NAME: SHIVAM SINGH ASSIGNMENT 1:

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).

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.

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.

OrderDetails Class:

Attributes:

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

Methods:

- CalculateSubtotal() - Calculate the subtotal for this order detail.
- GetOrderDetailInfo(): Retrieves and displays information about this order detail.
- UpdateQuantity(): Allows updating the quantity of the product in this order detail.
- AddDiscount(): Applies a discount to this order detail.

Inventory class:

Attributes:

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

Methods:

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

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

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).

Task 4: Composition:

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

- Orders Class with Composition:
    - 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.
    - 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:
    - 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.
    - 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:
    - 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:
    - 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.

Task 5: Exceptions handling

- **Data Validation**:

    - Challenge: Validate user inputs and data from external sources (e.g., user registration, order placement).
    - Scenario: When a user enters an invalid email address during registration.
    - Exception Handling: Throw a custom InvalidDataException with a clear error message.
- **Inventory Management**:
    - Challenge: Handling inventory-related issues, such as selling more products than are in stock.
    - Scenario: When processing an order with a quantity that exceeds the available stock.
    - Exception Handling: Throw an InsufficientStockException and update the order status accordingly.
- **Order Processing:**
    - Challenge: Ensuring the order details are consistent and complete before processing.
    - Scenario: When an order detail lacks a product reference.
    - Exception Handling: Throw an IncompleteOrderException with a message explaining the issue.
- **Payment Processing:**
    - Challenge: Handling payment failures or declined transactions.
    - Scenario: When processing a payment for an order and the payment is declined.
    - Exception Handling: Handle payment-specific exceptions (e.g., PaymentFailedException) and initiate retry or cancellation processes.
- **File I/O (e.g., Logging):**
    - Challenge: Logging errors and events to files or databases.
    - Scenario: When an error occurs during data persistence (e.g., writing a log entry).
    - Exception Handling: Handle file I/O exceptions (e.g., IOException) and log them appropriately.
- **Database Access:**
    - Challenge: Managing database connections and queries.
    - Scenario: When executing a SQL query and the database is offline.
    - Exception Handling: Handle database-specific exceptions (e.g., SqlException) and implement connection retries or failover mechanisms.
- **Concurrency Control:**
    - Challenge: Preventing data corruption in multi-user scenarios.
    - Scenario: When two users simultaneously attempt to update the same order.
    - Exception Handling: Implement optimistic concurrency control and handle ConcurrencyException by notifying users to retry.
- **Security and Authentication:**
    - Challenge: Ensuring secure access and handling unauthorized access attempts.
    - Scenario: When a user tries to access sensitive information without proper authentication.
    - Exception Handling: Implement custom AuthenticationException and AuthorizationException to handle security-related issues.

Task 6: Collections

- **Managing Products List:**

    - Challenge: Maintaining a list of products available for sale (List<Products>).
    - Scenario: Adding, updating, and removing products from the list.
    - Solution: Implement methods to add, update, and remove products. Handle exceptions for duplicate products, invalid updates, or removal of products with existing orders.
- **Managing Orders List:**
    - Challenge: Maintaining a list of customer orders (List<Orders>).
    - Scenario: Adding new orders, updating order statuses, and removing canceled orders.
    - Solution: Implement methods to add new orders, update order statuses, and remove canceled orders. Ensure that updates are synchronized with inventory and payment records.
- **Sorting Orders by Date:**
    - Challenge: Sorting orders by order date in ascending or descending order.
    - Scenario: Retrieving and displaying orders based on specific date ranges.
    - Solution: Use the List<Orders> collection and provide custom sorting methods for order date. Consider implementing SortedList if you need frequent sorting operations.
- **Inventory Management with SortedList:**
    - Challenge: Managing product inventory with a SortedList based on product IDs.
    - Scenario: Tracking the quantity in stock for each product and quickly retrieving inventory information.
    - Solution: Implement a SortedList<int, Inventory> where keys are product IDs. Ensure that inventory updates are synchronized with product additions and removals.
- **Handling Inventory Updates:**
    - Challenge: Ensuring that inventory is updated correctly when processing orders.
    - Scenario: Decrementing product quantities in stock when orders are placed.
    - Solution: Implement a method to update inventory quantities when orders are processed. Handle exceptions for insufficient stock.
- **Product Search and Retrieval:**
    - Challenge: Implementing a search functionality to find products based on various criteria (e.g., name, category).
    - Scenario: Allowing customers to search for products.
    - Solution: Implement custom search methods using LINQ queries on the List<Products> collection. Handle exceptions for invalid search criteria.
- **Duplicate Product Handling:**
    - Challenge: Preventing duplicate products from being added to the list.
    - Scenario: When a product with the same name or SKU is added.
    - Solution: Implement logic to check for duplicates before adding a product to the list. Raise exceptions or return error messages for duplicates.
- **Payment Records List:**
    - Challenge: Managing a list of payment records for orders (List<PaymentClass>).
    - Scenario: Recording and updating payment information for each order.
    - Solution: Implement methods to record payments, update payment statuses, and handle payment errors. Ensure that payment records are consistent with order records.
- **OrderDetails and Products Relationship:**
    - Challenge: Managing the relationship between OrderDetails and Products.

