DAT first semester exam 2

Cherry Rose Semeña & Emmely Lundberg

Continuing your work from last week, this week you will first of all improve your code (if it needs improving) so that it is easily testable - Expect heavy use of dependency injection. Last week you wrote 2 methods (at least) in java.

- This week you have to write 2 methods using Hamcrest or similar ("Fluent assertions for .NET). If you have used another language and can't find a library that is similar, you can always use Hamcrest on the 2 methods you wrote in java.
- Also you have to write 2 test methods that are "data driven" - No hard coded data! Your data-set doesn't have to be large, but it has to be representative of a proper set.

SOLUTION:

Source code can be found at https://github.com/cph-cs241/TEST_Assignment4_UnitTesting.

Overview of the program:

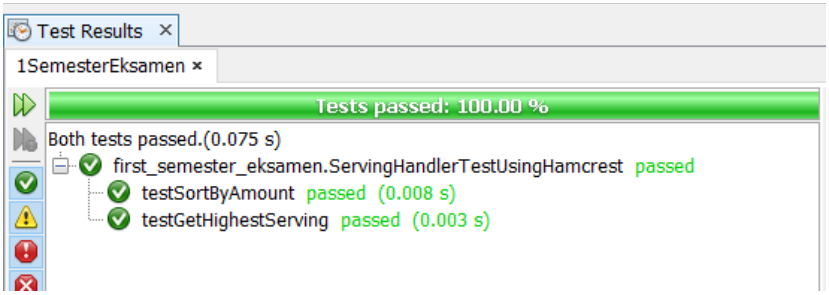
There is a data set for wine servings. An interface with some methods was provided to manage the data given for making inventories and determining how much wine has been served too much/less and by whom(waiter), as well as sorting it by amount of servings. Find the Servings data in Servings.csv.

- ***Testing 2 methods using Hamcrest:***

Hamcrest is very useful in creating matcher objects which is often used in automated testing. It is designed to make the tests readable by using static methods that creates an assertion grammar. In our solution, you find it easy to understand by simply reading it, like “Assert that result is any Serving class”, “Assert that result’s amount is equal to the expected result”, “Assert that first is less than or equal to next”. There are some more matchers that can be used such as anything(), arrayContaining(),startsWith(), endsWith(), arrayContainingInAnyOrder(), and so on.

```
@Test
public void testGetHighestServing() {
    System.out.println("getHighestServing");
    int expResult = 157;
    Serving result = instance.getHighestServing(servings);
    assertThat(result, is(any(Serving.class)));
    assertThat(result.getAmount(), is(equalTo(expResult)));
}

@Test
public void testSortByAmount(){
    System.out.println("sortByAmount");
    instance.sortByAmount(servings);
    for (int i = 0; i < servings.size()-1; i++) {
        int first = servings.get(i).getAmount();
        int next = servings.get(i+1).getAmount();
        assertThat(first, is(lessThanOrEqualTo(next))); //lessThan is only working on lower version of hamcrest 1.2
    }
}
```

Test Result:



- ***Data Driven Test for 2 methods:***

We have chosen Excel documents (.xls) as the source for the data driven test. The reasoning behind choosing Excel documents are formats are because that non technical person can read/write which increases the number of people that contribute to the test data in a project. The code and inspiration for reading the Excel sheet has been taken from this article. Inspiration: https://dzone.com/articles/data-driven-tests-junit-4-and.

Excerpts from the article:

"Data-driven testing is a great way to test calculation-based applications more thoroughly. In a real-world application, this Excel spreadsheet could be provided by the client or the end-user with the business logic encoded within the spreadsheet."

"The SpreadsheetData class uses the Apache POI project to load data from an Excel spreadsheet and transform it into a list of Object arrays compatible with the @Parameters annotation"

This is the code for testing getTotalExcessServings with data driven test:

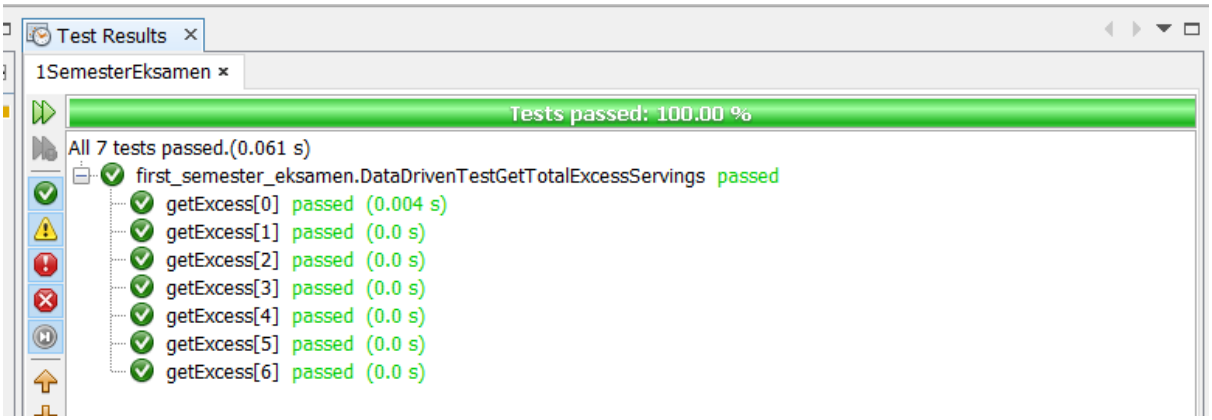
Data from Excel sheet:

amount1	amount2	amount3	result
167	166	165	3
167	165	164	2
0	-166	-166	0
-1	0	1	0
-1	0	165	0
-1	0	169	4
166	165	-200	1

