303.4.1 - Array Processing and Manipulation

Learning Objective

- By the end of this section: Learners should be able to
  - Explain and demonstrate Arrays Class and Its methods, how to use them
  - Describe and Demonstrate Copy One Array into another
  - Demonstrate and utilize Array processing and manipulation

Outline

- Topic 1: Arrays class methods
- Topic 1b: Insert elements using Arrays.fill()
- Topic 1c: Arrays.sort() Methods
- Topic 2: Copy One Array into another
- Topic 2a: Copying Arrays Using For Loop
- Topic 2b: Copying Arrays Using Object.clone() Method
- Topic 2c: Copying Arrays Using System.arraycopy Method
- Topic 2d: Copying Arrays Using Arrays.copyOf Method

- Topic 3: Processing Arrays
  - Initializing arrays with input values.
  - Initializing arrays with random values.
  - Summing all elements.
  - Finding Maximum/Largest element in Array
  - Printing Array In Reverse Order
  - Deletions and Insertions element from Array
  - Accessing Array Elements
    - Linear Search.
    - Binary Search.

Topic 1: Arrays class methods

- Java contains a special utility class that makes it easier for you to perform many often used array operations like copying and sorting arrays, filling in data, searching in arrays etc. The utility class is called Arrays and is located in the standard Java package ***java.util***. Thus, the fully qualified name of the class is: ***java.util.Arrays***

Method name	Description
`Arrays.binarySearch(array, value)`	returns index of value in a sorted array (< 0 if not found)
`Arrays.copyOf(array, length)`	returns a new copy of an array
`Arrays.equals(array1, array2)`	returns true if the two arrays contain same elements
`Arrays.fill(array, value)`	sets every element to the given value
`Arrays.sort(array)`	arranges the elements into sorted order
`Arrays.toString(array)`	returns a string for the array, such as "[10, 30, -25, 17]"

Click here for complete list of available Arrays class methods

Topic 1a:Printing Array Using **Arrays.toString()**

- The **Arrays.toString()** method returns a string representation of the contents of the specified int array.

- You can convert an Java array of primitive types to a String using the Arrays.toString() method.

Example: **Arrays.toString()**


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

    {// Let us create different types of arrays and print their
contents using Arrays.toString()
      boolean[] boolArr = new boolean[] { true, true, false, true };
      byte[] byteArr = new byte[] { 10, 20, 30 };
      char[] charArr = new char[] { 'g', 'e', 'e', 'k', 's' };
      double[] dblArr = new double[] { 1, 2, 3, 4 };
      float[] floatArr = new float[] { 1, 2, 3, 4 };
      int[] intArr = new int[] { 1, 2, 3, 4 };
      long[] lomgArr = new long[] { 1, 2, 3, 4 };
      short[] shortArr = new short[] { 1, 2, 3, 4 };

      System.out.println(Arrays.toString(boolArr));
      System.out.println(Arrays.toString(byteArr));
      System.out.println(Arrays.toString(charArr));
      System.out.println(Arrays.toString(dblArr));
      System.out.println(Arrays.toString(floatArr));
      System.out.println(Arrays.toString(intArr));
      System.out.println(Arrays.toString(lomgArr));
      System.out.println(Arrays.toString(shortArr));

    }
```

Output

```
[true, true, false, true]
[10, 20, 30]
[g, e, e, k, s]
[1.0, 2.0, 3.0, 4.0]
[1.0, 2.0, 3.0, 4.0]
[1, 2, 3, 4]
[1, 2, 3, 4]
[1, 2, 3, 4]
```

Topic 1b: Insert elements using **Arrays.fill()**

- These **Arrays.fill()** methods can fill/ insert an array with a given value. This is easier than iterating through the array and inserting the value yourself.

- Here is an example of using Arrays.fill() to fill an int array:

```
double dValues = new double[100];
Arrays.fill(dValues, 50.0); // set all values to 50.0

long[] lValues = new long[500];
Arrays.fill(lValues, 2057); // set all values to 2057
```

Please view our Wiki documentation for more about Arrays.fill()

Example - **Arrays.fill()**


