COMP1811 – Python Project Report

Name:	Aayush Sumit Pittie	Student ID	001328860
Partner's name:	Panagiotis Petsallari	Student ID	001294962

1. Brief statement of features you have completed

THIS SECTION SHOULD BE THE SAME FOR ALL GROUP MEMBERS

1.1 Circle the parts of the coursework you have **fully completed and are fully working**. Please be accurate.	**Features** **F1:** i ☒ ii ☒ iii ☒ iv ☒ v ☒ vi ☒ **F2:** i ☒ ii ☒ iii ☒ iv ☒ v ☒ vi ☒ **F3:** i ☒ ii ☒ iii ☒ iv ☒ v ☒
1.2 Circle the parts of the coursework you have **partly completed or are partly working.**	**Features** **F1:** i ☐ ii ☐ iii ☐ iv ☐ v ☐ vi ☐ **F2:** i ☐ ii ☐ iii ☐ iv ☐ v ☐ vi ☐ **F3:** i ☐ ii ☐ iii ☐ iv ☐ v ☐
Briefly explain your answer if you circled any parts in 1.2	

2. Concise List of Bugs and Weaknesses

*A concise list of bugs and/or weaknesses in your work (if you don't think there are any, then say so). Bugs that are declared in this list will lose you fewer marks than ones that you don't declare! (**100-200 words**, but word count depends heavily on the number of bugs and weaknesses identified.)*

THIS SECTION SHOULD BE COMPLETED INDIVIDUALLY FOR F1 AND F2 AND AS A GROUP FOR F3.

2.1 Bugs

List each bug plus a brief description. A bug is code that causes an error or produces unexpected results.

F2: No known bugs when generating a new Customer.

F3: In the generate customer method the flag for are_all_lanes_full() takes more time than expected to get updated and at the same time there is a chance that a customer is being generated which should be happening as the expected outcome was to wait for the flag to be turned off and the generate the next customers.

2.2 Weaknesses

List each weakness plus a brief description. A weakness is code that only works under limited scenarios and at some point produces erroneous or unexpected results or code/output that can be improved.

F2: There are no known weaknesses for this Class.

F3: Currently we do not have 8 Tills for our Self-checkout lane and we would like to improve upon that by adding that functionality with the idea: - Treating each till as a lane of 1 capacity and then using the Self-service Lane queue to insert/add customers to them.

We haven’t implemented logic for dynamic lane movement so when there are 2 lanes with customers that one lane it’s going to take more time than the other one then the last customer of that lane should be moved to a lane which takes less time.

The logic for processing customers and managing lane status may be a bit inaccurate as lane status is updated only every 5 seconds whereas processing of customer will be happening the whole time so sometimes there is going to be an empty lane with status still open.

3. DESCRIPTION OF THE FEATURES IMPLEMENTED

*Describe your implementation design and the choices made (e.g. choice of data structures, custom data types, code logic, choice of functions, etc) and indicate how the features developed were integrated. (**200-400 words***

THIS SECTION SHOULD BE COMPLETED INDIVIDUALLY FOR F1 AND F2 AND AS A GROUP FOR F3.

F2: For F2, I created a class that defines the object Customer, I wanted it to have a unique id for each customer. For the Lottery system when generating the customer, I assigned them the result and I designed the function so that they have a 10% chance of winning the lottery. This can easily be modified by changing the criteria. The number of items in basket is being generated by a Function that generates them with there being a 50% probability of the customer to generate with less than 10 Items and 50% chance for them to be generated with more than 10 items which contrasted with 33% before. My reasoning for this was in a real-life scenario as per the location of the store and time of the day the customers could range from buying evening lunch to weekly shopping for the family. This function allows them to change which kind of customers are generated more and for even generation it can be set to have 2 True choices and 1 False. I am also taking the lane they entered and printing it with their other details.

F3: The supermarket class is responsible for Initializing the lanes and setting up initial open lanes, one regular lane, and the self-checkout lane. We made it so it initializes 5 regular lanes and 1 self service lane and store them into a list for easy access and allowing it to be iterable. The initial customers is random from 1-10 customers and they have their own function so they can are able to enter a lane outside the thread and the while loop. This allows us to start the threads with a small gap with them to reduce the conflicts of print between the threads. To stop this we also imported lock from threading and used it to allow access of console to one thread at a time. We also input if they desire to have max number of customers to be generated or duration of the simulation.

After the initial preparation we have two threaded function that run the actual simulation, Customer_generation and lane_management which are defined in out simulation method which acts as our main method through which parameters are passed, threads are defined and started, and where our loop is broken once it reaches the criteria. Our first function that is essential to our code is lane_management which first processes the customers in each lane indicating a step/second passed and then displays the status of those lanes. We did it this way as we decided to process customer by slowly reducing their items from their basket which allowed us to have a real time usage of the lanes as we add up all the items of all the customers in the queue to define usage. This means that if a customer Is joining and there are two lanes with 1 customer each but the customer in one lane has just joined while in other, they are half way through their checkout, with this system they can join the latter as it is updating the usage data every step of the simulation.

Customer_generation as it says in its names generates the customer in random intervals that can range from 1-10s they are then giving a unique Id from an incrementing counter which also works as a way to check for the limit we took as an input. There is an if statement to check whether to generate customers or not that works on counting the total customers in all the lanes or if all the lanes of one single type are filled. The it Is passed onto enter_lane function which then finds the lane with least usage by creating a list of all their usages and the running min() command on that list for regular lanes.

It uses the lane add_custome function to add the customer to the lane. For self service lanes it just checks if the customer has less than 10 items and if there's space in the self_checkout lane and true it does the same. I also return the lane it entered to be printed by customer display. In the case when the lanes are full it shows lane saturation and waits for 5s then continues with the loop.

4. CLASSES AND OOP FEATURES

List the classes you developed and provide an exposition on the choice of classes, class design, and OOP features implemented. List all the classes used in your program and include the attributes and behaviours for each. You may use a class diagram to illustrate these classes – do not include the class code here. Your narrative for section 4.2 should describe the design decisions you made, and the OOP techniques used (abstraction, encapsulation, inheritance/polymorphism). ***Note****: stating definitions here will not get you marks, you must clearly outline how you implemented the techniques in your code and WHY. (****400-600 words****)*

THIS SECTION SHOULD BE COMPLETED INDIVIDUALLY FOR F1 AND F2 AND AS A GROUP FOR F3.

4.1 CLASSES USED

F2: Class:- Customer

Attributes: id, item_in_basket, lottery_result, self_checkout_time, cash_checkout_time

Methods: check_lottery_tkt, checkout_time, display_customer_details, random_number_of_items

F3: Class: Supermarket

Attributes: regular_lanes, self_service_lane, initial_customers, customer_counter

Methods: simulation, lane_management, generate_customers, lane_saturation, generate_initial_customers, are_all_lanes_full

4.2 BRIEF EXPLANATION OF CLASS DESIGN AND OOP FEATURES USED

F2 The customer class takes an id as a parameter. It uses abstracting to decide certain attributes of the customer. For eg. Item_in_basket is decided by a function that randomly generates a number using an algorithm which is abstracted as well as the calculation behind deciding the result for the lottery. Similarly, checkout time is abstracted. I also created a function that combines these attributes to display the customer details. This ensures that it is more efficient as it must access its own attributes rather than calling it every time if I did it outside. I also designed the code, so it takes in the lane as a parameter which using an if statement it is also able to decipher whether the customer was added to self service lane or not.

F3 The Supermarket class consists of 6 methods that form the F3 with the enter_lane method being for F1. We designed the code to run the simulation using an Try-Exception, which allows us to terminate it using the Keyboard Interrupt Error. The main framework for the simulation is in the method simulation so we can pass parameters when calling the main function that can define the simulation length or the total customers to be generated. We have also set the threads as daemon so they can terminate when the main code ends. Like customer we have abstracted lane_satruation as well as are_all_lanes_full which check Number of customers currently in lane and if all lanes of one type are full respectively. This allows us to use them at multiple locations like lane saturation is used when there's no space for a customer to join a lane and it must display the lane saturation as well as in generate_customers for the if condition that pauses the method if it reaches 40.

There is a while loop in the simulation that checks for the Duration and total customer which was designed so that once it reaches either one of those conditions it terminates the code.

The generate_initial_customers method runs a for loop which results from 0 to 9 initial customers, the for loop creates instances of the Customer class with id staring with a “C” prefix.

