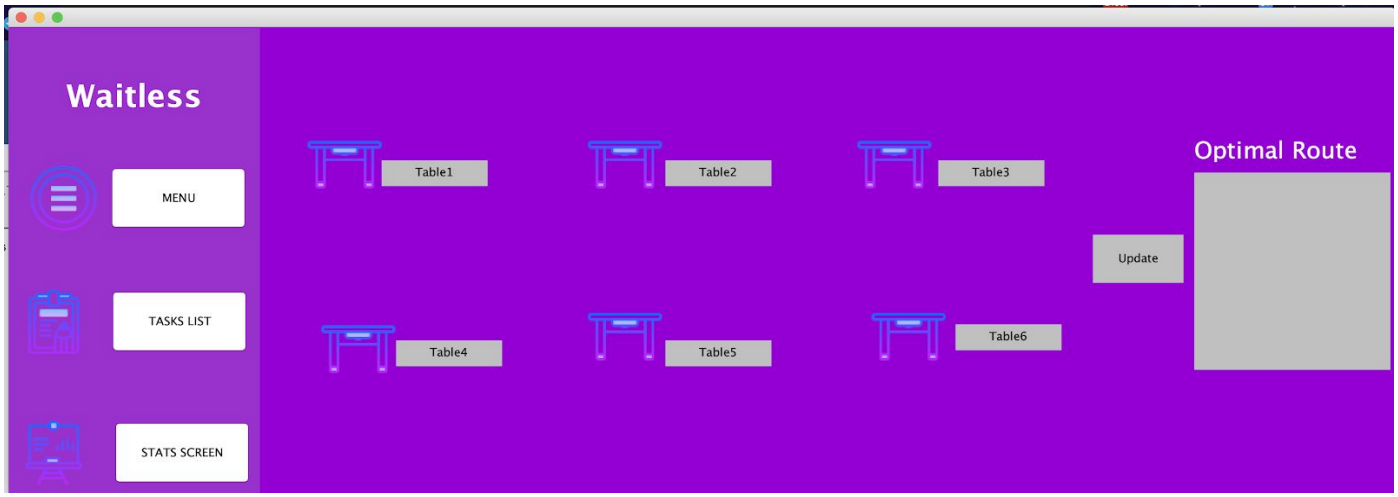
Waitless
Testing and Inspection Report



A Document for
Generating Consistent Professional Reports

Prepared by:
Maaz Ahmed,Bryan Gutierrez, Ammar Idrees, Zaid Patel
***for use in* CS 440**
at the
University of Illinois Chicago

April 2020

Table of Contents

List of Figures

List of Tables

I Project Description

1 Project Overview

Our Project is an application designed for waiters within a restaurant setting. The app offers a variety of functionality such as the ability to access a table, and determine if it's used or not. There is also a functioning menu, where users are able to add or delete items, and be able to view the orders of specific tables. Waiters are also able to access a task list, which will have current tasks that need to be tended to, implemented with a functional timer. An optimal route for the waiter is also calculated and shown, to give them the ability to serve customers efficiently. Finally, there is also a statistics screen, giving the waiter the ability to view different stats in the restaurant such as open tables and revenue generated.

2 Project Domain

Some relevant information regarding the domain of the project is that it is fully completed in Java, and the GUI was completed utilizing Java Fx. Some requirements for the project is that each functionality is created within its own package, Furthermore, for its own current prototype it is assumed that there are six tables within the restaurant. The stats and algorithm utilizes that assumption for the time being. In the future, the application will need to be transitioned into being fully functional with a tablet, and be able to utilize a database to store the information per waiter.

3 Relationship to Other Documents

The relationship between our product and the description/requirement document is quite similar. The overall goal is to increase efficiency in production and revenue, which should be obtainable with the most efficient paths to serve. Furthermore, the context diagram is quite similar as we notify the new server, order food, and then serve the marking table. Furthermore our tasklist follows the UML diagram quite similarly as we have a User Queue and Task that both extend a Task Controller.

4 Naming Conventions and Definitions

Task List: The list is used only to store basic tasks and timer tasks, not menu orders. Tasks can be added manually for a specific table and they will be added automatically when a timer for a table runs out.

Stats Screen: This screen is used to show statistics such as available/occupied tables, wait time, and revenue but it is not used to place customers in the tables.

Table Display: This is the main UI of the application where the user can select to move between three options(Task List, Stats Screen, and Menu), can select table availability, and can view the optimal route.

Optimal Route: The optimal route list is based on the price being paid for each table and the amount of time since the order was placed. This route will tell the waiter who they should attend first.

4a Definitions of Key Terms

Menu: The menu is used to order food for each table. This is not used by the table/customers. The waiter will enter the food in manually thru a tablet. They must first enter the table number as input to select the food. This can also be used to view the orders of all other tables.

Table: A table is used to seat customers and is occupied when the button is pressed(turns red). Once the table is pressed, they can then place in order from the menu and be included in the Optimal Route List

Task: A task is a basic job that a waiter must complete as soon as possible. This might include, for example, checking on each table when their timer is up or other jobs like taking water to a table. A task is not a menu order.

Restaurant: This is the name that was used for a table object, although the name used for this class could have been different such as “Table”.

4b UML and Other Notation Used in This Document

This document generally follows the Version 2.5.1 OMG UML Standard.

4c Data Dictionary for Any Included Models

MenuItems: A hash table which includes all the items in the menu. More items can be added at any time but any example would be:

->MenuItems = (“Steak”,30) + (“Lasagna”, 13) + (“Pizza”, 12) + (“Salmon”,17)

rowData = {{"Date and Time", dtf.format(now)}+ {"Tables Occupied", 6 - Tabledisplay.*getAvailable*()} + {"Tables Open", Tabledisplay.*getAvailable*()} + {"CustomersWaiting","0"}+{"Waittime(sec)",newDecimalFormat("#.##").format(wai tTime.*waiting*())} + {"Tasks Open", Queue.*getLength*()} + {"Tasks Completed", Queue.*getDone*()} + {"Revenue ($)", *rev*} + {"Fun Fact of the Day", "Please give an A :)"}}

pq: The priority queue used to determine which table goes first when being served

Check(An object containing a timer,String tableN, int second): It also contains an inner class CheckTable which creates a separate thread for the timer to run

Task: An object that consists of a static int count, int taskID, String taskCreator, Float timeCreated

User: An object that consists of Queue userQueue, static int countUsers, int userID, String userPosition, String userName

Queue: A class that contains Node head, static int length, static int done. It includes basic queue methods as well

Node: A linked list object containing Task singleTask, Node next, Node tail.

