

## TEST-DRIVEN DESIGN IN JAVA FOR NOVICE PROGRAMMERS

Viera K. Proulx  
College of Computer and Information Science  
Northeastern University  
Boston, MA 02115  
617-373-2225  
vkp@ccs.neu.edu

The tutorial will show how testing and test-first design can be incorporated into every introductory course on object-oriented programming using Java. We will show the syntax for the tests that use our novice-friendly *tester* library, and describe the pedagogy we use to enforce test-first design from the very first program students write. The *tester* library is freely available on our web site with extensive documentation that shows how to use it with Eclipse, NetBeans, BlueJ, or just command-line Java compilation.

Test-driven design or test-first design has been used in software industry for years. The reports on using this approach reveal faster development time, more robust program design, and a more readable code --- divided into shorter methods and smaller classes. On the other hand, many reports on catastrophic failures caused by software flaws point out to a small overlooked error in the program design. We all have experienced the times when after struggling with a bug in our code we find the problem in a small method that *just cannot be wrong*.

One would think that all courses on introductory computing would adopt this beneficial test-driven approach. However, this is not the case. A survey of over twenty current textbooks shows only two that attempt to require tests for the methods students design. The rest of them include a perfunctory section that proclaims how important testing is --- with no attempt to include it in the suggested curriculum.

One of the difficulties in enforcing test-first design for novices is the steep learning curve for use of standard libraries for testing. Students struggling with the syntax-heavy programming language (Java) are not able to digest the extra burden of designing tests in a separate environment (such as JUnit), especially if it requires that they override the `equals` method before they even understand what are the different kinds of equality between Java objects.

Over the past five years our introductory computing courses enforced test-first design from the very beginning. To make this possible we have designed a supporting

test library, *tester*, that enables the novice programmer to define all examples of data and all tests in an *Examples* class that represents the client for the code that the student has designed. There is no new syntax, no need to define equality, as all objects are compared for the value of their fields. *Tester* evaluates all tests defined in the *Examples* class, pretty-prints the values of all data defined in the *Examples* class, and reports on all failed tests with a display of both the actual and the expected values, as well as a link to the failed test. Additionally, there is a support for checking whether the actual value matches one of several expected ones (e.g a random number being one of 1, 2, 3, or 4), or whether the given value is within the given range (as determined by a `Comparator` or an implementation of the `Comparable` interface). There is also a support for detecting whether a method when invoked by the given object and the given argument list throws the expected exception --- with the expected message.

Our curriculum focuses initially on mutation-free program design: the result of every function or method is a new piece of data. This makes the test design easier, as all we need to do is to compare the actual outcome with the expected one. However, the tests for imperative methods (that produce `void`) are not much harder to design and will be covered in the tutorial.

No special materials, other than an overview handout, will be provided for the participants, as all of our course materials, the *tester* library, and a wealth of labs, examples, assignments, and lecture notes is already available on our web pages.

## **PRESENTER'S BACKGROUND**

Viera Proulx is a Professor in College of Computer and Information Science at Northeastern University in Boston, MA. She has been involved in curriculum and software development for introductory computing for nearly 20 years. During the past six years she has been developing and implementing curriculum for data-driven class based introductory course on object-oriented programming, working with the TeachScheme/ReachJava team. The draft of the textbook *How to Design Classes* that supports this curriculum is nearly complete.

Professor Proulx has led faculty development workshops and several conference workshops on the ReachJava curriculum. Currently the ReachJava team has an NSF grant that supports faculty development workshops at four locations (Utah, California, New York, and Massachusetts) with next round of workshops planned for the summer 2009.

## REFERENCES

- [1] Proulx, V. K., *Test-Driven Design for Introductory OO Programming*. SIGCSE Bulletin, 41(1) 2009 (to appear).
- [2] Proulx, V. K., and Gray, K. E., *Design of Class Hierarchies: An Introduction to OO Program Design*. SIGCSE Bulletin, 38 (1), 2006.