***README.txt* – Restaurant Project v4.2**

**Name: Tim Righettini**

----------------

Hello Mr. Grader

----------------

This is the readme doc about the first iteration (v4.1) of the restaurant project. Mainly, this document will tell you how to run all the test cases described in the Requirements section for v4.2.

The format will be like this:

1. The name of the case, EXACTLY as stated on the website, will appear first.

2. Afterward, a description of how to run the particular case will be elaborated upon to easily guide you to grading this project simply and efficiently.

\*Note 1: EVERY test case can be run in ONE simulation, assuming no concurrent medication exceptions occur. The GUI allows accommodates you to test every case in this assignment with relative case, once acquainted with the changes made to the GUI.

\*Note 2: Make sure to type in integers for food numbers and doubles for money values. If you don’t the values will not be accepted by the GUI (the program won’t crash, but the action will not perform on the text box until you enter in the correctly formatted value).

\*Note 3: For Reference, I kept in the v4.1 test cases, just in case you need them.

Now, on to describing how to test each one of these cases:

***For v4.2:***

**Scenario #1:** Make sure your agents are thread-safe.

How to Run/Grade:

This should be really simple to check. All you need to do is search through my code and find my Synchronized Lists and Loops. To make this task easier in Eclipse, do the following for the six main agents:

1. Use the find command and search for the word “List” (NOT as a whole word), and you will find all of my synchronized lists.

2. Use the find command and search for the word “for (” (EXACTLY like that), and you will find all of my synchronized loop searches.

**Scenario #2:** Implement Multi-step Actions

How to Run/Grade:

In the WaiterAgent, the method takeOrderAndGiveOrderToCook(...), and in the CustomerAgent, the method callWaiterAndOrderFood(), implement the multistep actions, along with the accompanying multiAction semaphore and other messages. These are the places to look to grade this functionality.

**Scenario #3:** Incorporate Shared Data Between Cook and Waiter

On the GUI, you will see that there is a new list for adding “SDWaiters.” When you click on the add button, you will add shared data waiters into the simulation, denoted by s1, s2,..., sN. These waiters will complete the shared data operation between the cook and themselves by using the revolving stand when active with the customer.

The revolving stand instance, or the Restaurant Producer-Consumer Monitor, is located in Restaurant within the layoutGUI directory – I did this because the cook and waiters already had a reference to this restaurant file. The actual code for this shared data structure is located where the other main agents are located.

How to Run/Grade:

1. Click the “Add” button for SDWaiters, and add in as many shared data waiters as you want.

2. Assign them to a customer, and everything will take off from there.

**Scenario #4:** Unit Test Your Cashier

How to Run/Grade:

This is really easy, if you are using Eclipse. If that is the case, just run the test class and you will see the results of every test method within the JUnit dialog box. Every Test Method name explains what the test is, and comments further elaborate upon it within each method. It is located in the test directory.

***For v4.1:***

|  |  |
| --- | --- |
|  |  |

**Normative Scenario #1:** 1 of every type of agent, no market interactions, customer orders, pays, and leaves

How to run:

Thankfully, this is really easy. All you have to do is:

1. Add waiter and customer agent as normal.

2. Add market agent in new list.

3. Set the customer to hungry

Everything will take care of itself past this point.

\*You will see that the money values of the customer update in real time once they change. This is the case for EVERY GUI value that emulate a number or boolean, which helps you see what is going on.

**Normative Scenario #2:** No customers, cook orders low items from market, when food arrives, customers arrive

If you have already created the market, then great! Else, refer to step 2 in normative scenario #1.

To force the cook to order items, just set the inventory of ONE of the cook’s food items to less than or equal to the threshold, which is 3.

How to run:

1. Click on the market you created, this will bring up the market’s AND cook’s inventories.

2. Click any text box for the cook, set it to an integer below or equal to 3, then press ENTER/RETURN to update the value. If you do not press ENTER/RETURN, this will not work

\*Everything will take care of itself past this point.

