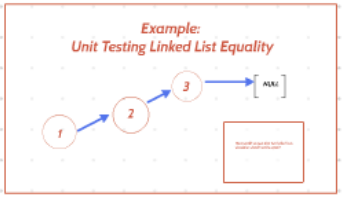
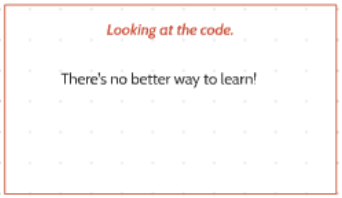
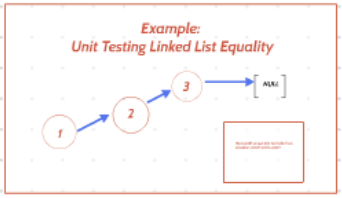
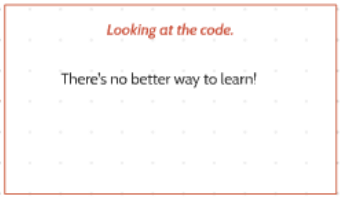
# Lecture 7 - Unit Testing

# Lecture 7 - Unit Testing

# What is Unit Testing?

### Unit Testing

Unit testing involves testing the smallest coherent 'units' of code, such as functions, methods, or classes.

### Examples

1. Testing that a .sort method actually sorts elements
2. Testing that a nil/null throws an exception
3. Testing that a formatNumber method formats a number properly
4. Checking that passing in a string to a function which expects an integer does not crash
5. Testing that a .send and .receive method exist on a class

### Unit Testing

This is usually done by the developer writing the code, another developer (esp. in pair programming), or (very occasionally), a white-box tester.

### What's the Point?

1. Problems found earlier
2. Faster turnaround time
3. Developer understands issues with his/her code
4. "Living documentation"
5. Able to tell if your changes caused issues elsewhere by running full test suite

### What do Unit Tests consist of?

1. (optional) Set up code
2. Preconditions
3. Execution Steps
4. Postconditions - a/k/a Assertions (a/k/a asserts, shoulds, musts)
5. (optional) Tear down code

### Example in Natural Language

I create two linked lists with the same number of nodes and same data in each node.

If I compare them with the equality operator, they SHOULD be equal.
(or "they MUST be equal.")
(or "I ASSERT that they will be equal")

When you think "should" or "must", that is the assertion. It's what you're testing for.

It's the EXPECTED BEHAVIOR of the unit test.

When you execute the test, that's when you'll find out the OBSERVED BEHAVIOR.

If they don't match, the test fails.

### Assertions

You can assert many things...

assertEquals, assertArrayEquals, assertTrue, assertFalse, assertNull, assertNotNull, assertSame, assertThat("something"), assertNotSame...

or you can fail().

### Fail

fail() makes a test automatically fail.

Usually because you know it's going to fail anyways and you don't want to waste time running it, or check that a certain part of code is not reached.

### Example

```
// 0. A LL should always equal itself
@Test
public void testEqualsSelf() {
  LinkedList<Integer> ll = new LinkedList<Integer>();
  assertEquals(ll, ll);
}
```

### JUnit vs Other Unit Testing Frameworks

We are going to cover JUnit in this class, but it is not the only one.

xUnit - JUnit-like tests in other languages
RSpec - Ruby testing framework
Quickcheck - Property-based testing framework for Haskell

Google "list of unit testing frameworks" if you're really interested.

### That being said...

xUnit-style tests are very popular, and the differences between those and other testing frameworks are relatively minor.

Remember, it's all Turing machines (or lambda calculus, whichever abstraction you prefer). Worry less about the syntax and more about the concepts.

### Things to Remember...

There's a trade-off between how much testing you want to do, and how fast you want to get the product out the door.

This applies to both test development and test execution time. If tests take nine hours to run, nobody will run the full test suite every time they compile!

### Solution

For a large system, it's usually not necessary to run the FULL test suite EVERY time you compile (on a properly architected, modular system).

Run a subset of the tests, or run them only on the class you're working on.

# Unit Testing

Unit testing involves testing the smallest coherent "units" of code, such as functions, methods, or classes.

# *Examples*

1. Testing that a .sort method actually sorts elements
2. Testing that a nil/null throws an exception
3. Testing that a formatNumber method formats a number properly
4. Checking that passing in a string to a function which expects an integer does not crash
5. Testing that a .send and .receive method exist on a class

# *Unit Testing*

This is usually done by the developer writing the code, another developer (esp. in pair programming), or (very occasionally), a white-box tester.

# *What's the Point?*

1. Problems found earlier
2. Faster turnaround time
3. Developer understands issues with his/her code
4. "Living documentation"
5. Able to tell if your changes caused issues elsewhere by running full test suite

# *What do Unit Tests consist of?*

1. (optional) Set up code
2. Preconditions
3. Execution Steps
4. Postconditions - a/k/a Assertions (a/k/a asserts, shoulds, musts)
5. (optional) Tear down code

## *Example in Natural Language*

I create two linked lists with the same number of nodes and same data in each node.