In are_all_lanes_full returns a Boolean value by using ‘all’ feature of python which only gives true if all conditions are true. Using this I run a short for loop which iterates for all the regular lanes on whether they are full or not.

In lane_saturation we just iterate over the loop for all lanes and add the length of their customers deque into a variable and then return the variable’s final value. This finds the total number of customers in all lanes.

In generate_customers there’s an Infinite While loop allowing us to pause the loop when one of the conditions is not met. Which are if the lane saturation reaches 40, which is the max for this simulation, or all regular lanes are full indicating there’s no space for customers to go to. If that happens it just prints the current lane saturation and sleeps for 5s

In lane_management we first process the customers in each lane, which means reducing the items_in_basket attribute then we check for empty lanes and close them after which we display lane status for all lanes. This happens again in an iterable loop which uses display_lane_status function of F1 to display this. At the end we sleep for 5s to not display the status very frequently. Though this means the customers are only processed every 5s, though this can be fixed by separating these two functions, at this stage we are just reducing 5s worth of items each loop.

5. Code for the Classes Created

*Add the **code for each of the classes you have implemented yourself** here. If you have contributed to parts of classes, please highlight those parts in a different colour and label them with your name. Copy and paste relevant code - actual code please, no screenshots! Make it easy for the tutor to read. Add an explanation if necessary – though your in-code comments should be clear enough. You will lose marks if screenshots are provided instead of code. **DO NOT provide a listing of the entire code. You will be marked down if a full code listing is provided, or you include the code as a screenshot**.*

THIS SECTION SHOULD BE COMPLETED INDIVIDUALLY FOR F1 AND F2 AND AS A GROUP FOR F3.

5.1 Class Customer

```
""" Creates Customer Object"""
    def __init__(self, cid):
        """ Initializes Customers with Customer ID."""
        self.id = cid  # Unique identifier of Customer
        self.item_in_basket = self.random_number_of_items()
        # Generating random number of items in basket
        self.lottery_result = self.check_lottery_tkt()  # Lottery result for the customer
        self.self_checkout_time = self.checkout_time(6)  # Check out time at Self-service till
        self.cash_checkout_time = self.checkout_time(4)  # Check out time at Cashier till


    def check_lottery_tkt(self):
        """Checks if a customer wins a lottery ticket"""
        if self.item_in_basket >= 10 and random.random() < 0.1:
         # The number of items should be >= 10
            result = "Winner"            # Sets a 10% Probability to win a ticket
        else:
            result = "Hard luck, no lottery ticket this time"
        return result


    def checkout_time(self, n):
        """Calculates the Checkout time upon number of items"""
        return self.item_in_basket * n
          # Multiplies by parameter n which is time taken to scan a single item
```

```
def display_customer_details(self, lane):
    """Prints the customer details in a Set Format"""
    with CheckoutLane.PRINT_LOCK:  # Allows multiple threads to access Print resource
        print("Customer details")  # and to prevent interference from other threads
        print(f"ID: {self.id}")
        print(f"Items in basket: {self.item_in_basket}")
        print(f"Lottery ticket result: {self.lottery_result}")
        if self.item_in_basket < 10 and lane:
            # Prints depending upon which lane they are likely to go
            print(f"Time to process basket at self checkout till: {self.self_checkout_time} Secs")
            print('Customer added to Self Checkout Lane 6')
        else:
            print(f"Time to process basket at cashier till: {self.cash_checkout_time} Secs")
            print(f'Customer added to {lane}')
        print('_____')
        print(f'_____ \n')

@staticmethod
def random_number_of_items():
    """Calculates the Items in basket of a Customer randomly"""
    if random.choice([True, False]):
    # This method allows more customers with less than 10 items to be generated
        return random.randint(10, 30)
    # Can be modified to have more or less of a type of customer
    else:  # Currently 50% chance to have less than 10 items which can be modified.
        return random.randint(1, 10)  # with changing the amount of true values in the list
```

5.2 CLASS SUPERMARKET

Code in Yellow :- Aayush Pittie 001328860

```
"""Main Simulation Class"""

    PRINT_LOCK = threading.Lock()


    def __init__(self):
        """Initialises Lanes and No. of initial customers"""
        self.regular_lanes = [RegularLane(i) for i in range(1, 6)]  # Creating 5 Regular Lanes
        self.self_service_lane = [SelfServiceLane(6)]  # Creating 1 Self-service Lanes
        self.regular_lanes[0].open_lane()  # Starts the simulation with two open lanes
        self.self_service_lane[0].open_lane()  # One regular and one Self service
        self.initial_customers = random.randint(1, 10)  # Initial Customers
        self.customer_counter = self.initial_customers  # Counts the total customers processed


    def are_all_lanes_full(self):
        """Checks are all Regular lanes Full"""
        return all(lane.lane_is_full() for lane in self.regular_lanes)


    def generate_initial_customers(self):
        """Generates and Displays initial customers"""
        for _ in range(1, self.initial_customers):
            customer = Customer(cid=f"C{_}")
            x = self.enter_lane(customer)  # Adding them to a lane
            customer.display_customer_details(x)  # Displays the customer details
```

```
    def enter_lane(self, customer):  # Panagiotis Petsallari 001294962 F1
        """Adds customers to lanes"""
        if customer.item_in_basket < 10 and not self.self_service_lane[0].lane_is_full():
            self.self_service_lane[0].add_customer(customer)  # Adds customer with less than 10 items only
            return True
        else:
            processing_time = [lane.lane_usage() for lane in self.regular_lanes]
            x = processing_time.index(min(processing_time))  # Finds the shortest lane for customer to join
            regular_lane_with_shortest_queue = self.regular_lanes[x]
            regular_lane_with_shortest_queue.add_customer(customer)  # Adds customer to the shortest lane
            return f'Regular Lane {x + 1}'

    def lane_saturation(self):
        """Calculates total customers Standing in line"""
        users = 0
        for lane in self.regular_lanes:  # Iterates over all lanes to add the customers up
            users = users + (len(lane.customers))
        users += len(self.self_service_lane[0].customers)
        return users

    def generate_customers(self):
        """Generates customers with unique iD"""
        while True:
            if self.lane_saturation() < 40 and not self.are_all_lanes_full():
                # Generates only when there is a space for  the customer to join in the lane
                # In other words it pauses customer generation when there is no space in the lanes
                customer = Customer(cid=f"C{self.customer_counter}")
                self.customer_counter += 1  # Increments the no. of customer generated
```

```
            x = self.enter_lane(customer)  # Enters the customer into a lane
            customer.display_customer_details(x)  # Displays the customer details
            self.enter_lane(customer)  # Enters the customer into a lane
            time.sleep(random.randint(1, 10))  # Generates a customer randomly between 1 and 10 Seconds
        else:
            print(f"Lane Saturation is {instance.lane_saturation() / 0.4} %")  # Dividing by 0.4 as 40 customers
            time.sleep(5)  # If all the lanes are full it waits for 5s and checks the Saturation again
            continue   # This allows customers to be processed


    def lane_management(self):
        """Manages all the Functions of the Lane Such as:
        Processing the Customer
        Displaying Lane Status
        Closing Empty Lane"""
        while True:
            for lane in self.regular_lanes + self.self_service_lane:
                lane.process_customer()  # This processes the customer
                lane.lane_close()  # This will close the lane if it's empty
            with Supermarket.PRINT_LOCK:
                for lanes in self.regular_lanes:
                    lanes.display_lane_status()  # Displays status of Regular lanes, lane by lane
                self.self_service_lane[0].display_lane_status()  # Displays status of Self service lane
                time.sleep(5)  # This defines the speed of the simulation


    def simulation(self, total_customers, duration):
        """Initialises the Total customers to be generated and the Duration of the Simulation
        Also calls all the Starting methods and Starts threading"""
        start_time = time.time()  # Start time is defined from real time clock
```

```
        end_time = start_time + duration
        self.generate_initial_customers()  # Calls the function to generate the initial customers
        customer_generation = threading.Thread(target=self.generate_customers)
        lane_generation = threading.Thread(target=self.lane_management)
        # Threads the two main Methods Customer Generation and lane management allowing both to execute independently
        # of each other. It also allows the ability to have different speeds at which
        # Lanes are processed or customers are generated
        customer_generation.daemon = True
        lane_generation.daemon = True
        current_time = 0  # Initialising for the While Loop
        lane_generation.start()  # Starts the first thread
        time.sleep(0.5)  # Time gap so the threads don't interfere as they have different phases
        customer_generation.start()  # Second thread starts
        while current_time < end_time and self.customer_counter < total_customers:
            # While loop that runs until Duration or Total customers is reached
            current_time = time.time()  # Updates the current time
            if current_time > end_time or self.customer_counter >= total_customers:
                # Terminates the loop when one of the condition is reached
                with CheckoutLane.PRINT_LOCK:
                    print(
                        f"Simulation ended. Duration: {current_time - start_time:.2f} seconds, "
                        f"Customers processed: {self.customer_counter}")
                    # End statement that tells the Customers processed in how much time
                    raise KeyboardInterrupt  # Raises error to terminate the main loop
```

6. TESTING

Describe the process you took to test your code and to make sure the program functions as required. ***Make sure you include a test plan and demonstrate thorough testing of your own code as well as the integrated code****.*

THIS SECTION SHOULD BE COMPLETED INDIVIDUALLY FOR F1 AND F2 AND AS A GROUP FOR F3.

