TechShop, an Electronic Gadgets Shop

Ahmed Sherif

Table Of Contents

INTRODUCTION

The TechShop Electronic Gadgets Management System is an object-oriented software solution designed to streamline operations for electronics retailers. This system leverages Python's object-oriented programming (OOP) paradigm to model real-world retail entities and processes, providing a structured approach to managing customers, products, inventory, and sales transactions.

This project addresses these challenges by implementing core OOP principles—encapsulation, inheritance, composition, and polymorphism—to create a modular, extensible architecture.

System Relevance

- **For Retailers:** Automates inventory tracking, reduces manual errors in order processing, and provides actionable insights through integrated reporting.
- **For Developers:** Demonstrates practical application of OOP concepts in a business context, serving as a template for similar systems.
- **For Customers:** Enhances shopping experiences through accurate stock visibility and order status updates.

This report documents the OOP implementation, highlighting class designs, key methodologies, and how object-oriented principles solve specific retail management challenges. The subsequent sections detail the system's architecture, class hierarchies, and the rationale behind critical design decisions.

PURPOSE OF THE PROJECT

The purpose of the TechShop Electronic Gadgets Management System project is to design and implement an object-oriented application that efficiently manages electronic gadget sales operations including customer information, product details, inventory management, and order processing. The system aims to provide a structured solution for managing all aspects of an electronic retail business using Python's object-oriented programming principles.

This project demonstrates how a TechShop Electronic Gadgets shop handles Customers, Products, Orders, Inventory and Payment.

The Project includes:

- Class Design: Creating well-structured entity classes (Customers, Products, Orders, OrderDetails, Inventory) with appropriate attributes and methods.
- Encapsulation: Implementing proper data hiding and validation through private attributes with getter/setter methods.
- Composition: Establishing relationships between classes to model real-world business relationships.
- Exception Handling: Creating custom exceptions to handle various business scenarios gracefully.
- Collections Management: Using Python collections to manage dynamic data like product lists and order histories.
- Database Integration: Connecting the OOP model with a backend database for persistent storage.

SCOPE OF THE PROJECT

System Overview:

The TechShop Gadget Hub is a comprehensive management solution tailored for electronics retailers, providing end-to-end control over customer relationships, product offerings, and sales operations. Built on Python's object-oriented framework, the system establishes a scalable architecture that seamlessly integrates business logic with future database and interface implementations.

Architectural Components

Core Business Entities

1. **Client Profile Module**
    - Central repository for shopper demographics and engagement metrics
    - Advanced functionality for purchase analytics and profile maintenance
    - Secure credential management and contact information storage
2. **Device Catalog Component**
    - Centralized repository for gadget specifications and pricing structures
    - Real-time stock availability indicators and product lifecycle tracking
    - Multimedia support for product demonstrations and technical documentation

3. **Transaction Management System**
    - Complete order lifecycle processing from creation to fulfillment
    - Integrated pricing calculators with automatic tax and discount applications
    - Multi-item order handling with bundled product support
4. **Order Itemization Engine**
    - Granular control over individual order components
    - Dynamic pricing adjustments based on quantity and promotions
    - Cross-selling and accessory recommendation capabilities
5. **Stock Control Center**
    - Real-time inventory monitoring across multiple locations
    - Automated reorder triggers and supply chain integration points
    - Product movement tracking with timestamped audit trails

Service Infrastructure

1. **Client Services**
    - New account onboarding workflows
    - Personalized dashboard with purchase history
    - Self-service profile customization portal
2. **Product Administration**
    - Batch updating for seasonal pricing changes
    - Discontinued product phase-out management
3. **Order Processing**
    - Status notification engine for customers
    - Returns and exchange processing
4. **Inventory Operations**
    - Automated stock level alerts
    - Warehouse transfer coordination
    - Supplier order generation

Operational Capabilities:

Client Management Features

- Digital enrollment for new shoppers
- Contact detail verification and synchronization

- Formatted customer entries

Product Administration Features

- New gadget introduction workflows
- Technical specification maintenance
- Detailed description of products
- Real-time availability indicators

Sales Processing Features

- Shopping cart implementation
- Order confirmation protocols
- Cancellation workflow with restocking
- Secure payment and sure balance mechanism
- Refund mechanism

Inventory Management Features

- Bulk stock receipt processing
- Cycle counting automation
- Easy sorting mechanism
- Low stock identifiers

SOFTWARES USED

- PyCharm: It is a python IDE
- MySQL : It is database management system used to store the information entered.

IMPLEMENT OOPs

Task 1: Classes and Their Attributes:

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).

Customers Class:

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

```
        self.last_name = last_name
        self.email = email
        self.phone = phone
        self.address = address

def calculate_total_orders(self):
    return len(self.__orders)
```

```
------ Customer Management ------
1. Add New Customer
2. View Customer Details
3. Update Customer Information
4. Delete Customer
5. List All Customers
6. Back to Main Menu

Enter your choice (1-6): 1

[Add New Customer]
First Name: Melina
Last Name: Wochester
Email: Melina@gmail.com
Phone: 9824753723
Address: Erd Tree Ave

Customer created successfully. ID- 12
```

CustomerID	FirstName	LastName	Email	Phone	Address
12	Melina	Wochester	Melina@gmail.com	9824753723	Erd Tree Ave

```
def get_customer_details(self,order_count=None):
    details= (f"Customer ID: {self.__customer_id}\n"
            f"Name: {self.__first_name} {self.__last_name}\n"
            f"Email: {self.__email}\n"
            f"Phone: {self.__phone}\n"
            f"Address: {self.__address}\n")
    if order_count is not None:
        details += f"\nTotal Orders: {order_count}"
    else:
        details += f"\nTotal Orders: {self.calculate_total_orders()}"
```

```
------ Customer Management ------
1. Add New Customer
2. View Customer Details
3. Update Customer Information
4. Delete Customer
5. List All Customers
6. Back to Main Menu

Enter your choice (1-6): 2

[View Customer Details]
Enter Customer ID: 1
Customer ID: 1
Name: Sungjinwoo Singh
Email: jinwoo@gmail.com
Address: Shadow Monarch Island
Total Orders: 4
```

```
ID  Name            Email          Orders
-------------------------------------------------------------
1   Sungjinwoo Singh            jinwoo@gmail.com       4
2   Ichigo Kumar              kurosaki.ichigo@gmai   2
3   Isagi Kahn                clown@gmail.com        1
4   Uzumaki Nair                boruto.nair@gmail.co   1
5   Gojo Reddy                gojo.suguru@gmail.co   1
6   Mohammed Aizen                Aizen.watashi@gmail    1
```

```
def update_customer_info(self, first_name=None, last_name=None, email=None,
phone=None, address=None):
    if first_name is not None:
        self.first_name = first_name
    if last_name is not None:
        self.last_name = last_name
    if email is not None:
        self.email = email
    if phone is not None:
        self.phone = phone
    if address is not None:
        self.address = address
```

```
[View Customer Details]
Enter Customer ID: 12
Customer ID: 12
Name: Nameless Wochester
Email: Blanks@gmail.com
Phone: 9824753723
Address: Erd Tree Ave

Total Orders: 0
```

```
[Update Customer Information]
Enter Customer ID to update: 12

Current Details:
Customer ID: 12
Name: Melina Wochester
Email: Melina@gmail.com
Phone: 9824753723
Address: Erd Tree Ave

Total Orders: 0

Enter new details (leave blank to keep current):
First Name [Melina]: Nameless
Last Name [Wochester]:
Email [Melina@gmail.com]: Blanks@gmail.com
Phone [9824753723]:
Address [Erd Tree Ave]:

 Customer updated successfully!
```

CustomerID	FirstName	LastName	Email	Phone	Address
12	Nameless	Wochester	Blanks@gmail.com	9824753723	Erd Tree Ave

Products Class:

Attributes:

• ProductID (int)

• ProductName (string)

• Description (string)

• Price (decimal)

```
class Product:
    def __init__(self, product_id, product_name,
    description, price, category):
        self.__product_id = product_id
        self.__product_name = product_name
        self.__description = description
        self.__price = price
        self.__category = category
        self.__stock_quantity = stock_quantity
```

```
--- Product Management ---
1. Add New Product
2. View Product Details
3. Update Product Information
4. Delete Product
5. List All Products
6. Search Products
7. Check Product Stock
8. Back to Main Menu

Enter your choice (1-7): 1

[Add New Product]
Product Name: Smart Phone
Description: Super fast gaming phone
Price: 220
Category: Electronics

 Product created successfully! ID: 9
```

1) GetProductDetails(): Retrieves and displays detailed information about the product.

```
def get_product_details(self):
    return (f"Product ID: {self.__product_id}\n"
            f"Name: {self.__product_name}\n"
            f"Category: {self.__category}\n"
            f"Description: {self.__description}\n"
            f"Price: {self.__price}")
```

```
--- Product Management ---
1. Add New Product
2. View Product Details
3. Update Product Information
4. Delete Product
5. List All Products
6. Search Products
7. Check Product Stock
8. Back to Main Menu

Enter your choice (1-7): 5

[List All Products]

ID  Name    Description Category    Price
------------------------------------------------------------
1   Shadow SSD      A high-speed 1TB SSD with rapi  Storage Device              825.00
2   Hollow VR Heads A VR headset with immersive au  Wearable Techno             2530.00
3   Tactical Smart  Advanced sports analyzing watc  Wearable Techno             1700.00
4   Rasengan Drone  High-speed drone with rotor bl  Drones                      3200.00
5   Infinity Projec Projector with limitless focus   Projector                  1980.00
6   Illusionary Sma Smart glasses with holographic  Wearable Techno             2200.00
7   Laptop          High Processing Gaming Laptop   Electronics                50.00
8   Tablet          Foldable Tablet                 Electronics                100.00
9   Smart Phone     Super fast gaming phone         Electronics                220.00
```

2) UpdateProductInfo(): Allows updates to product details (e.g., price, description).

```
def update_product_info(self, product_name=None, description=None, price=None,
category=None):
    if product_name is not None:
        self.product_name = product_name
    if description is not None:
        self.description = description
```

```
    if price is not None:
        self.price = price
    if category is not None:
        self.category = category
```

```
[Update Product Information]
Enter Product ID to update: 9

Current Details:
Product ID: 9
Name: Smart Phone
Category: Electronics
Description: Super fast gaming phone
Price: 220.00

Enter new details (leave blank to keep current):
Name [Smart Phone]: IPad
Description [Super fast gaming phone]: Super fast gaming Ipad
Price [220.00]: 350
Category [Electronics]:

 Product updated successfully!
```

```
[View Product Details]
Enter Product ID: 9

Product Details:
Product ID: 9
Name: IPad
Category: Electronics
Description: Super fast gaming Ipad
Price: 350.00
Current Stock: 0
```

ProductID	ProductName	Description	Price	Category
9	IPad	Super fast gaming Ipad	350.00	Electronics

3) IsProductInStock(): Checks if the product is currently in stock.

```
def is_in_stock(self, quantity=1):
            if self.__stock_quantity is None:
        raise ValueError("Stock quantity not initialized")
    return self.__stock_quantity >= quantity
```

```
--- Product Management ---
1. Add New Product
2. View Product Details
3. Update Product Information
4. Delete Product
5. List All Products
6. Search Products
7. Check Product Stock
8. Back to Main Menu

Enter your choice (1-7): 7

[Check Product Stock]
Enter Product ID to check stock: 1

Product Stock Information:
Product ID: 1
Product Name: Shadow SSD
Quantity in Stock: 21
```

Orders Class:

Attributes:

• OrderID (int)

• Customer (Customer) - Use composition to reference the Customer who placed the order.

