

# Assignment 1: TechShop, an electronic gadgets shop

## Implement OOPs Task 1:

### Classes and Their Attributes:

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

### Customers Class:

#### Attributes:

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

#### 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).

```
from exception.ExceptionHandling import InvalidDataException, Validation

class Customer:
    def __init__(self, customer_id, first_name, last_name, email, phone, address):
        self.__CustomerID = customer_id
        self.__FirstName = first_name
        self.__LastName = last_name
        self.__Email = email
        self.__Phone = phone
        self.__Address = address
        self.__Orders = []

    @property
    def CustomerID(self):
        return self.__CustomerID

    @property
    def FirstName(self):
        return self.__FirstName

    @FirstName.setter
    def FirstName(self, first_name):
        if isinstance(first_name, str):
            self.__FirstName = first_name
        else:
            raise InvalidDataException("First Name must be a string")
```

```

def FirstName(self, first_name):
    if isinstance(first_name, str):
        self.__FirstName = first_name
    else:
        raise Exception("First name must be a string.")

@property
def LastName(self):
    return self.__LastName

def LastName(self, last_name):
    if isinstance(last_name, str):
        self.__LastName = last_name
    else:
        raise Exception("Last name must be a string.")

@property
def Email(self):
    return self.__Email

@Email.setter
def Email(self, email):
    try:
        Validation.validate_email(email)
    except ValidationException as e:
        raise ValueError(str(e))

```

```

def Phone(self, phone):
    if isinstance(phone, str):
        self.__Phone = phone
    else:
        raise Exception("Phone must be a string.")

@property
def Orders(self):
    return self.__Orders

@Address.setter
def Address(self, address):
    if isinstance(address, str):
        self.__Address = address
    else:
        raise ValueError("Address must be a string.")

def calculate_total_orders(self, customer_id):
    pass

```

## Products Class: Attributes:

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

## Methods:

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

```

1 class Product():
2     def __init__(self, product_id, name, description, price):
3         self.__ProductID = product_id
4         self.__ProductName = name
5         self.__Description = description
6         self.__Price = price
7
8     @property
9     def ProductID(self):
10        return self.__ProductID
11
12    @ProductName.setter
13    def ProductName(self, product_name):
14        if isinstance(product_name, str):
15            self.__ProductName = product_name
16        else:
17            raise Exception("Product name should be string only")
18
19    @Price.setter
20    def Description(self, product_description):
21        if isinstance(product_description, str):
22            self.__Description = product_description
23        else:
24            raise Exception("Description should be String")
25
26    @Price.setter
27    def Price(self, product_price):
28        if isinstance(product_price, int) and product_price > 0:
29            self.__Price = product_price
30        else:
31            raise Exception("Price must be numeric and non negative")
32
33    def get_product_by_id(self, product_id):
34        pass
35
36    def update_product_info(self, price):
37

```

## Orders Class:

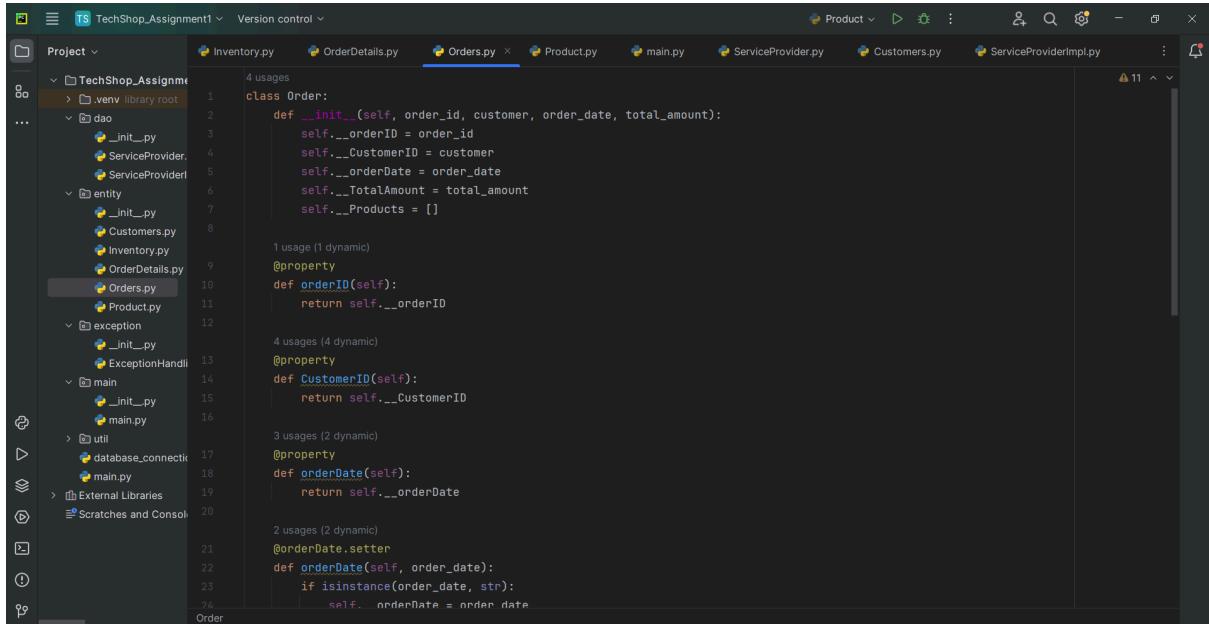
### Attributes:

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

### Methods:

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

status of the order (e.g., processing, shipped). • CancelOrder(): Cancels the order and adjusts stock levels for products.



```

class Order:
    def __init__(self, order_id, customer, order_date, total_amount):
        self.__orderID = order_id
        self.__CustomerID = customer
        self.__orderDate = order_date
        self.__TotalAmount = total_amount
        self.__Products = []

    @usage(1 dynamic)
    @property
    def orderID(self):
        return self.__orderID

    @usage(4 dynamic)
    @property
    def CustomerID(self):
        return self.__CustomerID

    @usage(2 dynamic)
    @property
    def orderDate(self):
        return self.__orderDate

    @orderDate.setter
    def orderDate(self, order_date):
        if isinstance(order_date, str):
            self.__orderDate = order_date
        else:
            raise Exception("Invalid input")

    @TotalAmount.getter
    def TotalAmount(self):
        return self.__TotalAmount

    @TotalAmount.setter
    def TotalAmount(self, total_amount):
        if isinstance(total_amount, (int, float)) and total_amount >= 0:
            self.__TotalAmount = total_amount
        else:
            raise Exception("Must be integer and non negative")

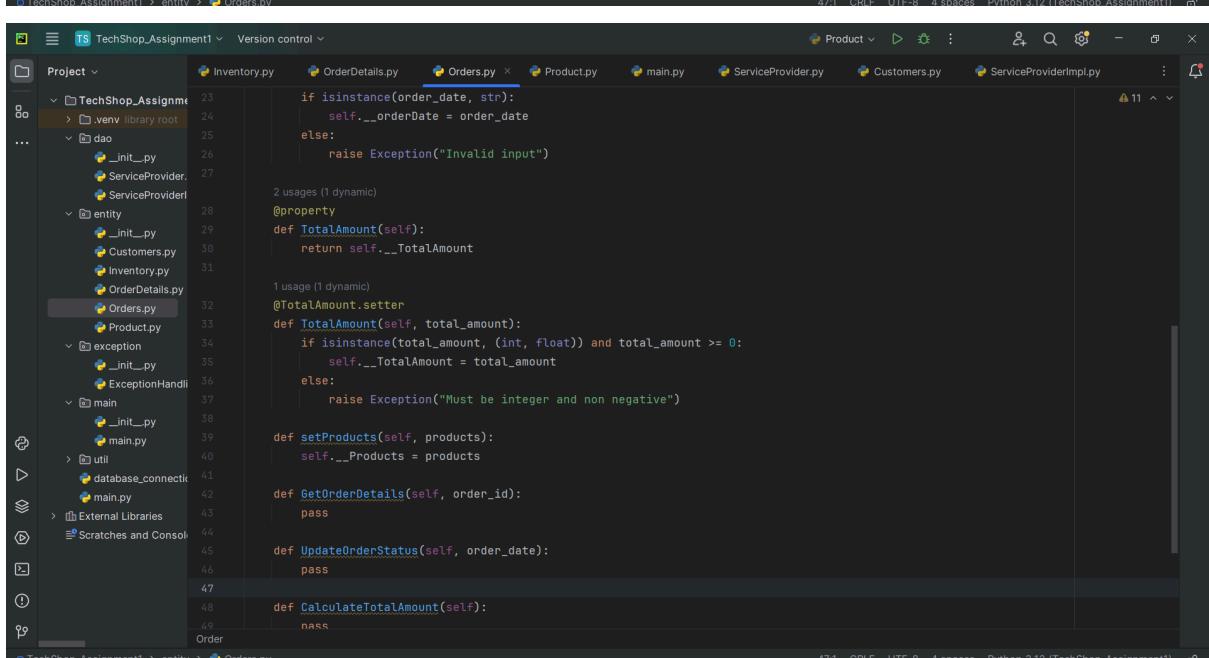
    def setProducts(self, products):
        self.__Products = products

    def GetOrderDetails(self, order_id):
        pass

    def UpdateOrderStatus(self, order_date):
        pass

    def CalculateTotalAmount(self):
        pass

```



## OrderDetails Class:

### Attributes:

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

### Methods:

- CalculateSubtotal() - Calculate the subtotal for this order detail.
- GetOrderDetailInfo(): Retrieves and displays information about this order

- **UpdateQuantity():** Allows updating the quantity of the product in this order detail.
- **AddDiscount():** Applies a discount to this order detail.

```

1 2 usages
2 class OrderDetail:
3     def __init__(self, order_details_id, order_id, product_id, quantity):
4         self.__OrderDetailID = order_details_id
5         self.__Order = order_id
6         self.__ProductID = product_id
7         self.__Quantity = quantity
8
9     @property
10    def OrderDetailID(self):
11        return self.__OrderDetailID
12
13    @property
14    def Order(self):
15        return self.__Order
16
17    4 usages (4 dynamic)
18    @property
19    def ProductID(self):
20        return self.__ProductID
21
22    3 usages (2 dynamic)
23    @property
24    def Quantity(self):
25        return self.__Quantity
26
27    2 usages (2 dynamic)
28    @Quantity.setter
29    def Quantity(self, quantity):
30        if quantity > 0:
31            self.__Quantity = quantity
32        else:
33            raise Exception("Quantity must be 0 or greater than 0")
34
35    def CalculateSubtotal(self):
36        pass
37
38    def GetOrderDetailInfo(self):
39        pass
40
41    def UpdateQuantity(self):
42        pass
43
44    def AddDiscount(self):
45        pass
46
47 OrderDetail : GetOrderDetailInfo()

