# Unit Testing

## 1 Overview

This lab provides an introduction to unit testing in Java using the popular JUnit library. We will be creating two unit testing classes to test classes designed earlier in the semester. We will be applying unit tests to our previous Sudoku class and a provided LightsOut class. We will use unit testing to ensure the perfect operation of our Sudoku class and find the existing bugs in the provided LightsOut class

## 2 Learning Outcomes

**By the end of this project students should be able to:**

- Write unit tests to validate components;
- Utilize java packages;
- work effectively with a partner using pair-programming;
- write an effective report that describes the students' problem solving process.

## 3 Pre-Lab Instructions

**Do this part before you come to lab:**

This project will require you to use java packages and the JUnit unit testing framework

- Read Big Java chapters 8.5 and the Vogella JUnit tutorial through section 6. (http://www.vogella.com/tutorials/JUnit/article.html)
- In order to unit test your code it will have be in a package. Explain how to create packages and populate those packages with .java files.
- JUnit is not part of the Java class library. Download the JUnit and Hamcrest jar files from the JUnit website and explain how to successfully compile programs using the JUnit library.

# 4 Lab Instructions

**Do this part in lab:**

### Step 1

First obtain the code from your previous Sudoku project. You can use the Sudoku code from any group member in your pair or team. You should be able to retrieve code submitted through BBLearn. Create a project folder and create two packages, one for Sudoku and one for LightsOut. The lights out package must be called simply 'lightsout'. Compile these packages and ensure that you are able to get the code working as well as it was during the previous project. It is alright if there are unmet requirements, as this unit testing project will prompt you to correct mistakes as we develop tests.

### Step 2

Follow the Vogella tutorial to start building a unit test suite for Sudoku. As you implement tests, run those tests. If they don't pass, correct your code until they do. If they do pass, make a modification to your original code so you can see the failure, and then correct the code to see the success. For each unit test run in this way, save a screen shot of its failure state and one screen shot where no tests fail at the end. For the LightsOut class you are unable to change the code, but should implement all the tests provided. There are two bugs built into the class file, and your final screen shot should demonstrate that these bugs have been found by your unit test.

### Sudoku

For Sudoku you will create at least one unit test for every function, and multiple tests for functions that have multiple requirements. In addition to the original requirements of the project, we will now include a requirement that a Sudoku puzzle should never be allowed to contain an invalid character. You must create a test for each of the following requirements.

- (new) Instantiating Sudoku puzzles with invalid characters or layouts should throw an IllegalArgumentException
- getSquare should return the correct character for a given location
- setSquare should set correctly for a given location
- (new) setSquare should throw an IllegalArgumentException if it is passed an invalid character
- isValid should return false if there are any repeated numbers in a row
- isValid should return false if there are any repeated numbers in a column

- isValid should return false if there are any repeated numbers in any of the nine 3x3 subsquares
- isValid should return true if no earlier rules are violated
- isSolved should reutrn false if there are any blank spaces
- isSolved should return true if no earlier solve or validity rules are violated

### LightsOut

For testing the provided LightsOut class you must verify the following requirements:

- LightsOut can be instantiated at a size of one or greater. All lights should start unlit.
- getSize should return the same size provided to the constructor.
- press should toggle the lights at the location pressed and at the four adjacent squares.
- press should throw an index out of bounds exception should it be used outside the range 0 to getSize()-1 for both rows and columns.
- press should **not** throw an exception just because one of the adjacent squares is out of bounds.
- forceLit should change the lit state of the location properly.

## 5 Lab Report

**Each pair of students will write a single lab report together and each student will turn in that same lab report on BBLearn. Submissions from each student on a pair should be identical.**

Your lab report should begin with a preamble that contains:

- The lab assignment number and name
- Your name(s)
- The date
- The lab section

It should then be followed by four numbered sections:

### 1. Problem Statement

In this section you should describe the problem in **your** own words. The problem statement should answer questions like:

- What are the important features of the problem?
- What are the problem requirements?

This section should also include a reasonably complete list of requirements in the assignment. Following your description of the problem, include a bulleted list of specific features to implement. If there are any specific funtions, classes or numeric requirements given to you, they should be represented in this bulleted list.

### 2. Planning

In the second section you should describe what planning you did in order to solve the problem. You should include planning artifacts like sketches, diagrams, or pseudocode you may have used. You should also describe your planning process. List the specific data structures or techniques you plan on using, and why.

### 3. Implementation and Testing

In the third section you should describe how you implemented your plan. As directed by the lab instructor you should (as appropriate) include:

- a copy of your source code (Submitted in BBLearn as .java files)
- a screen shot of your running application / solution
- results from testing

### 4. Reflection

In the last section you should reflect on the project. Consider different things you could have done to make your solution better. This might include code organization improvements, design improvements, etc.

You should also ask yourself what were the key insights or features of your solution? Were there alternative approaches or techniques you could have employed? How would these alternatives have impacted a different solution?

### 5. Partner Rating

Every assignment you are required to rate your partner with a score -1, 0 or +1. This should be submitted in the comment section of the BBLearn submission, and not in the report document. If you don't want to give your partner a negative rating making sure not to use a dash before listing the number! You do not have to tell your partner the rating you assign them. A rating of 1 indicates

that your partner was particularly helpful or contributed exceptional effort. A rating of 0 indcates that your partner met the class expectations of them. Rating your partner at -1 means that they refused contribute to the project, failed to put in a resonable effort or actively blocked you from participating. If a student recieves three ratings of -1 they must attend a mandatory meeting with the instructor to dicuss the situation, and recieving additional -1 ratings beyond that, the student risks losing a letter grade, or even failing the course.

## Colophon

This project was developed by Richard Lester and Dr. James Dean Palmer of Northern Arizona University. Except as otherwise noted, the content of this document is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.