TCSS 342 - Data Structures
Assignment 1 - Burger Baron

Guidelines

This assignment consists of programming and written work. Solutions should be a complete working Java program including your original work or **cited contributions** from other sources. These files should be compressed in a .zip file for submission through the Canvas link.

This assignment is to be completed on your own or in a group of two. If you choose to work in a group of two this must be clear in your submission. Please see the course syllabus or the course instructor for clarification on what is acceptable and unacceptable academic behavior regarding collaboration outside of a group of two.

Assignment

The Burger Baron will make custom burgers including any of the toppings on his famous Baron Burger. The Baron Burger is made on the special Baron Bun and has on it

- Patties - Beef, Chicken, or Veggie
- Cheese - Cheddar, Mozzarella, and Pepperjack
- Veggies - Lettuce, Tomato, Onions, Pickle, and Mushrooms
- Sauces - Ketchup, Mustard, Mayonnaise, and Baron-Sauce

The Burger Baron's official recipe constructs the Baron Burger in this order (top to bottom):

- Pickle - Skewered to the burger with the Baron's Blade.
- Top Bun
- Mayonnaise
- Baron Sauce
- Lettuce
- Tomato
- Onions
- Pepperjack
- Mozzarella
- Cheddar
- Patty - If there is more than one patty then the cheese goes on the bottom patty.
- Mushrooms
- Mustard
- Ketchup
- Bottom Bun

The Burger Baron has customers that will order in different ways and he wants his menus to automatically construct the burger ingredients in the proper order so he can display it to his

gourmet burger chef's. Here are some sample orders:

- Single Veggie Baron Burger.
- Double Baron Burger with no Cheese but Mozzarella.
- Single Burger with Veggies but no Lettuce.
- Double Chicken Burger with Ketchup Cheddar Onions Mushrooms.

That is, most customers use one of two styles to order.

- <Patty Count> <Patty Type> Baron Burger with no <omissions> but <exceptions>
- <Patty Count> <Patty Type> Burger with <additions> but no <exceptions>

The <omissions> and <additions> may include ingredients or categories of ingredients:

- Categories - Cheese, Sauce, Veggies
- Ingredients - Cheddar, Mozzarella, Pepperjack, Lettuce, Tomato, Onions, Pickle, Mushrooms, Ketchup, Mustard, Mayonnaise, and Baron-Sauce

The <exceptions> are always ingredients only and are exceptions to the categories listed in the <omissions> or <additions>.

Each of the components can be omitted when ordering with the following defaults.

- <Patty Count> defaults to Single.
- <Patty Type> defaults to Beef
- <omissions>, <additions>, <exceptions> default to empty.

The Burger Baron takes orders one at a time and gives them a number 0 to 99. The input to your program will be a file with one line per burger.

- Your program should assign each order a number starting with 0.
- Each line will be a string in the format above.
- Your output (to System.out) will be the ingredients of the ordered burger listed from top to bottom.

Formal Specifications

Your assignment is to create a program that can take text orders in the format described and construct a custom burger with all ingredients in the proper order.

Your program will implement the MyStack class as a **linked structure** that must function according to the following interface:

- MyStack <Type> () - a constructor that initializes an empty MyStack.
- boolean isEmpty() - returns true if the MyStack is empty.
- void push(Type item) - this method adds the item to the top of the MyStack .
- Type pop() - this method removes and returns the item on the top of the MyStack .
- Type peek() - this method returns the item on the top of the MyStack.
- int size() - this method returns the number of items in the MyStack.
- String toString() - this method converts the contents of the MyStack to a String for

display.

The MyStack class **may not use or extend** any List type in Java or arrays. It must be **built from scratch**.

Your program will implement the Burger class using only the MyStack class that must function according to the following interface:

- Burger (boolean theWorks) - a constructor that initializes a Burger with only a bun and patty if theWorks is false and a Baron Burger if theWorks is true.
- void changePatties(String pattyType) - this method changes all patties in the Burger to the pattyType.
- void addPatty() - this method adds one patty to the Burger up to the maximum of 3.
- void addCategory(String type) - this method adds all items of the type to the Burger in the proper locations.
- void removeCategory(String type) - this method removes all items of the type from the Burger.
- void addIngredient(String type) - this method adds the ingredient type to the Burger in the proper location.
- void removeIngredient(String type) - this method removes the ingredient type from the Burger.
- String toString() - this method converts the Burger to a String for display.

The Burger class **may not use** any List type in Java or arrays. It must **only use the MyStack class to store ingredients**. **Hint:** You may use a “recipe” stack in your program that has the complete burger according to the recipe.

Finally, your program will also provide a Main class that is used to read in the input file and test and run the Burger class.

- void main(String[] args) - static main method used to run the program and test the program elements.
- void parseLine(String line) - parses a line of input from the file and outputs the burger.
- void testMyStack() - test method for MyStack.
- void testBurger() - test method for Burger.

You may use List types in the main and you may create any other (private or public) classes that you expect will be useful in your simulation. However your solution must use the Burger class as described.

Submission

The following files are provided for you:

- customer.txt - a sample input file.
- trace.txt - a sample trace of my solution using the customer.txt input file.

You will submit a .zip file containing:

- Main.java - the controller and tester for your program.
- Burger.java - the Burger class.
- MyStack.java - your stack data structure.
- {filenames}.txt - input files you created to test your code.

Grading Rubric

This assignment is graded out of 25 points but there are 28 points available.

Correctness 12 points

- To get all 12 points your code must pass all tests in my secret test file. This file has over 100 different orders in it. All orders in this test file are well formed meaning they match the format stated earlier exactly.
- Points will be deducted for failed tests.
- Solutions that fail on some tests will get at most 6 points.
- Solutions that crash on any test will get at most 3 points.
- A 0 will be awarded if your MyStack or Burger class uses a List.
- A 0 will be awarded if your MyStack class implements non-Stack List operations like *contains*.
- One point is deducted if you do not output the input string along with the output.
- One point is deducted if you do not output the correct order number along with the output.

Interface 8 points

- All MyStack methods match the interface provided above. One point is deducted for each mismatch. Three points are deducted if MyStack is not generically typed.
- All Burger methods match the interface provided above. One point is deducted for each mismatch.

Testing 4 points

- All MyStack methods are tested. Points are deducted for missing tests or inadequate tests.
- All Burger methods are tested. Points are deducted for missing tests or inadequate tests.
- One point is deducted if an additional test file is not provided.

Miscellaneous 4 points

- All four points are awarded automatically with the following exceptions.
- One point is deducted if you work in a team of two.
- One point is deducted if your submission is late.
- One point is deducted if you resubmit after your assignment is graded.
- One point is deducted if your submission is not in the correct format (i.e. not in a ZIP, you submit .class files instead of .java, you submit code that needs to be altered to work, etc.)

Tips for maximizing your grade:

- Make sure your classes match the interface structure exactly. Do not change the method name (even capitalization), return type, argument number, type, and order. Make sure all parts of the interface are implemented.
- Only zip up the .java files. If you use eclipse these are found in the “src” directory, not the “bin” directory. I can do nothing with the .class files.
- All program components should be in the default package. If you use a different package it increases the difficulty of grading and thus affects your grade.
- Place your name in the comments at the top of every file. If you are a group of two make sure both names appear clearly in these comments.