Com S 417
Software Testing

Fall 2017 – Week 8, Lecture 13

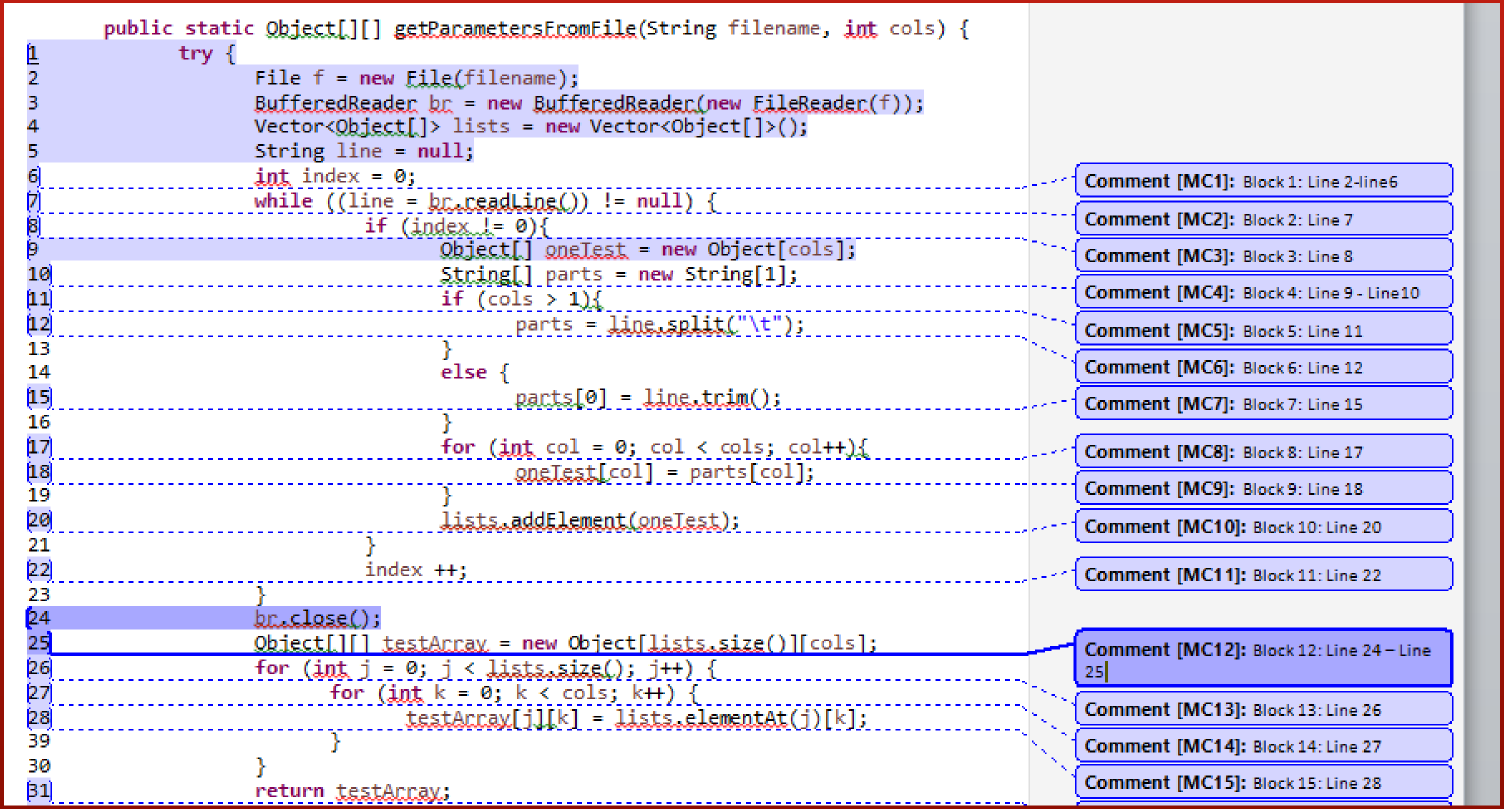
Announcements

- Lab 3 is available and due Oct. 17.
- We *will* have 5 labs.

