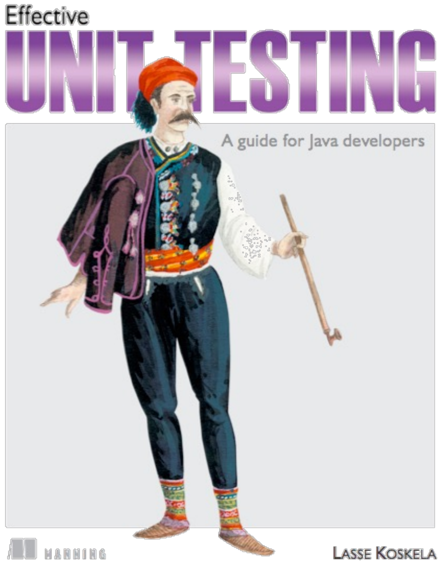
Unit Testing – the Basics

The bread and butter of testing

** Includes L0 **

Agenda

- Definition
- JUnit basics



What’s a unit?

- Smallest programming abstraction that can be tested independently
- Smallest programming abstraction that can be assigned to a single developer
  - OO languages: *methods, (or methods in a class?)*
  - Procedural languages: *procedures/subroutines, closely related collections of these (in the same file)*
  - Functional languages: *functions, closely related collections of functions (in the same module)*

What is unit testing?

- A low-level testing practice that focuses on:
  - checking the behavior of implemented functionality through examples that exercise a targeted portion of the code and produce results and side effects that can be inspected and verified against expectations
- Most effective as an *in-process* practice (continuously applied)

Operates on:

- *Methods*
- *Classes*

Attempts to isolate the targeted code from the rest of the system while testing it

What do you test?

The OO case…

The state of the entity under test in response to a stimulus (side effect):

Objects: send message, check the object state

The result returned by entity under test in response to a stimulus:

Objects: send message, check the returned value/object

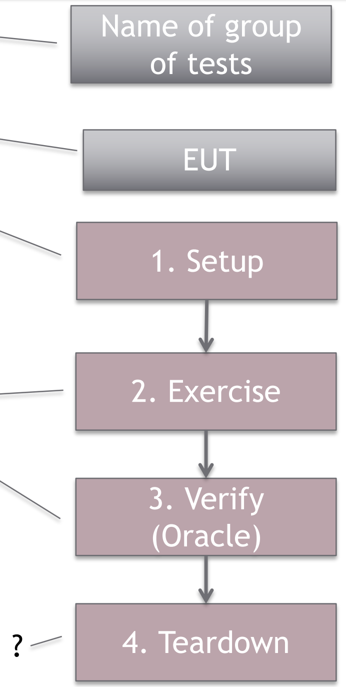
*Is the **state** and/or **return value/object** what you would expect?*

A JUnit Example

```java
public class TennisGamePlayerTest {
    Player p;

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

    @Test
    public void loveUponInitialization() {
        assertEquals("Love", p.getScore());
    }
    @Test
    public void fistPointGivesFifteen() {
        p.addPoint();
        assertEquals("Fifteen", p.getScore());
    }
    @Test
    public void secondPointGivesThirty() {
        p.addPoint();
        p.addPoint();
        assertEquals("Thirty", p.getScore());
    }
}
```



ARRANGE → ACT → ASSERT (→ TEAR-DOWN / CLEAN-UP)

Where is the driver?

```
public class TennisGamePlayerTest {
    Player p;

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

    @Test
    public void testLoveUponInitialization() {
        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());
    }
}
```

Junit is the driver

No test stub in this example!