- Scenario: Ensuring that order details accurately reflect the products available in the inventory.
- Solution: Implement methods to validate product availability in the inventory before adding order details. Handle exceptions for unavailable products.

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.

1: Customer Registration

Description: When a new customer registers on the TechShop website, their information (e.g., name, email, phone) needs to be stored in the database.

Task: Implement a registration form and database connectivity to insert new customer records. Ensure proper data validation and error handling for duplicate email addresses.

2: Product Catalog Management

Description: TechShop regularly updates its product catalog with new items and changes in product details (e.g., price, description). These changes need to be reflected in the database.

Task: Create an interface to manage the product catalog. Implement database connectivity to update product information. Handle changes in product details and ensure data consistency.

3: Placing Customer Orders

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

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

4: Tracking Order Status

Description: Customers and employees need to track the status of their orders. The order status information is stored in the database.

Task: Develop a feature that allows users to view the status of their orders. Implement database connectivity to retrieve and display order status information.

5: Inventory Management

Description: TechShop needs to manage product inventory, including adding new products, updating stock levels, and removing discontinued items.

Task: Create an inventory management system with database connectivity. Implement features for adding new products, updating quantities, and handling discontinued products.

6: Sales Reporting

Description: TechShop management requires sales reports for business analysis. The sales data is stored in the database.

Task: Design and implement a reporting system that retrieves sales data from the database and generates reports based on specified criteria.

7: Customer Account Updates

Description: Customers may need to update their account information, such as changing their email address or phone number.

Task: Implement a user profile management feature with database connectivity to allow customers to update their account details. Ensure data validation and integrity.

8: Payment Processing

Description: When customers make payments for their orders, the payment details (e.g., payment method, amount) must be recorded in the database.

Task: Develop a payment processing system that interacts with the database to record payment transactions, validate payment information, and handle errors.

9: Product Search and Recommendations

Description: Customers should be able to search for products based on various criteria (e.g., name, category) and receive product recommendations.

Task: Implement a product search and recommendation engine that uses database connectivity to retrieve relevant product information.

TASK 1-6:

Customers.py

```
class Customers:
    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 = []

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

    def get_customer_details(self):
        print(f'CustomerID: {self.CustomerID}, FirstName: {self.FirstName},
LastName: {self.LastName}, Email: {self.Email}, Phone: {self.Phone}, Address:
{self.Address}')

    def update_customer_info(self, email=None, phone=None, address=None):
        if email is not None:
            self.Email = email
        if phone is not None:
            self.Phone = phone
        if address is not None:
            self.Address = address
```

Products.py

```
class Products:
    def __init__(self, product_id, product_name, description, price):
        self.ProductID = product_id
        self.ProductName = product_name
        self.Description = description
        self.Price = price

    def get_product_details(self):
        print(f'ProductID: {self.ProductID}, ProductName: {self.ProductName},
Description: {self.Description}, Price: {self.Price}')

    def update_product_info(self, price=None, description=None):
        if price is not None:
            self.Price = price
        if description is not None:
            self.Description = description

    def is_product_in_stock(self):
        # Add implementation logic to check if the product is in stock
        return True
```