Topics

- In Class exercise
- Introduction to Mockito
- TDD (Test Driven Development)

Capturing Basic Blocks



```
        public static Object[][] getParametersFromFile(String filename, int cols) {
1           try {
2               File f = new File(filename);
3               BufferedReader br = new BufferedReader(new FileReader(f));
4               Vector<Object[]> lists = new Vector<Object[]>();
5               String line = null;
6               int index = 0;
7               while ((line = br.readLine()) != null) {
8                   if (index != 0){
9                       Object[] oneTest = new Object[cols];
10                      String[] parts = new String[1];
11                      if (cols > 1){
12                          parts = line.split("\t");
13                      }
14                      else {
15                          parts[0] = line.trim();
16                      }
17                      for (int col = 0; col < cols; col++){
18                          oneTest[col] = parts[col];
19                      }
20                      lists.addElement(oneTest);
21                  }
22                  index ++;
23              }
24              br.close();
25              Object[][] testArray = new Object[lists.size()][cols];
26              for (int j = 0; j < lists.size(); j++) {
27                  for (int k = 0; k < cols; k++) {
28                      testArray[j][k] = lists.elementAt(j)[k];
39                  }
30              }
31              return testArray;
```

Comment [MC1]: Block 1: Line 2-line6

Comment [MC2]: Block 2: Line 7

Comment [MC3]: Block 3: Line 8

Comment [MC4]: Block 4: Line 9 - Line 10

Comment [MC5]: Block 5: Line 11

Comment [MC6]: Block 6: Line 12

Comment [MC7]: Block 7: Line 15

Comment [MC8]: Block 8: Line 17

Comment [MC9]: Block 9: Line 18

Comment [MC10]: Block 10: Line 20

Comment [MC11]: Block 11: Line 22

Comment [MC12]: Block 12: Line 24 – Line 25

Comment [MC13]: Block 13: Line 26

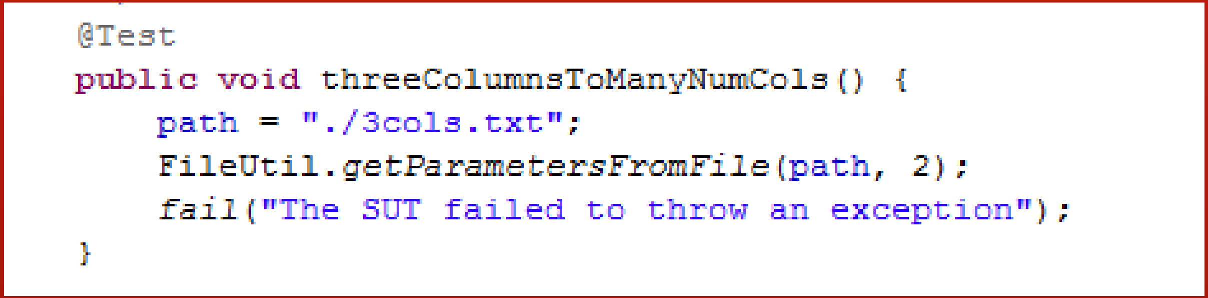
Comment [MC14]: Block 14: Line 27

Comment [MC15]: Block 15: Line 28

Alternative

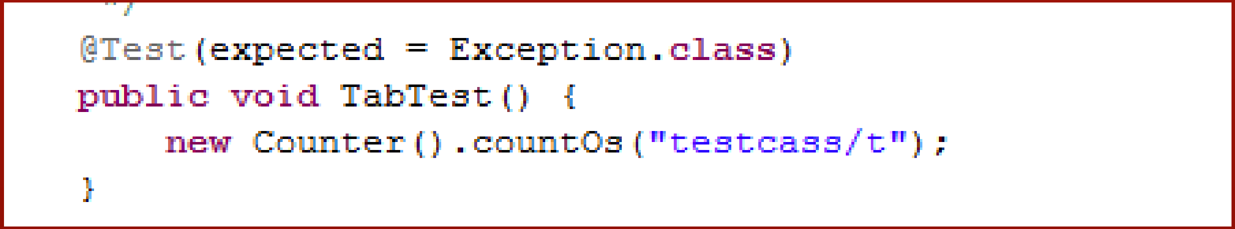
Detecting missing exceptions

- When all else fails:



```
@Test
public void threeColumnsToManyNumCols() {
    path = "./3cols.txt";
    FileUtil.getParametersFromFile(path, 2);
    fail("The SUT failed to throw an exception");
}
```

