Property-Based TDD: Units of Measure

Write code that can be used to perform calculations with units of measure.

Follow the strict rules of TDD As If You Meant It, but use a property-based test framework.

Units of Measure

Values can only be added or subtracted if they are in the same units.

1m + 3m = 4m
1.25s - 0.5s = 0.75s
1m + 3s = *error: incompatible units*

Multiplying or dividing results in derived units:

5w * 2h = 5wh
3m / 2s = 1.5m/s
4m * 3m = $12m^2$
$16m^2$ / 2m = 8m
(3m/2s) / 2s = $0.75m/s^2$

To keep things simple, you can ignore multiplication and division by unitless values in the language's numeric types, SI prefixes (k, M, etc.), units that are defined in terms of other units (e.g. 1 N = 1 kg m / s^2), and automatic conversion between units of the same dimension (e.g. feet and metres).

The Rules of TDD As If You Meant It

1. Write *exactly one* test, the smallest you can that seems to point in the direction of a solution.
2. Watch it fail.
3. Make the test pass by writing the least implementation code you can *in the test method*.
4. Refactor as required. To create a new, non-test structure, apply these steps to the code created in step 3.
    - *Extract Method* to create a new method in the test class
    - (If you must) move code into an existing method
    - Create classes only to provide a destination for *Move Method*. Populate classes by moving definitions from the tests, not the other way around.

The member of the pair without their hands on the keyboard must be **very strict** in enforcing these rules, especially the refactoring rules.