• OrderDate (DateTime)

• TotalAmount (decimal)

```
class Order:
   def __init__(self, order_id, customer, order_date=None, total_amount=0.0,
status="Pending",version= 1):
        self.__order_id = order_id
        self.__customer = customer  # Composition with Customer
        self.__order_date = order_date if order_date else datetime.now()
        self.__total_amount = total_amount
        self.__status = status
        self.__order_details = []
        self.__version = version
```

```
-------- Order Management ----------
1. Place New Order
2. View Order Details
3. Update Order Status
4. Cancel an Order
5. List All Orders
6. Payment Management
7. Return to Main Menu

Enter your choice (1-7): 1

[Place New Order]
Enter Customer ID: 7

Current Order:
Items: 0
Total: AED0.00

1. Add Product
2. Finalize Order
Select option: 1
Enter Product ID: 9
Enter Quantity: 5
Added 5x IPad at AED 350.00 each
Item subtotal: AED 1750.00
```

1) CalculateTotalAmount() - Calculate the total amount of the order.

```
def calculate_total_amount(self):
    return sum(detail.subtotal for detail in self.__order_details)
```

```
[View Order Details]
Enter Order ID: 14

Order #14
Customer: Ahmed Sherif
Date: 2025-04-05 12:41
Status: Pending
Total: AED1750.00

Items:
Product Name:IPad
 Quantity: 5
Product Price: AED 350.00
 AED 1750.00
```

• GetOrderDetails(): Retrieves and displays the details of the order (e.g., product list and quantities).

```
def get_order_details(self):
    details = f"Order #{self.__order_id}\n"
    details += f"Customer: {self.__customer.first_name} {self.__customer.last_name}\n"
    details += f"Date: {self.__order_date.strftime('%Y-%m-%d %H:%M')}\n"
    details += f"Status: {self.__status}\n"
    details += f"Total: AED{self.__total_amount:.2f}\n\n"
    details += "Items:\n"
    for detail in self.__order_details:
        details += (f"Product Name:{detail.product.product_name}\n Quantity:
{detail.quantity} \n"
                    f"Product Price: AED {detail.product.price:.2f}\n AED
{detail.calculate_subtotal():.2f}\n")
    return details
```


```
[View Order Details]
Enter Order ID: 14

Order #14
Customer: Ahmed Sherif
Date: 2025-04-05 12:41
Status: Pending
Total: AED1750.00

Items:
Product Name:IPad
 Quantity: 5
Product Price: AED 350.00
 AED 1750.00
```

• UpdateOrderStatus(): Allows updating the status of the order (e.g., processing, shipped).

```
def update(self, order):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()
        conn.autocommit = False

        query = """ update orders
        set status = %s
        where orderid = %s """
        cursor.execute(query, (
            order.status,
            order.order_id,
        ))

        conn.commit()
        return True
```

```
[Update Order Status]
Enter Order ID: 14

Current Status: Pending

Available statuses: Pending, Processing, Shipped, Delivered, Cancelled
Enter new status: Delivered

 Order status updated successfully!
```

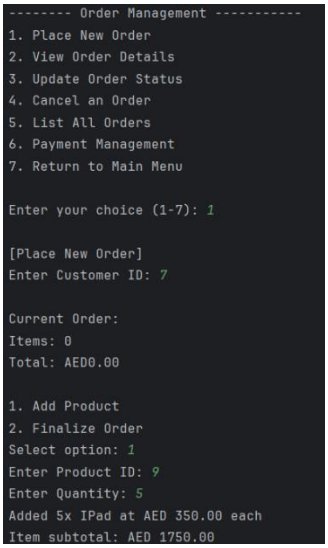
• CancelOrder(): Cancels the order and adjusts stock levels for products.

```
def delete(self, order_id):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()
        conn.autocommit = False

        details_query = "select productid, quantity from orderdetails where orderid = %s"
        cursor.execute(details_query, (order_id,))
        details = cursor.fetchall()

        for product_id, quantity in details:
            restock_query = """ update inventory
            set quantityinstock = quantityinstock + %s
            where productid = %s """
            cursor.execute(restock_query, (quantity, product_id))

        delete_query = "delete from orders where orderid = %s"
        cursor.execute(delete_query, (order_id,))

        if cursor.rowcount == 0:
```

```
[Cancel Order]
Enter Order ID to cancel: 14

Order to cancel:
Order #14
```

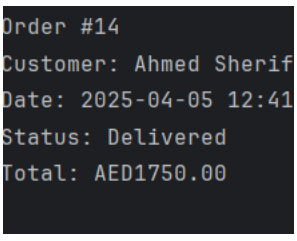
```
            raise OrderNotFoundException(f"Order with ID {order_id} not found")

        conn.commit()
        return True

    except Exception as e:
        if conn:
            conn.rollback()
        raise
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
```

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)




```
class OrderDetail:
    def __init__(self, order_detail_id, order, product, quantity, unit_price):
        self.__order_detail_id = order_detail_id
        self.__order = order  # Composition with Order
        self.__product = product  # Composition with Product
        self.__quantity = quantity
        self.__unit_price = unit_price
        self.__subtotal = self.calculate_subtotal()
```

Methods:
• CalculateSubtotal() - Calculate the subtotal for this order detail.




```
def calculate_subtotal(self):
    return self.__unit_price * self.__quantity
```

• GetOrderDetailInfo(): Retrieves and displays information about this order detail.

```
def count_orders_by_customer(self, customer_id):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()

        query = "select count(*) from orders where customerid = %s"
        cursor.execute(query, (customer_id,))
        count = cursor.fetchone()[0]

        return count

    except Exception as e:
        raise Exception(f"Error counting orders: {str(e)}")
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
```

```
[View Order Details]
Enter Order ID: 6

Order #6
Customer: Mohammed Aizen
Date: 2025-04-30 00:00
Status: pending
Total: AED22.00

Items:
Product Name:Illusionary Smart Glasses
 Quantity: 1
Product Price: AED 2200.00
 AED 22.00
```

• UpdateQuantity(): Allows updating the quantity of the product in this order detail.

```
def update_order_detail_quantity(self, order_detail_id, new_quantity):

    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()
        conn.autocommit = False

        get_query = """select productid, quantity, orderid
                    from orderdetails where orderdetailid = %s"""
        cursor.execute(get_query, (order_detail_id,))
        detail = cursor.fetchone()

        if not detail:
            raise OrderNotFoundException(f"Order detail with ID {order_detail_id} not found")

        product_id, old_quantity, order_id = detail
        quantity_diff = new_quantity - old_quantity


        stock_query = "select quantityinstock from inventory where productid = %s"
        cursor.execute(stock_query, (product_id,))
        stock = cursor.fetchone()[0]

        if quantity_diff > stock:
            raise InsufficientStockException(
```

```
            f"Not enough stock. Available: {stock}, Needed: {quantity_diff}")

        update_detail_query = """update orderdetails
                        set quantity = %s
                        where orderdetailid = %s"""
        cursor.execute(update_detail_query, (new_quantity, order_detail_id))

        update_inventory_query = """update inventory
                            set quantityinstock = quantityinstock - %s
                            where productid = %s"""
        cursor.execute(update_inventory_query, (quantity_diff, product_id))

        update_order_query = """update orders o
                        set totalamount = (
                        select sum(quantity * unitprice)
                        from orderdetails
                        where orderid = o.orderid)
                        where orderid = %s"""
        cursor.execute(update_order_query, (order_id,))

        conn.commit()
        return True

    except Exception as e:
        if conn:
            conn.rollback()
        raise
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
```

• AddDiscount(): Applies a discount to this order detail.

```
def apply_discount(self, percentage):
    if not 0 <= percentage <= 100:
        raise ValueError("Discount must be between 0-100%")
    self.__discount_percentage = percentage
    self.__subtotal = self.calculate_subtotal()
```

```
[Apply Discount to Order Item]
Enter Order Detail ID to discount: 9

Current Product: Shadow SSD
Current Price: 825.0
Current Quantity: 1
Current Subtotal: 825.0
Enter discount percentage (0-100): 50

Discount applied successfully!
New subtotal: 412.50
```

Inventory class:
Attributes:
• InventoryID(int)
• Product (Composition): The product associated with the inventory item.
• QuantityInStock: The quantity of the product currently in stock.
• LastStockUpdate

```
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
        self.__last_stock_update = datetime.now()
```

```
---------- Inventory Management ------------
1. View Product Stock
2. Add Stock
3. Remove Stock
4. Set Stock Quantity
5. Search Inventory
6. List Low Stock Items
7. Back to Main Menu

Enter your choice (1-7): 2

[Add Stock]
Enter Product ID: 2
Amount to add to stock: 5
 Stock updated. New quantity: 35
```

Methods:
• GetProduct(): A method to retrieve the product associated with this inventory item.

```
def get_product(self):
    return self.__product
```

```
[View Product Stock]
Enter Product ID: 5

Product Details:
ID: 5
Name: Infinity Projector
Category:  Projector
Current Stock: 2
```

• GetQuantityInStock(): A method to get the current quantity of the product in stock.

```
def get_quantity_in_stock(self):
    return self.__quantity_in_stock
```

```
[View Product Stock]
Enter Product ID: 5

Product Details:
ID: 5
Name: Infinity Projector
Category:  Projector
Current Stock: 2
```

• AddToInventory(int quantity): A method to add a specified quantity of the product to the inventory.

```
def add_to_inventory(self, quantity):
    if quantity <= 0:
        raise ValueError("Quantity must be positive")
    self.__quantity_in_stock += quantity
    self.__update_stock_time()
```

```
[Add Stock]
Enter Product ID: 2
Amount to add to stock: 20
 Stock updated. New quantity: 55
```

• RemoveFromInventory(int quantity): A method to remove a specified quantity of the product from the inventory.

```
def remove_from_inventory(self, quantity):
    if quantity <= 0:
        raise ValueError("Quantity must be positive")
    if quantity > self.__quantity_in_stock:
        raise ValueError("Insufficient stock")
    self.__quantity_in_stock -= quantity
    self.__update_stock_time()
```

```
[Remove Stock]
Enter Product ID: 9
Amount to remove from stock: 5
 Stock updated. New quantity: 95
```

• UpdateStockQuantity(int newQuantity): A method to update the stock quantity to a new value.

```
def update_stock(self, product_id, quantity_change):
    query = """ update inventory
    set quantityinstock = quantityinstock + %s,
    laststockupdate = current_timestamp
    where productid = %s """
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()

        if quantity_change < 0:
            current_stock = self.get_stock(product_id)
            if current_stock + quantity_change < 0:
                raise InsufficientStockException(
                    f"Cannot remove {-quantity_change} units. Only {current_stock} available."
                )

        cursor.execute(query, (quantity_change, product_id))
        conn.commit()
        return True
    except Exception as e:
        conn.rollback()
        raise Exception(f"Error updating stock: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()
```

```
[Add Stock]
Enter Product ID: 7
Amount to add to stock: 21
 Stock updated. New quantity: 22
```

• IsProductAvailable(int quantityToCheck): A method to check if a specified quantity of the product is available in the inventory.

