Informatics Institute of Technology
Department of Computing
Software Development II Coursework Report

Module	: 4COSC010C.3
Module Name	: Software Development II (2023)
Module Leader	: Mr. Deshan Sumanathilake
Date of submission	: 2023/07/16
IIT Student ID	: 20221889
UoW Student ID	: w1986580

Name	: Samaththuwa Wasan Ronath Tharana Wijayin
IIT Student ID	: 20221889
UoW Student ID	: w1986580

Test Cases

Task 1

	Test Case	Expected Result	Actual Result	Pass/Fail
1	Food Queue Initialized Correctly After program starts, 101 or VEQ	Displays ‘empty’ for all queues.	Displays ‘empty’ for all Queues.	Pass
2	Add customer to Queue 1 102 or ACQ Enter Name: Ronath Enter Queue: 1	Display “customer Ronath added to queue 1.”	Display “customer Ronath added to queue 1.”	Pass
3	Add customer to Queue 1 102 or ACQ Enter Name: Kamal Enter Queue: 1	Display “customer Kamal added to queue 1.”	Display “customer Kamal added to queue 1.”	Pass
4	Add customer to Queue 3 102 or ACQ Enter Name: Sunal Enter Queue: 3	Display “customer Sunal added to queue 3.”	Display “customer Sunal added to queue 3.”	Pass
5	Add customer to Queue 2 102 or ACQ Enter Name: Nimal Enter Queue: 2	Display “customer Nimal added to queue 2.”	Display “customer Nimal added to queue 2.”	Pass
6	Add customer to Queue 1 102 or ACQ Enter Name: Senuth Enter Queue: 1	Display “Queue 1 is full. Customer can not added.”	Display “Queue 1 is full. Customer can not added.”	Pass
7	View all Queues 100 or VFQ.	Display “***************** Cashiers ***************** O O O O X X X X X X O - Occupied X - Not Occupied”	Display “***************** Cashiers ***************** O O O O X X X X X X O - Occupied X - Not Occupied”	Pass

8	View all Empty Queues 101 or VEQ	Display "****************** Cashiers ***************** O O O O X X X X X X O - Occupied X - Not Occupied ================= ---- Queues ---- ================= Queue 1: Full Queue 2: Empty Queue 3: Empty"	Display "****************** Cashiers ***************** O O O O X X X X X X O - Occupied X - Not Occupied ================= ---- Queues ---- ================= Queue 1: Full Queue 2: Empty Queue 3: Empty"	Pass
9	View Customers Sorted in alphabetical order 105 or VCS.	Display " Kamal Nimal Ronath Sunal"	Display " Kamal Nimal Ronath Sunal"	Pass
10	Remove customer from Queue 3 specific location 103 or RCQ. Enter Queue: 3 Enter Position: 1	Display "Customer Sunal removed from queue 3."	Display "Customer Sunal removed from queue 3."	Pass
11	Remove customer from Queue 3 specific location 103 or RCQ. Enter Queue: 3 Enter Position: 2	Display "No customer found at position 2 in queue 3."	Display "No customer found at position 2 in queue 3."	Pass
12	Remove customer from Queue 2 specific location 103 or RCQ. Enter Queue: 2 Enter Position: 5	Display "Invalid customer position. Please try again."	Display "Invalid customer position. Please try again."	Pass
13	Remove served customer from Queue 1 104 or PCQ. Enter Queue: 1	Display "Customer Ronath removed from queue 1."	Display "Customer Ronath removed from queue 1."	Pass
14	Remove served customer from Queue 3 104 or PCQ. Enter Queue: 3	Display "No served customer found in queue 3."	Display "No served customer found in queue 3."	Pass

15	Store Program Data into file 106 or SPD	Display “All the details have been saved into the file successfully.”	Display “All the details have been saved into the file successfully.”	Pass
16	Load Program Data from file 107 or LPD.	Display “Last saved at: 2023/07/15 13:29:29 Customer name: Kamal Queue no: 1 Position no: 1 Customer name: Nimal Queue no: 2 Position no: 1 All the details have load successfully.”	Display “Last saved at: 2023/07/15 13:29:29 Customer name: Kamal Queue no: 1 Position no: 1 Customer name: Nimal Queue no: 2 Position no: 1 All the details have load successfully.”	Pass
17	View Remaining burgers in the Stock 108 or STK.	Display “Remaining burgers in the stock: 45.”	Display “Remaining burgers in the stock: 45.”	Pass
18	Remove served customer from Queue 2 104 or PCQ. Burger stock equals to 10 or Burger stock less than 10	Display “Warning!!! Low burgers in stock.”	Display “Warning!!! Low burgers in stock.”	Pass
19	Remove served customer from Queue 1 104 or PCQ. Burger stock equals to zero.	Display “No burgers in the stock!”	Display “No burgers in the stock!”	Pass
20	Add burgers to Stock 109 or AFS. Number of burgers: 5	Display “Burgers added to the stock. Burgers in the stock: 50”	Display “Burgers added to the stock. Burgers in the stock: 50”	Pass
21	Add burgers to Stock 109 or AFS. Number of burgers: 8	Display “There can be maximum number of 50 burgers in the stock. Number of burgers in the stock: 50 Maximum number of burgers that can be add to the stock: 0”	Display “There can be maximum number of 50 burgers in the stock. Number of burgers in the stock: 50 Maximum number of burgers that can be add to the stock: 0”	Pass
22	View all Empty Queues 101 or VEQ	Display “***************** Cashiers ***************** O O X X X X X X	Display “***************** Cashiers ***************** O O X X X X X X	Pass

		X X O - Occupied X - Not Occupied ================ ---- Queues ---- ================ Queue 1: Empty Queue 2: Empty Queue 3: Empty”	X X O - Occupied X - Not Occupied ================ ---- Queues ---- ================ Queue 1: Empty Queue 2: Empty Queue 3: Empty”	
23	Exit the Program 999 or EXT.	Display “Exiting the program...”	Display “Exiting the program...”	Pass

Test Case

Task 2 and Task 3

	Test Case	Expected Result	Actual Result	Pass/Fail
1	Food Queue Initialized Correctly After program starts, 101 or VEQ	Displays ‘empty’ for all queues.	Displays ‘empty’ for all Queues.	Pass
2	Add customer to Queue 102 or ACQ Enter First name: Ronath Enter Second name: Tharan Required burgers: 5	Display “Customer Ronath Tharana add to the queue.”	Display “Customer Ronath Tharana add to the queue.”	Pass
3	Add customer to Queue 102 or ACQ Enter First name: Kamal Enter Second name: Perera Required burgers: 3	Display “Customer Kamal Perera add to the queue.”	Display “Customer Kamal Perera add to the queue.”	Pass
4	Add customer to Queue 102 or ACQ Enter First name: Sunil Enter Second name: Kumara Required burgers: 2	Display “Customer Sunil Kumara add to the queue.”	Display “Customer Sunil Kumara add to the queue.”	Pass
5	Add customer to Queue 102 or ACQ Enter First name: Nimal Enter Second name: Tharaka Required burgers: 7	Display “Customer Nimal Tharaka add to the queue.”	Display “Customer Nimal Tharaka add to the queue.”	Pass