```

```

18     return self.__ProductID
19
20     3 usages (2 dynamic)
21     @property
22     def Quantity(self):
23         return self.__Quantity
24
25     2 usages (2 dynamic)
26     @Quantity.setter
27     def Quantity(self, quantity):
28         if quantity > 0:
29             self.__Quantity = quantity
30         else:
31             raise Exception("Quantity must be 0 or greater than 0")
32
33     def CalculateSubtotal(self):
34         pass
35
36     def GetOrderDetailInfo(self):
37         pass
38
39     def UpdateQuantity(self):
40         pass
41
42     def AddDiscount(self):
43         pass
44
45 OrderDetail : GetOrderDetailInfo()

```

## Inventory class:

### Attributes:

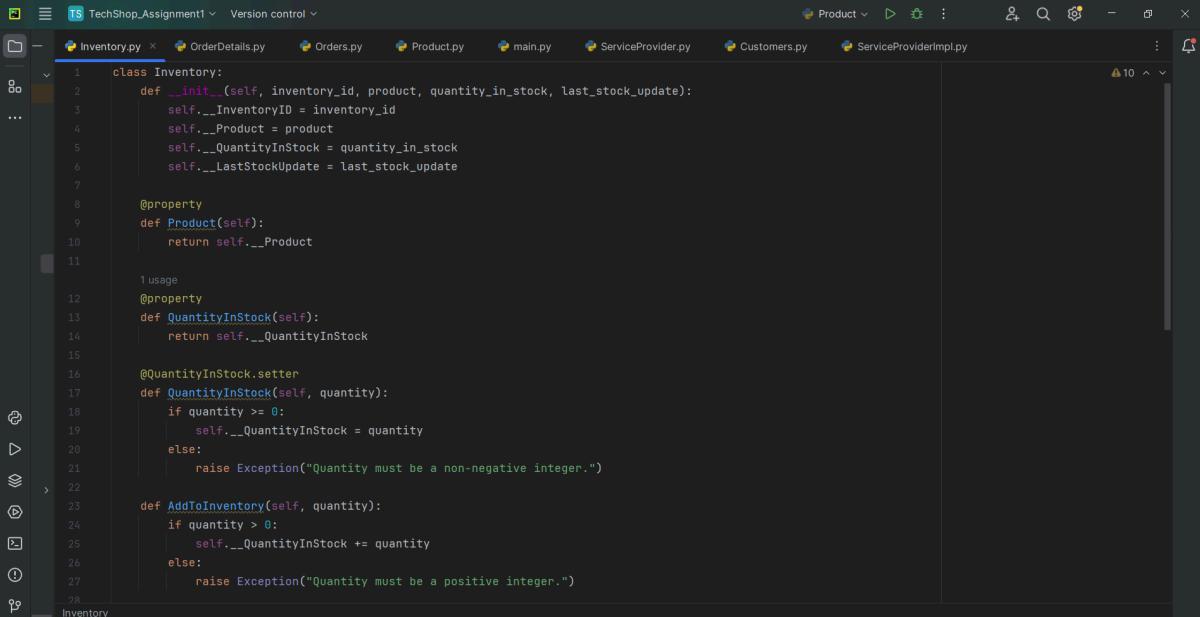
- **InventoryID(int)**
- **Product (Composition):** The product associated with the inventory item.
- **QuantityInStock:** The quantity of the product currently in stock.
- **LastStockUpdate**

### Methods:

- **GetProduct():** A method to retrieve the product associated with this inventory item.
- **GetQuantityInStock():** A method to get the current

quantity of the product in stock.

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



The screenshot shows a code editor window with the following details:

- Project Structure:** The left sidebar shows a project named "TechShop\_Assignment1" containing files: Inventory.py, OrderDetails.py, Orders.py, Product.py, main.py, ServiceProvider.py, Customers.py, and ServiceProviderImpl.py.
- Code Editor:** The main area displays the `Inventory.py` file with the following content:

```
1 class Inventory:
2     def __init__(self, inventory_id, product, quantity_in_stock, last_stock_update):
3         self.__InventoryID = inventory_id
4         self.__Product = product
5         self.__QuantityInStock = quantity_in_stock
6         self.__LastStockUpdate = last_stock_update
7
8     @property
9     def Product(self):
10        return self.__Product
11
12    @usage
13    @property
14    def QuantityInStock(self):
15        return self.__QuantityInStock
16
17    @QuantityInStock.setter
18    def QuantityInStock(self, quantity):
19        if quantity >= 0:
20            self.__QuantityInStock = quantity
21        else:
22            raise Exception("Quantity must be a non-negative integer.")
23
24    def AddToInventory(self, quantity):
25        if quantity > 0:
26            self.__QuantityInStock += quantity
27        else:
28            raise Exception("Quantity must be a positive integer.")
```

```

27     raise Exception("Quantity must be a positive integer.")
28
29     def RemoveFromInventory(self, quantity):
30         if 0 < quantity <= self.__QuantityInStock:
31             self.__QuantityInStock -= quantity
32         else:
33             raise Exception("Invalid quantity to remove from inventory.")
34
35     def UpdateStockQuantity(self, new_quantity):
36         if new_quantity >= 0:
37             self.__QuantityInStock = new_quantity
38         else:
39             raise Exception("New quantity must be a non-negative integer.")
40
41     def IsProductAvailable(self, quantity_to_check):
42         return quantity_to_check <= self.__QuantityInStock
43
44     def GetInventoryValue(self):
45         return self.__QuantityInStock * self.__Product.Price
46
47     def ListLowStockProducts(self, threshold):
48         if self.__QuantityInStock < threshold:
49             return self.__Product
50
51     def ListOutOfStockProducts(self):
52         if self.__QuantityInStock == 0:
53             return self.__Product
54

```

## 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.

```

1     from exception.ExceptionHandling import InvalidDataException, Validation
2
3
4 usages
5 class Customer:
6
7     def __init__(self, customer_id, first_name, last_name, email, phone, address):
8         self.__CustomerID = customer_id
9         self.__FirstName = first_name
10        self.__LastName = last_name
11        self.__Email = email
12        self.__Phone = phone
13        self.__Address = address
14        self.__Orders = []
15
16    4 usages (4 dynamic)
17    @property
18    def CustomerID(self):
19        return self.__CustomerID
20
21    @property
22    def FirstName(self):
23        return self.__FirstName
24
25    2 usages (2 dynamic)
26    @FirstName.setter
27    def FirstName(self, first_name):
28        if isinstance(first_name, str):
29            self.__FirstName = first_name
30
31

```

TechShop\_Assignment1 Version control

```
Project ▾ TechShop_Assignment1
  -> .venv library root
    -> dao
      -> __init__.py
      -> ServiceProvider.py
      -> ServiceProviderImpl.py
    -> entity
      -> __init__.py
      -> Customer.py
      -> Inventory.py
      -> OrderDetail.py
      -> Orders.py
      -> Product.py
      -> exception
        -> __init__.py
        -> Exception.py
        -> ExceptionHandler.py
      -> main
        -> __init__.py
        -> main.py
      -> util
        -> database_connect.py
        -> main.py
    -> External Libraries
    -> Scratches and Consol
  -> Product.py
  -> main.py
  -> ServiceProvider.py
  -> Customers.py
  -> ServiceProviderImpl.py
```

Product.py

```
1 class Product():
2     def __init__(self, product_id, name, description, price):
3         self.__ProductID = product_id
4         self.__ProductName = name
5         self.__Description = description
6         self.__Price = price
7
8     4 usages (4 dynamic)
9     @property
10    def ProductID(self):
11        return self.__ProductID
12
13    3 usages (2 dynamic)
14    @property
15    def ProductName(self):
16        return self.__ProductName
17
18    2 usages (2 dynamic)
19    @ProductName.setter
20    def ProductName(self, product_name):
21        if isinstance(product_name, str):
22            self.__ProductName = product_name
23        else:
24            raise Exception("Product name should be string only")
25
26    3 usages (2 dynamic)
27    @property
28    def Description(self):
29
```

TechShop\_Assignment1 Version control

```
Project ▾ TechShop_Assignment1
  -> .venv library root
    -> dao
      -> __init__.py
      -> ServiceProvider.py
      -> ServiceProviderImpl.py
    -> entity
      -> __init__.py
      -> Customer.py
      -> Inventory.py
      -> OrderDetail.py
      -> Orders.py
      -> Product.py
      -> exception
        -> __init__.py
        -> Exception.py
        -> ExceptionHandler.py
      -> main
        -> __init__.py
        -> main.py
      -> util
        -> database_connect.py
        -> main.py
    -> External Libraries
    -> Scratches and Consol
  -> Order.py
  -> main.py
  -> ServiceProvider.py
  -> Customers.py
  -> ServiceProviderImpl.py
```

Orders.py

```
1 class Order:
2     def __init__(self, order_id, customer, order_date, total_amount):
3         self.__orderID = order_id
4         self.__CustomerID = customer
5         self.__orderDate = order_date
6         self.__TotalAmount = total_amount
7         self.__Products = []
8
9     1 usage (1 dynamic)
10    @property
11    def orderID(self):
12        return self.__orderID
13
14    4 usages (4 dynamic)
15    @property
16    def CustomerID(self):
17        return self.__CustomerID
18
19    3 usages (2 dynamic)
20    @property
21    def orderDate(self):
22        return self.__orderDate
23
24    2 usages (2 dynamic)
25    @orderDate.setter
26    def orderDate(self, order_date):
27        if isinstance(order_date, str):
28            self.__orderDate = order_date
29
```

```

OrderDetails.py content:
1  class OrderDetail:
2      def __init__(self, order_details_id, order_id, product_id, quantity):
3          self.__OrderDetailID = order_details_id
4          self.__Order = order_id
5          self.__ProductID = product_id
6          self.__Quantity = quantity
7
8      @property
9          def OrderDetailID(self):
10             return self.__OrderDetailID
11
12     @property
13         def Order(self):
14             return self.__Order
15
16     @property
17         def ProductID(self):
18             return self.__ProductID
19
20     @property
21         def Quantity(self):
22             return self.__Quantity
23
24     @Quantity.setter
25         def Quantity(self, quantity):
26             GetOrderDetailInfo()

```

```

Inventory.py content:
1  class Inventory:
2      def __init__(self, inventory_id, product, quantity_in_stock, last_stock_update):
3          self.__InventoryID = inventory_id
4          self.__Product = product
5          self.__QuantityInStock = quantity_in_stock
6          self.__LastStockUpdate = last_stock_update
7
8      @property
9          def Product(self):
10             return self.__Product
11
12     @property
13         def QuantityInStock(self):
14             return self.__QuantityInStock
15
16     @QuantityInStock.setter
17         def QuantityInStock(self, quantity):
18             if quantity >= 0:
19                 self.__QuantityInStock = quantity
20             else:
21                 raise Exception("Quantity must be a non-negative integer.")
22
23     def AddToInventory(self, quantity):
24         if quantity > 0:
25             self.__QuantityInStock += quantity
26         else:
27             raise Exception("Quantity must be a positive integer.")