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

• GetInventoryValue(): A method to calculate the total value of the products in the inventory based on their prices and quantities.

```
def get_stock(self, product_id):
    query = """ select quantityinstock
    from inventory
    where productid = %s """
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()

        cursor.execute(query, (product_id,))
        result = cursor.fetchone()

        if not result:
            raise ProductNotFoundException(f"No inventory record for product {product_id}")

        return result[0]
    except Exception as e:
        raise Exception(f"Error getting stock: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()
```

```
[Search Inventory]
Leave field blank to ignore it
Product ID:
Product name contains:
Minimum stock quantity: 1
Maximum stock quantity:
Show low stock only (y/n):

Search Results:
------------------------------------------------------------
ProductID Product Name             Category            Stock
------------------------------------------------------------
1         Shadow SSD               Storage Device      21
2         Hollow VR Headset        Wearable Technology 55
3         Tactical Smart Watch     Wearable Technology 13
4         Rasengan Drone           Drones              3
```

• ListLowStockProducts(int threshold): A method to list products with quantities below a specified threshold, indicating low stock.

```
def get_low_stock_items(self, threshold=5):
    try:
```

```
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor(dictionary=True)

        query = """
        select p.ProductID, p.ProductName, p.Category, i.QuantityInStock AS Quantity, i.LastStockUpdate AS
LastUpdated
        from Inventory i
        join Products p ON i.ProductID = p.ProductID
        where i.QuantityInStock <= %s
        order by i.QuantityInStock """
        cursor.execute(query, (threshold,))
        return cursor.fetchall()

    except Exception as e:
        raise Exception(f"Error getting low stock items: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()
```

```
[Low Stock Items]
Enter low stock threshold (5): 5

Products with stock below 5:
------------------------------------------------------------
ID    Product                   Category                Stock
------------------------------------------------------------
8     Tablet                    Electronics                 0
5     Infinity Projector         Projector                  2
6     Illusionary Smart Glasse Wearable Technology         2
4     Rasengan Drone            Drones                     3
```

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

```
def ListAllProducts(self):
    query = """ select i.inventoryid, i.productid, p.productname, p.category, i.quantityinstock,
i.laststockupdate
    from inventory i
    join products p on i.productid = p.productid """
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor(dictionary=True)

        cursor.execute(query)
        return cursor.fetchall()
    except Exception as e:
        raise Exception(f"Error retrieving inventory: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()
```

```
Enter your choice (1-7): 1

[View Product Stock]
Enter Product ID: 1

Product Details:
ID: 1
Name: Shadow SSD
Category: Storage Device
Current Stock: 21
```

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.

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).

Class Customer:

```
    def __init__(self, customer_id, first_name, last_name, email=None, phone=None, address=None):
        self.__customer_id = customer_id
        self.__orders = []
        self.first_name = first_name
        self.last_name = last_name
        if email is not None:
            self.email = email
        else:
            self.__email = ""

        if phone is not None:
            self.phone = phone
        else:
            self.__phone = ""

        if address is not None:
            self.address = address
        else:
            self.__address = ""

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

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

    @first_name.setter
    def first_name(self, value):
        if not isinstance(value, str) or len(value.strip()) == 0:
            raise ValueError("First name must be a non-empty string")
        self.__first_name = value.strip()

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

    @last_name.setter
    def last_name(self, value):
        if not isinstance(value, str) or len(value.strip()) == 0:
            raise ValueError("Last name must be a non-empty string")
        self.__last_name = value.strip()

    @property
```

```
def email(self):
    return self.__email

@email.setter
def email(self, value):
    if value == "":
        self.__email = value
    elif not self.__validate_email(value):
        raise ValueError("Invalid email format")
    else:
        self.__email = value.strip()

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

@phone.setter
def phone(self, value):
    if value == "":
        self.__phone = value
    else:
        self.__phone = value

@property
def address(self):
    return self.__address

@address.setter
def address(self, value):
    if value == "":
        self.__address = value
    elif not isinstance(value, str) or len(value.strip()) == 0:
        raise ValueError("Address must be a non-empty string")
    else:
        self.__address = value.strip()

@property
def orders(self):
    return self.__orders.copy()

# Validation methods
def __validate_email(self, email):

    if not isinstance(email, str):
        return False
    email = email.strip()
    return ('@' in email and
            '.' in email and
            len(email) > 5 and
            email.count('@') == 1 and
            email[0] != '@' and
            email[-1] != '@')


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

def get_customer_details(self,order_count=None):
    details= (f"Customer ID: {self.__customer_id}\n"
            f"Name: {self.__first_name} {self.__last_name}\n"
            f"Email: {self.__email}\n"
            f"Phone: {self.__phone}\n"
```

```
            f"Address: {self.__address}\n")
        if order_count is not None:
            details += f"\nTotal Orders: {order_count}"
        else:
            details += f"\nTotal Orders: {self.calculate_total_orders()}"

        return details

    def update_customer_info(self, first_name=None, last_name=None, email=None, phone=None, address=None):
        if first_name is not None:
            self.first_name = first_name
        if last_name is not None:
            self.last_name = last_name
        if email is not None:
            self.email = email
        if phone is not None:
            self.phone = phone
        if address is not None:
            self.address = address

    def add_order(self, order):
        if order not in self.__orders:
            self.__orders.append(order)

    def remove_order(self, order):
        if order in self.__orders:
            self.__orders.remove(order)
```

Class Product:

```
    def __init__(self, product_id, product_name, description, price, category):
        self.__product_id = product_id
        self.__product_name = product_name
        self.__description = description
        self.__price = price
        self.__category = category

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

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

    @product_name.setter
    def product_name(self, value):
        if not isinstance(value, str) or len(value.strip()) == 0:
            raise ValueError("Product name must be a non-empty string")
        self.__product_name = value.strip()

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

    @description.setter
    def description(self, value):
        self.__description = value

    @property
    def price(self):
```

```
        return self.__price

    @price.setter
    def price(self, value):
        if not isinstance(value, (int, float)) or value < 0:
            raise ValueError("Price must be a non-negative number")
        self.__price = value

    @property
    def category(self):
        return self.__category

    @category.setter
    def category(self, value):
        if not isinstance(value, str) or len(value.strip()) == 0:
            raise ValueError("Category must be a non-empty string")
        self.__category = value.strip()

    def get_product_details(self):
        return (f"Product ID: {self.__product_id}\n"
                f"Name: {self.__product_name}\n"
                f"Category: {self.__category}\n"
                f"Description: {self.__description}\n"
                f"Price: {self.__price}")

    def update_product_info(self, product_name=None, description=None, price=None, category=None):
        if product_name is not None:
            self.product_name = product_name
        if description is not None:
            self.description = description
        if price is not None:
            self.price = price
        if category is not None:
            self.category = category

from datetime import datetime
from entity.Customers import Customer
from exception.dataException import IncompleteOrderException
```

class Order:

```
    def __init__(self, order_id, customer, order_date=None, total_amount=0.0, status="Pending",version= 1):
        self.__order_id = order_id
        self.__customer = customer  # Composition with Customer
        self.__order_date = order_date if order_date else datetime.now()
        self.__total_amount = total_amount
        self.__status = status
        self.__order_details = []
        self.__version = version

    @property
    def version(self):
        return self.__version

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

    @property
    def customer(self):
        return self.__customer
```

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

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

    @property
    def status(self):
        return self.__status

    @property
    def order_details(self):
        return self.__order_details.copy()

    @status.setter
    def status(self, value):
        valid_statuses = ["Pending", "Processing", "Shipped", "Delivered", "Cancelled"]
        if value not in valid_statuses:
            raise ValueError(f"Invalid status. Must be one of: {valid_statuses}")
        self.__status = value

    def add_order_detail(self, order_detail):
        self.__order_details.append(order_detail)
        self.__total_amount = sum(detail.subtotal for detail in self.__order_details)

    def calculate_total_amount(self):
        return sum(detail.subtotal for detail in self.__order_details)

    def get_order_details(self):
        details = f"Order #{self.__order_id}\n"
        details += f"Customer: {self.__customer.first_name} {self.__customer.last_name}\n"
        details += f"Date: {self.__order_date.strftime('%Y-%m-%d %H:%M')}\n"
        details += f"Status: {self.__status}\n"
        details += f"Total: AED{self.__total_amount:.2f}\n\n"
        details += "Items:\n"
        for detail in self.__order_details:
            details += (f"Product Name:{detail.product.product_name}\n Quantity: {detail.quantity} \n"
                        f"Product Price: AED {detail.product.price:.2f}\n AED {detail.calculate_subtotal():.2f}\n")
        return details
```

```
from entity.Products import Product
```

class OrderDetail:

```
    def __init__(self, order_detail_id, order, product, quantity, unit_price):
        self.__order_detail_id = order_detail_id
        self.__order = order
        self.__product = product
        self.__quantity = quantity
        self.__unit_price = unit_price
        self.__discount_percentage = 0
        self.__subtotal = self.calculate_subtotal()

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

    @property
    def order(self):
        return self.__order
```

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

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

    @property
    def subtotal(self):
        return self.__subtotal

    @quantity.setter
    def quantity(self, value):
        if value <= 0:
            raise ValueError("Quantity must be positive")
        self.__quantity = value
        self.__subtotal = self.calculate_subtotal()

    @property
    def discount_percentage(self):
        return self.__discount_percentage

    @discount_percentage.setter
    def discount_percentage(self, value):
        if not 0 <= value <= 100:
            raise ValueError("Discount must be between 0-100%")
        self.__discount_percentage = value
        self.__subtotal = self.calculate_subtotal()

    @property
    def unit_price(self):
        return self.__unit_price

    def calculate_subtotal(self):
        base_price = self.__unit_price * self.__quantity
        return base_price * (1 - self.__discount_percentage/100)


    def apply_discount(self, percentage):
        if not 0 <= percentage <= 100:
            raise ValueError("Discount must be between 0-100%")
        self.__discount_percentage = percentage
        self.__subtotal = self.calculate_subtotal()
```

