**$5 + 10 CHF = $10 if rate is 2:1**

~~$5 + $5 = $10~~

~~$5 \* 2 = $10~~

~~Make "amount" private~~

~~Dollar side-effects?~~

Money rounding?  
~~equals()~~hashCode()  
Equal null  
Equal object  
~~5 CHF \* 2 = 10 CHF  
Dollar/Franc duplication~~  
~~Common equals~~  
~~Common times~~  
~~Compare Francs with Dollars  
Currency?~~  
Delete testFrancMultiplication?

My Own Steps for TDD:

1. Write To-Do List to determine tests needed
2. Write a test
   * Test is a story about how we want to view the operation
   * Modify interface of previous tests if needed
3. Run all tests and fail (Not compiling is a failure)
4. Make the test work by adding production code
   * Made the test compile immediately by using clean, simple solution.
   * Quick green excuses all of programming sins for the moment
5. Refactor
   * Remove duplicate code
   * Go back to the narrow path of software righteousness
   * Makes tests “more speaking” whenever possible
6. Update to-do lists if possible

Testing Techniques

1. Triangulation
   * Only 2 or more examples could generalize if the code is correct
   * Provides a chance to think of a refactor solution if design thoughts are not coming, Responds to axes of variability question your design is trying to support
2. Quick Green Techniques
   * Fake it – temporarily use constants for variable
   * Use real implementation if obvious and easy
   * Interchange between two
     + Use real implementation until you have an unexpected red bar. If ever that happens, back up and use constants. At the refactor phase, replace constants with real implementation until test passes. When confidence returns, go back to using real and obvious implementation
3. Exception that allows production before test
   * If the prod code is used for debug output
   * Currently in the red phase already