Orders.py

```
from datetime import datetime

class Orders:
    def __init__(self, order_id, customer, order_date, total_amount):
        self.OrderID = order_id
        self.Customer = customer
        self.OrderDate = order_date
        self.TotalAmount = total_amount
        self.order_details = []

    def calculate_total_amount(self):
        # Add implementation logic to calculate total order amount
        return 0

    def get_order_details(self):
        for order_detail in self.order_details:
            order_detail.get_order_detail_info()

    def update_order_status(self, new_status):
        # Add implementation logic to update order status
        pass

    def cancel_order(self):
        # Add implementation logic to cancel the order and adjust stock levels
        pass
```

OrderDetails.py

```
class OrderDetails:
    def __init__(self, order_detail_id, order, product, quantity):
        self.OrderDetailID = order_detail_id
        self.Order = order
        self.Product = product
        self.Quantity = quantity

    def calculate_subtotal(self):
        # Add implementation logic to calculate subtotal for this order detail
        return self.Product.Price * self.Quantity

    def get_order_detail_info(self):
        print(f'OrderDetailID: {self.OrderDetailID}, Product:
{self.Product.ProductName}, Quantity: {self.Quantity}, Subtotal:
{self.calculate_subtotal()}')

    def update_quantity(self, new_quantity):
        self.Quantity = new_quantity

    def add_discount(self, discount_amount):
        # Add implementation logic to apply a discount to this order detail
        pass
```

Inventory.py

```
from datetime import datetime

class Inventory:
    def __init__(self, inventory_id, product, quantity_in_stock,
last_stock_update):
        self.InventoryID = inventory_id
        self.Product = product
        self.QuantityInStock = quantity_in_stock
        self.LastStockUpdate = last_stock_update

    def get_product(self):
        # Add logic to retrieve the product associated with this inventory item
        pass

    def get_quantity_in_stock(self):
        # Add logic to get the current quantity of the product in stock
        pass

    def add_to_inventory(self, quantity):
        # Add logic to add a specified quantity of the product to the inventory
        pass

    def remove_from_inventory(self, quantity):
        # Add logic to remove a specified quantity of the product from the
inventory
        pass

    def update_stock_quantity(self, new_quantity):
        # Add logic to update the stock quantity to a new value
        pass

    def is_product_available(self, quantity_to_check):
        # Add logic to check if a specified quantity of the product is
available in the inventory
        pass

    def get_inventory_value(self):
        # Add logic to calculate the total value of the products in the
inventory
        pass

    def list_low_stock_products(self, threshold):
        # Add logic to list products with quantities below a specified
threshold, indicating low stock
        pass

    def list_out_of_stock_products(self):
        # Add logic to list products that are out of stock
        pass

    def list_all_products(self):
        # Add logic to list all products in the inventory, along with their
quantities
        pass
```

Output without main.py:

```
"D:\apps\charm\assignment 1\TS\.venv\Scripts\python.exe" "D:\apps\charm\assignment 1\

Process finished with exit code 0
```

main.py:

```
from datetime import datetime
from Customers import Customers
from Products import Products
from Orders import Orders
from OrderDetails import OrderDetails
from Inventory import Inventory

def main():
    # Create sample instances
    shivam = Customers(customer_id=1, first_name="Shivam", last_name="Singh",
email="shivasingh414@gmail.com",
                       phone="8340508631", address="Street 12 Sector 9/B Bokaro
Steel City Jharkhand ")

    # Create sample instances
    laptop = Products(product_id=1, product_name="Iphone 15 pro max",
description="Real Titanium 1TB", price=150000)

    # Create sample instances
    order = Orders(order_id=1, customer=shivam, order_date=datetime.now(),
total_amount=150000)

    # Create sample instances
    order_detail = OrderDetails(order_detail_id=1, order=order, product=laptop,
quantity=2)

    # Create sample instances
    laptop_inventory = Inventory(inventory_id=1, product=laptop,
quantity_in_stock=10, last_stock_update=datetime.now())

    # Interactive menu for testing
    while True:
        print("\nTechShop Management System")
        print("1. View Customer Details")
        print("2. View Product Details")
        print("3. View Order Details")
        print("4. View Inventory Details")
        print("5. Exit")

        choice = input("Enter your choice (1-5): ")

        if choice == "1":
            shivam.get_customer_details()

        elif choice == "2":
            laptop.get_product_details()

        elif choice == "3":
            order.get_order_details()
```

```
        elif choice == "4":
            laptop_inventory.list_all_products()

        elif choice == "5":
            print("Exiting TechShop Management System.")
            break

        else:
            print("Invalid choice. Please enter a number between 1 and 5.")

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

Output:

```
TechShop Management System
1. View Customer Details
2. View Product Details
3. View Order Details
4. View Inventory Details
5. Exit
Enter your choice (1-5):
```

1.