```
import java.util.Arrays;
public class ArrayTofillDemo {
  public static void main(String[] args) {
     // TODO Auto-generated method stub
     int[] dValues = new int[10];
     Arrays.fill(dValues, 6);   // set all values to 6
     for(int i : dValues) {  // print using loop
        System.out.println(dValues[i]);
     }

     long[] lValues = new long[10];
     Arrays.fill(lValues, 2057);    // set all values to 2057
     System.out.println(Arrays.toString(lValues));  // print using toString()

     int ar[] = {2, 2, 2, 2, 2, 2, 2, 2, 2};
      // Fill from index 1 to index 4.
       Arrays.fill(ar, 1, 5, 10);
       System.out.println(Arrays.toString(ar)); // print using toString()
  }
}
```

Output

```
6
6
6
6
6
6
6
6
6
6
[2057, 2057, 2057, 2057,
2057, 2057, 2057, 2057,
2057, 2057]

[2, 10, 10, 10, 10, 2, 2, 2, 2]
```

TEKsystems Own change

Topic 1c: Arrays.sort() Methods

- Sorting is a common operation in programming.
- There are many **sorting** algorithms are available to solve only one problem i.e. **sort an array**. They have different time complexity and space complexity. But among all sorting algorithms **Quick Sort** gives the best performance.
- The java.util.Arrays class provide numerous **sort()** methods for sorting elements of the array of all the numeric types.
  - Basic Examples:

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
Arrays.sort(numbers);

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
Arrays.sort(chars);
```

Example: Arrays.sort()

Example:

```
public class SortExample {
   public static void main(String[] args)
   {
       // Our arr contains 8 elements
       int[] arr = {13, 7, 6, 45, 21, 9, 101, 102};

       Arrays.sort(arr);

       System.out.printf("Modified arr[] : %s",  Arrays.toString(arr));
   }
}
```

```
Modified arr[] : [6, 7, 9, 13, 21, 45, 101, 102]
```

Topic 2: Copy One Array into another

We can copy **one array into another**. There are several techniques you can use to copy arrays in Java.

- Copying Arrays Using Assignment Statement **(=)**
- Copying Arrays Using **Loop**
- Copying Arrays Using **clone()** Method
- Copying Arrays Using **System.arraycopy()** Method
- Copying Arrays Using **Arrays.copyOf** Method

Topic 2a: Copying Arrays Using **For** Loop

- While copying the arrays, we need **deep copy** rather than a **shallow copy.**
- Deep copies copy the values from an existing object and create a new array object. When you create a deep copy, you can change your new array without affecting the original one.
- One approach that can be used to create a deep copy is to create a **for** loop which iterates through the contents of an array and creates a new array.

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

    int[] sourceArrays = {2,3,4,5,10};
    int[] targetArrays = new int[sourceArrays.length];
    for(int i =0; i < sourceArrays.length; i++)
    {
       targetArrays[i] = sourceArrays[i];
    }
System.out.println(Arrays.toString(sourceArrays));
System.out.println(Arrays.toString(targetArrays));
if (targetArrays == sourceArrays) {
   System.out.println("Same instance");
} else {
   System.out.println("Different instance");
}
 }
}
```

Output
Source Array: [2, 3, 4, 5, **10**]
Target Array: [2, 3, 4, 5, 10]
Source Array after element change: [2, 3, 4, 5, **500**]

Topic 2b: Copying Arrays Using **Object.clone()** Method

- Cloning involves creating a new array of the same size and type, the entire ***sources array's*** elements are copied into the ***target array***. The **clone()** method is defined in the **Object** class. **Remember:** The array in Java is an **Object** because the direct **Superclass** of an Array is **Object Class**. So we can use object class's method /attributes with array

Example:

```
public class cloneExample {
   public static void main(String[] args)
   {
       // clone() method is inherited from Object class
       int[] sourceArray = {1,2,3};
       int[] targetArray = sourceArray.clone();
       sourceArray[2] = 500;