\*You will see the values update on the GUI as the process operates.

**Normative Scenario #3:** Multiple customers, multiple waiters, cashier operating normally, no running out of food

How to run:

This is really just an extension of Normative Scenario #1. Add more waiters and customers, set the customers to hungry, and then watch the restaurant in action.

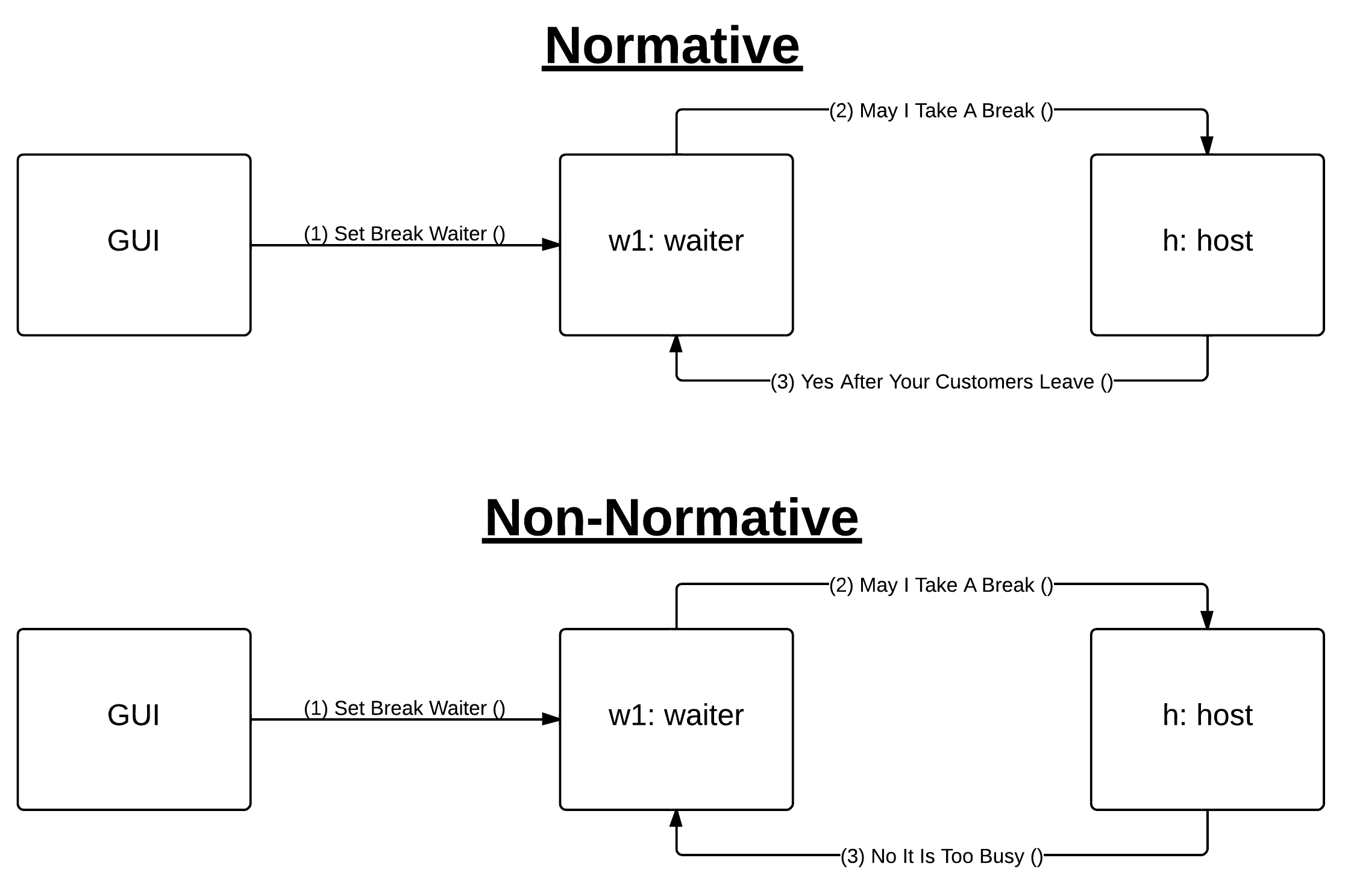
\*When adding new customers, make sure to add them one at a time with unique names. Otherwise, equality operations may mess up in my code, leading to problems exiting outside the scope of my upgraded code.