Inventory:

```
from datetime import datetime
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
        self.__last_stock_update = datetime.now()

    def get_product(self):
        return self.__product

    def get_quantity_in_stock(self):
        return self.__quantity_in_stock
```

```
def add_to_inventory(self, quantity):
    if quantity <= 0:
        raise ValueError("Quantity must be positive")
    self.__quantity_in_stock += quantity
    self.__update_stock_time()

def remove_from_inventory(self, quantity):
    if quantity <= 0:
        raise ValueError("Quantity must be positive")
    if quantity > self.__quantity_in_stock:
        raise ValueError("Insufficient stock")
    self.__quantity_in_stock -= quantity
    self.__update_stock_time()

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

def __update_stock_time(self):
    self.__last_stock_update = datetime.now()
```

Task 4: Composition:

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

1. Orders Class with Composition:

```
class Order:
    def __init__(self, order_id, customer, order_date=None, total_amount=0.0, status="Pending",version= 1):
        self.__order_id = order_id
        self.__customer = customer  # Composition with Customer
        self.__order_date = order_date if order_date else datetime.now()
        self.__total_amount = total_amount
        self.__status = status
        self.__order_details = []
        self.__version = version
```

2. OrderDetails Class with Composition:

```
class OrderDetail:
    def __init__(self, order_detail_id, order, product, quantity, unit_price):
        self.__order_detail_id = order_detail_id
        self.__order = order #Composition with order
        self.__product = product #composition with product
        self.__quantity = quantity
        self.__unit_price = unit_price
        self.__discount_percentage = 0
        self.__subtotal = self.calculate_subtotal()
```

3. Inventory Class:

```
class Inventory:
    def __init__(self, inventory_id, product, quantity_in_stock):
        self.__inventory_id = inventory_id
        self.__product = product #composition with product
        self.__quantity_in_stock = quantity_in_stock
        self.__last_stock_update = datetime.now()
```

Task 5: Exceptions handling

1:Data Validation

class InvalidDataException(Exception):
    """Raised when invalid data is provided"""
    pass

```
[Add New Customer]
First Name: ahm
Last Name: sher
Email: asjs
Phone: 12321424212
Address: wrwqr

An unexpected error occurred: Invalid email format
```

2:Inventory Management

class InsufficientStockException(Exception):
**   """Raised when there's not enough stock for a product"""**
**   pass**

```
Enter Product ID: 4
Enter Quantity: 213
Added 213x Rasengan Drone at AED 3200.00 each
Item subtotal: AED 681600.00

Current Order:
Items: 1
Total: AED681600.00

1. Add Product
2. Finalize Order
Select option: 2

 Error: Not enough stock for Rasengan Drone. Available: 3
```

3:Order Processing:

class IncompleteOrderException(Exception):
**   """Raised when order details are incomplete"""**
**   Pass**

```
1. Add Product
2. Finalize Order
Select option: 1
Enter Product ID:

 Invalid input. Please enter valid numbers.
```

4. Payment Processing:

class PaymentFailedException(Exception):
**   """Raised when payment processing fails"""**
**   pass**

```
[Process Payment]
Enter Order ID: 11
Payment Method (Credit/Debit/PayPal): PayPal
Payment Amount: 220

Payment Error: Cannot process payment for order in Paid status
```

5. File I/O (e.g., Logging):

class LoggingException(Exception):
**   """Raised when a there is an error in log entry """**

```
Error: Permission denied (errno 13)
Fallback: Logging to system console and /tmp/app_fallback.log
```

6. Database Access:

```
class SqlException(Exception):
    """Raised when there is an error in sql query"""
    Pass
```

```
CRITICAL: Cannot establish database connection
Error: Connection timed out (MySQL Server 5.7 not responding on port 3306)
```

7. Concurrency Control:

```
class ConcurrencyException(Exception):
    """Raised when concurrent modification is detected"""
    Pass
```

```
Error: Order #10042 was modified by another user while you were editing
```

Task 6: Collections

8. Managing Products List:

```
    def create(self, product):
        query = """ insert into products (productname, description, price, category)
        values (%s, %s, %s, %s) """
        try:
            conn = DBConnUtil.get_connection(self.__connection_string)
            cursor = conn.cursor()

            cursor.execute(query, (
                product.product_name,
                product.description,
                product.price,
                product.category
            ))

            cursor.execute("select last_insert_id()")
            product_id = cursor.fetchone()[0]

            cursor.execute("""
                insert into inventory (productid, quantityinstock)
                values (%s, 0)
            """, (product_id,))

            conn.commit()
            return product_id
        except Exception as e:
            conn.rollback()
            if "duplicate key" in str(e).lower():
                raise InvalidDataException("Product name already exists")
            raise Exception(f"Error creating product: {str(e)}")
        finally:
            if 'cursor' in locals():
                cursor.close()
            if 'conn' in locals():
```

```
            conn.close()
    def update(self, product):
        query = """ update products
        set productname = %s, description = %s, price = %s, category = %s
        where productid = %s """
        try:
            conn = DBConnUtil.get_connection(self.__connection_string)
            cursor = conn.cursor()

            cursor.execute(query, (
                product.product_name,
                product.description,
                product.price,
                product.category,
                product.product_id
            ))

            if cursor.rowcount == 0:
                raise ProductNotFoundException(f"Product with ID {product.product_id} not found")

            conn.commit()
            return True

        except Exception as e:
            conn.rollback()
            if "duplicate key" in str(e).lower():
                raise InvalidDataException("Product name already exists")
            raise Exception(f"Error updating product: {str(e)}")
        finally:
            if 'cursor' in locals():
                cursor.close()
            if 'conn' in locals():
                conn.close()

    def delete(self, product_id):
        check_query = "select count(*) from orderdetails where productid = %s"
        delete_query = "delete from products where productid = %s"

        try:
            conn = DBConnUtil.get_connection(self.__connection_string)
            cursor = conn.cursor()

            cursor.execute(check_query, (product_id,))
            if cursor.fetchone()[0] > 0:
                raise InvalidDataException("Cannot delete product with existing orders")

            cursor.execute(delete_query, (product_id,))

            if cursor.rowcount == 0:
                raise ProductNotFoundException(f"Product with ID {product_id} not found")

            conn.commit()
            return True

        except Exception as e:
            conn.rollback()
```

```
        raise Exception(f"Error deleting product: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()
```

```
[Add New Product]
Product Name: Ipad
Description: wrewr
Price: 32
Category: Electronics

Product created successfully! ID: 10
```

```
[Update Product Information]
Enter Product ID to update: 10

Current Details:
Product ID: 10
Name: Ipad
Category: Electronics
Description: wrewr
Price: 32.00

Enter new details (leave blank to keep current):
Name [Ipad]: Tab
Description [wrewr]:
Price [32.00]:
Category [Electronics]:

Product updated successfully!
```

```
[Delete Product]
Enter Product ID to delete: 10

Product deleted successfully!
```

9. Managing Orders List:

```
def create(self, order):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()
        conn.autocommit = False

        order_query = """ insert into orders (customerid, orderdate, totalamount, status)
        values (%s, %s, %s, %s) """
        cursor.execute(order_query, (
            order.customer.customer_id,
            order.order_date,
            order.total_amount,
            order.status
        ))
        order_id = cursor.lastrowid

        for detail in order.order_details:
            stock_query = "select quantityinstock from inventory where productid = %s"
            cursor.execute(stock_query, (detail.product.product_id,))
            stock = cursor.fetchone()[0]

            if stock < detail.quantity:
                raise InsufficientStockException(
                    f"Not enough stock for {detail.product.product_name}. Available: {stock}"
                )

            detail_query = """ insert into orderdetails (orderid, productid, quantity, unitprice)
            values (%s, %s, %s, %s) """
            cursor.execute(detail_query, (
                order_id,
                detail.product.product_id,
                detail.quantity,
                float(detail.product.price)
            ))

            update_query = """ update inventory
            set quantityinstock = quantityinstock - %s
            where productid = %s """
            cursor.execute(update_query, (
                detail.quantity,
```

```
                detail.product.product_id
            ))

        payment_query = """ insert into payments (orderid, amount, paymentmethod, status)
        values (%s, %s, %s, %s) """
        payment_status = "Completed" if order.total_amount >0 else "Failed"
        cursor.execute(payment_query, (
            order_id,
            order.total_amount,
            "Credit Card",
            payment_status
        ))

        if payment_status == "Failed":
            raise PaymentFailedException("Payment declined: Amount is not valid")

        conn.commit()
        return order_id


    except Exception as e:
        if conn:
            conn.rollback()
        raise
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
def update(self, order):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()
        conn.autocommit = False

        query = """ update orders
        set status = %s
        where orderid = %s """
        cursor.execute(query, (
            order.status,
            order.order_id,
        ))

        conn.commit()
        return True

    except ConcurrencyException as e:
        if conn:
            conn.rollback()
        raise
    except Exception as e:
        if conn:
            conn.rollback()
        raise Exception(f"Error updating order: {str(e)}")
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
```

```
def delete(self, order_id):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()
        conn.autocommit = False

        details_query = "select productid, quantity from orderdetails where orderid = %s"
        cursor.execute(details_query, (order_id,))
        details = cursor.fetchall()

        for product_id, quantity in details:
            restock_query = """ update inventory
            set quantityinstock = quantityinstock + %s
            where productid = %s """
            cursor.execute(restock_query, (quantity, product_id))

        delete_query = "delete from orders where orderid = %s"
        cursor.execute(delete_query, (order_id,))

        if cursor.rowcount == 0:
            raise OrderNotFoundException(f"Order with ID {order_id} not found")

        conn.commit()
        return True

    except Exception as e:
        if conn:
            conn.rollback()
        raise
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
```

```
1. Add Product
2. Finalize Order
Select option: 1
Enter Product ID: 2
Enter Quantity: 2
Added 2x Hollow VR Headset at AED 2530.00 each
Item subtotal: AED 5060.00

Current Order:
Items: 1
Total: AED5060.00

1. Add Product
2. Finalize Order
Select option: 2

 Order placed successfully! Order ID: 17
```

```
Enter your choice (1-8): 4

[Update Order Item Quantity]
Enter Order Detail ID to update: 2

Current Product: Hollow VR Headset
Current Quantity: 5
Enter new quantity: 2

Order item quantity updated successfully!
```

```
[Cancel Order]
Enter Order ID to cancel: 10

Order to cancel:
Order #10
Customer: Sungjinwoo Singh
Date: 2025-04-05 06:45
Status: Paid
Total: AED412.50

Items:
Product Name:Shadow SSD
 Quantity: 1
Product Price: AED 825.00
 AED 412.50


Are you sure you want to cancel this order? (y/n): y

 Order cancelled successfully. Inventory has been restocked.
```