       System.out.println("Source Array: " + Arrays.toString(sourceArray));
       System.out.println("Target Array: " + Arrays.toString(targetArray));
   }
}
```

Source Array: [1, 2, 500]
Target Array: [1, 2, 3]

Topic 2c: Copying Arrays Using **System.arraycopy** Method

In Java, the System class contains a method named ***arraycopy()*** to copy arrays. This method is a better approach to copy arrays than the previous approaches.
arraycopy() method allows us to copy **specified portion** of a ***source array*** to a ***target array***.

```
Syntax:        System.arraycopy(sourceArray, srcPos, dest, destPos, length);
```

Please view our **Wiki** documentation for more about **System.arraycopy()**

Topic 2d: Copying Arrays Using **Arrays.copyOf** Method

- Java class **Arrays** provides a method called Arrays.copyOf() that returns a copy of an array passed as a parameter to this method, followed by specifying its size. This methods have a second length argument, which allows us to specify a new size for the copy:

Syntax:

```
Arrays.copyOf(int[] templateArray, int length);
```

Please view our **Wiki** documentation for more about Arrays.copyOf()

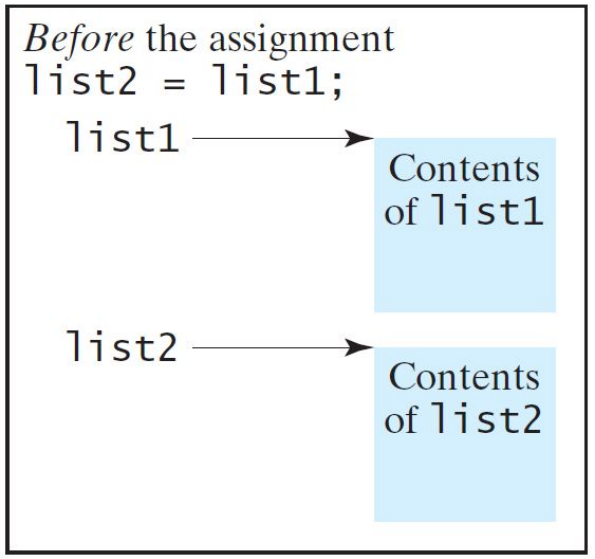
Topic 2c: Copying Arrays Using Assignment Statement (=)

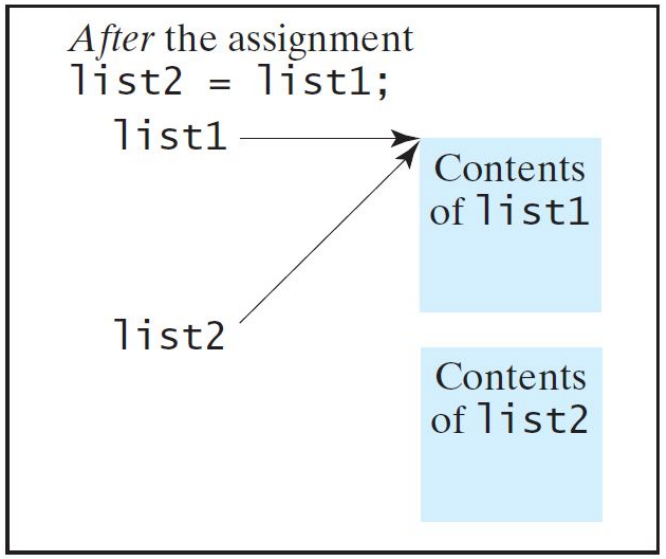
You could naively attempt to use the assignment statement (=) as follows:

```
list2 = list1;
```

But, there is a problem with this technique, Arrays are reference types, which means that the above statement will not make a copy. Actually we are copying a reference of the array.

List2 array starts referring to the List1. If we change elements of List 1 array, corresponding elements of the List 2 arrays also change





Click here for the program example.
You can go to Wiki Document.

Topic 3: Processing Arrays

- Array Operations:
  - Initializing arrays with input values.
  - Initializing arrays with random values.
  - Summing all elements.
  - Finding Maximum/Largest element in Array
  - Finding Maximum/Largest element in Array
  - Printing Array In Reverse Order
  - Deletions and Insertions element from Array
  - Accessing Array Elements

Initializing Arrays With Input Values

In this example, an array is initialized with values read from the console.

