# ASSIGNMENT

**TITLE:** TechShop, an electronic gadgets shop

**NAME:** J316 Sherin Sandra J

**DATE:** 15 / 04 / 2025

**TRAINER:** Mr. Madhu Kalla

# DATABASE

## TASK 1:

1. Create the database named "TechShop"
2. Define the schema for the Customers, Products, Orders, OrderDetails and Inventory tables
based on the provided schema.
3. Create an ERD (Entity Relationship Diagram) for the database.
4. Create appropriate Primary Key and Foreign Key constraints for referential integrity.
5. Insert at least 10 sample records into each of the following tables.
    a. Customers
    b. Products
    c. Orders
    d. OrderDetails
    e. Inventory

```
mysql> create database TechShop;
Query OK, 1 row affected (0.08 sec)
```

```
mysql> use Techshop;
Database changed
```

```
mysql> create table Customers(
    -> CustomerID int primary key auto_increment,
    -> Firstname varchar(50) NOT NULL,
    -> Lastname varchar(50) NOT NULL,
    -> Email varchar(100) unique not null,
    -> Address text
    -> );
Query OK, 0 rows affected (0.18 sec)
```

```
mysql> create table Products(
    -> ProductID int primary key auto_increment,
    -> ProductName varchar(100) NOT NULL,
    -> Description text NOT NULL,
    -> Price decimal(10,2)
    -> );
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> create table Orders(
    -> OrderID int primary key auto_increment,
    -> CustomerID int,
    -> OrderDate datetime default current_timestamp,
    -> TotalAmount decimal(10,2) NOT NULL,
    -> foreign key (CustomerID) references Customers(CustomerID) on delete cascade
    -> );
Query OK, 0 rows affected (0.10 sec)
```

```
mysql> create table OrderDetails(
    -> OrderDetailID int primary key auto_increment,
    -> OrderID int,
    -> ProductID int,
    -> Quantity int NOT NULL,
    -> foreign key (OrderID) references Orders(OrderID) on delete cascade,
    -> foreign key (ProductID) references Products(ProductID) on delete cascade
    -> );
Query OK, 0 rows affected (0.23 sec)
```

\

```
mysql> create table Inventory(
    -> InventoryID int primary key auto_increment,
    -> ProductID int,
    -> QuantityInStock int NOT NULL,
    -> LastStockUpdate datetime default current_timestamp NOT NULL,
    -> foreign key (ProductID) references Products(ProductID) on delete cascade
    -> );
Query OK, 0 rows affected (0.19 sec)
```

```
mysql> INSERT INTO Customers (FirstName, LastName, Email, Phone, Address) VALUES
    -> ('John', 'Doe', 'john@example.com', '1234567890', '123 Main St'),
    -> ('Alice', 'Smith', 'alice@example.com', '9876543210', '456 Market St'),
    -> ('Bob', 'Brown', 'bob@example.com', '1112223333', '789 Elm St'),
    -> ('Charlie', 'Davis', 'charlie@example.com', '4445556666', '101 Oak St'),
    -> ('David', 'Wilson', 'david@example.com', '7778889999', '202 Pine St'),
    -> ('Emma', 'Thomas', 'emma@example.com', '2223334444', '303 Maple St'),
    -> ('Frank', 'Johnson', 'frank@example.com', '5556667777', '404 Birch St'),
    -> ('Grace', 'Lee', 'grace@example.com', '6667778888', '505 Cedar St'),
    -> ('Henry', 'White', 'henry@example.com', '9990001111', '606 Spruce St'),
    -> ('Ivy', 'Clark', 'ivy@example.com', '1231231234', '707 Walnut St');
Query OK, 10 rows affected (0.03 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

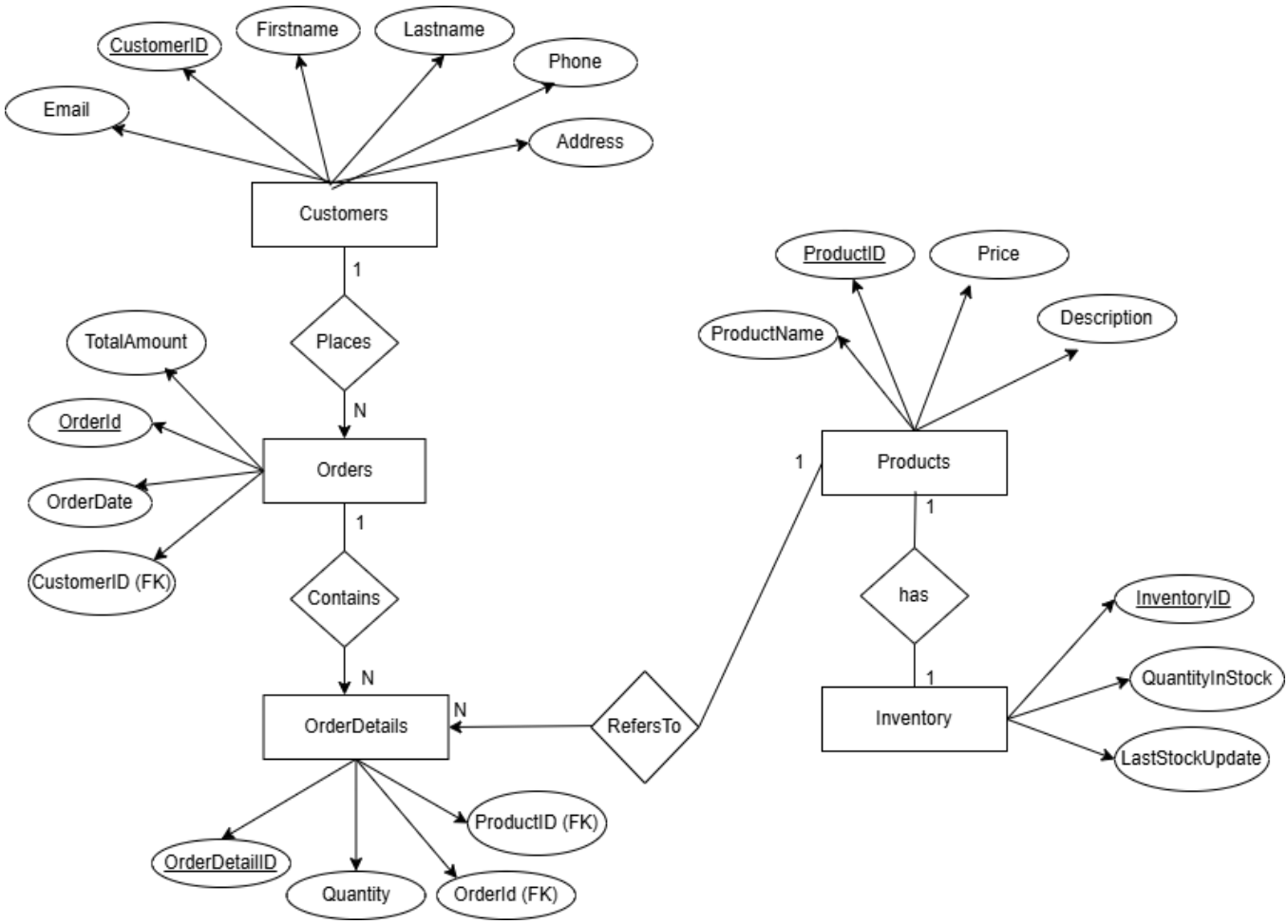
```
mysql> INSERT INTO Products (ProductName, Description, Price) VALUES
    -> ('Laptop', 'Gaming Laptop with 16GB RAM', 120000.00),
    -> ('Smartphone', '5G smartphone with AMOLED display', 7999.99),
    -> ('Tablet', '10-inch tablet with stylus support', 49999.99),
    -> ('Smartwatch', 'Water-resistant smartwatch with GPS', 19999.99),
    -> ('Wireless Earbuds', 'Noise-cancelling wireless earbuds', 14999.99),
    -> ('Monitor', '27-inch 4K UHD Monitor', 35000.00),
    -> ('Keyboard', 'Mechanical RGB keyboard', 999.99),
    -> ('Mouse', 'Wireless ergonomic mouse', 499.99),
    -> ('External Hard Drive', '2TB SSD external hard drive', 2999.99),
    -> ('Speaker', 'Portable Bluetooth speaker', 799.99);
Query OK, 10 rows affected (0.02 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

```
mysql> INSERT INTO Orders (CustomerID, OrderDate, TotalAmount) VALUES
    -> (1, '2024-03-01 10:00:00', 120000.00),
    -> (2, '2024-03-02 12:30:00', 7999.99),
    -> (3, '2024-03-03 15:45:00', 49999.99),
    -> (4, '2024-03-04 18:00:00', 19999.99),
    -> (5, '2024-03-05 20:15:00', 14999.99),
    -> (6, '2024-03-06 22:45:00', 35000.00),
    -> (7, '2024-03-07 09:30:00', 999.99),
    -> (8, '2024-03-08 11:45:00', 499.99),
    -> (9, '2024-03-09 14:00:00', 2999.99),
    -> (10, '2024-03-10 16:30:00', 799.99);
Query OK, 10 rows affected (0.03 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

