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

        // Get the Arrays
        int intArr[] = { 10, 20, 15, 22, 35 };

        // Get the second Arrays
        int intArr1[] = { 10, 15, 22 };

        // To compare both arrays
        System.out.println("Integer Arrays on comparison: "
                           + Arrays.compareUnsigned(intArr, intArr1));
    }
}
```

Output:

```
Integer Arrays on comparison: 1
```

6. **copyOf(originalArray, newLength)**: This method copies the specified array, truncating or padding with the default value (if necessary) so the copy has the specified length.

```
// Java program to demonstrate
// Arrays.copyOf() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To print the elements in one line
        System.out.println("Integer Array: "
                           + Arrays.toString(intArr));

        System.out.println("\nNew Arrays by copyOf:\n");

        System.out.println("Integer Array: "
                           + Arrays.toString(
                                 Arrays.copyOf(intArr, 10)));
    }
}
```

Output:

```
Integer Array: [10, 20, 15, 22, 35]

New Arrays by copyOf:
```

7. **copyOfRange(originalArray, fromIndex, endIndex)**: This method copies the specified

```
import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To print the elements in one line
        System.out.println("Integer Array: "
                           + Arrays.toString(intArr));

        System.out.println("\nNew Arrays by copyOfRange:\n");

        // To copy the array into an array of new length
        System.out.println("Integer Array: "
                           + Arrays.toString(
                                 Arrays.copyOfRange(intArr, 1, 3)));
    }
}
```

Output:

```
Integer Array: [10, 20, 15, 22, 35]

New Arrays by copyOfRange:

Integer Array: [20, 15]
```

8. **static boolean deepEquals(Object[] a1, Object[] a2)**: This method returns true if the two specified arrays are deeply equal to one another.

```
// Java program to demonstrate
// Arrays.deepEquals() method

import java.util.Arrays;

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

        // Get the Arrays
        int intArr[][] = { { 10, 20, 15, 22, 35 } };
```

```
        // To compare both arrays
```

Output:

```
Integer Arrays on comparison: false
```

9. **static int deepHashCode(Object[] a)**: This method returns a hash code based on the "deep contents" of the specified Arrays.

```
// Java program to demonstrate
// Arrays.deepHashCode() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[][] = { { 10, 20, 15, 22, 35 } };

        // To get the dep hashCode of the arrays
        System.out.println("Integer Array: "
                           + Arrays.deepHashCode(intArr));
    }
}
```

Output:

```
Integer Array: 38475344
```

10. **static String deepToString(Object[] a)**: This method returns a string representation of the "deep contents" of the specified Arrays.

```
// Java program to demonstrate
// Arrays.deepToString() method

import java.util.Arrays;

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

        // Get the Array
```

```
        System.out.println("Integer Array: "
                           + Arrays.deepToString(intArr));
```

Output:

```
Integer Array: [[10, 20, 15, 22, 35]]
```

11. **equals(array1, array2)**: This method checks if both the arrays are equal or not.

```
// Java program to demonstrate
// Arrays.equals() method

import java.util.Arrays;

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

        // Get the Arrays
        int intArr[] = { 10, 20, 15, 22, 35 };

        // Get the second Arrays
        int intArr1[] = { 10, 15, 22 };

        // To compare both arrays
        System.out.println("Integer Arrays on comparison: "
                           + Arrays.equals(intArr, intArr1));
    }
}
```

Output:

```
Integer Arrays on comparison: false
```

12. **fill(originalArray, fillValue)**: This method assigns this fillValue to each index of this Arrays.

```
// Java program to demonstrate
// Arrays.fill() method

import java.util.Arrays;

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

        // Get the Arrays
        int intArr[] = { 10, 20, 15, 22, 35 };
```

```
        Arrays.fill(intArr, intKey);
```

```
    }
```

Output:

```
 Integer Array on filling: [22, 22, 22, 22, 22]
```

13. **hashCode(originalArray)**: This method returns an integer hashCode of this array instance.