**Normative Scenario #4:** Waiter wants to go on break, he's told it's ok, goes on break when he finished all his current customers; goes on break, then goes off break.

How to run:

This is where it begins to get interesting. Select one of the waiters. You will notice that the “On Break” button is active when you click it, but be wary as the results will not show right as you click the button, for reasons which I will explain now:

1. When you click the button, the waiter agent and host agent will go through the process mentioned in the interaction diagram in my design:



2. The GUI button will only become checked IF AND ONLY IF the host allows the waiter to break at his/her point in the messaging process, and the onBreak boolean value in waiterAgent is true.

3. This will take some time to activate, since messaging is not instant. Pay attention to the command line for this case, even though the end result DOES show in the GUI “On Break” checkbox.

**Non-Normative Scenario #1:** Customer orders, food has run out, cook has obviously ordered but food hasn't arrived, Customer makes new choice.

How to run:

This case is a dual mechanism. There are 2 things to test for. The first is easy to test, but the second requires a little bit of timing.

**Case 1:** Food is out and cook does not have a delivery coming soon.

To test this, it is simple. Just set the inventory of an item to 0 for the cook that the customer is ordering. The randomization algorithm is still in the customer agent, but I knew that you would HATE having to wait for the “lucky” time that this would actually happen, so I added some GUI functionality to force the customer to ONLY ORDER CERTAIN ITEMS.

To set this up, do the following steps from the GUI:

1. Click on a customer’s button and find the “Choose what the customer orders” panel.

2. Uncheck all but one food value.

3. Set this food value to 0 in the cook’s inventory for this food item, and to really make sure that this will work, do the same to the correct section of the market’s inventory beforehand so nothing is ordered by the cook. If there is more than one market, repeat this step for the other markets.

4. Set the customer to hungry, and then watch the customer attempt to order the food. It won’t arrive, assuming the settings are correct based upon the previous step.

5. I can also guarantee that the customer will stay stuck in an infinite while loop attempting to order this unavailable item (a busy wait in the customer code, NOT infinite messaging to the waiter/cook – the latter would be bad design), because if he/she cannot order the only item he/she is allowed to order, he/she will be pondering what to do next. Make sure to allow the customer to order another item by checking the other boxes again (a shortcut is to click the lone checked checkbox to make them all checked again).

6. Once the customer can choose other items to order, the rest will play out the same as the settings you had previous.

**Case 2:** Food is out, but a delivery IS coming soon. Cook puts the order on hold until the shipment arrives then cooks it.

Take a look at the market agent again, and you will see that there is a checkbox called “Randomized Order Times.” To easily check for this case, uncheck that box – the orders will always arrive at the same time, and always within the period where the cook is willing to shelve an order instead of remove it. After un-checking that box, do the following steps.

1. Set the customer to only order 1 food type, DON’T click “Hungry?”

2. Make sure you take note of where to set the cook’s inventory for this food type to 0, and set the market’s inventory to 10 or more if below 10.

3. Once you have ascertained, or found, this information, set the customer to Hungry.

4. THIS TIMING IS CRITICAL: As the waiter is moving with the customer to seat him/her, set the cook’s inventory for the food that the customer is soon going to order to 0. If you are too slow, the order will NOT BE PROCESSED IN TIME and **case 1** will take over. Think of this like an iPhone application, and that you are trying to save the customer’s order before the time runs out (when the customer orders).

