CS 5010: Programming Design Paradigms

Fall 2017

HW 4 – Design Patterns, Recursion, and Halloween

Assigned: Sunday, October 15, 2017, Due: Monday, October 23, 2017 by 6pm

College of Computer and Information Science
Northeastern University – Seattle

Assignment Goals

The purpose of this assignment is to help you:

- Get more exposure to, and experience with design patterns,
- Keep internalizing and mastering object oriented design principles (abstraction, encapsulation, modularity, and polymorphism),
- Work with recursions,
- Work with tree data structures, and last but not least,
- Master the virtue of Halloween trick-or-treating :)

Background – Halloween in the Suburbs

October, the spookiest month of the year is here, and it's all about Halloween and trick-or-treating, a wonderful tradition where kids put on their costumes, take the biggest bag or bucket they can find, and walk around the neighborhood, visiting neighboring houses, in search for good tricks, or good candies. There are three unwritten goals of every Halloween (in a suburb):

- If you are a **kid**, you want to get the best treats (even if your parents will never let you eat it).
- If you are an **individual house**, you don't want to be the family that gives out the worst treats. (*Why? Because you don't want to get your house TP-ed. It's a lot of work to clean up all the toilet paper.*)
- If you are **neighborhood**, you want to be one of the coveted neighborhoods that has the best Halloween treats. (*Why? Because its fun to see all those creative costumes and masks, and you know that the kids will follow the candy trail, and come to a neighborhood known for good treats.*)

Your Task

In the day and age of big data, however, the rules of Halloween engagement are changing too, and everyone involved is looking to make **data-driven decisions**. For example, using some crowd-soured information about the past several Halloweens, every **kid** would like to learn what is the best way to visit the neighborhood, to get their preferred collection of threats (*yeah, the kids are getting picky these days :)*).

In this assignment, your goal is to help our young trick-or-treaters with their **data driven goal – finding a way to visit the households in some specific neighborhood, in order to get some preferred collection of treats.**

Attached to this assignment are two CSV files, **DreamCandy1** and **DreamCandy2**. **DreamCandy1** presents a list of desired candies that some kid 1 would like to collect during this year's Halloween trick-or-treating. Similarly, **DreamCandy2** presents a list of desired candies that some kid 2 would like to collect during this year's Halloween trick-or-treating.

Your task is to develop a command-line program, `HalloweenNeighborhoodTraversal`, that receives the name of one or more CSV files, and upon reading one CSV file:

1. Returns whether or not there exists a way for child X to visit the neighborhood, so that at the end of the trick-or-treating visit, child X has candy collection **DreamCandyX** (*We're making a simplifying, yet naive, assumption that kids will not eat any candy before they get home*).
2. If there exists a way to traverse the neighborhood to achieve kid's X dream candy collection, your program should generate an output file describing that neighborhood traversal.

Implementation Details

Please note that one of the goals of this assignment is to gain experience with design patterns. While other implementations might be possible, for this assignment we want you to find a design that uses either the visitor design pattern or the interpreter design pattern as a part of its implementation.

Program `HalloweenNeighborhoodTraversal:` Your program `HalloweenNeighborhoodTraversal` should be a command-line program, that accepts at least two input arguments:

1. The first argument specifies how many names of the CSV files will be specified as arguments.
2. All of the subsequent arguments represents the names of the CSV files that should be analyzed.

Upon reading a single CSV file, your program `HallowenNeighborhoodTraversal` should:

- Validate the file, and recognize all of the listed candies. If not, it should output an appropriate message, but it should not quit until all of the CSV files have been processed.
- For a valid list of desired candies, check whether or not there exists a neighborhood traversal that achieves the desired list. If not, it should once again output an appropriate message, but should not quit until all CSV files have been processed.
- If there exists a neighborhood traversal that achieves the desired list of candy, the program should generate an output file containing that traversal.

Information about the Dream Halloween Neighborhood: To simplify things, we will be focusing all of our analytical efforts on one neighborhood, the **Dream Halloween Neighborhood**. Over the years, the neighborhood kids in the **Dream Halloween Neighborhood** have been collecting and sharing information about the individual households in that neighborhood, and they have observed the following rules have traditionally been followed on Halloween:

1. There exist four different **types of households** in the neighborhood: mansions, detached houses, townhomes, and duplexes. Using the set notation, we can define **Household** as a finite set consisting of hour possible elements, **Mansions, Detached Houses, Townhomes, Duplexes**:

   Household := {Mansion, Detached House, Townhome, Duplex}

2. Traditionally, the households have been giving out four different **sizes of candy: super, king, regular, and fun**:

   Candy size := {Super size, King size, Regular size, Fun size}

3. The households have traditionally been giving out ten different **kinds of candy: Twix, Snickers, Mars, Kit Kat, Whoopers, Milky Way, Toblerone, Crunch, Baby Ruth** and **Almond Joy**:

   Candy := {Twix, Snickers, Mars, Kit Kat, Whoopers, Milky Way, Toblerone, Crunch, Baby Ruth, Almond Joy}

4. Last several Halloweens, mansions have been giving out:
   - **Super size:** Twix, Snickers, Mars
   - **King size:** Kit Kat, Whoopers, Crunch
   - **Fun size:** Toblerone, Baby Ruth, Almond Joy
5. Similarly, last several Halloweens, detached houses have been giving out:
   - **Super size:** Kit Kat, Whoopers, Milky Way, Crunch
   - **King size:** Toblerone
   - **Fun size:** Twix, Snickers, Mars
6. Traditionally, duplexes have been giving out:
   - **Super size:** Toblerone, Baby Ruth, Almond Joy
   - **King size:** Twix, Snickers, Mars
   - **Fun size:** Kit Kat, Whoopers, Milky Way, Crunch
7. Lastly, town homes have become known to traditionally give all of the listed candies, except Milky Way and Crunch, and all of the candy have been regular size.

Hint: You do not have to submit it, but you might find it helpful to graphically represent these crowd-sources knowledge.

Acceptable CSV Files: All of the considered CSV files will be named `DreamCandyX`, where X represents an integer, and represents the desired list of candies corresponding to some anonymous child X.

The items in the list will be separated with the comma, and every item may include up to two pieces of information:

- (**Required information:**) The name of the candy, where the name can be one from the set: `Twix, Snickers, Mars, Kit Kat, Whoopers, Milky Way, Toblerone, Crunch, Baby Ruth, Almond Joy`. If the list contains the candy that is not one of the candies traditionally given out in the **Dream Halloween Neighborhood**, it should be considered an invalid list of desired candies, and an appropriate message should be returned.
- (**Optional information:**) The size of the candy, where the sizes may by `super size`, `king size`, `regular size`, and `fun size`. If a size is not provided in the list, you can assume that the desired candy is regular size.

The lists of desired candies will be **case-insensitive**, but they are ordered. For example, desired candy lists:
List 1: super size twix, snickers, king size mars, kit kat
List 2: Super Size Twix, Snickers, King Size Mars, Kit Kat
List 3: SUPER SIZE TWIX, SNICKERS, KING SIZE MARS, KIT KAT
should be interpreted as the same list of desired candy.
On the other hand, lists:
List 4: Super Size Twix, King Size Mars, Kit Kat, Snickers
List 5: Super Size Twix, Snickers, King Size Mars, Kit Kat
should be considered different lists because the candy order in the list is not the same.

Determining Whether or Not There Exists An Achievable Neighborhood Traversal: There exist many possible ways to encode the assembled knowledge about the **Dream Halloween Neighboorhood**, and one of the challenges in this assignment is determining the appropriate encoding approach. An important part of that decision should be your program's ability to generate neighborhood traversals.

To simplify your search for an appropriate encoding, and an appropriate generation of a neighboorhood traversal, in this assignment we encourage you to consider **a top-down traversal approach**, and in particular **recursive descent**. The top-down approach should allow you to read one desired candy item (for example, super size twix), and to ask if there exist a specific household that could have given you that candy, and in that size.

Note 1: In this assignment, you **are not** allowed to encode the knowledge about the **Dream Halloween Neighborhood** as a look-up table.

Note 2: When implementing your neighborhood traversal, please keep in mind **the cleanliness and the readability of your code**. In this assignment, by **cleanliness and readability**, we mean that the code consisting of a large number of consecutive **if-else** statements, used to make decisions about certain types of things, likely will not be given a full credit. Instead, you may want to consider an **appropriate design patterns here**.

Note 3: Please note that we are not making any restrictions about the number of households of a specific type, and about the amount of candy of a specific type that they may have available to give out. Since this is the **Dream Halloween Neighborhood**, you can assume that the supply of the specific types of households, and specific types of candy will always be sufficient.

Note 4: Please also note that we are not making any assumptions about the geographical ordering of the households in the neighborhood (i.e., at the beginning of the neighborhood, in the middle, at the other end of the neighborhood). You can assume that the required effort to move from one household type to the other is always the same.

