Recipe book

Overview

We implement a set of classes to keep track of a set of recipes and to convert the recipes between different measurement scales. The assignment specification outlines the required methods, so refer to that specification for those details.

The problem decomposes into three main components: handling recipes, handling different measurement units, and approximating measurements to fit the granularity of the measurement system.

A recipe contains two types of data: ingredients and instructions. Ingredient quantities then come in two formats: fractions or decimal numbers. We need to handle both, so we ultimately convert fractions to decimal numbers internally.

The conversion rules are pairs of equivalent measurement values. The conversion rules are bi-directional: they can convert in either direction. Consequently, given a conversion between units A and B, we end up storing two unidirectional conversion scale: one from A to B and another from B to A. Having the unidirectional conversion scales simplifies the rest of the code since we don't need to worry about which direction we're using a conversion.

The wrinkle in the problem is that we want to represent converted values to a given degree of precision, which is set by fractions of measurements rather than by a number of significant decimal points. That's where the third component comes in: an approximation system. We pre-compute the acceptable fractions that a measurement system can handle and then look up the closest approximation to the converted value that we have to report the measurements.

Files and external data

The implementation divides into 6 classes (Figure 1):

- RecipeBook – acts as an overall manager of the tasks. Its role is to gather all the recipes and conversion rules in one place.
- Recipe – captures all the information for a single recipe, both a list of ingredients and a list of instructions
- Ingredient – captures one line of ingredients for a recipe (quantity, measurement units, and ingredient name)
- ConversionSet – gathers all the conversion rules in one class. It is more than a Collection object because it may need to pair two conversion rules to make a new third conversion rule.
- Conversion – all the information needed to convert one measurement unit into another measurement unit.
- Approximation – a class that knows how to take a given value and give you the nearest fraction…to a give acceptable fraction denominator level.

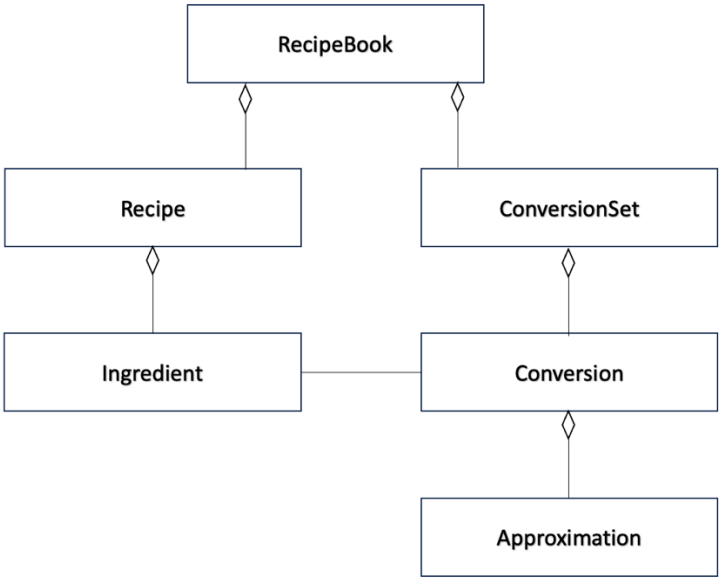

Figure 1 Class interactions

Data structures and their relations to each other

Time and space efficiency are not requirements of the solution. Consequently, we choose the simplest data structures that will do the task.

RecipeBook

- Stores the set of recipes as a Map, so that we can retrieve a given recipe quickly, and stores all the conversions in a single ConversionSet object.

Recipe

- The order of ingredient and instruction lines are important to a recipe. Consequently, we have one List of ingredient objects and a separate List of instruction lines.

ConversionSet

- Each measurement unit conversion is defined by the pair of units: what we are converting from and what we are converting to. That pair then has the unit conversion rules.
- The ConversionSet maps these measurement unit pairs into one Conversion object. Rather than store it as pairs, the ConversionSet has a Map of all "what we are converting from", so each known conversion unit has its bucket of other scales that it knows how to convert to. That bucket of other scales is then also a Map that uses the destination scale as the key and the conversion rules as the value.
- To convert from A to B, we then get the Map of all destination scales for A and extract the rules for scale B from that bucket.

