

Name: Dheeraj Vemula

Python Assignment:1- TechShop

Creating classes

Customer class:

```
class Customers:
    def __init__(self, customer_id, first_name, last_name, email, phone, address):
        self.CustomerID = customer_id
        self.FirstName = first_name
        self.LastName = last_name
        self.Email = email
        self.Phone = phone
        self.Address = address
        self.Orders = [] # Assuming orders will be stored as a list

1 usage
    def calculate_total_orders(self):
        return len(self.Orders)

    def get_customer_details(self):
        print("Customer ID:", self.CustomerID)
        print("Name:", self.FirstName, self.LastName)
        print("Email:", self.Email)
        print("Phone:", self.Phone)
        print("Address:", self.Address)
        print("Total Orders:", self.calculate_total_orders())

    def update_customer_info(self, new_email=None, new_phone=None, new_address=None):
        if new_email:
            self.Email = new_email
        if new_phone:
            self.Phone = new_phone
        if new_address:
            self.Address = new_address
        print("Customer information updated successfully.")
```

Product class:

```

class Products:
    def __init__(self, product_id, product_name, description, price):
        self.ProductID = product_id
        self.ProductName = product_name
        self.Description = description
        self.Price = price

    def get_product_details(self):
        print("Product ID:", self.ProductID)
        print("Product Name:", self.ProductName)
        print("Description:", self.Description)
        print("Price: $%.2f" % self.Price)

    def update_product_info(self, new_price=None, new_description=None):
        if new_price is not None:
            self.Price = new_price
        if new_description:
            self.Description = new_description
        print("Product information updated successfully.")

```

Order class:

```

class Orders:
    def __init__(self, order_id, customer, order_date, total_amount):
        self.OrderID = order_id
        self.Customer = customer
        self.OrderDate = order_date
        self.TotalAmount = total_amount
        self.OrderStatus = "Pending" # Default status
        self.OrderDetails = [] # Assuming order details will be stored as a list

1 usage
    def calculate_total_amount(self):
        return sum(detail.calculate_subtotal() for detail in self.OrderDetails)

```

```

def get_order_details(self):
    print("Order ID:", self.OrderID)
    print("Customer:", self.Customer.FirstName, self.Customer.LastName)
    print("Order Date:", self.OrderDate.strftime("%Y-%m-%d %H:%M:%S"))
    print("Total Amount: $%.2f" % self.calculate_total_amount())
    print("Order Status:", self.OrderStatus)

1 usage
def update_order_status(self, new_status):
    self.OrderStatus = new_status
    print("Order status updated successfully.")

def cancel_order(self):
    self.update_order_status("Cancelled")
    print("Order cancelled.")

```

Orderdetails class:

```

class OrderDetails:
    def __init__(self, order_detail_id, order, product, quantity):
        self.OrderDetailID = order_detail_id
        self.Order = order
        self.Product = product
        self.Quantity = quantity
        self.Discount = 0.0

3 usages (2 dynamic)
def calculate_subtotal(self):
    return (self.Product.Price - self.Discount) * self.Quantity

def get_order_detail_info(self):
    print("Order Detail ID:", self.OrderDetailID)
    print("Product:", self.Product.ProductName)
    print("Quantity:", self.Quantity)
    print("Subtotal: $%.2f" % self.calculate_subtotal())

def update_quantity(self, new_quantity):
    self.Quantity = new_quantity
    print("Quantity updated successfully.")

def add_discount(self, discount_amount):
    self.Discount += discount_amount
    print("Discount applied successfully.")

```

Inventory class:

```
class Inventory:
    def __init__(self, inventory_id, product, quantity_in_stock, last_stock_update):
        self.InventoryID = inventory_id
        self.Product = product
        self.QuantityInStock = quantity_in_stock
        self.LastStockUpdate = last_stock_update

    def get_product(self):
        return self.Product

    def get_quantity_in_stock(self):
        return self.QuantityInStock

    def add_to_inventory(self, quantity):
        self.QuantityInStock += quantity
        self.LastStockUpdate = datetime.now()
        print(quantity, self.Product.ProductName + "(s) added to the inventory.")

    def remove_from_inventory(self, quantity):
        if self.QuantityInStock >= quantity:
            self.QuantityInStock -= quantity
            self.LastStockUpdate = datetime.now()
            print(quantity, self.Product.ProductName + "(s) removed from the inventory.")
        else:
            print("Insufficient quantity in stock.")
```

```
    def update_stock_quantity(self, new_quantity):
        self.QuantityInStock = new_quantity
        self.LastStockUpdate = datetime.now()
        print("Stock quantity updated successfully.")

    def is_product_available(self, quantity_to_check):
        return self.QuantityInStock >= quantity_to_check

    def get_inventory_value(self):
        return self.Product.Price * self.QuantityInStock

    def list_low_stock_products(self, threshold):
        if self.QuantityInStock < threshold:
            print(self.Product.ProductName + " is low in stock with", self.QuantityInStock, "units.")

    def list_out_of_stock_products(self):
        if self.QuantityInStock == 0:
            print(self.Product.ProductName + " is out of stock.")

    def list_all_products(self):
        print("Product:", self.Product.ProductName + ",", "Quantity:", self.QuantityInStock)
```

Managing Product list.