```
// Java program to demonstrate
// Arrays.hashCode() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To get the hashCode of the arrays
        System.out.println("Integer Array: "
                           + Arrays.hashCode(intArr));
    }
}
```

Output:

```
 Integer Array: 38475313
```

14. **mismatch(array1, array2)**: This method finds and returns the index of the first unmatched element between the two specified arrays.

```
// Java program to demonstrate
// Arrays.mismatch() method

import java.util.Arrays;

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

```
        // Get the second Arrays
        int intArr1[] = { 10, 15, 22 };
```

```
    }
}
```

Output:

```
The element mismatched at index: 1
```

15. **parallelPrefix(originalArray, fromIndex, endIndex, functionalOperator)**: This method performs parallelPrefix for the given range of the array with the specified functional operator.
16. **parallelPrefix(originalArray, operator)**: This method performs parallelPrefix for complete array with the specified functional operator.
17. **parallelSetAll(originalArray, functionalGenerator)**: This method set all the elements of this array in parallel, using the provided generator function.
18. **parallelSort(originalArray)**: This method sorts the specified array using parallel sort.

```
// Java program to demonstrate
// Arrays.parallelSort() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To sort the array using parallelSort
        Arrays.parallelSort(intArr);

        System.out.println("Integer Array: "
                           + Arrays.toString(intArr));
    }
}
```

Output:

```
Integer Array: [10, 15, 20, 22, 35]
```

19. **setAll(originalArray, functionalGenerator)**: This method sets all the element of the specified array using the generator function provided.
20. **sort(originalArray)**: This method sorts the complete array in ascending order.

```
// Java program to demonstrate
// Arrays.sort() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To sort the array using normal sort-
        Arrays.sort(intArr);

        System.out.println("Integer Array: "
                           + Arrays.toString(intArr));
    }
}
```

Output:

```
Integer Array: [10, 15, 20, 22, 35]
```

21. **sort(originalArray, fromIndex, endIndex)**: This method sorts the specified range of array in ascending order.

```
// Java program to demonstrate
// Arrays.sort() method

import java.util.Arrays;

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

```
        int intArr[] = { 10, 20, 15, 22, 35 };
```

```
                                + Arrays.toString(intArr));
    }
}
```

Output:

```
Integer Array: [10, 15, 20, 22, 35]
```

22. **static <T> void sort(T[] a, int fromIndex, int toIndex, Comparator< super T> c)**: This method sorts the specified range of the specified array of objects according to the order induced by the specified comparator.

```
// Java program to demonstrate working of Comparator
// interface
import java.util.*;
import java.lang.*;
import java.io.*;

// A class to represent a student.
class Student {
    int rollno;
    String name, address;

    // Constructor
    public Student(int rollno, String name,
                   String address)
    {
        this.rollno = rollno;
        this.name = name;
        this.address = address;
    }

    // Used to print student details in main()
    public String toString()
    {
        return this.rollno + " "
            + this.name + " "
            + this.address;
    }
}

class Sortbyroll implements Comparator<Student> {
    // Used for sorting in ascending order of
    // roll number
    public int compare(Student a, Student b)
    {
```

```
// Driver class
```

```
                            new Student(131, "aaaa", "nyc"),
                            new Student(121, "cccc", "jaipur") };

        System.out.println("Unsorted");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);

        Arrays.sort(arr, 1, 2, new Sortbyroll());

        System.out.println("\nSorted by rollno");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
    }
}
```

Output:

```
Unsorted
111 bbbb london
131 aaaa nyc
121 cccc jaipur

Sorted by rollno
111 bbbb london
131 aaaa nyc
121 cccc jaipur
```

23. **static <T> void sort(T[] a, Comparator< super T> c)**: This method sorts the specified array of objects according to the order induced by the specified comparator.