Using: JUnit 4.12

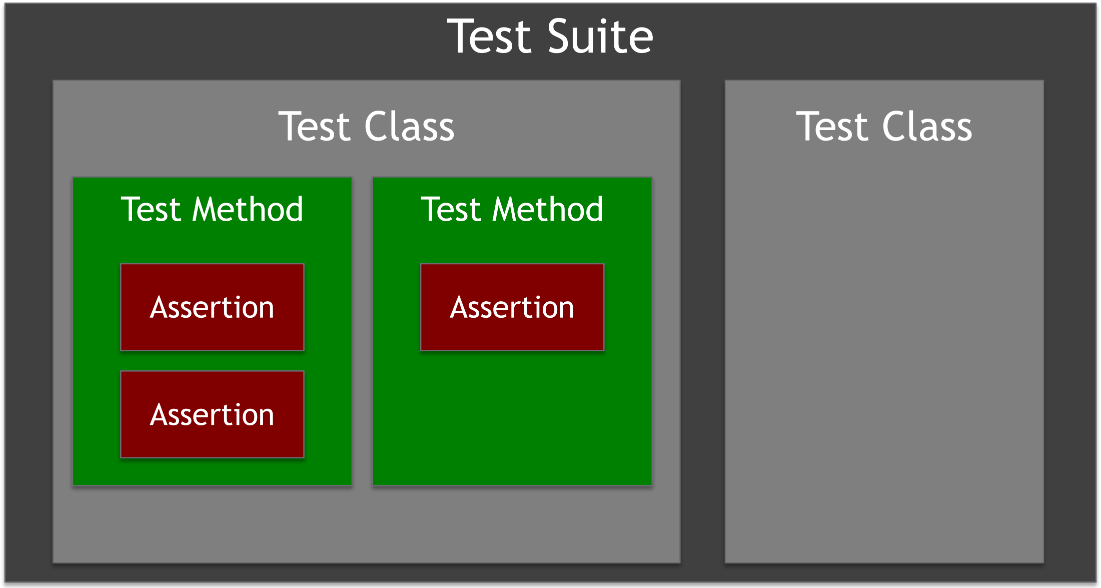
JUnit Basics

The de-facto **unit testing framework** for Java

Or

The de-facto **test driver** for Java

JUnit concepts



Some confusing terminology…

- *Sometimes in JUnit, a* Test Class is called a Test Case (like in Eclipse)
- *In standard testing terminology*: single test method = Test Case

Assert statements in JUnit4

- fail()
- uses equals() assertEquals(…, …)
- assertTrue(…)
- assertFalse(…)
- assertNotNull(…)
- assertNull(…)
- assertArrayEquals(…, …)
- assertNotSame(…, …)
- checks obj ref assertSame(…, …)

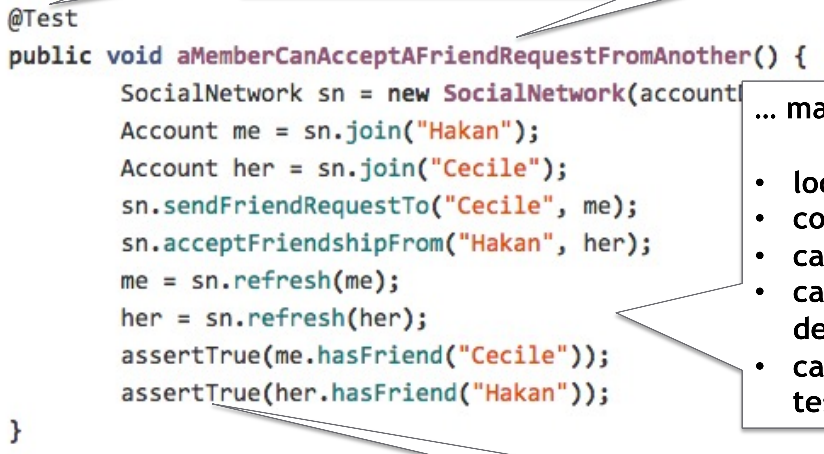
- *All assertions: have an optional first parameter that represents a failure message*
- *All assertions comparing two objects: expected value (oracle) is specified before the actual value:*

assertEquals(**expected**, **actual**)

Anatomy of a JUnit4 test method

… is indicated using @Test annotation (JUnit4)

… has a meaningful name



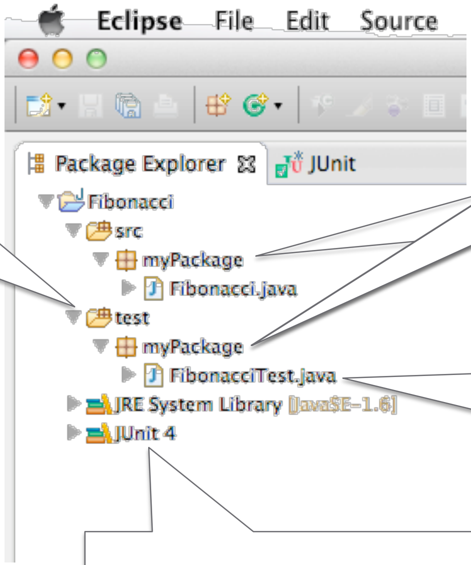
```
@Test
public void aMemberCanAcceptAFriendRequestFromAnother() {
        SocialNetwork sn = new SocialNetwork(account
        Account me = sn.join("Hakan");
        Account her = sn.join("Cecile");
        sn.sendFriendRequestTo("Cecile", me);
        sn.acceptFriendshipFrom("Hakan", her);
      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

… contains at least one assertion (or should expect an exception)

Typical code organization



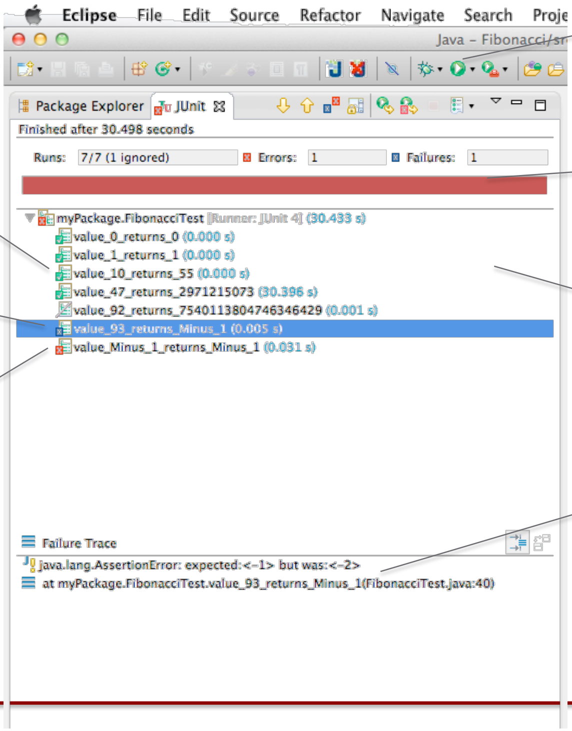
Test cases in test folder

Production classes and test classes in the same package

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

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

JUnit test execution in Eclipse (and other tools)



