If we need to create copies of our array, then we can use the `copyOf()` method from the `Arrays` class. We need to provide the array that needs to be copied and the new array's size as a parameter.

The below example shows how to create a copy of an array where the copied array size is the same as the original array. If the new array's size is greater than the original array, then the remaining positions are filled with zeros.

```
import java.util.Arrays;

public class ArraysDemo {

    public static void main(String args[]) {
        int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        int[] newArray = Arrays.copyOf(numbers, numbers.length);

        System.out.println("The copied array is: ");

        for (int i : newArray) {
            System.out.print(i + " ");
        }

        int[] newArrayBiggerSize = Arrays.copyOf(numbers, 20);
        System.out.println();
        System.out.println("The copied array is: ");

        for (int i : newArrayBiggerSize) {
            System.out.print(i + " ");
        }
    }
}
```

Run Save Reset

It is possible that we may only want to copy a part of our original array. In that case, we can use the `copyOfRange()` method. This method takes three arguments: the original array, the *from* index (which is inclusive), and a *to* index (which is exclusive).

```
import java.util.Arrays;

public class ArraysDemo {

    public static void main(String args[]) {
        int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        int[] newArray = Arrays.copyOfRange(numbers, 0, 5);

        System.out.println("The copied array is: ");

        for (int i : newArray) {
            System.out.println(i);
        }
    }
}
```

Run Save Reset

Have you ever wondered what would happen if we create a copy of an array that contains objects of a custom class?

If we change the object in the original array, will it be changed in the copied array?

Let's try to answer these questions using an example. In the below example, we have created an array of two Employee objects. Then we created a copy of this array. We will see what happens when one of the Employee objects is changed in the original array.

ArraysDemo.java

Employee.java

```
public class Employee {

    int empId;
    String empName;

    public Employee(int empId, String empName) {
        super();
        this.empId = empId;
        this.empName = empName;
    }

}
```

Run Save Reset

As we can see from the above program's output, the name did not change in the copied array. This means that the `copyOf()` method creates a deep copy of objects instead of just changing the references.