Unit Testing

The bread and butter of testing

http://blogs.msdn.com/b/brada/archive/2009/06/26/unit-testing-with-net-ria-services.aspx

Learning objectives

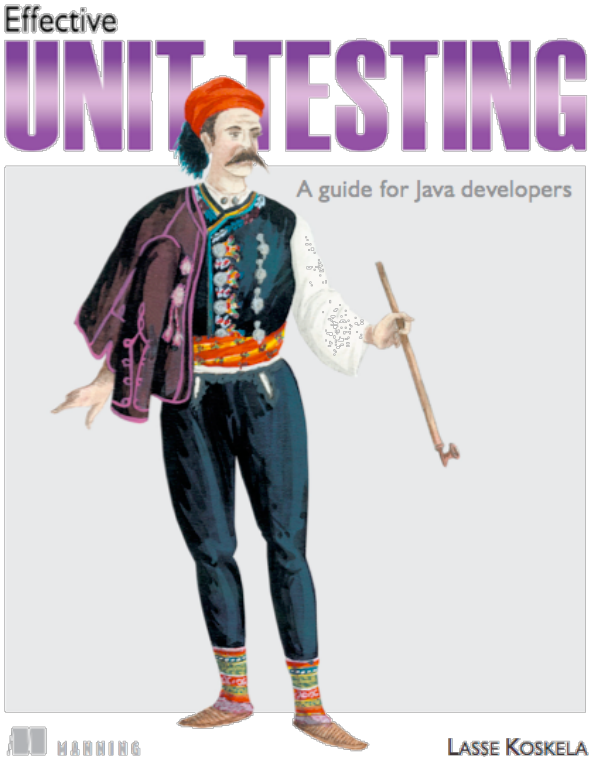
After viewing this lecture, you will be able to:

- Describe the scope of unit testing
- List and apply good unit testing practices
- Use JUnit with Eclipse for unit testing Java code
- Write basic unit tests in JUnit

Agenda

- Unit testing principles
- JUnit basics

Unit Testing Principles



What is unit testing?

- An in-process testing practice
- Checking the behavior of implemented functionality through examples that exercise a desired portion of the code and produce results and side effects that can be inspected and verified

Operates on:

- Classes and methods
- A small number of collaborating classes

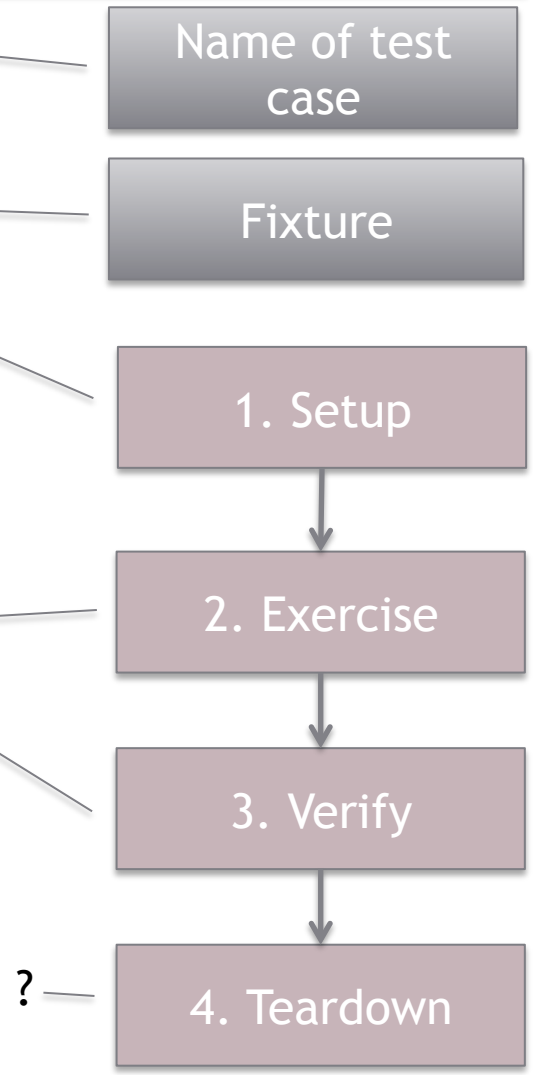
Attempts to isolate components from each other while testing them

