DESIGN PATTERNS TEST

Q7)
7) A Small-scale vegetable vendor wants to develop a system to manage his door-delivery service. The system should maintain the list of available vegetables for the day. The system should accept a list of vegetables with the individual quantities required by the customer and assign order numbers for tracking. The system should be able to generate the bill for the order and ensure that orders should be place for more than Rs. 100. The system should confirm the payment thorugh notifications to the customer and the vendor. Appending the order should be disabled after the payment. The availability of the vegetables should be updated to the vendor and the customer (during ordering). When the order is ready, a notification should be sent to the customer mentioning the tentative time of delivery. Use appropriate patterns to list the items eligible for door delivery. Demonstrate the unit testing for the given scenario. Implement multiple threading to do the work of the ordering and delivery work in parallell.

Python

```python
from abc import ABC, abstractmethod

#Subject interface
class VegetableVendorSubject(ABC):
  @abstractmethod
  def register_observer(self, observer):
    pass

  @abstractmethod
  def remove_observer(self, observer):
    pass

  @abstractmethod
  def notify_observers(self):
    pass

#Observer Interface
class VegetableObserver(ABC):
  @abstractmethod
  def update_vegetable_availability(self, vegetables):
    pass

  @abstractmethod
  def confirm_payment(self, order_amount):
```

```
    pass

  @abstractmethod
  def notify_order_ready(self, delivery_time):
    pass

#Concrete Subject
class VegetableVendorSystem(VegetableVendorSubject):
  def __init__(self):
    self._observers = []
    self._vegetable_availability = {}
    self._orders = []
    self._total_sales = 0

  def register_observer(self, observer):
    self._observers.append(observer)

  def remove_observer(self, observer):
    self._observers.remove(observer)

  def notify_observers(self):
    for observer in self._observers:
      observer.update_vegetable_availability(self._vegetable_availability)

  def update_vegetable_availability(self, vegetables):
    self._vegetable_availability.update(vegetables)
    self.notify_observers()

  def confirm_payment(self, order_amount):
    if order_amount > 100:
      self._total_sales += order_amount
      for observer in self._observers:
        observer.confirm_payment(order_amount)
      self._orders = None
    else:
      print("Orders should be placed for more than Rs. 100")

  def place_order(self, order):
    if self._orders is not None:
      self._orders.append(order)
      print(f"Order placed. Order number: {len(self._orders)}")
            for key, val in order.items():
                self._vegetable_availability[key.title()] -= val
            print("Remainging vegetables:", self._vegetable_availability)
        else:
```

```
            print("Order placement is disabled after payment.")

    def notify_order_ready(self, delivery_time):
        for observer in self._observers:
            observer.notify_order_ready(delivery_time)

#Concrete Observer
class CustomerObserver(VegetableObserver):
    def update_vegetable_availability(self, vegetables):
        print(f"Available Vegetables: {vegetables}")

    def confirm_payment(self, order_amount):
        print(f"Payment confirmed. Amount: Rs. {order_amount}")

    def notify_order_ready(self, delivery_time):
        print(f"Your order is ready. It will be delivered at {delivery_time}")

class VendorObserver(VegetableObserver):
    def update_vegetable_availability(self, vegetables):
        pass

    def confirm_payment(self, order_amount):
        print(f"Payment received from customer. Amount: Rs. {order_amount}")

    def notify_order_ready(self, delivery_time):
        print(f"Order ready for delivery at {delivery_time}. Notify customer.")

#Iterator Pattern
class Iterable:

    def create_iterator(self):
        pass

    def append(self, item):
        pass

class Iterator:

    def has_next(self):
        pass
    def next(self):
        pass

class ObserversList(Iterable):
```

```
    def __init__(self):
        self._observers = []

    def create_iterator(self):
        return ObserversListIterator(self)

    def append(self, obs):
        self._observers.append(obs)

class ObserversListIterator(Iterator):

    def __init__(self, iterable):
        self._iterable = iterable
        self._index = 0

    def has_next(self):
        return self._index < len(self._iterable._observers)

    def next(self):
        if self.has_next():
            value = self._iterable._observers[self._index]
            self._index += 1
        else:
            raise StopIteration("No more Elements")

if __name__ == '__main__':
  vendor_system = VegetableVendorSystem()
  stock = {"Tomato": 50, "Carrot": 50, "Spinach": 50, "Onion": 50, "Garlic": 50}

  customer_observer = CustomerObserver()
  vendor_observer = VendorObserver()

  vendor_system.register_observer(customer_observer)
  vendor_system.register_observer(vendor_observer)

  vendor_system.update_vegetable_availability(stock)

  placeOrder = {}
  for name in stock.keys():
    placeOrder[name] = 0

  process = True

  n = int(input("Enter how many vegetables u r going to purchase: "))
  for i in range(n):
```

```
    name = input(f"Enter vegetable{i+1} name: ")
        qty = int(input(f"Enter the quantity: "))
        if stock[name.title()] > qty:
            placeOrder[name.title()] += qty
        else:
            print(f"Sorry we ran out of {name}")

print(f'Your Current Order is:', placeOrder)

vendor_system.place_order(placeOrder)

vendor_system.confirm_payment(120)
```

