Assignment 1 – TechShop

Implementation of oops and exception handling

```python
class Customer:
    def __init__(self, customer_id, first_name, last_name, email, phone,
address):
        self.customer_id = customer_id
        self.first_name = first_name
        self.last_name = last_name
        self._email = email
        self.phone = phone
        self.address = address
        self.orders = []

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

    @email.setter
    def email(self, value):
        if not isinstance(value, str) or '@' not in value:
            raise ValueError("Invalid email address format.")
        self._email = value

    def calculate_total_orders(self):
        return len(self.orders)

    def get_customer_details(self):
        print("Customer ID:", self.customer_id)
        print("Name:", self.first_name, self.last_name)
        print("Email:", self._email)
        print("Phone:", self.phone)
        print("Address:", self.address)
        print("Total Orders:", self.calculate_total_orders())

    def update_customer_info(self, email=None, phone=None, address=None):
        if email:
            self.email = email
        if phone:
            self.phone = phone
        if address:
            self.address = address
        print("Customer information updated successfully.")


customer1 = Customer(1, "Asutosh", "Mishra", "asu@example.com",
"1234567890", "123 Main St")
customer1.get_customer_details()
customer1.update_customer_info(email="asutosh@example.com")
customer1.get_customer_details()
```

```python
class Product:
    def __init__(self, product_id, product_name, description, price,
in_stock=True):
        self.product_id = product_id
        self.product_name = product_name
        self.description = description
        self.price = price
        self.in_stock = in_stock

    def get_product_details(self):
        print("Product ID:", self.product_id)
        print("Product Name:", self.product_name)
        print("Description:", self.description)
        print("Price:", self.price)
        print("In Stock:", "Yes" if self.in_stock else "No")

    def update_product_info(self, price=None, description=None,
in_stock=None):
        if price:
            self.price = price
        if description:
            self.description = description
        if in_stock is not None:
            self.in_stock = in_stock
        print("Product information updated successfully.")

    def is_product_in_stock(self):
        return self.in_stock


class ProductManager:
    def __init__(self):
        self.products = {}

    def add_product(self, product):
        if product.product_id in self.products:
            raise ValueError("Product ID already exists. Please use a
different ID.")
        self.products[product.product_id] = product
        print(f"Product {product.product_id} added successfully.")

    def update_product(self, product_id, **kwargs):
        if product_id not in self.products:
            raise ValueError("Product ID does not exist.")

        product = self.products[product_id]
        product.update_product_info(**kwargs)
        print(f"Product {product_id} updated successfully.")

    def remove_product(self, product_id):
        if product_id not in self.products:
            raise ValueError("Product ID does not exist.")

        del self.products[product_id]
        print(f"Product {product_id} removed successfully.")


product_manager = ProductManager()


laptop = Product(1, "Laptop", "High-performance laptop", 999.99, True)
smartphone = Product(2, "Smartphone", "Latest smartphone model", 799.99,
True)

try:
    product_manager.add_product(laptop)
    product_manager.add_product(smartphone)
except ValueError as e:
    print(f"Error adding product: {e}")

try:
    product_manager.update_product(1, price=1099.99, description="Updated
laptop description", in_stock=False)
except ValueError as e:
    print(f"Error updating product: {e}")
try:
    product_manager.remove_product(2)
except ValueError as e:
    print(f"Error removing product: {e}")
laptop.get_product_details()
print("Is the product in stock?", laptop.is_product_in_stock())
```

```
class Orders:
    def __init__(self, order_id, customer, order_date, total_amount,
status, order_details):
        self.order_id = order_id
        self.customer = customer
        self.order_date = order_date
        self.total_amount = total_amount
        self.status = status
        self.order_details = order_details

    def CalculateTotalAmount(self):
        total_amount = sum(detail.product.price * detail.quantity for
detail in self.order_details)
        self.total_amount = total_amount
        return total_amount

    def GetOrderDetails(self):
        details = f"Order ID: {self.order_id}\nCustomer:
{self.customer}\nOrder Date: {self.order_date}\nStatus:
{self.status}\n\nOrder Details:\n"
        for detail in self.order_details:
            details += f"- Product: {detail.product.product_name},
Quantity: {detail.quantity}\n"
        return details

    def UpdateOrderStatus(self, new_status):
        self.status = new_status

    def CancelOrder(self, inventory_manager):
        for detail in self.order_details:
            inventory_manager.add_to_inventory(detail.product,
detail.quantity)
        self.status = "Canceled"
        print("Order canceled successfully.")