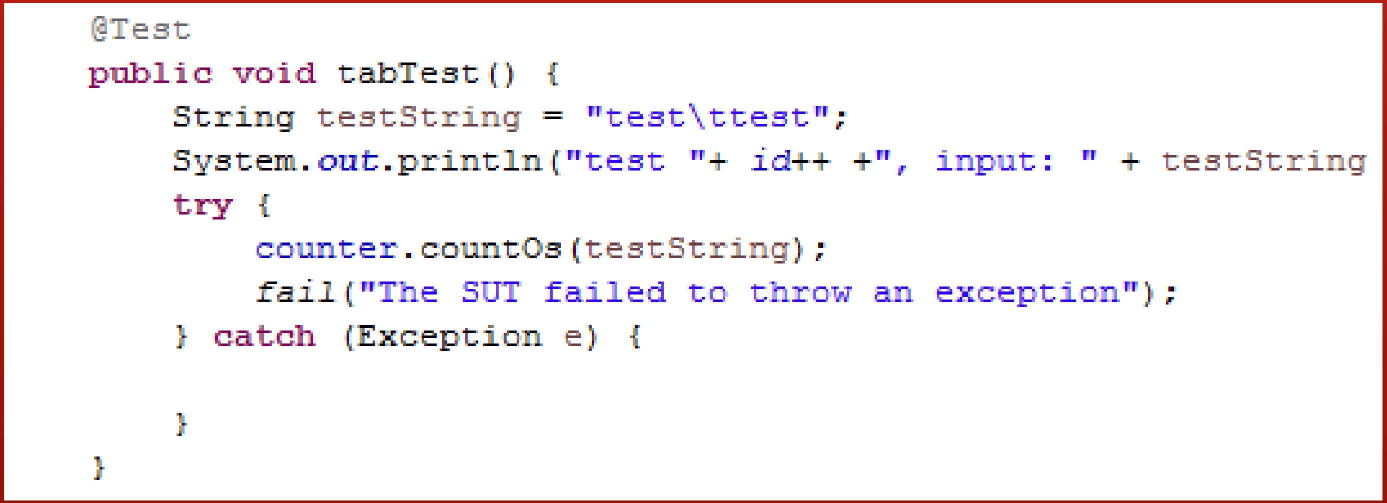
- Better



```
@Test(expected = Exception.class)
public void TabTest() {
    new Counter().countOs("testcass/t");
}
```

