

```

1 package ch.publiccept;
2
3 import java.util.Arrays;
4
5 /**
6  * 08-101_References-Types_vs_Value-Types
7  *
8  * @author created by Urs Albisser, on 2020-01-03
9  * @version 0.0.1
10 */
11
12
13 /**
14  * Main
15  * main(String[] args)
16  *
17  * modifyArray(int[] array)
18  * dereferenceArray(int[] array)
19  */
20 public class Main {
21
22     public static void main(String[] args) {
23
24
25         // == reference of primitive type ==
26         int myIntValue = 10;
27         int anotherIntValue = myIntValue;    // make a copy of myIntValue (= independent variable)
28
29         System.out.println("myIntValue " + myIntValue);           // is 10
30         System.out.println("anotherIntValue " + anotherIntValue); // is 10
31         System.out.println();
32
33         anotherIntValue++; // increase only one var
34
35         System.out.println("myIntValue " + myIntValue);           // is 10
36         System.out.println("anotherIntValue " + anotherIntValue); // is 11 (as it is independent)
37         System.out.println();
38
39
40         // == reference types (e.g. arrays or classes) ==
41         // A reference variable (new keyword) holds a reference to the object (address in memory),
42         // but not the object itself.
43         int[] myIntArray = new int[5];
44         int[] anotherIntArray = myIntArray; // another reference to the same array in memory!
45
46         // print out contents of the array as String of Strings: [element1, element2, ...] .
47         System.out.println("myIntArray " + Arrays.toString(myIntArray)); // is [0, 0, 0, 0, 0]
48         System.out.println("anotherIntArray " + Arrays.toString(anotherIntArray)); // is [0, 0, 0, 0, 0]
49         System.out.println();
50
51         anotherIntArray[0] = 1; // assign 1 to the element 0 in the array
52         System.out.println("after change myIntArray " + Arrays.toString(myIntArray)); // is [1, 0, 0, 0, 0]
53         System.out.println("after change anotherIntArray " + Arrays.toString(anotherIntArray)); // is [1, 0, 0, 0, 0]
54         System.out.println();
55
56         // modify array by calling a method
57         modifyArray(myIntArray);
58         System.out.println("after modify myIntArray " + Arrays.toString(myIntArray)); // is [2, 0, 0, 0, 0]
59         System.out.println("after modify anotherIntArray " + Arrays.toString(anotherIntArray)); // is [2, 0, 0, 0, 0]
60         System.out.println();
61
62         // de-reference array by calling a method
63         dereferenceArray(myIntArray);
64         System.out.println("after de-referencing myIntArray "
65             + Arrays.toString(myIntArray)); // is [2, 0, 0, 0, 0]
66         System.out.println("after de-referencing anotherIntArray "
67             + Arrays.toString(anotherIntArray)); // is [2, 0, 0, 0, 0]
68         System.out.println();
69
70         // de-reference anotherIntArray by new keyword creates a new object in memory.
71         anotherIntArray = new int[] {5, 6, 7, 8, 9};
72         modifyArray(myIntArray);
73         System.out.println("after de-referencing myIntArray "
74             + Arrays.toString(myIntArray)); // is [2, 0, 0, 0, 0]
75         System.out.println("after de-referencing anotherIntArray and new initialization "
76             + Arrays.toString(anotherIntArray)); // is [5, 6, 7, 8, 9]
77         System.out.println();
78     }
79
80
81     /**
82     * modifyArray()
83     * @param array array to be modified
84     */
85     private static void modifyArray(int[] array) {
86
87         array[0] = 2;
88     }
89
90
91     /**
92     * modifyArrayAgain()
93     * De-referencing the reference by using the new keyword creates a new object in memory.
94     * @param array array to be modified
95     */
96     private static void dereferenceArray(int[] array) {
97

```

```
98     array = new int[] {1, 2, 3, 4, 5}; // de-referencing the reference with "new" -> new object in memory
99 }
100 }
101
```