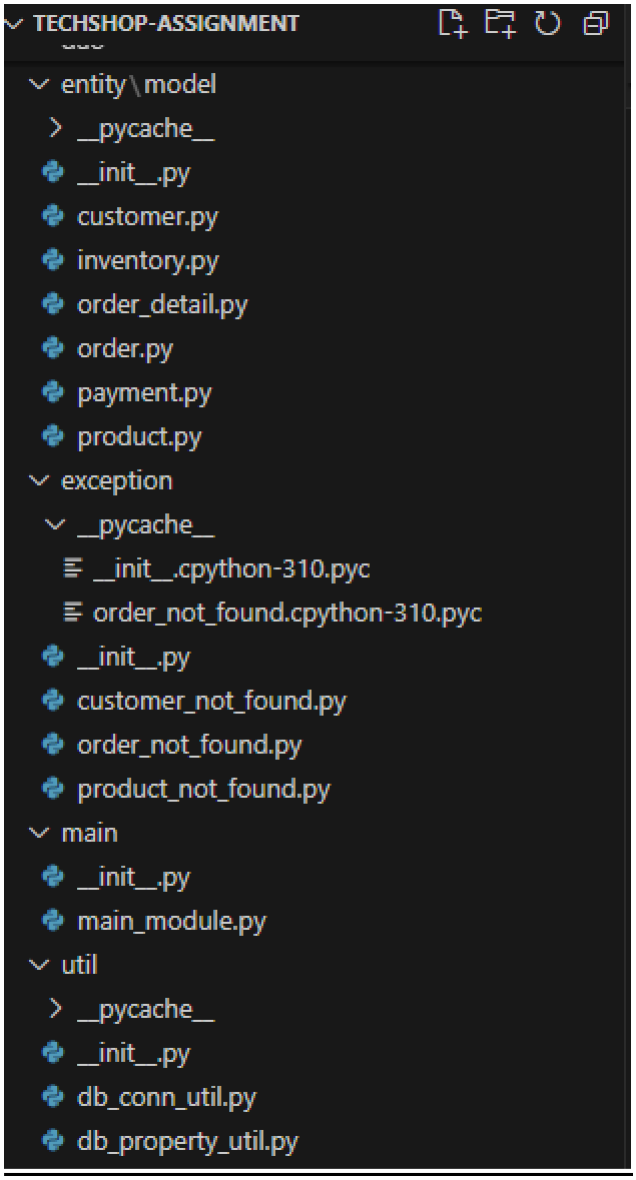
Assignment : Techshop

Name: Seemalamudi Nikhil Sai -nikhilsai16802@gmail.com

Problem Statement:

1. 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:

→



2. Task 2: Class Creation:

- **Create the classes (Customers, Products, Orders, OrderDetails and Inventory) with the specified attributes.**
- **Implement the constructor for each class to initialize its attributes.**
- **Implement methods as specified.**

→ Customers Class:

```
class Customer:
    def __init__(self, first_name, last_name, email, phone, address):
        self.__first_name = first_name
        self.__last_name = last_name
        self.__email = email
        self.__phone = phone
        self.__address = address

    @property
    def first_name(self):
        return self.__first_name

    @first_name.setter
    def first_name(self, value):
        self.__first_name = value

    @property
    def last_name(self):
        return self.__last_name

    @last_name.setter
```

```
    def last_name(self, value):
        self.__last_name = value

    @property
    def email(self):
        return self._email

    @property
    def phone(self):
        return self._phone

    @property
    def address(self):
        return self._address
```

Product Class:

```
class Product:
    def __init__(self, product_name, product_description, product_price, product_quantity, category):
        self.__product_id = None  # Will be set when inserting into the database
        self.__product_name = product_name
        self.__description = product_description
        self.__price = product_price
        self.__stock_quantity = product_quantity
        self.__category = category  # New attribute

    @property
```

```
def product_id(self):
    return self.__product_id

@property
def product_name(self):
    return self.__product_name

@property
def description(self):
    return self.__description

@property
def price(self):
    return self.__price

@property
def stock_quantity(self):
    return self.__stock_quantity

@stock_quantity.setter
def stock_quantity(self, value):
    if value < 0:
        raise ValueError("Stock quantity cannot be negative.")
    self.__stock_quantity = value

@property
def category(self):
```

```
        return self.__category
```