6	View all Queues 100 or VFQ.	Display “***************** Cashiers ***************** O O O O X X X X X X O - Occupied X - Not Occupied”	Display “***************** Cashiers ***************** O O O O X X X X X X O - Occupied X - Not Occupied”	Pass
7	View all Empty Queues 101 or VEQ	Display “================ ---- Queues ---- ================ queue 1 : Full queue 2 : Empty queue 3 : Empty”	Display “================ ---- Queues ---- ================ queue 1 : Full queue 2 : Empty queue 3 : Empty”	Pass
9	View Customers Sorted in alphabetical order 105 or VCS.	Display “Kamal Perera Nimal Tharaka Ronath Tharana Sunil Kumara”	Display “Kamal Perera Nimal Tharaka Ronath Tharana Sunil Kumara”	Pass
10	Remove customer from Queue 3 specific location 103 or RCQ. Enter Queue: 3 Enter Position: 1	Display “Customer Sunil Kumara removed from queue 3.”	Display “Customer Sunil Kumara removed from queue 3.”	Pass
11	Remove customer from Queue 3 specific location 103 or RCQ. Enter Queue: 3 Enter Position: 2	Display “No customer found in that queue position.”	Display “No customer found in that queue position.”	Pass
12	Remove customer from Queue 2 specific location 103 or RCQ. Enter Queue: 2 Enter Position: 5	Display “Invalid customer position. Please try again.”	Display “Invalid customer position. Please try again.”	Pass
13	Remove served customer from Queue 1 104 or PCQ. Enter Queue: 1	Display “Customer Ronath Tharana remove from the queue 1.”	Display “Customer Ronath Tharana remove from the queue 1.”	Pass
14	Remove served customer from Queue 3 104 or PCQ. Enter Queue: 3	Display “Selected queue is empty. Please try again.”	Display “Selected queue is empty. Please try again.”	Pass

15	Store Program Data into file 106 or SPD	Display "All the details have been saved into the file successfully."	Display "All the details have been saved into the file successfully."	Pass
16	Load Program Data from file 107 or LPD.	Display " Last saved at: 2023/07/15 14:16:40 Queue 1 customer details Customer name: Nimal Tharaka Number of burgers required: 7 Customer position in the queue: 1 Queue 2 customer details Customer name: Kamal Perera Number of burgers required: 3 Customer position in the queue: 1 All the details have load successfully."	Display " Last saved at: 2023/07/15 14:16:40 Queue 1 customer details Customer name: Nimal Tharaka Number of burgers required: 7 Customer position in the queue: 1 Queue 2 customer details Customer name: Kamal Perera Number of burgers required: 3 Customer position in the queue: 1 All the details have load successfully."	Pass
17	View Remaining burgers in the Stock 108 or STK.	Display "Remaining burgers in the stock: 45"	Display "Remaining burgers in the stock: 45"	Pass
18	Remove served customer from Queue 2 104 or PCQ. Burger stock equals to 10 or Burger stock is less than 10.	Display "Warning!!! Low burgers in stock."	Display "Warning!!! Low burgers in stock."	Pass
19	Remove served customer from Queue 1 104 or PCQ. Burger stock is less than required burgers.	Display "Burger stock do not contain that much burgers."	Display "Burger stock do not contain that much burgers."	Pass
	Add burgers to Stock 109 or AFS. Number of burgers: 5	Display "Burgers added to the stock. Burgers in the stock: 50"	Display "Burgers added to the stock. Burgers in the stock: 50"	Pass
20	Add burgers to Stock 109 or AFS. Number of burgers: 8	Display "There can be maximum number of 50 burgers in the stock.	Display "There can be maximum number of 50 burgers in the stock.	Pass

		Number of burgers in the stock: 50 Maximum number of burgers that can be add to the stock: 0”	Number of burgers in the stock: 50 Maximum number of burgers that can be add to the stock: 0”	
21	View all Empty Queues 101 or VEQ	================ ---- Queues ---- ================ queue 1 : Empty queue 2 : Empty queue 3 : Empty”	================ ---- Queues ---- ================ queue 1 : Empty queue 2 : Empty queue 3 : Empty”	Pass
22	Income of each queue 110 or IFQ	Display “=================== ---- Income of Queues ---- =================== Price of a burger RS: 650/= Queue 1 Served burgers in queue: 5 Income of queue: Rs: 3250 Queue 2 Served burgers in queue: 0 Income of queue: Rs: 0 Queue 3 Served burgers in queue: 0 Income of queue: Rs: 0 Total income in all 3 queues: Rs: 3250”	Display “=================== ---- Income of Queues ---- =================== Price of a burger RS: 650/= Queue 1 Served burgers in queue: 5 Income of queue: Rs: 3250 Queue 2 Served burgers in queue: 0 Income of queue: Rs: 0 Queue 3 Served burgers in queue: 0 Income of queue: Rs: 0 Total income in all 3 queues: Rs: 3250”	Pass
23	Remove served customer from Queue 1 104 or PCQ. Burger stock equals to zero.	Display “No burgers in the stock!”	Display “No burgers in the stock!”	Pass
24	Add customer to Queue 102 or ACQ all the queues are full. Enter First name: Senuth	Display “All the queues are full. Customer Senuth Nethwin add to the queue.”	Display “All the queues are full. Customer Senuth Nethwin add to the queue.”	Pass

	Enter Second name: Nethwin Required burgers: 6			
25	Remove customer from Queue 1 specific location 103 or RCQ. Enter Queue: 1 Enter Position: 1 (all the queues are full)	Display “Customer Nimal Tharaka removed from queue 1.”	Display “Customer Nimal Tharaka removed from queue 1.”	Pass
26	Remove customer from Queue 1 specific location 103 or RCQ. Enter Queue: 1 Enter Position: 2	Display “Customer Senuth Nethwin removed from queue 1.”	Display “Customer Senuth Nethwin removed from queue 1.”	Pass
27	Exit the Program 999 or EXT.	Display “Exiting the program...”	Display “Exiting the program...”	Pass

Discussion

First, I run the program and check that the View all Empty Queues option is working properly and that all queues are empty. Then I added four customers to the selected queues. Then I try to add customers to a full queue to check if the queue size is maintained properly. Then I use the View All Queues option to check that the customers are added to the correct positions. After that, I checked the View all Empty Queues option again to ensure it worked properly. Then, before removing the customers from any queue, I sorted the customer names in alphabetical order to check that the ordering was working properly. Then I remove some customers from the selected queues to check that the customers I added to the queue are in the correct position and that the Remove a Customer from a Queue option is working properly. Then I try to use the Remove a customer from a Queue option in an empty queue to ensure that no errors occur. After that, I try to remove a customer from a position that is never in the queue to check for errors. Then I use the Remove a served customer option to check that the correct customer is removed from the queue. After that, I try to remove a customer from an empty queue to ensure that no errors occur. Then I use the Store Program Data into File option to store customer data in a file. Then I use the Load Program Data from File option to store data that I saved previously. After that, I use the View Remaining Burger Stock Option to check for the correct number of burgers removed from the burger stock. Then I remove some customers to check that the warning displays when the burgers in stock reach 10, and it continues until the stock becomes zero. After that, I used the Add Burgers to Stock option to check that the burger addition would happen correctly, and that the burger stock capacity remained at the correct count. Then I use the View all Empty Queues option again to check that the removed customers were correctly removed. After that, I use the income of each queue option to check whether the income of each queue was calculated properly. Then I use the waiting queue after all the queues are full to check if the newly added customer will be added to the waiting queue. After that, I use the Remove a customer from a Queue option to remove a customer from the full queue. After that, I use the Remove a Customer from a Queue option again to remove the last customer of that full queue to check to ensure that the waiting queue customer was added to the empty queue properly, and then I use the Exit the Program option to check that the program ended correctly.

