

# Project 5: Mutation Testing for BST

**DUE: Apr. 25th at 11:59pm**

**Extra Credit Available for Early Submissions!**

## Setup

- Download the **p5.zip** and unzip it. This will create a folder **section-yourGMUUserName-p5**.
- Rename the folder in the same way as previous projects. After renaming, your folder should be named something like: **DL1-krusselc-p5**.
- Complete the **readme.txt** file (an example file is included: **exampleReadmeFile.txt**).

## Submission Instructions

- Same as on previous projects!
- Submit to blackboard. **DOWNLOAD AND VERIFY WHAT YOU HAVE UPLOADED THE RIGHT THING.**  
*Submitting the wrong files will result in a 0 on the assignment!*

## Basic Procedures

You must:

- Have code that compiles with the command: **javac -cp .;../junit-4.11.jar \*.java** (Windows) or **javac -cp .:../junit-4.11.jar \*.java** (Linux/MacOS) in your user directory.
- Have code that runs with the command: **java -cp .;../junit-4.11.jar MyTestCases** (Windows) or **java -cp .:../junit-4.11.jar MyTestCases** (Linux/MacOS) in your user directory.

You may:

- Add as many public methods as you want to **MyTestCases.java** but you may not alter any other file.

You may NOT:

- Make your program part of a package.
- Add any additional libraries/packages which require downloading from the internet.
- Use any code from the internet
- Import any additional libraries/packages or add any additional import statements (or use the “fully qualified name” to get around adding import statements). So you cannot use any built in Java Collections Framework classes that are not already imported for you.
- Alter any method/class signatures defined in this document of the template code. Note: “throws” is part of the method signature in Java, don’t add/remove these.
- Alter the provided classes (e.g. **BinaryNode**, **BinarySearchTree**, **BinaryTreeIterator**, **DuplicateItemException**, **ItemNotFoundException**).
- Add **@SuppressWarnings** to any methods unless they are private helper methods for use with a method we provided which already has an **@SuppressWarnings** on it.

## Grading Rubric

### No Credit

- Same as previous projects! (Non submitted assignments. Assignments submitted more than 48 hours late. Non-compiling assignments. Non-independent work. Code that violates and restrictions or “you may not” mandates. "Hard coded" solutions. Code that would win an obfuscated code competition with the rest of CS310 students.)

### How will my assignment be graded?

- Automatic Testing (100%): To assess the correctness of programs.
- You CANNOT get points for code that doesn't compile or for submitting just the files given to you.
- Extra credit for early submissions:
  - 1% extra credit rewarded for every 24 hours your submission made before the due time
  - Up to 5% extra credit will be rewarded
  - Your latest submission before the due time will be used for grading and extra credit checking. You CANNOT choose which one counts.

### Automated Testing Rubric

Your score will be proportional to the number of mutants that your tester will manage to kill. When you run the mutation testing the report you receive concludes with the following lines:

```
=====
- Statistics
=====
>> Generated 152 mutations Killed XX (YY%)
```

The percentage YY% will be your grade in this assignment. The more mutants your tester can kill the higher your grade.

## Overview

The goal of this project is to get you thinking about data structures from a different point of view: from the testing perspective. By learning how to test your data structure code and experimenting with the implementation of unit tests, you will develop a better understanding of how data structures work and what are the pitfalls.

In this project we'll be using a testing technique named "mutation testing". The idea is that you modify a program in small ways and then you check if your tester can catch the error. Each mutated version of your code is called a mutant and the tests detect and reject mutants by causing the behavior of the original version to differ from the mutant. This is called killing the mutant. Test suites are measured by the percentage of mutants that they kill. The purpose is to develop effective tests and identify weaknesses in the test data or in sections of the code that are seldom or never accessed during execution. Mutation testing is a form of white-box testing. Source: [https://en.wikipedia.org/wiki/Mutation\\_testing](https://en.wikipedia.org/wiki/Mutation_testing)

So, we're going to simulate a novice programmer that makes many errors in their code by using some very simple mutation software named [Pitest](#) (it supports only a few "mutation operators"). If you're interested in real mutation testing, you might want to check out [muJava](#) which has a very nice Eclipse plugin and is very powerful.

## Task

Start by writing 3-4 JUnit tests in **MyTestCases.java** to see if the code does what it claims. A template for JUnit tests has been provided, but we recommend that you check out the JUnit tests provided on P0 for examples as well as the lecture notes on JUnit from CS 211. The following is the *anatomy* of a single JUnit test:

```
/**
 * Make a peanut and put it in the box. See if
 * you can get it back from the box.
 */
@Test(timeout = 2000)
public void testGetItem() {
    //make an peanut
    Peanut p = new Peanut();

    //put the peanut in a box
    SingleItemBox<Peanut> box = new SingleItemBox<>(p);

    //check that the peanut was put in the box
    assertEquals("Unable to put an item in and get it back.", p, box.getItem());
}
```

Good comment explaining the test, so that you remember.