Orders Class:

```
from datetime import datetime

class Order:
    def __init__(self, customer_id, order_date, total_amount):
        self.__order_id = None  # Will be set when inserting into the database
        self.__customer_id = customer_id
        self.__order_date = order_date  # Updated to accept order date
        self.__total_amount = total_amount

    @property
    def order_id(self):
        return self.__order_id

    @property
    def customer_id(self):
        return self.__customer_id

    @property
    def order_date(self):
        return self.__order_date

    @property
    def total_amount(self):
        return self.__total_amount
```

OrderDetails Class:

```
# entity/order_detail.py

class OrderDetail:
    def __init__(self, order_detail_id, product_id, quantity):
        self.__order_detail_id = order_detail_id
        self.__product_id = product_id
        self.__quantity = quantity

    @property
    def order_detail_id(self):
        return self.__order_detail_id

    @property
    def product_id(self):
        return self.__product_id

    @property
    def quantity(self):
        return self.__quantity

    def calculate_subtotal(self, product_price):
        return product_price * self.__quantity

    def get_order_detail_info(self):
```

```
        return f"Order Detail ID: {self.order_detail_id}, Product ID: {self.product_id}, Quantity: {self.quantity}"
```

Inventory class:

```
class Inventory:
    def __init__(self, inventory_id, product, quantity_in_stock):
        self.__inventory_id = inventory_id
        self.__product = product
        self.__quantity_in_stock = quantity_in_stock

    @property
    def inventory_id(self):
        return self.__inventory_id

    @property
    def product(self):
        return self.__product

    @property
    def quantity_in_stock(self):
        return self.__quantity_in_stock
```

---

3. Task 3: Encapsulation:

- **Implement encapsulation by making the attributes private and providing public properties (getters and setters) for each attribute.**
- **Add data validation logic to setter methods (e.g., ensure that prices are non-negative, quantities are positive integers).**

→

```
from abc import ABC, abstractmethod
```

```
class ServiceProvider(ABC):

    @abstractmethod
    def add_customer(self, customer):
        pass

    @abstractmethod
    def add_product(self, product):
        pass

    @abstractmethod
    def place_order(self, order, order_details):
        pass

    @abstractmethod
    def get_product_price(self, product_id):
        pass

    @abstractmethod
    def update_product_quantity(self, product_id, new_quantity):
        pass

    @abstractmethod
    def get_order_status(self, order_id):
        pass
```

```
@abstractmethod
def track_order_status(self, order_id):
    pass

@abstractmethod
def generate_sales_report(self, start_date, end_date):
    pass

@abstractmethod
def update_customer_info(self, customer_id, email=None, phone=None):
    pass

@abstractmethod
def process_payment(self, payment):
    pass

@abstractmethod
def search_products_by_category(self, category):
    pass

@abstractmethod
def delete_product(self, product_id):
    pass

@abstractmethod
def list_customers(self):  # New method
    pass
```

```
@abstractmethod
def get_order_total(self, customer_id, order_id):  # New method
    pass
```

---

4. Task 4: Composition: Ensure that the Order and OrderDetail classes correctly use composition to reference Customer and Product objects.

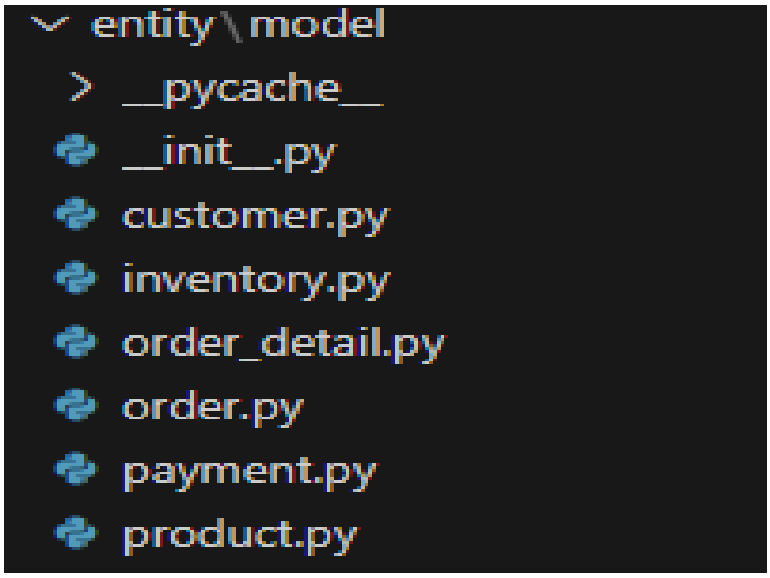
• Orders Class with Composition: o In the Orders class, we want to establish a composition relationship with the Customers class, indicating that each order is associated with a specific customer. o In the Orders class, we've added a private attribute customer of type Customers, establishing a composition relationship. The Customer property provides access to the Customers object associated with the order.