class OrderManager:
    def __init__(self):
        self.orders = []

    def add_order(self, order):
        self.orders.append(order)
        print("Order added successfully.")

    def update_order_status(self, order_id, new_status):
        for order in self.orders:
            if order.order_id == order_id:
                order.UpdateOrderStatus(new_status)
                print(f"Order {order_id} status updated to {new_status}.")
                return
        print(f"Order with ID {order_id} not found.")

    def cancel_order(self, order_id, inventory_manager):
        for order in self.orders:
            if order.order_id == order_id:
                order.CancelOrder(inventory_manager)
                self.orders.remove(order)
                print(f"Order {order_id} canceled.")
                return
        print(f"Order with ID {order_id} not found.")


def main():

    order_manager = OrderManager()
    order1 = Orders(1, "John Doe", "2024-01-30", 0, "Pending", [])
    order2 = Orders(2, "Jane Doe", "2024-01-31", 0, "Pending", [])

    order_manager.add_order(order1)
    order_manager.add_order(order2)
    order_manager.update_order_status(1, "Shipped")
    inventory_manager = InventoryManager()
    order_manager.cancel_order(2, inventory_manager)


class InventoryManager:
    def add_to_inventory(self, product, quantity):
        pass


if __name__ == "__main__":
    main()
```

```
from Customers import Customer
from Orders import Orders
from Products import Product


class OrderDetails:
    def __init__(self, order_detail_id, order, product, quantity,
discount=0):
        self.order_detail_id = order_detail_id
        self._order = order
        self._product = product
        self.quantity = quantity
        self.discount = discount

    def calculate_subtotal(self):
        return self.quantity * self._product.price * (1 - self.discount)

    def get_order_detail_info(self):
        print("Order Detail ID:", self.order_detail_id)
        print("Product:", self._product.product_name)
        print("Quantity:", self.quantity)
        print("Discount:", self.discount)
        print("Subtotal:", self.calculate_subtotal())
    def update_quantity(self, new_quantity):
        self.quantity = new_quantity
        print("Quantity updated to:", self.quantity)

    def add_discount(self, discount_amount):
        self.discount += discount_amount
        print("Discount added. New discount:", self.discount)


product1 = Product(1, "Laptop", "High-performance laptop", 999.99)

customer1 = Customer(1, "John", "Doe", "john@example.com", "1234567890",
"123 Main St")
order1 = Orders(1, customer1, 0)
order_detail1 = OrderDetails(1, order1, product1, 2)

order_detail1.get_order_detail_info()

order_detail1.update_quantity(3)
order_detail1.get_order_detail_info()
order_detail1.add_discount(0.1)
order_detail1.get_order_detail_info()
```

```
from sortedcontainers import SortedList
class InventoryItem:
    def __init__(self, product, quantity):
        self.product = product
        self.quantity = quantity

class Inventory:
    def __init__(self):
        self.inventory_list = SortedList()

    def add_to_inventory(self, product_id, quantity):
        # Check if the product already exists in the inventory
        def add_product(self, product, quantity):
            inventory_item = InventoryItem(product, quantity)
            self.inventory_items.append(inventory_item)

        # If the product is not found, add it to the inventory
        self.inventory_list.add([product_id, quantity])

    def remove_from_inventory(self, product_id, quantity):
        for item in self.inventory_list:
            if item[0] == product_id:
                if item[1] >= quantity:
                    item[1] -= quantity
                    if item[1] == 0:
                        self.inventory_list.remove(item)
                    return
                else:
                    print("Error: Insufficient quantity in stock.")
                    return
        print("Error: Product not found in inventory.")

    def update_stock_quantity(self, product_id, new_quantity):
        for item in self.inventory_list:
            if item[0] == product_id:
                item[1] = new_quantity
                return
        print("Error: Product not found in inventory.")