```
Enter your choice (1-5): 1
CustomerID: 1, FirstName: Shivam, LastName: Singh, Email: shivasingh414@gmail.com, Phone: 8340508631, Address: Street 12 Sector 9/B Bokaro Steel City Jharkhand
```

2.

```
Enter your choice (1-5): 2
ProductID: 1, ProductName: Iphone 15 pro max, Description: Real Titanium 1TB, Price: 150000
```

3/4/5.

```
Enter your choice (1-5): 4

TechShop Management System
1. View Customer Details
2. View Product Details
3. View Order Details
4. View Inventory Details
5. Exit
Enter your choice (1-5): 5
Exiting TechShop Management System.

Process finished with exit code 0
```

TASK 7:

In this we will make a "Database connector.py" separately as taught in our class, as it will increase our readability as well as reusability of the code at the same time will keep the codes separate and by importing it in the main file our job will be done.

DatabaseConnector.py

```
import mysql.connector

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

    def open_connection(self):
        try:
            self.connection = mysql.connector.connect(
                host=self.host,
                user=self.username,
                password=self.password,
                database=self.database
            )
            print("Connected to the database.")
        except mysql.connector.Error as err:
            print(f"Error: {err}")

    def close_connection(self):
        if self.connection:
            self.connection.close()
            print("Connection closed.")

    def execute_query(self, query, values=None):
        cursor = self.connection.cursor()
        try:
            cursor.execute(query, values)
            self.connection.commit()
            print("Query executed successfully.")
        except mysql.connector.Error as err:
            print(f"Error: {err}")
        finally:
            cursor.close()
```

For database connectivity: python -m pip install mysql-connector-python on cmd

And in Pycharm<Settings<Project<Interpreter<pip<mysql-connector (install)< mysql-connector-python(install) and then it is ready to "import mysql.connector"

main.py:

```
# main.py
from datetime import datetime
from Customers import Customers
from Products import Products
from Orders import Orders
from OrderDetails import OrderDetails
from Inventory import Inventory
from DatabaseConnector import DatabaseConnector

# Set your MySQL database credentials
db_host = "localhost"
db_user = "root"
db_password = "765795"
db_name = "TechShopDB"

# Create a DatabaseConnector instance
db_connector = DatabaseConnector(host=db_host, username=db_user,
password=db_password, database=db_name)

# Open the database connection
db_connector.open_connection()

def insert_data():
    # Create a sample instance for Karthika Mam
    karthika = Customers(customer_id=2, first_name="Karthika", last_name="Mam",
email="karthika@example.com",
                         phone="9876543210", address="123 Main Street")

    # Insert data into the Customers table
    db_connector.execute_query(
        "INSERT INTO Customers (CustomerID, FirstName, LastName, Email, Phone,
Address) VALUES (%s, %s, %s, %s, %s, %s)",
        (karthika.CustomerID, karthika.FirstName, karthika.LastName,
karthika.Email, karthika.Phone, karthika.Address)
    )

    # Create a sample instance for MacBook
    macbook = Products(product_id=2, product_name="MacBook", description="Apple
MacBook Pro", price=2000)

    # Insert data into the Products table for MacBook
    db_connector.execute_query(
        "INSERT INTO Products (ProductID, ProductName, Description, Price)
VALUES (%s, %s, %s, %s)",
        (macbook.ProductID, macbook.ProductName, macbook.Description,
macbook.Price)
    )

    # Create a sample instance for iPhone
    iphone = Products(product_id=3, product_name="iPhone", description="Apple
iPhone 13 Pro", price=1200)

    # Insert data into the Products table for iPhone
    db_connector.execute_query(
        "INSERT INTO Products (ProductID, ProductName, Description, Price)
VALUES (%s, %s, %s, %s)",
        (iphone.ProductID, iphone.ProductName, iphone.Description,
iphone.Price)
```

```
    )

    # Create a sample instance for an order
    order = Orders(order_id=2, customer=karthika, order_date=datetime.now(),
total_amount=3200)