Conversion

- A conversion object contains two items: pairs of measurement units with their numeric equivalents (like 1 tablespoon = 3 teaspoons) and rules on how to round quantities for the measurement system.
- Like the ConversionSet, the pairs of measurements are modelled as a Map of a Map. The outer Map is the unit that we want to convert from. Its Map value is a bucket of all

the units that we can convert to; that bucket is stored as a Map, with the destination unit as the key and the numeric equivalent of units as the value.
- A conversion object also stores two Approximation objects. A measurement system conversion can have two rounding rules: one for small quantities and one for big quantities. The two Approximation objects are for each of these two rounding rules.

Approximation

- An approximation object has two Lists in that operate in parallel, linked by having a common list index. One List stores the decimal values, sorted in increasing order, that match acceptable output fractions for the approximation. The second List stores the string of what that acceptable output fraction looks like.

Assumptions

- No ingredient quantity will mix decimal numbers and fractions.
- A conversion rule has at least one approximation mode.
- The valid denominators for rounding fractions will not cause fractions that we can't distinguish from one another at the computer's precision level

Choices

- The code decides that a recipe must have at least one ingredient. It can have zero instructions, though. A recipe without an ingredient has nothing to work with.
- An ingredient quantity in an instruction is not split across two instruction lines.
- We do not store converted recipes.
- When we approximate fractions, we introduce some error. The requirements say that we need to flag big errors. Consequently, when we do an approximation, we need both the string to show as the approximate value and some way of knowing how far off our value is. The current design clumsily packs both of these values into a string array (coming out of method approximateValue() in the Approximation class) solely out of a lack of desire to create another class to just carry a pair of values around. It's not elegant and should be improved at a later date.

Key algorithms and design elements

Managing paired inputs

Both the notion of converting one measurement system to another and converting one measurement unit to another are identified by an ordered pair: the thing being converted from and the thing being converted to.

The data structure that we use for the pairs is a pair of Maps rather than creating a class that holds the pair. Consequently, the structure is Map< from, Map< to, conversion info>> rather than a more traditional Map< ordered pair object, conversion info > format.

This alternate format lets us access all the set of target conversions for a source conversion without having to do a search. That access then simplifies the process of chaining conversion systems together to make a new conversion system.

Approximating values

When we approximate values, particularly as fractions, we pre-compute the decimal value of all valid fractions and store those values for a later search. Valid fractions are between 0 and 1.

As fractions should be in reduced form, we avoid having to find common factors in fractions to reduce them by ordering how we generate fractions and then only store the first fraction that generates decimal values.

More practically, if we want all denominators up to 8, then we start with the fractions with denominator 2, which would be 0.5 as ½. Next, we generate the fractions with denominator 3, which would be 0.33 and 0.67 as 1/3 and 2/3 respectively. When we next generate fractions with denominator 4, we get 0.25, 0.5, and 0.75 as ¼, 2/4, and ¾. Rather than reduce 2/4, we note that 0.5 has already been computed, so we don't add 2/4 to our set of fractions, keeping the previous ½ instead, which is in reduced form.

Approximation nearness

When we make an approximation to a value, the receiver needs to know by how far the approximation has happened because the requirements ask us to highlight big errors in a conversion.

Reporting these errors is clumsy in the current implementation. When we return an approximation, we return a String array to return multiple values rather than create a new class to just carry all of the disparate data. More specifically, the first string of the array is the string to use in printing while the second string of the array is the decimal value (as a string) of the first string. A receiver of this information can then use the second string, converted back to a decimal number, to test how far the approximation is from the original value asked to approximate.

Limitations

- Unit conversions doesn't add or remove plural notation to units, which can be offputting when the units are in the instructions as sentences.
- We chain together at most two measurement system conversion rules into a new rule. We don't chain more rules together.