```
public static void main(String[] args)
    {
        int n;
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter the number of elements you want to
store: ");
//reading the number of elements from the that we want to enter
        n=sc.nextInt();
//creates an array in the memory of length 10
        int[] array = new int[10];
        System.out.println("Enter the elements of the array: ");
        for(int i=0; i<n; i++)
        {
//reading array elements from the user
            array[i]=sc.nextInt();
        }
        System.out.println("Array elements are: ");
// accessing array elements using the for loop
        for (int i=0; i<n; i++)
        {
            System.out.println(array[i]);
        }
    }
```

More often, we might initialize an array with values read from a file, a network, or a database.

Initializing Arrays With Random Values

- Some applications require random numbers.
- In this example, we use the random() method of the Math class to initialize an array:

```
public static void main(String[] args)
   {
          double[] arr = new double[5];
          for (int i = 0; i < 5; i++) {
           arr[i] = Math.random();
       }
       System.out.println("Random numbers = "+ Arrays.toString(arr));
   }
```

Summing all Elements in Array

In the below program, we are going to add all the elements of an array.

```
// Calculate the sum of all elements of an array

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

   // an array of numbers
   int[] numbers = {3, 4, 5, -5, 0, 12};
   int sum = 0;

   // iterating through each element of the array
   for (int number: numbers) {
     sum += number;
   }

   System.out.println("Sum = " + sum);
 }
 }
```

Finding **Minimum/Smallest** element in Array

Java does not have any built-in functions for finding minimum element, so Let see, how to find minimum value / element in Array.

```
public class findsmallestElement{
   public static void main(String[] args) {
     int[] myarray = {0, 2, 4, 6, 8, 10};
     int minVal = Integer.MAX_VALUE;

     for (int i = 0; i < myarray.length; i++) {
       if (myarray[i] < minVal) {
       minVal = myarray[i];
       }
     }
       System.out.println("minVal = " + minVal);
   }
}
```

Output:
minVal = 0

Explanation :

First sets the **minVal** to **Integer.MAX_VALUE** which is the highest possible value an int can take. This is done to make sure that the initial value is not by accident smaller than the smallest value in the array.

Second, the example iterates through the array and compares each value to **minValue**. If the element in the array is smaller than **minVal** then **minVal** is set to the value of the element.

Finally the minimum value found in the array is printed out. In this example the minimum value is 0.

Finding **Maximum/Largest** element in Array

Java does not have any built-in functions for finding maximum element, so Let see, how to find maximum value / element in Array

```
public class findlargestnumber{
   public static void main(String[] args) {
     int[] arr = {25,0,2,4,6,8,10};
     int maxVal = Integer.MIN_VALUE; // or 0
     for(int i=0; i < arr.length; i++){
      if(arr[i] > maxVal){
        maxVal = arr[i];
      }
    }
     System.out.println("Max Element = " + maxVal);
   }
}
```

It is pretty similar to finding the minimum value. The major differences to finding the maximum value is the initialization of **`maxVal,`** Notice how **`maxVal`** is initialized to the smallest possible value. and the comparison of **`maxVal`** to the elements in the array.

Output
Max Element = 25

Printing Array In **Reverse** Order

If we want to print the array in the **reverse** order, without actually reversing it, then we can do that just by providing a ***for loop*** that will start printing from the end of the array. This is a good option as long as we just want to print the array in reverse order without doing any processing with it.

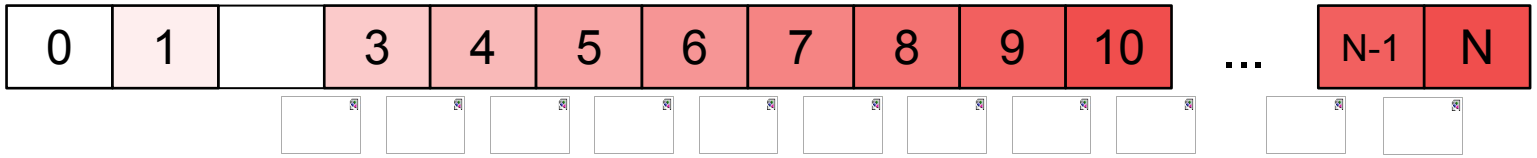
Output