Code :

Task 01

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;

public class FoodCenter {

    // Creating a max burger count in the burger stock.
    private static final int maxBurgerStock = 50;

    // Creating a stock that can add or remove burgers.
    private static int burgerStock = maxBurgerStock;

    // Creating a array to store queues data.
    // https://www.w3schools.com/java/java_arrays_multi.asp
    private static String[][] queues = new String[3][5];

    // Initializing a scanner to get inputs.
    private static Scanner getInput = new Scanner(System.in);


    public static void main(String[] args) {
        String line = "=====";
        String blank = "     ";
        boolean options;
        System.out.println("\n" + line.repeat(20));
        System.out.println(blank.repeat(6) + "Welcome to Foodies Fave Food Center");

        // Creating the menu options and printing the menu options.
        options = true;

        while (options) {
            System.out.println("\n" + line.repeat(20));
            System.out.println("Please select an option: ");
            System.out.println("100 or VFQ: View all Queues.");
            System.out.println("101 or VEQ: View all Empty Queues.");
            System.out.println("102 or ACQ: Add customer to a Queue.");
            System.out.println("103 or RCQ: Remove a customer from a Queue. (From a specific
location)");
            System.out.println("104 or PCQ: Remove a served customer.");
            System.out.println("105 or VCS: View Customers Sorted in alphabetical order.");
            System.out.println("106 or SPD: Store Program Data into file.");
            System.out.println("107 or LPD: Load Program Data from file.");
            System.out.println("108 or STK: View Remaining burgers Stock.");
            System.out.println("109 or AFS: Add burgers to Stock.");
            System.out.println("999 or EXT: Exit the Program.");
            System.out.println(line.repeat(20));

            // Getting the menu option input.
            String choice = readString("Enter your option: ");

            // Using switch case to give the output according to the input.
            https://www.w3schools.com/java/java_switch.asp

            switch (choice) {
```

```
case "100":
case "VFQ":
   // This option shows the occupied and not occupied queue positions.
   viewAllQueues();
   break;

case "101":
case "VEQ":
   // This option shows that queue is full or empty.
   viewAllEmptyQueues();
   break;

case "102":
case "ACQ":
   // This option add a customer to the queue.
   addCustomerToQueue();
   break;

case "103":
case "RCQ":
   // This option will remove a customer from a selected position.
   removeCustomerFromAQueue();
   break;

case "104":
case "PCQ":
   // This option will remove a served customer from a queue.
   removeServedCustomer();
   break;
```

```
case "105":
case "VCS":
    // This option will show the customer names in the alphabetical order.
    viewCustomersSorted();
    break;

case "106":
case "SPD":
    // This option will save customer details in to a file.
    storeProgramData();
    break;

case "107":
case "LPD":
    // This option will read customer details from the save file.
    loadProgramData();
    break;

case "108":
case "STK":
    // This option will show the remaining burgers in the stock.
    remainingBurgersStock();
    break;

case "109":
case "AFS":
    // This option will add burgers to the stock.
    addBurgersToStock();
    break;
```

```
            case "999":
            case "EXT":
                // This option will end the program and exit.
                options = false;
                System.out.println("Exiting the program...");
                break;

            default:
                System.out.println("Invalid option");
                break;
        }
    }
}

/**
 * Method to print the queues occupied and not occupied positions.
 */
private static void viewAllQueues() {
    System.out.println("******************");
    System.out.println("     Cashiers      ");
    System.out.println("******************");

    // Creating a copy of the queues array.
    // https://www.programiz.com/java-programming/copy-arrays

    String[][] queuesCopy = new String[3][5];
    for (int i = 0; i < queues.length; i++) {
        System.arraycopy(queues[i], 0, queuesCopy[i], 0, queues[i].length);
    }
```

```
        // Using ternary operator to maintain each queue length.
        // https://www.w3schools.com/java/java_conditions_shorthand.asp

        for (int i = 0; i < queuesCopy.length; i++) {

            int maxCapacity = (i == 2) ? 5 : i + 2;
            for (int j = 0; j < queuesCopy[i].length; j++) {
                if (j < maxCapacity) {
                    if (queuesCopy[i][j] == null) {
                        queuesCopy[i][j] = "X";
                    } else {
                        queuesCopy[i][j] = "O";
                    }
                } else {
                    queuesCopy[i][j] = " ";
                }
            }
        }

        for (int i = 0; i < queuesCopy.length - 2; i++) {
            for (int j = 0; j < queuesCopy[i].length; j++) {
                System.out.println("    " + queuesCopy[i][j] + "    " + queuesCopy[i + 1][j] + "    " +
queuesCopy[i + 2][j]);
            }
        }
        System.out.println("O - Occupied \nX - Not Occupied");
    }

    /**
     * Method to show the queues condition.
     */
    private static void viewAllEmptyQueues() {
        viewAllQueues(); // printing the queues empty and occupied positions.

        System.out.println("================");
        System.out.println("---- Queues ----");
        System.out.println("================");


        for (int i = 0; i < queues.length; i++) {

            int occupiedPositions = 0;
            int maxCapacity = (i == 2) ? 5 : i + 2; // Using ternary operator to maintain each
queue length.
            for (int j = 0; j < maxCapacity; j++) {
                if (queues[i][j] != null) {
                    occupiedPositions++;
                }
            }

            String status = "";
            if (occupiedPositions == maxCapacity) {
                status = "Full";
            } else {
                status = "Empty";
            }
            System.out.println("Queue " + (i + 1) + ": " + status);
        }
    }

    /**
     * Method to add a new customer to the selected queue.
     */
    private static void addCustomerToQueue() {
        String customerName = readString("Enter customer name: ");
        int queueNumber = selectQueue();
        String capitalizeCustomerName = customerName.substring(0, 1).toUpperCase() +
customerName.substring(1); // Capitalize the first letter in customer name.

        int queueIndex = queueNumber - 1;
        int positions = availablePosition(queueIndex);

        // Using ternary operator to maintain each queue length.
        int maxPositions = (queueIndex == 0) ? 2 : (queueIndex == 1 ? 3 : 5);

        if (positions >= 0 && positions < maxPositions) {
            queues[queueIndex][positions] = capitalizeCustomerName;
            System.out.println("customer " + capitalizeCustomerName + " added to queue " +
queueNumber + " .");
        }
        else {
            System.out.println("Queue " + queueNumber + " is full. Customer can not added.");
        }
    }

    /**
     * Method to remove a customer from a selected position in a queue.
     */
    private static void removeCustomerFromAQueue() {
        int queueNumber = selectQueue();
        int queueIndex = queueNumber - 1;

        int position = readInteger("Enter customer position in the queue: ");
        int positionIndex = position - 1;

        // Using ternary operator to maintain each queue length.
        int positionLimit = (queueIndex == 0) ? 2 : (queueIndex == 1 ? 3 : 5);

        if (position >= 1 && position <= positionLimit) {

            if (queues[queueIndex][positionIndex] != null) {
                String customerName = queues[queueIndex][positionIndex];
                queues[queueIndex][positionIndex] = null;
                System.out.println("Customer " + customerName + " removed from queue " +
queueNumber + ".");

                // Move customers forward to fill the empty position in the queue.
                for (int i = positionIndex + 1; i < positionLimit; i++) {
                    if (queues[queueIndex][i] != null) {
                        queues[queueIndex][i - 1] = queues[queueIndex][i];
                        queues[queueIndex][i] = null;
                    }
                }
            } else {
                System.out.println("No customer found at position " + position + " in queue " +
queueNumber + ".");
            }
        } else {
            System.out.println("Invalid customer position. Please try again.");
        }
    }