Maintaining a list of products:

```
def get_product_details(self, product_id):
    query = """
        SELECT *
        FROM Products
        WHERE ProductID = %s;
    """
    cursor.execute(query, params: (product_id,))
    result = cursor.fetchone()
    return result
```

Updating product info:

```
def update_product_info(self, product_id, price, description):
    query = """
        UPDATE Products
        SET Price = %s, Description = %s
        WHERE ProductID = %s;
    """
    cursor.execute(query, params: (price, description, product_id))
    conn.commit()
```

Adding products:

```
def add_product(self, product):
    try:
        # Check for duplicate products based on name or SKU
        if any(p['ProductName'].lower() == product['ProductName'].lower() or
              p['SKU'].lower() == product['SKU'].lower() for p in self.products):
            raise DuplicateProductException()

        self.products.append(product)
        print("Product added successfully.")
    except DuplicateProductException as e:
        print(f"Error: {e}")
```

Searching products:

```

def search_products(self, search_criteria):
    try:
        query = """
            SELECT *
            FROM Products
            WHERE Category LIKE %s ;
        """
        cursor.execute(query, params: (f"%{search_criteria}%",))
        result = cursor.fetchall()
        return result
    except Exception as e:
        print(f"Error: {e}")

```

Managing order list.

Get order details:

```

def get_order_details(self):
    query = """
        SELECT *
        FROM OrderDetails
        WHERE OrderID = %s;
    """
    cursor.execute(query, params: (self.OrderID,))
    result = cursor.fetchall()
    return result

```

Order status updation:

```

def update_order_status(self, new_status):
    try:
        self._validate_order_status(new_status)
        query = """
            UPDATE Orders
            SET Status = %s
            WHERE OrderID = %s;
        """
        cursor.execute(query, params: (new_status, self.OrderID))
        conn.commit()
        print(f"Order {self.OrderID} status updated to {new_status}.")
    except InvalidDataException as e:
        print(f"Error: {e}")

```

Order cancellation:

```
def cancel_order(self):
    query = """
        DELETE FROM Orders
        WHERE OrderID = %s;
    """
    cursor.execute(query, params: (self.OrderID,))
    conn.commit()
    print(f"Order {self.OrderID} canceled.")
```

Adding new orders:

```
def add_new_order(self, product_id, quantity):
    try:
        # Check if the product is available in inventory
        if not self._is_product_available(product_id, quantity):
            raise InvalidDataException("Product not available in sufficient quantity.")

        # Insert new order
        oi=self.CustomerID
        order_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        query_insert_order = """
            INSERT INTO Orders (orderid, CustomerID, OrderDate, TotalAmount)
            VALUES (%s,%s, %s, %s);
        """
        cursor.execute(query_insert_order, params: (oi,self.CustomerID, order_date, 0))
        conn.commit()

        # Get the newly created order ID
        query_get_new_order_id = "SELECT LAST_INSERT_ID();"
        cursor.execute(query_get_new_order_id)
        new_order_id = cursor.fetchone()[0]

        # Insert order details
        query_insert_order_details = """
            INSERT INTO OrderDetails (OrderID, ProductID, Quantity)
            VALUES (%s, %s, %s);
        """
        cursor.execute(query_insert_order_details, params: (new_order_id, product_id, quantity))
        conn.commit()

        print(f"New order {new_order_id} added successfully.")
```

Handling inventory updates

Checking stock quantity:

```
def get_quantity_in_stock(self, product_id):  
    query = """  
        SELECT QuantityInStock  
        FROM Inventory  
        WHERE ProductID = %s;  
    """  
    cursor.execute(query, params: (product_id,))  
    result = cursor.fetchone()  
    return result[0] if result else None
```

Removing stock from inventory:

```
def remove_from_inventory(self, product_id, quantity):  
    query = """  
        UPDATE Inventory  
        SET QuantityInStock = QuantityInStock - %s  
        WHERE ProductID = %s;  
    """  
    cursor.execute(query, params: (quantity, product_id))  
    conn.commit()
```

Adding stock to inventory:

```
def add_to_inventory(self, product_id, quantity):  
    query = """  
        UPDATE Inventory  
        SET QuantityInStock = QuantityInStock + %s  
        WHERE ProductID = %s;  
    """  
    cursor.execute(query, params: (quantity, product_id))  
    conn.commit()
```

Database Connectivity:

Established the connection to techshop database.

```
import mysql.connector
from datetime import datetime

# Database Connection
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Kevink25*",
    database="techshop"
)
cursor = conn.cursor()
```

Main app:

Customer Registration:

```
def register_customer(customerid, first_name, last_name, email, phone):
    try:
        # Check for duplicate email
        cursor.execute(operation="SELECT * FROM Customers WHERE Email = %s", params=(email,))
        existing_customer = cursor.fetchone()

        if existing_customer:
            print("Error: Duplicate email address.")
            return

        # Insert new customer
        query = "INSERT INTO Customers (customerid, FirstName, LastName, Email, Phone) VALUES (%s,%s, %s, %s, %s)"
        cursor.execute(query, params=(customerid, first_name, last_name, email, phone))
        conn.commit()
        print("Customer registered successfully.")
    except Exception as e:
        print(f"Error: {e}")
```

Product Catalog Management:

```
def update_product(product_id, new_price, new_description):
    try:
        # Update product information
        query = "UPDATE Products SET Price = %s, Description = %s WHERE ProductID = %s"
        cursor.execute(query, params=(new_price, new_description, product_id))
        conn.commit()
        print("Product information updated successfully.")
    except Exception as e:
        print(f"Error: {e}")
```

Placing Customer Orders:

```
def place_order(orderid, customer_id, product_id, quantity):
    try:
        # Check product availability in inventory
        cursor.execute(operation="SELECT QuantityInStock FROM Inventory WHERE ProductID = %s", params=(product_id,))
        available_quantity = cursor.fetchone()[0]

        if available_quantity < quantity:
            print("Error: Insufficient stock.")
            return

        # Insert new order
        order_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        query_insert_order = "INSERT INTO Orders (orderid, CustomerID, OrderDate, TotalAmount) VALUES (%s, %s, %s, %s)"
        cursor.execute(query_insert_order, params=(orderid, customer_id, order_date, 0))
        conn.commit()

        # Get the newly created order ID
        query_get_new_order_id = "SELECT LAST_INSERT_ID();"
        cursor.execute(query_get_new_order_id)
        new_order_id = cursor.fetchone()[0]

        print("Order placed successfully.")
    except Exception as e:
        print(f"Error: {e}")
```

Tracking Order Status:

```
def track_order_status(order_id):
    try:
        # Retrieve order status
        query = "SELECT Status FROM Orders WHERE OrderID = %s"
        cursor.execute(query, params=(order_id,))
        status = cursor.fetchone()

        if status:
            print(f"Order {order_id} status: {status[0]}")
        else:
            print("Error: Order not found.")
    except Exception as e:
        print(f"Error: {e}")
```

Inventory Management:

```
def add_new_product(productid, product_name, description, price, quantity_in_stock):
    try:
        # Insert new product
        query_insert_product = "INSERT INTO Products (productid, ProductName, Description, Price) VALUES (%s,%s, %s, %s)"
        cursor.execute(query_insert_product, params: (productid, product_name, description, price))
        conn.commit()

        # Get the newly created product ID
        query_get_new_product_id = "SELECT LAST_INSERT_ID();"
        cursor.execute(query_get_new_product_id)
        new_product_id = cursor.fetchone()[0]
        '''
        # Add product to inventory
        inventoryid=productid
        query_add_to_inventory = "INSERT INTO Inventory (inventoryid, ProductID, QuantityInStock) VALUES (%s,%s, %s)"
        cursor.execute(query_add_to_inventory, (inventoryid, new_product_id, quantity_in_stock))
        conn.commit()
        '''

        print("New product added to inventory successfully.")
    except Exception as e:
        print(f"Error: {e}")
```

Sales Reporting:

```
def generate_sales_report(start_date, end_date):
    try:
        # Retrieve sales data based on specified criteria
        query = """
            SELECT Orders.OrderID, Customers.FirstName, Customers.LastName, Orders.OrderDate, Orders.TotalAmount
            FROM Orders
            JOIN Customers ON Orders.CustomerID = Customers.CustomerID
            WHERE Orders.OrderDate BETWEEN %s AND %s
            """
        cursor.execute(query, params: (start_date, end_date))
        sales_data = cursor.fetchall()

        if sales_data:
            print("Sales Report:")
            for row in sales_data:
                print(f"OrderID: {row[0]}, Customer: {row[1]} {row[2]}, OrderDate: {row[3]}, TotalAmount: {row[4]}")
        else:
            print("No sales data found for the specified period.")
    except Exception as e:
        print(f"Error: {e}")
```

Customer Account Updates:

```
def update_customer_account(customer_id, new_email, new_phone):
    try:
        # Update customer account details
        query = "UPDATE Customers SET Email = %s, Phone = %s WHERE CustomerID = %s"
        cursor.execute(query, params: (new_email, new_phone, customer_id))
        conn.commit()
        print("Customer account updated successfully.")
    except Exception as e:
        print(f"Error: {e}")
```

Product Search and Recommendations:

```
def search_products(search_criteria):
    try:
        # Search for products based on criteria
        query = "SELECT * FROM Products WHERE ProductName LIKE %s OR Description LIKE %s"
        cursor.execute(query, params: (f"%{search_criteria}%", f"%{search_criteria}%"))
        search_result = cursor.fetchall()

        if search_result:
            print("Search Results:")
            for product in search_result:
                print(product)
        else:
            print("No products found for the specified criteria.")
    except Exception as e:
        print(f"Error: {e}")
```