TechShop, an electronic gadgets shop

Implement OOPs

Task 1: Classes and Their Attributes:

You are working as a software developer for TechShop, a company that sells electronic gadgets. Your task is to design and implement an application using Object-Oriented Programming (OOP) principles to manage customer information, product details, and orders. Below are the classes you need to create:

Customers Class:

Attributes:

- CustomerID (int)
- FirstName (string)
- LastName (string)
- Email (string)
- Phone (string)
- Address (string)

```
class customers:
    def __init__(self , customerID : int ,
                 firstname : str ,
                 lastname : str ,
                 email : str,
                 phone : str,
                 address : str):
        self.customerID = customerID
        self.firstname = firstname
        self.lastname = lastname
        self.email = email
        self.phone = phone
        self.address = address
        self.totalorders = 0
```

Methods:

- CalculateTotalOrders(): Calculates the total number of orders placed by this customer.
- GetCustomerDetails(): Retrieves and displays detailed information about the customer.
- UpdateCustomerInfo(): Allows the customer to update their information (e.g., email, phone, or address).

```
    def CalculateTotalOrders(self):
        return self.totalorders
    def GetCustomerDetails(self):
        print("customerID  : " + str(self.customerID))
        print("firstname   : " + self.firstname)
        print("lastname    : " + self.lastname)
        print("email       : " + self.email)
        print("phone       : " + self.phone)
        print("address     : " + self.address)
    def UpdateCustomerInfo(self , new_email , new_phone , new_add ):
        self.email = new_email
        self.phone = new_phone
        self.address = new_add
```

Products Class:

Attributes:

- ProductID (int)
- ProductName (string)
- Description (string)
- Price (decimal)

```
class products:
    def __init__(self , productID : int ,
                 productname : str ,
                 description : str ,
                 price : float):
        self.productID = productID
        self.productname = productname
        self.description = description
        self.price = price
```

Methods:

- GetProductDetails(): Retrieves and displays detailed information about the product.
- UpdateProductInfo(): Allows updates to product details (e.g., price, description).
- IsProductInStock(): Checks if the product is currently in stock.

```
    def GetProductDetail(self):
        print("product id    = " + str(self.productID))
        print("product name  = " + self.productname)
        print("description   = " + self.description)
        print("price         = " + str(self.price))

    def UpdateProductInfo(self , newprice , newdesc):
        self.price = newprice
        self.description = newdesc

    def IsProductInStock(self):
        pass
```

Orders Class:

Attributes:

- OrderID (int)
- Customer (Customer) - Use composition to reference the Customer who placed the order.
- OrderDate (DateTime)
- TotalAmount (decimal)

```
import datetime

from customers import c1
from customers import customers

3 usages
class orders(customers) :
    def __init__(self , orderID : int , customer , orderdate , totalAmount : float):
        self.orderID = orderID
        super().__init__(customer.customerID , customer.firstname , customer.lastname , customer.email , customer.phone , customer.address )
        #self.customer = customer
        self.orderdate = datetime.datetime.strptime(orderdate, _format: "%Y-%m-%d").date()
        self.totalAmount = totalAmount
        self.status = 'processing'
```

Methods:

- CalculateTotalAmount() - Calculate the total amount of the order.
- GetOrderDetails(): Retrieves and displays the details of the order (e.g., product list and quantities).
- UpdateOrderStatus(): Allows updating the status of the order (e.g., processing, shipped).

- CancelOrder(): Cancels the order and adjusts stock levels for products.

```
    def calculateTotalAmount(self):
        print(f'total amount = {self.totalAmount}')
    1 usage
    def getOrderDetails(self):
        print(f'order id = {self.orderID}')
        print(f'customer id = {self.customerID} ')
        print(f'first name = {self.firstname} ')
        print(f'lastname = {self.lastname} ')
        print(f'order date = {self.orderdate} ')
        print(f'total amount = {self.totalAmount} ')

    1 usage
    def updateOrderStatus(self,status):
        self.status = status
        print(f'current status = {self.status}')

    def cancelOrder(self,orderID):
        pass
```

OrderDetails Class:

Attributes:

- OrderDetailID (int)
- Order (Order) - Use composition to reference the Order to which this detail belongs.
- Product (Product) - Use composition to reference the Product included in the order detail.
- Quantity (int)

```python
from orders import orders
from products import products

1 usage
class orderDeatail(orders , products):
    def __init__(self , orderdetailID : int , orders ,  products , quantity : int):
        self.orderdetailID = orderdetailID
        self.orderID = orders
        self.productID = products
        self.product = products
        self.quantity = quantity
```

Methods:

- CalculateSubtotal() - Calculate the subtotal for this order detail.
- GetOrderDetailInfo(): Retrieves and displays information about this order detail.
- UpdateQuantity(): Allows updating the quantity of the product in this order detail.
- AddDiscount(): Applies a discount to this order detail.

```python
    def calculateSubtotal(self):
        totalAmount = self.order.totalAmount
        return totalAmount

    def getOrderDetailInfo(self):
        print(f'order deatil ID = {self.orderdetailID}')
        print(f'order id  = {self.order.orderID}')
        print(f'product id = {self.product.productID}')
        print(f'quantity = {self.quantity}')

    def UpdateQuantity(self, newQuantity):
            if newQuantity >= 0:
                self.quantity = newQuantity
            else:
                raise ValueError("Quantity cannot be negative")

    def addDiscount(self):
        pass
```

Inventory class:

Attributes:

- InventoryID(int)

- Product (Composition): The product associated with the inventory item.
- QuantityInStock: The quantity of the product currently in stock.
- LastStockUpdate

```python
from products import products
from products import p1


1 usage
class inventory(products):
    def __init__(self, inventoryID: int, product, quantityInStock: int, lastStockUpdate: int):
        self.inventoryID = inventoryID
        super().__init__(product.productID , product.productname ,product.description,product.price)
        self.quantityInStock = quantityInStock
        self.lastStockUpdate = lastStockUpdate
```

Methods:

- GetProduct(): A method to retrieve the product associated with this inventory item.
- GetQuantityInStock(): A method to get the current quantity of the product in stock.
- AddToInventory(int quantity): A method to add a specified quantity of the product to the inventory.
- RemoveFromInventory(int quantity): A method to remove a specified quantity of the product from the inventory.
- UpdateStockQuantity(int newQuantity): A method to update the stock quantity to a new value.
- IsProductAvailable(int quantityToCheck): A method to check if a specified quantity of the product is available in the inventory.
- GetInventoryValue(): A method to calculate the total value of the products in the inventory based on their prices and quantities.
- ListLowStockProducts(int threshold): A method to list products with quantities below a specified threshold, indicating low stock.
- ListOutOfStockProducts(): A method to list products that are out of stock.

```
1 usage
def getProduct(self):
    print(f"Inventory Id = {self.inventoryID}")
    print(f"Product Id = {self.productID}")
    print(f"Product Name = {self.productname}")
    print(f"Product Description = {self.description}")
    print(f"Product Price = {self.price}")

def getQuantityInStock(self):
    return self.quantityInStock

def addToInventory(self, quantity):
    self.quantityInStock += quantity

def removeFromInventory(self, quantity):
        if quantity <= self.quantityInStock:
            self.quantityInStock -= quantity
            self.lastStockUpdate = self.quantityInStock
        else:
            raise ValueError("cannot remove this quantity")
```

```
def isProductAvailable(self, quantity):
    if quantity >= self.quantityInStock:
        print("yes the product is available")
    else:
        print("the product is not available")

1 usage
def getInventoryValue(self):
    return self.quantityInStock * self.price

def listLowStockProducts(self, minStock):
    if self.quantityInStock < minStock:
        print(f" {self.productName} is low in quantity")
    else:
        print("stock available")

def listOutOfStockProducts(self):
    pass

def listAllProducts(self):
    pass
```

- ListAllProducts(): A method to list all products in the inventory, along with their quantities.