• OrderDetails Class with Composition: o Similarly, in the OrderDetails class, we want to establish composition relationships with both the Orders and Products classes to represent the details of each order, including the product being ordered. o In the OrderDetails class, we've added two private attributes, order and product, of types Orders and Products, respectively, establishing composition relationships. The Order property provides access to the Orders object associated with the order detail, and the Product property provides access to the Products object representing the product in the order detail.

• Customers and Products Classes: o The Customers and Products classes themselves may not have direct composition relationships with other classes in this scenario. However, they serve as the basis for composition relationships in the Orders and OrderDetails classes, respectively.

• Inventory Class: o The Inventory class represents the inventory of products available for sale. It can have composition relationships with the Products class to indicate which products are in the inventory.

→



---

5. Task 5: Exceptions handling

→

Customer_not_found.py:

```
class CustomerNotFound(Exception):
    def __init__(self, message="Customer not found"):
        self.message = message
        super().__init__(self.message)
```

order_not_found.py:

```
class OrderNotFound(Exception):
    def __init__(self, message="Order not found"):
        super().__init__(message)
```

Product_not_found.py:

```
class ProductNotFound(Exception):
    def __init__(self, message="Product not found"):
        super().__init__(message)
```

---

6. Task 6: Collections

→ service_provider_impl.py:

```
import pyodbc
from dao.service_provider import ServiceProvider
from entity.model.order import Order
from entity.model.order_detail import OrderDetail
from entity.model.payment import Payment
from exception.order_not_found import OrderNotFound

class ServiceProviderImpl(ServiceProvider):
    def __init__(self, connection_string):
        self.connection_string = connection_string

    def _get_connection(self):
        return pyodbc.connect(self.connection_string)

    def add_customer(self, customer):
        try:
            with self._get_connection() as conn:
                cursor = conn.cursor()
                cursor.execute("INSERT INTO Customers (FirstName, LastName, Email, Phone,
Address) VALUES (?, ?, ?, ?, ?)",
                               (customer.first_name, customer.last_name, customer.email, customer.phone,
customer.address))
                conn.commit()
                print("Customer added successfully.")
        except Exception as e:
            print(f"Error adding customer: {e}")
```

```
    def add_product(self, product):
        try:
            with self._get_connection() as conn:
                cursor = conn.cursor()
                cursor.execute("INSERT INTO Products (ProductName, Description, Price,
StockQuantity) VALUES (?, ?, ?, ?)",
                               (product.product_name, product.description, product.price,
product.stock_quantity))
                conn.commit()
                print("Product added successfully.")
        except Exception as e:
            print(f"Error adding product: {e}")


    def place_order(self, order: Order, order_details: list):
        try:
            with self._get_connection() as conn:
                cursor = conn.cursor()


                if not all([order.customer_id, order.order_date, order.total_amount]):
                    print("Order is missing required attributes.")
                    return


                cursor.execute("""
                    INSERT INTO Orders (CustomerID, OrderDate, TotalAmount)
                    VALUES (?, ?, ?)
                """, (order.customer_id, order.order_date, order.total_amount))


                order_id = cursor.execute("SELECT SCOPE_IDENTITY()").fetchone()[0]
```

```
            if order_id is None:
                print("")
                return

            print(f"Retrieved OrderID: {order_id}")

            for detail in order_details:
                if not detail.product_id or detail.quantity <= 0:
                    print(f"Invalid order detail: {detail}")
                    continue
                cursor.execute("""
                    INSERT INTO OrderDetails (OrderID, ProductID, Quantity)
                    VALUES (?, ?, ?)
                """, (order_id, detail.product_id, detail.quantity))

            conn.commit()
            print("Order placed successfully.")
        except Exception as e:
            print(f"Error placing order: {e}")

    def get_product_price(self, product_id):
        try:
            with self._get_connection() as conn:
                cursor = conn.cursor()
                cursor.execute("SELECT Price FROM Products WHERE ProductID = ?",
(product_id,))
```

```
            price = cursor.fetchone()
            return price[0] if price else None
        except Exception as e:
            print(f"Error fetching product price: {e}")
            return None

    def update_product_quantity(self, product_id, new_quantity):
        try:
            with self._get_connection() as conn:
                cursor = conn.cursor()
                cursor.execute("UPDATE Products SET StockQuantity = ? WHERE ProductID = ?", (new_quantity, product_id))
                conn.commit()
                print("Product quantity updated successfully.")
        except Exception as e:
            print(f"Error updating product quantity: {e}")

    def get_order_status(self, order_id):
        try:
            with self._get_connection() as conn:
                cursor = conn.cursor()
                cursor.execute("SELECT Status FROM Orders WHERE OrderID = ?", (order_id,))
                status = cursor.fetchone()
                return status[0] if status else None
        except Exception as e:
            print(f"Error fetching order status: {e}")
            return None
```

```
def track_order_status(self, order_id):
    try:
        with self._get_connection() as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT Status FROM Orders WHERE OrderID = ?", (order_id,))
            status = cursor.fetchone()
            if status:
                print(f"Order Status for Order ID {order_id}: {status[0]}")
            else:
                print(f"No order found with Order ID {order_id}.")
    except Exception as e:
        print(f"Error tracking order status: {e}")

def generate_sales_report(self, start_date, end_date):
    try:
        with self._get_connection() as conn:
            cursor = conn.cursor()
            query = """
                SELECT
                    Orders.OrderID,
                    Orders.OrderDate,
                    SUM(OrderDetails.Quantity * Products.Price) AS TotalSales,
                    COUNT(Orders.OrderID) AS NumberOfOrders
                FROM
                    Orders
                JOIN
```

```
                OrderDetails ON Orders.OrderID = OrderDetails.OrderID
            JOIN
                Products ON OrderDetails.ProductID = Products.ProductID
            WHERE
                Orders.OrderDate BETWEEN ? AND ?
            GROUP BY
                Orders.OrderID, Orders.OrderDate
        """
        cursor.execute(query, (start_date, end_date))
        sales_data = cursor.fetchall()

        if sales_data:
            print("Sales Report:")
            print(f"{'Order ID':<10} {'Order Date':<20} {'Total Sales':<15} {'Number of Orders':<15}")
            for row in sales_data:
                print(f"{row.OrderID:<10} {row.OrderDate:<20} {row.TotalSales:<15} {row.NumberOfOrders:<15}")
        else:
            print("No sales data found for the specified date range.")

    except Exception as e:
        print(f"Error generating sales report: {e}")

def update_customer_info(self, customer_id, email=None, phone=None):
    try:
        with self._get_connection() as conn:
            cursor = conn.cursor()
```

```
            query = "UPDATE Customers SET"
            params = []

            if email:
                query += " Email = ?"
                params.append(email)
            if phone:
                query += ", Phone = ?"
                params.append(phone)

            query += " WHERE CustomerID = ?"
            params.append(customer_id)

            cursor.execute(query, params)
            conn.commit()

            if cursor.rowcount > 0:
                print("Customer information updated successfully.")
            else:
                print("No customer found with the given ID.")
        except Exception as e:
            print(f"Error updating customer information: {e}")

    def process_payment(self, payment: Payment):
        try:
            with self._get_connection() as conn:
```

```
            cursor = conn.cursor()
            cursor.execute(
                "INSERT INTO Payments (OrderID, PaymentMethod, Amount, PaymentDate) "
                "VALUES (?, ?, ?, ?)",
                (payment.get_order_id(), payment.get_payment_method(), payment.get_amount(),
payment.get_payment_date())
            )
            conn.commit()
            print("Payment processed successfully.")
        except pyodbc.Error as e:
            raise Exception(f"Error processing payment: {e}")

    def search_products_by_category(self, category):
        try:
            with self._get_connection() as conn:
                cursor = conn.cursor()
                cursor.execute("SELECT * FROM Products WHERE Category = ?", (category,))
                products = cursor.fetchall()
                return products
        except Exception as e:
            print(f"Error searching products: {e}")
            return None

    def delete_product(self, product_id):
        try:
            with self._get_connection() as conn:
                cursor = conn.cursor()
```

```
            cursor.execute("DELETE FROM Products WHERE ProductID = ?", (product_id,))
            if cursor.rowcount == 0:
                raise Exception("Product not found.")
            conn.commit()
    except Exception as e:
        print(f"Error deleting product: {e}")
        raise

def list_customers(self):
    try:
        with self._get_connection() as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM Customers")
            customers = cursor.fetchall()
            return customers  # Return the list of customers
    except Exception as e:
        print(f"Error retrieving customers: {e}")
        return None

def get_order_total(self, customer_id, order_id):
    try:
        with self._get_connection() as conn:
            cursor = conn.cursor()
            cursor.execute("""
                SELECT TotalAmount FROM Orders
                WHERE CustomerID = ? AND OrderID = ?
            """, (customer_id, order_id))
```

```
            result = cursor.fetchone()
            if result:
                return result[0]  # TotalAmount
            else:
                raise OrderNotFound(f"No such order found for the given customer ID: {customer_id} and order ID: {order_id}.")
        except Exception as e:
            print(f"Error retrieving order total: {e}")
            raise
```