```
mysql> INSERT INTO OrderDetails (OrderID, ProductID, Quantity) VALUES
    -> (1, 1, 1),
    -> (2, 2, 1),
    -> (3, 3, 1),
    -> (4, 4, 2),
    -> (5, 5, 1),
    -> (6, 6, 1),
    -> (7, 7, 1),
    -> (8, 8, 2),
    -> (9, 9, 1),
    -> (10, 10, 1);
Query OK, 10 rows affected (0.02 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

```
mysql> INSERT INTO Inventory (ProductID, QuantityInStock, LastStockUpdate) VALUES
    -> (1, 10, '2024-03-01 09:00:00'),
    -> (2, 20, '2024-03-02 10:30:00'),
    -> (3, 15, '2024-03-03 11:45:00'),
    -> (4, 30, '2024-03-04 12:00:00'),
    -> (5, 25, '2024-03-05 13:15:00'),
    -> (6, 18, '2024-03-06 14:30:00'),
    -> (7, 22, '2024-03-07 15:45:00'),
    -> (8, 12, '2024-03-08 16:00:00'),
    -> (9, 8, '2024-03-09 17:30:00'),
    -> (10, 14, '2024-03-10 18:45:00');
Query OK, 10 rows affected (0.01 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

# ER DIAGRAM:


CustomerID
Firstname
Lastname
Phone
Email
Address
Customers
1
Places
N
TotalAmount
OrderId
Orders
OrderDate
CustomerID (FK)
1
Contains
N
OrderDetails
N
RefersTo
1
Products
ProductID
Price
ProductName
Description
1
has
1
Inventory
InventoryID
QuantityInStock
LastStockUpdate
OrderDetailID
Quantity
OrderId (FK)
ProductID (FK)

desc customers;

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| CustomerID | int | NO | PRI | NULL | auto_increment |
| Firstname | varchar(50) | NO | | NULL | |
| Lastname | varchar(50) | NO | | NULL | |
| Email | varchar(100) | NO | UNI | NULL | |
| Address | text | YES | | NULL | |

desc products;

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| ProductID | int | NO | PRI | NULL | auto_increment |
| ProductName | varchar(100) | NO | | NULL | |
| Description | text | NO | | NULL | |
| Price | decimal(10,2) | YES | | NULL | |

desc orders;

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| OrderID | int | NO | PRI | NULL | auto_increment |
| CustomerID | int | YES | MUL | NULL | |
| OrderDate | datetime | YES | | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
| TotalAmount | decimal(10,2) | NO | | NULL | |

desc orderdetails;

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| OrderDetailID | int | NO | PRI | NULL | auto_increment |
| OrderID | int | YES | MUL | NULL | |
| ProductID | int | YES | MUL | NULL | |
| Quantity | int | NO | | NULL | |

desc inventory;

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| InventoryID | int | NO | PRI | NULL | auto_increment |
| ProductID | int | YES | MUL | NULL | |
| QuantityInStock | int | NO | | NULL | |
| LastStockUpdate | datetime | NO | | CURRENT_TIMESTAMP | DEFAULT_GENERATED |

## TASK 2:

## 1. Write an SQL query to retrieve the First Name, Last Name, and Email of all customers.

```
mysql> select firstname, lastname ,email from customers;
+-----------+----------+---------------------+
| firstname | lastname | email               |
+-----------+----------+---------------------+
| John      | Doe      | john@example.com    |
| Alice     | Smith    | alice@example.com   |
| Bob       | Brown    | bob@example.com     |
| Charlie   | Davis    | charlie@example.com |
| David     | Wilson   | david@example.com   |
| Emma      | Thomas   | emma@example.com    |
| Frank     | Johnson  | frank@example.com   |
| Grace     | Lee      | grace@example.com   |
| Henry     | White    | henry@example.com   |
| Ivy       | Clark    | ivy@example.com     |
+-----------+----------+---------------------+
10 rows in set (0.00 sec)
```

## 2.Write an SQL query to list all orders along with their order dates and the customer's first and last name.

```
mysql> select o.orderid, o.orderdate, c.firstname, c.lastname
    -> from orders o inner join customers c
    -> on o.customerid=c.customerid;
+---------+---------------------+-----------+----------+
| orderid | orderdate           | firstname | lastname |
+---------+---------------------+-----------+----------+
|       1 | 2024-03-01 10:00:00 | John      | Doe      |
|       2 | 2024-03-02 12:30:00 | Alice     | Smith    |
|       3 | 2024-03-03 15:45:00 | Bob       | Brown    |
|       4 | 2024-03-04 18:00:00 | Charlie   | Davis    |
|       5 | 2024-03-05 20:15:00 | David     | Wilson   |
|       6 | 2024-03-06 22:45:00 | Emma      | Thomas   |
|       7 | 2024-03-07 09:30:00 | Frank     | Johnson  |
|       8 | 2024-03-08 11:45:00 | Grace     | Lee      |
|       9 | 2024-03-09 14:00:00 | Henry     | White    |
|      10 | 2024-03-10 16:30:00 | Ivy       | Clark    |
+---------+---------------------+-----------+----------+
10 rows in set (0.00 sec)
```

**3.Write an SQL Query to insert a new customer record into the "Customers" table, include customer information such as name, email and address.**

```
mysql> insert into customers (firstname, lastname, email, phone, address) values
('Sherin' , 'Sandra' , 'sandra.sherin@j.com', '9892839829' , '204 Roe street, Bel
ton, BA');
Query OK, 1 row affected (0.02 sec)
```

**4.Write an SQL query to update the prices of all electronic gadgets in the "Products" table by increasing them by 10%.**

