Hokie Example

- ExHokieClass
  - overloaded constructor
  - getters, setters, pid immutable
  - potential design issues?
    - pid aliases
    - multiple VT degrees
  - style issues
    - parameter names, instance variable names
  - main method and JUnit
    - Run both
    - Investigate failures (notice test cases for testNextReunion())

JUnit Demo follow up

- What is JUnit testing?
  - Framework for unit testing
  - Unit testing is writing tests for small units of code
  - Typically instantiate an object in the set up
  - Call various methods in different test cases
  - Use assert statements to check for correctness of method calls
- It's okay not to immediately understand everything demoed in Eclipse, go practice on your own, use the examples to study and reference!

Testing

- Have an **ADVERSARIAL MINDSET**
  - goal should be to try to **break the code**, rather than to see if it works
  - testing easy paths through the code is a fine place to start, but you learn very little from such tests
  - look for edge cases, making sure that the code succeeds or fails as expected around things like the boundaries of conditional statements and loops
  - ***Try to write tests that are better than our reference tests on WebCAT!!***

- The role of unit testing in longer-lived projects: As an example, on some teams every bug submitted to a bug-tracking system is accompanied by a unit test that elicits the bug. The bug is considered fixed when the test no longer fails. The test is then added to a pool of regression tests that are run periodically — ideally on every future change — to make sure the bug hasn't been reintroduced.

Testing (from posted Guidelines)

- **N simple conditions, N+1 branches and tests**

  Assertions in a test method need to make it to every condition of an if-else statement. It isn't enough that the test reaches the 'else' condition. To test an if-else statement properly, the body of each condition must be run during testing.

  if (x == 0 && y ==1) // 2 conditions, 3 checks- TF, FT, TT

  if (x == 0 || y == 1) // 2 conditions, 3 checks- TF, FT, FF

- **Assert Statements**

  Common ones:

  assertEquals

  assertTrue

  assertFalse

  assertNull

  assertNotNull

  Checking if values are equal?

  NO assertEquals(true, shortStack.peek ().equals("A is for Array"));

  MEH assertTrue(shortStack.peek ().equals("A is for Array"));

  YES!!!assertEquals("A is for Array",shortStack.peek ());

- Remember to reference Canvas| General Course Information | Resources | Writing JUnit Tests with student.TestCase

BEWARE

TESTING

- if statements in test cases - should just be using assert statements
- remember can use assertTrue and assertFalse
- should have no parameters and return types in test methods
- helps debugging to put expected value before actual value in assertEquals statements

Testing null and Avoiding NullPointerException

- bad: assertEquals(null, nullObject);
- bad: assertTrue(nullObject == null);

- **good: assertNull(nullObject);**