Table: An object containing String name, double cost

II Testing

1 Items to be Tested

- SD: Stats Display
  - SD - REV: Displays current revenue
  - SD - SW: Displays proper time for stopwatch
  - SD- OT: Displays number of open tasks
  - SD - TA: Displays how many tables are occupied
- MN: Menu
  - MN-A: Add an orders to tables
  - MN-VO:Displays a tables order in their designated column
  - MN-D:Delete an item from an order
- TD: Table Display
  - TD - L: Displays table layout
  - TD - TA: Display if table is occupied/free
  - TD - M: Displays Menu
  - TD - TL: Displays Task List
  - TD - SS: Displays Stat Screen
  - TD - OR: Display Optimal route from priority queue
- TL: Tasks list
  - TL - N: Adds a new task
  - TL - D: Deletes a task
  - TL - U: Update button working properly

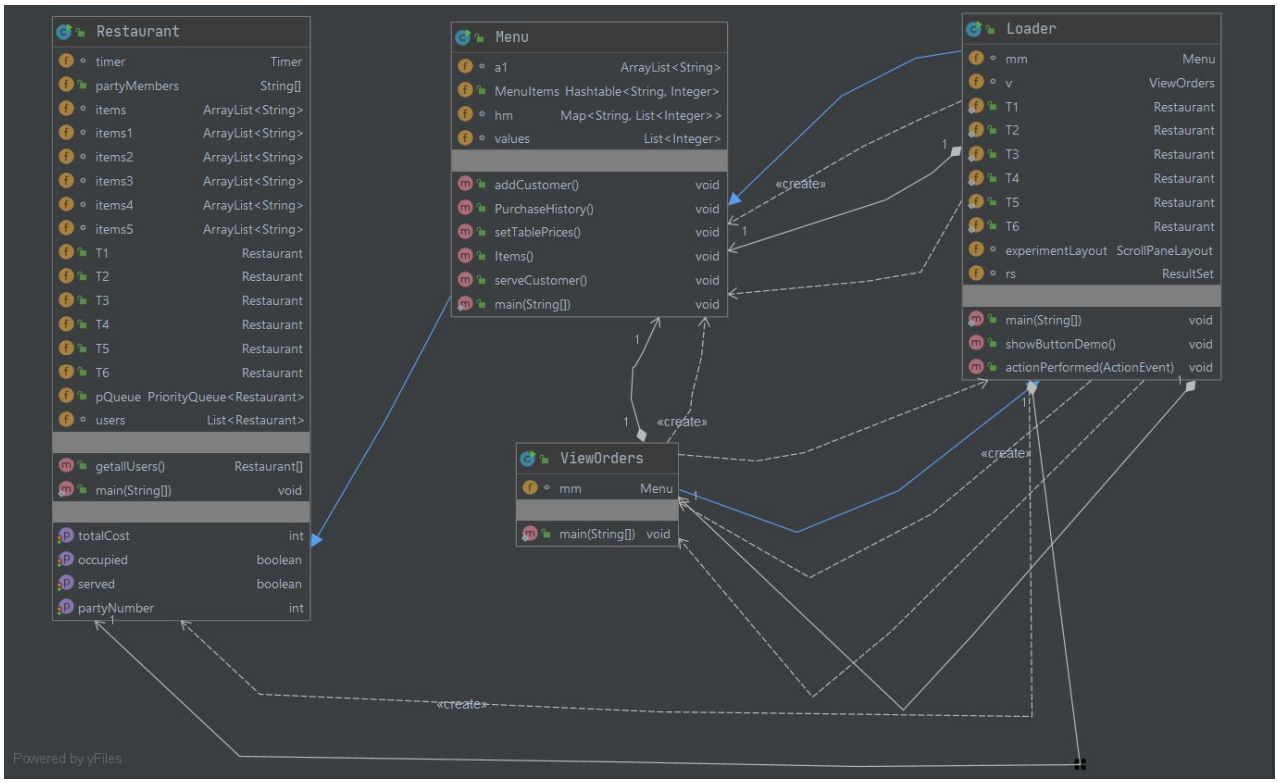


Figure 1 - Menu Display UML

Figure 2 - Stat Display UML

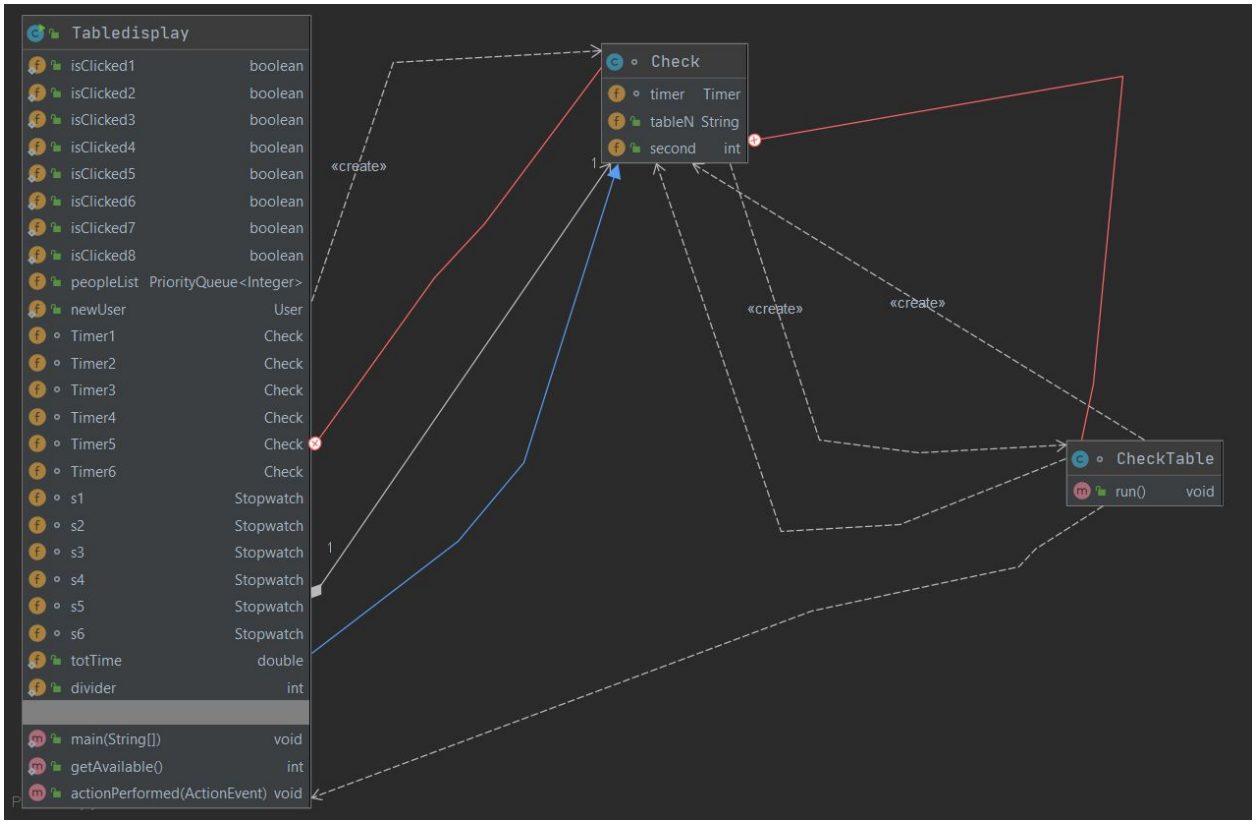


Figure 3 - Table Display UML

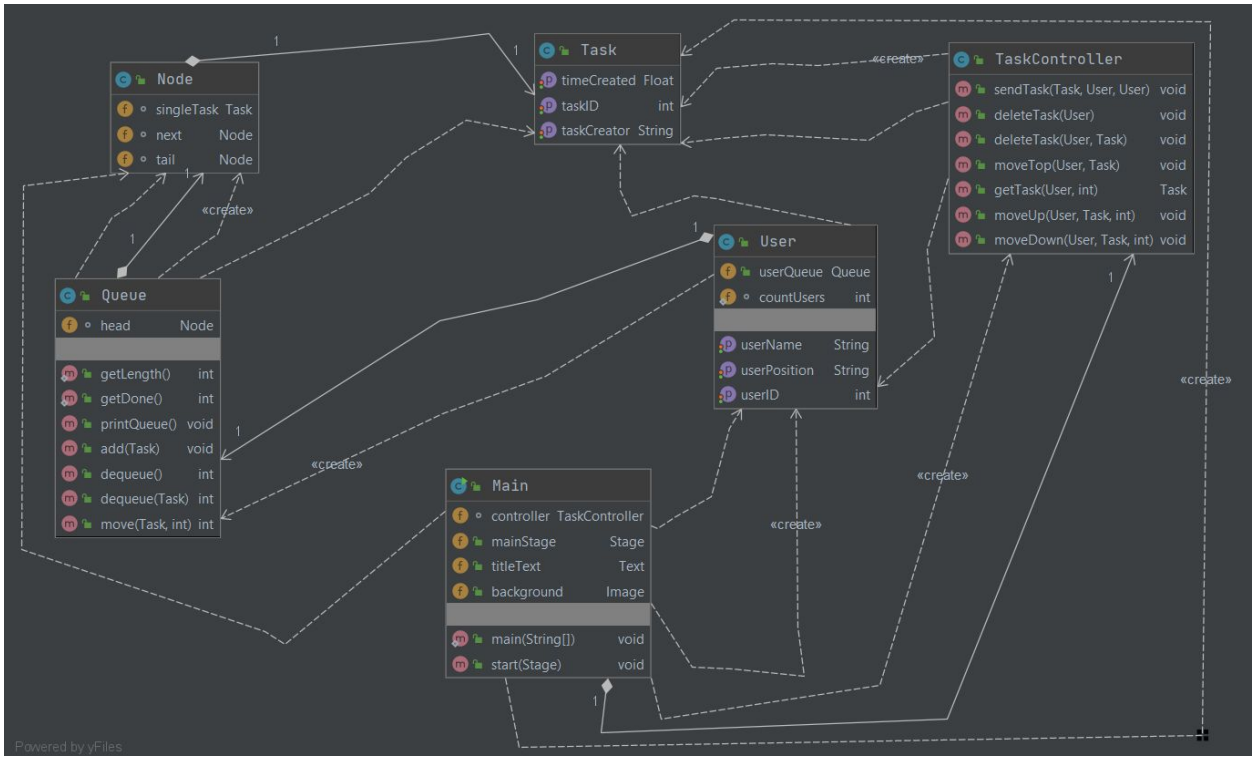


Figure 4 - Task List UML

2 Test Specifications

SD - REV: Display current revenue

Description: Testing the Stats display class, to ensure within the array the proper amount of revenue is being displayed.

Items covered by this test: The test ensures that the correct amount of revenue is present based on when users add or delete an item

Requirements addressed by this test: N/A

Environmental needs: The testing could be done utilizing JUnit, or using the automation tool Maven.

Intercase Dependencies: There are dependencies with the menuDisplay package, as the revenue is also dependent on the Loader class.

Test Procedures:

```
@Test
public void testRevenue(){
    Loader l = new Loader();
    Loader m = new Loader();
    statsDisplay s = new statsDisplay()
    m.getTotalCost = 30;
    l.getTotalCost = 10;
    assertEquals(40, s.rowData[7])) }
```

Input Specification: The case will take in integers based on a tables getTotal and add them all up.

Output Specifications: The output in our case needs to be a positive number, and for this test case 40.

Pass/Fail Criteria: The test would pass if the returned element is a positive integer and for our case 40.

SD - SW: Proper time outputted from StopWatch

Description: Testing the timer function, to ensure proper time within seconds is being outputted from the stopwatch

Items covered by this test: The test ensures that the waitTime function properly outputs a correct positive integer..