    /**
    * Method to remove a served customer from the queue.
    */
   private static void removeServedCustomer() {
      int queueNumber = selectQueue();
      int queueIndex = queueNumber - 1;

      int positionIndex = -1;
      for (int i = 0; i < queues[queueIndex].length; i++) {
         if (queues[queueIndex][i] != null) {
            positionIndex = i;
           break;
         }
      }

      // Using ternary operator to maintain each queue length.
      int positionLimit = (queueIndex == 0) ? 2 : (queueIndex == 1 ? 3 : 5);

      if (positionIndex == 0) {
         String customerName = queues[queueIndex][positionIndex];
         queues[queueIndex][positionIndex] = null;

         // Move customers forward to fill the empty position in the queue.
         for (int j = positionIndex + 1; j < positionLimit; j++) {
            if (queues[queueIndex][j] != null) {
               queues[queueIndex][j - 1] = queues[queueIndex][j];
               queues[queueIndex][j] = null;
            }
         }

         System.out.println("Customer " + customerName + " removed from queue " +
queueNumber + ".");
```

```
            // Remove burgers from burger stock and show a warning if the burger count is low.
            burgerStock -= 5;
            if (burgerStock <= 10 && burgerStock > 0) {
                System.out.println("\nWarning!!! Low burgers in stock \nRemaining burgers: " +
burgerStock);
            }
            if (burgerStock <= 0) { // When the burger count reach zero ask to add burgers to the
stock.
                System.out.println("\nNo burgers in the stock!");
                question();
            }
        } else {
            System.out.println("No served customer found in queue " + queueNumber + ".");
        }
    }

    /**
     * Method to sort the customer names in alphabetical order and print the names by the
order.
     */
    private static void viewCustomersSorted() {
        System.out.println("===================================");
        System.out.println("----- Sorted Customer Names -----");
        System.out.println("===================================");

        String[] customers = allCustomers();

        if (customers.length == 0) {
            System.out.println("No customers found in queues.");
        }
        else {
            // https://stackoverflow.com/questions/18689672/how-to-sort-a-string-array-
alphabetically-without-using-compareto-or-arrays-sor

            for (int i = 0; i < customers.length; i++) {
                for (int j = 0; j < customers.length - 1 - i; j++) {
                    if (compareCustomerNames(customers[j], customers[j + 1]) > 0) {
                        String tempName = customers[j];
                        customers[j] = customers[j + 1];
                        customers[j + 1] = tempName;
                    }
                }
            }
        }

        for (String customerName : customers) {
            System.out.println(customerName);
        }
    }

    /**
     * Method to save customer data in to a file.
     */
    // https://www.w3schools.com/java/java_files_create.asp
    private static void storeProgramData() {
        try {
            // Creating date and time format and get the live date and time.
            // https://www.javatpoint.com/java-get-current-
date#:~:text=Get%20Current%20Date%20%26%20Time%3A%20java,the%20current%20da
te%20and%20time.
            DateTimeFormatter liveDateTime = DateTimeFormatter.ofPattern("yyyy/MM/dd
HH:mm:ss");
```

```
            LocalDateTime live = LocalDateTime.now();

            File detailFile = new File("FoodCenter.txt");

            if (detailFile.createNewFile()) {
                System.out.println("File created : " + detailFile.getName());
                System.out.println("Path : " + detailFile.getAbsolutePath());
            } else {
                System.out.println("File exist : " + detailFile.getName());
                System.out.println("Path : " + detailFile.getAbsolutePath());
            }

            FileWriter writeDetails = new FileWriter("FoodCenter.txt");

            writeDetails.write("Last saved at: " + liveDateTime.format(live) + "\n\n");

            for (int i = 0; i < queues.length; i++) {
                for (int j = 0; j < queues[i].length; j++) {
                    String customerName = queues[i][j];
                    if (customerName != null) {
                        writeDetails.write("Customer name: " + customerName + "\nQueue no: " + (i
+ 1) + "\nPosition no: " + (j + 1) + "\n\n");
                    }
                }
            }
            writeDetails.close();
            System.out.println("\nAll the details have been saved into the file successfully.");

        } catch (IOException error) {
            System.out.println("An error occurred. \nPlease try again.");
            error.printStackTrace();
```

```
    }
}

/**
 * Method to read customer data from the save file.
 */
// https://www.w3schools.com/java/java_files_read.asp
private static void loadProgramData() {
    try {
        File detailFile = new File("FoodCenter.txt");
        Scanner readDetails = new Scanner(detailFile);
        System.out.println();
        while (readDetails.hasNextLine()) {
            String details = readDetails.nextLine();
            System.out.println(details);
        }
        readDetails.close();
        System.out.println("\nAll the details have load successfully.");
    }
    catch (FileNotFoundException error) {
        System.out.println("An error occurred. \nPlease try again.");
       error.printStackTrace();
    }
}

/**
 * Method to print the remaining burgers in the stock.
 */
private static void remainingBurgersStock() {
    System.out.println("Remaining burgers in the stock: " + burgerStock);
    }


    /**
     * Method to add burgers to the stock.
     */
    private static void addBurgersToStock() {
        int addBurgers = readInteger("Enter the number of burgers that need to add: ");


        if (addBurgers >= 0 && addBurgers <= 50) {
            if ((addBurgers + burgerStock) >= 0 && (addBurgers + burgerStock) <=
maxBurgerStock) {  // Check that when add burgers to the stock the max burger count pass
the limit.
                burgerStock += addBurgers;
                System.out.println("Burgers added to the stock. \nBurgers in the stock: " +
burgerStock);
            } else {
                System.out.println("\nThere can be maximum number of 50 burgers in the stock.
\nNumber of burgers in the stock: " + burgerStock);
                System.out.println("Maximum number of burgers that can be add to the stock: " +
(maxBurgerStock - burgerStock));
            }
        } else {
            System.out.println("Incorrect number of burgers! Please try again.");
        }
    }


    /**
     * Printing the queue options and get the input.
     * @return the selected queue number
     */
    private static int selectQueue() {
        System.out.println("\nSelect Queue:");
        System.out.println("1: Queue 01");
        System.out.println("2: Queue 02");
        System.out.println("3: Queue 03");
        int number = readInteger("Enter queue number: ");
        if (number <= queues.length && number >= 1) {
            return number;
        } else {
            System.out.println("Invalid queue number. Please Try again");
            return selectQueue();
        }
    }

    /**
     * Check the queues positions are occupied or not.
     * @param queueIndex the selected queue array index
     * @return array position index
     */
    private static int availablePosition(int queueIndex) {
        for (int i = 0; i < queues[queueIndex].length; i++) {
            if (queues[queueIndex][i] == null) {
                return i;
            }
        }
        return -1;
    }

    /**
     * Create an array and add all the customers names in to the array.
     * @return the array that contain all the customers names
     */
    private static String[] allCustomers() {
        int totalCustomers = 0;

        for (int i = 0; i < queues.length; i++) {
            for (int j = 0; j < queues[i].length; j++) {
                if (queues[i][j] != null) {
                    totalCustomers++;
                }
            }
        }