Detecting Missing Exceptions

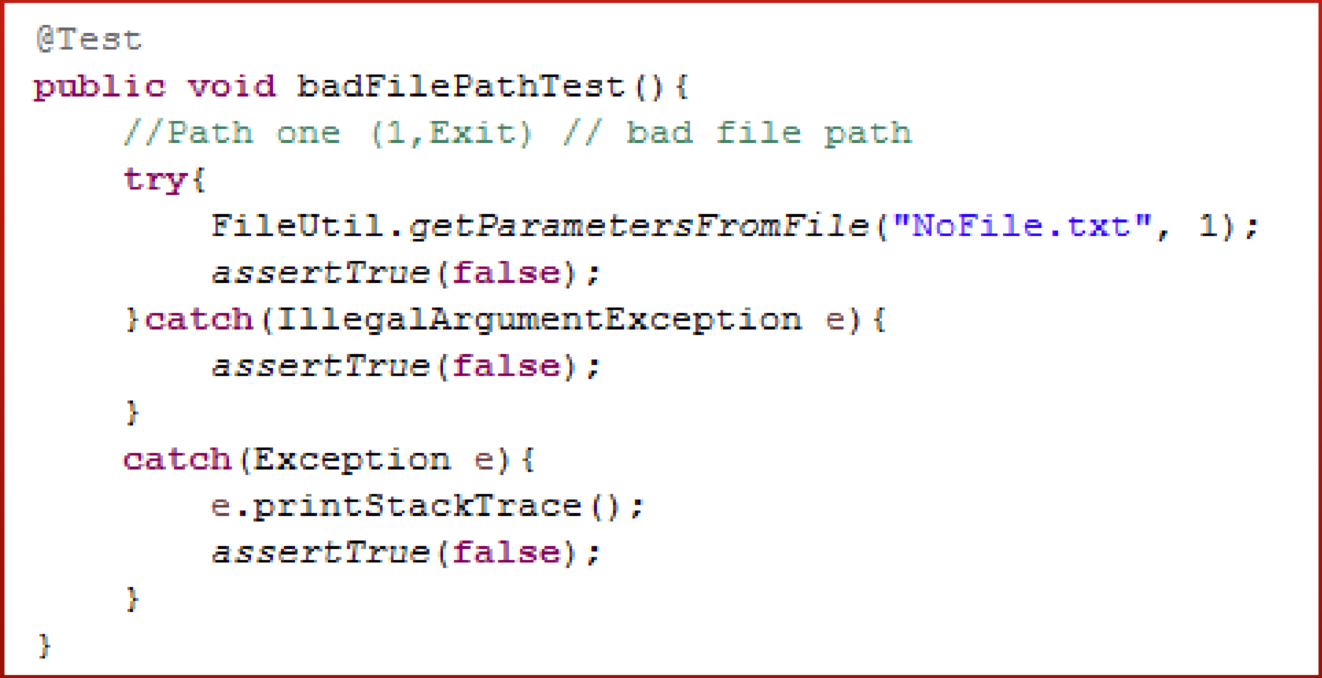
- Why write a try/catch that doesn’t catch anything?



```
@Test
public void tabTest() {
    String testString = "test\ttest";
    System.out.println("test "+ id++ +", input: " + testString
    try {
        counter.countOs(testString);
        fail("The SUT failed to throw an exception");
    } catch (Exception e) {

    }
}
```

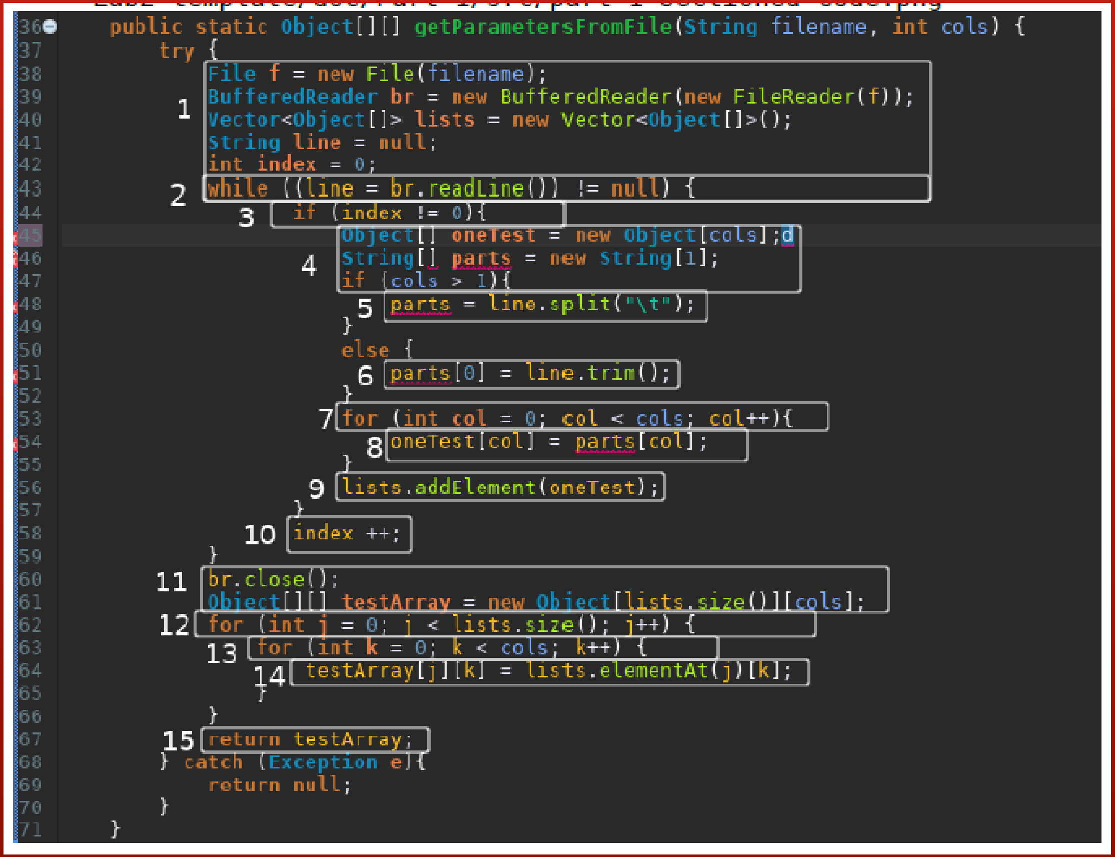
More ...