```
mysql> update products
    -> set price = price + (price*0.10);
Query OK, 10 rows affected, 8 warnings (0.02 sec)
Rows matched: 10  Changed: 10  Warnings: 8

mysql> select * from products;
+-----------+--------------------+-------------------------------------+-----------+
| ProductID | ProductName        | Description                         | Price     |
+-----------+--------------------+-------------------------------------+-----------+
|         1 | Laptop             | Gaming Laptop with 16GB RAM         | 132000.00 |
|         2 | Smartphone         | 5G smartphone with AMOLED display   |   8799.99 |
|         3 | Tablet             | 10-inch tablet with stylus support  |  54999.99 |
|         4 | Smartwatch         | Water-resistant smartwatch with GPS |  21999.99 |
|         5 | Wireless Earbuds   | Noise-cancelling wireless earbuds   |  16499.99 |
|         6 | Monitor            | 27-inch 4K UHD Monitor              |  38500.00 |
|         7 | Keyboard           | Mechanical RGB keyboard             |   1099.99 |
|         8 | Mouse              | Wireless ergonomic mouse            |    549.99 |
|         9 | External Hard Drive | 2TB SSD external hard drive        |   3299.99 |
|        10 | Speaker            | Portable Bluetooth speaker          |    879.99 |
+-----------+--------------------+-------------------------------------+-----------+
10 rows in set (0.00 sec)
```

**5. Write an SQL query to delete a specific order and its associated order details from the "Orders" and "OrderDetails" tables. Allow users to input the order ID as a parameter.**

```
mysql> delete from orders where orderid=5;
Query OK, 1 row affected (0.02 sec)

mysql> select * from orders;
+---------+------------+---------------------+-------------+
| OrderID | CustomerID | OrderDate           | TotalAmount |
+---------+------------+---------------------+-------------+
|       1 |          1 | 2024-03-01 10:00:00 |   120000.00 |
|       2 |          2 | 2024-03-02 12:30:00 |     7999.99 |
|       3 |          3 | 2024-03-03 15:45:00 |    49999.99 |
|       4 |          4 | 2024-03-04 18:00:00 |    19999.99 |
|       6 |          6 | 2024-03-06 22:45:00 |    35000.00 |
|       7 |          7 | 2024-03-07 09:30:00 |      999.99 |
|       8 |          8 | 2024-03-08 11:45:00 |      499.99 |
|       9 |          9 | 2024-03-09 14:00:00 |     2999.99 |
|      10 |         10 | 2024-03-10 16:30:00 |      799.99 |
+---------+------------+---------------------+-------------+
9 rows in set (0.00 sec)

mysql> select * from orderdetails;
+---------------+---------+-----------+----------+
| OrderDetailID | OrderID | ProductID | Quantity |
+---------------+---------+-----------+----------+
|             1 |       1 |         1 |        1 |
|             2 |       2 |         2 |        1 |
|             3 |       3 |         3 |        1 |
|             4 |       4 |         4 |        2 |
|             6 |       6 |         6 |        1 |
|             7 |       7 |         7 |        1 |
|             8 |       8 |         8 |        2 |
|             9 |       9 |         9 |        1 |
|            10 |      10 |        10 |        1 |
+---------------+---------+-----------+----------+
9 rows in set (0.00 sec)
```

## 6. Write an SQL query to insert a new order into the "Orders" table. Include the customer ID, order date, and any other necessary information.

```
mysql> insert into orders(customerid,orderdate,totalamount) values(11, '2024-03-11 16:38:00', 8999.99);
Query OK, 1 row affected (0.01 sec)

mysql> select * from orders;
+---------+------------+---------------------+-------------+
| OrderID | CustomerID | OrderDate           | TotalAmount |
+---------+------------+---------------------+-------------+
|       1 |          1 | 2024-03-01 10:00:00 |   120000.00 |
|       2 |          2 | 2024-03-02 12:30:00 |     7999.99 |
|       3 |          3 | 2024-03-03 15:45:00 |    49999.99 |
|       4 |          4 | 2024-03-04 18:00:00 |    19999.99 |
|       6 |          6 | 2024-03-06 22:45:00 |    35000.00 |
|       7 |          7 | 2024-03-07 09:30:00 |      999.99 |
|       8 |          8 | 2024-03-08 11:45:00 |      499.99 |
|       9 |          9 | 2024-03-09 14:00:00 |     2999.99 |
|      10 |         10 | 2024-03-10 16:30:00 |      799.99 |
|      11 |         11 | 2024-03-11 16:38:00 |     8999.99 |
+---------+------------+---------------------+-------------+
10 rows in set (0.00 sec)
```

## 7. Write an SQL query to update the contact information (e.g., email and address) of a specific customer in the "Customers" table. Allow users to input the customer ID and new contact information.

```
mysql> update customers
    -> set Phone=8938338600, Email='sera.jemi@home',address='890 tet street, Kalvi, PA'
    -> where customerid=6;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> select * from customers;
+------------+-----------+----------+------------+--------------------+---------------------------+
| CustomerID | Firstname | Lastname | Phone      | Email              | Address                   |
+------------+-----------+----------+------------+--------------------+---------------------------+
|          1 | John      | Doe      | 1234567890 | john@example.com    | 123 Main St               |
|          2 | Alice     | Smith    | 9876543210 | alice@example.com   | 456 Market St             |
|          3 | Bob       | Brown    | 1112223333 | bob@example.com     | 789 Elm St                |
|          4 | Charlie   | Davis    | 4445556666 | charlie@example.com | 101 Oak St                |
|          5 | David     | Wilson   | 7778889999 | david@example.com   | 202 Pine St               |
|          6 | Emma      | Thomas   | 8938338600 | sera.jemi@home      | 890 tet street, Kalvi, PA |
|          7 | Frank     | Johnson  | 5556667777 | frank@example.com   | 404 Birch St              |
|          8 | Grace     | Lee      | 6667778888 | grace@example.com   | 505 Cedar St              |
|          9 | Henry     | White    | 9990001111 | henry@example.com   | 606 Spruce St             |
|         10 | Ivy       | Clark    | 1231231234 | ivy@example.com     | 707 Walnut St             |
|         11 | Sherin    | Sandra   | 9892839829 | sandra.sherin@j.com | 204 Roe street, Belton, BA |
+------------+-----------+----------+------------+--------------------+---------------------------+
11 rows in set (0.00 sec)
```

## 8. Write an SQL query to recalculate and update the total cost of each order in the "Orders" table based on the prices and quantities in the "OrderDetails" table.

```
mysql> update orders
    -> set totalamount = (
    ->   select ifnull(sum(od.quantity * p.price), 0)
    ->   from orderdetails od
    ->   join products p on od.productid = p.productid
    ->   where od.orderid = orders.orderid
    -> );
mysql> select * from orders;
+---------+------------+---------------------+-------------+-----------+
| OrderID | CustomerID | OrderDate           | TotalAmount | status    |
+---------+------------+---------------------+-------------+-----------+
|       1 |          1 | 2024-03-01 10:00:00 |   132000.00 | Delivered |
|       2 |          2 | 2024-03-02 12:30:00 |     8799.99 | pending   |
|       4 |          4 | 2024-03-04 18:00:00 |    43999.98 | Delivered |
|       6 |          6 | 2024-03-06 22:45:00 |    38500.00 | pending   |
|       7 |          7 | 2024-03-07 09:30:00 |     1099.99 | Delivered |
|       8 |          8 | 2024-03-08 11:45:00 |     1099.98 | cancelled |
|       9 |          9 | 2024-03-09 14:00:00 |     3299.99 | pending   |
|      10 |         10 | 2024-03-10 16:30:00 |      879.99 | cancelled |
|      11 |         11 | 2024-03-11 16:38:00 |     2999.98 | pending   |
|      12 |         12 | 2025-03-23 23:07:14 |   264000.00 | Confirmed |
|      13 |          1 | 2025-03-30 23:15:27 |        0.00 | Confirmed |
+---------+------------+---------------------+-------------+-----------+
```

**9. Write an SQL query to delete all orders and their associated order details for a specific customer from the "Orders" and "OrderDetails" tables. Allow users to input the customer ID as a parameter.**