service_provider.py:

```
from abc import ABC, abstractmethod

class ServiceProvider(ABC):

    @abstractmethod
    def add_customer(self, customer):
        pass

    @abstractmethod
    def add_product(self, product):
        pass

    @abstractmethod
    def place_order(self, order, order_details):
        pass
```

```
@abstractmethod
def get_product_price(self, product_id):
    pass

@abstractmethod
def update_product_quantity(self, product_id, new_quantity):
    pass

@abstractmethod
def get_order_status(self, order_id):
    pass

@abstractmethod
def track_order_status(self, order_id):
    pass

@abstractmethod
def generate_sales_report(self, start_date, end_date):
    pass

@abstractmethod
def update_customer_info(self, customer_id, email=None, phone=None):
    pass

@abstractmethod
def process_payment(self, payment):
    pass
```

```
    @abstractmethod
    def search_products_by_category(self, category):
        pass

    @abstractmethod
    def delete_product(self, product_id):
        pass

    @abstractmethod
    def list_customers(self):  # New method
        pass

    @abstractmethod
    def get_order_total(self, customer_id, order_id):  # New method
        pass
```

---

7. Task 7: Database Connectivity

→

db_conn_util.py:

```
import pyodbc

class DBConnUtil:
    @staticmethod
    def get_connection(connection_string):
        return pyodbc.connect(connection_string)
```

db_property_util.py:

```
class DBPropertyUtil:
    @staticmethod
    def get_connection_string():
        return 'Driver={SQL Server};Server=NIKKYPC\SQLEXPRESS;Database=TechShopDB;Trusted_Connection=yes;'
```

