**Java Copy Arrays**

In Java, we can copy one array into another. There are several techniques you can use to copy arrays in Java.

---

**1. Copying Arrays Using Assignment Operator**

Let's take an example,

```
class Main {
    public static void main(String[] args) {
        int [] numbers = {1, 2, 3, 4, 5, 6};
        int [] positiveNumbers = numbers;    // copying arrays
        for (int number: positiveNumbers) {
            System.out.print(number + ", ");
        }
    }
}
```

Run Code

**Output**:

```
1, 2, 3, 4, 5, 6
```

In the above example, we have used the assignment operator (=) to copy an array named numbers to another array named positiveNumbers.

This technique is the easiest one and it works as well. However, there is a problem with this technique. If we change elements of one array, corresponding elements of the other arrays also change. For example,

```
class Main {
    public static void main(String[] args) {
        int [] numbers = {1, 2, 3, 4, 5, 6};
        int [] positiveNumbers = numbers;    // copying arrays
        // change value of first array
        numbers[0] = -1;
        // printing the second array
        for (int number: positiveNumbers) {
            System.out.print(number + ", ");
        }
    }
}
```

Run Code

**Output**:

```
-1, 2, 3, 4, 5, 6
```

Here, we can see that we have changed one value of the numbers array. When we print the positiveNumbers array, we can see that the same value is also changed.

It's because both arrays refer to the same array object. This is because of the shallow copy. To learn more about shallow copy, visit shallow copy.

Now, to make new array objects while copying the arrays, we need deep copy rather than a shallow copy.

---

**2. Using Looping Construct to Copy Arrays**

Let's take an example:

```
import java.util.Arrays;

class Main {

   public static void main(String[] args) {

      int [] source = {1, 2, 3, 4, 5, 6};

      int [] destination = new int[6];

      // iterate and copy elements from source to destination

      for (int i = 0; i < source.length; ++i) {

         destination[i] = source[i];

      }

       // converting array to string

      System.out.println(Arrays.toString(destination));

   }

}
```

Run Code

**Output**:

```
[1, 2, 3, 4, 5, 6]
```

In the above example, we have used the for loop to iterate through each element of the source array. In each iteration, we are copying elements from the source array to the destination array.

Here, the source and destination array refer to different objects (deep copy). Hence, if elements of one array are changed, corresponding elements of another array is unchanged.

Notice the statement,

```
System.out.println(Arrays.toString(destination));
```

Here, the toString() method is used to convert an array into a string. To learn more, visit the toString() method (official Java documentation).

**3. Copying Arrays Using 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 above two.

The arraycopy() method allows you to copy a specified portion of the source array to the destination array. For example,

```
arraycopy(Object src, int srcPos,Object dest, int destPos, int length)
```

Here,

- src - source array you want to copy
- srcPos - starting position (index) in the source array
- dest - destination array where elements will be copied from the source
- destPos - starting position (index) in the destination array
- length - number of elements to copy

Let's take an example:

```
// To use Arrays.toString() method

import java.util.Arrays;

class Main {

  public static void main(String[] args) {

    int[] n1 = {2, 3, 12, 4, 12, -2};

    int[] n3 = new int[5];

    // Creating n2 array of having length of n1 array

    int[] n2 = new int[n1.length];

    // copying entire n1 array to n2

    System.arraycopy(n1, 0, n2, 0, n1.length);

    System.out.println("n2 = " + Arrays.toString(n2));

    // copying elements from index 2 on n1 array

    // copying element to index 1 of n3 array

    // 2 elements will be copied

    System.arraycopy(n1, 2, n3, 1, 2);

    System.out.println("n3 = " + Arrays.toString(n3));

  }

}
```

Run Code

**Output**:

```
n2 = [2, 3, 12, 4, 12, -2]

n3 = [0, 12, 4, 0, 0]
```

In the above example, we have used the arraycopy() method,

- System.arraycopy(n1, 0, n2, 0, n1.length) - entire elements from the n1 array are copied to n2 array
- System.arraycopy(n1, 2, n3, 1, 2) - 2 elements of the n1 array starting from index 2 are copied to the index starting from 1 of the n3 array

As you can see, the default initial value of elements of an int type array is 0.

---

**4. Copying Arrays Using copyOfRange() method**

We can also use the copyOfRange() method defined in Java Arrays class to copy arrays. For example,

```
// To use toString() and copyOfRange() method

import java.util.Arrays;

class ArraysCopy {

   public static void main(String[] args) {

      int[] source = {2, 3, 12, 4, 12, -2};

      // copying entire source array to destination

      int[] destination1 = Arrays.copyOfRange(source, 0, source.length);

      System.out.println("destination1 = " + Arrays.toString(destination1));

      // copying from index 2 to 5 (5 is not included)

      int[] destination2 = Arrays.copyOfRange(source, 2, 5);

      System.out.println("destination2 = " + Arrays.toString(destination2));

   }

}
```

Run Code

**Output**

```
destination1 = [2, 3, 12, 4, 12, -2]

destination2 = [12, 4, 12]
```

In the above example, notice the line,

```
int[] destination1 = Arrays.copyOfRange(source, 0, source.length);
```

Here, we can see that we are creating the destination1 array and copying the source array to it at the same time. We are not creating the destination1 array before calling the copyOfRange() method. To learn more about the method, visit Java copyOfRange.

**5. Copying 2d Arrays Using Loop**

Similar to the single-dimensional array, we can also copy the 2-dimensional array using the for loop. For example,

```
import java.util.Arrays;

class Main {

  public static void main(String[] args) {

    int[][] source = {

        {1, 2, 3, 4},

        {5, 6},

        {0, 2, 42, -4, 5}

        };

    int[][] destination = new int[source.length][];

    for (int i = 0; i < destination.length; ++i) {

      // allocating space for each row of destination array

      destination[i] = new int[source[i].length];

      for (int j = 0; j < destination[i].length; ++j) {

        destination[i][j] = source[i][j];

      }

    }

    // displaying destination array

    System.out.println(Arrays.deepToString(destination));

  }

}
```

Run Code

**Output**:

```
[[1, 2, 3, 4], [5, 6], [0, 2, 42, -4, 5]]
```

In the above program, notice the line,

```
System.out.println(Arrays.deepToString(destination);
```

Here, the deepToString() method is used to provide a better representation of the 2-dimensional array. To learn more, visit Java deepToString().

**Copying 2d Arrays using arraycopy()**

To make the above code more simpler, we can replace the inner loop with System.arraycopy() as in the case of a single-dimensional array. For example,

```
import java.util.Arrays;

class Main {

  public static void main(String[] args) {

    int[][] source = {

        {1, 2, 3, 4},

        {5, 6},

        {0, 2, 42, -4, 5}

        };

    int[][] destination = new int[source.length][];

    for (int i = 0; i < source.length; ++i) {

        // allocating space for each row of destination array

        destination[i] = new int[source[i].length];

        System.arraycopy(source[i], 0, destination[i], 0, destination[i].length);

    }

    // displaying destination array

    System.out.println(Arrays.deepToString(destination));

  }

}
```

Run Code

**Output**:

```
[[1, 2, 3, 4], [5, 6], [0, 2, 42, -4, 5]]
```

Here, we can see that we get the same output by replacing the inner for loop with the arraycopy() method.