```
Finished after 30.498 seconds

Runs: 7/7 (1 ignored)    Errors: 1    Failures: 1

myPackage.FibonacciTest [Runner: JUnit 4] (30.433 s)
    value_0_returns_0 (0.000 s)
    value_1_returns_1 (0.000 s)
    value_10_returns_55 (0.000 s)
    value_47_returns_2971215073 (30.396 s)
    value_92_returns_7540113804746346429 (0.001 s)
    value_93_returns_Minus_1 (0.005 s)
    value_Minus_1_returns_Minus_1 (0.031 s)

Failure Trace
java.lang.AssertionError: expected:<-1> but was:<-2>
at myPackage.FibonacciTest.value_93_returns_Minus_1(FibonacciTest.java:40)
```

Run test classes

Status bar:

🟥 *at least one fails*

🟩 *all pass*

Results

Failure trace with reason for error or failed assertion

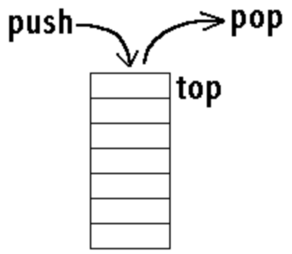
Success

Assertion Failure

Run-Time Error

L0: JUnit exercise

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



Operations

- `push(x)`: add an item on the top
- `pop`: remove the item at the top and return its value
- `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

Stack

```
public class MyStack {
  private int maxSize = 10;
  private int[] stackArray;
  private int top;

  public MyStack() {
      stackArray =
               new int[maxSize];
      top = -1;
  }

  public void push(int j) {
      stackArray[++top] = j;
  }
```

```
  public int pop() {
      return stackArray[top--];
  }

  public int peek() {
      return stackArray[top];
  }

  public int size() {
      return top + 1;
  }

  public boolean isEmpty() {
      return (top == -1);
  }

}
```

Spec in next slide

JUnit exercise: stack spec
Task: write a test case for each behavior

- `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

1. A stack is empty on creation
2. A stack has size 0 on creation
3. After n pushes to an empty stack (n > 0), the stack is non-empty and its size equals n
4. If one pushes x then pops, the value popped is x and the size equals what it was before the push
5. If one pushes x then peeks, the value returned is x, but the size stays the same as after the push
6. If the size is n (n > 0), then after n pops, the stack is empty and has size 0
7. Popping from an empty stack throws an exception: InvalidOperationException
   - Fix production code to make this test pass, if it fails!
8. Peeking into an empty stack throws an exception: InvalidOperationException
   - Fix production code to make this test pass, if it fails!
9. What happens if you push too many elements onto the stack?
   - What extra behavior do you need in MyStack to test this and let the client code protect itself against an overflow?
   - Is ArrayIndexOutOfBounds runtime exception acceptable when an overflow happens? If not, what should you do?
   - Write a test case for this situation, and modify the production code

JUnit4 shorthand for expecting an exception:

`@Test(expected=NoSuchElementException.class)`

L0: Unit Testing Warmup: Using Vocareum

- Open Canvas
- Go to Assignments -> L0: Unit Testing Warmup (or see under “Coming Up” tab)
- Authorize Vocareum
- Look under ‘src’ and ‘test’ subfolders in your Vocareum work area for starter code
- Copy starter code in your IDE to a new Java project Stack
- Run test class MyStackTest (one trivial test should pass)
- Add new tests to MyStackTest, modify MyStack if necessary
- Copy code back to Vocareum work area
- Submit and see results/report on Vocareum console
- YOUR CODE MUST COMPILE CORRECTLY LOCALLY AND ON VOCAREUM TO GET ANY MARKS