# Test Driven Development (TDD) and using JUnit Testing

Due date: by the end of the recitation session

## Introduction

Every program requires testing and maintenance. There are many different types of testing approaches and many different philosophies regarding what things should be tested and how. The exercises below are designed to introduce you to a couple of tools and approaches that are used in testing. This is by no means an exhaustive introduction and using these techniques will not guarantee that you write bug-free code, but it may help.

**Worksheet**: https://goo.gl/0zfdA9

## Part 1

Test driven development (TDD) is something that many consider the right way of approaching the software development and software testing. What is TDD? An intro to the course on TDD at lynda.com: https://www.youtube.com/watch?v=QCif_-r8eK4 . As NYU students you have free access to all lynda.com tutorials, so you could follow the entire course. It is well worth it.
Watch the video to answer some of the questions in this part. (You can enable closed-captioning on the video so all of the members of your group can "see" what is being said. That's the CC button at the lower right corner of the video. )
You may also find the following tutorial useful: Introduction to Test Driven Development at http://www.agiledata.org/essays/tdd.html

Note that the definition of TDD given by google when you just search for "Test driven development" does not really capture the testing aspect of it. Do not copy and paste that definition to your answers on the worksheet. TDD is an approach to developing software. Any such approach uses several different types of testing. The kinds of tests that you should be thinking about at this point of your software developer career are unit tests and integration tests. Use the web to try to find definitions/descriptions of what these two types of tests are and how they differ. You should look at least at two or three sources to make sure you get the best view of what these types of tests are.

**Questions (answer them in the worksheet)**:
1. What is the difference between TDD way of software development and the way that you are developing the code right now?
2. What is meant by unit testing? Specify your sources.
3. What is meant by integration testing? Specify your sources.

## Part 2 JUnit and Integration with Eclipse

"JUit is a simple framework to write repeatable tests." (Quote from http://junit.org/junit4 ). As the name suggests, it is a framework for developing unit tests. There are many tutorials that you can use as an introduction to writing your own JUnit tests. The rest of this section makes references to the tutorial

at Vogella website. Lars Vogel is one of the Eclipse developers as well as the author of the tutorial, see http://www.vogella.com/people/larsvogel.html for his very short bio. The JUnit tutorial, http://www.vogella.com/tutorials/JUnit/article.html or just google for ”JUnit Vogella”, is fairly long, but you should concentrate on small sections of it that these instructions refer to (you may want to go over the entire thing on your own, though).
The latest versions of Eclipse comes with JUnit installed, so you do not need to worry about the installation process.
The steps below take you through creating a stub class and developing a test class for it (in that order). The idea is to implement tests and think about the tests and only then try to get the code class implementation working.

### `Number` class specification and ”interface”

You are going to work with a very simple class called `Number`. This class is supposed to represent an integer value. It should provide the following methods:

- `add` expects one parameter of type `Number`, adds the value of the parameter to the object on which it was called
- `subtract` expects one parameter of type `Number`, subtracts the value of the parameter from the object on which it was called
- `multiply` expects one parameter of type `Number`, multiplies the value of the parameter by the object on which it was called
- `divide` expects one parameter of type `Numbers`, divides the value stored in the object by the value of the parameter
- `getNumber` expects no parameter and returns the current value of the number

All of the methods above modify the object on which they are called. They do not return anything. The division method should throw an `ArithmeticException` if division by zero is attempted.
In addition, the class should provide the following methods:

- `duplicate` takes no parameters and returns a duplicate object (this method should actually create a new object containing the same data as itself)
- `toString` returns a `String` representation of the object (i.e., the number itself)
- `equals` overrides the method defined in the Object; it should return true if the values in both objects are exactly the same and false otherwise.