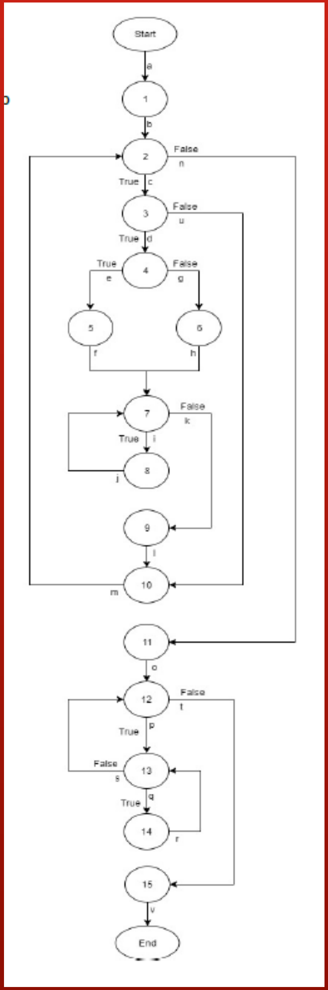
```
@Test
public void badFilePathTest(){
    //Path one (1,Exit) // bad file path
    try{
        FileUtil.getParametersFromFile("NoFile.txt", 1);
        assertTrue(false);
    }catch(IllegalArgumentException e){
        assertTrue(false);
    }
    catch(Exception e){
        e.printStackTrace();
        assertTrue(false);
    }
}
```