```
// Java program to demonstrate working of Comparator
// interface
import java.util.*;
import java.lang.*;
import java.io.*;

// A class to represent a student.
class Student {
    int rollno;
    String name, address;

    // Constructor
    public Student(int rollno, String name,
                   String address)
```

```
            this.address = address;
        }

        return this.rollno + " "
            + this.name + " "
            + this.address;
    }
}

class Sortbyroll implements Comparator<Student> {

    // Used for sorting in ascending order of
    // roll number
    public int compare(Student a, Student b)
    {
        return a.rollno - b.rollno;
    }
}

// Driver class
class Main {
    public static void main(String[] args)
    {
        Student[] arr = { new Student(111, "bbbb", "london"),
                          new Student(131, "aaaa", "nyc"),
                          new Student(121, "cccc", "jaipur") };

        System.out.println("Unsorted");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);

        Arrays.sort(arr, new Sortbyroll());

        System.out.println("\nSorted by rollno");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
    }
}
```

Output:

```
Unsorted
111 bbbb london
131 aaaa nyc
121 cccc jaipur

Sorted by rollno
111 bbbb london
```

24. **spliterator(originalArray)**: This method returns a Spliterator covering all of the

```
import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To sort the array using normal sort
        System.out.println("Integer Array: "
                           + Arrays.spliterator(intArr));
    }
}
```

Output:

```
Integer Array: java.util.Spliterators$IntArraySpliterator@232204a1
```

25. **spliterator(originalArray, fromIndex, endIndex)**: This method returns a Spliterator of the type of the array covering the specified range of the specified Arrays.

```
// Java program to demonstrate
// Arrays.spliterator() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To sort the array using normal sort
        System.out.println("Integer Array: "
                           + Arrays.spliterator(intArr, 1, 3));
    }
}
```

Output:

```
Integer Array: java.util.Spliterators$IntArraySpliterator@232204a1
```

```
// Java program to demonstrate
// Arrays.stream() method
```

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To get the Stream from the array
        System.out.println("Integer Array: "
                           + Arrays.stream(intArr));
    }
}
```

Output:

```
Integer Array: java.util.stream.IntPipeline$Head@4aa298b7
```

27. **toString(originalArray)**: This method returns a String representation of the contents of this Arrays. The string representation consists of a list of the array's elements, enclosed in square brackets ("[]"). Adjacent elements are separated by the characters a comma followed by a space. Elements are converted to strings as by String.valueOf() function.

```
// Java program to demonstrate
// Arrays.toString() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To print the elements in one line
        System.out.println("Integer Array: "
                           + Arrays.toString(intArr));
    }
}
```

Output:

```
Integer Array: [10, 20, 15, 22, 35]
```

This article is contributed by **Rishabh Mahrsee**. If you like GeeksforGeeks and would like to

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Attention reader! Don't stop learning now. Get hold of all the important Java and Collections concepts with the **Fundamentals of Java and Java Collections Course** at a student-friendly price and become industry ready.

Recommended Posts:

Java.lang.Class class in Java | Set 1

Java.lang.Class class in Java | Set 2

Java.util.Arrays.parallelSetAll(), Arrays.setAll() in Java

Using predefined class name as Class or Variable name in Java

Java.util.TimeZone Class (Set-2) | Example On TimeZone Class

Implement Pair Class with Unit Class in Java using JavaTuples

Implement Triplet Class with Pair Class in Java using JavaTuples

Implement Quintet Class with Quartet Class in Java using JavaTuples

Implement Quartet Class with Triplet Class in Java using JavaTuples

Implement Octet Class from Septet Class in Java using JavaTuples

Implement Ennead Class from Octet Class in Java using JavaTuples

Implement Decade Class from Ennead Class in Java using JavaTuples

Inner Class And Anonymous Inner Class that Implements Runnable | Concurrent Programming Approach 3

Java.util.BitSet class methods in Java with Examples | Set 2

Java.Lang.Float class in Java

Improved By : RishabhPrabhu

Article Tags : Java Java - util package Java-Arrays Java-Collections

Practice Tags : Java Java-Collections

22

2.3

To-do Done

Based on **37** vote(s)

Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

5th Floor, A-118,