A JUnit Example

```
public class TennisGamePlayerTest {
    Player p;

    @Before
    public void createAPlater() {
        p = new Player("P");
    }

    @Test
    public void testLoveUopnInitialization() {
        assertEquals("Love", p.getScore());
    }
    @Test
    public void testFifteen() {
        p.addPoint();
        assertEquals("Fifteen", p.getScore());
    }
    @Test
    public void testThirty() {
        p.addPoint();
        p.addPoint();
        assertEquals("Thirty", p.getScore());
    }
}
```



Good unit testing practices

- Simplicity
- Understandability
- Essentiality
- Single purpose
- Behavior first
- Maintainability
- Determinism
- Independence
- Failability
- Comprehensiveness
- Speed

Simplicity

Keep it simple, keep it safe

Take small steps
Avoid complex fixtures

http://www.delcoalarm.com/3-simple-tips-to-keep-your-home-safe/

Understandability

Tests should be and readable and meaningful

Make tests self-documenting

http://abc-lang.com/translation-service/document-translation/

Understandability

Avoid magic numbers

Essentiality

A test should not be overprotective

Remove redundant assertions

http://www.legacydad.com/overprotective-parents-underdeveloped-children-part-1/

Single purpose

A test should have one reason to fail

Avoid testing split logic in a single test

http://www.recycleaway.com/

Behavior first

100% test coverage is not the goal

Focus on behavior, not implementation

Maintainability

Tests should be easy to maintain

Avoid duplication

Refactor test code

http://innovativemlmsuccess.com/tag/duplication/

Maintainability

Avoid conditional logic in test code

http://innovativemlmsuccess.com/tag/duplication/

Determinism

Tests should not fail at random!