```

### 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).

### Task 4:

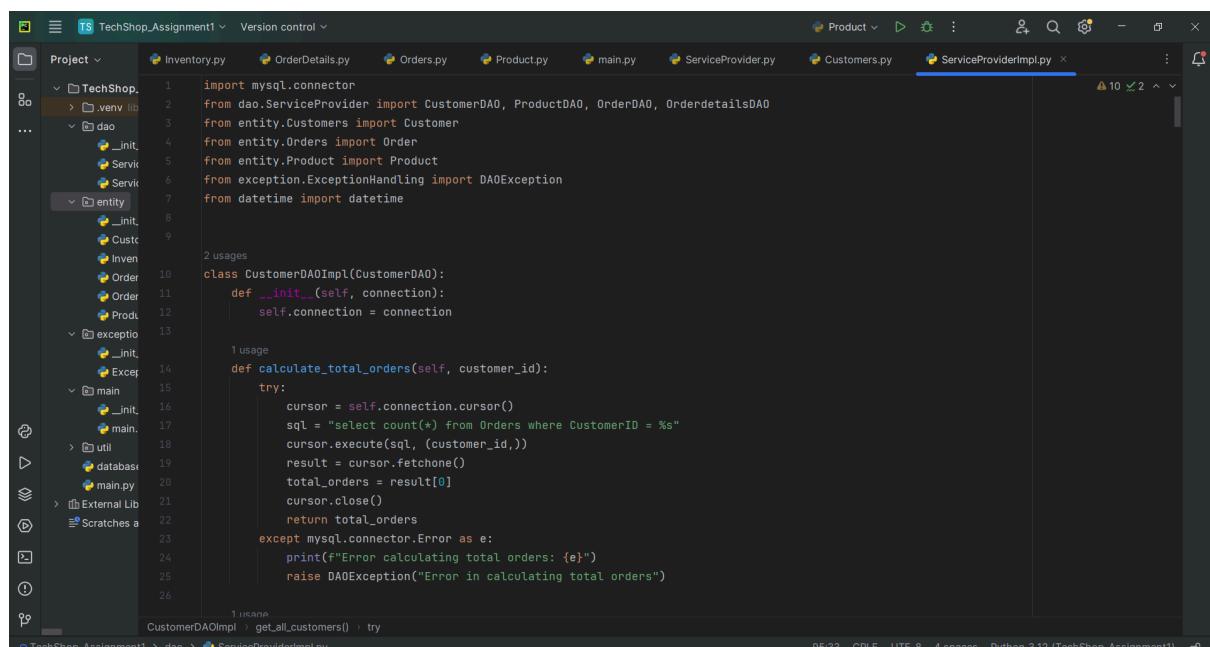
#### Composition:

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

- Orders Class with Composition:
  - o In the Orders class, we want to establish a composition relationship with the Customers class, indicating

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

- OrderDetails Class with Composition: o Similarly, in the OrderDetails class, we want to establish composition relationships with both the Orders and Products classes to represent the details of each order, including the product being ordered. o In the OrderDetails class, we've added two private attributes, order and product, of types Orders and Products, respectively, establishing composition relationships. The Order property provides access to the Orders object associated with the order detail, and the Product property provides access to the Products object representing the product in the order detail.
- Customers and Products Classes: o The Customers and Products classes themselves may not have direct composition relationships with other classes in this scenario. However, they serve as the basis for composition relationships in the Orders and OrderDetails classes, respectively.
- Inventory Class: o The Inventory class represents the inventory of products available for sale. It can have composition relationships with the Products class to indicate which products are in the inventory.



The screenshot shows a code editor interface with the following details:

- Project Structure:** The project is named "TechShop\_Assignment1". It contains several packages and modules:
  - dao**: Contains `CustomerDAO.py`, `ProductDAO.py`, `OrderDAO.py`, and `OrderdetailsDAO.py`.
  - entity**: Contains `Customer.py`, `Order.py`, and `Product.py`.
  - exception**: Contains `DAOException.py`.
  - main**: Contains `main.py`.
  - util**: Contains `util.py`.
  - database**: Contains `database.py`.
  - Scratches**: Contains `a.py` and `b.py`.
- Code Editor:** The current file is `ServiceProviderImpl.py`. The code implements a `CustomerDAOImpl` class that calculates the total number of orders for a given customer. It uses MySQL connector to execute a SQL query and handle exceptions.

```
import mysql.connector
from dao.ServiceProvider import CustomerDAO, ProductDAO, OrderDAO, OrderdetailsDAO
from entity.Customer import Customer
from entity.Orders import Order
from entity.Product import Product
from exception.ExceptionHandling import DAOException
from datetime import datetime

2 usages
class CustomerDAOImpl(CustomerDAO):
    def __init__(self, connection):
        self.connection = connection

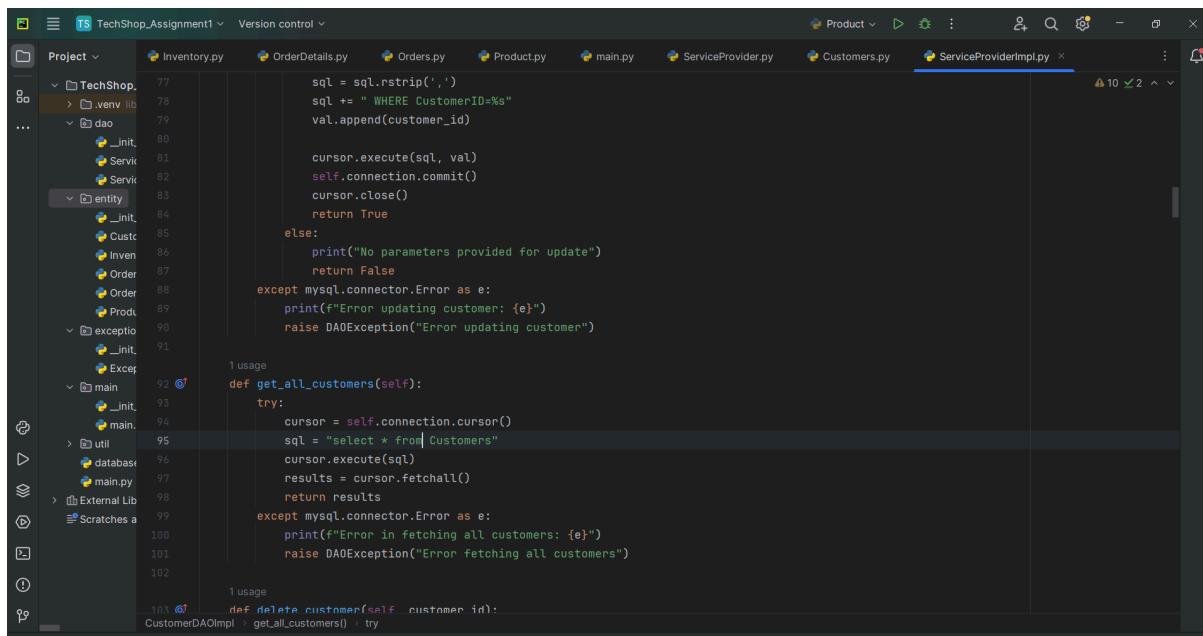
    1 usage
    def calculate_total_orders(self, customer_id):
        try:
            cursor = self.connection.cursor()
            sql = "select count(*) from Orders where CustomerID = %s"
            cursor.execute(sql, (customer_id,))
            result = cursor.fetchone()
            total_orders = result[0]
            cursor.close()
            return total_orders
        except mysql.connector.Error as e:
            print(f"Error calculating total orders: {e}")
            raise DAOException("Error in calculating total orders")
```

```
1 usage
def add_customer(self, customer):
    try:
        cursor = self.connection.cursor()
        sql = ("insert into Customers ( CustomerID, FirstName, LastName, Email, Phone, Address) "
               "VALUES (%s, %s, %s, %s, %s, %s)")
        val = (customer.CustomerID, customer.FirstName, customer.LastName, customer.Email, customer.Phone,
               customer.Address)
        cursor.execute(sql, val)
        self.connection.commit()
        return True
    except mysql.connector.Error as e:
        print(f"Error in creating customer: {e}")
        raise DAOException("Error creating customer")

4 usages
def get_customer_by_id(self, customer_id):
    try:
        cursor = self.connection.cursor()
        sql = "select * from Customers where CustomerID = %s"
        cursor.execute(sql, (customer_id,))
        result = cursor.fetchone()
        cursor.close()
        if result:
            customer1 = Customer(result[0], result[1], result[2], result[3], result[4], result[5])
            return customer1
        else:
            return None
    except DAOException as e:
        print(f"Error in fetching customer: {e}")
        raise DAOException("Error fetching customer")
```

```
else:
    return None
except mysql.connector.Error as e:
    print(f"Error in fetching customer: {e}")
    raise DAOException("Error fetching customer")

1 usage
def update_customer_info(self, customer_id, email=None, phone=None, address=None):
    try:
        cursor = self.connection.cursor()
        sql = "UPDATE Customers SET"
        val = []
        if email is not None:
            sql += " Email=%s,"
            val.append(email)
        if phone is not None:
            sql += " Phone=%s,"
            val.append(phone)
        if address is not None:
            sql += " Address=%s,"
            val.append(address)
        # Remove the trailing comma from the SQL query
        if len(val) > 0:
            sql = sql.rstrip(',')
            sql += " WHERE CustomerID=%s"
            val.append(customer_id)
        cursor.execute(sql, val)
        self.connection.commit()
    except DAOException as e:
        print(f"Error in updating customer: {e}")
        raise DAOException("Error updating customer")
```



```
sql = sql.rstrip('.')
sql += " WHERE CustomerID=%s"
val.append(customer_id)

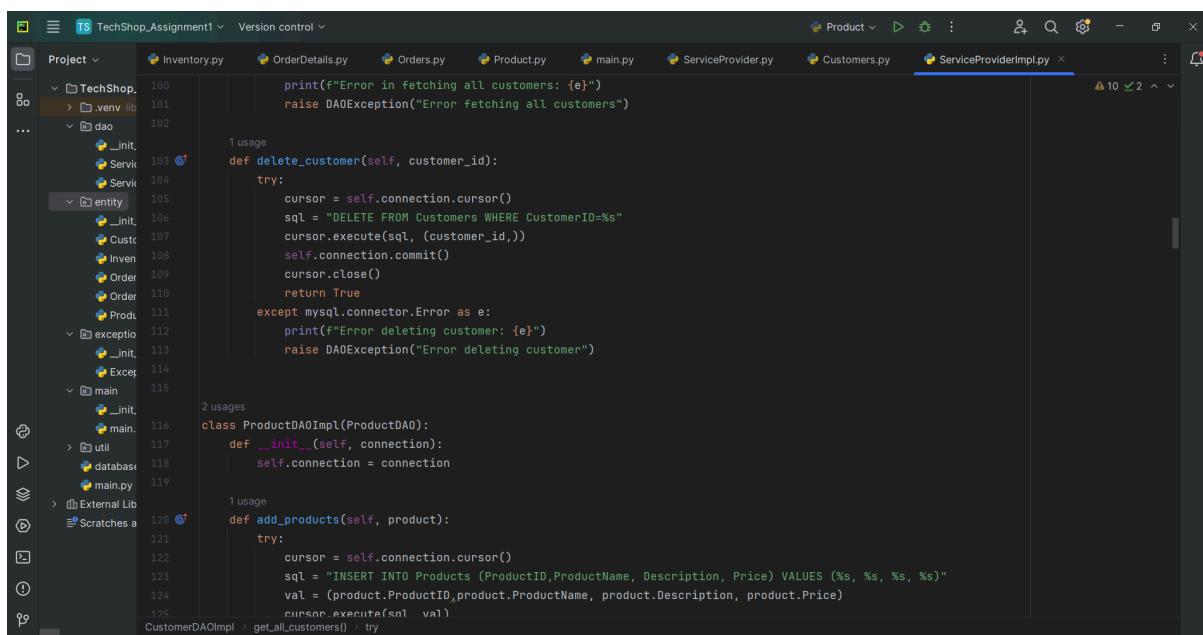
cursor.execute(sql, val)
self.connection.commit()
cursor.close()

return True
else:
    print("No parameters provided for update")
    return False
except mysql.connector.Error as e:
    print(f"Error updating customer: {e}")
    raise DAOException("Error updating customer")
```