Task 5: Exceptions handling

- **Data Validation**:
  - ```
    def checkEmail(self , email):
        pat = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b'
        if re.match(pat, email):
            print("Valid Email")
        else:
            raise Exception("Invalid Email")
    ```
- **Inventory Management**:
  - ```
    def updateStockQuantity(self, newquantity):
            if newquantity >= 0:
                self.quantityInStock += newquantity
                self.lastStockUpdate = self.quantityInStock
            else:
                raise ValueError("Quanitity cannot be negative.")
    ```
- **Order Processing:**
  - Challenge: Ensuring the order details are consistent and complete before processing.
  - Scenario: When an order detail lacks a product reference.
  - Exception Handling: Throw an IncompleteOrderException with a message explaining the
  - issue.
  - ```
    if not orders:
        raise Exception("No orders found for the customer.")
        return
    ```

Task 7: Database Connectivity

- Implement a DatabaseConnector class responsible for establishing a connection to the "TechShopDB" database. This class should include methods for opening, closing, and managing database connections.
- Implement classes for Customers, Products, Orders, OrderDetails, Inventory with properties, constructors, and methods for CRUD (Create, Read, Update, Delete) operations.
    ```

```python
import mysql.connector
import re

con = mysql.connector.connect(
    host="localhost",
    user="root",
    password="root",
    port="3306",
    database="TechShopDB"
)

cur = con.cursor()
cur.execute("use TechShopDB")

```

1: Customer Registration

Description: When a new customer registers on the TechShop website, their information (e.g., name, email, phone) needs to be stored in the database.

Task: Implement a registration form and database connectivity to insert new customer records. Ensure proper data validation and error handling for duplicate email addresses.

```python
1 usage
def customer_registration():
    id = int(input("Enter customer id: "))
    firstname = input("Enter first name: ")
    lastname = input("Enter last name: ")
    email = input("Enter email: ")
    pat = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b'
    if re.match(pat, email):
        print("Valid Email")
    else:
        raise Exception("Invalid Email")
    phone = input("Enter phone: ")
    address = input("Enter address: ")
    str1 = "insert into Customers values(%s,%s,%s,%s,%s,%s)"
    cur.execute(str1, params: (id, firstname, lastname, email, phone, address))
    con.commit()
    print("User registered successfully")
```

2: Product Catalog Management

Description: TechShop regularly updates its product catalog with new items and changes in product details (e.g., price, description). These changes need to be reflected in the database.

Task: Create an interface to manage the product catalog. Implement database connectivity to update product information. Handle changes in product details and ensure data consistency.

```
1 usage
def update_product():
    print("Select 1 for product name update\n"
          "Select 2 for product description update\n"
          "Select 3 for price update\n")

    select = int(input("Enter choice: "))

    if select not in [1, 2, 3]:
        print("Invalid choice")
        return

    prodid = int(input("Enter product id: "))

    if select == 1:
        prodname = input("Enter new product name: ")
        query = "UPDATE products SET productname = %s WHERE productID = %s"
        cur.execute(query, params: (prodname, prodid))

    elif select == 2:
        desc = input("Enter new product description: ")
        query = "UPDATE products SET description = %s WHERE productID = %s"
        cur.execute(query, params: (desc, prodid))

    elif select == 3:
        price = float(input("Enter new product price: "))
        query = "UPDATE products SET price = %s WHERE productID = %s"
        cur.execute(query, params: (price, prodid))

    con.commit()
```

3: Tracking Order Status

Description: Customers and employees need to track the status of their orders. The order status information is stored in the database.

Task: Develop a feature that allows users to view the status of their orders. Implement database connectivity to retrieve and display order status information.

```python
def order_tracking():
    id = int(input("Enter customer id: "))

    # Use a parameterized query to avoid SQL injection
    query = "SELECT orderID FROM orders WHERE customerID = %s AND orderdate > '2024-01-15'"
    cur.execute(query, params: (id,))

    orders = cur.fetchall()

    if not orders:
        raise Exception("No orders found for the customer.")
        return

    for order in orders:
        order_id = order[0]
        print(f"Order ID: {order_id}, Status: {'shipped' if order[1] > '2024-01-15' else 'pending'}")


1 usage
```

4: Inventory Management

Description: TechShop needs to manage product inventory, including adding new products, updating stock levels, and removing discontinued items.

Task: Create an inventory management system with database connectivity. Implement features for adding new products, updating quantities, and handling discontinued products.