Partitioning the String

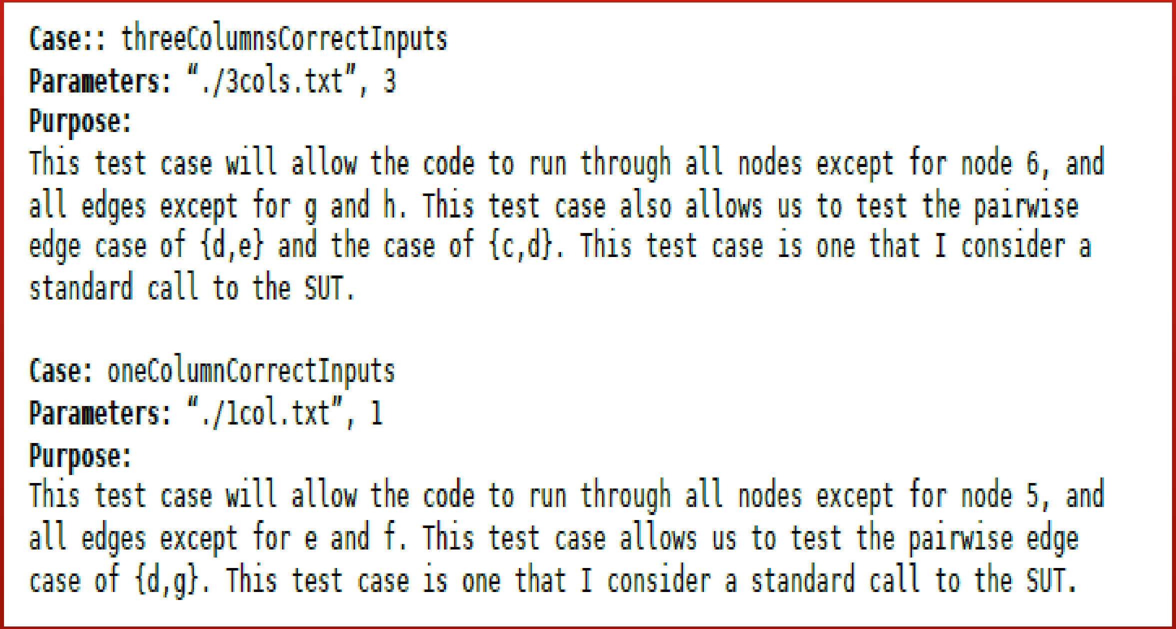
Documenting Basic Blocks



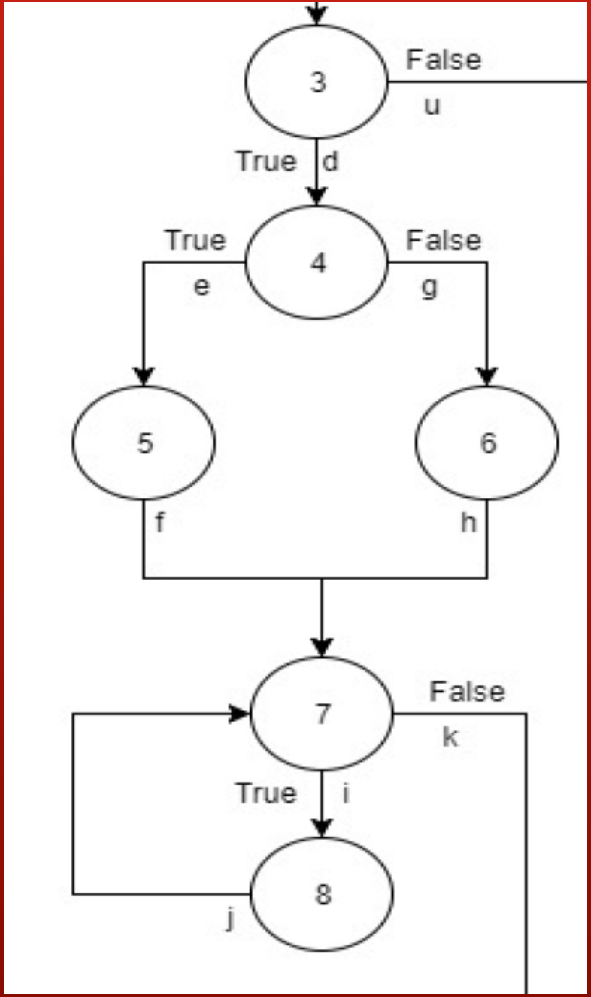
```
    public static Object[][] getParametersFromFile(String filename, int cols) {
        try {
            File f = new File(filename);
            BufferedReader br = new BufferedReader(new FileReader(f));
      1     Vector<Object[]> lists = new Vector<Object[]>();
            String line = null;
            int index = 0;
      2     while ((line = br.readLine()) != null) {
          3     if (index != 0){
                    Object[] oneTest = new Object[cols];d
              4     String[] parts = new String[1];
                    if (cols > 1){
                  5     parts = line.split("\t");
                    }
                    else {
                  6     parts[0] = line.trim();
                    }
                  7 for (int col = 0; col < cols; col++){
                  8     oneTest[col] = parts[col];
                    }
                  9 lists.addElement(oneTest);
                }
             10 index ++;
            }
     11     br.close();
            Object[][] testArray = new Object[lists.size()][cols];
     12     for (int j = 0; j < lists.size(); j++) {
         13     for (int k = 0; k < cols; k++) {
            14      testArray[j][k] = lists.elementAt(j)[k];
                }
            }
     15     return testArray;
        } catch (Exception e){
            return null;
        }
    }
```



Edge-Pair vs. Edge Coverage?