F2

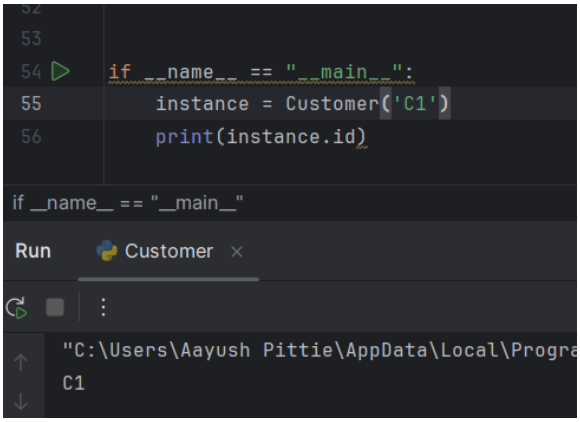
Test Case 1

Description: Test ID Initialization

Input: cid = C1

Expected output: self.id = 'C1'

Actual Output:



Test Pass/Fail: Pass

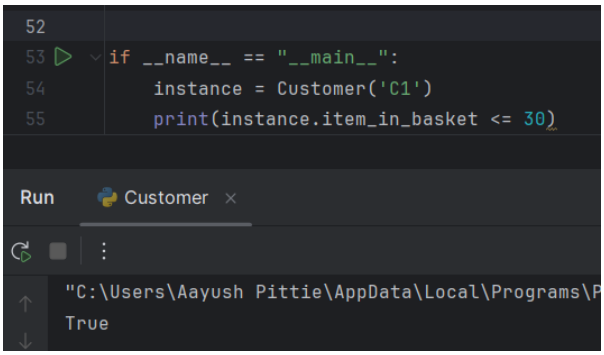
Test Case 2

Description: Test Item count Initialization

Input: cid = C1
print(self.item_in_basket<=30)

Expected output: True

Actual Output:



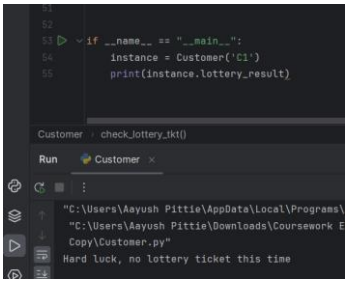
Test Pass/Fail: Pass

Test Case 3

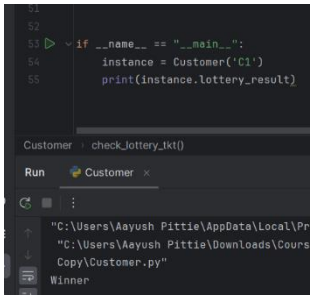
Description: Test Lottery Ticket Initialization

Input: cid = C1
print(self. lottery_result)

Expected output "Winner" or “"Hard luck, no lottery ticket this time"

Actual Output: 

or



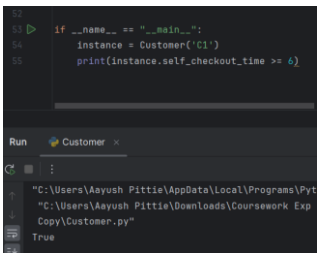
Test Pass/Fail: Pass

Test Case 4

Description: Test Self-checkout time Initialization

Input: cid = C1

Expected output: self. self_checkout_time >= 6

Actual Output: 

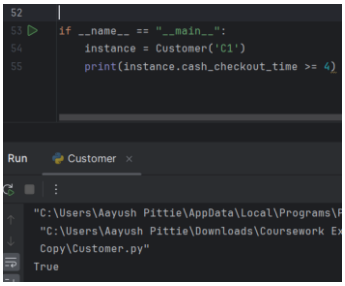
Test Pass/Fail: Pass

Test Case 5

Description: Test Self-checkout time Initialization

Input: cid = C1
self. cash_checkout_time >= 4

Expected output: True

Actual Output: 

Test Pass/Fail: Pass

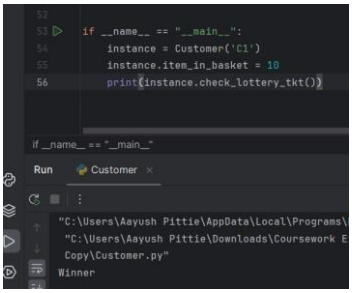
Test Case 6

Description: Test Lottery Ticket Winning Initialization

Input: cid = C1
instance.item_in_basket = 10

print(instance.check_lottery_tkt())

Expected output: Winner

Actual Output: 

Test Pass/Fail: Pass

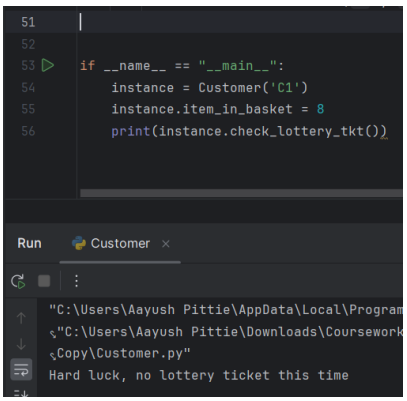
Test Case 7

Description: Test Lottery Ticket Losing Initialization

Input: cid = C1

instance.item_in_basket < 10
print(instance.check_lottery_tkt())

Expected output: Hard luck, no lottery ticket this time

Actual Output: 

Test Pass/Fail: Pass

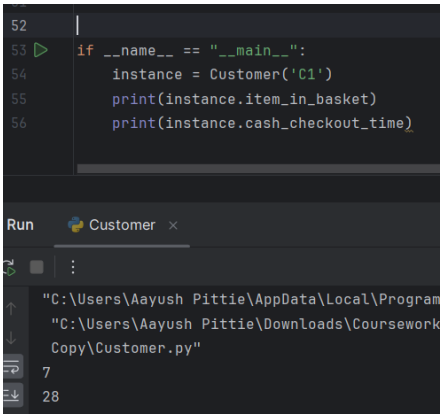
Test Case 8

Description: Checkout time Calculation

Input: cid = C1
print(instance.item_in_basket)

print(instance.cash_checkout_time)

Expected output: items * 4

Actual Output: 

Test Pass/Fail: Pass

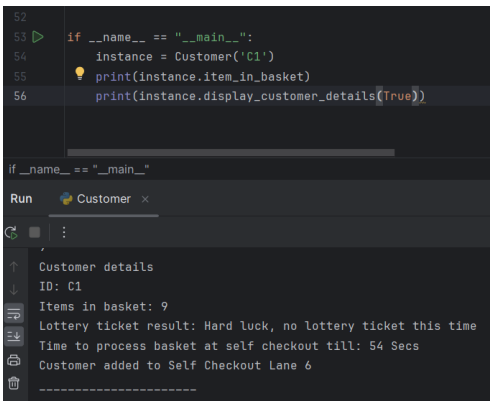
Test Case 9

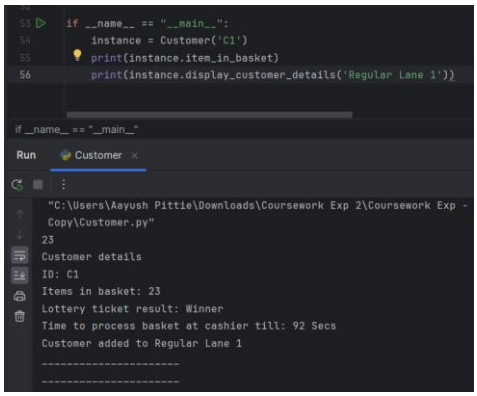
Description: Display details according to basket size

Input: cid = C1
print(instance.item_in_basket)

print(instance. display_customer_details(lane))