1 usage

```
def get_all_customers(self):
    try:
        cursor = self.connection.cursor()
        sql = "select * from Customers"
        cursor.execute(sql)
        results = cursor.fetchall()
        return results
    except mysql.connector.Error as e:
        print(f"Error in fetching all customers: {e}")
        raise DAOException("Error fetching all customers")
```

CustomerDAOImpl > get\_all\_customers() > try



```
print(f"Error in fetching all customers: {e}")
raise DAOException("Error fetching all customers")
```

1 usage

```
def delete_customer(self, customer_id):
    try:
        cursor = self.connection.cursor()
        sql = "DELETE FROM Customers WHERE CustomerID=%s"
        cursor.execute(sql, (customer_id,))
        self.connection.commit()
        cursor.close()
        return True
    except mysql.connector.Error as e:
        print(f"Error deleting customer: {e}")
        raise DAOException("Error deleting customer")
```

2 usages

```
class ProductDAOImpl(ProductDAO):
    def __init__(self, connection):
        self.connection = connection
```

1 usage

```
def add_products(self, product):
    try:
        cursor = self.connection.cursor()
        sql = "INSERT INTO Products (ProductID,ProductName, Description, Price) VALUES (%s, %s, %s, %s)"
        val = (product.ProductID, product.ProductName, product.Description, product.Price)
        cursor.execute(sql, val)
    except mysql.connector.Error as e:
        print(f"Error adding product: {e}")
        raise DAOException("Error adding product")
```

CustomerDAOImpl > get\_all\_customers() > try

Project: TechShop\_Assignment1

File: ServiceProviderImpl.py

```
    val = (product.ProductID, product.ProductName, product.Description, product.Price)
    cursor.execute(sql, val)
    self.connection.commit()
    return True
except mysql.connector.Error as e:
    print(f"Error creating customer: {e}")
    raise DAOException("Error creating customer")
```

4 usages

```
def get_product_by_id(self, product_id):
    try:
        cursor = self.connection.cursor()
        sql = "SELECT * FROM Products WHERE ProductID = %s"
        cursor.execute(sql, (product_id,))
        result = cursor.fetchone()
        cursor.close()
        if result:
            product1 = Product(result[0], result[1], result[2], result[3])
            return product1
        else:
            return None
    except mysql.connector.Error as e:
        print(f"Error fetching customer: {e}")
        raise DAOException("Error fetching customer")
```

1 usage

```
def update_product_info(self, product_id, new_price=None):
    try:
        cursor = self.connection.cursor()
        CustomerDAOImpl.get_all_customers() > try
```

Project: TechShop\_Assignment1

File: ServiceProviderImpl.py

```
try:
    cursor = self.connection.cursor()
    sql = "UPDATE Products SET"
    val = []
    if new_price is not None:
        sql += " Price=%s"
        val.append(new_price)
    if len(val) > 0:
        sql = sql.rstrip(',')
        sql += " WHERE ProductID=%s"
        val.append(product_id)
    cursor.execute(sql, val)
    self.connection.commit()
    cursor.close()
    return True
else:
    print("No parameters provided for update")
    return False
except mysql.connector.Error as e:
    print(f"Error updating customer: {e}")
    raise DAOException("Error updating customer")
```

1 usage

```
def is_product_in_stock(self, product_id):
    try:
        cursor = self.connection.cursor()
        CustomerDAOImpl.get_all_customers() > try
```

```
1 usage
def is_product_in_stock(self, product_id):
    try:
        cursor = self.connection.cursor()
        sql = "SELECT QuantityInStock FROM Inventory WHERE ProductID = %s"
        cursor.execute(sql, (product_id,))
        total_orders = cursor.fetchone()
        cursor.close()
        if total_orders is not None:
            return True
        else:
            return False
    except mysql.connector.Error as e:
        print(f"Error calculating total orders: {e}")
        raise DAOException("Error calculating total orders")

1 usage
def get_all_products(self):
    try:
        cursor = self.connection.cursor()
        sql = "SELECT * FROM Products"
        cursor.execute(sql)
        results = cursor.fetchall()
        return results
    except mysql.connector.Error as e:
        print(f"Error fetching all customers: {e}")
        raise DAOException("Error fetching all customers")
```

```
raise DAOException("Error fetching all customers")

1 usage
def delete_products(self, product_id):
    try:
        cursor = self.connection.cursor()
        sql = "DELETE FROM Products WHERE ProductID = %s"
        cursor.execute(sql, (product_id,))
        self.connection.commit()
        cursor.close()
        return True
    except mysql.connector.Error as e:
        print(f"Error deleting product: {e}")
        raise DAOException("Error deleting product")

2 usages
class OrderDAOImpl(OrderDAO):
    def __init__(self, connection):
        self.connection = connection

    1 usage
    def create_orders(self, order, order_details):
        try:
            cursor = self.connection.cursor()
            total_amount = 0
            for order_detail in order_details:
                sql_get_price = "SELECT Price FROM Products WHERE ProductID = %s"
                cursor.execute(sql_get_price, (order_detail.ProductID))
                cursor.execute(sql_get_price, (order_detail.ProductID))
                cursor.execute(sql_get_price, (order_detail.ProductID))
```

TechShop\_Assignment1 Version control

```
total_amount += order_detail.Quantity * price
sql_insert_order = "INSERT INTO Orders (CustomerID, OrderDate, TotalAmount) VALUES (%s, %s, %s)"
order_data = (order.CustomerID, order.orderDate, total_amount)
cursor.execute(sql_insert_order, order_data)
order_id = cursor.lastrowid
for order_detail in order_details:
    sql_insert_order_detail = "INSERT INTO OrderDetails (OrderID, ProductID, Quantity) VALUES (%s, %s, %s)"
    order_detail_data = (order_id, order_detail.ProductID, order_detail.Quantity)
    cursor.execute(sql_insert_order_detail, order_detail_data)
self.connection.commit()
return True
except mysql.connector.Error as e:
    print(f"Error creating customer: {e}")
    raise DAOException("Error adding order")
```