        String[] customers = new String[totalCustomers]; // Creating an array and select the
array length.
        int customerIndex = 0;

        for (int i = 0; i < queues.length; i++) {
            for (int j = 0; j < queues[i].length; j++) {
                if (queues[i][j] != null) {
                    customers[customerIndex] = queues[i][j];
                    customerIndex++;
                }
            }
        }

        return customers;
    }

    /**
     * Asking a question when burger stock reach zero.
     */
    private static void question() {
        int number = readInteger("Would you like to add burgers to the stock \n1 : Yes \n2 : No
\nEnter your option: ");

        if (number == 1) {
            System.out.println("Preparing to add burgers to stock...\n");
            addBurgersToStock();
        } else if (number == 2) {
            System.out.println("Returning to the menu...");
        } else {
            System.out.println("Invalid number. Please Try again");
        }
    }

    /**
     * Get two customers names and check for the largest size name.
     * @param customer1 the name of the first selected customer.
     * @param customer2 the name of the second selected customer.
     * @return the difference between selected two names letters or names length.
     */
    // https://stackoverflow.com/questions/18689672/how-to-sort-a-string-array-alphabetically-
without-using-compareto-or-arrays-sor
    private static int compareCustomerNames(String customer1, String customer2) {
        int smallLengthName = Math.min(customer1.length(), customer2.length()); // Selecting
the lowest length name length.

        for (int i = 0; i < smallLengthName; i++) {
            char letter1 = customer1.charAt(i);
            char letter2 = customer2.charAt(i);

            if (letter1 != letter2) {
                return letter1 - letter2;
            }
        }
        return customer1.length() - customer2.length();
    }

    /**
     * Reading a string using scanner.
     * @param massage the statement that ask for an input.
     * @return the selected string type input.
     */
    private static String readString(String massage) {
        System.out.println(massage);
        try {
            String input = getInput.next();
            getInput.nextLine();
            return input;

        }
        catch (Exception error) {
            getInput.nextLine();
            System.out.println("Invalid response. Please try again.");
            return readString(massage);
        }
    }

    /**
     * Reading an integer using scanner.
     * @param massage the statement that ask for an input.
     * @return the selected integer type input.
     */
```

```
    private static int readInteger(String massage) {
        System.out.println(massage);
        try {
            int input = getInput.nextInt();
            getInput.nextLine();
            return input;


        }
        catch (Exception error) {
            getInput.nextLine();
            System.out.println("Invalid response. Please try again.");
            return readInteger(massage);
        }
    }
}
```

Task 02 and Task 03

```
package FoodCenter;


import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Scanner;

public class Main {



  // Creating a max burger count in the burger stock.
  private static final int maxBurgerStock = 50;

  // Creating a stock that can add or remove burgers.
  private static int burgerStock = maxBurgerStock;

  // Initializing a scanner to get inputs.
  private static Scanner getInput = new Scanner(System.in);

  // Creating an arraylist to store food queue class objects.
  // https://www.w3schools.com/java/java_arraylist.asp
  private static ArrayList<FoodQueue> queues = new ArrayList<>();

  // Calculate all the served burger count in queue 1.
  private static int queue1ServedBurgers = 0;
```

```
// Calculate all the served burger count in queue 2.
private static int queue2ServedBurgers = 0;

// Calculate all the served burger count in queue 3.
private static int queue3ServedBurgers = 0;



public static void main(String[] args) {

    // Creating four objects in food queue class.
    FoodQueue foodQueue1 = new FoodQueue(2);
    FoodQueue foodQueue2 = new FoodQueue(3);
    FoodQueue foodQueue3 = new FoodQueue(5);

    FoodQueue foodWaitingQueue = new FoodQueue(50);

    // Add created four objects to the arraylist.
    queues.add(foodQueue1);
    queues.add(foodQueue2);
    queues.add(foodQueue3);
    queues.add(foodWaitingQueue);

    String line = "=====";
    String blank = "     ";
    boolean options;
    System.out.println("\n" + line.repeat(20));
    System.out.println(blank.repeat(6) + "Welcome to Foodies Fave Food Center");

    // Creating the menu options and printing the menu options.
```

```
options = true;

while (options) {
    System.out.println("\n" + line.repeat(20));
    System.out.println("Please select an option: ");
    System.out.println("100 or VFQ: View all Queues.");
    System.out.println("101 or VEQ: View all Empty Queues.");
    System.out.println("102 or ACQ: Add customer to a Queue.");
    System.out.println("103 or RCQ: Remove a customer from a Queue. (From a specific
location)");
    System.out.println("104 or PCQ: Remove a served customer.");
    System.out.println("105 or VCS: View Customers Sorted in alphabetical order.");
    System.out.println("106 or SPD: Store Program Data into file.");
    System.out.println("107 or LPD: Load Program Data from file.");
    System.out.println("108 or STK: View Remaining burgers Stock.");
    System.out.println("109 or AFS: Add burgers to Stock.");
    System.out.println("110 or IFQ: Income of each queue.");
    System.out.println("999 or EXT: Exit the Program.");
    System.out.println(line.repeat(20));

    // Getting the menu option input.
    String choice = readString("Enter your option: ");

    // Using switch case to give the output according to the input.
    https://www.w3schools.com/java/java_switch.asp

    switch (choice) {
        case "100":
        case "VFQ":
            // This option shows the occupied and not occupied queue positions.
            viewAllQueues();
    break;

case "101":
case "VEQ":
    // This option shows that queue is full or empty.
    viewAllEmptyQueues();
    break;

case "102":
case "ACQ":
    // This option add a customer to the queue.
    addCustomerToQueue();
    break;

case "103":
case "RCQ":
    // This option will remove a customer from a selected position.
    removeCustomerFromAQueue();
    break;

case "104":
case "PCQ":
    // This option will remove a served customer from a queue.
    removeServedCustomer();
    break;

case "105":
case "VCS":
    // This option will show the customer names in the alphabetical order.
    viewCustomersSorted();
    break;

case "106":
case "SPD":
    // This option will save customer details in to a file.
    storeProgramData();
    break;

case "107":
case "LPD":
    // This option will read customer details from the save file.
    loadProgramData();
    break;

case "108":
case "STK":
    // This option will show the remaining burgers in the stock.
    remainingBurgersStock();
    break;

case "109":
case "AFS":
    // This option will add burgers to the stock.
    addBurgersToStock();
    break;

case "110":
case "IFQ":
    // This option will show the income of each queue.
    incomeOfQueues();
```

```
                break;

            case "999":
            case "EXT":
                // This option will end the program and exit.
                options = false;
                System.out.println("Exiting the program...");
                break;

            default:
                System.out.println("Invalid option");
                break;
        }
    }
}

/**
 * Method to print the queues occupied and not occupied positions.
 */
private static void viewAllQueues() {

    System.out.println("*****************");
    System.out.println("     Cashiers     ");
    System.out.println("*****************");

    String[][] allQueues = addCustomerNames();

    // Using ternary operator to maintain each queue length.
    // https://www.w3schools.com/java/java_conditions_shorthand.asp
    for (int i = 0; i < allQueues.length; i++) {
            int maxPositions = (i == 2) ? 5 : i + 2;
            for (int j = 0; j < allQueues[i].length; j++) {
                if (j < maxPositions) {
                    if (allQueues[i][j] == null) {
                        allQueues[i][j] = "X";
                    }
                    else {
                        allQueues[i][j] = "O";
                    }
                }
                else {
                    allQueues[i][j] = " ";
                }
            }
        }
        for (int i = 0; i < allQueues.length - 2; i++) {
            for (int j = 0; j < allQueues[i].length; j++) {
                System.out.println("    " + allQueues[i][j] + "    " + allQueues[i + 1][j] + "    " +
allQueues[i + 2][j]);
            }
        }
        System.out.println("O - Occupied \nX - Not Occupied");
    }