```
Original Array
10   20   30   40   50   60   70   80   90

Original Array printed in reverse
order:
90   80   70   60   50   40   30   20   10
```

```java
public class ReverseOrderExample {
    public static void main(String[] args) {
        Integer[] intArray = {10,20,30,40,50,60,70,80,90};
        //-----print array starting from first element---------
        System.out.println("Original Array:");
        for(int i=0;i<intArray.length;i++) {
            System.out.print(intArray[i] + "   ");
        }
        System.out.println();
        //----------print array starting from last element----
        System.out.println("Original Array printed in reverse
order:");
        for(int i=intArray.length-1;i>=0;i--) {
            System.out.print(intArray[i] + "   ");
        }
    }
}
```

Deletions and Insertions element from Array

- We occasionally need to delete or insert an element from or into an array. However, because arrays are of ***fixed sized***, we cannot actually ***"delete"*** or ***"insert"*** But we can shift elements:
  - To "delete" an element, we shift all elements above the index leftward.



  - To "insert" an element, we create a gap by shifting all elements above the index rightward.

Please view our **Wiki** documentation for more explaination and details about **deletion and insertion element from Array**

Accessing Array Elements

- If you know a value but do not know its index, then you must search the array. Let's look at two search algorithms:
  - Linear Search.
  - Binary Search.

Example: Binary Search
Arrays.binarySearch() method

We do not need to develop a Binary search algorithm because The Arrays class contains a set of methods called ***binarySearch()***. This method helps you perform a binary search in an array.

The ***binarySearch()*** method will return the **index number** in the array in which the element was found

Remember for the binary search, The array must first be sorted. You can do so yourself, or via the ***Arrays.sort()*** method covered earlier in this text.

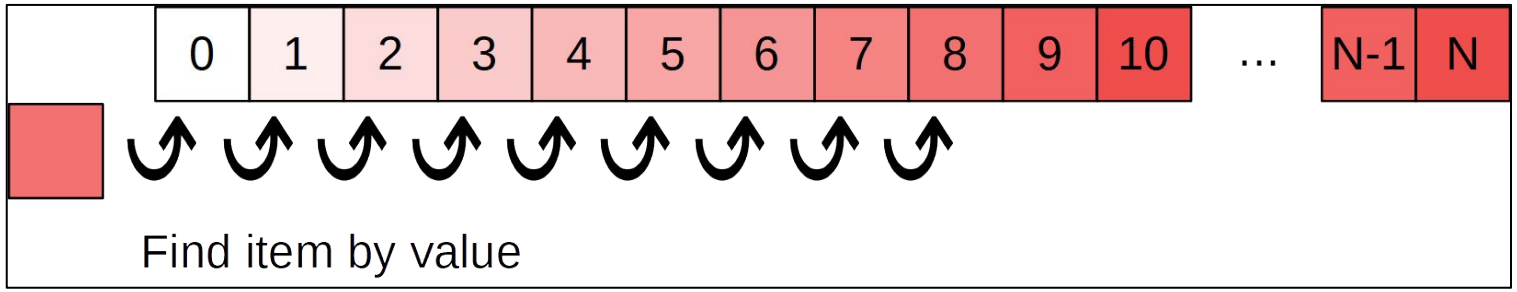
```
import java.util.Arrays;
public class binarysearchexample {
   public static void main(String[] args) {
       int intArr[] = {10, 20, 15, 22, 35};

