Yasmin Fazelinia
yfazelinia@gmail.com
ThoughtWorks Junior Consultant Coding Challenge

PROBLEM TWO: SALES TAX

This application solves the sales tax problem by simulating an actual shopping experience. Data is input as a text file. Upon purchasing items, a receipt is printed that lists the name of all the items and their price (including tax), total cost, and total sales tax.

Basic sales tax is applicable at a rate of 10% on all goods, except books, food, and medical products that are exempt. Import duty is an additional sales tax applicable on all imported goods at a rate of 5%, with no exemptions.

This application prints out the receipt details for these shopping baskets:

INPUT:

Input 1:
1 book at 12.49
1 music CD at 14.99
1 chocolate bar at 0.85

Input 2:
1 imported box of chocolates at 10.00
1 imported bottle of perfume at 47.50

Input 3:
1 imported bottle of perfume at 27.99
1 bottle of perfume at 18.99
1 packet of headache pills at 9.75
1 box of imported chocolates at 11.25

OUTPUT

Output 1:
1 book: 12.49
1 music CD: 16.49
1 chocolate bar: 0.85
Sales Taxes: 1.50
Total: 29.83

Output 2:
1 imported box of chocolates: 10.50
1 imported bottle of perfume: 54.65
Sales Taxes: 7.65
Total: 65.15

Output 3:

1 imported bottle of perfume: 32.19
1 bottle of perfume: 20.89
1 packet of headache pills: 9.75
1 imported box of chocolates: 11.85
Sales Taxes: 6.70
Total: 74.68

To run the app:

From the command line, run:

checkout input/<filename>.txt

In the input folder are three input text files with the data from this problem.

Pass in one of the provided input files in this format to get corresponding output in the form of a receipt.

Results in Output 1:

checkout input/input1.txt

Results in Output 2:

checkout input/input2.txt

Results in Output 3:

checkout input/input3.txt

Custom Shopping Experience

This app can also be run entirely in the command line. In IRB, run lib/command_line.rb. The classes can be instantiated to create items and shopping carts, and all methods can be run in order to alter the cart and purchase it. This will result in a receipt being output for each custom "shopping experience."

Testing:

Testing was done using RSpec. Each class has a corresponding test file in the spec folder.

Run bundle install to install the RSpec testing gem.

To run an individual test:

rspec spec/<filename>.rb

To run all tests:

rspec

Design:

The app has seven classes: Calculator, CommandLine, Government, InventoryFile, Item, Receipt, ShoppingCart. Each class is an abstraction of the real-world object it derives its name from.

1. The Item class is responsible for creating new items and storing the name, price, taxable, and imported attributes. This class sets the tax type of each item as determined by the Government class.

2. The Government class sets the rules for what items are taxable and imported. In this app, only items that contain the following words in their names are tax-exempt: "book, chocolate, chocolates, headache pills." If the item's name contains the word "imported", its imported tax status is set to true.

3. The ShoppingCart class creates a shopping cart and stores items in a set data structure. Items can be added to and removed from the shopping cart, the quantity of a specific item in the cart can be changed, and the cart's contents can be viewed. When a user is finished shopping, the shopping cart is purchased and a receipt is printed.

4. The Calculator class is responsible for the tax calculations for each item based on its tax type. The rounded sales tax and total taxed item price are calculated.

5. The Receipt class is responsible for calculating the shopping cart sales tax and grand total. It is through this class that a receipt is printed out (and output to the console).

- Two classes are responsible for reading input data from text files so that the app can execute:

6. The InventoryFile class reads text files, parses input data so that items can be created, and adds these items to a shopping cart. The data is parsed so that each line of an input text file is a unique item. Each line is parsed using regular expressions into three groupings. The first is the quantity, the second is the item name, and the third is the price (the word "at" is excluded). This data is used to create a new item and add it to a shopping cart according to its quantity.

7. The CommandLine class is responsible for executing the app with a specified input text file and calling the purchase method on the resulting shopping cart. This runs the program and results in a receipt.

Flow:
1. App is executed and a new CommandLine instance is created, with an input text file passed to it.
2. The CommandLine instance creates a new InventoryFile instance.
3. The InventoryFile instance reads the input file and parses it to create Item instances (using the Government class to set tax type).
4. The InventoryFile instance creates a ShoppingCart instance. Item instances are added to the ShoppingCart instance.
4. The CommandLine instance purchases the shopping cart (which uses the Calculator class to calculate taxed item prices and the Receipt class to calculate total cart price and total sales tax) and a receipt is output.

Assumptions:

1. Both item quantity and price are positive integers.
2. The input data is in the following format:
"(quantity)(item) at (price)"
3. Since the items were limited for the scope of this exercise, only the keywords "book, chocolate, chocolates, headache pills" are used to determine if an item is tax-exempt. This does not include all types of food, books, or medical supplies.

3. If an item's name includes the word "imported", it is taxed as an imported item.