Case:: threeColumnsCorrectInputs
Parameters: "./3cols.txt", 3
Purpose:
This test case will allow the code to run through all nodes except for node 6, and all edges except for g and h. This test case also allows us to test the pairwise edge case of {d,e} and the case of {c,d}. This test case is one that I consider a standard call to the SUT.

Case: oneColumnCorrectInputs
Parameters: "./1col.txt", 1
Purpose:
This test case will allow the code to run through all nodes except for node 5, and all edges except for e and f. This test case allows us to test the pairwise edge case of {d,g}. This test case is one that I consider a standard call to the SUT.



Remember the Purpose/Audience

- Who is the audience for this document? (not me.)
- Why are they reading it?
- What is the organizational benefit?

An Experiment: CyBuzz

- Write a class named CyBuzz with one method
- public string cyBuzz(int x);
  - if x is a multiple of 3, return “Cy”.
  - if x is a multiple of 5, return “Buzz”.
  - if x is a multiple of 3 AND 5, return “CyBuzz”.
  - otherwise return x.
- No devices, no books, no notes.
- When complete write down the time and turn your paper over.

TDD: The Discipline

The Three Rules of TDD:

- You are not allowed to write any production code unless it is to make a failing unit test pass.
- You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
- You are not allowed to write any more production code than is sufficient to pass the one failing unit test

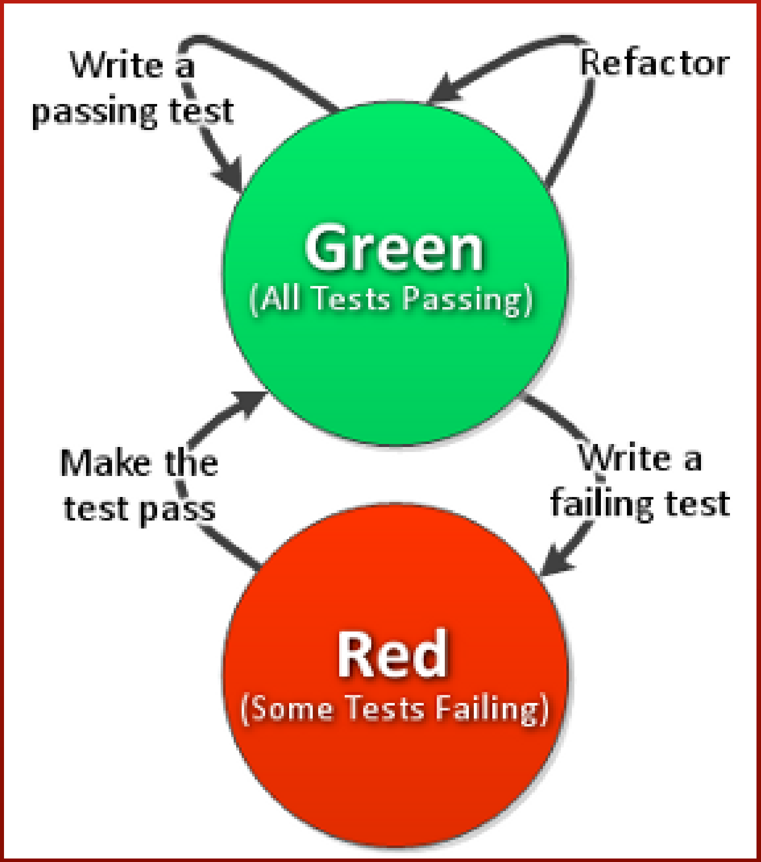


Figure from: http://www.peterprovost.org/blog/2012/05/02/kata-the-only-way-to-learn-tdd/
Rules from http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd

The FizzBuzz Code Kata

https://www.youtube.com/watch?v=JyRouDwzCoo

- Code Kata
  - Code Katas are to programming as Compulsory Figures are to figure skating.
  - https://www.youtube.com/watch?v=n2LwMId43uU
  - A Code Kata is an exercise designed to build and demonstrate technical precision, accuracy, and skill. For more, see CodeKata.com

Note: watch the lower left corner of the screen for the test result when the camera zooms out.

Writing code == writing tests

No handoffs. No deferred activities. Seamless integration.