```python
1 usage
def inventory_management():
    print("1. Add New Product")
    print("2. Update Product Quantity")
    print("3. Discontinue Product")
    choice = int(input("Enter your choice: "))

    if choice == 1:
        add_new_product()
    elif choice == 2:
        update_product_quantity()
    elif choice == 3:
        discontinue_product()
    else:
        print("Invalid choice")

```

```
1 usage
def add_new_product():
    name = input("Enter product name: ")
    description = input("Enter product description: ")
    price = float(input("Enter product price: "))
    quantity = int(input("Enter product quantity: "))

    # Insert new product into the Products table
    query = "INSERT INTO Products (ProductName, Description, Price) VALUES (%s, %s, %s)"
    cur.execute(query, params: (name, description, price))
    con.commit()

    # Add the product to the Inventory
    product_id = cur.lastrowid
    query_inventory = "INSERT INTO Inventory (ProductID, QuantityInStock) VALUES (%s, %s)"
    cur.execute(query_inventory, params: (product_id, quantity))
    con.commit()

    print("New product added successfully.")
```

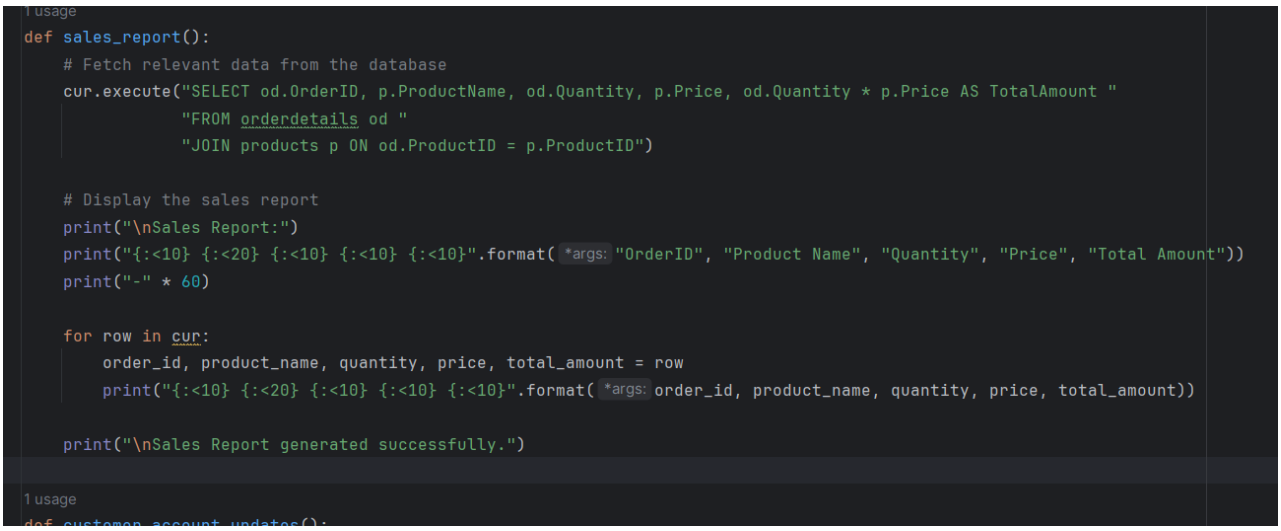
```
1 usage
def update_product_quantity():
    product_id = int(input("Enter product ID: "))
    new_quantity = int(input("Enter new product quantity: "))

    # Update product quantity in the Inventory
    query = "UPDATE Inventory SET QuantityInStock = %s WHERE ProductID = %s"
    cur.execute(query, params: (new_quantity, product_id))
    con.commit()

    print("Product quantity updated successfully.")
```

```
def discontinue_product():
    product_id = int(input("Enter product ID to discontinue: "))

    # Mark the product as discontinued in the Products table
    query_product = "UPDATE Products SET Discontinued = 1 WHERE ProductID = %s"
    cur.execute(query_product, params: (product_id,))
    con.commit()

    # Remove the product from the Inventory
    query_inventory = "DELETE FROM Inventory WHERE ProductID = %s"
    cur.execute(query_inventory, params: (product_id,))
    con.commit()

    print("Product discontinued successfully.")
```

5: Sales Reporting

Description: TechShop management requires sales reports for business analysis. The sales data is stored in the database.

Task: Design and implement a reporting system that retrieves sales data from the database and generates reports based on specified criteria.