    def is_product_available(self, product_id, quantity_to_check):
        for item in self.inventory_list:
            if item[0] == product_id:
                return item[1] >= quantity_to_check
        return False

    def get_inventory_value(self):
        total_value = sum(item[1] * self.get_product_price(item[0]) for
item in self.inventory_list)
        return total_value

    def list_low_stock_products(self, threshold):
        low_stock_products = [item for item in self.inventory_list if
item[1] < threshold]
        return low_stock_products

    def list_out_of_stock_products(self):
        out_of_stock_products = [item for item in self.inventory_list if
item[1] == 0]
        return out_of_stock_products

    def list_all_products(self):
        return self.inventory_list

    def get_product_price(self, product_id):

        return 0

def main():
    inventory = Inventory()
    inventory.add_to_inventory(1, 10)
    inventory.add_to_inventory(2, 5)

    # Removing products from inventory
    inventory.remove_from_inventory(1, 3)

    # Updating stock quantity
    inventory.update_stock_quantity(2, 7)

    # Checking product availability
    print("Product 1 available:", inventory.is_product_available(1, 5))
    print("Product 2 available:", inventory.is_product_available(2, 10))

    # Getting inventory value
    print("Inventory value:", inventory.get_inventory_value())

    # Listing low stock products
    print("Low stock products:", inventory.list_low_stock_products(5))

    # Listing out of stock products
    print("Out of stock products:", inventory.list_out_of_stock_products())

    # Listing all products
    print("All products:", inventory.list_all_products())

if __name__ == "__main__":
    main()
```

Implementation of database connectivity

```python
import mysql.connector
from mysql.connector import Error


class DatabaseConnector:
    def __init__(self, host, database, user, password, port):
        self.host = host
        self.database = database
        self.user = user
        self.password = password
        self.connection = None
        self.port = port

    def open_connection(self):
        try:
            self.connection = mysql.connector.connect(
                host=self.host,
                database=self.database,
                user=self.user,
                password=self.password,
                port=self.port
            )
            if self.connection.is_connected():
                print("Connected to MySQL database")
        except Error as e:
            print("Error connecting to MySQL database:", e)

    def close_connection(self):
        if self.connection and self.connection.is_connected():
            self.connection.close()
            print("Connection to MySQL database closed")


class Customer:
    def __init__(self, customer_id, first_name, last_name, email, phone,
address):
        self.customer_id = customer_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone = phone
        self.address = address

    @staticmethod
    def create(customer, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("INSERT INTO Customers (CustomerID, FirstName,
LastName, Email, Phone, Address) "
                           "VALUES (%s, %s, %s, %s, %s, %s)",
                           (customer.customer_id, customer.first_name,
customer.last_name,
                            customer.email, customer.phone,
customer.address))
            db_connector.connection.commit()
            print("Customer created successfully.")
        except Error as e:
            print("Error creating customer:", e)
        finally:
            db_connector.close_connection()

    @staticmethod
    def read(customer_id, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("SELECT * FROM Customers WHERE CustomerID = %s",
(customer_id,))
            customer_data = cursor.fetchone()

            if customer_data:
                return Customer(*customer_data)
            else:
                print("Customer not found.")
                return None
        except Error as e:
            print("Error reading customer:", e)
        finally:
            db_connector.close_connection()

    def update(self, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("UPDATE Customers SET FirstName = %s, LastName =
%s, "
                           "Email = %s, Phone = %s, Address = %s WHERE
CustomerID = %s",
                           (self.first_name, self.last_name, self.email,
self.phone,
                            self.address, self.customer_id))
            db_connector.connection.commit()
            print("Customer updated successfully.")
        except Error as e:
            print("Error updating customer:", e)
        finally:
            db_connector.close_connection()

    @staticmethod
    def delete(customer_id, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("DELETE FROM Customers WHERE CustomerID = %s",
(customer_id,))
            db_connector.connection.commit()
            print("Customer deleted successfully.")
        except Error as e:
            print("Error deleting customer:", e)
        finally:
            db_connector.close_connection()


class Product:
    def __init__(self, product_id, product_name, description, price):
        self.product_id = product_id
        self.product_name = product_name
        self.description = description
        self.price = price
    @staticmethod
    def create(product, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("INSERT INTO Products (ProductID, ProductName,
Description, Price) "
                           "VALUES (%s, %s, %s, %s)",
                           (product.product_id, product.product_name,
product.description, product.price))
            db_connector.connection.commit()
            print("Product created successfully.")
        except Error as e:
            print("Error creating product:", e)
        finally:
            db_connector.close_connection()

    @staticmethod
    def read(product_id, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("SELECT * FROM Products WHERE ProductID = %s",
(product_id,))
            product_data = cursor.fetchone()

            if product_data:
                return Product(*product_data)
            else:
                print("Product not found.")
                return None
        except Error as e:
            print("Error reading product:", e)
        finally:
            db_connector.close_connection()

    def update(self, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("UPDATE Products SET ProductName = %s,
Description = %s, "
                           "Price = %s WHERE ProductID = %s",
                           (self.product_name, self.description,
self.price, self.product_id))
            db_connector.connection.commit()
            print("Product updated successfully.")