Expected output: Display self checkout lane if items < 10 or else display regular lane if items > 10

Actual Output: 



Test Pass/Fail: Pass

Test Case 10

Description: Test Random item generation

Input: x = y = 0

```
for i in range(0,100):
    instance = Customer(f'C{i}')
    if 30 >= instance.item_in_basket >= 10:
        x += 1
    elif instance.item_in_basket < 10:
        y += 1
    else:
        print('Error in Code')
print(x,y)
```

Expected output: ~50 ~50

```
52
53  if __name__ == "__main__":
54      x = 0
55      y = 0
56      for i in range(0,100):
57          instance = Customer(f'C{i}')
58          if 30 >= instance.item_in_basket >= 10:
59              x += 1
60          elif instance.item_in_basket < 10:
61              y += 1
62          else:
63              print('Error in Code')
64      print(x,y)
```

```
Run    Customer
"C:\Users\Aayush Pittie\AppData\Local\Programs\Pytho
 "C:\Users\Aayush Pittie\Downloads\Coursework Exp 2\
 Copy\Customer.py"
48 52
```

Actual Output:

Test Pass/Fail: Pass

F3

Test Case 1

Description Test simulation with set duration and customers

Input: instance.simulation(total_customers=10, duration=200)

Expected output: The last customer is C9

Actual Output:

Test Pass/Fail: Pass

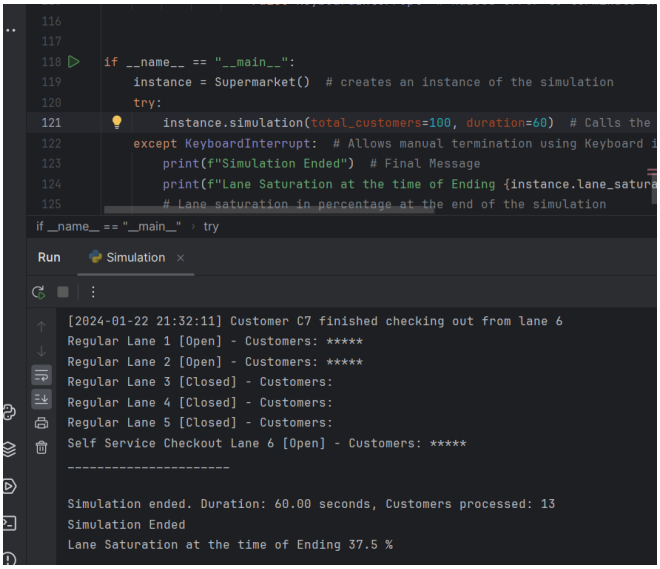
Test Case 2

Description Test simulation with set duration and customers

Input: instance.simulation(total_customers=100, duration=60)

Expected output: The simulation ends in a minute

Actual Output:



Test Pass/Fail: Pass

7. Annotated Screenshots Demonstrating Implementation

*Provide screenshots that demonstrate the features implemented running – i.e. showing the output produced by all of the subfeatures. Annotate each screenshot and if necessary, provide a brief description for **each** (**up to 100 words**) to explain the code in action.*

THIS SECTION SHOULD BE COMPLETED INDIVIDUALLY FOR F1 AND F2 AND AS A GROUP FOR F3.

7.1 Feature F2

a. Sub-feature I- screenshots

```
instance = Customer(1)
print(instance.display_customer_details('Regular Lane 1'))
```

(Code)

After creating the object Customer with ID: 1

Printing the Details of the customers that include random amount of items

```
ID: 1
Items in basket: 15
Lottery ticket result: Hard luck, no lottery ticket this time
Time to process basket at cashier till: 60 Secs
Customer added to Regular Lane 1
----------------------
----------------------
```

(Output)

b. Sub-feature II- screenshots …

```
instance = Customer(1)
print(instance.item_in_basket)
```

(Code)

Printing the Attribute that stores the item in basket

```
15

Process finished with exit code 0
```

(Output)

c. SUB-FEATURE III- SCREENSHOTS …

```
instance = Customer(1)
print(instance.display_customer_details('Regular Lane 1'))
```

(Code)

After creating the object Customer with ID: 1
Printing the Details of the customers that include their checkout time.
This is based on which lane they enter self-checkout or regular lane.

```
ID: 1
Items in basket: 15
Lottery ticket result: Hard luck, no lottery ticket this time
Time to process basket at cashier till: 60 Secs
Customer added to Regular Lane 1
----------------------
----------------------
```

(Output)

d. SUB-FEATURE IV- SCREENSHOTS …

```
instance = Customer(1)
print(instance.lottery_result)
```

(Code)

After creating the object Customer with ID:
Printing whether they won the lottery ticket or not.
It results in either of the two

```
Winner
```

```
Hard luck, no lottery ticket this time
```

(Outputs)

e. SUB-FEATURE V- SCREENSHOTS …

Calling the customer's display_customer_details method gives the following result

```
instance = Customer(1)
print(instance.display_customer_details('Regular Lane 1'))
```

(Code)

```
ID: 1
Items in basket: 15
Lottery ticket result: Hard luck, no lottery ticket this time
Time to process basket at cashier till: 60 Secs
Customer added to Regular Lane 1
----------------------
----------------------
```

(Output)

7.2 Feature F3

a. Sub-feature I- screenshots …

```
def __init__(self):
    """Initialises Lanes and No. of initial customers"""
    self.regular_lanes = [RegularLane(i) for i in range(1, 6)]  # Creating 5 Regular Lanes
    self.self_service_lane = [SelfServiceLane(6)]  # Creating 1 Self-service Lanes
    self.regular_lanes[0].open_lane()  # Starts the simulation with two open lanes
    self.self_service_lane[0].open_lane()  # One regular and one Self service
    self.initial_customers = random.randint( a: 1,  b: 10)  # Initial Customers
    self.customer_counter = self.initial_customers  # Counts the total customers processed
```

(Code)

The above code initializes 6 Lanes and opens two of them. The lane management method however closes these lanes if they are empty. In the below image I have the closing lane turned off hence the lanes are still open. Also, I didn't allow initial customers

```
Regular Lane 1 [Open] - Customers:
Regular Lane 2 [Closed] - Customers:
Regular Lane 3 [Closed] - Customers:
Regular Lane 4 [Closed] - Customers:
Regular Lane 5 [Closed] - Customers:
Self Service Checkout Lane 6 [Open] - Customers:
----------------------
```

(Output)

In this instance I ran it as normal and it generated 5 customers initially.

```
Customer details
ID: C5
Items in basket: 25
Lottery ticket result: Hard luck, no lottery ticket this time
Time to process basket at cashier till: 100 Secs
Customer added to Regular Lane 1
----------------------
----------------------

[2024-01-22 16:33:48] Processing customer C2 in lane 1
[2024-01-22 16:33:48] Processing customer C1 in lane 6
Regular Lane 1 [Open] - Customers: ***
Regular Lane 2 [Closed] - Customers:
Regular Lane 3 [Closed] - Customers:
Regular Lane 4 [Closed] - Customers:
Regular Lane 5 [Closed] - Customers:
Self Service Checkout Lane 6 [Open] - Customers: **
```

(Output- In lanes 5 Customers, last customer ID = C5)