If I compare them with the equality operator, they SHOULD be equal.
(or "they MUST be equal.")
(or "I ASSERT that they will be equal")

When you think "should" or "must", that is the assertion.  It's what you're testing for.

It's the EXPECTED BEHAVIOR of the unit test.

When you execute the test, that's when you'll find out the OBSERVED BEHAVIOR.

If they don't match, the test fails.

# *Assertions*

You can assert many things...

assertEquals, assertArrayEquals, assertTrue, assertFalse, assertNull, assertNotNull, assertSame, assertThat(*something*), assertNotSame...

or you can fail().

# Fail

fail() makes a test automatically fail.

Usually because you know it's going to fail anyways and you don't want to waste time running it, or check that a certain part of code is not reached.

# *Example*

```
//  0. A LL should always equal itself
@Test
public void testEqualsSelf() {
   LinkedList<Integer> ll = new LinkedList<Integer>();
   assertEquals(ll, ll);
}
```

# *JUnit vs Other Unit Testing Frameworks*

We are going to cover JUnit in this class, but it is not the only one.

xUnit - JUnit-like tests in other languages
RSpec - Ruby testing framework
Quickcheck - Property-based testing framework for Haskell

Google "list of unit testing frameworks" if you're really interested.

## *That being said...*

xUnit-style tests are very popular, and the differences between those and other testing frameworks are relatively minor.

Remember, it's all Turing machines (or lambda calculus, whichever abstraction you prefer). Worry less about the syntax and more about the concepts.

## *Things to Remember...*

There's a trade-off between how much testing you want to do, and how fast you want to get the product out the door.

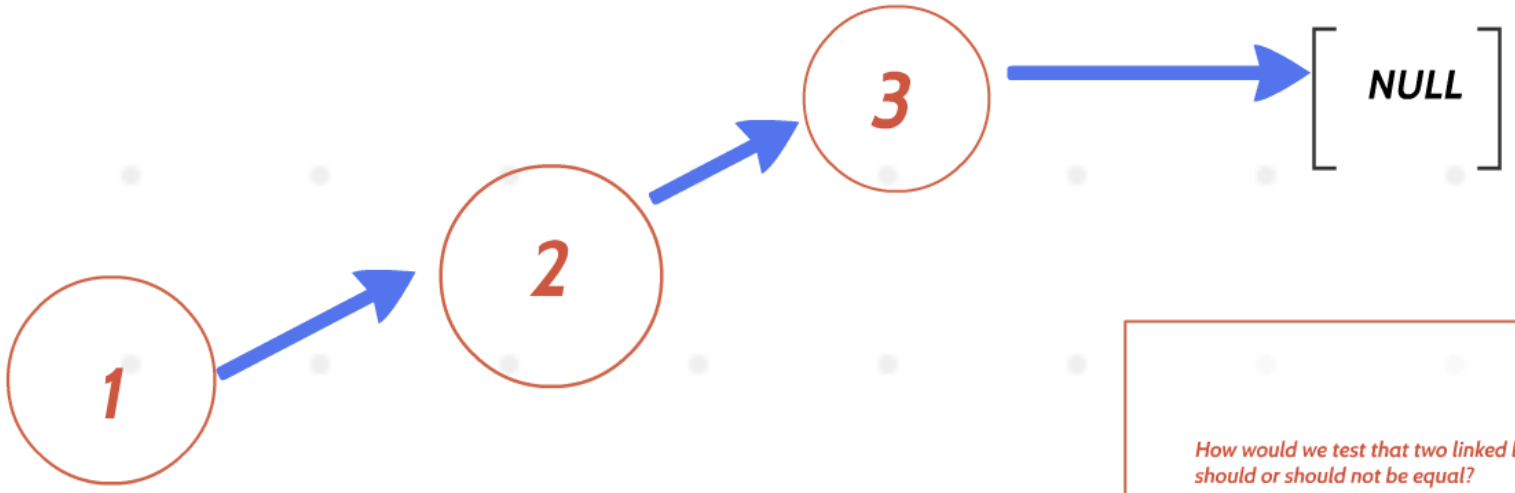
This applies to both test development and test execution time. If tests take nine hours to run, nobody will run the full test suite every time they compile!

# *Solution*

For a large system, it's usually not necessary to run the FULL test suite EVERY time you compile (on a properly architected, modular system).

Run a subset of the tests, or run them only on the class you're working on.

# Example:
# Unit Testing Linked List Equality




*How would we test that two linked lists should or should not be equal?*


1


2


3

# NULL

*How would we test that two linked lists should or should not be equal?*

*Looking at the code.*

There's no better way to learn!

# Lecture 7 - Unit Testing