Submitted by : Mridul Bhardwaj

Assignment -1

Tech 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)

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).

```python
import re
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
```

```python
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
```

```python
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..")
```

Products Class:

Attributes:

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

• Price (decimal)

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.

```python
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

    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)

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.

```
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'
    def calculateTotalAmount(self):
        print(f'total amount = {self.totalAmount}')
    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} ')
```

```
    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)

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.

```
import orders
import products

class orderDetail(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
```

```
    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
```

```
od1 = orderDetail( orderdetailID: 315 ,  orders: 215 , products: 115, quantity: 5 )
```

Inventory class:

Attributes:

• InventoryID (int)

• Product (Composition): The product associated with the inventory item.

• QuantityInStock: The quantity of the product currently in stock.

• 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.

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

```python
import datetime

import products
from products import p1


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
```

```
    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 updateStockQuantity(self, newquantity):
        if newquantity >= 0:
            self.quantityInStock += newquantity
            self.lastStockUpdate = self.quantityInStock
        else:
            raise ValueError("Quanitity cannot be negative.")


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

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

i1 = inventory( inventoryID: 401 , p1 , quantityInStock: 15 , lastStockUpdate: 5)
i1.getProduct()
print(i1.getInventoryValue())
```

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.")


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

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")
```

• Order Processing:

• Payment Processing:

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.

```
# Database Connection
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="9927559686",
    database="techshop"
)
cursor = conn.cursor()
```

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.

```
# Task 1: Customer Registration
def register_customer(customerid,first_name, last_name, email, phone):
    try:
        # Check for duplicate email
        cursor.execute("SELECT * FROM Customers WHERE Email = %s", (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,
        cursor.execute(query, (customerid,first_name, last_name, email, phone))
        conn.commit()
        print("Customer registered successfully.")
    except Exception as e:
        print(f"Error: {e}")
```

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.

```
# Task 2: 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, (new_price, new_description, product_id))
        conn.commit()
        print("Product information updated successfully.")
    except Exception as e:
        print(f"Error: {e}")
```

3: Placing Customer Orders

Description: Customers browse the product catalog and place orders for products they want to purchase. The orders need to be stored in the database.

Task: Implement an order processing system. Use database connectivity to record customer orders, update product quantities in inventory, and calculate order totals.

```
# Task 3: Placing Customer Orders
def place_order(orderid,customer_id, product_id, quantity):
    try:
        # Check product availability in inventory
        cursor.execute("SELECT QuantityInStock FROM Inventory WHERE ProductID = %s", (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 (
        cursor.execute(query_insert_order, (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]
```

```
    '''
    orderdetailid=orderid
    # Insert order details
    query_insert_order_details = "INSERT INTO OrderDetails
      (orderdetailid,OrderID, ProductID, Quantity) VALUES (%s,%s, %s, %s)"
    cursor.execute(query_insert_order_details,
    (orderdetailid,new_order_id, product_id, quantity))
    conn.commit()

    # Update inventory quantity
    query_update_inventory = "UPDATE Inventory SET QuantityInStock = QuantityInStock - %s WHERE P
    cursor.execute(query_update_inventory, (quantity, product_id))
    conn.commit()
    '''

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

4: 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.

```
# Task 4: Tracking Order Status
def track_order_status(order_id):
    try:
        # Retrieve order status
        query = "SELECT Status FROM Orders WHERE OrderID = %s"
        cursor.execute(query, (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}")
```

5: 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
# Task 5: 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) VALU
        cursor.execute(query_insert_product, (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) VALU
        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}")
```

6: 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.

```python
# Task 6: 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.
            FROM Orders
            JOIN Customers ON Orders.CustomerID = Customers.CustomerID
            WHERE Orders.OrderDate BETWEEN %s AND %s
        """
        cursor.execute(query, (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]}, TotalAmo
        else:
            print("No sales data found for the specified period.")
    except Exception as e:
        print(f"Error: {e}")
```

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
# Task 7: 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, (new_email, new_phone, customer_id))
        conn.commit()
        print("Customer account updated successfully.")
    except Exception as e:
        print(f"Error: {e}")
```

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.

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.

```python
# Task 9: 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, (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}")
```