b. SUB-FEATURE II- SCREENSHOTS

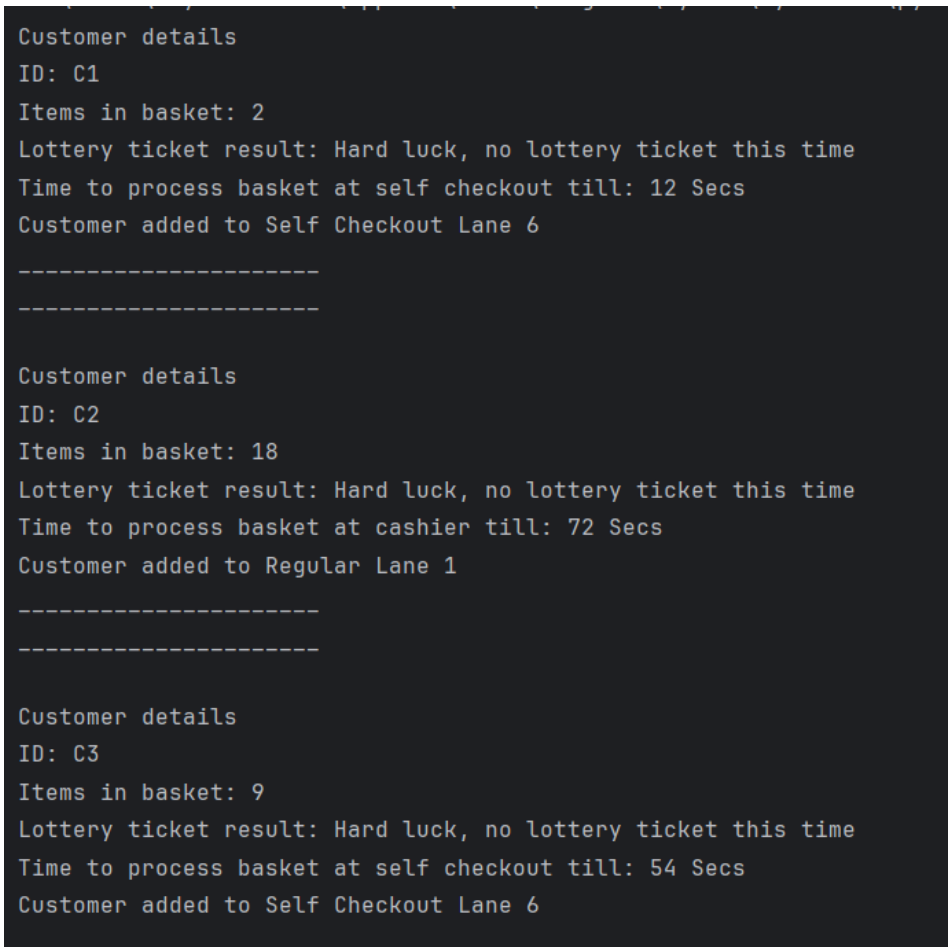
To record the time stamp we are using a variable and to initiate up to 10 random customers we have a method that uses self.inital_customers to randomly generate e the number and then runs the for loop to create that many customers.

```python
start_time = time.time()  # Start time is defined from real time clock
end_time = start_time + duration
self.generate_initial_customers()  # Calls the function to generate the initial customers
```

```python
self.initial_customers = random.randint( a: 1,  b: 10)  # Initial Customers
```

```python
1 usage
def generate_initial_customers(self):
    """Generates and Displays initial customers"""
    for _ in range(1, self.initial_customers):
        customer = Customer(cid=f"C{_}")
        x = self.enter_lane(customer)  # Adding them to a lane
        customer.display_customer_details(x)  # Displays the customer details
```

(Code)



(Output- Initial Customers)

c. SUB-FEATURE III- SCREENSHOTS …

To run the continuous simulation, I have defined two threads that perform these functions. Customers are generated in random time intervals between 1-10 seconds and the lanes are updated and displayed every 5 seconds.

In the code below you see two threads that make the simulation run continously

```python
def generate_customers(self):
    """Generates customers with unique iD"""
    while True:
        if self.lane_saturation() < 40 and not self.are_all_lanes_full():
            # Generates only when there is a space for  the customer to join in the lane
            # In other words it pauses customer generation when there is no space in the lanes
            customer = Customer(cid=f"C{self.customer_counter}")
            self.customer_counter += 1  # Increments the no. of customer generated
            x = self.enter_lane(customer)  # Enters the customer into a lane
            customer.display_customer_details(x)  # Displays the customer details
            self.enter_lane(customer)  # Enters the customer into a lane
            time.sleep(random.randint( a: 1,  b: 10))  # Generates a customer randomly between 1 and 10 Seconds
        else:
            print(f"Lane Saturation is {instance.lane_saturation() / 0.4} %")  # Dividing by 0.4 as 40 customers
            time.sleep(5)  # If all the lanes are full it waits for 5s and checks the Saturation again
            continue   # This allows customers to be processed

1 usage
def lane_management(self):
    """Manages all the Functions of the Lane Such as:
    Processing the Customer
    Displaying Lane Status
    Closing Empty Lane"""
    while True:
        for lane in self.regular_lanes + self.self_service_lane:
            lane.process_customer()  # This processes the customer
            lane.lane_close()  # This will close the lane if it's empty
        with Supermarket.PRINT_LOCK:
            for lanes in self.regular_lanes:
                lanes.display_lane_status()  # Displays status of Regular lanes, lane by lane
            self.self_service_lane[0].display_lane_status()  # Displays status of Self service lane
            time.sleep(5)  # This defines the speed of the simulation

1 usage
```

(Code)

```
[2024-01-22 17:21:25] Processing customer C2 in lane 1
[2024-01-22 17:21:25] Processing customer C1 in lane 6
Regular Lane 1 [Open] - Customers: ***
Regular Lane 2 [Closed] - Customers:
Regular Lane 3 [Closed] - Customers:
Regular Lane 4 [Closed] - Customers:
Regular Lane 5 [Closed] - Customers:
Self Service Checkout Lane 6 [Open] - Customers: ****
----------------------

Customer details
ID: C8
Items in basket: 13
Lottery ticket result: Hard luck, no lottery ticket this time
Time to process basket at cashier till: 52 Secs
Customer added to Regular Lane 1
----------------------
----------------------

[2024-01-22 17:21:30] Processing customer C2 in lane 1
[2024-01-22 17:21:30] Customer C1 finished checking out from lane 6
Regular Lane 1 [Open] - Customers: *****
Regular Lane 2 [Closed] - Customers:
Regular Lane 3 [Closed] - Customers:
Regular Lane 4 [Closed] - Customers:
Regular Lane 5 [Closed] - Customers:
Self Service Checkout Lane 6 [Open] - Customers: ***
----------------------
```

(Output- Note the Time stamps between two processing round, they are exactly 5s apart)

d. SUB-FEATURE IV- SCREENSHOTS …

To end the simulation I am taking a time duration and the max number of customers they want to generate when initiating the simulation Which is then checked by a while loop that runs until that criteria's are met. To terminate from user interruption I am running a try lop to end the simulation when a Keyboard Interrupt error is raised.

```python
if __name__ == "__main__":
    instance = Supermarket()  # creates an instance of the simulation
    try:
        instance.simulation(total_customers=100, duration=300)  # Calls the simulation
    except KeyboardInterrupt:  # Allows manual termination using Keyboard interrupt error
        print(f"Simulation Ended")  # Final Message
        print(f"Lane Saturation at the time of Ending {instance.lane_saturation() / 0.4} %")
        # Lane saturation in percentage at the end of the simulation
```

(Code)

```
while current_time < end_time and self.customer_counter < total_customers:
    # While loop that runs until Duration or Total customers is reached
    current_time = time.time()  # Updates the current time
    if current_time > end_time or self.customer_counter >= total_customers:
        # Terminates the loop when one of the condition is reached
        with Supermarket.PRINT_LOCK:
            print(
                f"Simulation ended. Duration: {current_time - start_time:.2f} seconds, "
                f"Customers processed: {self.customer_counter}")
            # End statement that tells the Customers processed in how much time
            raise KeyboardInterrupt  # Raises error to terminate the main loop
```

(Code)

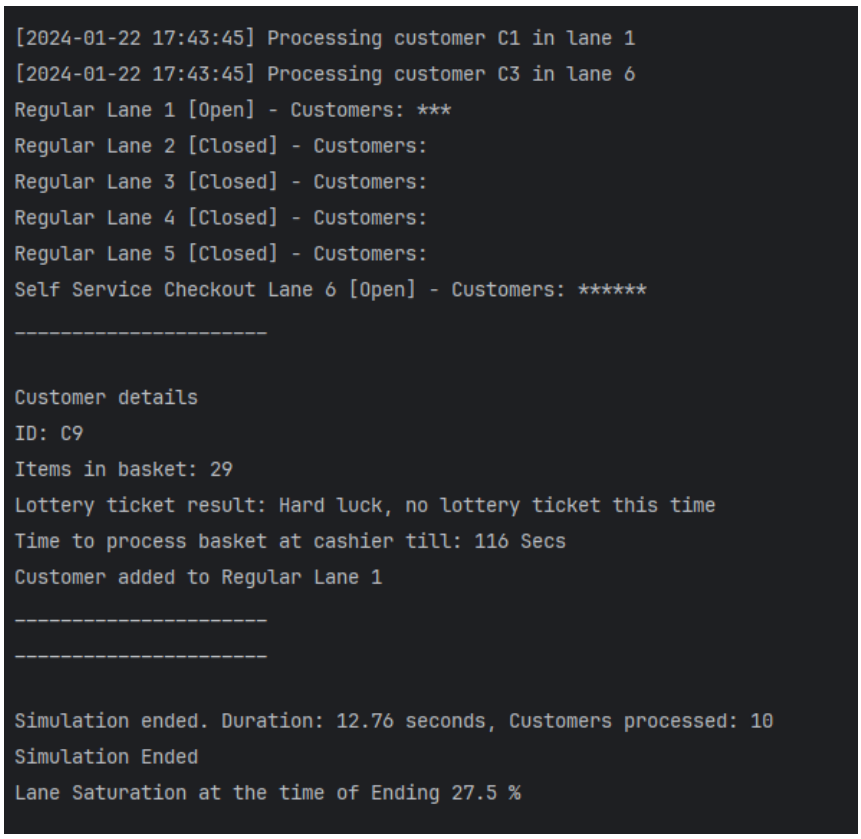
```
Regular Lane 3 [Closed] - Customers:
Regular Lane 4 [Closed] - Customers:
Regular Lane 5 [Closed] - Customers:
Self Service Checkout Lane 6 [Open] - Customers: ****
----------------------

[2024-01-22 17:40:23] Processing customer C2 in lane 1
[2024-01-22 17:40:23] Customer C3 finished checking out from lane 6
Regular Lane 1 [Open] - Customers: ****
Regular Lane 2 [Closed] - Customers:
Regular Lane 3 [Closed] - Customers:
Regular Lane 4 [Closed] - Customers:
Regular Lane 5 [Closed] - Customers:
Self Service Checkout Lane 6 [Open] - Customers: ***
----------------------

Simulation Ended
Lane Saturation at the time of Ending 17.5 %

Process finished with exit code 0
```