Isolate and remove sources of nondeterminism

http://psmag.com

Independence

Tests should be able to run in any order,
and give the same results

Tests should not depend on each other

Isolate the tests

Be careful when they share fixtures

Failability

Never failing tests tests don’t make sense

Make sure that you can make tests fail

Don’t write tests without assertions

Make sure that unfinished tests fail

Comprehensiveness

Test should span diverse worlds

Test both happy and sad paths

Test corner cases

Test representative cases

Test without overkill

Test boundary cases

Speed

Tests should provide fast feedback

Use test doubles if tests rely on expensive resources

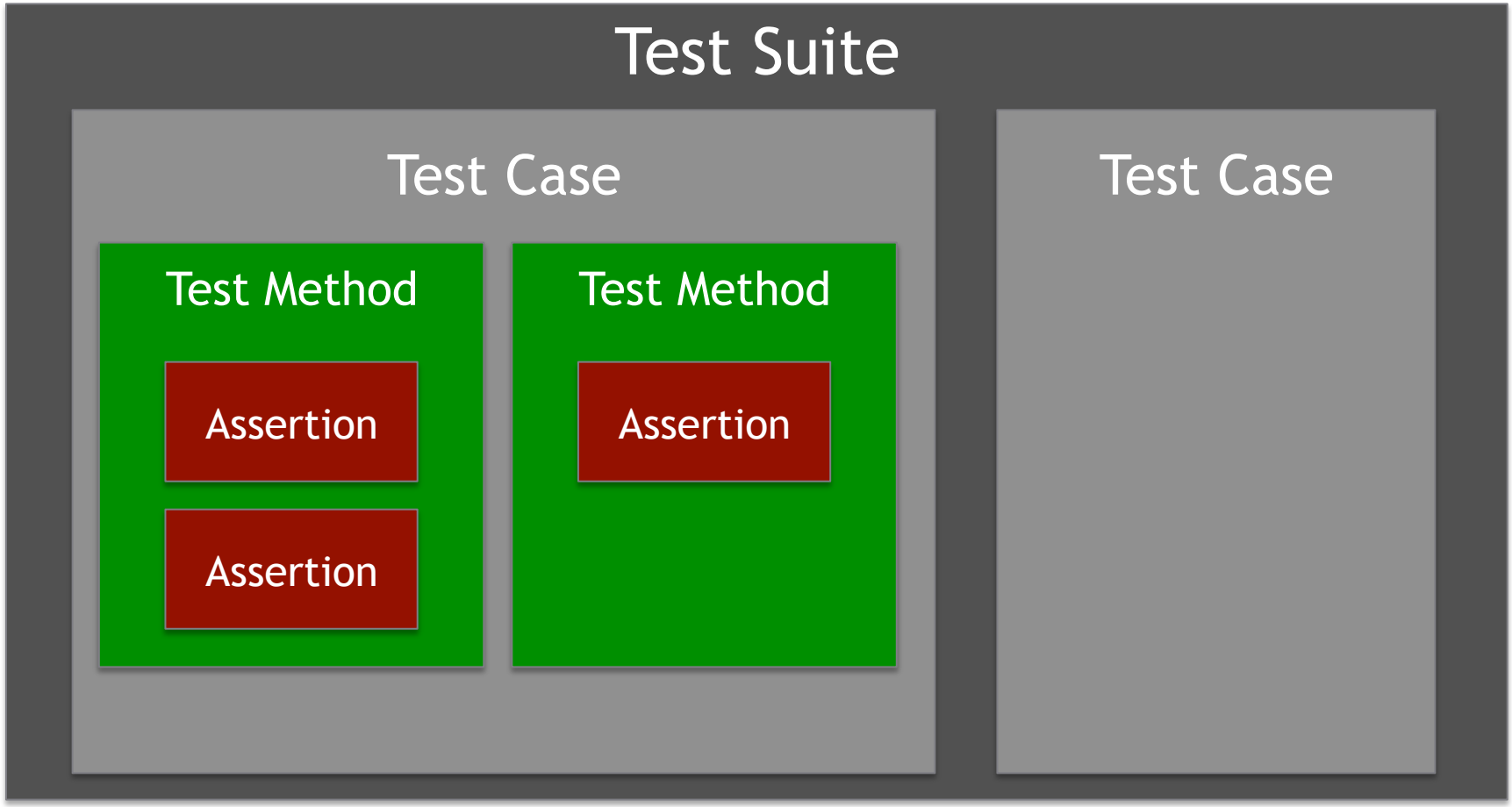
Rethink your tests if they are sluggish

http://images.businessweek.com/ss/10/04/0406_most_expensive_watches/18.htm

JUnit Basics

The de-facto unit testing framework for Java

JUnit concepts



Typically: Test Case = Test Class → tests a single class

Assert statements in JUnit

- fail()
- assertEquals(…, …)
- assertTrue(…)
- assertFalse(…)
- assertNotNull(…)
- assertNull(…)
- assertArrayEquals(…)
- assertNotSame(…, …)
- assertSame(…)

- *All assertions: have an optional first parameter that represents a failure message*

- *In all assertions comparing two objects: expected value is specified before the actual value:*

assertEquals(expected, actual)

Anatomy of JUnit test methods

… are indicated using @Test annotation (JUnit4)

