Tax

A project by Clayton Lewis. claytonhenrylewis@gmail.com

Overview

This application computes sales tax and total cost for lists of items.

Compilation Instructions

To compile the program automatically, run the following command:

```
./compile.sh
```

To compile the program manually, run the following command in the src directory:

```
javac -cp .:../bin/stdlib.jar:../bin/algs4.jar Tax.java Item.java ItemList.java
Exemptions.java WordNet.java SAP.java BreadthFirstDirectedPaths.java
```

Execution Instructions

To compile and run the program automatically, run one of the following commands:

```
./run-all.sh [sales-tax-rate] [import-tax-rate]
./run [sales-tax-rate] [import-tax-rate] [input-file] [input-file] ...
```

To run the program manually, run the following command in the src directory:

```
java -cp .:../bin/stdlib.jar:../bin/algs4.jar Tax [sales-tax-rate] [import-tax-rate]
[input-file] [input-file] ...
```

To run the program with sample input (sales tax rate = 0.10, import tax rate = 0.05) and for all files in the *input* directory, run the following command:

```
./run-sample.sh
```

Arguments

1. **sales-tax-rate**: The sales tax rate as a decimal number (i.e. 0.10)
2. **import-tax-rate**: The import tax rate as a decimal number (i.e. 0.05)
3. **input-file**: The name of the files to read. Files are assumed to be in the directory *input*. Only pass the file name. Any number of files are accepted. Executing the run-all script will read from all files in the directory *input*.

Example execution

```
./run 0.10 0.05 input1.txt
```

Input Instructions

Shopping Lists

Input files should be .txt files in the *input* directory. Files should follow the following format:

```
<count> <item> at <price>
...
```

Sample input file:

```
1 imported bottle of perfume at 27.99
1 bottle of perfume at 18.99
1 packet of headache pills at 9.75
1 imported box of chocolates at 11.25
```

Exemptions

The user may also configure the items that are exempt from sales tax. Included is a file named *exemptions.txt*. The user may add any items to that list. For best performance, it is recommended that items be elements of the WordNet. An ambitious user might check sysnsets.txt to verify that the item is part of the WordNet. Each item should occupy a new line, as follows:

```
<item>
<item>
...
```

Sample exemptions:

```
book
food
medical_care
medical_instrument
medicine
```

Output Information

Receipts are printed to stdout (generally the terminal) and to a file named *output.txt* in the directory *output*. Receipts are in the following format:

```
<count> <item>: <cost>
...
Sales Taxes: <total tax>
Total: <total cost>
```

Sample output:

```
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 Tax: 6.70
Total: 74.68
```

Solution Description and Methodology

This solution is comprised mainly of the following classes:

1. **Tax**: This class contains the main and makes the necessary calls to run the program given the user input.
2. **Item**: This class represents one item on a shopping list. Contains functions to compute sales tax and total cost.
3. **ItemList**: This class represents a list of Items. Contains function to print the receipt with tax and cost information for all items.
4. **Exemptions**: This class represents a list of all items that are exempt from sales tax. Item has a static Exemptions object which is used to check each Item.

The solution also includes the following classes. These classes use the WordNet database, a useful resource for natural language processing. They also use some resources from Princeton's "Algorithms". Most of this code is borrowed from a previous project (of mine) with WordNet.

1. **WordNet**: This class takes the files *synsets.txt* and *hypernyms.txt* and builds a list of all nouns in the database and a graph showing relationships between those nouns. WordNet is used in this solution to check if items are exempt from sales tax. The method *isAncestor* checks if one noun is an ancestor another noun. It is used in the program to check if a given item has an ancestor that is in the list of exemptions. If so, the item is exempt. [Note: a consequence of this solution is that "medicine" is an ancestor of "music" and thus an item such as "music CD" is exempt from sales tax. While music can be medicinal, the user may not intend for this interpretation. With enough time, options could be added to the application to address cases such as this. For now, the ambitious user might edit the *hypernyms.txt* to remove an unwanted relationship]
2. **SAP**: This class finds the shortest ancestral path between two nodes of a digraph. It is used in WordNet.
3. **BreadthFirstDirectedPaths**: This class builds a graph of all nodes connected to the given one. It is used in WordNet to check for ancestry.

Project development history can be found at this GitHub repository.