TCSS 342A - Data Structures
Group Project 1 - Burger Baron
Due Date: January 25, 2017 (Wednesday) 9:30am via Canvas

This group project 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 project is to be completed on your own or in a group of 2 or 3 members. One submission per group. State the official names (as appear in myUW) of all your group members clearly in your submission.

This project will be graded out of a total of 16 points.
Breakdown of points: 8 (code compiles and pass tests) + 3 (documentation) + 5 (analysis)

Project Description
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 and 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 but does not alter 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**. In other words, it is ok if you implement your own stack from scratch using an array or a linked list. Remember that the project asks for algorithm analyses as well, so you want to keep the running times of push, pop, peek etc. in mind when choosing an underlying data structure.

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 removePatty() - this method removes one patty from the Burger down to the minimum of 1.

- 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:** The Burger Baron chef's tend to stack ingredients in **two stacks** on top of each bun and join them top-to-top in the final construction.

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.

Note that the supplied **Main.java** is for reference only. You are free to write your own Main. Just *make sure that your **Burger.java** and **MyStack.java** work with the Main I supplied*. Make sure you submit your own Main.java to show how you tested your code. In fact, you need to modify Main.java in order to properly test your code since Main.java has the filename "customer.txt" hardcoded.

You may create any other (private or public) classes that you expect will be useful in your simulation.

Analysis

In addition to this programming assignment you will also complete a detailed code analysis of your *changePatties* method implementation. You will assume for your analysis that the *n* is the current number of ingredients in the Burger. For the *changePatties* method give:

- A line-by-line analysis of operation costs.
- A big-oh expression of the cost of the method.

Also, in your analysis, include **documentation of how you tested your code**. Submit your test input files, and explain what condition that your test input files test for. See programming guidelines on Canvas for additional guidance.

Submission

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 used to test your code.
- Analysis.pdf - a pdf with your run time analysis of changePatties.

Points will be deducted if you do not follow the above file format or filenames.

Grading Rubric

MyStack

- push is efficient and correct.
- pop is efficient and correct.
- size is efficient and correct.
- isEmpty is efficient and correct.
- toString is efficient and correct.
- MyStack is generic.

Burger

- Constructor options.
- addPatty is efficient and correct.
- changePatties is efficient and correct.
- addCategory is efficient and correct.
- removeCategory is efficient and correct.
- addIngredient is efficient and correct.
- removeIngredient is efficient and correct.

Main, Input Files

- File I/O carried out efficiently.
- parseLine is efficient and correct.
- testMyStack is efficient and correct.
- testBurger is efficient and correct.
- Additional input files provided.

Analysis

- Line-by-line analysis is correct.
- Summations used properly.
- All terms are gathered into the sum properly.
- Simplification is correct.
- Correct big-oh bound.

Tips for maximizing your grade:

- Make sure your classes match the interface structure exactly. I will use my own controller (Main.java) and test files on your code and it should work without changes. Do not change the method name (even capitalization), return type, argument number, type, and order. Make sure all parts of the interface are implemented.
- 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 2-3 make sure all names appear clearly in these comments.

Grading logistics

I will compile your code using

```
javac *.java
```

I will use the provided “Main.java” to test your code. You are free to write your own Main.java, but your routines must work with my “Main.java”.

Sample input file: (*customer.txt*)

```
Burger
Double Baron Burger
Triple Chicken Burger with Onions Cheese but Cheddar
Baron Burger with no Veggies Sauce but Baron-Sauce
```

I will execute your code
```
java Main customer.txt
```

Expected output:
```
Processing Order 0: Burger
[Bun, Beef, Bun]

Processing Order 1: Double Baron Burger
[Pickle, Bun, Mayonnaise, Baron-Sauce, Lettuce, Tomato, Onions, Beef,
Pepperjack, Mozzarella, Cheddar, Beef, Mushrooms, Mustard, Ketchup,
Bun]

Processing Order 2: Triple Chicken Burger with Onions Cheese but
Cheddar
[Bun, Onions, Chicken, Chicken, Pepperjack, Mozzarella, Chicken, Bun]

Processing Order 3: Baron Burger with no Veggies Sauce but Baron-Sauce
[Bun, Baron-Sauce, Pepperjack, Mozzarella, Cheddar, Beef, Bun]
```

Yet another sample input file:
```
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 and Mushrooms
Triple Baron Burger
```

Expected output:
```
Processing Order 0: Single Veggie Baron Burger
[Pickle, Bun, Mayonnaise, Baron-Sauce, Lettuce, Tomato, Onions,
Pepperjack, Mozzarella, Cheddar, Veggie, Mushrooms, Mustard, Ketchup,
Bun]

Processing Order 1: Double Baron Burger with no Cheese but Mozzarella
[Pickle, Bun, Mayonnaise, Baron-Sauce, Lettuce, Tomato, Onions, Beef,
Mozzarella, Beef, Mushrooms, Mustard, Ketchup, Bun]

Processing Order 2: Single Burger with Veggies but no Lettuce
[Pickle, Bun, Tomato, Onions, Beef, Mushrooms, Bun]

Processing Order 3: Double Chicken Burger with Ketchup Cheddar Onions
and Mushrooms
[Bun, Onions, Chicken, Cheddar, Chicken, Mushrooms, Ketchup, Bun]
Processing Order 4: Triple Baron Burger
[Pickle, Bun, Mayonnaise, Baron-Sauce, Lettuce, Tomato, Onions, Beef,
Beef, Pepperjack, Mozzarella, Cheddar, Beef, Mushrooms, Mustard,
Ketchup, Bun]
```