        except Error as e:
            print("Error updating product:", e)
        finally:
            db_connector.close_connection()

    @staticmethod
    def delete(product_id, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("DELETE FROM Products WHERE ProductID = %s",
(product_id,))
            db_connector.connection.commit()
            print("Product deleted successfully.")
        except Error as e:
            print("Error deleting product:", e)
        finally:
            db_connector.close_connection()


class Order:
    def __init__(self, order_id, customer_id, order_date, total_amount):
        self.order_id = order_id
        self.customer_id = customer_id
        self.order_date = order_date
        self.total_amount = total_amount

    @staticmethod
    def create(order, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("INSERT INTO Orders (OrderID, CustomerID,
OrderDate, TotalAmount) "
                           "VALUES (%s, %s, %s, %s)",
                           (order.order_id, order.customer_id,
order.order_date, order.total_amount))
            db_connector.connection.commit()
            print("Order created successfully.")
        except Error as e:
            print("Error creating order:", e)
        finally:
            db_connector.close_connection()

    @staticmethod
    def read(order_id, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("SELECT * FROM Orders WHERE OrderID = %s",
(order_id,))
            order_data = cursor.fetchone()

            if order_data:
                return Order(*order_data)
            else:
                print("Order not found.")
                return None
        except Error as e:
            print("Error reading order:", e)
        finally:
            db_connector.close_connection()

    def update(self, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("UPDATE Orders SET CustomerID = %s, OrderDate =
%s, "
                           "TotalAmount = %s WHERE OrderID = %s",
                           (self.customer_id, self.order_date,
self.total_amount, self.order_id))
            db_connector.connection.commit()
            print("Order updated successfully.")
        except Error as e:
            print("Error updating order:", e)
        finally:
            db_connector.close_connection()

    @staticmethod
    def delete(order_id, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("DELETE FROM Orders WHERE OrderID = %s",
(order_id,))
            db_connector.connection.commit()
            print("Order deleted successfully.")
        except Error as e:
            print("Error deleting order:", e)
        finally:
            db_connector.close_connection()


class Inventory:
    def __init__(self, inventory_id, product_id, quantity_in_stock,
last_stock_update):
        self.inventory_id = inventory_id
        self.product_id = product_id
        self.quantity_in_stock = quantity_in_stock
        self.last_stock_update = last_stock_update

    @staticmethod
    def create(inventory, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("INSERT INTO Inventory (InventoryID, ProductID,
QuantityInStock, LastStockUpdate) "
                           "VALUES (%s, %s, %s, %s)",
                           (inventory.inventory_id, inventory.product_id,
inventory.quantity_in_stock,
                            inventory.last_stock_update))
            db_connector.connection.commit()
            print("Inventory created successfully.")
        except Error as e:
            print("Error creating inventory:", e)
        finally:
            db_connector.close_connection()

    @staticmethod
    def read(inventory_id, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("SELECT * FROM Inventory WHERE InventoryID =
%s", (inventory_id,))
            inventory_data = cursor.fetchone()

            if inventory_data:
                return Inventory(*inventory_data)
            else:
                print("Inventory not found.")
                return None
        except Error as e:
            print("Error reading inventory:", e)
        finally:
            db_connector.close_connection()

    def update(self, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("UPDATE Inventory SET ProductID = %s,
QuantityInStock = %s, "
                           "LastStockUpdate = %s WHERE InventoryID = %s",
                           (self.product_id, self.quantity_in_stock,
self.last_stock_update, self.inventory_id))
            db_connector.connection.commit()
            print("Inventory updated successfully.")
        except Error as e:
            print("Error updating inventory:", e)
        finally:
            db_connector.close_connection()

    @staticmethod
    def delete(inventory_id, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("DELETE FROM Inventory WHERE InventoryID = %s",
(inventory_id,))
            db_connector.connection.commit()
            print("Inventory deleted successfully.")
        except Error as e:
            print("Error deleting inventory:", e)
        finally:
            db_connector.close_connection()


class OrderDetail:
    def __init__(self, order_detail_id, order_id, product_id, quantity):
        self.order_detail_id = order_detail_id
        self.order_id = order_id
        self.product_id = product_id
        self.quantity = quantity

    @staticmethod
    def create(order_detail, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("INSERT INTO OrderDetails (OrderDetailID,
OrderID, ProductID, Quantity) "
                           "VALUES (%s, %s, %s, %s)",
                           (order_detail.order_detail_id,
order_detail.order_id, order_detail.product_id,
                            order_detail.quantity))
            db_connector.connection.commit()
            print("Order detail created successfully.")