Requirements addressed by this test: N/A

Environmental needs: The testing could be done utilizing JUnit, or using the automation tool Maven.

Intercase Dependencies: There are no dependencies for the class, but statsDisplay utilizes the waitTime.

Test Procedures:

```
@Test
public void testTimer(){
    int x = 0;
    assertEquals(x>0. waiting()) }
```

Input Specification: The case will take in no integers but for stop watch will check if a double is returned and an integer for the waitTime.

Output Specifications: The output needs to be a positive integer above 0.

Pass/Fail Criteria: The test would pass if the returned element is a positive integer which would be deduced as an amount of seconds.

SD - OT: Display number of open tasks

Description: Testing the Stats display class, to ensure within the array the proper amount of open tasks are correct.

Items covered by this test: The test ensures that the correct amount of tasks is present based on when users add or delete an item

Requirements addressed by this test: N/A

Environmental needs: The testing could be done utilizing JUnit, or using the automation tool Maven.

Intercase Dependencies: There are dependencies with the taskList package, as the tasks are also dependent on the taskList class.

Test Procedures:

```
@Test
public void testTaks(){
    statsDisplay s = new statsDisplay()
    taskList t = new taskList();
    t.queue.push(task1);
    t.queue.push(task2);
    t.queue.push(task3);
    assertEquals(3, stats.rowData[4]) }
@Test
public void testArray(){
    statsDisplay s = new statsDisplay()
    assertFalse("Queue should be full". s.objects[1].isEmpty()) }
@Test
public void checkTables(){
    statsDisplay s = new statsDisplay()
    assertEquals(6, s.objects[2]) }
```

Input Specification: The case will take in a queue, and check on the size of them.

Output Specifications: The output in our case needs to be a positive number, and for this test case should be 3 based on the queue size.

Pass/Fail Criteria: The test would pass if the returned element is a positive integer and for our case 3.

SD - TA: Testing Table Availability

Description: Testing the Stats display class, to ensure within the array the proper amount of tables available are accurate

Items covered by this test: The test ensures that the correct amount of tables is present based on when the user marks the table in use or not in use.

Requirements addressed by this test: N/A

Environmental needs: The testing could be done utilizing JUnit, or using the automation tool Maven.

Intercase Dependencies: There are dependencies with the taskList package, as the tasks are also dependent on the tableDisplay class.

Test Procedures:

```
@Test

    public void testTables(){

        statsDisplay s = new statsDisplay()

        assertFalse("Queue should be full". s.objects[1].isEmpty()) }

@Test

    public void checkTablesOpen(){

        statsDisplay s = new statsDisplay()

        assertEquals(6, s.objects[2]) }
```

Input Specification: The case will take in information into the objects array and check whether the correct numbers are outputted.

Output Specifications: The output in our case needs to be a positive number, and for this test case 6. For the other assert it should be false.

Pass/Fail Criteria: The test would pass if the returned element is a positive integer and for our case 6, and the other assert statement properly returned false.

MN-A: Adding items to order

Description: Testing whether adding an item to the order of a table from the menu works properly. This includes viewing the "View Orders" screen.

Items to be covered by this test: MN

Requirements addressed by this test:

Environmental Needs: Testing could be done using JUnit or Maven

Intercase Dependencies: N/A

Test Procedures:

```
@Test

public void testMN-A() {

//Once a button has been clicked between the selection of orders

//For example, if Table 1 orders Lasagna

assertEquals("Lasagna Ordered by Table "T1" Cost is $13",
  l1.lastElement(), "Not correct output);

assertEquals("Lasagna", T1.items.get(0), "Should be Lasagna");

assertEquals(13, T1.getTotalCost(), "Only item in List is Lasagna = 13");

}
```

Input Specifications: As for input, clicking a button depending on the item would add the item to the order. The table number was also needed for input

MenuItemsList:

Lasagna,Pizza,Steak,Salmon, Cake, Milkshake, Ice Cream, Donut

Tables Input: "T1","T2","T3","T4","T5","T6"

Output Specifications:

- For the first test case, output should be "Lasagna Ordered by Table "T1" Cost is $13". This test basically tested a list "l1" that would output what was being ordered every time an item was clicked.
- For the second test case, "Lasagna" should be the output. This is different from the first because this one is specific to a table number and this is just the name of the item that was last ordered.
- For the last test case, 13 should be the output. This time it is an integer because this tests that the item is being added to the total cost of the order of the table.

Lasagna Ordered by Table "T1" Cost is $13"

Pass/Fail Criteria: Passing in this means that the correct item is being added to the order and to the correct table. This means that the item should be added to the arrayList items respectfully to each table and the total cost should be updated every time an item is added.

MN-D: Deleting items to order

Description: Testing whether deleting an item to the order of a table from the menu works properly. This includes viewing the "View Orders" screen.

Items to be covered by this test: MN

Requirements addressed by this test:

Environmental Needs: Testing could be done using JUnit or Maven

Intercase Dependencies: MN-A because items must be added before you can delete them.

Test Procedures:

```
@Test
public void testViewOrders()
{
T1.items.add("Cake");
T1.items.add("Lasagna");
T1.items.add("Pizza");
T1.items.add("Lasagna");
//After adding items, they can be deleted typing in the name of the
//table and typing item to be deleted
assertEquals("Lasagna cancelled by Table T1",
l1.lastElement(), "Not correct output);
assertEquals("[Cake, Pizza, Lasagna]", T1.items, "Should print
only 1 lasagna in list");
```

Input Specifications: After adding items to a table's list, the user can then delete items from a list if it is in the list. For input, they must enter the table

number as T1-T6 and then type the menu item to be deleted and click delete.

Output Specifications: Output in l1 list should say that the item was cancelled, for example "Lasagna cancelled by Table T1" and the item should be removed from the tables list T1.items.

Pass/Fail Criteria: Passing means that the screen displays that an order was cancelled AND that it was removed from the list. Both must be done in order for this test to pass. This is because it must inform/ensure the waiter that the order was removed.