TechShop\_Assignment1 Version control

```
def display_orders(self):
    try:
        cursor = self.connection.cursor()
        sql = "SELECT * FROM Orders"
        cursor.execute(sql)
        results = cursor.fetchall()
        return results
    except mysql.connector.Error as e:
        print(f"Error fetching all customers: {e}")
        raise DAOException("Error fetching all customers")
```

```
def CancelOrder(self, order_id):
    CustomerDAOImpl.get_all_customers() try
```

```
def CancelOrder(self, order_id):
    try:
        cursor = self.connection.cursor()
        sql_select_order = "SELECT OrderID FROM Orders WHERE OrderID = %s"
        cursor.execute(sql_select_order, (order_id,))
        result = cursor.fetchone()
        if not result:
            raise DAOException("Order not found")
        sql_select_order_details = "SELECT ProductID, Quantity FROM OrderDetails WHERE OrderID = %s"
        cursor.execute(sql_select_order_details, (order_id,))
        order_details = cursor.fetchall()
        for product_id, quantity in order_details:
            sql_update_inventory = ("UPDATE Inventory SET QuantityInStock ="
                                   " QuantityInStock + %s WHERE ProductID = %s")
            cursor.execute(sql_update_inventory, (quantity, product_id))
        sql_delete_order_details = "DELETE FROM OrderDetails WHERE OrderID = %s"
        cursor.execute(sql_delete_order_details, (order_id,))
        sql_delete_order = "DELETE FROM Orders WHERE OrderID = %s"
        cursor.execute(sql_delete_order, (order_id,))
        self.connection.commit()
        cursor.close()
    return True
    except mysql.connector.Error as e:
        print(f"Error deleting product: {e}")
        raise DAOException("Error deleting product")
```

```
CustomerDAOImpl.get_all_customers()
```

TechShop\_Assignment1 Version control

```
Project Inventory.py OrderDetails.py Orders.py Product.py main.py ServiceProvider.py Customers.py ServiceProviderImpl.py
```

2 usages

```
def GetOrderDetails(self, order_id):
    try:
        cursor = self.connection.cursor()
        sql = "SELECT * FROM Orders WHERE OrderID = %s"
        cursor.execute(sql, (order_id,))
        results = cursor.fetchall()
        cursor.close()
        orders = []
        if results:
            for result in results:
                order = Order(result[0], result[1], result[2], result[3])
                orders.append(order)
        return orders
    else:
        return None
    except mysql.connector.Error as e:
        print(f"Error fetching customer: {e}")
        raise DAOException("Error fetching customer")
```

1 usage

```
def CalculateTotalAmount(self):
    try:
        cursor = self.connection.cursor()
        sql = "SELECT SUM(TotalAmount) from Orders"
        cursor.execute(sql)
        total_price_info = cursor.fetchone()
        cursor.close()
        if total_price_info:
            if total_price_info[0]:
                return total_price_info[0]
            else:
                return 0
    except mysql.connector.Error as e:
        print(f"Error calculating total amount for order: {e}")
        raise DAOException("Error calculating total amount for order")
```

CustomerDAOImpl.get\_all\_customers() > try

05:33 CR/LF UTF-8 4 spaces Python 3.12 (TechShop Assignment1)

TechShop\_Assignment1 Version control

```
Project Inventory.py OrderDetails.py Orders.py Product.py main.py ServiceProvider.py Customers.py ServiceProviderImpl.py
```

299

```
sql = "SELECT SUM(TotalAmount) from Orders"
cursor.execute(sql)
total_price_info = cursor.fetchone()
cursor.close()
if total_price_info:
    return total_price_info[0]
else:
    return 0
except mysql.connector.Error as e:
    print(f"Error calculating total amount for order: {e}")
    raise DAOException("Error calculating total amount for order")
```

1 usage

```
def UpdateOrderStatus(self, order_id):
    try:
        cursor = self.connection.cursor()
        sql_get_order_date = "SELECT OrderDate FROM Orders WHERE OrderID = %s"
        cursor.execute(sql_get_order_date, (order_id,))
        result = cursor.fetchone()
        if result:
            order_date = result[0]
            current_date = datetime.now()
            order_date = datetime.combine(order_date, datetime.min.time())
            difference = current_date - order_date
            if difference.days > 3:
                return "shipped"
            else:
                return "Processing"
        else:
            CustomerDAOImpl.get_all_customers() > try
    except mysql.connector.Error as e:
        print(f"Error updating order status: {e}")
        raise DAOException("Error updating order status")
```

CustomerDAOImpl.get\_all\_customers() > try

05:33 CR/LF LITE 8 4 spaces Python 3.12 (TechShop Assignment1)

```

TechShop_Assignment1> dao>>> ServiceProviderImpl.py
325             return "Processing"
326         else:
327             return "Order not found"
328     except mysql.connector.Error as e:
329         print(f"Error updating order status: {e}")
330         raise DAOException("Error updating order status")
331
332     2 usages
333     class OrderdetailsDAOImpl(OrderdetailsDAO):
334         def __init__(self, connection):
335             self.connection = connection
336
337     1 usage
338     def GetAllOrderDetail(self):
339         try:
340             cursor = self.connection.cursor()
341             sql = "SELECT * FROM orderdetails"
342             cursor.execute(sql)
343             results = cursor.fetchall()
344             return results
345         except mysql.connector.Error as e:
346             print(f"Error fetching all customers: {e}")
347             raise DAOException("Error fetching all customers")
348
349     2 usages
350     def CalculateSubtotal(self, order_detail_id):
351         try:
352             cursor = self.connection.cursor()
353             cursor.execute("SELECT od.ProductID WHERE od.OrderDetailID = %s", (order_detail_id,))
354             result = cursor.fetchone()
355             if result:
356                 price, quantity = result
357                 subtotal = price * quantity
358                 return subtotal
359             else:
360                 return None
361         except mysql.connector.Error as e:
362             print(f"Error calculating subtotal: {e}")
363             raise DAOException("Error calculating subtotal")
364
365     1 usage
366     def GetOrderDetailInfo(self, order_detail_id):
367         try:
368             cursor = self.connection.cursor()
369             sql = ("SELECT od.OrderDetailID, o.OrderID, p.ProductName, od.Quantity FROM OrderDetails od INNER JOIN "
370                   "Orders o ON od.OrderID = o.OrderID INNER JOIN Products p ON od.ProductID = p.ProductID "
371                   "WHERE od.OrderDetailID = %s")
372             cursor.execute(sql, (order_detail_id,))
373             result = cursor.fetchone()
374             if result:
375                 order_detail_id, order_id, product_name, quantity = result
376                 print("Order Detail ID:", order_detail_id)
377                 print("Order ID:", order_id)
378                 print("Product Name:", product_name)
379                 print("Quantity:", quantity)
380             else:
381                 CustomerDAOImpl.get_all_customers() > try

```

## Task 5: Exceptions handling

- Data Validation:
- Inventory Management:
- Order Processing:
- Payment Processing:
- Database Access:
- Concurrency Control:
- Security and Authentication:

```

4 usages
class InvalidDataException(Exception):
    pass

class InsufficientStockException(Exception):
    pass

class IncompleteOrderException(Exception):
    pass

class DuplicateEmailException(Exception):
    pass

25 usages
class DAOException(Exception):
    pass

2 usages
class Validation:
    1 usage
    @staticmethod
    def validate_email(email):
        if '@' not in email or '.' not in email:
            raise InvalidDataException("Invalid email format")

```

## Task 7: Database Connectivity

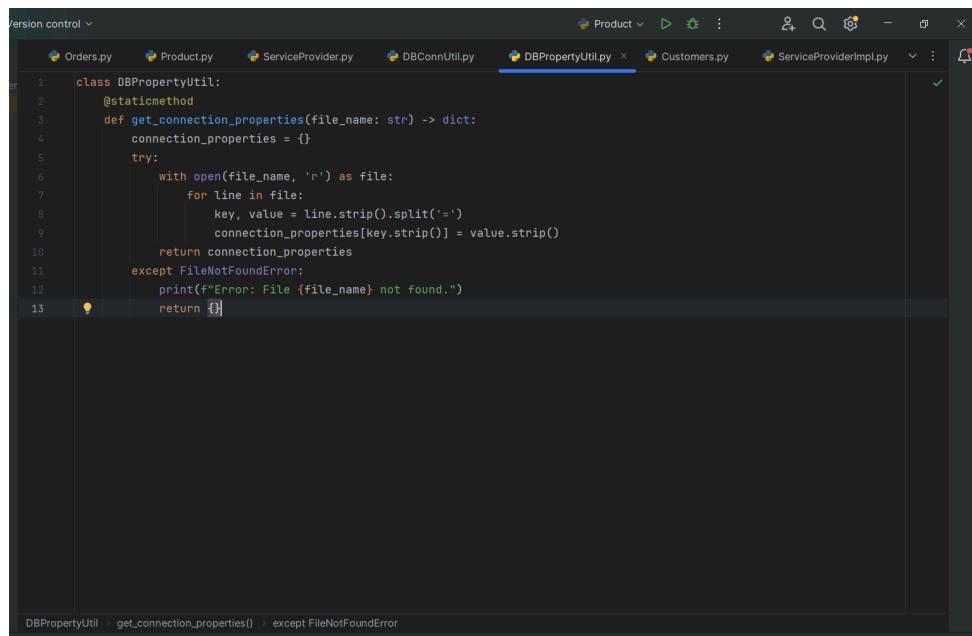
- Implement a DatabaseConnector class responsible for establishing a connection to the "TechShopDB" database. This class should include methods for opening, closing, and managing database connections.
- Implement classes for Customers, Products, Orders, OrderDetails, Inventory with properties, constructors,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

```

import mysql.connector

class DBConnUtil:
    @staticmethod
    def get_connection(connection_properties: dict):
        try:
            connection = mysql.connector.connect(
                host=connection_properties['host'],
                port=connection_properties['port'],
                user=connection_properties['user'],
                password=connection_properties['password'],
                database=connection_properties['database']
            )
            return connection
        except mysql.connector.Error as e:
            print(f"Error connecting to database: {e}")

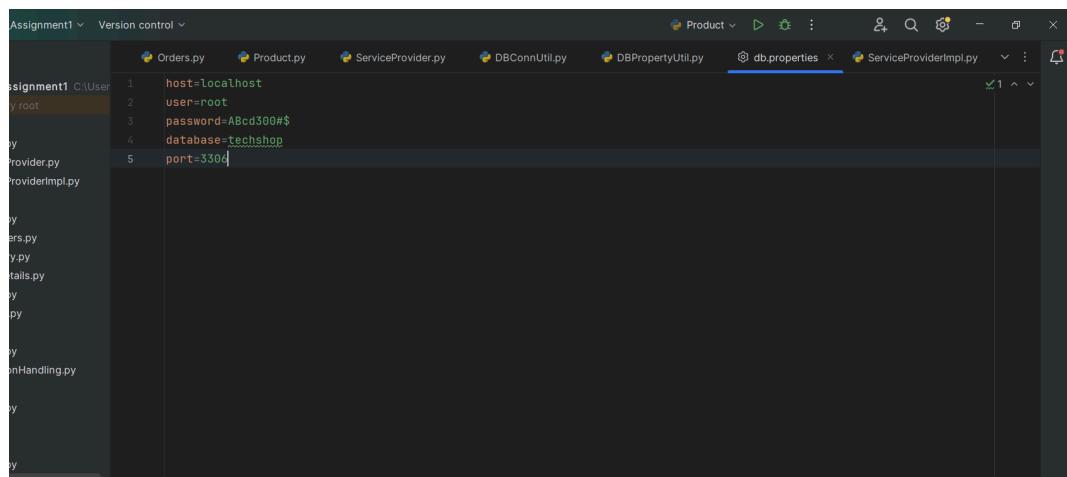
```



A screenshot of a code editor showing the file `DBPropertyUtil.py`. The code defines a class `DBPropertyUtil` with a static method `get_connection_properties`. This method reads a file named `file_name` and parses it into a dictionary where keys and values are separated by an equals sign (`=`). If the file is not found, it prints an error message and returns an empty dictionary.

