Unit testing

* Can only prove that it works given the set of limits and conditions present in tests
* Code shouldn’t drift away from limits put on them by the tests
* Help communicate codes intended use
  + Illustrates how you expect the code to perform on various inputs and conditions

Preparing unit tests

* Usually testing a method of an object
  + Therefore may interact with object’s other attributes or methods
  + Often need to set up an instance of the object to test
* Write the test code before/during the code is being written, never afterwards
* Run tests while developing code base
* Things to check for
  + Are the boundary results or the edge cases correct
  + Are the boundary conditions correct
  + Can you check the inverse relations
    - For the mathematical problems
    - Ex: use multiplication to check the division
  + Can you cross-check the results by some other method
  + Can you force error conditions to occur
  + Are the performance characteristics within bounds
* Properties of a good test
  + Test everything (look above)
  + Automatic
  + Thorough
  + Repeatable (should produce same results)
  + Independent of other tests and the environment
  + Professional
    - Written and maintained to the same standard as shipped code
* Testing the tests
  + Improve the tests while fixing bugs
    - Once encounter a bug, create a test to test the bug
  + Prove the tests by introducing bugs
  + Verify that test fails as expected
* Structuring unit tests
  + Naming conventions
    - testMethodName
    - EX: method: createaccount, test: testCreateAccount
  + Should set up all necessary things such as instantiating object, allocate resources
  + Call the method to be tested on the object
  + Verify that the method works as desired
  + Remove anything it set up, clean up after itself
  + Test code is written and compiled for any source in your projects
    - Imports any required classes from testing framework package used
  + The production code does not need to be run when testing
  + Usually have tests for a class situated within a secondary class
    - Account is tested by TestAccount class

Junit

* Framework
  + 2 main frameworks for unit tests in Java: JIUnit 4 and JUnit 5
  + Use JUnit 4 in this module
* Assertions
  + Used to create a unit test
  + 1 unit test contain many assertions
  + A unit test can contain 0 assertions and still be valid, use fail instead
  + Fail causes an error to be propagated
  + EX: a try and catch statement
    - Try {
    - Fail (“No NullPointerException”);
    - Catch (NullPointerException)
    - }
    - If the test wants to create a null pointer exception and doesn’t then it will fail at the fail keyword
* AssertEquals
  + Overloading method
  + assertEquals (String msg, expected, actual)
    - String is optional and is displayed when the assert fails
    - Expected is entered as a literal and is the expected value
    - Value can be method call directly, or the output of a method that gets stored here
* AssertEquals and floating points
  + Remember to specify tolerances or a range that floating point can be within, otherwise may show notEquals
* Import classes from framework that are needed
* Inform the compiler using annotations which methods will be used

Guidelines

* + Each unit test has multiple assertions
  + If 1 assertion fails, the test exits and the rest are not checked
  + Failing code needs to be fixed for the rest to be fixed
  + When a bug is introduced only 1-2 unit tests should fail
  + Don’t add features while code fails
  + Fix tests as soon as they fail and keep adding tests with features