

 Evgen959 / Advanced_Backend Public[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#)[Advanced_Backend](#) / [Lesen 032](#) / [code](#) / [hw_na_les031_1](#) / [test](#) / MainTest.java 

Evgen959 newLes32

74dd8bb · 10 hours ago



119 lines (93 loc) · 4.36 KB

Code

Blame

Raw



```
1  import org.junit.jupiter.api.Assertions;
2  import org.junit.jupiter.api.DisplayName;
3  import org.junit.jupiter.api.Test;
4
5  import java.util.ArrayList;
6  import java.util.List;
7
8  import static org.junit.jupiter.api.Assertions.*;
9
10  class MainTest {
11
12      @Test
13      @DisplayName("selectEmployeeForBonus: regular case")
14      void selectEmployeeForBonus() {
15          List<Employee> list = new ArrayList<>();
16          list.add(new Employee("John", 2020));
17          list.add(new Employee("Anna", 2016));
18          list.add(new Employee("Jack", 1999));
19          list.add(new Employee("Mark", 1995));
20          list.add(new Employee("Mark", 1997));
21          list.add(new Employee("Oleg", 2000));
22          list.add(new Employee("Valerii", 2005));
23
24          List<Employee> expectedList = new ArrayList<>();
25          expectedList.add(new Employee("Mark", 1997));
26          expectedList.add(new Employee("Mark", 1995));
27          expectedList.add(new Employee("Jack", 1999));
28
29          List<Employee> result = Main.selectEmployeeForBonus(list, 3);
30          Assertions.assertTrue(isEqualsIgnoreOrder(expectedList, result),
31              String.format("%nExpected: %s%nActual: %s%n", expectedList, result));
32
33          //Assertions.assertEquals(expectedList, result);
34      }
35
36      @Test
37      @DisplayName("selectEmployeeForBonus: regular case case| more than N employees")
38      void selectEmployeeForBonus2() {
39          List<Employee> list = new ArrayList<>();
40          list.add(new Employee("John", 2020));
41          list.add(new Employee("Anna", 2016));
42          list.add(new Employee("Jack", 1999));
```

```
43     list.add(new Employee("Tom", 1999));
44     list.add(new Employee("Mark", 1995));
45     list.add(new Employee("Mark", 1997));
46     list.add(new Employee("Igor", 1999));
47     list.add(new Employee("Oleg", 2000));
48     list.add(new Employee("Valerii", 2005));
49
50     List<Employee> expectedList = new ArrayList<>();
51     expectedList.add(new Employee("Mark", 1997));
52     expectedList.add(new Employee("Mark", 1995));
53     expectedList.add(new Employee("Jack", 1999));
54     expectedList.add(new Employee("Tom", 1999));
55     expectedList.add(new Employee("Igor", 1999));
56
57
58     List<Employee> result = Main.selectEmployeeForBonus(list, 3);
59     Assertions.assertTrue(isEqualsIgnoreOrder(expectedList,result),
60         String.format("%nExpected: %s%nActual:   %s%n", expectedList, result));
61
62     //Assertions.assertEquals(expectedList, result);
63 }
64
65 @Test
66 @DisplayName("selectEmployeeForBonus: all employee sould be selected")
67 void selectAllEmployeeForBonus() {
68     List<Employee> list = new ArrayList<>();
69     list.add(new Employee("John", 2020));
70     list.add(new Employee("Anna", 2016));
71     list.add(new Employee("Jack", 1999));
72     list.add(new Employee("Mark", 1995));
73
74     List<Employee> expectedList = new ArrayList<>();
75     expectedList.add(new Employee("John", 2020));
76     expectedList.add(new Employee("Anna", 2016));
77     expectedList.add(new Employee("Jack", 1999));
78     expectedList.add(new Employee("Mark", 1995));
79
80     List<Employee> result = Main.selectEmployeeForBonus(list, 7);
81     Assertions.assertTrue(isEqualsIgnoreOrder(expectedList,result),
82         String.format("%nExpected: %s%nActual:   %s%n", expectedList, result));
83
84     //Assertions.assertEquals(expectedList, result);
85 }
86
87
88 @Test
89 @DisplayName("createSortedCopy: by year")
90 void createSortedCopy() {
91     List<Employee> list = new ArrayList<>();
92     list.add(new Employee("John", 2020));
93     list.add(new Employee("Anna", 2016));
94     list.add(new Employee("Jack", 1999));
95     list.add(new Employee("Mark", 1995));
96
97     List<Employee> expected = new ArrayList<>();
98     expected.add(new Employee("Mark", 1995));
99     expected.add(new Employee("Jack", 1999));
100    expected.add(new Employee("Anna", 2016));
101    expected.add(new Employee("John", 2020));
```

```
101     expected.add(new Employee( John , 2020));
102
103
104     List<Employee> result = Main.createSortedCopy(list, new ComparatorEmployeeByYear());
105
106     Assertions.assertEquals(expected, result);
107     Assertions.assertFalse(result==list); // проверяем, что листы разные
108
109
110 }
111
112 ✓ private boolean isEqualIgnoreOrder(List<Employee> list1,List<Employee> list2){
113     return     list1!=null
114         && list2!=null
115         && list1.size() == list2.size()
116         && list1.containsAll(list2);
117 }
118
119 }
```