```
mysql> delete from orders where customerid=3;
Query OK, 1 row affected (0.07 sec)

mysql> select * from orders;
+---------+------------+---------------------+-------------+
| OrderID | CustomerID | OrderDate           | TotalAmount |
+---------+------------+---------------------+-------------+
|       1 |          1 | 2024-03-01 10:00:00 |   120000.00 |
|       2 |          2 | 2024-03-02 12:30:00 |     7999.99 |
|       4 |          4 | 2024-03-04 18:00:00 |    19999.99 |
|       6 |          6 | 2024-03-06 22:45:00 |    35000.00 |
|       7 |          7 | 2024-03-07 09:30:00 |      999.99 |
|       8 |          8 | 2024-03-08 11:45:00 |      499.99 |
|       9 |          9 | 2024-03-09 14:00:00 |     2999.99 |
|      10 |         10 | 2024-03-10 16:30:00 |      799.99 |
|      11 |         11 | 2024-03-11 16:38:00 |     8999.99 |
+---------+------------+---------------------+-------------+
9 rows in set (0.00 sec)

mysql> select * from orderdetails;
+---------------+---------+-----------+----------+
| OrderDetailID | OrderID | ProductID | Quantity |
+---------------+---------+-----------+----------+
|             1 |       1 |         1 |        1 |
|             2 |       2 |         2 |        1 |
|             4 |       4 |         4 |        2 |
|             6 |       6 |         6 |        1 |
|             7 |       7 |         7 |        1 |
|             8 |       8 |         8 |        2 |
|             9 |       9 |         9 |        1 |
|            10 |      10 |        10 |        1 |
|            12 |      11 |        11 |        2 |
+---------------+---------+-----------+----------+
9 rows in set (0.01 sec)
```

**10. Write an SQL query to insert a new electronic gadget product into the "Products" table, including product name, category, price, and any other relevant details.**

```
mysql> insert into products (productname,description,price) values('Headphones','RGB wireless
headphones',699.00);
Query OK, 1 row affected (0.01 sec)

mysql> select * from products;
+-----------+--------------------+-------------------------------------+-----------+
| ProductID | ProductName        | Description                         | Price     |
+-----------+--------------------+-------------------------------------+-----------+
|         1 | Laptop             | Gaming Laptop with 16GB RAM         | 132000.00 |
|         2 | Smartphone         | 5G smartphone with AMOLED display   |   8799.99 |
|         3 | Tablet             | 10-inch tablet with stylus support  |  54999.99 |
|         4 | Smartwatch         | Water-resistant smartwatch with GPS |  21999.99 |
|         5 | Wireless Earbuds   | Noise-cancelling wireless earbuds   |  16499.99 |
|         6 | Monitor            | 27-inch 4K UHD Monitor              |  38500.00 |
|         7 | Keyboard           | Mechanical RGB keyboard             |   1099.99 |
|         8 | Mouse              | Wireless ergonomic mouse            |    549.99 |
|         9 | External Hard Drive | 2TB SSD external hard drive        |   3299.99 |
|        10 | Speaker            | Portable Bluetooth speaker          |    879.99 |
|        11 | Gaming Mouse       | Ergonomic gaming mouse with RGB     |   1499.99 |
|        12 | Headphones         | RGB wireless headphones             |    699.00 |
+-----------+--------------------+-------------------------------------+-----------+
12 rows in set (0.00 sec)
```

**11. Write an SQL query to update the status of a specific order in the "Orders" table (e.g., from "Pending" to "Shipped"). Allow users to input the order ID and the new status.**

```
mysql> alter table orders add status varchar(20) default 'pending';
Query OK, 0 rows affected (0.29 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> select * from orders;
+---------+------------+---------------------+-------------+-----------+
| OrderID | CustomerID | OrderDate           | TotalAmount | status    |
+---------+------------+---------------------+-------------+-----------+
|       1 |          1 | 2024-03-01 10:00:00 |   120000.00 | Delivered |
|       2 |          2 | 2024-03-02 12:30:00 |     7999.99 | pending   |
|       4 |          4 | 2024-03-04 18:00:00 |    19999.99 | Delivered |
|       6 |          6 | 2024-03-06 22:45:00 |    35000.00 | pending   |
|       7 |          7 | 2024-03-07 09:30:00 |      999.99 | pending   |
|       8 |          8 | 2024-03-08 11:45:00 |      499.99 | cancelled |
|       9 |          9 | 2024-03-09 14:00:00 |     2999.99 | pending   |
|      10 |         10 | 2024-03-10 16:30:00 |      799.99 | cancelled |
|      11 |         11 | 2024-03-11 16:38:00 |     8999.99 | pending   |
+---------+------------+---------------------+-------------+-----------+
9 rows in set (0.00 sec)
```

```
mysql> update orders set status='Delivered' where orderid=7;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from orders;
+---------+------------+---------------------+-------------+-----------+
| OrderID | CustomerID | OrderDate           | TotalAmount | status    |
+---------+------------+---------------------+-------------+-----------+
|       1 |          1 | 2024-03-01 10:00:00 |   120000.00 | Delivered |
|       2 |          2 | 2024-03-02 12:30:00 |     7999.99 | pending   |
|       4 |          4 | 2024-03-04 18:00:00 |    19999.99 | Delivered |
|       6 |          6 | 2024-03-06 22:45:00 |    35000.00 | pending   |
|       7 |          7 | 2024-03-07 09:30:00 |      999.99 | Delivered |
|       8 |          8 | 2024-03-08 11:45:00 |      499.99 | cancelled |
|       9 |          9 | 2024-03-09 14:00:00 |     2999.99 | pending   |
|      10 |         10 | 2024-03-10 16:30:00 |      799.99 | cancelled |
|      11 |         11 | 2024-03-11 16:38:00 |     8999.99 | pending   |
+---------+------------+---------------------+-------------+-----------+
9 rows in set (0.00 sec)
```

**12. Write an SQL query to calculate and update the number of orders placed by each customer in the "Customers" table based on the data in the "Orders" table.**

```
mysql> alter table customers add OrderCount int default 0;
Query OK, 0 rows affected (0.16 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> update customers c set c.ordercount = (select COUNT(*) from orders o where
o.customerid = c.customerid);
Query OK, 10 rows affected (0.02 sec)
Rows matched: 12  Changed: 10  Warnings: 0
```