OUTPUTS:

```
1. Add Customer
2. List Customers
3. Add Product
4. Place Order
5. Update Product Quantity
6. Track Order Status
7. Generate Sales Report
8. Update Customer Account
9. Payment Processing
10. Delete Product
11. Search Products by Category
12. Get Order Total
13. Exit
Enter your choice: 1
Enter first name: Sanjit
Enter last name: Jha
Enter email: sanjit@gmail.com
Enter phone: 7412369085
Enter address: Mumbai
Customer added successfully.
```

```
----- TechShop Menu -----

1. Add Customer
2. List Customers
3. Add Product
4. Place Order
5. Update Product Quantity
6. Track Order Status
7. Generate Sales Report
8. Update Customer Account
9. Payment Processing
10. Delete Product
11. Search Products by Category
12. Get Order Total
13. Exit
Enter your choice: 2
List of Customers:
Customer ID  First Name          Last Name           Email                       Phone
1            Sarthak             Kulkarni            sarthak@gmail.com           123456789
2            Lakshita            Sathe               lakshitasathe@mail.com      321654987
3            Vikas               Reddy               vikas@gmail.com             7410258963
4            Sanjit              Jha                 sanjit@gmail.com            7412369085
```

```
----- TechShop Menu -----

1. Add Customer
2. List Customers
3. Add Product
4. Place Order
5. Update Product Quantity
6. Track Order Status
7. Generate Sales Report
8. Update Customer Account
9. Payment Processing
10. Delete Product
11. Search Products by Category
12. Get Order Total
13. Exit
Enter your choice: 3
Enter product name: Monitor
Enter product description: 4k Hd display
Enter product price: 5999
Enter product quantity: 40
Enter product category: Display
Product added successfully.
```

```
----- TechShop Menu -----

1. Add Customer
2. List Customers
3. Add Product
4. Place Order
5. Update Product Quantity
6. Track Order Status
7. Generate Sales Report
8. Update Customer Account
9. Payment Processing
10. Delete Product
11. Search Products by Category
12. Get Order Total
13. Exit
Enter your choice: 4
Enter customer ID: 1
Enter product ID to order: 1
Enter quantity: 2
Do you want to add more products? (y/n): n

Order placed successfully.
```

```
----- TechShop Menu -----

1. Add Customer
2. List Customers
3. Add Product
4. Place Order
5. Update Product Quantity
6. Track Order Status
7. Generate Sales Report
8. Update Customer Account
9. Payment Processing
10. Delete Product
11. Search Products by Category
12. Get Order Total
13. Exit
Enter your choice: 5
Enter product ID to update quantity: 1
Enter new quantity: 1000
Product quantity updated successfully.
```

```
----- TechShop Menu -----

1. Add Customer
2. List Customers
3. Add Product
4. Place Order
5. Update Product Quantity
6. Track Order Status
7. Generate Sales Report
8. Update Customer Account
9. Payment Processing
10. Delete Product
11. Search Products by Category
12. Get Order Total
13. Exit
Enter your choice: 6
Enter order ID to track status: 3
Order Status for Order ID 3: Pending
```

```
----- TechShop Menu -----

1. Add Customer
2. List Customers
3. Add Product
4. Place Order
5. Update Product Quantity
6. Track Order Status
7. Generate Sales Report
8. Update Customer Account
9. Payment Processing
10. Delete Product
11. Search Products by Category
12. Get Order Total
13. Exit
Enter your choice: 7
Generate Sales Report
Enter start date (YYYY-MM-DD): 2024-10-1
Enter end date (YYYY-MM-DD): 2024-10-15
Sales Report:
Order ID   Order Date           Total Sales     Number of Orders
2          <20 89999.00        1
3          <20 179998.00       1
```

```
----- TechShop Menu -----

1. Add Customer
2. List Customers
3. Add Product
4. Place Order
5. Update Product Quantity
6. Track Order Status
7. Generate Sales Report
8. Update Customer Account
9. Payment Processing
10. Delete Product
11. Search Products by Category
12. Get Order Total
13. Exit
Enter your choice: 8
Enter your Customer ID: 4
Enter new email (leave blank to keep current): sarthakkul@gmail.com
Enter new phone number (leave blank to keep current): 9074125863
Customer information updated successfully.
```

```
----- TechShop Menu -----

1. Add Customer
2. List Customers
3. Add Product
4. Place Order
5. Update Product Quantity
6. Track Order Status
7. Generate Sales Report
8. Update Customer Account
9. Payment Processing
10. Delete Product
11. Search Products by Category
12. Get Order Total
13. Exit
Enter your choice: 9
Payment Processing
Enter the order ID: 8
Enter payment method (e.g., Credit Card, PayPal): Paypal
Enter payment amount: 9198
Payment processed successfully.
Payment processed successfully.
```

```
----- TechShop Menu -----

1. Add Customer
2. List Customers
3. Add Product
4. Place Order
5. Update Product Quantity
6. Track Order Status
7. Generate Sales Report
8. Update Customer Account
9. Payment Processing
10. Delete Product
11. Search Products by Category
12. Get Order Total
13. Exit
Enter your choice: 10
Enter product ID to delete: 8
Product deleted successfully.
```

```
----- TechShop Menu -----

1. Add Customer
2. List Customers
3. Add Product
4. Place Order
5. Update Product Quantity
6. Track Order Status
7. Generate Sales Report
8. Update Customer Account
9. Payment Processing
10. Delete Product
11. Search Products by Category
12. Get Order Total
13. Exit
Enter your choice: 11
Enter category to search: Electronics
Product ID: 1, Name: HP Laptop, Price: 89999.00, Stock: 1000
Product ID: 3, Name: Apple iPhone 15, Price: 129999.00, Stock: 75
Product ID: 4, Name: Sony Bravia TV, Price: 149999.00, Stock: 50
Product ID: 6, Name: Samsung Galaxy Tab S7, Price: 55999.00, Stock: 200
```

```
----- TechShop Menu -----

1. Add Customer
2. List Customers
3. Add Product
4. Place Order
5. Update Product Quantity
6. Track Order Status
7. Generate Sales Report
8. Update Customer Account
9. Payment Processing
10. Delete Product
11. Search Products by Category
12. Get Order Total
13. Exit
Enter your choice: 12
Enter customer ID: 1
Enter order ID: 3
Total amount for order ID 3: $179998.00
```

```
----- TechShop Menu -----

1. Add Customer
2. List Customers
3. Add Product
4. Place Order
5. Update Product Quantity
6. Track Order Status
7. Generate Sales Report
8. Update Customer Account
9. Payment Processing
10. Delete Product
11. Search Products by Category
12. Get Order Total
13. Exit
Enter your choice: 13
Exiting...
```