MN - VO: Orders displayed in View Orders Screen

Description: This is to test the View Orders Screen of the Menu. View Orders displays all the orders from all the tables

Items to be covered by this test: MN

Requirements addressed by this test:

Environmental Needs: Testing could be done using JUnit or Maven

Intercase Dependencies: MN-A and MN-D

Test Procedures:

```
@Test
public void testViewOrders()
{
T1.items.add("Cake");
T1.items.add("Lasagna");
T1.items.add("Pizza");
T2.items1.add("Salmon");
T2.items1.add("Steak");
T2.items1.add("Milkshake");
```

//Once the items are added to the items list of a table, click View Orders to
// view all items for every table

assertEquals("[Table 1 Orders, Cake, Lasagna, Pizza]", System.out.print(lister) , "Should print out a complete list of items in table 1);

assertEquals("[Table 2 Orders, Salmon,Steak,Milkshake]", System.out.print(lister1) , "Should print out a complete list of items in table 1);

//Removing an item from the list should be displayed in View orders

T1.items.remove("Lasagna");

assertEquals("[Table 1 Orders, Cake, Pizza]",, System.out.print(lister) , "Should print out a complete list of items in table 1 after deleting Lasagna");

Input Specifications: N/A -> Input is based on adding and Deleting items in Menu from MN-A and MN-D

Output Specifications: Output in this test depends on the items added from the menu. But basically JList listy1-listy6 will print the items in the items arraylist that each table has. This is done by storing items in a DefaultListModel lister-lister5.

Pass/Fail Criteria: Pass or fail is determined on whether the correct list is printed in the display per table.

TD - L: Displays table layout

Description: Be able to login and view the table display.

Items covered by this test: TD

Requirements addressed by this test: SPLA - 2

Environmental needs: N/A

Intercase Dependencies: N/A

Test Procedures:

1. Log in to application

2. Observe and ensure that all the tables 1-6 are shown.

Input Specification: Enter Username as “admin” and password as “password” and press the login button.

Output Specifications: The main screen containing the tables is displayed.

Pass/Fail Criteria: The test will pass if the main table display is shown: 6 tables.

TD - TA: Display if table is occupied/free

Description: Be able to login and view the table display.

Items covered by this test: TD

Requirements addressed by this test: N/A

Environmental needs: N/A

Intercase Dependencies: TD - L

Test Procedures:

1. Click on buttons “Table 1” up to “Table 6”.
2. Observe if buttons turn red.
3. Click on all buttons from step 1 again.
4. Observe if all buttons turn white.

Input Specification: Click on each table button. Then click all of them again.

Output Specifications: Buttons should turn red, then white on clicking again.

Pass/Fail Criteria: The test will pass if all the buttons turn red after clicking the first time, then white after clicking the second time.

TD - M: Displays Menu

Description: Be able to load the menu.

Items covered by this test: TD, MN

Requirements addressed by this test: N/A

Environmental needs: N/A

Intercase Dependencies: TD - L

Test Procedures:

1. Click on the menu.
2. Observe if the menu is displayed.

Input Specification: Click on menu on left side of screen.

Output Specifications: Menu should pop up to the user.

Pass/Fail Criteria: The test will pass if the menu is displayed correctly.

TD - TL: Displays Task List

Description: Be able to correctly view tasks.

Items covered by this test: TD, TL

Requirements addressed by this test: DATA - 3

Environmental needs: N/A

Intercase Dependencies: TD - L

Test Procedures:

1. Click on task list
2. Observe if the task list is displayed.

Input Specification: Click on task list button on left side of display

Output Specifications: The task list should pop up.

Pass/Fail Criteria: The test will pass if the task list is shown.

TD - SS: Displays Stat Screen

Description: Be able to view the stat screen from selecting it.

Items covered by this test: TD, SD

Requirements addressed by this test: FUNC - 4

Environmental needs: N/A

Intercase Dependencies: TD - L

Test Procedures:

1. Click on stat screen button on left side
2. Observe that all stats are displayed to user (Date/Time, Tables Occupied/Open, Customers Waiting, Wait time, Tasks Open, Tasks Completed, Revenue, Fun fact)

Input Specification: Click on the Stat Screen button on the left side of the display.

Output Specifications: The stat screen and all specific states mentioned in procedures is displayed.

Pass/Fail Criteria: The test passes if the stat screen and all corresponding stats are displayed.

TD - OR: Display Optimal route for priority queue

Description: Be able to display optimal route after customers order.

Items covered by this test: TD, MN

Requirements addressed by this test: SPLA - 1, PRAC - 1, PRAC - 2

Environmental needs: N/A

Intercase Dependencies: TD - L, TD - TA, MN - A, MN - D

Test Procedures:

1. Click on tables 1 and 2 in display and make sure they are red
2. Click on the menu on the left side.
3. Input “1” for the table number and order any item.
4. Close the menu
5. Open the menu again
6. Input “2” for the table number and order any item
7. Close the menu
8. On the right side, click on “Update”
9. Observe route and ensure table 1 and 2 are listed.

Table 1 - Optimal Route Input/Output

Input	Output
Click on table 1	Table turns red
Click on table 2	Table turns red
Click on menu	Menu pops up.
Press “1”	Text box displays table 1.
Click on food item	Food item with price displays on right.
Press the top corner “x”	Closes menu.
Click on menu	Menu pops up.
Press “2”	Text box displays table 2.
Click on food item	Food item with price displays on right.
Press the top corner “x”	Closes menu.
Click “Update”	The two tables should display on the right.

Pass/Fail Criteria: The test will pass if both tables show in the route, and the table with the least expensive order is on the top.

TL - N : Adding new tasks to task list

Description: Testing whether adding a new task to the tasks list screen works properly.

Items covered by this test: The test ensures that the new button works properly and outputs the correct task on the task list.

Requirements addressed by this test: N/A

Environmental needs: The testing could be done utilizing JUnit, or using the automation tool Maven.