5. Assuming the cook’s market order was correctly processed for the food item before the customer orders, the cook will hold the order and begin cooking it as soon as the shipment for that food type arrives.

\*\* This case was not explicitly mentioned in the requirements, but I did it anyway for the sake of realism (and possibly extra credit).

\*\* Make sure the customer can afford the item, and the he/she has not run out of money, or else the customer will leave the restaurant!

**Non-Normative Scenario #2:** Cook orders from favorite Market, but is told they can't fulfill his order; must order from one of his backup markets.

How to run:

1. Click the button of the top-most market, and set its inventory for an item to soon be ordered to 0.

2. If you have not done so, create a second market, and make sure that the market has AT LEAST an inventory of 10 for the item to soon be ordered.

3. Set the cook’s inventory for that item below the threshold, and then everything will take off from there in the restaurant.

**Non-Normative Scenario #3:** Customer doesn't have enough money to order anything and leaves.

How to run:

1. Go to any customer in the GUI, and make sure to change the “Total $” textbox to a value less than that of the price of a salad.

2. The customer will leave the restaurant after the waiter seats him/her. Look to the console for this action when it occurs.

**Non-Normative Scenario #4:** Customer has only enough money to order the cheapest item.

How to run:

This is similar to how Non-Normative Scenario #3 works.

1. Set the “Total $” of a customer to less than the price of a Pizza. That way, only a Salad will be bought by the customer.

2. Make sure the “Pay Fully” checkbox is checked, or else the customer will choose other items and go into debt.

3. Click “Hungry?,” and then the restaurant simulation will take everything from there.

\*\* Make sure that the customer can actually order a Salad from the “Choose what the customer orders:” panel.

**Non-Normative Scenario #5:** Customer orders, eats, but hasn't enough money to pay.

How to run:

This is similar to how Non-Normative Scenario #4 works.

1. Set the “Pay Fully” checkbox as UN-CHECKED, or else the customer will not go into debt.

2. Make sure to set the customer’s “Total $” to be less than that of a salad.

3. Click “Hungry?,” and when the customer pays the cashier, and the customer does not have enough money, the cashier will send the customer how much he/she has to pay the restaurant back.

4. To pay the restaurant back, add some money into the customer’s “Total $” textbox, making sure that it values more than the debt.

5. Click the “Pay Debt” button after giving the customer the money, and if you gave the customer enough money, you will pay off the debt.

\* Congratulations! You just helped a customer with budgeting!

**Non-Normative Scenario #6:** Waiter wants to go on break, he's told it's NOT OK and must keep working.

How to run:

1. Click the “On Break?” button for a waiter while the waitList, or the number of customers waiting to be served in line, is > 0.

\* The host will NOT allow the waiter to break if there are still people to be served, or if the other waiters are on all on break. Look in the console for the messaging to see the results of this scenario, because the GUI button will not change states as a result of this.

**Non-Normative Scenario #7:** Customer comes to restaurant and restaurant is full, customer is told and waits.

If you didn’t notice already, there is also a “Will Wait” checkbox in the GUI customer panel. If this checkbox is checked, then this case will be enacted. Else, the Non-Normative Scenario #8 will be enacted.

How to run:

1. Make sure that a combination of four customers are seated or waiting to be seated.

2. Once the condition is achieved, check the “Wail Wait” checkbox if it is not checked already, and then add a fifth customer to the line by setting the fifth customer to hungry.

3. Assuming the “Will Wait” checkbox is checked, the rest of the scenario takes off from here, and the result is that the customer sits in the waitList.

**Non-Normative Scenario #8:** Customer comes to restaurant and restaurant is full, customer is told and leaves.

How to run:

1. Make sure that a combination of four customers are seated or waiting to be seated.

2. Once the condition is achieved, UN-check the “Wail Wait” checkbox, and then add a fifth customer to the line by setting the fifth customer to hungry.

3. Assuming the “Will Wait” checkbox is un-checked, the rest of the scenario takes off from here, and the result is that the customer leaves the restaurant.