```
mysql> select * from customers;
+------------+-----------+----------+------------+---------------------+---------------------------+------------+
| CustomerID | Firstname | Lastname | Phone      | Email               | Address                   | OrderCount |
+------------+-----------+----------+------------+---------------------+---------------------------+------------+
|          1 | John      | Doe      | 1234567890 | john@example.com    | 123 Main St               |          2 |
|          2 | Alice     | Smith    | 9876543210 | alice@example.com   | 456 Market St             |          1 |
|          3 | Bob       | Brown    | 1112223333 | bob@example.com     | 789 Elm St                |          0 |
|          4 | Charlie   | Davis    | 4445556666 | charlie@example.com | 101 Oak St                |          1 |
|          5 | David     | Wilson   | 7778889999 | david@example.com   | 202 Pine St               |          0 |
|          6 | Emma      | Thomas   | 8938338600 | sera.jemi@home      | 890 tet street, Kalvi, PA |          1 |
|          7 | Frank     | Johnson  | 5556667777 | frank@example.com   | 404 Birch St              |          1 |
|          8 | Grace     | Lee      | 6667778888 | grace@example.com   | 505 Cedar St              |          1 |
|          9 | Henry     | White    | 9990001111 | henry@example.com   | 606 Spruce St             |          1 |
|         10 | Ivy       | Clark    | 1231231234 | ivy@example.com     | 707 Walnut St             |          1 |
|         11 | Sherin    | Sandra   | 9892839829 | sandra.sherin@j.com | 204 Roe street, Belton, BA |          1 |
|         12 | Maya      | Ram      | 9940086456 | mata.ram@example.com | 675 kul street, Heton, JA |          1 |
+------------+-----------+----------+------------+---------------------+---------------------------+------------+
12 rows in set (0.00 sec)
```

## TASK 3:

## 1. Write an SQL query to retrieve a list of all orders along with customer information (e.g., customer name) for each order.

```
mysql> select o.orderid, o.orderdate, o.totalamount, o.status, c.firstname, c.lastname
    -> from orders o
    -> left join customers c on o.customerid = c.customerid;
+---------+---------------------+-------------+-----------+-----------+----------+
| orderid | orderdate           | totalamount | status    | firstname | lastname |
+---------+---------------------+-------------+-----------+-----------+----------+
|       1 | 2024-03-01 10:00:00 |   120000.00 | Delivered | John      | Doe      |
|       2 | 2024-03-02 12:30:00 |     7999.99 | pending   | Alice     | Smith    |
|       4 | 2024-03-04 18:00:00 |    19999.99 | Delivered | Charlie   | Davis    |
|       6 | 2024-03-06 22:45:00 |    35000.00 | pending   | Emma      | Thomas   |
|       7 | 2024-03-07 09:30:00 |      999.99 | Delivered | Frank     | Johnson  |
|       8 | 2024-03-08 11:45:00 |      499.99 | cancelled | Grace     | Lee      |
|       9 | 2024-03-09 14:00:00 |     2999.99 | pending   | Henry     | White    |
|      10 | 2024-03-10 16:30:00 |      799.99 | cancelled | Ivy       | Clark    |
|      11 | 2024-03-11 16:38:00 |     8999.99 | pending   | Sherin    | Sandra   |
+---------+---------------------+-------------+-----------+-----------+----------+
9 rows in set (0.01 sec)
```

## 2. Write an SQL query to find the total revenue generated by each electronic gadget product. Include the product name and the total revenue.

```
mysql> select p.productname, SUM(p.price * o.quantity) as TotalRevenue
    -> from products p join orderdetails o
    -> on p.productid = o.productid
    -> group by p.productname;
+---------------------+--------------+
| productname         | TotalRevenue |
+---------------------+--------------+
| Laptop              |    132000.00 |
| Smartphone          |      8799.99 |
| Smartwatch          |     43999.98 |
| Monitor             |     38500.00 |
| Keyboard            |      1099.99 |
| Mouse               |      1099.98 |
| External Hard Drive |      3299.99 |
| Speaker             |       879.99 |
| Gaming Mouse        |      2999.98 |
+---------------------+--------------+
9 rows in set (0.01 sec)
```

**3. Write an SQL query to list all customers who have made at least one purchase. Include their names and contact information.**

```
mysql> select c.customerid, o.orderid, c.firstname, c.lastname, c.phone, c.email
    -> from customers c join orders o
    -> on c.customerid = o.customerid;
+------------+---------+-----------+----------+------------+---------------------+
| customerid | orderid | firstname | lastname | phone      | email               |
+------------+---------+-----------+----------+------------+---------------------+
|          1 |       1 | John      | Doe      | 1234567890 | john@example.com    |
|          2 |       2 | Alice     | Smith    | 9876543210 | alice@example.com   |
|          4 |       4 | Charlie   | Davis    | 4445556666 | charlie@example.com |
|          6 |       6 | Emma      | Thomas   | 8938338600 | sera.jemi@home      |
|          7 |       7 | Frank     | Johnson  | 5556667777 | frank@example.com   |
|          8 |       8 | Grace     | Lee      | 6667778888 | grace@example.com   |
|          9 |       9 | Henry     | White    | 9990001111 | henry@example.com   |
|         10 |      10 | Ivy       | Clark    | 1231231234 | ivy@example.com     |
|         11 |      11 | Sherin    | Sandra   | 9892839829 | sandra.sherin@j.com |
+------------+---------+-----------+----------+------------+---------------------+
9 rows in set (0.00 sec)
```

**4. Write an SQL query to find the most popular electronic gadget, which is the one with the highest total quantity ordered. Include the product name and the total quantity ordered.**

```
mysql> select p.productname, o.quantity as TotalQuantitySold
    -> from products p join orderdetails o on p.productid = o.productid
    -> WHERE o.Quantity = (SELECT MAX(Quantity) FROM OrderDetails);
+--------------+-------------------+
| productname  | TotalQuantitySold |
+--------------+-------------------+
| Smartwatch   |                 2 |
| Mouse        |                 2 |
| Gaming Mouse |                 2 |
+--------------+-------------------+
3 rows in set (0.01 sec)
```

**5. Write an SQL query to retrieve a list of electronic gadgets along with their corresponding categories.**

```
mysql> select productname, category from products;
+---------------------+--------------------+
| productname         | category           |
+---------------------+--------------------+
| Laptop              | Computing          |
| Smartphone          | Mobile Devices     |
| Tablet              | Mobile Devices     |
| Smartwatch          | Wearables          |
| Wireless Earbuds    | Audio Accessories  |
| Monitor             | Computing          |
| Keyboard            | Peripherals        |
| Mouse               | Peripherals        |
| External Hard Drive | Storage Devices    |
| Speaker             | Audio Accessories  |
| Gaming Mouse        | Peripherals        |
| Headphones          | Audio Accessories  |
+---------------------+--------------------+
12 rows in set (0.00 sec)
```

**6. Write an SQL query to calculate the average order value for each customer. Include the customer's name and their average order value.**

```
mysql> select c.firstname, c.lastname , AVG(o.totalamount) as AverageOrderValue
    -> from customers c join orders o on c.customerid = o.customerid
    -> group by c.customerid;
+-----------+----------+-------------------+
| firstname | lastname | AverageOrderValue |
+-----------+----------+-------------------+
| John      | Doe      |     120000.000000 |
| Alice     | Smith    |       7999.990000 |
| Charlie   | Davis    |      19999.990000 |
| Emma      | Thomas   |      35000.000000 |
| Frank     | Johnson  |        999.990000 |
| Grace     | Lee      |        499.990000 |
| Henry     | White    |       2999.990000 |
| Ivy       | Clark    |        799.990000 |
| Sherin    | Sandra   |       8999.990000 |
+-----------+----------+-------------------+
9 rows in set (0.03 sec)
```

**7. Write an SQL query to find the order with the highest total revenue. Include the order ID, customer information, and the total revenue.**