Intercase Dependencies: N/A

Test Procedures: @Test

public void testTL-N() {

//Once a table number and task has been entered and then the new task button is clicked

//For example, if you want to assign a task to table 3 to check if customers need refill on drink

assertEquals("Table 3: Check if customers need refill on their drinks" TaskList.getItems().add(temp.singleTask.getTasksCreator());, "Not correct output);

assertEquals("3", singleTask.getTaskID(), "Should be 3");

assertEquals("Check if customers need refill on their drinks", singleTask.getTaskCreator(), Output should be: "Check if customers need refill on their drinks");

}

Input Specification: As for input, the table number and task must be entered by the user. Then the New Task button must be clicked.

Tables Input: "1", "2", "3", "4", "5", "6"

Task input: Anything user wants

Output Specifications:

- For the first test case, output should be "Table 3: Check if customers need refill on their drinks". This test basically tested that what the user was entering in the textbox was being correctly outputted in the gui.
- For the second test case, 3 should be the output. This is different from the first test because it's checking the getTaskID() function to ensure the correct table number is being outputted.
- For the last test case, "Check if customers need refill on their drinks" should be the output. This is the string user chose to enter and ensures that data is being properly inputted and is outputted on the tasks list.

Pass/Fail Criteria: Passing in this means that the correct table number and task is being displayed on the task list. This means that the task should be added to the gui and displayed properly.

TL - D : Deleting a task from the tasks list

Description: Testing whether adding deleting a task from the tasks lists works properly.

Items covered by this test: TL

Requirements addressed by this test: N/A

Environmental needs: The testing could be done utilizing JUnit, or using the automation tool Maven.

Intercase Dependencies: TL-N because tasks must be added before you can delete them.

Test Procedures: @Test

public void testTL-D() {

//Enter tasks. In this example, there were two new tasks added to the tasks list

assertEquals("Table 3: Check if customers need refill on their drinks", TaskList.getItems().add(temp.singleTask.getTasksCreator());, "Not correct output);

assertEquals("Table 4: Check if customers want to order dessert" ,TaskList.getItems().add(temp.singleTask.getTasksCreator());, "Not correct output);

//now the user selects the task for table 4 and deletes it.

assertEquals("Table 3: Check if customer need refills on their drinks", controller.deleteTasks(tabledisplay.newUser, temp.SingleTask);

}

Input Specification: After adding tasks to the task's list, the user can then delete any item from the list as long as it's in the task list. For input, the user must

enter a table number from 1-6 and then type any tasks they want to add. They must click the specific item that they want to delete and then click the delete button.

Output Specifications: The item should be removed from the Tasks list and be correctly updated as so in the gui.

Pass/Fail Criteria: Passing means that the tasks list no longer displays that task and it's removed from the queue. Both must be done in order to pass this test so that it doesn't reappear in the queue for some reason.

TL - U : Update button working properly

Description: When the update button on the task list appears, any timer tasks should appear on the task list.

Items covered by this test: TL - T

Requirements addressed by this test: N/A

Environmental needs: N/A

Intercase Dependencies: TD-TA because the table must be in use for the timer to start and then update in the task list.

Test Procedures:

1. Click on Table 1 and table 2 on display and make sure they are red
2. Click tasklist button on left side
3. Insert tasks for table 1 and table 2
4. Wait 2 minutes
5. Click update button
6. Ensure that a notification popped on task list page saying to go check on table 1 and table 2

Table 2 - Testing Tasks List Update Button

Input	Output
Click on table 1	Table turns red
Click on table 2	Table turns red
Click on Tasks list	Tasks list gui pops up
Add a task for table 1	The task you entered should appear
Ask a task for table 2	The tasks you entered should appear

Wait 2 minutes	N/A
Click Update Button	Check Table 2! Time is up

Pass/Fail Criteria: This test will pass if the notification appears on the task list after clicking the update button for the tables that are occupied.

3 Test Results

SD - REV: Display current revenue

Date(s) of Execution: 4/22/20 the test was completed.

Staff conducting tests: Zaid Patel had tested the Stats screen

Expected Results: Output should be a 40

Actual Results: The output was 40

Test Status: The stats screen tests had passed for displaying the correct current revenue

SD - SW: Proper time being outputted from the stopwatch

Date(s) of Execution: 4/22/20 the test was completed.

Staff conducting tests: Zaid Patel had tested the Stats screen

Expected Results: Output should be a positive integer greater than 0.

Actual Results: Output was a positive integer greater than 0.

Test Status: The stats screen tests had passed for making sure the stopwatch is accurate

SD - OT: Display number of open tasks

Date(s) of Execution: 4/22/20 the test was completed.

Staff conducting tests: Zaid Patel had tested the Stats screen

Expected Results: Output should be the integer 3 because that reflects the queue size.

Actual Results: The returned integer was 3 and it outputted correctly,

Test Status: The stats screen tests had passed for displaying the accurate number of open tasks

SD - TA: Testing Table Availability

Date(s) of Execution: 4/22/20 the test was completed.

Staff conducting tests: Zaid Patel had tested the Stats screen

Expected Results: Output should be the integer 6 because no table is currently in use

Actual Results: The returned integer was 6 and it was outputted correctly

Test Status: The stats screen tests had passed for displaying the accurate number of available tables.

MN - A: Adding to the menu order

Date(s) of Execution: 4/25/20 the test was completed.

Staff conducting tests: Bryan Gutierrez had tested the Menu Screen

Expected Results: Output should be “Lasagna Ordered by Table "T1" Cost is $13”, “Lasagna”, and 13(the price)

Actual Results: The correct output was received

Test Status: The Menu passed the test for adding items to an order.

MN - D: Deleting items from the menu order

Date(s) of Execution: 4/25/20 the test was completed.

Staff conducting tests: Bryan Gutierrez had tested the Menu Screen

Expected Results: Output should be “Lasagna cancelled by Table T1" Cost is $13”, [Cake, Pizza, Lasagna]

Actual Results: The correct output was received

Test Status: The Menu passed the test for deleting items from an order.

MN - VO: Orders displayed in View Orders Screen

Date(s) of Execution: 4/25/20 the test was completed.

Staff conducting tests: Bryan Gutierrez had tested the Menu Screen

Expected Results: Output should be [Table 1 Orders, Cake, Lasagna, Pizza], [Table 2 Orders, Salmon,Steak,Milkshake], [Table 1 Orders, Cake, Pizza]

Actual Results: The correct output was received

Test Status: The Menu passed the test for deleting items from an order.