10. Sorting Orders by Date:

```
def get_all(self, customer_id=None, status=None, start_date=None, end_date=None):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor(dictionary=True)

        query = """
        select o.OrderID, o.CustomerID, o.OrderDate, o.TotalAmount, o.Status,
        c.FirstName AS first_name,
```

```
        c.LastName AS last_name
        from orders o
        join customers c ON o.CustomerID = c.CustomerID
        where 1=1 """
        params = []

        if customer_id:
            query += " and o.CustomerID = %s"
            params.append(customer_id)
        if status:
            query += " and o.Status = %s"
            params.append(status)
        if start_date:
            query += " and o.OrderDate >= %s"
            params.append(start_date)
        if end_date:
            query += " and o.OrderDate <= %s"
            params.append(end_date)
```

```
Filter options (leave blank to ignore):
Customer ID:
Status (Pending/Processing/Shipped/Delivered/Cancelled/Paid/Refunded):
Start date (YYYY-MM-DD): 2025-03-24
End date (YYYY-MM-DD):

ID  Customer         Date         Status      Total
-------------------------------------------------------------------------------
6   Mohammed Aizen               2025-04-30  pending     AED22.00
17  Gojo Reddy                   2025-04-05  Pending     AED5060.00
15  Uzumaki Nair                 2025-04-05  Pending     AED1650.00
12  Sungjinwoo Singh             2025-04-05  Pending     AED825.00
11  Ichigo Kumar                 2025-04-05  Paid        AED50.00
9   Sungjinwoo Singh             2025-04-05  Paid        AED1650.00
3   Isagi Kahn                   2025-03-25  pending     AED200.00
```

11. Inventory Management with SortedList:

```
def search_inventory(self, product_id=None, product_name=None,
                     min_stock=None, max_stock=None, low_stock_only=False):
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor(dictionary=True)

        query = """select i.ProductID, p.ProductName, p.Category, i.QuantityInStock, i.LastStockUpdate
        from Inventory i
        join Products p ON i.ProductID = p.ProductID
        where 1=1 """
        params = []

        # Build query dynamically
        if product_id:
            query += " AND i.ProductID = %s"
            params.append(product_id)

        if product_name:
            query += " AND p.ProductName LIKE %s"
            params.append(f"%{product_name}%")

        if min_stock is not None:
            query += " AND i.QuantityInStock >= %s"
            params.append(min_stock)

        if max_stock is not None:
            query += " AND i.QuantityInStock <= %s"
            params.append(max_stock)
```

```
[Search Inventory]
Leave field blank to ignore it
Product ID:
Product name contains:
Minimum stock quantity:
Maximum stock quantity:
Show low stock only (y/n):

Search Results:
-------------------------------------------------------------------------------
ProductID Product Name            Category           Stock    Last Updated
-------------------------------------------------------------------------------
1         Shadow SSD              Storage Device     20       2025-04-05 03:51
2         Hollow VR Headset       Wearable Technology 56      2025-04-05 14:08
3         Tactical Smart Watch    Wearable Technology 13      2025-04-05 03:51
4         Rasengan Drone          Drones             3        2025-04-05 03:51
```

```
        if low_stock_only:
            query += " AND i.QuantityInStock < 5"  # Assuming 5 is low stock threshold

        cursor.execute(query, params)
        return cursor.fetchall()

    except Exception as e:
        raise Exception(f"Error searching inventory: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()
```

12. Handling Inventory Updates:

```
def delete(self, order_id):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()
        conn.autocommit = False

        details_query = "select productid, quantity from orderdetails
where orderid = %s"
        cursor.execute(details_query, (order_id,))
        details = cursor.fetchall()

        for product_id, quantity in details:
            restock_query = """ update inventory
            set quantityinstock = quantityinstock + %s
            where productid = %s """
            cursor.execute(restock_query, (quantity, product_id))
```

```
Order to cancel:
Order #17
Customer: Gojo Reddy
Date: 2025-04-05 15:44
Status: Pending
Total: AED5060.00

Items:
Product Name:Hollow VR Headset
 Quantity: 2
Product Price: AED 2530.00
 AED 5060.00


Are you sure you want to cancel this order? (y/n): y

 Order cancelled successfully. Inventory has been restocked.
```

13. Product Search and Retrieval:

```
def search_products(self, id=None, name=None, category=None, min_price=None, max_price=None):
    query = "select * from products where 1=1"
    params = []

    if id:
        query += " and productid = %s"
        params.append(int(id))
    if name:
        query += " and productname like %s"
        params.append(f"%{name}%")
    if category:
        query += " and category = %s"
        params.append(category)
    if min_price is not None:
        query += " and price >= %s"
        params.append(min_price)
    if max_price is not None:
        query += " and price <= %s"
        params.append(max_price)

    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()
        cursor.execute(query, params)
```

```
[Search Products]
Leave field blank to ignore it
Product id:
Product name contains:
Category: Electronics
Minimum price:
Maximum price:
```

```
        return [
            Product(
                product_id=row[0],
                product_name=row[1],
                description=row[2],
                price=row[3],
                category=row[4]
            ) for row in cursor.fetchall()
        ]
    except Exception as e:
        raise Exception(f"Error searching products: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()
```

14. Payment Records List:

```
def get_payment_details(self, order_id):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor(dictionary=True)

        query = """ select p.*
        from payments p
        where p.orderid = %s
        order by p.orderid
        limit 1 """
        cursor.execute(query, (order_id,))
        payment_data = cursor.fetchone()

        if not payment_data:
            raise PaymentFailedException(f"No payment found for order {order_id}")

        return {
            'payment_id': payment_data['PaymentID'],
            'order_id': payment_data['OrderID'],
            'amount': float(payment_data['Amount']),
            'method': payment_data['PaymentMethod'],
            'status': payment_data['Status']
        }

    except Exception as e:
        if isinstance(e, PaymentFailedException):
            raise
        raise Exception(f"Error retrieving payment details: {str(e)}")
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
```

```
[View Payment Details]
Enter Order ID: 5

Payment Details:
----------------------------
Payment ID: 5
Order ID: 5
Amount: AED39.00
Method: Credit Card
Status: Pending
```

15. OrderDetails and Products Relationship:

```
def update_order_detail_quantity(self, order_detail_id, new_quantity):
```

```
conn = None
cursor = None
try:
    conn = DBConnUtil.get_connection(self.__connection_string)
    cursor = conn.cursor()
    conn.autocommit = False

    get_query = """select productid, quantity, orderid
                from orderdetails where orderdetailid = %s"""
    cursor.execute(get_query, (order_detail_id,))
    detail = cursor.fetchone()

    if not detail:
        raise OrderNotFoundException(f"Order detail with ID {order_detail_id} not found")

    product_id, old_quantity, order_id = detail
    quantity_diff = new_quantity - old_quantity


    stock_query = "select quantityinstock from inventory where productid = %s"
    cursor.execute(stock_query, (product_id,))
    stock = cursor.fetchone()[0]

    if quantity_diff > stock:
        raise InsufficientStockException(
            f"Not enough stock. Available: {stock}, Needed: {quantity_diff}")

    update_detail_query = """update orderdetails
                        set quantity = %s
                        where orderdetailid = %s"""
    cursor.execute(update_detail_query, (new_quantity, order_detail_id))

    update_inventory_query = """update inventory
                            set quantityinstock = quantityinstock - %s
                            where productid = %s"""
    cursor.execute(update_inventory_query, (quantity_diff, product_id))

    update_order_query = """update orders o
                        set totalamount = (
                        select sum(quantity * unitprice)
                        from orderdetails
                        where orderid = o.orderid)
                        where orderid = %s"""
    cursor.execute(update_order_query, (order_id,))

    conn.commit()
    return True

except Exception as e:
    if conn:
        conn.rollback()
    raise
finally:
    if cursor:
        cursor.close()
    if conn:
        conn.close()
```

```
Enter Product ID: 4
Enter Quantity: 213
Added 213x Rasengan Drone at AED 3200.00 each
Item subtotal: AED 681600.00

Current Order:
Items: 1
Total: AED681600.00

1. Add Product
2. Finalize Order
Select option: 2

 Error: Not enough stock for Rasengan Drone. Available: 3
```

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.

```
import mysql.connector
from exception.dataException import DatabaseConnectionException


class DBConnUtil:
    @staticmethod
    def get_connection(connection_string):
        try:
            params = {}
            for item in connection_string.split():
                if '=' in item:
                    key, value = item.split('=', 1)
                    params[key] = value

            conn = mysql.connector.connect(
                host=params['host'],
                database=params['dbname'],
                user=params['user'],
                password=params['password'],
                port=params.get('port', '3306')
            )
            return conn

        except mysql.connector.Error as e:
            raise DatabaseConnectionException(f"MySQL Connection Error: {str(e)}")
```

CRUD Operations:

Customer:

```
from Dao.ordrs import ServiceProvider
from entity.Customers import Customer
from util.db_conn_util import DBConnUtil
from util.db_property_util import DBPropertyUtil
from exception.dataException import InvalidDataException, CustomerNotFoundException

class CustomerDAO(ServiceProvider):
    def __init__(self):
        self.__connection_string = DBPropertyUtil.get_connection_string("db.properties")

#__________________CRUD Operations_____________________
    def create(self, customer):
        query = """insert into customers (firstname,lastname, email,phone, address)
        values (%s, %s, %s, %s, %s)"""
        try:
            conn = DBConnUtil.get_connection(self.__connection_string)
            cursor = conn.cursor()

            cursor.execute(query, (
                customer.first_name,
                customer.last_name,
                customer.email,
                customer.phone,
                customer.address
            ))
```

```
            cursor.execute("select last_insert_id()")
            customer_id = cursor.fetchone()[0]

            conn.commit()
            return customer_id

        except Exception as e:
            conn.rollback()
            if "duplicate key" in str(e).lower():
                raise InvalidDataException("Email already exists")
            raise Exception(f"Error creating customer: {str(e)}")
        finally:
            if 'cursor' in locals():
                cursor.close()
            if 'conn' in locals():
                conn.close()
def get_all(self):
    query = """select customerid, firstname,lastname, email, phone, address
    from customers"""
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()

        cursor.execute(query)
        records = cursor.fetchall()

        customers = []
        for record in records:
            customer = Customer(
                customer_id=record[0],
                first_name=record[1],
                last_name=record[2],
                email=record[3],
                phone=record[4],
                address=record[5]
            )
            customers.append(customer)
        return customers

    except Exception as e:
        raise Exception(f"Error retrieving customers: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()

def get_all_with_order_counts(self, order_dao):
    customers = self.get_all()
    for customer in customers:
        try:
            order_count = order_dao.count_orders_by_customer(customer.customer_id)
            customer.order_count = order_count
        except Exception:
            customer.order_count = 0
    return customers
def update(self, customer):
    query = """ update customers set firstname= %s, lastname = %s, email=%s, phone= %s, address= %s
    where customerid = %s """
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()
```

```
            cursor.execute(query, (
                customer.first_name,
                customer.last_name,
                customer.email,
                customer.phone,
                customer.address,
                customer.customer_id
            ))

            if cursor.rowcount == 0:
                raise CustomerNotFoundException(f"Customer with ID {customer.customer_id} not found")

            conn.commit()
            return True

        except Exception as e:
            conn.rollback()
            if "duplicate key" in str(e).lower():
                raise InvalidDataException("Email already exists")
            raise Exception(f"Error updating customer: {str(e)}")
        finally:
            if 'cursor' in locals():
                cursor.close()
            if 'conn' in locals():
                conn.close()

    def delete(self, customer_id):
        query = "delete from customers where customerid= %s"
        try:
            conn = DBConnUtil.get_connection(self.__connection_string)
            cursor = conn.cursor()

            cursor.execute(query, (customer_id,))

            if cursor.rowcount == 0:
                raise CustomerNotFoundException(f"Customer with ID {customer_id} not found")

            conn.commit()
            return True

        except Exception as e:
            conn.rollback()
            raise Exception(f"Error deleting customer: {str(e)}")
        finally:
            if 'cursor' in locals():
                cursor.close()
            if 'conn' in locals():
                conn.close()
```