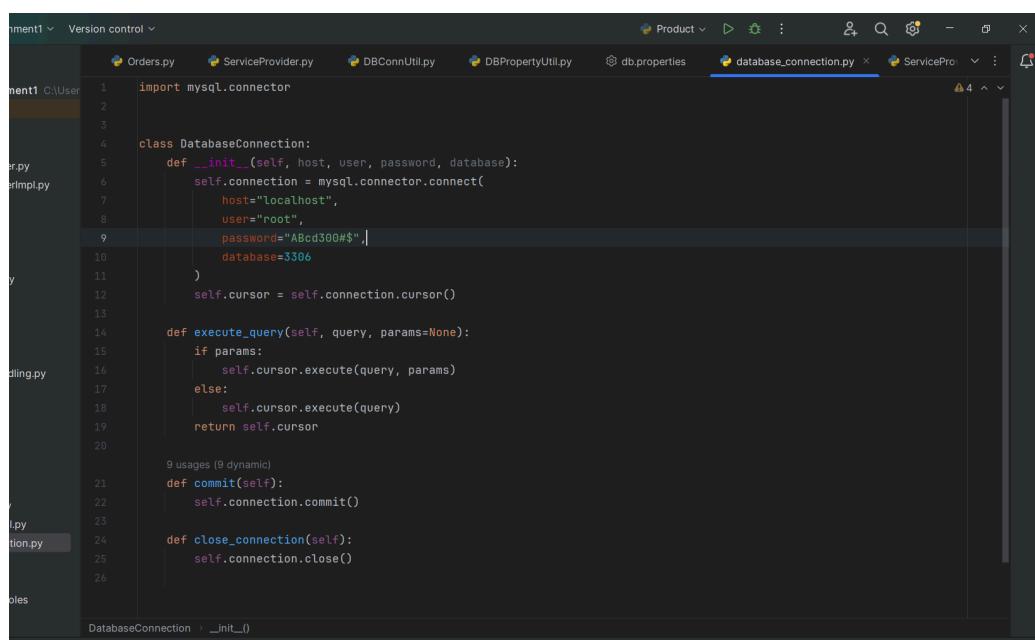
```
1  class DBPropertyUtil:
2      @staticmethod
3      def get_connection_properties(file_name: str) -> dict:
4          connection_properties = {}
5          try:
6              with open(file_name, 'r') as file:
7                  for line in file:
8                      key, value = line.strip().split('=')
9                      connection_properties[key.strip()] = value.strip()
10             return connection_properties
11         except FileNotFoundError:
12             print(f"Error: File {file_name} not found.")
13         return {}
```

DBPropertyUtil : get\_connection\_properties() > except FileNotFoundError



A screenshot of a code editor showing the file `db.properties`. It contains a single line of configuration: `port=3306`.

```
host=localhost
user=root
password=ABcd300#$
database=techshop
port=3306
```



A screenshot of a code editor showing the file `database_connection.py`. It defines a class `DatabaseConnection` with methods `__init__`, `execute_query`, `commit`, and `close_connection`. The `__init__` method initializes a MySQL connection using the properties defined in `db.properties`.

```
import mysql.connector

class DatabaseConnection:
    def __init__(self, host, user, password, database):
        self.connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="ABcd300#",
            database="techshop"
        )
        self.cursor = self.connection.cursor()

    def execute_query(self, query, params=None):
        if params:
            self.cursor.execute(query, params)
        else:
            self.cursor.execute(query)
        return self.cursor

    def commit(self):
        self.connection.commit()

    def close_connection(self):
        self.connection.close()
```

- 1: Customer Registration
- 2: Product Catalog Management'
- 3: Placing Customer Orders
- 4: Tracking Order Status
- 6: Sales Reporting
- 7: Customer Account Updates
- 8: Payment Processing
- 9: Product Search and Recommendations

```

import mysql.connector
from entity.Customers import Customer
from dao.ServiceProviderImpl import CustomerDAOImpl, ProductDAOImpl, OrderDAOImpl, OrderdetailsDAOImpl
from entity.OrderDetails import OrderDetail
from entity.Orders import Order
from entity.Product import Product
from datetime import datetime

class MainClass:
    def __init__(self):
        self.connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='ABcd300#$', 
            database='techshop'
        )
        self.customers_dao = CustomerDAOImpl(self.connection)
        self.products_dao = ProductDAOImpl(self.connection)
        self.orders_dao = OrderDAOImpl(self.connection)
        self.order_detail_dao = OrderdetailsDAOImpl(self.connection)

    @staticmethod
    def display_menu():
        print("\nWelcome to TechShop, an Electronic Gadget Shop")
        print("1. Manage Customers")
        print("2. Manage Products")

```

```

        print("3. Manage Orders")
        print("4. Manage Order Details")
        print("5. Manage Inventory")
        print("6. Exit")

    def run(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice: ")

            if choice == '1':
                self.manage_customers()
            elif choice == '2':
                self.manage_product()
            elif choice == '3':
                self.manage_orders()
            elif choice == '4':
                self.manage_order_details()
            elif choice == '5':
                self.manage_inventory()
            elif choice == '6':
                print("Program Ends, Bye!")
                break
            else:
                print("Invalid choice. Please enter again.")

```

The image shows two vertically stacked code snippets in a code editor, likely from a Java IDE. Both snippets are part of a file named `main.py`.

**Top Snippet (Lines 55-80):**

```
1 usage
56     def manage_customers(self):
57         print("\n*****CUSTOMER MENU*****")
58         while True:
59             print("\nMenu:")
60             print("1. Add Customer(C)")
61             print("2. Get all the Customer Details(R)")
62             print("3. Update the Customer Information(U)")
63             print("4. Delete the Customer(D)")
64             print("5. View Specific Customer Details")
65             print("6. Calculate Total Orders")
66             print("7. Exit")
67
68             choice = input("Enter your choice (1-7): ")
69
70             if choice == "1":
71                 customer_id = input("Enter ID: ")
72                 first_name = input("Enter First Name: ")
73                 last_name = input("Enter Last Name: ")
74                 email = input("Enter Email: ")
75                 phone = input("Enter Phone: ")
76                 address = input("Enter Address: ")
77                 customer = Customer(customer_id, first_name, last_name, email, phone, address)
78                 self.customers_dao.add_customer(customer)
79                 print("Customer added successfully.")
80
81             elif choice == "2":
```

**Bottom Snippet (Lines 81-109):**

```
81
82             elif choice == "2":
83                 customers = self.customers_dao.get_all_customers()
84                 if customers:
85                     for customer in customers:
86                         print(customer)
87                 else:
88                     print("Sorry! No customer found")
89
90             elif choice == "3":
91                 customer_id = int(input("Enter Customer ID: "))
92                 if self.customers_dao.get_customer_by_id(customer_id):
93                     new_email = input("Enter new Email (Leave blank to keep unchanged): ").strip() or None
94                     new_phone = input("Enter new Phone (Leave blank to keep unchanged): ").strip() or None
95                     new_address = input("Enter new Address (Leave blank to keep unchanged): ").strip() or None
96                     self.customers_dao.update_customer_info(customer_id, new_email, new_phone, new_address)
97                     print("Customer information updated successfully.")
98                 else:
99                     print("Not found.")
100
101             elif choice == "4":
102                 customer_id = int(input("Enter Customer ID: "))
103                 if self.customers_dao.get_customer_by_id(customer_id):
104                     self.customers_dao.delete_customer(customer_id)
105                     print("Customer deleted successfully.")
106                 else:
107                     print("id not found")
108
109             elif choice == "5":
```

Version control

```
    elif choice == "5":
        customer_id = int(input("Enter Customer ID: "))
        customer = self.customers_dao.get_customer_by_id(customer_id)
        if customer:
            print(customer.CustomerID, customer.FirstName, customer.LastName, customer.Email, customer.Phone,
                  customer.Address)
        else:
            print("Customer not found.")

    elif choice == "6":
        customer_id = int(input("Enter Customer ID: "))
        if self.customers_dao.get_customer_by_id(customer_id):
            total_orders = self.customers_dao.calculate_total_orders(customer_id)
            print(f"Total orders for customer {customer_id}: {total_orders}")
        else:
            print("Customer not found.")

    elif choice == "7":
        print("program Ends.")
        break
    else:
        print("Invalid choice. Please enter a valid option.")

1 usage
132 def manage_product(self):
133     print("\n*****PRODUCT MENU*****")
134     while True:
135         print("\nMenu:")
136         print("1. Add Product(c)")
137         print("2. Get all Product Details(R)")
138         print("3. Update Product Information(U)")
139         print("4. Delete Product(D)")
140         print("5. View Specific Product Details")
141         print("6. Is Product In Stock?")
142         print("7. Exit")

143         choice = input("Enter your choice (1-7): ")

146         if choice == "1":
147             product_id = input("Enter Product ID: ")
148             product_name = input("Enter Product Name: ")
149             description = input("Enter the Description: ")
150             price = float(input("Enter the Price: "))
151             product = Product(product_id, product_name, description, price)
152             self.products_dao.add_products(product)
153             print("Product added successfully.")

155         elif choice == "2":
156             products = self.products_dao.get_all_products()
157             if products:
158                 for product in products:
159                     print(product)
160             else:
161                 print("No Product found")
```

