```java
package ch.publicept;

import java.util.Arrays;

/**
 * 08-101_References-Types_vs_Value-Types
 *
 * @author created by Urs Albisser, on 2020-01-03
 * @version 0.0.1
 */


/**
 * Main
 * main(String[] args)
 *
 * modifyArray(int[] array)
 * dereferenceArray(int[] array)
 */
public class Main {

    public static void main(String[] args) {


        // == reference of primitive type ==
        int myIntValue = 10;
        int anotherIntValue = myIntValue;   // make a copy of myIntValue (= independent variable)

        System.out.println("myIntValue " + myIntValue);             // is 10
        System.out.println("anotherIntValue " + anotherIntValue);   // is 10
        System.out.println();

        anotherIntValue++; // increase only one var

        System.out.println("myIntValue " + myIntValue);             // is 10
        System.out.println("anotherIntValue " + anotherIntValue);   // is 11 (as it is independent)
        System.out.println();


        // == reference types (e.g. arrays or classes) ==
        // A reference variable (new keyword) holds a reference to the object (address in memory),
        // but not the object itself.
        int[] myIntArray = new int[5];
        int[] anotherIntArray = myIntArray; // another reference to the same array in memory!

        // print out contents of the array as String of Strings: [element1, element2, ...] .
        System.out.println("myIntArray " + Arrays.toString(myIntArray));            // is [0, 0, 0, 0, 0]
        System.out.println("anotherIntArray " + Arrays.toString(anotherIntArray));  // is [0, 0, 0, 0, 0]
        System.out.println();

        anotherIntArray[0] = 1; // assign 1 to the element 0 in the array
        System.out.println("after change myIntArray " + Arrays.toString(myIntArray));            // is [1, 0, 0, 0, 0]
        System.out.println("after change anotherIntArray " + Arrays.toString(anotherIntArray)); // is [1, 0, 0, 0, 0]
        System.out.println();

        // modify array by calling a method
        modifyArray(myIntArray);
        System.out.println("after modify myIntArray " + Arrays.toString(myIntArray));            // is [2, 0, 0, 0, 0]
        System.out.println("after modify anotherIntArray " + Arrays.toString(anotherIntArray)); // is [2, 0, 0, 0, 0]
        System.out.println();

        // de-reference array by calling a method
        dereferenceArray(myIntArray);
        System.out.println("after de-referencing myIntArray "
                + Arrays.toString(myIntArray));         // is [2, 0, 0, 0, 0]
        System.out.println("after de-referencing anotherIntArray "
                + Arrays.toString(anotherIntArray));    // is [2, 0, 0, 0, 0]
        System.out.println();

        // de-reference anotherIntArray by new keyword creates a new object in memory.
        anotherIntArray = new int[] {5, 6, 7, 8, 9};
        modifyArray(myIntArray);
        System.out.println("after de-referencing myIntArray "
                + Arrays.toString(myIntArray));         // is [2, 0, 0, 0, 0]
        System.out.println("after de-referencing anotherIntArray and new initialization "
                + Arrays.toString(anotherIntArray));    // is [5, 6, 7, 8, 9]
        System.out.println();
    }


    /**
     * modifyArray()
     * @param array array to be modified
     */
    private static void modifyArray(int[] array) {

        array[0] = 2;
    }


    /**
     * modifyArrayAgain()
     * De-referencing the reference by using the new keyword creates a new object in memory.
     * @param array array to be modified
     */
    private static void dereferenceArray(int[] array) {

        array = new int[] {1, 2, 3, 4, 5};  // de-referencing the reference with "new" -> new object in memory
    }
}

```