    /**
     * Method to show the queues condition.
     */
    private static void viewAllEmptyQueues() {


        System.out.println("================");
        System.out.println("---- Queues ----");
    System.out.println("================");

    ArrayList<Customer> queue1 = queues.get(0).getCustomers();
    ArrayList<Customer> queue2 = queues.get(1).getCustomers();
    ArrayList<Customer> queue3 = queues.get(2).getCustomers();



    if (queue1.size() == 2) {
        System.out.println(" queue 1 : Full");
    }
    else {
        System.out.println(" queue 1 : Empty");
    }
    if (queue2.size() == 3) {
        System.out.println(" queue 2 : Full");
    }
    else {
        System.out.println(" queue 2 : Empty");
    }
    if (queue3.size() == 5) {
        System.out.println(" queue 3 : Full");
    }
    else {
        System.out.println(" queue 3 : Empty");
    }
}

/**
 * Method to add a new customer to the queue with the minimum length.
 */
```

```
    private static void addCustomerToQueue() {

        String firstName = readString("Enter customer first name: ");
        String capitalizeFirstName = firstName.substring(0, 1).toUpperCase() +
firstName.substring(1); // Capitalize the first letter in customer first name.

        String secondName = readString("Enter customer second name: ");
        String capitalizeSecondName = secondName.substring(0, 1).toUpperCase() +
secondName.substring(1); // Capitalize the first letter in customer second name.

        int burgerNumber = readInteger("Enter the number of  burgers required: ");

        // Add customer details to the customer class and create the customer class object.
        Customer customer = new Customer(capitalizeFirstName, capitalizeSecondName,
burgerNumber);


        ArrayList<Customer> queue1 = queues.get(0).getCustomers();
        ArrayList<Customer> queue2 = queues.get(1).getCustomers();
        ArrayList<Customer> queue3 = queues.get(2).getCustomers();
        ArrayList<Customer> waitingQueue = queues.get(3).getCustomers();

        // Check all the queues are full or not and if full add customers to the waiting queue.
        if (queue1.size() == 2 && queue2.size() == 3 && queue3.size() == 5) {
            System.out.println("All the queues are full.\nCustomer " + customer.getFirstName() +
" " + customer.getSecondName() + " add to the waiting queue.");
            waitingQueue.add(customer);
        }
        else {
            if (queue1.size() < 2 && queue1.size() <= queue2.size() && queue1.size() <=
queue3.size()) {  // select the minimum length queue.
                queue1.add(customer);
        } else if (queue2.size() < 3 && queue2.size() <= queue1.size() && queue2.size() <=
queue3.size()) {
            queue2.add(customer);
        } else {
            queue3.add(customer);
        }

        System.out.println("Customer " + customer.getFirstName() + " " +
customer.getSecondName() + " add to the queue.");
    }
  }



  /**
   * Method to remove a customer from a selected position in a queue.
   */
  private static void removeCustomerFromAQueue() {

      ArrayList<Customer> queue1 = queues.get(0).getCustomers();
      ArrayList<Customer> queue2 = queues.get(1).getCustomers();
      ArrayList<Customer> queue3 = queues.get(2).getCustomers();
      ArrayList<Customer> waitingQueue = queues.get(3).getCustomers();

      int queueNumber = selectQueue();
      int positionNumber = readInteger("Enter customer position: ");
      int positionIndex = positionNumber - 1;

      // Using ternary operator to maintain each queue length
      int positionLimit = (queueNumber == 1) ? 2 : (queueNumber == 2 ? 3 : 5);

      if (positionIndex < positionLimit) {
        if (queueNumber == 1 && !queue1.isEmpty() && positionNumber <= queue1.size())
{
            System.out.println("Customer " + queue1.get(positionIndex).getFirstName() + " " +
queue1.get(positionIndex).getSecondName() + " removed from queue " + queueNumber +
".");
            queue1.remove(positionIndex);
            if (queue1.size() < 2 && !waitingQueue.isEmpty()) { // Check for customers in the
waiting queue and add customers in to the empty queue.
                queue1.add(waitingQueue.remove(0));
            }

        } else if (queueNumber == 2 && !queue2.isEmpty() && positionNumber <=
queue2.size()) {
            System.out.println("Customer " + queue2.get(positionIndex).getFirstName() + " " +
queue2.get(positionIndex).getSecondName() + " removed from queue " + queueNumber +
".");
            queue2.remove(positionIndex);
            if (queue2.size() < 3 && !waitingQueue.isEmpty()) { // Check for customers in the
waiting queue and add customers in to the empty queue.
                queue2.add(waitingQueue.remove(0));
            }

        } else if (queueNumber == 3 && !queue3.isEmpty() && positionNumber <=
queue3.size()){
            System.out.println("Customer " + queue3.get(positionIndex).getFirstName() + " " +
queue3.get(positionIndex).getSecondName() + " removed from queue " + queueNumber +
".");
            queue3.remove(positionIndex);
            if (queue3.size() < 5 && !waitingQueue.isEmpty()) { // Check for customers in the
waiting queue and add customers in to the empty queue.
                queue3.add(waitingQueue.remove(0));
            }
        }
        else {
            System.out.println("No customer found in that queue position.");
          }
        }
        else {
          System.out.println("Invalid customer position. Please try again.");
        }
      }

      /**
       * Method to remove a served customer from the queue.
       */
      private static void removeServedCustomer() {

        ArrayList<Customer> queue1 = queues.get(0).getCustomers();
        ArrayList<Customer> queue2 = queues.get(1).getCustomers();
        ArrayList<Customer> queue3 = queues.get(2).getCustomers();
        ArrayList<Customer> waitingQueue = queues.get(3).getCustomers();

        int queueNumber = selectQueue();
        int positionIndex = 0;

        if (queueNumber == 1 && !queue1.isEmpty()) {
          int number = burgerLimit(queueNumber, positionIndex);
          if (number == 1) {
            queue1ServedBurgers += queue1.get(positionIndex).getBurgersNumber();
            System.out.println("Customer " + queue1.get(positionIndex).getFirstName() + " " +
queue1.get(positionIndex).getSecondName() + " remove from the queue " + queueNumber +
".");
            queue1.remove(positionIndex);
            if (queue1.size() < 2 && !waitingQueue.isEmpty()) { // Check for customers in the
waiting queue and add customers in to the empty queue.
```

```
                queue1.add(waitingQueue.remove(0));
            }
        }

    } else if (queueNumber == 2 && !queue2.isEmpty()) {
        int number = burgerLimit(queueNumber, positionIndex);
        if (number == 1) {
            queue2ServedBurgers += queue2.get(positionIndex).getBurgersNumber();
            System.out.println("Customer " + queue2.get(positionIndex).getFirstName() + " " +
queue2.get(positionIndex).getSecondName() + " remove from the queue " + queueNumber +
".");
            queue2.remove(positionIndex);
            if (queue2.size() < 3 && !waitingQueue.isEmpty()) { // Check for customers in the
waiting queue and add customers in to the empty queue.
                queue2.add(waitingQueue.remove(0));
            }
        }

    } else if (queueNumber == 3 && !queue3.isEmpty()) {
        int number = burgerLimit(queueNumber, positionIndex);
        if (number == 1) {
            queue3ServedBurgers += queue3.get(positionIndex).getBurgersNumber();
            System.out.println("Customer " + queue3.get(positionIndex).getFirstName() + " " +
queue3.get(positionIndex).getSecondName() + " remove from the queue " + queueNumber +
".");
            queue3.remove(positionIndex);
            if (queue3.size() < 5 && !waitingQueue.isEmpty()) { // Check for customers in the
waiting queue and add customers in to the empty queue.
                queue3.add(waitingQueue.remove(0));
            }
        }
    }
```