TD - L: Displays table layout

Date(s) of Execution: 4/25

Staff conducting tests: Ammar Idrees

Expected Results: Table layout displays 6 tables.

Actual Results: Table layout displays 6 tables.

Test Status: Pass

TD - TA: Display if table is occupied/free

Date(s) of Execution: 4/25

Staff conducting tests: Ammar Idrees

Expected Results: Tables should turn red then white again.

Actual Results: Tables turned red then white again.

Test Status: Pass

TD - M: Displays Menu

Date(s) of Execution: 4/25

Staff conducting tests: Ammar Idrees

Expected Results: Menu should display.

Actual Results: Menu displays.

Test Status: Pass

TD - TL: Displays Task List

Date(s) of Execution: 4/25

Staff conducting tests: Ammar Idrees

Expected Results: Task list should appear.

Actual Results: Task list appears.

Test Status: Pass

TD - SS: Displays Stat Screen

Date(s) of Execution: 4/25

Staff conducting tests: Ammar Idrees

Expected Results: Stat screen should appear with all stats (Date/Time, Tables Occupied/Open, Customers Waiting, Wait time, Tasks Open, Tasks Completed, Revenue, Fun fact)

Actual Results: Stat Screen displays with all stats.

Test Status: Pass

TD - OR: Displays optimal route

Date(s) of Execution: 4/25

Staff conducting tests: Ammar Idrees

Expected Results: The tables should display and the table with the least expensive order is listed first.

Actual Results: The tables are displayed and the table with the least expensive order is listed first.

Test Status: Pass

TL - N: Adding to the tasks list

Date(s) of Execution: 4/24/20 the test was completed.

Staff conducting tests: Maaz Ahmed had tested the Tasks List

Expected Results: Output should be "Table 3: Check if customers need refill on their drinks" , check getTaskID is 3 and check getTaskCreator is "Check if customers need refill on their drinks"

Actual Results: The correct output was received

Test Status: The Tasks list passed the test for adding a new task

TL - D: Deleting tasks from the task list

Date(s) of Execution: 4/24/20 the test was completed.

Staff conducting tests: Maaz Ahmed has tested the task list

Expected Results: Output should be "Table 4:Check if customers want to order a dessert", Table 3's task should be gone from the list and only table 4 should be displayed.

Actual Results: The correct output was received

Test Status: The Task List passed the test for deleting an item in the list

TL - U: Update button working properly

Date(s) of Execution: 4/24/20 the test was completed.

Staff conducting tests: Maaz Ahmed has tested the task list

Expected Results: The tasks list should have a notification appear for tables 1 and 2 telling them to go check on the tables as time is up

Actual Results: The correct output was received

Test Status: The Update button is working properly

4 Regression Testing

No tests were needed to repeat.

III Inspection

1 Items to be Inspected

statsDisplay Class, tableLayout Class, menuDisplay class, tasksList Class

2 Inspection Procedures

Each person selected the code that they were the main contributor to, and explained their intentions of what they did to the other three group members. Discord has a convenient feature where you can screen share so each member was able to display the screen and show what the code they wanted to present. After that, the other three members inspected the code to see if it adhered to the requirements. Afterwards,

everyone who inspected the code would present what they found and any feedback that should be given. This was accomplished through the discord application as it has a feature where you can share your screen while voice chatting. This allowed each member of the group to screenshare the specific segment of code they wanted inspected.

For inspections, no specific checklist was used, but all code was subject to general things such as being bug-free, readable, professional, organized, and if it was consistent throughout the project.

3 Inspection Results

Inspection 1: April 22rd at 12 pm

- Inspectors: Zaid Patel, Bryan Gutierrez, Maaz Ahmed
- Code Inspected: statsDisplay Class, by Ammar Idrees
- Procedures: Overall there were a few things that were inspected within the statsScreen class, and all of its corresponding classes. First off, we checked if there were any time constraints, and if there were any open threads from the timer class. Another key aspect was if the application were to be modified for a larger use case, for the current code suffice.
- Results: While analyzing the current stats screen we had deduced that the overall revenue was correctly generated when customers added and deleted items from their menu. Furthermore, the class utilized a stop watch and there were no fixed delays within that aspect. Also, the coinciding stat for available tables properly updated at real time when we closed a table after being served.

Inspection 2: April 23rd at 4 pm

- Inspectors: Bryan Gutierrez,Ammar Idrees,Maaz Ahmed
- Code Inspected: menuDisplay Class, by Zaid Patel
- Procedures: For the menu, the main thing that we inspected was whether or not the items were being added to a table after a menu item was selected. The menuDisplay also included the View orders screen so we wanted to make sure that the changes made in the menu(adding and removing items) were reflected in the View Orders Screen. We know that if the restaurant would like to expand the menu, it should be possible, so adding a button should be simple and easy.
- Results: After reviewing the Menu screen, we knew that it was all working properly in terms of adding/removing any orders from any table. Orders for different tables could be added at the same time and this was all reflected in the view orders screen correctly. One thing that we noticed in the code was the fact that there were if statements for each order item. But overall, the Menu was correctly implemented and worked as intended.

Inspection 3: April 24th at 3 pm

- Inspectors: Maaz Ahmed, Zaid Patel, Ammar Idrees
- Code Inspected: tasksList Class by Bryan Gutierrez
- Procedures: For the tasks page, it was important to check that tasks list was being added or deleted by the user's choosing. We want the user to be able to specify the table number and be able to enter any tasks they wanted to that table. Furthemore, it was important to check that the update button was working so the user can be reminded to go check on tables after a designated time period.
- Results: After reviewing the code, it was confirmed that it was working as attended. The code was easy to follow and you could understand the flow of the program. The variables were relevant and allowed us to determine what variable was used for what. The code had some minor bugs such as not being able to be reopened once it's closed but other than that it looked good. The update button worked well and gave updates based on the timer we set. Multiple classes were used effectively so not all the code was in one file. One critique would be to include more comments that help clarify confusing things within the code. Overall, the code was organized and efficient in getting the job it needed to do.

