Exercise 7: State-based Testing

The objective of this exercise is the use of methods for state-based testing to design test cases.

7.1 State-based Testing (2 points)

The hand-outs for the lecture contain a state diagram for a Stack and a receipt for systematically deriving test cases. Use the state diagram for Stack as a starting point and draw a similar state diagram for the *RingBuffer* excluding – for now – the class *RingBufferIterator*. Derive test cases from this diagram according to the receipt in the lecture notes and implement these test cases using JUnit.

Consider following hints:

- ☐ Start with sketching a state diagram for *RingBuffer* (excluding *RingBufferIterator*). Submit this diagram as *StateDiagramRingBuffer.png* (or *.gif*).
- ☐ Design the test cases by drawing a "transition tree" for the state diagram (*TransitionTreeRingBuffer.png*). List all derived test cases in a text file (*RingBufferTests.txt*) like shown in the lecture notes.
- ☐ Add the method *isFull()* to the class *RingBuffer* to expose the internal state. (Note: The state is *filled*, if *RingBuffer* is neither empty nor full.)
- ☐ Create the class `RingBufferStatesTest` for the JUnit test cases. The test cases should check (1) the state of the RingBuffer after each transition and (2) the results returned by the different methods.
- ☐ Put all files in the project SQE07-*Lastname*_RingBufferStatesTest; *Lastname* should be replaced by your name.
- ☐ **Submission:** Submit (1) the **.png* and **.txt* files from test design, (2) the source code of all test classes and (3) the modified source code of *RingBuffer* containing the method *isFull()*.

7.2 State-based Testing II (3 points)

RingBuffer includes the iterator *RingBufferIterator*, which depends on the size of *RingBuffer*. Use *State Testing* to design test cases, implement these tests with JUnit and add them to the test cases you have written in the previous example.

Draw a state diagram for *RingBufferIterator*. Derive test cases from this diagram according to the receipt in the lecture notes and implement these test cases using JUnit.

Consider following hints:

- ☐ Start with a state diagram for *RingBufferIterator* (*StateDiagramIterator.png*). Consider the iterator's dependencies on *RingBuffer*!

- ☐ Design the test cases by drawing a "transition tree" for the state diagram (*TransitionTreeIterator.png*) and list all derived test cases in *IteratorTests.txt.*
- ☐ Implement test cases in the class *RingBufferIteratorStatesTest.java*. The test cases should check (1) the state after each transition and (2) the results returned by the different methods.
- ☐ **Submission:** Submit (1) the **.png* and **.txt* files from test design and (2) the source code of all tests other modified classes.

Exercise 8: Black-box Test Design and Parameterized Tests

The objective of this exercise is the application of equivalence partitioning and boundary value analysis in combination with test automation using parameterized tests.

8.1 Equivalence Partitioning and Boundary Value Analysis (3 points)

Equivalence partitioning and boundary value analysis are simple black-box test design techniques. They are used to split the range of input values into partitions so that all inputs of a partition show an equivalent behavior or result. The representative input values for each equivalence partition are selected from the boundaries of the partition.

Conduct (a) an equivalence partitioning and (b) a boundary value analysis for the class *Triangle*. Use the results to create test cases for the method *isValid()* of the class *Triangle.* Specify these test cases in form of a table:

TriangleTests.txt			
a	b	c	isValid()
1	1	1	true
...	...	...	...

Consider following hints:

- ☐ Derive the equivalence partitions from the distinguished types of the triangle based on the method *isValid().*
- ☐ In addition, consider all invalid equivalence partitions.
- ☐ Take the range of the data type (e.g., *Integer*) into account when conducting the boundary value analysis.
- ☐ Define the test cases by combining all identified representative values for the equivalence partitions according to the combination rules. Avoid any unnecessary (i.e., redundant) test cases.
- ☐ Store the test cases in the text file *TriangleTests.txt*; separate the values of the different columns by tabulator or comma.
- ☐ **Submission:** Submit a text file containing the tests.

8.2 Parameterized Tests(2 points)

When test cases should be repeated several times, each time with different values, the JUnit specialized runner `Parameterized` can be used. This special feature has been introduced since JUnit 4 - http://junit.sourceforge.net/javadoc/org/junit/runners/Parameterized.html

In this exercise you should combine the concepts from the previous exercises with JUnit to create parameterized tests. Additional details on parameterized JUnit test cases can be found in the tutorial at http://www.tutorialspoint.com/junit/pdf/junit_parameterized_test.pdf.

Create a parameterized JUnit 4 test case *TriangleParameterizedTest* that takes a list of parameters consisting of (1) the input values for the three sides and (2) the expected result of *isValid()*. The test case should conduct the check for validity and determine whether the obtained actual result is in equal to the expected result.

Consider following hints:

- ☐ Submit the source code of the test case *TriangleParameterizedTest.java* plus any other source files you created.
- ☐ Put all files in the project SQE08-*Lastname*_TriangleParameterizedTest; *Lastname* should be replaced by your name.
- ☐ **Submission:** Submit the source code of all tests and any other required classes.

General Submission Instructions

- Add a comment at the beginning of every source code file containing your full name and your student ID number.
- Submit the complete Eclipse project including all relevant additional files as specified via packing in a Zip archive. Adhere to the naming convention for projects: SQE##-*Lastname_Title* (## = exercise number, *Lastname* = your name, *Title* = exercise title).
- Upload the archive at the specified submission link on the elearning platform http://elearning.fh-hagenberg.at/ until the specified date and time.