```
        else {
            System.out.println("Selected queue is empty. Please try again.");
        }

        // Show a warning if the burger count is low.
        if (burgerStock <= 10 && burgerStock > 0) {
            System.out.println("\nWarning!!! Low burgers in stock \nRemaining burgers: " +
burgerStock);
        }
        if (burgerStock <= 0) { // When the burger count reach 0 ask to add burgers to the stock.
            System.out.println("\nNo burgers in the stock!");
            question();
        }
    }

    /**
     * Method to sort the customer names in alphabetical order and print the names by the
order.
     */
    private static void viewCustomersSorted() {

        System.out.println("=============================");
        System.out.println("----- Sorted Customers -----");
        System.out.println("=============================");

        ArrayList<String> customers = allCustomers();

        if (customers.isEmpty()) {
            System.out.println("No customers found in queues.");
        }
        else {
            // https://stackoverflow.com/questions/18689672/how-to-sort-a-string-array-
alphabetically-without-using-compareto-or-arrays-sor
            // https://www.programiz.com/java-programming/library/arraylist/set
            for (int i = 0; i < customers.size() - 1; i++) {
                for (int j = 0; j < customers.size() - 1 - i; j++) {
                    if (compareCustomerNames(customers.get(j), customers.get(j + 1)) > 0) {
                        String tempName = customers.get(j);
                        customers.set(j, customers.get(j + 1));
                        customers.set((j + 1), tempName);
                    }
                }
            }
        }
        for (String customerName : customers) {
            System.out.println(customerName);
        }
    }

    /**
     * Method to save customer data in to a file
     */
    // https://www.w3schools.com/java/java_files_create.asp
    private static void storeProgramData() {
        try {
            // Creating date and time format and get the live date and time.
            // https://www.javatpoint.com/java-get-current-
date#:~:text=Get%20Current%20Date%20%26%20Time%3A%20java,the%20current%20da
te%20and%20time.
            DateTimeFormatter liveDateTime = DateTimeFormatter.ofPattern("yyyy/MM/dd
HH:mm:ss");
            LocalDateTime live = LocalDateTime.now();
            File detailFile = new File("FoodCenter.txt");

            if (detailFile.createNewFile()) {
                System.out.println("File created : " + detailFile.getName());
                System.out.println("Path : " + detailFile.getAbsolutePath());
            } else {
                System.out.println("File exist : " + detailFile.getName());
                System.out.println("Path : " + detailFile.getAbsolutePath());
            }

            FileWriter writeDetails = new FileWriter("FoodCenter.txt");

            ArrayList<Customer> queue1 = queues.get(0).getCustomers();
            ArrayList<Customer> queue2 = queues.get(1).getCustomers();
            ArrayList<Customer> queue3 = queues.get(2).getCustomers();
            ArrayList<Customer> waitingQueue = queues.get(3).getCustomers();

            writeDetails.write("Last saved at: " + liveDateTime.format(live));

            if (!queue1.isEmpty()) {
                writeDetails.write("\n\nQueue 1 customer details");
                for (int i = 0; i < queue1.size(); i++) {
                    writeDetails.write("\n\nCustomer name: " + queue1.get(i).getFirstName() + " " +
queue1.get(i).getSecondName());
                    writeDetails.write("\nNumber of burgers required: " +
queue1.get(i).getBurgersNumber());
                    writeDetails.write("\nCustomer position in the queue: " + (i + 1));
                }
            }

            if (!queue2.isEmpty()) {
                writeDetails.write("\n\nQueue 2 customer details");
                for (int j = 0; j < queue2.size(); j++) {
                    writeDetails.write("\n\nCustomer name: " + queue2.get(j).getFirstName() + " " +
queue2.get(j).getSecondName());
                    writeDetails.write("\nNumber of burgers required: " +
queue2.get(j).getBurgersNumber());
                    writeDetails.write("\nCustomer position in the queue: " + (j + 1));
                }
            }

            if (!queue3.isEmpty()) {
                writeDetails.write("\n\nQueue 3 customer details");
                for (int k = 0; k < queue3.size(); k++) {
                    writeDetails.write("\n\nCustomer name: " + queue3.get(k).getFirstName() + " "
+ queue3.get(k).getSecondName());
                    writeDetails.write("\nNumber of burgers required: " +
queue3.get(k).getBurgersNumber());
                    writeDetails.write("\nCustomer position in the queue: " + (k + 1));
                }
            }

            if (!waitingQueue.isEmpty()) {
                writeDetails.write("\n\nWaiting queue customer details");
                for (int l = 0; l < waitingQueue.size(); l++) {
                    writeDetails.write("\n\nCustomer name: " + waitingQueue.get(l).getFirstName()
+ " " + waitingQueue.get(l).getSecondName());
                    writeDetails.write("\nNumber of burgers required: " +
waitingQueue.get(l).getBurgersNumber());
                    writeDetails.write("\nCustomer position in the waiting queue: " + (l + 1));
                }
            }
            writeDetails.close();
            System.out.println("\nAll the details have been saved into the file successfully.");
        }
        catch (Exception error) {
            System.out.println("An error occurred. \nPlease try again.");
            error.printStackTrace();
        }
}

/**
 * Method to read customer data from the save file.
 */
// https://www.w3schools.com/java/java_files_read.asp
private static void loadProgramData() {

        try {
            File detailFile = new File("FoodCenter.txt");
            Scanner readDetails = new Scanner(detailFile);
            System.out.println();
            while (readDetails.hasNextLine()) {
                String details = readDetails.nextLine();
                System.out.println(details);
            }
            readDetails.close();
            System.out.println("\nAll the details have load successfully.");
        }
        catch (FileNotFoundException error) {
            System.out.println("An error occurred. \nPlease try again.");
            error.printStackTrace();
        }
    }

    /**
     * Method to print the remaining burgers in the stock.
     */
    private static void remainingBurgersStock() {

        System.out.println("Remaining burgers in the stock: " + burgerStock);
    }

    /**
     * Method to add burgers to the stock.
     */
    private static void addBurgersToStock() {
        int addBurgers = readInteger("Enter the number of burgers that need to add: ");

        if (addBurgers >= 0 && addBurgers <= 50) {
            if ((addBurgers + burgerStock) >= 0 && (addBurgers + burgerStock) <=
maxBurgerStock) {  // Check that when add burgers to the stock the max burger count pass
the limit.
                burgerStock += addBurgers;
                System.out.println("Burgers added to the stock. \nBurgers in the stock: " +
burgerStock);
            } else {
                System.out.println("\nThere can be maximum number of 50 burgers in the stock.
\nNumber of burgers in the stock: " + burgerStock);
                System.out.println("Maximum number of burgers that can be add to the stock: " +
(maxBurgerStock - burgerStock));
            }
        } else {
            System.out.println("Incorrect number of burgers! Please try again.");
        }
    }

    /**
     * Method to calculate income of each queue.
     */
    private static void incomeOfQueues() {

        System.out.println("==========================");
        System.out.println("---- Income of Queues ----");
        System.out.println("==========================");

        int queue1Income = queue1ServedBurgers * 650;
        int queue2Income = queue2ServedBurgers * 650;
        int queue3Income = queue3ServedBurgers * 650;
        int totalIncome = queue1Income + queue2Income + queue3Income;