Generating an Output File `DreamTravesalX`: For an achievable list of desired candies, `DreamCandyX`, your resulting output file `DreamTraversalX` should be another CSV file, consisting of three columns: `Candy type, Candy size, Household type`, and these three columns also define the header of your CSV file. Every desired candy from the `DreamCandyX` will define a separate row in your output file, `DreamTraversalX`.

For example, consider some input file `DreamCandy5`, given as:

```
Super Size Twix, Snickers, King Size Mars, Kit Kat, Fun Size Toblerone
```

An achievable neighborhood traversal for the given desired list should be stored in the output file `DreamTravesal5`, and it should be organized as:

```
Candy type, Candy size, Household type
Super Size, Twix, Mansion
Regular Size, Snickers, Townhome
King Size, Mars, Duplex
Regular Size, Kit Kat, Townhome
Fun Size, Toblerone, Mansion
```

Assignment Summary

Given the specific behavior of **Dream Halloween Neighborhood**, described in subsection **Implementation Details**, your task is to:

- Develop a command-line program `HallowenNeighborhoodTraversal`, that takes two or more input arguments:
  1. The first argument specifies how many names of the CSV files will be specified as arguments.
  2. All of the subsequent arguments represent the names of the CSV files that should be analyzed.
- Upon reading a single CSV file, your program `HallowenNeighborhoodTraversal` should:
  - Validate the file, and recognize all of the listed candies. If not, it should output an appropriate message, but should not quit.
  - For a valid list of desired candies, check whether or not there exists a neighborhood traversal that achieves the desired list. If not, it should once again output an appropriate message, but should not quit until all CSV files have been processed.
  - If there exists a neighborhood traversal that achieves the desired list of candy, the program should generate an output file containing that traversal.
- Please use the supplied example files to help you develop and test your code. In doing so, note that file `DreamCandy1` should result in a meaningful traversal, but `DreamCandy2` should not. Your code will be tested on those two files, and numerous similar CSV files named `DreamCandyX`, where X represents an integer.
- If there exists a traversal for a given `DreamCandyX` CSV file, your program should generate a corresponding output file named `DreamTraversalX`.
- Similar to Assignment 3, you should make sure that your program works correctly regardless of how your operating system represents paths and files.
- Similar to Assignment 3 again, please include:
  - A UML diagram, corresponding to the design of your program.
  - A brief write-up, which summarizes the main relations between your classes, and how does your program handle errors and/or exceptions.
  - There exist at least two different design patterns (visitor and interpreter) that you might find useful in this assignment. In your write-up, please briefly explain which design pattern have you decided to use an why.

Bonus Part

(Note: The possible bonus points do not reflect the time and effort that will be required to implement and solve the bonus problem. Please focus on the the required part of the assignment first, and attempt the bonus part only after your have fulfilled all of the components of the required part.)

- (**2 points**) Write program `HalloweenNeighborhoodTraversal2` that satisfies the same requirements as the original `HalloweenNeighborhoodTraversal` program, but does so by implementing the design pattern that you didn't originally implement. For example, if your original solution relies on the **visitor design pattern**, implement your bonus solution by relying on the **interpreter design pattern**, and compare the two approaches. For your bonus approach, you are allowed to reuse all of the objects and interfaces that you wrote for the original approach.

– (**1 point**) Please implement program `HalloweenNwighborhoodTraversal3` that relies on the **observer design pattern** to generate the output CSV file. In your writeup explain why or why not would such an implementation be preferable? Once again, for your bonus approach, you are allowed to reuse all of the objects and interfaces that you wrote for the original approach.

What To Submit?

When submitting your assignment, you should continue using the same Maven archetype that we used in Assignments 2 and 3. You will want to submit the following:

– Class `HalloweenNeighborhoodTraversal`. That class will be assumed to be the starting point of your program, and will be called from the command line with input arguments.
– Interface `Visitor`.
– All classes that implement your interface `Visitor`.
– All classes that accept the `Visitor`.
– All other classes that you may have developed for this assignment.
– All classes that you have developed to test your code.
– A UML diagram, corresponding to the design of your program.
– A brief write-up, which summarizes the main relations between your classes, and how does your program handle errors and/or exceptions.
– If you attempted bonus question 1, please include class `HalloweenNeighborhoodTraversal2`, all of the classes it uses, and the new UML that corresponds to this program. Please also update your write-up, to comment on the difference between this bonus approach, and your original approach.
– If you attempted bonus question 2, please include class `HalloweenNeighborhoodTraversal3`, all of the classes it uses, and the new UML that corresponds to this program. Please also update your write-up, to comment on how preferable or not preferable this approach is.