OUTPUT:

```
========= RESTART: C:\Users\exam44\Desktop\design patterns\roughtry.py =========
Available Vegetables: {'Tomato': 50, 'Carrot': 50, 'Spinach': 50, 'Onion': 50, 'Garlic': 50}
Enter q to quit
Enter vegetable name: tomato
Enter the quantity: 30
Enter vegetable name: onion
Enter the quantity: 30
Enter vegetable name: garlic
Enter the quantity: 10
Enter vegetable name: q
Your Current Order is: {'Tomato': 30, 'Carrot': 0, 'Spinach': 0, 'Onion': 30, 'Garlic': 10}
Order placed. Order number: 1
Remainging vegetables: {'Tomato': 20, 'Carrot': 50, 'Spinach': 50, 'Onion': 20, 'Garlic': 40}
Payment confirmed. Amount: Rs. 120
Payment received from customer. Amount: Rs. 120
```

Q10) An organisation like SSNCE wants to develop a Helpdesk ticketing system. Any member of the organisation should be able to log a request/complaint regarding certain services. The system should assign a unique ticket number for each request . A ticket should be assigned to a service engineer based on the service type. A service engineer should be able to view all the tickets assigned to them. once the service is completed, the service engineer will close the ticket. The management should be able to generate various reports for analyzing and improving the services. The service may be hardware/software/civil maintanence. Different tickets should be dispatched to different departments. Implement suitable patterns for the ticket.

Python

```python
from abc import ABC, abstractmethod
import random

#Receiver
class System:

  def __init__(self):
    self.ticket = []

  def add_ticket(self, ticket):
    self.ticket.append(ticket)

  def remove_ticket(self, ticket):
    self.ticket.pop()

#Icommand Interface
class Icommand(ABC):

  @abstractmethod
  def execute(self):
    pass

#Command1
class Request(Icommand):

  def __init__(self, system, ticket_type, req_no):
    self.system = system
    self.ticket_type = ticket_type
    self.req_no = req_no

  def execute(self):
    print(f"Your Request is accepted. Your Request Number is {self.req_no}")
        self.system.add_ticket((self.ticket_type, self.req_no))
```

```
#Command2
class Complaint(Icommand):

    def __init__(self, system,ticket_type, comp_no):
        self.system = system
        self.ticket_type = ticket_type
        self.comp_no = comp_no

    def execute(self):
        print("Your Complaint is Raised. Your Complaint Number is {self.comp_no}")
        self.system.add_ticket((self.ticket_type, self.comp_no))


#Invoker
class Service_Engineer:
    def __init__(self):
        self.command = None

    def set_command(self, command):
        self.command = command

    def execute_command(self):
        if self.command:
            self.command.execute()
        else:
            print("No Command Set")


#Driver code
if __name__ == '__main__':
  print("----------------SSNCE HELPDESK-----------------")
  print()
  system = System()
  invoker = Service_Engineer()

  #Request1
  while True:
    check = input("Do you want to exit(y/n): ")
    if check.lower() == 'y':
      print("Thank you for using SSNCE Helpdesk")
      if len(system.ticket) > 0:
        print("Our Service Engineer will look into your Queries")
      break
```

```
else:
  name = input("Do you have a request or complaint: ")
  service = input("What is it about? ")
  tick_num = random.randint(1,1000000)
  print()
  if name.lower() == 'request':
    req = Request(system, service, tick_num)
    print("Initial Tickets:", system.ticket)
    print()
    invoker.set_command(req)
    invoker.execute_command()
    print("After Command:", system.ticket)
    print()
  elif name.lower() == 'complaint':
    comp = Complaint(system, service, tick_num)
    print("Initial Tickets:", system.ticket)
    print()
    invoker.set_command(comp)
    invoker.execute_command()
    print("After Command:", system.ticket)
    print()
  else:
    print("Invalid operation")
    break
  print()
```

Output:

```
----------------SSNCE HELPDESK-----------------

Do you want to exit(y/n): n
Do you have a request or complaint: request
What is it about? hardware

Initial Tickets: []

Your Request is accepted. Your Request Number is 319858
After Command: [('hardware', 319858)]


Do you want to exit(y/n): n
Do you have a request or complaint: complaint
What is it about? software

Initial Tickets: [('hardware', 319858)]

Your Complaint is Raised. Your Complaint Number is {self.comp_no}
After Command: [('hardware', 319858), ('software', 856652)]


Do you want to exit(y/n): y
Thank you for using SSNCE Helpdesk
Our Service Engineer will look into your Queries
```