… have a meaningful name

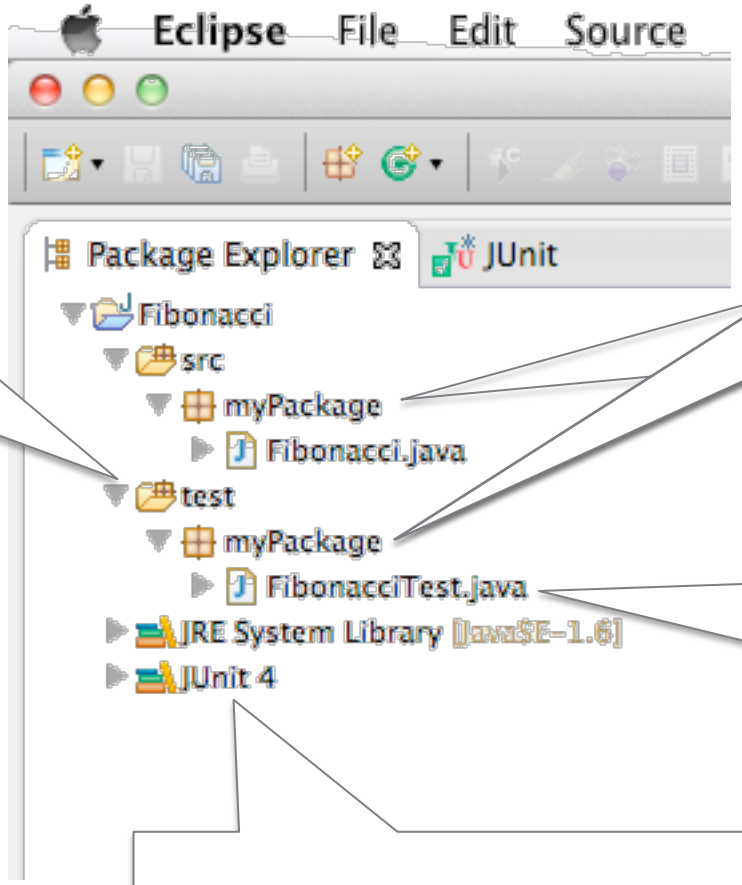
```
@Test
public void aMemberCanAcceptAFriendRequestFromAnother() {
        SocialNetwork sn = new SocialNetwork(accountI
        Account me = sn.join("Hakan");
        Account her = sn.join("Cecile");
        sn.sendFriendRequestTo("Cecile", me);
        sn.acceptFriendshipFrom("Hakan", her);
        me = sn.refresh(me);
        her = sn.refresh(her);
        assertTrue(me.hasFriend("Cecile"));
        assertTrue(her.hasFriend("Hakan"));
}
```

… may contain any code

- local variables
- control structures
- calls to utility classes
- calls to helper methods defined inside test case
- calls to classes under test

