BinarySearch using Built-in Methods in Java

If you are a Java programmer, you must have used built-in methods in Java at some point for basic operations like sorting, reversing etc. Java also provides us methods to perform Binary Search on both Arrays and Collection classes. The most commonly used methods in Java to perform Binary Search are:

- **Arrays.binarySearch()**
- **Collections.binarySearch()**

Let's look at each of the above two methods in details:

Arrays.binarySearch()

Arrays.binarySearch() is the simplest and most efficient method to find an element in a sorted array in Java

Declaration:

```
public static int binarySearch(data_type arr, data_type key
)
```

Where **data_type** can be any of the primitive data types: *byte, char, double, int, float, short, long* and *Object* as well.

Description: This method searches the specified array of the given data type for the specified value using the binary search algorithm. The array must be sorted prior to making this call. If it is not sorted, the results are undefined. If the array contains multiple elements with the specified value, there is no guarantee which one will be found.

Parameters:

- arr - the array to be searched
- key - the value to be searched for

Return Value: It returns the index of the search key, if it is contained in the array; otherwise, (-(insertion point) - 1). The insertion point is defined as the point at which the key would be inserted into the array: the index of the first element greater than the key, or a.length if all elements in the array are less than the specified key. Note that this guarantees that the return value will be >= 0 if and only if the key is found.

Examples:

```
Searching for 35 in byteArr[] = {10,20,15,22,35}
will give result as 4 as it is the index of 35

Searching for g in charArr[] = {'g','p','q','c','i'}
will give result as 1 as it is the index of 'g'

Searching for 22 in intArr[] = {10,20,15,22,35};
will give result as 3 as it is the index of 22

Searching for 1.5 in doubleArr[] = {10.2,15.1,2.2,3.5}
will give result as -1 as it is the insertion point of 1.5

Searching for 35.0 in floatArr[] = {10.2f,15.1f,2.2f,3.5f}
will give result as -5 as it is the insertion point of 35.0

Searching for 5 in shortArr[] = {10,20,15,22,35}
will give result as -1 as it is the insertion point of 5
```

Implementation:

```
// Java program to demonstrate working of Arrays.
// binarySearch() in a sorted array
import java.util.Arrays;

public class GFG{
    public static void main(String[] args)
    {
        byte byteArr[] = {10,20,15,22,35};
        char charArr[] = {'g','p','q','c','i'};
        int intArr[] = {10,20,15,22,35};
        double doubleArr[] = {10.2,15.1,2.2,3.5};
        float floatArr[] = {10.2f,15.1f,2.2f,3.5f};
        short shortArr[] = {10,20,15,22,35};
        Arrays.sort(byteArr);
        Arrays.sort(charArr);
        Arrays.sort(intArr);
        Arrays.sort(doubleArr);
        Arrays.sort(floatArr);
        Arrays.sort(shortArr);

        byte byteKey = 35;
        char charKey = 'g';
        int intKey = 22;
        double doubleKey = 1.5;
        float floatKey = 35;
        short shortKey = 5;

        System.out.println(byteKey + " found at index = "
                           +Arrays.binarySearch(byteArr,byteK
        System.out.println(charKey + " found at index = "
                           +Arrays.binarySearch(charArr,charK
        System.out.println(intKey + " found at index = "
                           +Arrays.binarySearch(intArr,intKey
        System.out.println(doubleKey + " found at index = "
                           +Arrays.binarySearch(doubleArr,dou
        System.out.println(floatKey + " found at index = "
                           +Arrays.binarySearch(floatArr,floa
        System.out.println(shortKey + " found at index = "
                           +Arrays.binarySearch(shortArr,shor
    }
}
```

Output:

```
35 found at index = 4
g found at index = 1
22 found at index = 3
1.5 found at index = -1
35.0 found at index = -5
5 found at index = -1
```

Important Points:

- If input list is not sorted, the results are undefined.
- If there are duplicates, there is no guarantee which one will be found.

Collections.binarySearch()

The Collections.binarySearch() method is a Collections class method in Java that returns position of an object in a sorted list.

Declaration:

```
// Returns index of key in sorted list sorted in
// ascending order
public static int binarySearch(List slist, T key)

// Returns index of key in sorted list sorted in
// order defined by Comparator c.
public static int binarySearch(List slist, T key, Comparato
r c)

If key is not present, the it returns "(-(insertion point)
- 1)".
The insertion point is defined as the point at which the ke
y
would be inserted into the list.
```

The method throws **ClassCastException** if elements of list are not comparable using the specified comparator, or the search key is not comparable with the elements.

Searching an int key in a list sorted in ascending order:

```
// Java program to demonstrate working of Collections.
// binarySearch()
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

public class GFG{
    public static void main(String[] args)
    {
        List al = new ArrayList();
        al.add(1);
        al.add(2);
        al.add(3);
        al.add(10);
        al.add(20);

        // 10 is present at index 3.
        int index = Collections.binarySearch(al, 10);
        System.out.println(index);

        // 13 is not present. 13 would have been inserted
        // at position 4. So the function returns (-4-1)
        // which is -5.
        index = Collections.binarySearch(al, 15);
        System.out.println(index);
    }
}
```

Output :

```
3
-5
```

Searching an int key in a list sorted in descending order.

```
// Java program to demonstrate working of Collections.
// binarySearch() in an array sorted in descending order.
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

public class GFG{
    public static void main(String[] args)
    {
        List al = new ArrayList();
        al.add(100);
        al.add(50);
        al.add(30);
        al.add(10);
        al.add(2);

        // The last parameter specifies the comparator method
        // used for sorting.
        int index = Collections.binarySearch(al, 50, Collecti
        System.out.println("Found at index " + index);
    }
}
```

Output :

```
Found at index 1
```

> **Note**: Arrays.binarysearch() works for arrays which can be of primitive data type also. Collections.binarysearch() works for objects Collections like ArrayList and LinkedList.