(Output – The code was terminated manually)



(Output – The code terminated Naturally)

8. OpenAI ComparisonF

Provide the code generated using OpenAI along with a listing of the code you initially wrote from scratch in a table showing the generated and your code side-by-side for each feature. Examine and explain the generated code's design, describing its quality and efficiency compared to the initial code you wrote. The narrative must also describe how you used the generated code to improve your own code or describe how the generated code may be improved.

Old Code	Open AI Suggestion

import random
 	
class Supermarket:
 	
  def __init__(self):
  """Initializes Lanes and No. of initial customers"""
  self.regular_lanes = [RegularLane(i) for i in range(1, 6)] # Creating 5 Regular Lanes
  self.self_service_lane = [SelfServiceLane(6)] # Creating 1 Self-service Lanes
  self.regular_lanes[0].open_lane() # Starts the simulation with one open regular lane
  self.self_service_lane[0].open_lane() # Starts the simulation with the self-service lane open
  self.initial_customers = random.randint(1, 10) # Initial Customers
  self.customer_counter = self.initial_customers # Counts the total customers processed
 	
  def are_all_lanes_full(self):
  """Checks are all Regular lanes Full"""
  return all(lane.lane_is_full() for lane in self.regular_lanes)
 	
  def generate_initial_customers(self):
  """Generates and Displays initial customers"""
  for _ in range(self.initial_customers):
  customer = Customer(cid=f"C{_}")
  x = self.enter_lane(customer) # Add them to a lane
  customer.display_customer_details(x)
 	
  def enter_lane(self, customer):
  """Adds customers to lanes"""
	import random
	import time
	import threading
	class SimplifiedSupermarket:
	  """Simplified Main Simulation Class with Threading"""
	  def __init__(self, max_customers, duration):
	  """Initializes Lanes and No. of initial customers"""
	  self.regular_lanes = [RegularLane(i) for i in range(1, 6)] # Creating 5 Regular Lanes
	  self.self_service_lane = [SelfServiceLane(6)] # Creating 1 Self-service Lane
	  self.regular_lanes[0].open_lane() # Starts the simulation with one open regular lane
	  self.self_service_lane[0].open_lane() # Starts the simulation with the self-service lane open
	  self.initial_customers = random.randint(1, 10) # Initial Customers
	  self.customer_counter = self.initial_customers # Counts the total customers processed
	  self.max_customers = max_customers # maximum number of customers for the simulation
	  self.duration = duration # Duration of the simulation in seconds
	  self.stop_event = threading.Event() # Event to stop the threads
	  def generate_customers(self):
	  """Generates customers with unique ID"""
	  while not self.stop_event.is_set() and self.customer_counter < self.max_customers:
	  if self.lane_saturation() <= 40 and not self.are_all_lanes_full():
	  customer = Customer(cid=f"C{self.customer_counter}")
	  self.customer_counter += 1 # Increments the no. of customer generated

```
        if customer.item_in_basket < 10 and not self.self_service_lane[0].lane_is_full():

            self.self_service_lane[0].add_customer(customer)  # Adds customer with less than 10 items only
            return True
        else:
            processing_time = [lane.lane_usage() for lane in self.regular_lanes]
            x = processing_time.index(min(processing_time))
            regular_lane_with_shortest_queue = self.regular_lanes[x]

            regular_lane_with_shortest_queue.add_customer(customer)  # Adds customer to the shortest lane
            return f"Regular Lane {x + 1}"

    def lane_saturation(self):
        """Calculates total customers Standing in line"""
        users = sum(len(lane.customers) for lane in self.regular_lanes)
        users += len(self.self_service_lane[0].customers)
        return users

    def simulation(self, total_customers):
        """Initializes the Total customers to be generated"""
        self.generate_initial_customers()  # Calls the function to generate the initial customers
        while self.customer_counter < total_customers:
            # Process customers
            for lane in self.regular_lanes + self.self_service_lane:
                lane.process_customer()  # This processes the customer
                lane.lane_close()  # This will close the lane if it's empty
            for lane in self.regular_lanes + self.self_service_lane:
                lane.display_lane_status()  # Displays status of lanes
            self.customer_counter += 1  # Increment the number of processed customers
            x = self.enter_lane(customer)  # Enters the customer into a lane
            customer.display_customer_details(x)  # Displays the customer details
            time.sleep(random.randint(1, 10))  # Generates a customer randomly between 1 and 10 Seconds
        else:
            print(f"Lane Saturation is {self.lane_saturation() / 0.4} %")
            time.sleep(5)  # If all the lanes are full, it waits for 5s and checks the Saturation again
    def lane_management(self):
        """Manages all the Functions of the Lane"""
        while not self.stop_event.is_set():
            for lane in self.regular_lanes + self.self_service_lane:
                lane.process_customer()  # This processes the customer
                lane.lane_close()  # This will close the lane if it's empty
            for lane in self.regular_lanes + self.self_service_lane:
                lane.display_lane_status()  # Displays status of lanes
            time.sleep(5)  # This defines the speed of the simulation
    def simulation(self):
        """Starts the simulation with threading"""
        self.generate_initial_customers()  # Calls the function to generate the initial customers
        customer_generation = threading.Thread(target=self.generate_customers)
        lane_management = threading.Thread(target=self.lane_management)
        # Starting the threads
        customer_generation.start()
        lane_management.start()
        # Running the simulation for a specified duration
        time.sleep(self.duration)
        # Stopping the threads
        self.stop_event.set()
        customer_generation.join()
        lane_management.join()
```

print(f"Simulation ended. Customers processed: {self.customer_counter}")	print(f"Simulation ended. Customers processed: {self.customer_counter}")

The supermarket simulation's upgrade introduces threading, enabling simultaneous operations like customer generation and lane management, closely mimicking real-life supermarkets. Dynamic customer generation continues until reaching a limit or a stop signal, enhancing realism. Concurrently, lane management efficiently processes customers, with the simulation's duration and customer capacity controllable, ensuring a realistic, time-bound flow. The introduction of a graceful shutdown mechanism ensures orderly completion, with time.sleep adding a realistic final note to customer arrival and processing, making the simulation a more accurate reflection of an actual supermarket.

9. SELF-ASSESSMENT

Please assess yourself objectively for each section shown below and then enter the total mark you expect to get. Marks for each assessment criterion are indicated between parentheses.