```
mysql> select c.firstname, c.lastname, o.totalamount as HighestTotalRevenue
    -> from customers c join orders o on c.customerid = o.customerid
    -> where o.totalamount = (select MAX(totalamount) from orders);
+-----------+----------+----------------------+
| firstname | lastname | HighestTotalRevenue |
+-----------+----------+----------------------+
| John      | Doe      |           120000.00 |
+-----------+----------+----------------------+
1 row in set (0.01 sec)
```

**8. Write an SQL query to list electronic gadgets and the number of times each product has been ordered.**

```
mysql> select p.productid, p.productname, sum(o.quantity) as TimeSold
    -> from products p join orderdetails o on p.productid = o.productid
    -> group by p.productid;
+-----------+--------------------+----------+
| productid | productname        | TimeSold |
+-----------+--------------------+----------+
|         1 | Laptop             |        1 |
|         2 | Smartphone         |        1 |
|         4 | Smartwatch         |        2 |
|         6 | Monitor            |        1 |
|         7 | Keyboard           |        1 |
|         8 | Mouse              |        2 |
|         9 | External Hard Drive |        1 |
|        10 | Speaker            |        1 |
|        11 | Gaming Mouse       |        2 |
+-----------+--------------------+----------+
9 rows in set (0.00 sec)
```

**9.Write an SQL query to find customers who have purchased a specific electronic gadget product. Allow users to input the product name as a parameter.**

```
mysql> select c.firstname, c.lastname, p.productname
    -> from customers c
    -> join orders o on c.customerid = o.customerid
    -> join orderdetails od on o.orderid = od.orderid
    -> join products p on od.productid = p.productid
    -> where p.productname = 'laptop';
+-----------+----------+-------------+
| firstname | lastname | productname |
+-----------+----------+-------------+
| John      | Doe      | Laptop      |
| Maya      | Ram      | Laptop      |
+-----------+----------+-------------+
2 rows in set (0.00 sec)
```

**10. Write an SQL query to calculate the total revenue generated by all orders placed within a specific time period. Allow users to input the start and end dates as parameters.**

```
mysql> select SUM(totalamount) from orders where orderdate between '2024-03-01' and '2024-03-07';
+-----------------+
| SUM(totalamount) |
+-----------------+
|       182999.98 |
+-----------------+
1 row in set (0.01 sec)
```

## TASK 4:

### 1. Write an SQL query to find out which customers have not placed any orders.

```
mysql> select customerid, firstname, lastname
    -> from customers
    -> where customerid NOT IN ( select customerid from orders);
+------------+-----------+----------+
| customerid | firstname | lastname |
+------------+-----------+----------+
|          3 | Bob       | Brown    |
|          5 | David     | Wilson   |
+------------+-----------+----------+
2 rows in set (0.02 sec)
```

### 2. Write an SQL query to find the total number of products available for sale.

```
mysql> select SUM(quantityinstock) as TotalProd from inventory;
+-----------+
| TotalProd |
+-----------+
|       174 |
+-----------+
1 row in set (0.00 sec)
```

### 3. Write an SQL query to calculate the total revenue generated by TechShop.

```
mysql> select ( select SUM(totalamount) from orders ) as TotalRev;
+-----------+
| TotalRev  |
+-----------+
| 277299.92 |
+-----------+
1 row in set (0.00 sec)
```

**4. Write an SQL query to calculate the average quantity ordered for products in a specific category. Allow users to input the category name as a parameter.**

```
mysql> select p.category, AVG(o.quantity) as AvgQunatity
    -> from products p join orderdetails o on p.productid = o.productid
    -> where p.category = 'peripherals'
    -> group by p.category;
+-------------+-------------+
| category    | AvgQunatity |
+-------------+-------------+
| Peripherals |      1.6667 |
+-------------+-------------+
1 row in set (0.00 sec)
```

**5. Write an SQL query to calculate the total revenue generated by a specific customer. Allow users to input the customer ID as a parameter.**

```
mysql> select sum(totalamount) as totalrev
    -> from orders
    -> where customerid in (
    ->     select customerid
    ->     from customers
    ->     where customerid = 4
    -> );
+----------+
| totalrev |
+----------+
| 19999.99 |
+----------+
1 row in set (0.01 sec)
```

**6. Write an SQL query to find the customers who have placed the most orders. List their names and the number of orders they've placed.**

```
mysql> select firstname, lastname, ordercount from customers
    -> where ordercount = (select MAX(ordercount) from customers);
+-----------+----------+------------+
| firstname | lastname | ordercount |
+-----------+----------+------------+
| John      | Doe      |          2 |
+-----------+----------+------------+
1 row in set (0.00 sec)
```

**7. Write an SQL query to find the most popular product category, which is the one with the highest total quantity ordered across all orders.**

```
mysql> select p.category as PopularCategory, SUM(o.quantity) as TotQuant from products p
 join orderdetails o on p.productid = o.productid
    -> group by p.category
    -> order by TotQuant DESC
    -> LIMIT 1;
+-----------------+----------+
| PopularCategory | TotQuant |
+-----------------+----------+
| Peripherals     |        5 |
+-----------------+----------+
1 row in set (0.00 sec)
```

**8. Write an SQL query to find the customer who has spent the most money (highest total revenue) on electronic gadgets. List their name and total spending.**

```
mysql> select c.firstname, c.lastname ,SUM(o.totalamount) as HighestSpent from
 customers c join orders o on c.customerid = o.customerid
    -> group by c.customerid
    -> order by HighestSpent DESC
    -> LIMIT 1;
+-----------+----------+--------------+
| firstname | lastname | HighestSpent |
+-----------+----------+--------------+
| John      | Doe      |    124999.99 |
+-----------+----------+--------------+
1 row in set (0.00 sec)
```

**9. Write an SQL query to calculate the average order value (total revenue divided by the number of orders) for all customers.**

```
mysql> select AVG(totalamount) as AverageOrderValue from orders;
+--------------------+
| AverageOrderValue |
+--------------------+
|      25663.628182 |
+--------------------+
1 row in set (0.00 sec)
```

## 10. Write an SQL query to find the total number of orders placed by each customer and list their names along with the order count.

```
mysql>  select firstname, lastname, ordercount from customers where ordercount>0;
+-----------+----------+------------+
| firstname | lastname | ordercount |
+-----------+----------+------------+
| John      | Doe      |          2 |
| Alice     | Smith    |          1 |
| Charlie   | Davis    |          1 |
| Emma      | Thomas   |          1 |
| Frank     | Johnson  |          1 |
| Grace     | Lee      |          1 |
| Henry     | White    |          1 |
| Ivy       | Clark    |          1 |
| Sherin    | Sandra   |          1 |
| Maya      | Ram      |          1 |
+-----------+----------+------------+
10 rows in set (0.00 sec)
```

```
mysql> select c.firstname, c.lastname , count(o.orderid) as ordercount from
customers c join orders o on c.customerid = o.customerid group by c.customer
id ;
+-----------+----------+------------+
| firstname | lastname | ordercount |
+-----------+----------+------------+
| John      | Doe      |          2 |
| Alice     | Smith    |          1 |
| Charlie   | Davis    |          1 |
| Emma      | Thomas   |          1 |
| Frank     | Johnson  |          1 |
| Grace     | Lee      |          1 |
| Henry     | White    |          1 |
| Ivy       | Clark    |          1 |
| Sherin    | Sandra   |          1 |
| Maya      | Ram      |          1 |
+-----------+----------+------------+
10 rows in set (0.00 sec)
```

# IMPLEMENT OOPs

## Task 1: Classes and Their Attributes

This task specifies the list of all the classes and their attributes as listed below:

**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

## 1. Customer.java

This class represents a customer in the TechShop system. It stores customer details and provides methods to update and display information.