        System.out.println("\nPrice of a burger RS: 650/=\n");

        System.out.println("Queue 1");
        System.out.println("Served burgers in queue: " + queue1ServedBurgers +"\nIncome of queue: \nRs: " + queue1Income);

        System.out.println("\nQueue 2");
        System.out.println("Served burgers in queue: " + queue2ServedBurgers +"\nIncome of queue: \nRs: " + queue2Income);

        System.out.println("\nQueue 3");
        System.out.println("Served burgers in queue: " + queue3ServedBurgers +"\nIncome of queue: \nRs: " + queue3Income);

        System.out.println("\nTotal income in all 3 queues: \nRs: " + totalIncome);
```

```
}

/**
 * Add all 3 queues customer names in to an array.
 * @return the array that contain all customers first names.
 */
private static String[][] addCustomerNames() {

    String[][] allQueues = new String[3][5];

    ArrayList<Customer> queue1 = queues.get(0).getCustomers();
    ArrayList<Customer> queue2 = queues.get(1).getCustomers();
    ArrayList<Customer> queue3 = queues.get(2).getCustomers();

    for (int i = 0; i < queue1.size(); i++) {
        allQueues[0][i] = queue1.get(i).getFirstName();
    }
    for (int j = 0; j < queue2.size(); j++) {
        allQueues[1][j] = queue2.get(j).getFirstName();
    }
    for (int k = 0; k < queue3.size(); k++) {
        allQueues[2][k] = queue3.get(k).getFirstName();
    }
    return allQueues;
}

/**
 * Printing the queue options and get the input.
 * @return the selected queue number.
```

```
     */
    private static int selectQueue() {
        System.out.println("\nSelect Queue:");
        System.out.println("1: Queue 01");
        System.out.println("2: Queue 02");
        System.out.println("3: Queue 03");

        int number = readInteger("Enter queue number: ");
        if (number <= 3 && number >= 1) {
            return number;
        } else {
            System.out.println("Invalid queue number. Please Try again");
            return selectQueue();
        }
    }

    /**
     * Check that the number of burgers customer required is more than the burgers in the
stock.
     * @param queueNumber the selected queue number.
     * @param positionIndex the index number of the customer position.
     * @return the number that decide the code other part execute or not.
     */
    private static int burgerLimit (int queueNumber, int positionIndex) {

        ArrayList<Customer> queue1 = queues.get(0).getCustomers();
        ArrayList<Customer> queue2 = queues.get(1).getCustomers();
        ArrayList<Customer> queue3 = queues.get(2).getCustomers();

        if (queueNumber == 1) {
            if (burgerStock < queue1.get(positionIndex).getBurgersNumber()) {
```

```
            System.out.println("Burger stock do not contain that much burgers.");
        }
        else {
            burgerStock -= queue1.get(positionIndex).getBurgersNumber();
            return 1;
        }
    } else if (queueNumber == 2) {
        if (burgerStock < queue2.get(positionIndex).getBurgersNumber()) {
            System.out.println("Burger stock do not contain that much burgers.");
        }
        else {
            burgerStock -= queue2.get(positionIndex).getBurgersNumber();
            return 1;
        }
    }
    else {
        if (burgerStock < queue3.get(positionIndex).getBurgersNumber()) {
            System.out.println("Burger stock do not contain that much burgers.");
        }
        else {
            burgerStock -= queue3.get(positionIndex).getBurgersNumber();
            return 1;
        }
    }
    return 0;
}

/**
 * Add all the customers fist name and the last name to an arraylist.
 * @return the arraylist that contain all the customers names.
```

```
     */
    private static ArrayList<String> allCustomers() {

        ArrayList<Customer> queue1 = queues.get(0).getCustomers();
        ArrayList<Customer> queue2 = queues.get(1).getCustomers();
        ArrayList<Customer> queue3 = queues.get(2).getCustomers();

        ArrayList<String> customers = new ArrayList<>();

        for (int i = 0; i < queue1.size(); i++) {
            customers.add(queue1.get(i).getFirstName() + " " + queue1.get(i).getSecondName());
        }

        for (int j = 0; j < queue2.size(); j++) {
            customers.add(queue2.get(j).getFirstName() + " " + queue2.get(j).getSecondName());
        }

        for (int k = 0; k < queue3.size(); k++) {
            customers.add(queue3.get(k).getFirstName() + " " +
queue3.get(k).getSecondName());
        }
        return customers;
    }

    /**
     * Get two customers names and check for the largest size name.
     * @param customer1 the name of the first selected customer.
     * @param customer2 the name of the second selected customer.
     * @return the difference between selected two names letters or names length.
     */
    // https://stackoverflow.com/questions/18689672/how-to-sort-a-string-array-alphabetically-
without-using-compareto-or-arrays-sor
    private static int compareCustomerNames(String customer1, String customer2) {
        int smallLengthName = Math.min(customer1.length(), customer2.length()); // Selecting
the lowest length name.

        for (int i = 0; i < smallLengthName; i++) {
            char letter1 = customer1.charAt(i);
            char letter2 = customer2.charAt(i);

            if (letter1 != letter2) {
                return letter1 - letter2;
            }
        }
        return customer1.length() - customer2.length();
    }

    /**
     * Asking a question when burger stock reach zero.
     */
    private static void question() {
        int number = readInteger("Would you like to add burgers to the stock \n1 : Yes \n2 : No
\nEnter your option: ");

        if (number == 1) {
            System.out.println("Preparing to add burgers to stock...\n");
            addBurgersToStock();
        } else if (number == 2) {
            System.out.println("Returning to the menu...");
        } else {
            System.out.println("Invalid number. Please Try again");
        }
    }

    /**
     * Reading a string using scanner.
     * @param massage the statement that ask for an input.
     * @return the selected string type input.
     */
    private static String readString(String massage) {
        System.out.println(massage);
        try {
            String input = getInput.next();
            getInput.nextLine();
            return input;


        }
        catch (Exception error) {
            getInput.nextLine();
            System.out.println("Invalid response. Please try again.");
            return readString(massage);
        }
    }

    /**
     * Reading an integer using scanner.
     * @param massage the statement that ask for an input.
     * @return the selected integer type input.
     */
    private static int readInteger(String massage) {
        System.out.println(massage);
```

```
        try {
            int input = getInput.nextInt();
            getInput.nextLine();
            return input;


        }
        catch (Exception error) {
            getInput.nextLine();
            System.out.println("Invalid response. Please try again.");
            return readInteger(massage);
        }
    }
}
package FoodCenter;

import java.util.ArrayList;

public class FoodQueue {

    private int maxQueueSize;
    private ArrayList<Customer> customers;

    public FoodQueue (int maxQueueSize) {

        this.maxQueueSize = maxQueueSize;
        this.customers = new ArrayList<>(maxQueueSize);
    }

    public int getMaxQueueSize() {
        return maxQueueSize;
    }

    // Get the selected arraylist.
    public ArrayList<Customer> getCustomers() {

        return customers;
    }
}
package FoodCenter;

public class Customer {

    private String firstName;
    private String secondName;
    private int burgersNumber;

    public Customer(String firstName, String secondName, int burgersNumber) {
        this.firstName = firstName;
        this.secondName = secondName;
        this.burgersNumber = burgersNumber;
    }

    // Get the customer first name.
    public String getFirstName() {

        return firstName;
    }

    // Get the customer second name.
    public String getSecondName() {

        return secondName;
    }

    // Get the number of burgers customer required.
    public int getBurgersNumber() {

        return burgersNumber;
    }
}
```