The class should also provide two constructors:

- default constructor initializes the value to zero
- one parameter constructor initializes the value to the value of the parameter

Here is the actual "stub" for this class (you can also download it from the course website).

```java
public class Number {
  private int num;

  public Number ( ) {
    //TODO: needs to be implemented
  }

  public Number ( int num ) {
    //TODO: needs to be implemented
  }

  public void add ( Number n )  {
    //TODO: needs to be implemented
  }
  public void subtract ( Number n ) {
    //TODO: needs to be implemented
  }

  public void multiply ( Number n ) {
    //TODO: needs to be implemented
  }

  public void divide ( Number n ) {
    //TODO: needs to be implemented
  }

  public int getNumber () {
    return num;
  }

  public Number duplicate ( ) {
    //TODO: needs to be implemented
    return null;
  }

  @Override
  public String toString ( ) {
    //TODO: needs to be implemented
    return null;
  }

  @Override
  public boolean equals (Object o ) {
    //TODO: needs to be implemented
    return false;
  }
}
```

Download the above source code and place it in a new package in Eclipse (you can copy and paste it from the worksheet document).

**Testing the Number class - NumberTest class**

Follow the instructions below to create the JUnit test for this class. If you run into trouble, see section 6 of the JUnit tutorial.

- Right click on the Number.java class in the PackageExplorer. Go to New and then pick *JUnit Test Case*.
- You should be looking at a dialog box with several options. Notice that the suggested name for the test class is `NumberTest` - this is a convention that you should follow. Do not change any options, just click *Next*.
- Select the checkbox next to the `Number` class (this should mark all checkboxes next to all methods from the Number class) and click *Finish*.
- It you are asked if you wish to add JUnit 4 library to the build path, agree to it.
- Run the `NumberTest` class. It should produce a report showing 10 failures (well, we have not written any tests or code yet, so what else could it be).

Now you have a class (unimplemented) and a test suit for it (also unimplemented). This is a good place to start thinking about tests. The specification of the `Number` class is the driving force for developing tests. We want tests that make sure that the class is really implemented according to the spec.

We will need to use `assert` statements to see the state of the variables in the code. Review the description of `assert` statements in section 5.2 of the JUnit tutorial.
We will need the `getTest` method to verify how most of the other functions work. It is not likely that a getter has a serious bug, so we will assume that it is correct.
Let's start by writing a test for the default constructor. What might go wrong?

- the value of the data field may not be set correctly (the spec says that it should be zero)
- the constructor may return a `null` reference instead of a reference to an actual new object (not likely, but technically possible, and we are trying to come up with a list of all possible issues)
- an exception could be thrown

We are going to test for all of these scenarios. Here is the test function that accomplishes the three goals:

```
@Test
public void testNumber() {
  try {
    Number n = new Number();
    assertNotNull("Default constructor returned null reference.", n);
    assertEquals("Default value should be 0.", 0, n.getNumber() );
  }
  catch (Exception e) {
    fail("testNumber failed: Exception thrown\n" + e.getClass() + "\n" + e.getMessage());
  }
}
```

This uses two different `assert...` statements and a `fail` statement.

If we run the `NubmberTest` class now, the test `testNumber` actually passes, even though the body of the default constructor is not yet implemented. Can you think of why this is the case?

**Write your own tests and implementations**

Write the rest of the tests for the class and then implement the methods for the `Number` class.
Pick one method at a time:

- Think of all the different possibilities in which something may go wrong. Use the spec to decide what the expected values should be and test if that's what they are. Use the spec to decide if there is a possibility of an exception and when that should and should not happen - test for both possibilities. Use the spec to decide what values should be returned by functions - test for correct and incorrect return values.
- Run the test and watch it fail. (With methods other than the default constructor, it should not be passing. If it does, there is most likely something wrong with the test.)
- Implement the method in the `Name` class.
- Repeat the steps above (revisiting the test and fixing the function implementation) until the test for your function passes.

Write a test for one of the methods on the worksheet.
Repeat the above process until ALL tests pass and the class has a complete implementation.

**Congratulations! If you followed these instructions you just completed your first Test Driven Development project!**