Order and Order Details:

```
from Dao.ordrs import ServiceProvider
from decimal import Decimal
from entity.Orders import Order
from entity.Customers import Customer
from entity.Products import Product
from entity.OrderDetails import OrderDetail
from util.db_conn_util import DBConnUtil
from util.db_property_util import DBPropertyUtil
from exception.dataException import (IncompleteOrderException, PaymentFailedException,OrderNotFoundException,
InsufficientStockException,ConcurrencyException)
```

```
class OrderDAO(ServiceProvider):
    def __init__(self):
        self.__connection_string = DBPropertyUtil.get_connection_string("db.properties")

    def create(self, order):
        conn = None
        cursor = None
        try:
            conn = DBConnUtil.get_connection(self.__connection_string)
            cursor = conn.cursor()
            conn.autocommit = False

            order_query = """ insert into orders (customerid, orderdate, totalamount, status)
            values (%s, %s, %s, %s) """
            cursor.execute(order_query, (
                order.customer.customer_id,
                order.order_date,
                order.total_amount,
                order.status
            ))
            order_id = cursor.lastrowid

            for detail in order.order_details:
                stock_query = "select quantityinstock from inventory where productid = %s"
                cursor.execute(stock_query, (detail.product.product_id,))
                stock = cursor.fetchone()[0]

                if stock < detail.quantity:
                    raise InsufficientStockException(
                        f"Not enough stock for {detail.product.product_name}. Available: {stock}"
                    )

                detail_query = """ insert into orderdetails (orderid, productid, quantity, unitprice)
                values (%s, %s, %s, %s) """
                cursor.execute(detail_query, (
                    order_id,
                    detail.product.product_id,
                    detail.quantity,
                    float(detail.product.price)
                ))

                update_query = """ update inventory
                set quantityinstock = quantityinstock - %s
                where productid = %s """
                cursor.execute(update_query, (
                    detail.quantity,
                    detail.product.product_id
                ))

            payment_query = """ insert into payments (orderid, amount, paymentmethod, status)
            values (%s, %s, %s, %s) """
            payment_status = "Completed" if order.total_amount >0 else "Failed"
            cursor.execute(payment_query, (
                order_id,
                order.total_amount,
                "Credit Card",
                payment_status
            ))

            if payment_status == "Failed":
                raise PaymentFailedException("Payment declined: Amount is not valid")
```

```
            conn.commit()
            return order_id


        except Exception as e:
            if conn:
                conn.rollback()
            raise
        finally:
            if cursor:
                cursor.close()
            if conn:
                conn.close()
def update(self, order):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()
        conn.autocommit = False

        query = """ update orders
        set status = %s
        where orderid = %s """
        cursor.execute(query, (
            order.status,
            order.order_id,
        ))

        conn.commit()
        return True

    except ConcurrencyException as e:
        if conn:
            conn.rollback()
        raise
    except Exception as e:
        if conn:
            conn.rollback()
        raise Exception(f"Error updating order: {str(e)}")
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()

def delete(self, order_id):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()
        conn.autocommit = False

        details_query = "select productid, quantity from orderdetails where orderid = %s"
        cursor.execute(details_query, (order_id,))
        details = cursor.fetchall()

        for product_id, quantity in details:
            restock_query = """ update inventory
            set quantityinstock = quantityinstock + %s
            where productid = %s """
```

```
            cursor.execute(restock_query, (quantity, product_id))

        delete_query = "delete from orders where orderid = %s"
        cursor.execute(delete_query, (order_id,))

        if cursor.rowcount == 0:
            raise OrderNotFoundException(f"Order with ID {order_id} not found")

        conn.commit()
        return True

    except Exception as e:
        if conn:
            conn.rollback()
        raise
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()

def get_all(self, customer_id=None, status=None, start_date=None, end_date=None):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor(dictionary=True)

        query = """
        select o.OrderID, o.CustomerID, o.OrderDate, o.TotalAmount, o.Status,
        c.FirstName AS first_name,
        c.LastName AS last_name
        from orders o
        join customers c ON o.CustomerID = c.CustomerID
        where 1=1 """
        params = []

        if customer_id:
            query += " and o.CustomerID = %s"
            params.append(customer_id)
        if status:
            query += " and o.Status = %s"
            params.append(status)
        if start_date:
            query += " and o.OrderDate >= %s"
            params.append(start_date)
        if end_date:
            query += " and o.OrderDate <= %s"
            params.append(end_date)

        query += " order by o.OrderDate DESC"
        cursor.execute(query, params)

        orders = []
        for order_data in cursor.fetchall():
            customer = Customer(
                order_data['CustomerID'],
                order_data['first_name'],
                order_data['last_name']

            )
```

```
            order = Order(
                order_data['OrderID'],
                customer,
                order_data['OrderDate'],
                float(order_data['TotalAmount']),
                order_data['Status']
            )
            orders.append(order)

        return orders

    except Exception as e:
        raise Exception(f"Error retrieving orders: {str(e)}")
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
```

Product:

```
    from Dao.ordrs import ServiceProvider
    from entity.Products import Product
    from util.db_conn_util import DBConnUtil
    from util.db_property_util import DBPropertyUtil
    from exception.dataException import InvalidDataException, ProductNotFoundException

    class ProductDAO(ServiceProvider):
        def __init__(self):
            self.__connection_string = DBPropertyUtil.get_connection_string("db.properties")

        def create(self, product):
            query = """ insert into products (productname, description, price, category)
            values (%s, %s, %s, %s) """
            try:
                conn = DBConnUtil.get_connection(self.__connection_string)
                cursor = conn.cursor()

                cursor.execute(query, (
                    product.product_name,
                    product.description,
                    product.price,
                    product.category
                ))

                cursor.execute("select last_insert_id()")
                product_id = cursor.fetchone()[0]

                cursor.execute("""
                    insert into inventory (productid, quantityinstock)
                    values (%s, 0)
                """, (product_id,))

                conn.commit()
                return product_id
            except Exception as e:
                conn.rollback()
                if "duplicate key" in str(e).lower():
                    raise InvalidDataException("Product name already exists")
                raise Exception(f"Error creating product: {str(e)}")
            finally:
                if 'cursor' in locals():
```

```
            cursor.close()
        if 'conn' in locals():
            conn.close()

def get_all(self):
    query = """
    select *
    from products
    """
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()

        cursor.execute(query)
        return [
            Product(
                product_id=row[0],
                product_name=row[1],
                description=row[2],
                price=row[3],
                category=row[4]
            ) for row in cursor.fetchall()
        ]

    except Exception as e:
        raise Exception(f"Error retrieving products: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()

def update(self, product):
    query = """ update products
    set productname = %s, description = %s, price = %s, category = %s
    where productid = %s """
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()

        cursor.execute(query, (
            product.product_name,
            product.description,
            product.price,
            product.category,
            product.product_id
        ))

        if cursor.rowcount == 0:
            raise ProductNotFoundException(f"Product with ID {product.product_id} not found")

        conn.commit()
        return True

    except Exception as e:
        conn.rollback()
        if "duplicate key" in str(e).lower():
            raise InvalidDataException("Product name already exists")
        raise Exception(f"Error updating product: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
```

```
        if 'conn' in locals():
            conn.close()

def delete(self, product_id):
    check_query = "select count(*) from orderdetails where productid = %s"
    delete_query = "delete from products where productid = %s"

    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()

        cursor.execute(check_query, (product_id,))
        if cursor.fetchone()[0] > 0:
            raise InvalidDataException("Cannot delete product with existing orders")

        cursor.execute(delete_query, (product_id,))

        if cursor.rowcount == 0:
            raise ProductNotFoundException(f"Product with ID {product_id} not found")

        conn.commit()
        return True

    except Exception as e:
        conn.rollback()
        raise Exception(f"Error deleting product: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()
```

Inventory:

```
from Dao.ordrs import ServiceProvider
from util.db_conn_util import DBConnUtil
from util.db_property_util import DBPropertyUtil
from exception.dataException import InvalidDataException, InsufficientStockException, ProductNotFoundException

class InventoryDAO(ServiceProvider):
    def __init__(self):
        self.__connection_string = DBPropertyUtil.get_connection_string("db.properties")

    def create(self, inventory_item):
        query = """ insert into inventory (productid, quantityinstock)
        values (%s, %s) """
        try:
            conn = DBConnUtil.get_connection(self.__connection_string)
            cursor = conn.cursor()

            cursor.execute(query, (
                inventory_item['product_id'],
                inventory_item['quantity']
            ))

            cursor.execute("select last_insert_id()")
            inventory_id = cursor.fetchone()[0]
            conn.commit()
            return inventory_id
        except Exception as e:
            conn.rollback()
            if "foreign key constraint" in str(e).lower():
```

```
            raise ProductNotFoundException("Product does not exist")
        raise Exception(f"Error creating inventory record: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()
def get_all(self):
    query = """ select i.inventoryid, i.productid, p.productname, p.category, i.quantityinstock, i.laststockupdate
    from inventory i
    join products p on i.productid = p.productid """
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor(dictionary=True)

        cursor.execute(query)
        return cursor.fetchall()
    except Exception as e:
        raise Exception(f"Error retrieving inventory: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()

def update(self, inventory_item):
    query = """ update inventory
    set productid = %s, quantityinstock = %s
    where inventoryid = %s """
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()

        cursor.execute(query, (
            inventory_item['product_id'],
            inventory_item['quantity'],
            inventory_item['inventory_id']
        ))

        if cursor.rowcount == 0:
            raise Exception("No inventory record was updated")

        conn.commit()
        return True
    except Exception as e:
        conn.rollback()
        raise Exception(f"Error updating inventory: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()

def delete(self, inventory_id):
    query = "delete from inventory where inventoryid = %s"
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()

        cursor.execute(query, (inventory_id,))

        if cursor.rowcount == 0:
```

```
            raise Exception("No inventory record was deleted")

        conn.commit()
        return True
    except Exception as e:
        conn.rollback()
        raise Exception(f"Error deleting inventory: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()
```