Version control

```
    while True:
        print("\nMenu:")
        print("1. Add Product(c)")
        print("2. Get all Product Details(R)")
        print("3. Update Product Information(U)")
        print("4. Delete Product(D)")
        print("5. View Specific Product Details")
        print("6. Is Product In Stock?")
        print("7. Exit")

        choice = input("Enter your choice (1-7): ")

        if choice == "1":
            product_id = input("Enter Product ID: ")
            product_name = input("Enter Product Name: ")
            description = input("Enter the Description: ")
            price = float(input("Enter the Price: "))
            product = Product(product_id, product_name, description, price)
            self.products_dao.add_products(product)
            print("Product added successfully.")

        elif choice == "2":
            products = self.products_dao.get_all_products()
            if products:
                for product in products:
                    print(product)
            else:
                print("No Product found")
```

Version control

```
161     print("No Product found")
162
163 elif choice == "3":
164     product_id = int(input("Enter Product ID: "))
165     if self.products_dao.get_product_by_id(product_id):
166         new_price = input("Enter new Price (leave blank to keep unchanged): ").strip() or None
167         self.products_dao.update_product_info(product_id, new_price)
168         print("Product information updated successfully.")
169     else:
170         print("Not found.")
171
172 elif choice == "4":
173     product_id = int(input("Enter Product ID: "))
174     if self.products_dao.get_product_by_id(product_id):
175         self.products_dao.delete_products(product_id)
176         print("Product deleted successfully.")
177     else:
178         print("Product not found")
179
180 elif choice == "5":
181     product_id = int(input("Enter Product ID: "))
182     product = self.products_dao.get_product_by_id(product_id)
183     if product:
184         print(product.ProductID, product.ProductName, product.Description, product.Price)
185     else:
186         print("Product not found.")
187
188 elif choice == "6":
189     product_id = int(input("Enter Product ID: "))
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
```

MainClass

Version control

```
188     elif choice == "6":
189         product_id = int(input("Enter Product ID: "))
190         if self.products_dao.get_product_by_id(product_id):
191             if self.products_dao.is_product_in_stock(product_id):
192                 print("Currently in Stock")
193             else:
194                 print("Out of Stock")
195         else:
196             print("Product not found.")
197
198     elif choice == "7":
199         print("Exiting program.")
200         break
201     else:
202         print("Invalid choice. Please enter a valid option.")
203
204
205
206
207
208
209
210
211
212
213
214
215
```

1 usage

```
def manage_orders(self):
    print("\n*****ORDER MENU*****")
    while True:
        print("\nMenu:")
        print("1. Create the Order(C)")
        print("2. Display the Orders(R)")
        print("3. Cancel the Order(D)")
        print("4. Get Order Details")
        print("5. Calculate Total Amount")
        print("6. UpdateOrderStatus (Processed/shipped)")
        print("7. Exit")
```

MainClass



```
order_id = int(input("Enter Order ID: "))
print(self.orders_dao.UpdateOrderStatus(order_id))

elif choice == "7":
    print("Exiting program.")
    break
else:
    print("Invalid choice. Please enter a valid option.")

1 usage
def manage_order_details(self):
    print("\n*****ORDER DETAILS MENU*****")
    while True:
        print("\nMenu:")
        print("1. Calculate Subtotal")
        print("2. Get Order Detail Info")
        print("3. Update Quantity")
        print("4. Add Discount")
        print("5. Display All Order Details")
        print("6. Exit")

        choice = input("Enter your choice (1-6): ")

        if choice == "1":
            order_detail_id = int(input("Enter Order Detail ID: "))
            subtotal = self.order_detail_dao.CalculateSubtotal(order_detail_id)
            if subtotal is not None:
                print("Subtotal:", subtotal)
            else:
```

```
elif choice == "4":
    order_detail_id = int(input("Enter Order Detail ID: "))
    discount_percentage = float(input("Enter discount percentage: "))
    self.order_detail_dao.AddDiscount(order_detail_id, discount_percentage)

elif choice == "5":
    orders = self.order_detail_dao.GetAllOrderDetail()
    if orders:
        for order in orders:
            print(order)
    else:
        print("No Order found")

elif choice == "6":
    print("Exiting program.")
    break
else:
    print("Invalid choice. Please enter a valid option.")

1 usage
def manage_inventory(self):
    print("\n*****MANAGE INVENTORY MENU*****")
    while True:
        print("\nMenu:")
        print("1. Get Product")
        print("2. Get QuantityInStock")
        print("3. Remove from Inventory")
        print("4. Update Stock Quantity")
MainClass  manage_order_details() : while True
```