        except Error as e:
            print("Error creating order detail:", e)
        finally:
            db_connector.close_connection()

    @staticmethod
```

```python
    def read(order_detail_id, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("SELECT * FROM OrderDetails WHERE OrderDetailID
= %s", (order_detail_id,))
            order_detail_data = cursor.fetchone()

            if order_detail_data:
                return OrderDetail(*order_detail_data)
            else:
                print("Order detail not found.")
                return None
        except Error as e:
            print("Error reading order detail:", e)
        finally:
            db_connector.close_connection()

    def update(self, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("UPDATE OrderDetails SET OrderID = %s, ProductID
= %s, "
                           "Quantity = %s WHERE OrderDetailID = %s",
                           (self.order_id, self.product_id, self.quantity,
self.order_detail_id))
            db_connector.connection.commit()
            print("Order detail updated successfully.")
        except Error as e:
            print("Error updating order detail:", e)
        finally:
            db_connector.close_connection()

    @staticmethod
    def delete(order_detail_id, db_connector):
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("DELETE FROM OrderDetails WHERE OrderDetailID =
%s", (order_detail_id,))
            db_connector.connection.commit()
            print("Order detail deleted successfully.")
        except Error as e:
            print("Error deleting order detail:", e)
        finally:
            db_connector.close_connection()


def main():
    # Database connection details
    host = 'localhost'
    database = 'techshopdb'
    user = 'root'
    password = 'root'
    port = '3306'

    # Create a DatabaseConnector instance
    db_connector = DatabaseConnector(host, database, user, password, port)

    # Create a new customer
    new_customer = Customer("101", "John", "Doe", "john@example.com",
"1234567890", "123 Main St")

    # Create the customer
    Customer.create(new_customer, db_connector)

    # Read the customer
    customer_id_to_read = "101"
    customer_read = Customer.read(customer_id_to_read, db_connector)
    if customer_read:
        print("Customer read:", customer_read.__dict__)

    # Update the customer
    if customer_read:
        customer_read.phone = "9876543210"
        customer_read.update(db_connector)

    # Delete the customer
    customer_id_to_delete = "101"
    Customer.delete(customer_id_to_delete, db_connector)

    # Create a new product
    new_product = Product("P101", "Laptop", "High-performance laptop",
999.99)

    # Create the product
    Product.create(new_product, db_connector)

    # Read the product
    product_id_to_read = "P101"
    product_read = Product.read(product_id_to_read, db_connector)
    if product_read:
        print("Product read:", product_read.__dict__)

    # Update the product
    if product_read:
        product_read.price = 1099.99
        product_read.update(db_connector)

    # Delete the product
    product_id_to_delete = "P101"
    Product.delete(product_id_to_delete, db_connector)

    # Create a new order
    new_order = Order("O101", "C101", "2023-01-15", 499.99)

    # Create the order
    Order.create(new_order, db_connector)

    # Read the order
    order_id_to_read = "O101"
    order_read = Order.read(order_id_to_read, db_connector)
    if order_read:
        print("Order read:", order_read.__dict__)

    # Update the order
    if order_read:
        order_read.total_amount = 599.99
        order_read.update(db_connector)

    # Delete the order
    order_id_to_delete = "O101"
    Order.delete(order_id_to_delete, db_connector)

    # Create a new inventory
    new_inventory = Inventory("I101", "P101", 100, "2023-01-15")

    # Create the inventory
    Inventory.create(new_inventory, db_connector)

    # Read the inventory
    inventory_id_to_read = "I101"
    inventory_read = Inventory.read(inventory_id_to_read, db_connector)
    if inventory_read:
        print("Inventory read:", inventory_read.__dict__)

    # Update the inventory
    if inventory_read:
        inventory_read.quantity_in_stock = 200
        inventory_read.update(db_connector)

    # Delete the inventory
    inventory_id_to_delete = "I101"
    Inventory.delete(inventory_id_to_delete, db_connector)

    # Create a new order detail
    new_order_detail = OrderDetail("OD101", "O101", "P101", 5)

    # Create the order detail
    OrderDetail.create(new_order_detail, db_connector)

    # Read the order detail
    order_detail_id_to_read = "OD101"
    order_detail_read = OrderDetail.read(order_detail_id_to_read,
db_connector)