DATABASE CONNECTIVITY:

1. Customer Registration:

```
from Dao.ordrs import ServiceProvider
from entity.Customers import Customer
from util.db_conn_util import DBConnUtil
from util.db_property_util import DBPropertyUtil
from exception.dataException import InvalidDataException, CustomerNotFoundException

class CustomerDAO(ServiceProvider):
    def __init__(self):
        self.__connection_string = DBPropertyUtil.get_connection_string("db.properties")
```

```
ID  Name            Email           Orders
-----------------------------------------------------------------
1   Sungjinwoo Singh                jinwoo@gmail.com        3
2   Ichigo Kumar                kurosaki.ichigo@gmai    2
3   Isagi Kahn                  clown@gmail.com         1
4   Uzumaki Nair                    boruto.nair@gmail.co    2
5   Gojo Reddy                  gojo.suguru@gmail.co    1
6   Mohammed Aizen                  Aizen.watashi@gmail.    1
7   Ahmed Sherif                ahmedashiq2k17@gmail    0
12  Nameless Wochester              Blanks@gmail.com        0
```

CustomerID	FirstName	LastName	Email	Phone	Address
1	Sungjinwoo	Singh	jinwoo@gmail.com	9471823912	321 Aura Farm
2	Ichigo	Kumar	kurosaki.ichigo@gmail.com	9184728248	123 Bankai Ave
3	Isagi	Kahn	clown@gmail.com	9432109876	1 Strika St
4	Uzumaki	Nair	boruto.nair@gmail.com	9321098765	106 Sasuke Blvd
5	Gojo	Reddy	gojo.suguru@gmail.com	9420987654	753 Shibuya Ave
6	Mohammed	Aizen	Aizen.watashi@gmail.com	9105576543	852 Yokoso Rd
7	Ahmed	Sherif	ahmedashiq2k17@gmail.com	9827481829	11, Yoruichi Ave
12	Nameless	Wochester	Blanks@gmail.com	9824753723	Erd Tree Ave

2: Product Catalog Management

```
from Dao.ordrs import ServiceProvider
from entity.Products import Product
from util.db_conn_util import DBConnUtil
from util.db_property_util import DBPropertyUtil
from exception.dataException import InvalidDataException, ProductNotFoundException

class ProductDAO(ServiceProvider):
    def __init__(self):
        self.__connection_string = DBPropertyUtil.get_connection_string("db.properties")
```

```
ID  Name    Description Category    Price
-----------------------------------------------------------
1   Shadow SSD      A high-speed 1TB SSD with rapi  Storage Device          825.00
2   Hollow VR Heads A VR headset with immersive au  Wearable Techno         2530.00
3   Tactical Smart  Advanced sports analyzing watc  Wearable Techno         1700.00
4   Rasengan Drone  High-speed drone with rotor bl  Drones                  3200.00
5   Infinity Projec Projector with limitless focus   Projector              1980.00
6   Illusionary Sma Smart glasses with holographic  Wearable Techno         2200.00
7   Laptop          High Processing Gaming Laptop   Electronics             50.00
8   Tablet          Foldable Tablet                 Electronics             100.00
9   IPad            Super fast gaming Ipad          Electronics             350.00
```

ProductID	ProductName	Description	Price	Category
1	Shadow SSD	A high-speed 1TB SSD with rapid data transfer	825.00	Storage Device
2	Hollow VR Headset	A VR headset with immersive audio and visuals	2530.00	Wearable Technology
3	Tactical Smart Watch	Advanced sports analyzing watch	1700.00	Wearable Technology
4	Rasengan Drone	High-speed drone with rotor blades	3200.00	Drones
5	Infinity Projector	Projector with limitless focus and crystal-clear vi...	1980.00	Projector
6	Illusionary Smart Glasses	Smart glasses with holographic displays	2200.00	Wearable Technology
7	Laptop	High Processing Gaming Laptop	50.00	Electronics
8	Tablet	Foldable Tablet	100.00	Electronics
9	IPad	Super fast gaming Ipad	350.00	Electronics

3: Placing Customer Orders

```
from Dao.ordrs import ServiceProvider
from decimal import Decimal
from entity.Orders import Order
from entity.Customers import Customer
from entity.Products import Product
from entity.OrderDetails import OrderDetail
from util.db_conn_util import DBConnUtil
from util.db_property_util import DBPropertyUtil
from exception.dataException import (IncompleteOrderException, PaymentFailedException,OrderNotFoundException,
InsufficientStockException,ConcurrencyException)

class OrderDAO(ServiceProvider):
    def __init__(self):
        self.__connection_string = DBPropertyUtil.get_connection_string("db.properties")
```

```
ID  Customer        Date        Status      Total
--------------------------------------------------------------------
6   Mohammed Aizen              2025-04-30  pending     AED22.00
15  Uzumaki Nair                2025-04-05  Pending     AED1650.00
12  Sungjinwoo Singh            2025-04-05  Pending     AED825.00
11  Ichigo Kumar                2025-04-05  Paid        AED50.00
9   Sungjinwoo Singh            2025-04-05  Paid        AED1650.00
3   Isagi Kahn                  2025-03-25  pending     AED200.00
5   Gojo Reddy                  2025-03-19  pending     AED39.00
4   Uzumaki Nair                2025-03-15  Pending     AED25.00
1   Sungjinwoo Singh            2025-02-05  pending     AED200.00
2   Ichigo Kumar                2025-01-28  pending     AED200.00
```

OrderID	CustomerID	OrderDate	TotalAmount	Status
1	1	2025-02-05 00:00:00	200.00	pending
2	2	2025-01-28 00:00:00	200.00	pending
3	3	2025-03-25 00:00:00	200.00	pending
4	4	2025-03-15 00:00:00	25.00	Pending
5	5	2025-03-19 00:00:00	39.00	pending
6	6	2025-04-30 00:00:00	22.00	pending
9	1	2025-04-05 06:17:59	1650.00	Paid
11	2	2025-04-05 07:01:44	50.00	Paid
12	1	2025-04-05 10:51:34	825.00	Pending
15	4	2025-04-05 15:07:19	1650.00	Pending

4: Tracking Order Status

```
def get_all(self, customer_id=None, status=None, start_date=None, end_date=None):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor(dictionary=True)

        query = """
        select o.OrderID, o.CustomerID, o.OrderDate, o.TotalAmount, o.Status,
        c.FirstName AS first_name,
        c.LastName AS last_name
        from orders o
        join customers c ON o.CustomerID = c.CustomerID
        where 1=1 """
        params = []

        if customer_id:
            query += " and o.CustomerID = %s"
            params.append(customer_id)
        if status:
            query += " and o.Status = %s"
            params.append(status)
        if start_date:
            query += " and o.OrderDate >= %s"
            params.append(start_date)
        if end_date:
            query += " and o.OrderDate <= %s"
            params.append(end_date)

        query += " order by o.OrderDate DESC"
        cursor.execute(query, params)

        orders = []
```

```
[View Order Details]
Enter Order ID: 2

Order #2
Customer: Ichigo Kumar
Date: 2025-01-28 00:00
Status: pending
Total: AED200.00
```

OrderID	CustomerID	OrderDate	TotalAmount	Status
1	1	2025-02-05 00:00:00	200.00	pending
2	2	2025-01-28 00:00:00	200.00	pending
3	3	2025-03-25 00:00:00	200.00	pending
4	4	2025-03-15 00:00:00	25.00	Pending
5	5	2025-03-19 00:00:00	39.00	pending
6	6	2025-04-30 00:00:00	22.00	pending
9	1	2025-04-05 06:17:59	1650.00	Paid
11	2	2025-04-05 07:01:44	50.00	Paid
12	1	2025-04-05 10:51:34	825.00	Pending
15	4	2025-04-05 15:07:19	1650.00	Pending

```
        for order_data in cursor.fetchall():
            customer = Customer(
                order_data['CustomerID'],
                order_data['first_name'],
                order_data['last_name']

            )

            order = Order(
                order_data['OrderID'],
                customer,
                order_data['OrderDate'],
                float(order_data['TotalAmount']),
                order_data['Status']
            )
            orders.append(order)

        return orders

    except Exception as e:
        raise Exception(f"Error retrieving orders: {str(e)}")
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
```

5: Inventory Management

```
from Dao.ordrs import ServiceProvider
from util.db_conn_util import DBConnUtil
from util.db_property_util import DBPropertyUtil
from exception.dataException import InvalidDataException,InsufficientStockException,ProductNotFoundException
class InventoryDAO(ServiceProvider):
    def __init__(self):
        self.__connection_string = DBPropertyUtil.get_connection_string("db.properties")
```

InventoryID	ProductID	QuantityInStock	LastStockUpdate
1	1	20	2025-04-05 03:51:24
2	2	56	2025-04-05 15:56:52
3	3	13	2025-04-05 03:51:24
4	4	3	2025-04-05 03:51:24
5	5	2	2025-04-05 03:51:24
6	6	2	2025-04-05 03:51:24
7	7	22	2025-04-05 14:10:11
8	8	0	2025-04-05 11:55:28
9	9	95	2025-04-05 14:09:06
10	10	0	2025-04-05 15:37:33

ProductID	Product Name	Category	Stock	Last Updated
1	Shadow SSD	Storage Device	20	2025-04-05 03:51
2	Hollow VR Headset	Wearable Technology	56	2025-04-05 15:56
3	Tactical Smart Watch	Wearable Technology	13	2025-04-05 03:51
4	Rasengan Drone	Drones	3	2025-04-05 03:51
5	Infinity Projector	Projector	2	2025-04-05 03:51
6	Illusionary Smart Glasse	Wearable Technology	2	2025-04-05 03:51
7	Laptop	Electronics	22	2025-04-05 14:10
8	Tablet	Electronics	0	2025-04-05 11:55
9	IPad	Electronics	95	2025-04-05 14:09

6:Sales Reporting

```
def get_all(self, customer_id=None, status=None, start_date=None, end_date=None):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
```

```
    cursor = conn.cursor(dictionary=True)

    query = """
    select o.OrderID, o.CustomerID, o.OrderDate, o.TotalAmount, o.Status,
    c.FirstName AS first_name,
    c.LastName AS last_name
    from orders o
    join customers c ON o.CustomerID = c.CustomerID
    where 1=1 """
    params = []

    if customer_id:
        query += " and o.CustomerID = %s"
        params.append(customer_id)
    if status:
        query += " and o.Status = %s"
        params.append(status)
    if start_date:
        query += " and o.OrderDate >= %s"
        params.append(start_date)
    if end_date:
        query += " and o.OrderDate <= %s"
        params.append(end_date)

    query += " order by o.OrderDate DESC"
    cursor.execute(query, params)

    orders = []
    for order_data in cursor.fetchall():
        customer = Customer(
            order_data['CustomerID'],
            order_data['first_name'],
            order_data['last_name']

        )

        order = Order(
            order_data['OrderID'],
            customer,
            order_data['OrderDate'],
            float(order_data['TotalAmount']),
            order_data['Status']
        )
        orders.append(order)

    return orders

except Exception as e:
    raise Exception(f"Error retrieving orders: {str(e)}")
finally:
    if cursor:
        cursor.close()
    if conn:
        conn.close()
```