… contain at least one assertion

Typical code organization



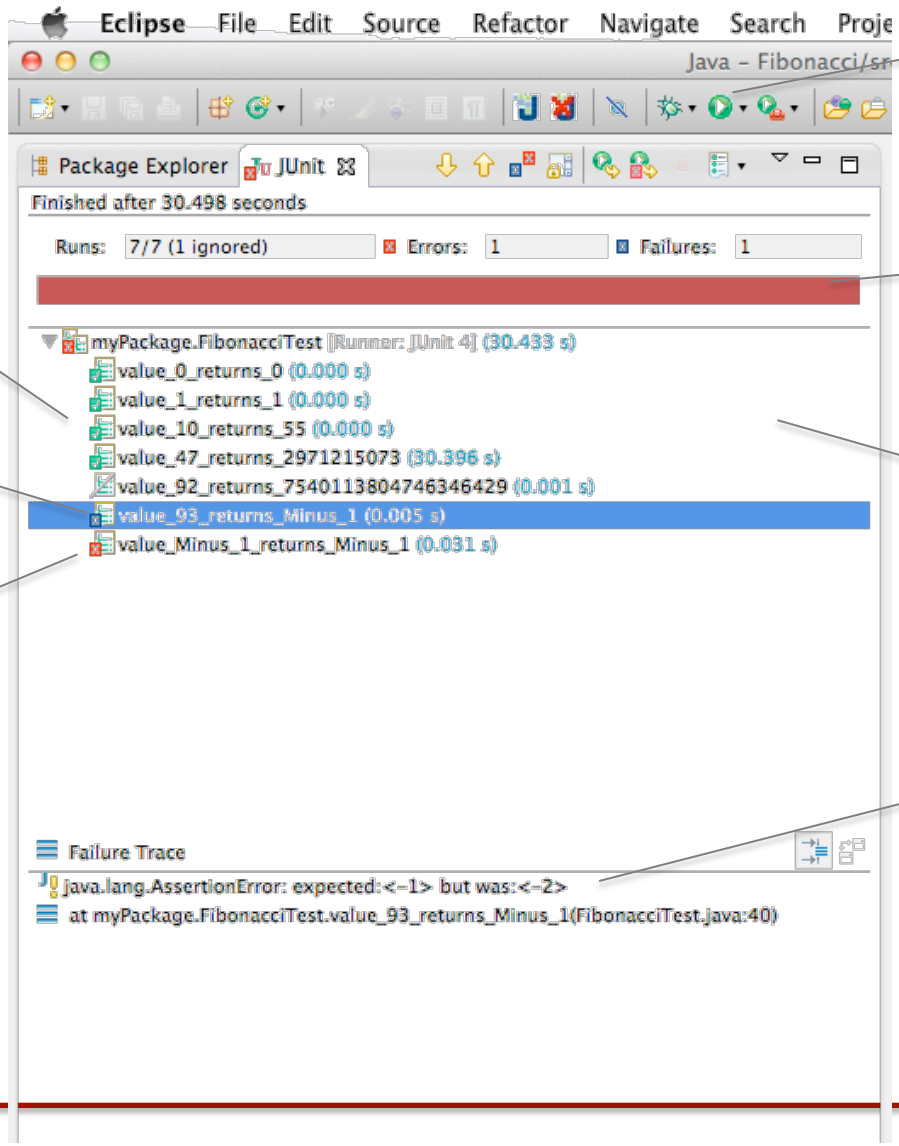
Test cases in test folder

Production classes and test cases in the same package

Convention for naming test cases:
MyClassTest.java
or
TestMyClass.java

JUnit library:
Test cases must import static org.junit.Assert.*

JUnit test execution in Eclipse



Success

Failure

Error

Run test cases

Status bar:

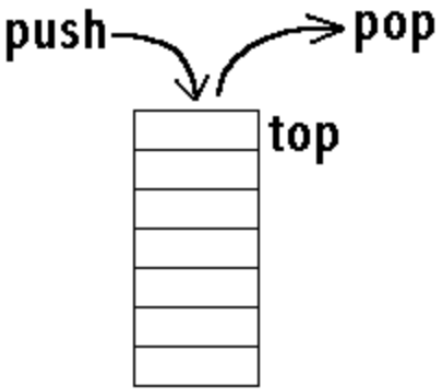
fail *all pass*

Results

Failure trace with reason for failed assertion

JUnit exercise (to try on your own)

A stack is a LIFO sequence. Addition and removal takes place **only** at one end, called the top.



Operations

- `push(x)`: add an item on the top
- `pop`: remove the item at the top
- `peek`: return the item at the top (without removing it)
- `size`: return the number of items in the stack
- `isEmpty`: return whether the stack has no items

JUnit exercise: stack

- A stack is empty on construction
- A stack has size 0 on construction
- After n pushes to an empty stack (n > 0), the stack is non-empty and its size is equals n
- If one pushes x then pops, the value popped is x
- If one pushes x then peeks, the value returned is x, but the size stays the same
- If the size is n, then after n pops, the stack is empty and has a size 0
- Popping from an empty stack throws an exception: NoSuchElementException
- Peeking into an empty stack throw an exception: NoSuchElementException

Summary

- Unit testing is applied throughout construction.
- Good unit tests involve simple, understandable, meaningful, deterministic, independent tests that have a single purpose, can demonstrably fail, provide fast feedback, are easy to maintain, focus on behavior, and are comprehensive without being overprotective.
- JUnit is an easy-to-use unit testing framework for Java with excellent Eclipse integration.