Test:

```
/** @author Cherry Rose Semeña & Emmely Lundberg*/
@RunWith(Parameterized.class)
public class DataDrivenTestGetTotalExcessServings {


    private int amount1;
    private int amount2;
    private int amount3;
    private int expResult;

    @Parameterized.Parameters
    public static Collection spreadsheetData() throws IOException {
        InputStream spreadsheet = new FileInputStream("gettotalexcessservings.xls");
        return new SpreadsheetData(spreadsheet).getData();
    }
    /** Inspiration: https://dzone.com/articles/data-driven-tests-junit-4-and */
    public DataDrivenTestGetTotalExcessServings(double amount1, double amount2, double amount3,  double expResult) {
        super();
        this.amount1 = (int) amount1;
        this.amount2 = (int) amount2;
        this.amount3 = (int) amount3;
        this.expResult = (int) expResult;
    }

    @Test
    public void getExcess() {
        ServingHandler instance = new ServingHandlerImpl();
        ArrayList<Serving> servings = new ArrayList<Serving>();
        Time t = new Time("10:01");
        servings.add(new ServingImpl("24-12-2017", t, this.amount1, "André"));
        servings.add(new ServingImpl("24-12-2017", t, this.amount2, "Sofia"));
        servings.add(new ServingImpl("24-12-2017", t, this.amount3, "Paula"));

        int result = instance.getTotalExcessServings(servings);
        assertEquals(this.expResult, result);
    }
}
```

Test Result:



This is the code for testing getValidServings with data driven test:

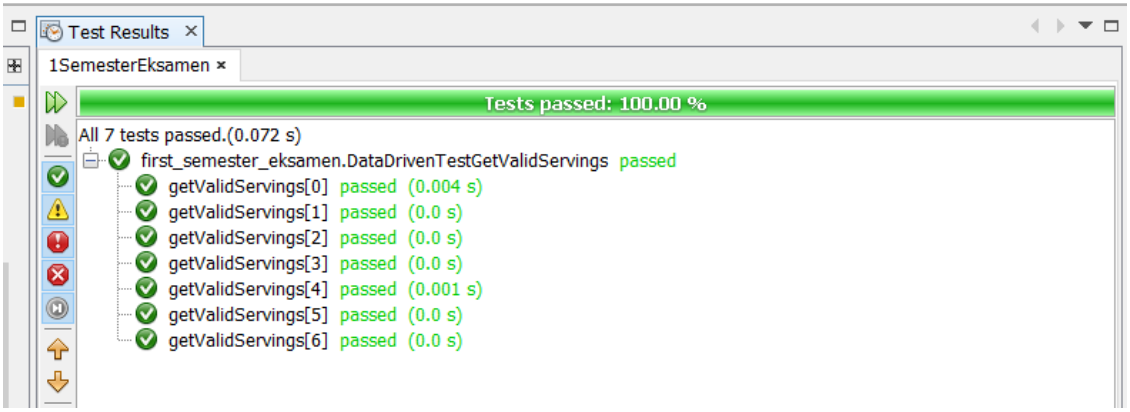
Data from Excel sheet:

amount1	amount2	amount3	result
167	166	165	1
167	165	164	2
0	-166	-166	0
-1	0	1	0
-1	0	165	1
-1	0	169	0
166	165	-200	1

Test:

```java
/** @author Cherry Rose Semeña & Emmely Lundberg*/
@RunWith(Parameterized.class)
public class DataDrivenTestGetValidServings {

    private int amount1;
    private int amount2;
    private int amount3;
    private int expResult;

    @Parameterized.Parameters
    public static Collection spreadsheetData() throws IOException {
        InputStream spreadsheet = new FileInputStream("gettotalvalid.xls");
        return new SpreadsheetData(spreadsheet).getData();
    }
    /** Inspiration: https://dzone.com/articles/data-driven-tests-junit-4-and */
    public DataDrivenTestGetValidServings(double amount1, double amount2, double amount3,  double expResult) {
        super();
        this.amount1 = (int) amount1;
        this.amount2 = (int) amount2;
        this.amount3 = (int) amount3;
        this.expResult = (int) expResult;
    }

    @Test
    public void getValidServings() {
        ServingHandler instance = new ServingHandlerImpl();
        ArrayList<Serving> servings = new ArrayList<Serving>();
        Time t = new Time("10:01");
        servings.add(new ServingImpl("24-12-2017", t, this.amount1, "André"));
        servings.add(new ServingImpl("24-12-2017", t, this.amount2, "Sofia"));
        servings.add(new ServingImpl("24-12-2017", t, this.amount3, "Paula"));

        int result = instance.getValidServings(servings).size();
        assertEquals(this.expResult, result);
    }
}
```

Test Result:


Test Results
1SemesterEksamen
Tests passed: 100.00 %
All 7 tests passed.(0.072 s)
first_semester_eksamen.DataDrivenTestGetValidServings passed
getValidServings[0] passed (0.004 s)
getValidServings[1] passed (0.0 s)
getValidServings[2] passed (0.0 s)
getValidServings[3] passed (0.0 s)
getValidServings[4] passed (0.001 s)
getValidServings[5] passed (0.0 s)
getValidServings[6] passed (0.0 s)