Code development (70)

a. Features Implemented [36] (group work and integration will be assessed here)

Partner A or Partner B features (up to 18)

Sub-features have not been implemented – 0
Attempted, not complete or very buggy – 1 to 5
Implemented and functioning without errors but not integrated – 6 to 10
Implemented and fully integrated but buggy – 11 to 15
Implemented, fully integrated and functioning without errors – 16 to 18

Group Features (up to 18)

Sub-features has not been implemented – 0
Attempted, not complete or very buggy – 1 to 5
Implemented and functioning without errors but not integrated – 6 to 10
Implemented and fully integrated but buggy – 11 to 15
Implemented, fully integrated and functioning without errors – 16 to 18

For this criterion I think I got: 34 out of 36

b. Use of OOP techniques [24]

Abstraction (up to 8)

No classes have been created – 0
Classes have been created superficially and not instantiated or used – 1 or 2
Classes have been created but only some have been instantiated and used – 3 or 4
Useful classes and objects have been created and used correctly – 5 or 6
The use of classes and objects exceeds the specification – 7 or 8

Encapsulation (up to 8)

No encapsulation has been used – 0
Class variables and methods have been encapsulated superficially – 1 to 3
Class variables and methods have been encapsulated correctly – 4 to 6
The use of encapsulation exceeds the specification – 6 to 8

Inheritance or polymorphism (up to 8)

No inheritance or polymorphism has been used – 0
Inheritance or polymorphism has been used superficially – – 1 to 3
Inheritance or polymorphism has been used correctly – 4 to 6
The use of inheritance or polymorphism exceeds the specification – 6 to 8

For this criterion I think I got: 21 out of 24

c. Quality of Code [10]

Code Duplication (up to 4)

Code contains too many unnecessary code repetition – 0
Regular occurrences of duplicate code – 1
Occasional duplicate code – 2
Very little duplicate code – 3

No duplicate code – 4

PEP8 Conventions and naming of variables, methods and classes (up to 3)

PEP8 and naming convention has not been used – 0
PEP8 and naming convention has been used occasionally – 1
PEP8 and naming convention has been used regularly – 2
PEP8 convention used professionally and all items have been named correctly – 3

In-code Comments (up to 3)

No in-code comments – 0
Code contains occasional in-code comments – 1
Code contains useful and regular in-code comments – 2
Thoroughly commented, good use of docstrings, and header comments describing.py files – 3

For this criterion I think I got: 9 out of 10

2. Documentation (20)

Design (up to 10) clear exposition about the design and decisions for OOP use

The documentation cannot be understood on first reading or is mostly incomplete – 0
The documentation is readable, but a section(s) are missing – 1 to 3
The documentation is complete – 4 to 6
The documentation is complete and of a high standard – 7 to 10

Testing (10)

Testing has not been demonstrated in the documentation – 0
A test plan has been included but is incomplete – 1 or 2
A test plan has been included with some appropriate test cases – 3 to 6
A full test plan has been included with thorough test cases and evidence of carrying it out – 7 to 10

For this criterion I think I got: 20 out of 20

3. Acceptance Test - Demonstration (10)

Final Demo (up to 10)

Not attended or no work demonstrated – 0
Work demonstrated was not up to the standard expected, superficial team contribution – 1 to 3
Work demonstrated was up to the standard expected, sufficient team contribution – 4 to 7
Work demonstrated exceeded the standard expected – 8 to 10

For this criterion I think I got: 10 out of 10

I think my overall mark would be: 94 out of 100

APPENDIX A: CODE LISTING

*Provide a complete listing of all the *.py files in your PyCharm project. Make sure your code is well commented and applies professional Python convention (refer to PEP 8 for details). The code listed here must match that uploaded to Moodle. Please copy and paste the actual code – no screenshots please! You will lose marks if screenshots are provided instead of code. Clearly label the parts each partner created with their name and SID.*

Lane.py

```
# Panagiotis Petsallari 001294962 F1


import time
from collections import deque
import threading


class CheckoutLane:
    """Creates Checkout Lane Object"""
    PRINT_LOCK = threading.Lock()  # Defines PRINT_LOCK as Lock function of
threading

    def __init__(self, lane_id, capacity):
        """Initializes CheckoutLane with lane_id and capacity."""
        self.lane_id = lane_id  # Identifier of Lane
        self.status = 'closed'  # Lane status is closed by default
        self.customers = deque()  # Queue to manage customers
        self.capacity = capacity  # Maximum number of customers the lane can hold

    def lane_usage(self):
        """Calculates total number of items in customers' baskets."""
        usage = 0
        for cust in self.customers:  # Uses this usage value ot determine the
shortest queue
            usage += cust.item_in_basket
        if self.lane_is_full():  # Increasing the value of a Full lane, so it
never gets selected
            usage += 150
        elif self.status == 'closed':  # Closed lane has some value as to not open
the first available closed lane
            usage += 80                # until all other lanes have usage value
over
        return usage

    def lane_is_full(self):
        """Returns boolean value on whether lane is full"""
        return len(self.customers) == self.capacity

    def lane_close(self):
        """Closes lane if no customers are present."""
        if self.is_empty():  # Is initiated each time lanes are processed
            self.status = 'closed'

    def is_empty(self):
        """Checks if lane is empty."""
        return len(self.customers) == 0

    def open_lane(self):
```

```
        """Opens lane"""
        self.status = 'open'


    def close_lane(self):
        """Closes lane"""
        self.status = 'closed'

    def process_customer(self):
        """Processes the first customer in the lane."""
        with CheckoutLane.PRINT_LOCK:  # Allows multiple threads to access Print
resource
            if self.customers:         # and to prevent interference from other
threads
                customer = self.customers[0]    # processes the first customer
                if customer.item_in_basket > 0:  # checks if it's basket is
empty(0)
                    customer.item_in_basket -= 2  # Decrement remaining items in
the basket
                    current_time_formatted = time.strftime('%Y-%m-%d %H:%M:%S',
time.localtime())
                    print(f"[{current_time_formatted}] Processing customer
{customer.id} in lane {self.lane_id}")
                    #  Prints the time at processing and the lane at which it is
being processed
                elif customer.item_in_basket <= 0:  # Customer leaves lane when
items are zero
                    completed_customer = self.customers.popleft()
                    current_time_formatted = time.strftime('%Y-%m-%d %H:%M:%S',
time.localtime())
                    print(
                        f"[{current_time_formatted}] Customer
{completed_customer.id} finished "
                        f"checking out from lane {self.lane_id}")

    def add_customer(self, customer):
        """Adds the customer to a Lane"""
        with CheckoutLane.PRINT_LOCK:
            # Adds a customer to the lane if there's capacity, otherwise indicates
the lane is full.
            # Opens the lane if it's currently closed.
            if self.status == 'closed':
                self.open_lane()
            if len(self.customers) < self.capacity:
                self.customers.append(customer)  # Add the customer to the lane
            else:
                print("Lane is full. Customer cannot be added.")


class RegularLane(CheckoutLane):
    """Initializing a lane with capacity of 5 people"""
    def __init__(self, lane_id):
        """Inherits from Parent Class Checkout Lane"""
        super().__init__(lane_id, capacity=5)

    def display_lane_status(self):
        """Displays Lane Status"""
        with CheckoutLane.PRINT_LOCK:
            # Code to display the current status of the lane with "Regular" prefix
            status = "Open" if self.status == 'open' else "Closed"
            customers_in_line = ''.join(['*' for _ in self.customers])  #
visualizes customers as Stars
```

```
            print(f"Regular Lane {self.lane_id} [{status}] - Customers:
{customers_in_line}")


class SelfServiceLane(CheckoutLane):
    """Initializing a lane with capacity of 15 people"""
    def __init__(self, lane_id):
        super().__init__(lane_id, capacity=15)

    def process_customer(self):
        """Processes the first customer in the lane for Self service lanes."""
        # This code is duplicated so, we can have different processing speeds for
regular lanes and self-checkout lanes
        with CheckoutLane.PRINT_LOCK:
            if self.customers:
                customer = self.customers[0]  # processes the first customer
                if customer.item_in_basket > 0:   # checks if it's basket is
empty(0)
                    customer.item_in_basket -= 8   # Decrement remaining items in
the basket
                    current_time_formatted = time.strftime('%Y-%m-%d %H:%M:%S',
time.localtime())
                    print(f"[{current_time_formatted}] Processing customer
{customer.id} in lane {self.lane_id}")
                    #  Prints the time at processing and the lane at which it is
being processed
                elif customer.item_in_basket <= 0:      # Customer leaves lane
when items are zero
                    completed_customer = self.customers.popleft()
                    current_time_formatted = time.strftime('%Y-%m-%d %H:%M:%S',
time.localtime())
                    print(
                        f"[{current_time_formatted}] Customer
{completed_customer.id} finished "
                        f"checking out from lane {self.lane_id}")

    def display_lane_status(self):
        """Displays Lane Status"""
        with CheckoutLane.PRINT_LOCK:
            # Code to display the current status of the lane with "Self Service
Checkout" prefix
            status = "Open" if self.status == 'open' else "Closed"
            customers_in_line = ''.join(['*' for _ in self.customers])  #
visualizes customers as Stars
            print(f"Self Service Checkout Lane {self.lane_id} [{status}] -
Customers: {customers_in_line}")
            print(f'_____________________ \n')
```

