Preparation

Download links and other details can be found via the course web page:

`http://www.win.tue.nl/~wstomv/edu/sc-hse/`

1. In the book *Introduction to Programming Using Java* by David Eck, read §3.7, 8.3, and 8.4.1.
2. Review the slides of Lecture 5 (initially, you can skip slides 14, 17–19, 22–31, 37).
3. Review the robust code for `SimpleClass.divide()` in the NetBeans project `JUnit4Examples.zip` of Series 4, and how the exception is tested in `SimpleClassTest`.

Assignments

1. Improve your work for *Count Digits with Radix* and *Powerize*, where applicable. In particular, pay attention to
   - Coding Standard
   - Javadoc Contracts (pre/post/return)
   - Design: Functional Decomposition (especially with `powerize()`)
   - Functional Correctness and Performance
   - JUnit Test Cases (boundary cases, code coverage)
2. (a) Compare your method contracts for `DiceGameDecomposed` (Series 3) to our contracts (see the downloads for Series 5).

   (b) Make the following methods in your `DiceGameDecomposed` as robust as reasonably possible:
   - `simulate()`
   - `roll()`
   - `max()`
   - `find()`

   Use `IllegalArgumentException`, `NullPointerException` as appropriate.

   For `find()`, you can put the throwing of `IllegalArgumentException` somewhere inside the "algorithm", rather than before it.

   For example, you can use a **`try`** statement to wrap the indexing in array `a`, catch a possible `IndexOutOfBoundsException` and then throw `IllegalArgumentException`.

© 2014, TUE.NL

(c) Extend the given `DiceGameDecomposedTest` with test cases to check robustness, that is, for proper exception throwing, of the methods above.

3. Read the documentation for class `Statistics`. For example, in DrJava, use Tools > Javadoc > Preview Javadoc for Current Document, or in the list of open documents, right-click on the file and select Preview Javadoc for File.

   All methods, except `update()` have been implemented.

   (a) Design and implement some JUnit test cases in `StatisticsTest`. For testing of `getMean()` use

   ```
   static void assertEquals(String message,
           double expected, double actual, double EPSILON)
   ```

   with `EPSILON` sufficently small, e.g., 10^{-18}. This method is predefined in the JUnit testing framework.

   Note that you cannot test each method in complete isolation. You need to test them together. In particular, `update()` will be tested implicitly when testing `reset()` and the queries `getXxx()`. Start with testing `reset()`, using `update()` and `getCount()`. All other tests should start with `SUT.reset()`.

   (b) Try the test cases on the incomplete `Statistics` as given. Observe the failures caused by the missing implementation of `update()`.

   (c) Design and implement the method `update()`, and test the implementation.

   Do *not* add global variables in the class.

   It might seem to be easier to maintain the *sum of the values*, rather than the *mean*. However, the sum could easily grow to cause an overflow (make sure you include a test case for that). Such an overflow is avoided by maintaining the mean.

   Later we shall add the standard deviation.

   (d) Add test cases to check the robustness.

   Make the methods in `Statistics` as robust as possible, throwing `IllegalStateException` when the precondition fails. Finally, test your robust version of `Statistics`.

Submit your work for the second and third assignments to peach[3]. Submit both the class and the test cases.

Deadline: Thursday 9 October 23:00 (Moscow time)

© 2014, TUE.NL