    if order_detail_read:
        print("Order detail read:", order_detail_read.__dict__)

    # Update the order detail
    if order_detail_read:
        order_detail_read.quantity = 10
        order_detail_read.update(db_connector)

    # Delete the order detail
    order_detail_id_to_delete = "OD101"
    OrderDetail.delete(order_detail_id_to_delete, db_connector)


if __name__ == "__main__":
    main()
```

Various Database operations

```
from DBconnector import *


def registerCustomer(dbConnector):
    first_name = input("Enter your first name: ")
    last_name = input("Enter your last name: ")
    email = input("Enter your email: ")
    phone = input("Enter your phone number: ")
    address = input("Enter your address: ")

    customer = Customer(None, first_name, last_name, email, phone, address)
    Customer.create(customer, dbConnector)


def updateAccountDetails(dbConnector):
    c_id = input("Enter your customer id: ")
    ch = int(input("what do you want to update: 1.Email\n2.phone number"))
    if ch == 1:
        email = input("enter the new email: ")
        Customer.update(dbConnector, c_id, email)
    elif ch == 2:
        phone = input("Enter phone number: ")
        Customer.update(dbConnector, c_id, phone)


def updateProductInfo(dbConnector):
    p_id = input("Enter the product id: ")
    prod_read = Product.read(p_id, dbConnector)
    if prod_read:
        ch = int(input("Enter what to update: 1.Price\n2.Description"))
        if ch == 1:
            price = input("Enter the price: ")
            prod_read.price = price
            prod_read.update(dbConnector)
        elif ch == 2:
            desc = input("Enter the new description: ")
            prod_read.description = desc
            prod_read.update(dbConnector)

    else:
        print("No such Product Found.")


def product_search(db_connector):
    p_name = input("Enter the product name: ")
    try:
        db_connector.open_connection()
        cursor = db_connector.connection.cursor()
        cursor.execute("SELECT * FROM Products WHERE product_name = %s",
(p_name,))
        product_data = cursor.fetchone()

        if product_data:
            return Product(*product_data)
        else:
            print("Product not found.")
            return None
    except Error as e:
        print("Error reading product:", e)
    finally:
        db_connector.close_connection()
def track_order_status(db_connector):
    o_id = input("Enter your order id: ")
    db_connector.open_connection()
    cursor = db_connector.connection.cursor()
    cursor.execute("SELECT OrderID,CASE WHEN DATEDIFF(CURDATE(), OrderDate)
= 0 THEN 'Pending' WHEN DATEDIFF(CURDATE("
                   "), OrderDate) >= 3 THEN 'Processing' WHEN
DATEDIFF(CURDATE(), OrderDate) >= 7 THEN 'Delivered' "
```

```
                   "ELSE 'Completed' END AS Status FROM Orders where
orderid = %s", (o_id,))
    res = cursor.fetchall()
    for i in res:
        print(i)
def inventory_management(dbConnector):
    ch = input("Enter what you want to do: 1.Add new Product\n2.Update
quantities\n3.Delete a product")
    if ch == 1:
        prod_id = input("Enter the product id: ")
        quantity = input("Enter the quantity: ")
        last_stock_update = input("Enter the date of last stock update: ")
        new_inventory = Inventory("I101", prod_id, quantity,
last_stock_update)
        Inventory.create(new_inventory, db_connector)
    elif ch == 2:
        p_id = input("Enter the product id: ")
        quantity = input("Enter the new quantity:")
        db_connector.open_connection()
        cursor = db_connector.connection.cursor()
        cursor.execute("SELECT * FROM Inventory WHERE product_id = %s",
(p_id,))
        inventory_data = cursor.fetchone()
        if inventory_data:
            inventory_data.quantity_in_stock = 200
            inventory_data.update(db_connector)
    elif ch == 3:
        i_id = input("Enter the inventory id: ")
        try:
            db_connector.open_connection()
            cursor = db_connector.connection.cursor()
            cursor.execute("DELETE FROM Inventory WHERE InventoryID = %s",
(i_id,))
            db_connector.connection.commit()
            print("Inventory deleted successfully.")
        except Error as e:
            print("Error deleting inventory:", e)
        finally:
            db_connector.close_connection()

# Database connection details
host = 'localhost'
database = 'techshopdb'
user = 'root'
password = 'root'
port = '3306'

# Create a DatabaseConnector instance
db_connector = DatabaseConnector(host, database, user, password, port)
registerCustomer(db_connector)
updateAccountDetails(db_connector)
updateProductInfo(db_connector)
product_search(db_connector)
track_order_status(db_connector)
inventory_management(db_connector)
```