Simulation.py

```
# Panagiotis Petsallari 001294962
# Aayush Pittie 001328860
# The code for F3 below was done jointly by both of the members and the
contribution was equal

import time
import random
from Lanes import SelfServiceLane, RegularLane, CheckoutLane
from Customer import Customer
```

```
import threading


class Supermarket:
    """Main Simulation Class"""
    PRINT_LOCK = threading.Lock()

    def __init__(self):
        """Initialises Lanes and No. of initial customers"""
        self.regular_lanes = [RegularLane(i) for i in range(1, 6)]  # Creating 5
Regular Lanes
        self.self_service_lane = [SelfServiceLane(6)]  # Creating 1 Self-service
Lanes
        self.regular_lanes[0].open_lane()  # Starts the simulation with two open
lanes
        self.self_service_lane[0].open_lane()  # One regular and one Self service
        self.initial_customers = random.randint(1, 10)  # Initial Customers
        self.customer_counter = self.initial_customers  # Counts the total
customers processed

    def are_all_lanes_full(self):
        """Checks are all Regular lanes Full"""
        return all(lane.lane_is_full() for lane in self.regular_lanes)

    def generate_initial_customers(self):
        """Generates and Displays initial customers"""
        for _ in range(1, self.initial_customers):
            customer = Customer(cid=f"C{_}")
            x = self.enter_lane(customer)  # Adding them to a lane
            customer.display_customer_details(x)  # Displays the customer details

    def enter_lane(self, customer):  # Panagiotis Petsallari 001294962 F1
        """Adds customers to lanes"""
        if customer.item_in_basket < 10 and not
self.self_service_lane[0].lane_is_full():
            self.self_service_lane[0].add_customer(customer)  # Adds customer with
less than 10 items only
            return True
        else:
            processing_time = [lane.lane_usage() for lane in self.regular_lanes]
            x = processing_time.index(min(processing_time))  # Finds the shortest
lane for customer to join
            regular_lane_with_shortest_queue = self.regular_lanes[x]
            regular_lane_with_shortest_queue.add_customer(customer)  # Adds
customer to the shortest lane
            return f'Regular Lane {x + 1}'

    def lane_saturation(self):
        """Calculates total customers Standing in line"""
        users = 0
        for lane in self.regular_lanes:  # Iterates over all lanes to add the
customers up
            users = users + (len(lane.customers))
        users += len(self.self_service_lane[0].customers)
        return users

    def generate_customers(self):
        """Generates customers with unique iD"""
        while True:
            if self.lane_saturation() < 40 and not self.are_all_lanes_full():
                # Generates only when there is a space for  the customer to join
in the lane
```

```
                # In other words it pauses customer generation when there is no
space in the lanes
                customer = Customer(cid=f"C{self.customer_counter}")
                self.customer_counter += 1  # Increments the no. of customer
generated
                x = self.enter_lane(customer)  # Enters the customer into a lane
                customer.display_customer_details(x)  # Displays the customer
details
                self.enter_lane(customer)  # Enters the customer into a lane
                time.sleep(random.randint(1, 10))  # Generates a customer randomly
between 1 and 10 Seconds
            else:
                print(f"Lane Saturation is {instance.lane_saturation() / 0.4} %")
# Dividing by 0.4 as 40 customers
                time.sleep(5)  # If all the lanes are full it waits for 5s and
checks the Saturation again
                continue   # This allows customers to be processed

    def lane_management(self):
        """Manages all the Functions of the Lane Such as:
        Processing the Customer
        Displaying Lane Status
        Closing Empty Lane"""
        while True:
            for lane in self.regular_lanes + self.self_service_lane:
                lane.process_customer()  # This processes the customer
                lane.lane_close()  # This will close the lane if it's empty
            with Supermarket.PRINT_LOCK:
                for lanes in self.regular_lanes:
                    lanes.display_lane_status()  # Displays status of Regular
lanes, lane by lane
                self.self_service_lane[0].display_lane_status()  # Displays status
of Self service lane
                time.sleep(5)  # This defines the speed of the simulation

    def simulation(self, total_customers, duration):
        """Initialises the Total customers to be generated and the Duration of the
Simulation
        Also calls all the Starting methods and Starts threading"""
        start_time = time.time()  # Start time is defined from real time clock
        end_time = start_time + duration
        self.generate_initial_customers()  # Calls the function to generate the
initial customers
        customer_generation = threading.Thread(target=self.generate_customers)
        lane_generation = threading.Thread(target=self.lane_management)
        # Threads the two main Methods Customer Generation and lane management
allowing both to execute independently
        # of each other. It also allows the ability to have different speeds at
which
        # Lanes are processed or customers are generated
        customer_generation.daemon = True
        lane_generation.daemon = True
        current_time = 0  # Initialising for the While Loop
        lane_generation.start()  # Starts the first thread
        time.sleep(0.5)  # Time gap so the threads don't interfere as they have
different phases
        customer_generation.start()  # Second thread starts
        while current_time < end_time and self.customer_counter < total_customers:
            # While loop that runs until Duration or Total customers is reached
            current_time = time.time()  # Updates the current time
            if current_time > end_time or self.customer_counter >=
total_customers:
```

```
                # Terminates the loop when one of the condition is reached
                with CheckoutLane.PRINT_LOCK:
                    print(
                        f"Simulation ended. Duration: {current_time -
start_time:.2f} seconds, "
                        f"Customers processed: {self.customer_counter}")
                    # End statement that tells the Customers processed in how much
time
                    raise KeyboardInterrupt  # Raises error to terminate the main
loop


if __name__ == "__main__":
    instance = Supermarket()  # creates an instance of the simulation
    try:
        instance.simulation(total_customers=100, duration=400)  # Calls the
simulation
    except KeyboardInterrupt:  # Allows manual termination using Keyboard
interrupt error
        print(f"Simulation Ended")  # Final Message
        print(f"Lane Saturation at the time of Ending {instance.lane_saturation()
/ 0.4} %")
        # Lane saturation in percentage at the end of the simulation
```

Customer.py

```
# Aayush Pittie 001328860 F2
import random
from Lanes import CheckoutLane


class Customer:
    """ Creates Customer Object"""
    def __init__(self, cid):
        """ Initializes Customers with Customer ID."""
        self.id = cid  # Unique identifier of Customer
        self.item_in_basket = self.random_number_of_items()  # Generating random
number of items in basket
        self.lottery_result = self.check_lottery_tkt()  # Lottery result for the
customer
        self.self_checkout_time = self.checkout_time(6)  # Check out time at Self-
service till
        self.cash_checkout_time = self.checkout_time(4)  # Check out time at
Cashier till

    def check_lottery_tkt(self):
        """Checks if a customer wins a lottery ticket"""
        if self.item_in_basket >= 10 and random.random() < 0.1:  # The number of
items should be >= 10
            result = "Winner"                                    # Sets a 10%
Probability to win a ticket
        else:
            result = "Hard luck, no lottery ticket this time"
        return result

    def checkout_time(self, n):
        """Calculates the Checkout time upon number of items"""
        return self.item_in_basket * n  # Multiplies by parameter n which is time
taken to scan a single item
```

```
    def display_customer_details(self, lane):
        """Prints the customer details in a Set Format"""
        with CheckoutLane.PRINT_LOCK:  # Allows multiple threads to access Print
resource
            print("Customer details")  # and to prevent interference from other
threads
            print(f"ID: {self.id}")
            print(f"Items in basket: {self.item_in_basket}")
            print(f"Lottery ticket result: {self.lottery_result}")
            if self.item_in_basket < 10 and lane:  # Prints depending upon which
lane they are likely to go
                print(f"Time to process basket at self checkout till:
{self.self_checkout_time} Secs")
                print('Customer added to Self Checkout Lane 6')
            else:
                print(f"Time to process basket at cashier till:
{self.cash_checkout_time} Secs")
                print(f'Customer added to {lane}')
            print('_________________________')
            print(f'_________________________ \n')

    @staticmethod
    def random_number_of_items():
        """Calculates the Items in basket of a Customer randomly"""
        if random.choice([True, False]):   # This method allows more customers
with less than 10 items to be generated
            return random.randint(10, 30)  # Can be modified to have more or less
of a type of customer
        else:  # Currently 50% chance to have less than 10 items which can be
modified
            return random.randint(1, 10)  # with changing the amount of true
values in the list
```