OrderID	CustomerID	OrderDate	TotalAmount	Status
1	1	2025-02-05 00:00:00	200.00	pending
2	2	2025-01-28 00:00:00	200.00	pending
3	3	2025-03-25 00:00:00	200.00	pending
4	4	2025-03-15 00:00:00	25.00	Pending
5	5	2025-03-19 00:00:00	39.00	pending
6	6	2025-04-30 00:00:00	22.00	pending
9	1	2025-04-05 06:17:59	1650.00	Paid
11	2	2025-04-05 07:01:44	50.00	Paid
12	1	2025-04-05 10:51:34	825.00	Pending
15	4	2025-04-05 15:07:19	1650.00	Pending

7: Customer Account Updates

```
def get_customer_by_email(self, email):
    query = """ select *
    from customers where email = %s """
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()

        cursor.execute(query, (email,))
        record = cursor.fetchone()

        if record is None:
            raise CustomerNotFoundException(f"Customer with email {email} not found")

        customer = Customer(
            customer_id=record[0],
            first_name=record[1],
            last_name=record[2],
            email=record[3],
            phone=record[4],
            address=record[5]
        )
        return customer

    except Exception as e:
        raise Exception(f"Error retrieving customer by email: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()
```

CustomerID	FirstName	LastName	Email	Phone	Address
3	Isagi	Kahn	fineclown@mgail.com	9432109876	1 Strika St

CustomerID	FirstName	LastName	Email	Phone	Address
3	Isagi	Kahn	clown@gmail.com	9432109876	1 Strika St

```
Current Details:
Customer ID: 3
Name: Isagi Kahn
Email: clown@gmail.com
Phone: 9432109876
Address: 1 Strika St

Total Orders: 0

Enter new details (leave blank to keep current):
First Name [Isagi]:
Last Name [Kahn]:
Email [clown@gmail.com]: fineclown@mgail.com
Phone [9432109876]:
Address [1 Strika St]:

 Customer updated successfully!
```

```
def get_by_id(self, customer_id, include_order_count=False, order_dao=None):
    query = """select customerid, firstname,lastname,email,phone,address
    from customers
    where customerid= %s"""
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()

        cursor.execute(query, (customer_id,))
        record = cursor.fetchone()

        if record is None:
            raise CustomerNotFoundException(f"Customer with ID {customer_id} not found")

        customer = Customer(
            customer_id=record[0],
            first_name=record[1],
            last_name=record[2],
            email=record[3],
            phone=record[4],
            address=record[5]
```

```
        )
        if include_order_count and order_dao:
            order_count = order_dao.count_orders_by_customer(customer_id)
            customer.order_count = order_count
        return customer

    except Exception as e:
        raise Exception(f"Error retrieving customer: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()
```

8: Payment Processing

```
def process_payment(self, order_id, payment_method, amount):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor(dictionary=True)
        conn.autocommit = False

        order_query = """
        select o.TotalAmount, o.Status, c.FirstName AS first_name, c.LastName AS last_name,
        c.Email AS email, c.Phone AS phone, c.Address AS address
        from orders o
        join customers c ON o.CustomerID = c.CustomerID
        where o.OrderID = %s """
        cursor.execute(order_query, (order_id,))
        order_data = cursor.fetchone()

        if not order_data:
            raise OrderNotFoundException(f"Order with ID {order_id} not found")

        order_amount = Decimal(str(order_data['TotalAmount']))
        order_status = order_data['Status']

        if order_status != 'pending':
            raise PaymentFailedException(f"Cannot process payment for order in {order_status} status")

        balance = amount - order_amount
        if balance < 0:
            raise PaymentFailedException(
                f"Payment amount ${amount:.2f} is less than order total ${order_amount:.2f}")

        payment_query = """insert into payments (OrderID, Amount, PaymentMethod, Status)
        VALUES (%s, %s, %s, %s)"""
        cursor.execute(payment_query, (
            order_id,
            order_amount,
            payment_method,
            'Completed'
```

PaymentID	OrderID	Amount	PaymentMethod	Status
1	1	200.00	Credit Card	Completed
2	2	100.00	PayPal	Completed
3	3	50.00	Credit Card	Pending
4	4	25.00	Debit Card	Completed
5	5	39.00	Credit Card	Pending
6	6	22.00	Debit Card	Completed
8	9	1650.00	Credit Card	Completed

```
        ))

        if balance > 0:
            cursor.execute(payment_query, (
                order_id,
                -balance,
                'Balance',
                'Completed'
            ))

        update_query = "UPDATE orders SET Status = 'Paid' WHERE OrderID = %s"
        cursor.execute(update_query, (order_id,))

        conn.commit()

        customer = Customer(
            None,
            order_data['first_name'],
            order_data['last_name'],
            order_data['email'],
            order_data['phone'],
            order_data['address']
        )

        return {
            'order': Order(
                order_id,
                customer,
                None,
                order_amount,
                'Paid'
            ),
            'amount_paid': amount,
            'balance_given': balance if balance > 0 else 0
        }

    except Exception as e:
        if conn:
            conn.rollback()
        if isinstance(e, (OrderNotFoundException, PaymentFailedException)):
            raise
        raise Exception(f"Error processing payment: {str(e)}")
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()

def get_payment_details(self, order_id):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor(dictionary=True)
```

ID	Customer	Date	Status	Total
6	Mohammed Aizen	2025-04-30	pending	AED22.00
15	Uzumaki Nair	2025-04-05	Pending	AED1650.00
12	Sungjinwoo Singh	2025-04-05	Pending	AED825.00
11	Ichigo Kumar	2025-04-05	Paid	AED50.00
9	Sungjinwoo Singh	2025-04-05	Paid	AED1650.00
3	Isagi Kahn	2025-03-25	pending	AED200.00
5	Gojo Reddy	2025-03-19	pending	AED39.00
4	Uzumaki Nair	2025-03-15	Pending	AED25.00
1	Sungjinwoo Singh	2025-02-05	pending	AED200.00
2	Ichigo Kumar	2025-01-28	pending	AED200.00

```
        query = """ select p.*
        from payments p
        where p.orderid = %s
        order by p.orderid
        limit 1 """
        cursor.execute(query, (order_id,))
        payment_data = cursor.fetchone()

        if not payment_data:
            raise PaymentFailedException(f"No payment found for order {order_id}")

        return {
            'payment_id': payment_data['PaymentID'],
            'order_id': payment_data['OrderID'],
            'amount': float(payment_data['Amount']),
            'method': payment_data['PaymentMethod'],
            'status': payment_data['Status']
        }

    except Exception as e:
        if isinstance(e, PaymentFailedException):
            raise
        raise Exception(f"Error retrieving payment details: {str(e)}")
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()

def refund_payment(self, order_id, amount=None):
    conn = None
    cursor = None
    try:
        conn = DBConnUtil.get_connection(self.__connection_string)
        cursor = conn.cursor()
        conn.autocommit = False

        payment = self.get_payment_details(order_id)
        if payment['status'] != 'Completed':
            raise PaymentFailedException("Cannot refund - payment not completed")

        refund_amount = amount if amount is not None else payment['amount']
        if refund_amount > payment['amount']:
            raise PaymentFailedException(
                f"Refund amount ${refund_amount} exceeds original payment ${payment['amount']}")

        refund_query = """ insert into payments (orderid, amount, paymentmethod, status)
        values (%s, %s, %s, %s) """
        cursor.execute(refund_query, (
            order_id,
            -refund_amount,
            'Refund',
            'Completed'
```

```
        ))

        if refund_amount == payment['amount']:
            update_query = "update orders set status = 'Refunded' where orderid = %s"
            cursor.execute(update_query, (order_id,))

        conn.commit()
        return True

    except Exception as e:
        if conn:
            conn.rollback()
        if isinstance(e, PaymentFailedException):
            raise
        raise Exception(f"Error processing refund: {str(e)}")
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
```

9: Product Search and Recommendations

```
from entity.Products import Product
from util.db_conn_util import DBConnUtil
from util.db_property_util import DBPropertyUtil
from exception.dataException import InvalidDataException, ProductNotFoundException

class ProductDAO(ServiceProvider):
    def __init__(self):
        self.__connection_string = DBPropertyUtil.get_connection_string("db.properties")

    def get_by_id(self, product_id):
        query = """ select productid, productname, description, price, category
        from products
        where productid = %s """
        try:
            conn = DBConnUtil.get_connection(self.__connection_string)
            cursor = conn.cursor()

            cursor.execute(query, (product_id,))
            record = cursor.fetchone()

            if record is None:
                raise ProductNotFoundException(f"Product with ID {product_id} not found")

            return Product(
                product_id=record[0],
                product_name=record[1],
                description=record[2],
                price=record[3],
                category=record[4]
```

```
    )

except Exception as e:
    raise Exception(f"Error retrieving product: {str(e)}")
finally:
    if 'cursor' in locals():
        cursor.close()
    if 'conn' in locals():
        conn.close()
```

```
ID  Name    Description Category    Price
------------------------------------------------------------
1   Shadow SSD      A high-speed 1TB SSD with rapi  Storage Device          825.00
2   Hollow VR Heado A VR headset with immersive au  Wearable Techno         2530.00
3   Tactical Smart  Advanced sports analyzing watc  Wearable Techno         1700.00
4   Rasengan Drone  High-speed drone with rotor bl  Drones                  3200.00
5   Infinity Projec Projector with limitless focus   Projector             1980.00
6   Illusionary Sma Smart glasses with holographic  Wearable Techno         2200.00
7   Laptop          High Processing Gaming Laptop   Electronics            50.00
8   Tablet          Foldable Tablet                 Electronics            100.00
9   IPad            Super fast gaming Ipad          Electronics            350.00
```

ProductID	ProductName	Description	Price	Category
1	Shadow SSD	A high-speed 1TB SSD with rapid data transfer	825.00	Storage Device
2	Hollow VR Headset	A VR headset with immersive audio and visuals	2530.00	Wearable Technology
3	Tactical Smart Watch	Advanced sports analyzing watch	1700.00	Wearable Technology
4	Rasengan Drone	High-speed drone with rotor blades	3200.00	Drones
5	Infinity Projector	Projector with limitless focus and crystal-clear vi...	1980.00	Projector
6	Illusionary Smart Glasses	Smart glasses with holographic displays	2200.00	Wearable Technology
7	Laptop	High Processing Gaming Laptop	50.00	Electronics
8	Tablet	Foldable Tablet	100.00	Electronics
9	IPad	Super fast gaming Ipad	350.00	Electronics