Test annotation and timeout (in case of infinite loop)

Good, meaningful, test name.

Setup (make a peanut)

Scenario (put item in box)

Check what we wanted to happen did happen.

Expected Value    Actual Value

The provided implementation should pass all your test cases. Once you think you've written some good JUnit tests and your tests are compiled and run correctly, it's time to test your tests. In your project folder (not your user directory!) should be three jar files: **pitest.jar**, **pitest-entry.jar** and **pitest-cmd.jar**. These jar files contain code that performs extremely simple mutation testing. From the command line in your user directory, you should be able to run the following command:

```
java -cp ../junit-4.11.jar;../pitest.jar;../pitest-cmd.jar;../pitest-entry.jar
org.pitest.mutationtest.commandline.MutationCoverageReport
--reportDir output
--targetClasses BinarySearchTree,BinaryTreeIterator
--targetTests MyTestCases
--sourceDirs .
--timestampedReports=false
--mutators DEFAULTS,REMOVE_CONDITIONALS,INLINE_CONSTS,CONSTRUCTOR_CALLS,NON_VOID_METHOD_CALLS,REMOVE_INCREMENTS
```

**Note:** There aren't supposed to be any line breaks in the above command, those are just for displaying in this PDF; the command should be given all on one line. If you're curious about what the command actually means, see: <http://pitest.org/quickstart/commandline/>

This will (a) create a set of mutants, (b) run your JUnit tests on each mutant, and then (c) generate a report about whether or not your JUnit tests noticed that the mutant had a "typo" in it. The report will be printed to the console, but you'll also find a directory called "output" which contains an **index.html** page you can open. You're looking to "kill" the mutants (in other words, your tests identify the typo as a mistake). Both the console output and the HTML report will give you the details of how many mutants your tests killed.

In the console output that is the lines:

```
>> Generated 152 mutations Killed 0 (0%)
>> Ran 0 tests (0 tests per mutation)
```

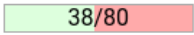
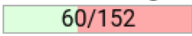
**If you don't see "152 mutations" you did something wrong with your command!**

In the HTML report, you can see the “Breakdown by class” and click on a specific class to get detailed information about the mutants you covered:

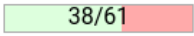
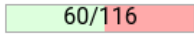
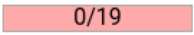
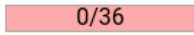
## Pit Test Coverage Report

### Package Summary

#### default

Number of Classes	Line Coverage	Mutation Coverage
2	48% 	39% 

#### Breakdown by Class

Name	Line Coverage	Mutation Coverage
<a href="#">BinarySearchTree.java</a>	62% 	52% 
<a href="#">BinaryTreeIterator.java</a>	0% 	0% 

---

Report generated by [PIT](#) 1.5.2

## Summary/Reference Instructions

The following commands assume Windows. Use : instead of ; if in Linux/Mac

COMPILE:

```
javac -cp .;../junit-4.11.jar *.java
```

RUN:

```
java -cp .;../junit-4.11.jar MyTestCases
```

MUTATION:

```
java -cp .;../junit-4.11.jar;../pitest.jar;../pitest-cmd.jar;../pitest-entry.jar
org.pitest.mutationtest.commandline.MutationCoverageReport
--reportDir output --targetClasses BinarySearchTree,BinaryTreeIterator
--targetTests MyTestCases --sourceDirs . --timestampedReports=false
--mutators DEFAULTS,REMOVE_CONDITIONALS,INLINE_CONSTS,CONSTRUCTOR_CALLS,
NON_VOID_METHOD_CALLS,REMOVE_INCREMENTS
```

[NO NEW LINES OR SPACES AFTER COMMAS IN THE ABOVE COMMAND!]

### Steps to follow:

- 1) Write tests in **MyTestCases.java**
- 2) Compile (first command above)
- 3) Make sure all tests pass on the working code (second command above)
- 4) Run mutation tester (third command above). It will output a lot of things with this at the end:

```
=====
- Statistics
=====
>> Generated 152 mutations Killed XX (YY%)
>> Ran ZZ tests (Z tests per mutation)
```

YY% is your grade. If you don't see "152 mutations" you did something wrong with your command (and no, you won't get the grade your expecting if you make the assignment easier by messing up the command).

If you see this: **Exception in thread "main" org.pitest.help.PitHelpError: X tests did not pass without mutation when calculating line coverage. Mutation testing requires a green suite.** You didn't check that the working code could pass the tests! Go back to Step 3.

- 5) If you didn't kill everything, write more tests, but also look in the "output" folder at **index.html**. If you click on the class name you can see the mutations it made (and line numbers), then go look at class to see how that would affect the Binary Search Tree and/or Iterator.

### LINKS:

- All the Assert options for JUnit: <https://junit.org/junit4/javadoc/latest/org/junit/Assert.html>
- More information on what each mutation does: <https://pitest.org/quickstart/mutators/>
- More information on JUnit: <https://junit.org/junit4/>
- More command line options for PITest: <https://pitest.org/quickstart/commandline/>