    # Insert data into the Orders table
    db_connector.execute_query(
        "INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount)
VALUES (%s, %s, %s, %s)",
        (order.OrderID, order.Customer.CustomerID, order.OrderDate,
order.TotalAmount)
    )

    # Create a sample instance for Inventory
    inventory = Inventory(inventory_id=2, product=macbook, quantity_in_stock=5,
last_stock_update=datetime.now())

    # Insert data into the Inventory table
    db_connector.execute_query(
        "INSERT INTO Inventory (InventoryID, ProductID, QuantityInStock,
LastStockUpdate) VALUES (%s, %s, %s, %s)",
        (inventory.InventoryID, inventory.Product.ProductID,
inventory.QuantityInStock, inventory.LastStockUpdate)
    )

    # Create a sample instance for OrderDetails
    order_detail = OrderDetails(order_detail_id=2, order=order, product=iphone,
quantity=3)

    # Insert data into the OrderDetails table
    db_connector.execute_query(
        "INSERT INTO OrderDetails (OrderDetailID, OrderID, ProductID, Quantity)
VALUES (%s, %s, %s, %s)",
        (order_detail.OrderDetailID, order_detail.Order.OrderID,
order_detail.Product.ProductID, order_detail.Quantity)
    )

# Insert data
insert_data()

# Close the database connection when done
db_connector.close_connection()
```

Output at pycharm:

```
Connected to the database.
Error: 1062 (23000): Duplicate entry '2' for key 'customers.PRIMARY'
Error: 1062 (23000): Duplicate entry '2' for key 'products.PRIMARY'
Error: 1062 (23000): Duplicate entry '3' for key 'products.PRIMARY'
Error: 1062 (23000): Duplicate entry '2' for key 'orders.PRIMARY'
Error: 1062 (23000): Duplicate entry '2' for key 'inventory.PRIMARY'
Error: 1062 (23000): Duplicate entry '2' for key 'orderdetails.PRIMARY'
Connection closed.
```

Here in our output duplicate entry is showing because at the time of code we have run it once and we have already added the data so let's check that on "My SQL Command Line".

Output:

```
mysql> select*from customers;
+------------+-----------+----------+------------------------+------------+-----------------------------------------------+
| CustomerID | FirstName | LastName | Email                  | Phone      | Address                                       |
+------------+-----------+----------+------------------------+------------+-----------------------------------------------+
|          1 | Shivam    | Singh    | shivasingh414@gmail.com | 8340508631 | Street 12 Sector 9/B Bokaro Steel City Jharkhand |
|          2 | Karthika  | Mam      | karthika@example.com   | 9876543210 | 123 Main Street                               |
+------------+-----------+----------+------------------------+------------+-----------------------------------------------+
2 rows in set (0.00 sec)

mysql> select*from products;
+-----------+-------------+--------------------+---------+
| ProductID | ProductName | Description        | Price   |
+-----------+-------------+--------------------+---------+
|         2 | MacBook     | Apple MacBook Pro  | 2000.00 |
|         3 | iPhone      | Apple iPhone 13 Pro | 1200.00 |
+-----------+-------------+--------------------+---------+
2 rows in set (0.01 sec)

mysql> select*from inventory;
+-------------+-----------+-----------------+--------------------+
| InventoryID | ProductID | QuantityInStock | LastStockUpdate    |
+-------------+-----------+-----------------+--------------------+
|           2 |         2 |               5 | 2024-02-01 16:09:55 |
+-------------+-----------+-----------------+--------------------+
1 row in set (0.00 sec)

mysql> select*from orders;
+---------+------------+---------------------+-------------+
| OrderID | CustomerID | OrderDate           | TotalAmount |
+---------+------------+---------------------+-------------+
|       2 |          2 | 2024-02-01 16:09:55 |     3200.00 |
+---------+------------+---------------------+-------------+
1 row in set (0.00 sec)

mysql> select*from orderdetails;
+---------------+---------+-----------+----------+
| OrderDetailID | OrderID | ProductID | Quantity |
+---------------+---------+-----------+----------+
|             2 |       2 |         3 |        3 |
+---------------+---------+-----------+----------+
1 row in set (0.00 sec)
```

The database used is "TechShopDB" in which tables were same as we did it in our "Mysql assignment1" with some alteration.