       // sorting the array
       Arrays.sort(intArr);

       // declare element for searching
       int element = 22;
System.out.println(element + " found at index = " +
Arrays.binarySearch(intArr, element));
   }
}
```

Please view our **Wiki** documentation for more explanation and details about Binary Search

Output
22 found at index = 3

Overview of Linear Search

- If we know nothing about the array, and know only the value of an item, then we must perform a linear search.
- We start at the beginning of the array and **compare each element to our search value.**
- If we find the item, we return the index.
- If the item is not found after checking every element in the array, we return a sentinel value, typically -1.



Please view our **Wiki** documentation for more explanation and details about Linear Search

When Does **ArrayIndexOutOfBoundsException** Occur?

- The ***ArrayIndexOutOfBoundsException*** occurs whenever we access an index which is not present in the array or invalid index
- Indexing in an array starts from **zero** and must never be greater **than** or **equal to the size of the array**.
  - ➢ In short, **the rule of thumb is 0 <= index < (size of array).**
- For example see the code snippet below:

```
String[] vowels = new String[]{"a", "i", "u", "e", "o"}

String vowel = vowels[10]; // throws the ArrayIndexOutOfBoundsException
```

Check you knowledge

Ques1: What is the representation of the third element in an array called a?

A. a[2]

B. a(2)

C. a[3]

D. a(3)

Check you knowledge

Ques 2: What would be the result of attempting to compile and run the following code?

```
public class Test {
  public static void main(String[] args) {
    double[] x = new double[]{1, 2, 3};
    System.out.println("Value is " + x[1]);
  }
}
```

A. The program has a compile error because the syntax new double[]{1, 2, 3} is wrong and it should be replaced by {1, 2, 3}.

B. The program has a compile error because the syntax new double[]{1, 2, 3} is wrong and it should be replaced by new double[3]{1, 2, 3};

C. The program has a compile error because the syntax new double[]{1, 2, 3} is wrong and it should be replaced by new double[]{1.0, 2.0, 3.0};

D. The program compiles and runs fine and the output "Value is 1.0" is printed.

E. The program compiles and runs fine and the output "Value is 2.0" is printed.

Check you knowledge

Ques 2: What would be the result of attempting to compile and run the following code?

```
public class Test {
  public static void main(String[] args) {
    int[] x = new int[5];
    int i;
    for (i = 0; i < x.length; i++)
      x[i] = i;
    System.out.println(x[i]);
  }
}
```

A. The program displays 0 1 2 3 4.

B. The program displays 4.

C. The program has a runtime error because the last statement in the main method causes ArrayIndexOutOfBoundsException.

D. The program has a compile error because i is not defined in the last statement in the main method.

Answers

1. A. a[2]
2. E. The program compiles and runs fine and the output "Value is 2.0" is printed.
3. C. The program has a runtime error because the last statement in the main method causes ArrayIndexOutOfBoundsException.

Practice Assignment - #**303.3.2 Array**

Complete this assignment 303.3.2 Practice Assignment -Array **You can find this assignment on Canvas, under the Assignment section.**

Note: Use your office hours to complete this assignment. If you have technical questions while performing the practice assignment, ask your instructors for assistance.

Practice Assignment - #**303.3.3 Array**

This assignment will be administered through HackerRank.

Click here for Hackerank link - Java Subarray

Note: Use your office hours to complete this assignment, If you have any technical questions while performing the assignment, ask your instructors for assistance.

Summary

- Arrays are one of the core functionalities of Java; therefore, it is really important to understand how arrays work and to know what we can and cannot do with them.
- In this presentation, we learned copying arrays, Arrays Class, and how to find the largest element, delete from and insert into arrays, and reverse an array.

- We covered basic and advanced uses of Arrays in Java. We saw that Java offers several methods to deal with Arrays through the Arrays utility class.

Questions?