Inspection 4: April 25th at 2pm

- Inspectors: Ammar Idrees, Zaid Patel. Bryan Gutierrez
- Code Inspected: tableDisplay Class by Maaz Ahmed
- Procedures: For the display, the main thing that needed to be inspected was if it acted as the main hub of the program. We needed to ensure that the table display had accurate commands to navigate through the other features of the application such as the menu, tasks, and stats. It should also show clear code on how it is able to generate the optimal route algorithm as well.
- Results: The code works as intended and is able to effectively navigate the application. The code is clear on how to do a task when the user performs a specific action. Furthermore, the code does not seem to contain any major bugs, is organized, and is easy to read.

IV Recommendations and Conclusions

All of our testing procedures passed, as we intended for them to. Yet, going through this process did make us aware of new bugs found during the inspection process. As for the menu, after inspecting the code we realized that we could have made some parts of it more efficient. Instead of making a bunch of if statements for every item ordered, we could have gone through a loop to check every item. This would not only clean up the code a bit, but it would also facilitate adding more items to the menu

when needed. The Stats screen worked correctly. The included statistics were useful for viewing things such as wait time, tables available/taken. In the future, a better/more efficient implementation for wait time could be made but the implementation that was created worked properly. The task list worked as expected with adding and deleting tasks to the list when they were completed. There are definitely ideas that can be added to the task list to make it more interesting. An idea would be adding a notification system for the timer task to notify the waiter immediately with some sort of ping on a phone when a table should be checked. The table display was meant to show the availability of each table, to see the optimal routes list, and to navigate around the rest of the application. The table availability worked very well with changing the color of the tables' button when it was taken and it would only add a table to the optimal routes list when it was in use. The only recommendation for the table display is possibly fixing the graphical user interface to make it more appealing to the eye. Overall, the application did its job and it was a successful prototype for Waitless.

V Project Issues

1 Open Issues

One issue we found during testing was that the tasks lists cannot be opened again after it has been closed. Even though we weren't able to find the reasoning behind why this is happening, we implemented a workaround. The workaround is that we incorporated multithreading with the task list. This means that it can always be running even when using other features at the menu. This bug will need to be fixed in a future release.

Another issue we found was the menu was that with the current setup there's a restriction on how many items can be added before the screen gets full. The limit is around 28 items right now. This can easily be fixed in a future release by making the GUI bigger and incorporating wrap text or by having a reset button after a table places their order.

Lastly, we found this rare bug that sometimes with the optimal route queue the table might appear twice or not delete properly. The issue was this bug was rarely happening. More extensive test cases need to be done to determine what's causing it and then figuring out to fix it in a future release.

2 Waiting Room

Table 3: Waiting Room Table

Requirement	Priority	Motivation	Intended Version Number
Allow account	Very high	Allows each waiter	Version 2

creation and store information in database for each user		to have a personalized screen based on the tables they are assigned for their shift.	
Allow managers to have their own screen where they can view any waiters page and see stats for restaurant overall	Very high	Allow the manager to see how the night is going and see each waiter's performance. This could be helpful for evaluating job performance.	Version 3.
Add additional features to the menu page	High	We want to make the menu as convenient as possible for the waiter and add new features to speed up the process.	Version 4
Allow customers to ping waiters on the application	Medium	We feel this would be a helpful feature to add. There are situations in which customers might need something to fully enjoy their experience and a ping system would work best with the waitless product.	Version 5
Add additional stats that the manager feels is useful	Low	This would be beneficial for the manager so they can keep track of everything in one place and see continual progress.	Version 6

3 Ideas for Solutions

A major part of the application that we could implement in the future is the use of databases. These would primarily be used in instances such account creation, orders, employees, and other things like that. We could use tools such as Teradata or Microsoft SQL Server to make this possible.

Another possibility for future iterations is to use a client system where a customer has access to the application as well. Doing so would allow a customer to send their order to a waiter rather than a waiter manually putting in the order themselves. Additionally we could add a notification system that would allow staff members to notify other employees of tasks. This would be possible using Java, so no additional tools would be necessary.

4 Project Retrospective

The methods we used that work well was our weekly meetings and assigning a task with each person. This was effective because every person was on the same page and development was most efficient that way. In addition to the meetings, the "eating a bicycle" approach was effective because it allowed us to build one piece of the program at a time, rather than build the entire program at once. This resulted in little less stress and intimidation when facing certain problems. For example, rather than thinking "How do we make sure every table is able to receive orders", we thought "How do we get one table to order" and then repeated that process for each table.

A process that didn't quite work was how to implement the functionality of the program with the user interface. In the development of the project, some of the group members were making the GUI but were also inputting features within the GUI at the same time. Meanwhile, other group members were focused on making sure their code works in a terminal setting, and then building an interface on top of it. This approach led to a lot of "spaghetti code" where inspecting the code of some group members became confusing and difficult to find bugs. Furthermore, it was rather inefficient and resulted in us creating an unfinished UI with an incomplete application.

For future development, it would be best if we focused on a feature, implementing just the code without a gui to ensure it is functional in its simplest form, and then build an interface so we have something that looks good, and works as intended.

VI Glossary

A distributed denial-of-service (DDoS): a malicious attempt to disrupt normal traffic of a targeted server, service or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic.

Consultant: a person who provides expert advice professionally.

Table: a piece of furniture with a flat top and one or more legs, providing a level surface on which objects may be placed, and that can be used for such purposes as eating.

Waiter/Waitress: a man/woman whose job is to serve customers at their tables in a restaurant.

VII References / Bibliography

[1] Robertson and Robertson, Mastering the Requirements Process.

[2] A. Silberschatz, P. B. Galvin and G. Gagne, Operating System Concepts, Ninth ed., Wiley, 2013.

[3] J. Bell, "Underwater Archaeological Survey Report Template: A Sample Document for Generating Consistent Professional Reports," Underwater Archaeological Society of Chicago, Chicago, 2012.

[4] M. Fowler, UML Distilled, Third Edition, Boston: Pearson Education, 2004.

[5] Wang, http://people.cs.ksu.edu/~jia/Inspection_Check_List.html

[6] Jeremy, Damian, Fabian, Marcos, *Wait-Less: A Program to Help Restaurants*, 2019