```
package hexa.org.entity;


public class Customer {
	private int customerId;
	private String firstName;
	private String lastName;
	private String email;
```

```
    private String phone;
    private String address;

    public Customer() {
        super();
    }

    public Customer(int customerId, String firstName, String lastName,
String email, String phone, String address) {
        super();
        this.customerId = customerId;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.phone = phone;
        this.address = address;
    }

    public int getCustomerId() {
        return customerId;
    }

    public void setCustomerId(int customerId) {
        this.customerId = customerId;
    }
```

```
public String getFirstName() {
        return firstName;
}

public void setFirstName(String firstname) {
        this.firstName = firstname;
}

public String getLastName() {
        return lastName;
}

public void setLastname(String lastName) {
        this.lastName = lastName;
}

public String getEmail() {
        return email;
}

public void setEmail(String email) {
        this.email = email;
}
```

```
public String getPhone() {
        return phone;
}

public void setPhone(String phone) {
        this.phone = phone;
}

public String getAddress() {
        return address;
}

public void setAddress(String address) {
        this.address = address;
}

public int calculateTotalOrders() {
        return 0;
}

public void getCustomerDetails() {
        System.out.println("Firstname: "+firstName);
        System.out.println("Lastname: "+lastName);
        System.out.println("Email: "+email);
        System.out.println("Phone no: "+phone);
```

```
            System.out.println("Address: "+address);
        }

        public void updateCustomerInfo(String newEmail, String newPhone,
String newAddress) {
            this.email = newEmail;
            this.phone =newPhone;
            this.address = newAddress;
        }

        @Override
        public String toString() {
            return "Customer [customerId=" + customerId + ", firstName="
+ firstName + ", lastName=" + lastName + ", email="
                    + email + ", phone=" + phone + ", address=" +
address + "]";
        }


}
```

## 2. Product.java

Represents an electronic product with fields like ID, name, description, and price. Includes methods to display product details, update product info and check stock.

```
package hexa.org.entity;

public class Product {
    private int productId;
    private String productName;
    private String description;
    private double price;
    private String category;

    public Product(int productId, String productName, String description,
double price, String category) {
        this.productId = productId;
        this.productName = productName;
        this.description = description;
        this.price = price;
        this.category = category;
    }

    public int getProductId() {
        return productId;
    }
```

```
public void setProductId(int productId) {
    this.productId = productId;
}

public String getProductName() {
    return productName;
}

public void setProductName(String productName) {
    this.productName = productName;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public double getPrice() {
    return price;
}

public void setPrice(double price) {
```

```
        this.price = price;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    @Override
    public String toString() {
        return "Product{" +
                "productId=" + productId +
                ", productName='" + productName + '\'' +
                ", description='" + description + '\'' +
                ", price=" + price +
                ", category='" + category + '\'' +
                '}';
    }
}
```

## 3. Order.java

Represents a customer's order. Holds the customer, order date, total amount, and status.

Methods allow viewing, cancelling and managing order status.

```
package hexa.org.entity;

import java.time.LocalDateTime;
import java.util.List;

public class Order {
   private int orderId;
   private Customer customer;
   private LocalDateTime orderDate;
   private double totalAmount;
   private String orderStatus;
   private List<OrderDetail> orderDetails;

   public Order() {
            super();
      }

      public Order(int orderId, Customer customer, LocalDateTime orderDate, double
totalAmount, String orderStatus) {
            super();
            this.orderId = orderId;
            this.customer = customer;
            this.orderDate = orderDate;
            this.totalAmount = totalAmount;
            this.orderStatus = orderStatus;
      }

      public int getOrderId() {
      return orderId;
   }

   public void setOrderId(int orderId) {
      this.orderId = orderId;
   }

   public Customer getCustomer() {
      return customer;
   }
```

```
public void setCustomer(Customer customer) {
   this.customer = customer;
}

public LocalDateTime getOrderDate() {
   return orderDate;
}

public void setOrderDate(LocalDateTime orderDate) {
   this.orderDate = orderDate;
}

public double getTotalAmount() {
   return totalAmount;
}

public void setTotalAmount(double totalAmount) {
     if(totalAmount>=0) {
            this.totalAmount = totalAmount;
     }
     else {
            System.out.println("Amount cannot be negative");
     }
}

public String getOrderStatus() {
     return orderStatus;
}

public void setOrderStatus(String orderStatus) {
     this.orderStatus = orderStatus;
}
public double calculateTotalAmount() {
     return totalAmount;
}

public List<OrderDetail> getOrderDetails() {
   return orderDetails;
}

public void updateOrderStatus(String newStatus) {
   this.orderStatus = newStatus;
```

```
    }

    public void cancelOrder() {
        System.out.println("Order #" + orderId + " has been cancelled.");
        }

    @Override
        public String toString() {
                return "Order [orderId=" + orderId + ", customer=" + customer + ",
orderDate=" + orderDate + ", totalAmount="
                                + totalAmount + ", orderStatus=" + orderStatus + "]";
        }

        public void setOrderDetails(List<OrderDetail> orderDetails) {
                this.orderDetails = orderDetails;
        }

}
```

## 4. OrderDetail.java

Represents the link between a product and an order, including quantity and subtotal.

Also allows applying discounts and updating quantity.

```
package hexa.org.entity;


public class OrderDetail {
    private int orderDetailId;
    private Order order;
    private Product product;
    private int quantity;


    public OrderDetail() {
                super();
        }
```

```
        public OrderDetail(int orderDetailId, Order order, Product product, int quantity)
{
                super();
                this.orderDetailId = orderDetailId;
                this.order = order;
                this.product = product;
                this.quantity = quantity;
        }

        public int getOrderDetailId() {
      return orderDetailId;
   }

   public void setOrderDetailId(int orderDetailId) {
      this.orderDetailId = orderDetailId;
   }

   public Order getOrder() {
      return order;
   }

   public void setOrder(Order order) {
      this.order = order;
   }

   public Product getProduct() {
      return product;
```

```
}

public void setProduct(Product product) {
    this.product = product;
}

public int getQuantity() {
    return quantity;
}

public void setQuantity(int quantity) {
     if(quantity>0) {
            this.quantity = quantity;
     }
     else {
            System.out.println("Quantity should not be negative");
     }
}

public double calculateSubtotal() {
    return product.getPrice() * quantity;
}

public void getOrderDetailInfo() {
    System.out.println("Order Detail ID: " +orderDetailId);
    System.out.println("Product Name: " +product.getProductName());
    System.out.println("Quantity: " +quantity);
    System.out.println("Subtotal: " +calculateSubtotal());
```

```
    }

    public void updateQuantity(int newQty) {
        setQuantity(newQty);
    }

    public double addDiscount(double percent) {
        double subtotal = calculateSubtotal();
        double discountAmount = subtotal * (percent / 100);
        return subtotal - discountAmount;
    }

    @Override
         public String toString() {
                 return "OrderDetail [orderDetailId=" + orderDetailId + ", order=" + order
+ ", product=" + product
                                 + ", quantity=" + quantity + "]";
         }

}
```

## 5. Inventory.java:

The Inventory class manages the stock details of each product. It keeps track of the quantity in stock, the associated product, and the last updated time. The class provides functionality to update, check, and monitor stock levels.