```
1 usage
def sales_report():
    # Fetch relevant data from the database
    cur.execute("SELECT od.OrderID, p.ProductName, od.Quantity, p.Price, od.Quantity * p.Price AS TotalAmount "
                "FROM orderdetails od "
                "JOIN products p ON od.ProductID = p.ProductID")

    # Display the sales report
    print("\nSales Report:")
    print("{:<10} {:<20} {:<10} {:<10} {:<10}".format( *args: "OrderID", "Product Name", "Quantity", "Price", "Total Amount"))
    print("-" * 60)

    for row in cur:
        order_id, product_name, quantity, price, total_amount = row
        print("{:<10} {:<20} {:<10} {:<10} {:<10}".format( *args: order_id, product_name, quantity, price, total_amount))

    print("\nSales Report generated successfully.")

1 usage
def customer_account_updates():
```

7: Customer Account Updates

Description: Customers may need to update their account information, such as changing their email address or phone number.

Task: Implement a user profile management feature with database connectivity to allow customers to update their account details. Ensure data validation and integrity.

```python
def customer_account_updates():
    customer_id = int(input("Enter customer ID: "))
    print("Select the information to update:")
    print("1. Email")
    print("2. Phone")
    choice = int(input("Enter your choice: "))

    if choice not in [1, 2]:
        print("Invalid choice")
        return

    if choice == 1:
        new_email = input("Enter new email: ")
        validate_email(new_email)
        query = "UPDATE Customers SET Email = %s WHERE CustomerID = %s"
        cur.execute(query, params: (new_email, customer_id))
    elif choice == 2:
        new_phone = input("Enter new phone number: ")
        query = "UPDATE Customers SET Phone = %s WHERE CustomerID = %s"
        cur.execute(query, params: (new_phone, customer_id))

    con.commit()
    print("Customer information updated successfully.")
```

8: Payment Processing

Description: When customers make payments for their orders, the payment details (e.g., payment method, amount) must be recorded in the database.

Task: Develop a payment processing system that interacts with the database to record payment transactions, validate payment information, and handle errors.

```python
1 usage
def payment_processing():
    order_id = int(input("Enter order ID: "))
    payment_method = input("Enter payment method: ")
    amount = float(input("Enter payment amount: "))

    # Record payment details in the Payments table
    query_payment = "INSERT INTO Payments (OrderID, PaymentMethod, Amount) VALUES (%s, %s, %s)"
    cur.execute(query_payment, params: (order_id, payment_method, amount))
    con.commit()

    print("Payment processed successfully.")
```

9: Product Search and Recommendations

Description: Customers should be able to search for products based on various criteria (e.g., name, category) and receive product recommendations.

Task: Implement a product search and recommendation engine that uses database connectivity to retrieve relevant product information.

```
1 usage
def product_search_and_recommendations():
    print("Select the search criteria:")
    print("1. Search by product name")
    print("2. Search by category")
    choice = int(input("Enter your choice: "))

    if choice not in [1, 2]:
        print("Invalid choice")
        return

    if choice == 1:
        product_name = input("Enter product name to search: ")
        query = "SELECT * FROM Products WHERE ProductName LIKE %s"
        cur.execute(query, params: ('%' + product_name + '%',))
    elif choice == 2:
        category = input("Enter category to search: ")
        query = "SELECT * FROM Products WHERE Category = %s"
        cur.execute(query, params: (category,))

    products = cur.fetchall()

    if not products:
        print("No products found.")
    else:
        print("\nSearch Results:")
        for product in products:
            print(product)
```

Main method for controlling the database:

```python
import techshopIMPL


1 usage
class Main:
    1 usage
    def start(self):
        while True:
            print("Please select the choices from below:")
            print("1. Register new customer.")
            print("2. Change product information.")
            print("3. Tracking Order Status")
            print("4. Inventory Management")
            print("5. Sales Reporting")
            print("6. Customer Account Updates")
            print("7. Payment Processing")
            print("8. Product Search and Recommendations")
            print("0. Exit")
            choice = int(input("Enter your choice: "))

    match choice:
        case 1:
            techshopIMPL.customer_registration()
        case 2:
            techshopIMPL.update_product()
        case 3:
            techshopIMPL.order_tracking()
        case 4:
            techshopIMPL.inventory_management()
        case 5:
            techshopIMPL.sales_report()
        case 6:
            techshopIMPL.customer_account_updates()
        case 7:
            techshopIMPL.payment_processing()
        case 8:
            techshopIMPL.product_search_and_recommendations()
        case 0:
            break
```