## OUTPUTS

```
Welcome to TechShop, an Electronic Gadget Shop
1. Manage Customers
2. Manage Products
3. Manage Orders
4. Manage Order Details
5. Manage Inventory
6. Exit
Enter your choice: 1

*****CUSTOMER MENU*****

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 2
(1, 'Aman', 'Kumar', 'aman@gmail.com', '1234567897', 'Motihar', 1)
(2, 'Ansu', 'Singh', 'anshu@gmail.com', '1111111111', 'Bihar', 1)
(3, 'Amit', 'Sah', 'amit@gmail.com', '1111111112', 'Rachi', 1)
(4, 'Asmita', 'Si', 'asmita@gmail.com', '1111111121', 'Durgapur', 1)
(5, 'Swati', 'Singh', 'swati@gmail.com', '1111111122', 'Asansol', 2)

315/44 CR/LF_ UTF-8 _ 4 spaces _ Python 3.12 (TechShop Assignment1) _
```

```
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 2
(1, 'Aman', 'Kumar', 'aman@gmail.com', '1234567897', 'Motihar', 1)
(2, 'Ansu', 'Singh', 'anshu@gmail.com', '1111111111', 'Bihar', 1)
(3, 'Amit', 'Sah', 'amit@gmail.com', '1111111112', 'Rachi', 1)
(4, 'Asmita', 'Si', 'asmita@gmail.com', '1111111121', 'Durgapur', 1)
(5, 'Swati', 'Singh', 'swati@gmail.com', '1111111122', 'Asansol', 2)
(6, 'Prateeti', 'Maji', 'prateeti@gmail.com', '1111111133', 'Durgapur', 1)
(7, 'Anjali', 'Sneha', 'anjali@gmail.com', '1111111134', 'Raghunathpur', 1)
(8, 'Om', 'Kumar', 'om123@gmail.com', '1111111135', 'Bihar', 1)
(9, 'Roshan', 'Singh', 'roshan@gmail.com', '1111111137', 'Gandhinager', 0)
(10, 'Arava', 'Krishnavenni', 'arava@gmail.com', '1111111139', 'Chennai', 0)
(11, 'Jasmine', 'Roy', 'jasmine123@gmail.com', '2258746984', 'Mumbai', 0)

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 1
Enter ID: |
```

```
Project ▾ Version control ▾
Run ExceptionHandling × main (1) ×
... ▾
... ▾
    Menu:
    1. Add Customer(C)
    2. Get all the Customer Details(R)
    3. Update the Customer Information(U)
    4. Delete the Customer(D)
    5. View Specific Customer Details
    6. Calculate Total Orders
    7. Exit
Enter your choice (1-7): 1
Enter ID: 12
Enter First Name: Ankush
Enter Last Name: Singh
Enter Email: ankush@gmail.com
Enter Phone: 4587123650
Enter Address: Delhi
Customer added successfully.

    Menu:
    1. Add Customer(C)
    2. Get all the Customer Details(R)
    3. Update the Customer Information(U)
    4. Delete the Customer(D)
    5. View Specific Customer Details
    6. Calculate Total Orders
    7. Exit
Enter your choice (1-7): 1
TechShop_Assignment1 > main > main.py
315:44 CRLF UTF-8 4 spaces Python 3.12 (TechShop_Assignment1)
```

```
Project ▾ Version control ▾
Run ExceptionHandling × main (1) ×
... ▾
... ▾
    4. Delete the Customer(D)
    5. View Specific Customer Details
    6. Calculate Total Orders
    7. Exit
Enter your choice (1-7): 2
(1, 'Aman', 'Kumar', 'aman@gmail.com', '1234567897', 'Motihari', 1)
(2, 'Ansul', 'Singh', 'anshu@gmail.com', '1111111111', 'Bihar', 1)
(3, 'Amit', 'Sah', 'amit@gmail.com', '1111111112', 'Rachi', 1)
(4, 'Asmita', 'Si', 'asmita@gmail.com', '1111111121', 'Durgapur', 1)
(5, 'Swati', 'Singh', 'swati@gmail.com', '1111111122', 'Asansol', 2)
(6, 'Prateeti', 'Mai', 'prateeti@gmail.com', '1111111133', 'Durgapur', 1)
(7, 'Anjali', 'Sneha', 'anjali@gmail.com', '1111111134', 'Raghunathpur', 1)
(8, 'Om', 'Kumar', 'om123@gmail.com', '1111111136', 'Bihar', 1)
(9, 'Roshan', 'Singh', 'roshan@gmail.com', '1111111137', 'Gandhinagar', 0)
(10, 'Arava', 'Krishnavenni', 'arava@gmail.com', '1111111139', 'Chennai', 0)
(11, 'Jasmine', 'Roy', 'jasmine123@gmail.com', '2258746984', 'Mumbai', 0)
(12, 'Ankush', 'Singh', 'ankush@gmail.com', '4587123650', 'Delhi', None)

    Menu:
    1. Add Customer(C)
    2. Get all the Customer Details(R)
    3. Update the Customer Information(U)
    4. Delete the Customer(D)
    5. View Specific Customer Details
    6. Calculate Total Orders
    7. Exit
Enter your choice (1-7): 1
TechShop_Assignment1 > main > main.py
315:44 CRLF UTF-8 4 spaces Python 3.12 (TechShop_Assignment1) 08:14
```

```
(8, 'Om', 'Kumar', 'om123@gmail.com', '1111111136', 'Bihar', 1)
(9, 'Roshan', 'Singh', 'roshan@gmail.com', '1111111137', 'Gandhinagar', 0)
(10, 'Arava', 'Krishnavenni', 'arava@gmail.com', '1111111139', 'Chennai', 0)
(11, 'Jasmine', 'Roy', 'jasmine123@gmail.com', '2258746984', 'Mumbai', 0)
(12, 'Ankush', 'Singh', 'ankush@gmail.com', '4587123650', 'Delhi', None)

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 4
Enter Customer ID: 12
Customer deleted successfully.

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7):
```

```
315:44 CRLF UTF-8 4 spaces Python 3.12 (TechShop_Assignment1) 09:14

6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 4
Enter Customer ID: 12
Customer deleted successfully.

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): 6
Enter Customer ID: 5
Total orders for customer 5: 2

Menu:
1. Add Customer(C)
2. Get all the Customer Details(R)
3. Update the Customer Information(U)
4. Delete the Customer(D)
5. View Specific Customer Details
6. Calculate Total Orders
7. Exit
Enter your choice (1-7): |
```

```
*****PRODUCT MENU*****  
Menu:  
1. Add Product(C)  
2. Get all Product Details(R)  
3. Update Product Information(U)  
4. Delete Product(D)  
5. View Specific Product Details  
6. Is Product In Stock?  
7. Exit  
Enter your choice (1-7): 2  
(5, 'abc', 'dede', Decimal('78.37'))  
(11, 'Laptop', 'electronic gadget', Decimal('550000.15'))  
(12, 'Mouse', 'electronic gadget', Decimal('550.26'))  
(13, 'Keyboard', 'electronic gadget', Decimal('880.59'))  
(14, 'Printer', 'electronic gadget', Decimal('8800.37'))  
(15, 'Phone', 'electronic gadget', Decimal('19800.92'))  
(16, 'WallClock', 'Home appliances', Decimal('5000.14'))  
(17, 'Headphone', 'electronic gadget', Decimal('2200.15'))  
(18, 'Camera', 'electronic gadget', Decimal('220001.09'))  
(19, 'Smartwatch', 'electronic gadget', Decimal('2200.61'))  
(20, 'Speaker', 'Home appliances', Decimal('20000.85'))  
(21, 'Monitor', 'electronic gadget', Decimal('8000.00'))  
  
Menu:  
1. Add Product(C)  
2. Get all Product Details(R)  
TechShop_Assignment1 > main > main.py
```

```
(20, 'Speaker', 'Home appliances', Decimal('20000.85'))  
(21, 'Monitor', 'electronic gadget', Decimal('8000.00'))  
  
Menu:  
1. Add Product(C)  
2. Get all Product Details(R)  
3. Update Product Information(U)  
4. Delete Product(D)  
5. View Specific Product Details  
6. Is Product In Stock?  
7. Exit  
Enter your choice (1-7): 1  
Enter Product ID: 25  
Enter Product Name: Iphone  
Enter the Description: electronic gadget  
Enter the Price: 25842.33  
Product added successfully.  
  
Menu:  
1. Add Product(C)  
2. Get all Product Details(R)  
3. Update Product Information(U)  
4. Delete Product(D)  
5. View Specific Product Details  
6. Is Product In Stock?  
7. Exit  
Enter your choice (1-7): |  
TechShop_Assignment1 > main > main.py
```

```
Project TS TechShop_Assignment1 Version control
Run ExceptionHandling x main (1) x
... 5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): 2
(5, 'abc', 'dede', Decimal('78.37'))
(11, 'Laptop', 'electronic gadget', Decimal('550000.15'))
(12, 'Mouse', 'electronic gadget', Decimal('550.26'))
(13, 'Keyboard', 'electronic gadget', Decimal('880.59'))
(14, 'Printer', 'electronic gadget', Decimal('8800.37'))
(15, 'Phone', 'electronic gadget', Decimal('19800.92'))
(16, 'WallClock', 'Home appliances', Decimal('5000.14'))
(17, 'Headphone', 'electronic gadget', Decimal('2200.15'))
(18, 'Camera', 'electronic gadget', Decimal('220001.09'))
(19, 'Smartwatch', 'electronic gadget', Decimal('2200.61'))
(20, 'Speaker', 'Home appliances', Decimal('20000.85'))
(21, 'Monitor', 'electronic gadget', Decimal('8000.00'))
(25, 'Iphone', 'electronic gadget', Decimal('25842.33'))

Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): |
```

```
Project TS TechShop_Assignment1 Version control
Run ExceptionHandling x main (1) x
... 5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): 6
Enter Product ID: 14
Currently in Stock

Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): |
```

```
Project ▾ Run ExceptionHandling × main (1) ×
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): 6
Enter Product ID: 14
Currently in Stock

Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7): 5
Enter Product ID: 25
25 Iphone electronic gadget 25842.33

Menu:
1. Add Product(C)
2. Get all Product Details(R)
3. Update Product Information(U)
4. Delete Product(D)
5. View Specific Product Details
6. Is Product In Stock?
7. Exit
Enter your choice (1-7):
```

```
Project ▾ Run ExceptionHandling × main (1) ×
1. Create the Order(C)
2. Display the Orders(R)
3 Cancel the Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/shipped)
7. Exit
Enter your choice (1-8): 2
(101, 1, datetime.date(2024, 1, 1), Decimal('550000.15'), 'Shipped')
(102, 2, datetime.date(2024, 1, 2), Decimal('1100.52'), 'Pending')
(103, 3, datetime.date(2024, 1, 3), Decimal('2641.77'), 'Pending')
(104, 4, datetime.date(2024, 1, 4), Decimal('35201.48'), 'Pending')
(105, 5, datetime.date(2024, 1, 5), Decimal('99004.60'), 'Pending')
(106, 6, datetime.date(2024, 1, 6), Decimal('30000.84'), 'Pending')
(107, 7, datetime.date(2024, 1, 7), Decimal('15401.05'), 'Pending')
(108, 8, datetime.date(2024, 1, 8), Decimal('1760008.72'), 'Pending')
(111, 5, datetime.date(2024, 1, 11), None, 'Pending')

Menu:
1. Create the Order(C)
2. Display the Orders(R)
3 Cancel the Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/shipped)
7. Exit
Enter your choice (1-8):
```

```
Project TS TechShop_Assignment1 Version control 
Run ExceptionHandling main (1) 
... 
(108, 8, datetime.date(2024, 1, 8), Decimal('1760008.72'), 'Pending')
(111, 5, datetime.date(2024, 1, 11), None, 'Pending')

Menu:
1. Create the Order(C)
2. Display the Orders(R)
3 Cancel the Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/shipped)
7. Exit
Enter your choice (1-8): 4
Enter Order ID: 107
Order ID: 107
Customer ID: 7
Order Date: 2024-01-07
Total Amount: 15401.05

Menu:
1. Create the Order(C)
2. Display the Orders(R)
3 Cancel the Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/shipped)
7. Exit
Enter your choice (1-8): |
```

```
Project TS TechShop_Assignment1 Version control 
Run ExceptionHandling main (1) 
... 
Enter Order ID: 107
Order ID: 107
Customer ID: 7
Order Date: 2024-01-07
Total Amount: 15401.05

Menu:
1. Create the Order(C)
2. Display the Orders(R)
3 Cancel the Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/shipped)
7. Exit
Enter your choice (1-8): 6
Enter Order ID: 107
shipped

Menu:
1. Create the Order(C)
2. Display the Orders(R)
3 Cancel the Order(D)
4. Get Order Details
5. Calculate Total Amount
6. UpdateOrderStatus (Processed/shipped)
7. Exit
Enter your choice (1-8): |
```

```
Project TS TechShop_Assignment1 Version control
Run ExceptionHandling main (1) x
main (1) C G S O : & S ? - : D L
...:
6. Exit
Enter your choice: 4
*****ORDER DETAILS MENU*****
Menu:
1. Calculate Subtotal
2. Get Order Detail Info
3. Update Quantity
4. Add Discount
5. Display All OrderDetails
6. Exit
Enter your choice (1-8): 2
Enter Order Detail ID: 25
Order Detail ID: 25
Order ID: 105
Product Name: Phone
Quantity: 5

Menu:
1. Calculate Subtotal
2. Get Order Detail Info
3. Update Quantity
4. Add Discount
5. Display All OrderDetails
6. Exit
Enter your choice (1-8): |
```

```
Project TS TechShop_Assignment1 Version control
Run ExceptionHandling main (1) x
main (1) C G S O : & S ? - : D L
...:
Quantity: 5
Menu:
1. Calculate Subtotal
2. Get Order Detail Info
3. Update Quantity
4. Add Discount
5. Display All OrderDetails
6. Exit
Enter your choice (1-8): 5
(21, 101, 11, '1')
(22, 102, 12, '2')
(23, 103, 13, '3')
(24, 104, 14, '4')
(25, 105, 15, '5')
(26, 106, 16, '6')
(27, 107, 17, '7')
(28, 108, 18, '8')

Menu:
1. Calculate Subtotal
2. Get Order Detail Info
3. Update Quantity
4. Add Discount
5. Display All OrderDetails
6. Exit
Enter your choice (1-8):
```

Quantity: 5

Menu:

1. Calculate Subtotal
2. Get Order Detail Info
3. Update Quantity
4. Add Discount
5. Display All OrderDetails
6. Exit

Enter your choice (1-8): 5

(21, 101, 11, '1')  
(22, 102, 12, '2')  
(23, 103, 13, '3')  
(24, 104, 14, '4')  
(25, 105, 15, '5')  
(26, 106, 16, '6')  
(27, 107, 17, '7')  
(28, 108, 18, '8')

Menu:

1. Calculate Subtotal
2. Get Order Detail Info
3. Update Quantity
4. Add Discount
5. Display All OrderDetails
6. Exit

Enter your choice (1-8): 3

TechShop\_Assignment1 > main > main.py

315:44 CRLF UTF-8 4 spaces Python 3.12 (TechShop\_Assignment1) 06:18