```
package hexa.org.entity;

import java.time.LocalDateTime;

public class Inventory {
    private int inventoryId;
    private Product product;
    private int quantityInStock;
    private LocalDateTime lastStockUpdate;

    public Inventory() {
            super();
        }

        public Inventory(int inventoryId, Product product, int
quantityInStock, LocalDateTime lastStockUpdate) {
            super();
            this.inventoryId = inventoryId;
            this.product = product;
            this.quantityInStock = quantityInStock;
            this.lastStockUpdate = lastStockUpdate;
```

```
    }


     public int getInventoryId() {
    return inventoryId;
}

public void setInventoryId(int inventoryId) {
    this.inventoryId = inventoryId;
}

public Product getProduct() {
    return product;
}

public void setProduct(Product product) {
    this.product = product;
}

public int getQuantityInStock() {
    return quantityInStock;
}

public void setQuantityInStock(int quantityInStock) {
     if(quantityInStock>=0) {
```

```
            this.quantityInStock = quantityInStock;
        }
        else {
            System.out.println("Quantity should not be negative");
        }
    }

    public LocalDateTime getLastStockUpdate() {
       return lastStockUpdate;
    }

    public void setLastStockUpdate(LocalDateTime lastStockUpdate) {
       this.lastStockUpdate = lastStockUpdate;
    }

    public void addToInventory(int quantity) {
        this.quantityInStock += quantity;
        this.lastStockUpdate = LocalDateTime.now();
    }

    public void removeFromInventory(int quantity) {
        this.quantityInStock -= quantity;
        this.lastStockUpdate = LocalDateTime.now();
    }
```

```
    public void updateStockQuantity(int newQuantity) {
        setQuantityInStock(newQuantity);
        this.lastStockUpdate = LocalDateTime.now();
    }

    public boolean isProductAvailable(int quantityToCheck) {
        return quantityInStock>=quantityToCheck;
    }

    public double getInventoryValue() {
        return product.getPrice()*getQuantityInStock();
    }

    public void listLowStockProducts(int threshold) {
        if(quantityInStock<threshold) {
        System.out.println(product.getProductName()+ "is low in stock");
        }
    }
    public void listOutOfStockProducts() {
        if(quantityInStock==0) {
                System.out.println(product.getProductName()+" is out of
stock");
        }
    }
```

```
    public void listAllProducts() {
        System.out.println("Product: " + product.getProductName());
       System.out.println("Quantity in stock: " + quantityInStock);


    }


    @Override
        public String toString() {
                return "Inventory [inventoryId=" + inventoryId + ", product=" +
product + ", quantityInStock=" + quantityInStock
                                + ", lastStockUpdate=" + lastStockUpdate + "]";
        }
}
```

## Task 3: Encapsulation

All entity classes follow the principle of encapsulation by declaring attributes as private and providing controlled access through public getter and setter methods. This ensures data hiding and integrity.

Additionally, basic validation logic has been implemented in relevant setter methods to prevent invalid data.

## Product.java

```
public void setQuantity(int quantity) {
    if(quantity>0) {
        this.quantity = quantity;
    }
    else {
        System.out.println("Quantity should not be negative");
    }
}

public void updateQuantity(int newQty) {
    setQuantity(newQty);
}
```

## OrderDetail.java

```
public void setQuantity(int quantity) {
    if(quantity>0) {
        this.quantity = quantity;
    }
    else {
        System.out.println("Quantity should not be negative");
    }
}

public void updateQuantity(int newQty) {
    setQuantity(newQty);
}
```

## Inventory.java

```java
public void setQuantityInStock(int quantityInStock) {
    if(quantityInStock>=0) {
        this.quantityInStock = quantityInStock;
    }
    else {
        System.out.println("Quantity should not be negative");
    }
}
```

```java
public void updateStockQuantity(int newQuantity) {
    setQuantityInStock(newQuantity);
    this.lastStockUpdate = LocalDateTime.now();
}
```

## Order.java

```java
public void setTotalAmount(double totalAmount) {
    if(totalAmount>=0) {
        this.totalAmount = totalAmount;
    }
    else {
        System.out.println("Amount cannot be negative");
    }
}
```

# Task 4: Composition

## Order.java

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.

```
public class Order {
    private int orderId;
    private Customer customer;
    private LocalDateTime orderDate;
    private double totalAmount;
    private String orderStatus;
```

## OrderDetail.java

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

```
public class OrderDetail {
    private int orderDetailId;
    private Order order;
    private Product product;
    private int quantity;
```

## Inventory.java

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

```
public class Inventory {
    private int inventoryId;
    private Product product;
    private int quantityInStock;
    private LocalDateTime lastStockUpdate;
```

## Task 5: Exceptions handling

## AuthenticationException.java

```
package hexa.org.exception;


public class AuthenticationException extends Exception{


public AuthenticationException(String message) {
super(message);
}
}
```

## AuthorizationException.java

```
package hexa.org.exception;

public class AuthorizationException extends Exception{

public AuthorizationException(String message) {
super(message);
}
}
```

## ConcurrencyException.java

```
package hexa.org.exception;

public class ConcurrencyException extends Exception{

	public ConcurrencyException(String message) {
		super(message);
	}
}
```

**IncompleteOrderException.java**

```
package hexa.org.exception;

public class IncompleteOrderException extends Exception{

public IncompleteOrderException(String message) {
super(message);
}
}
```

**InsufficientStockException.java**

```
package hexa.org.exception;

public class InsufficientStockException extends Exception{

public InsufficientStockException(String message) {
super(message);
}
}
```

## InvalidDataException.java

```
package hexa.org.exception;


public class InvalidDataException extends Exception{
public InvalidDataException(String message) {
super(message);
}
}
```

## PaymentFailedException.java

```
package hexa.org.exception;


public class PaymentFailedException extends Exception{


public PaymentFailedException(String message) {
super(message);
}
}
```

## Task 6: Collections

The e-commerce system manages products, orders, inventory, and payments using collections and ensures smooth operations like adding, updating, and sorting items. It handles duplicates, product searches, and payment synchronization. Methods are defined in the ServiceProvider interface and implemented in the ServiceProviderImpl class for modularity.

**ServiceProvider.java**

```
package hexa.org.dao;

import java.util.List;

import hexa.org.entity.Inventory;
import hexa.org.entity.Order;
import hexa.org.entity.OrderDetail;
import hexa.org.entity.Payment;
import hexa.org.entity.Product;
import hexa.org.entity.SalesReport;
import hexa.org.exception.IncompleteOrderException;
import hexa.org.exception.InsufficientStockException;
import hexa.org.exception.PaymentFailedException;

public interface ServiceProvider {

    void addProduct(Product product);
```

```
    void removeProduct(int productId);
    Product searchProductByName(String name);
    List<Product> getAllProducts();

    void addOrder(Order order) throws IncompleteOrderException;
    void cancelOrder(int orderId);
    void updateOrderStatus(int orderId, String newStatus);
    void sortOrdersByDateAscending();

    void addInventory(Inventory inventory);
    Inventory getInventory(int productId);
    void removeInventory(int productId);
    void updateInventoryAfterOrder(int productId, int orderedQuantity)
throws InsufficientStockException;

    void findProductByName(String name);

    void recordPayment(Payment payment) throws
PaymentFailedException;
    void updatePaymentStatus(int paymentId, String newStatus);
    void listAllPayments();

    void addOrderDetail(OrderDetail orderDetail) throws
IncompleteOrderException, InsufficientStockException;
```

```
    String getOrderStatus(int orderId);

    List<SalesReport> getSalesByProduct();

}
```

**ServiceProviderImpl.java**

```
package hexa.org.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.TreeMap;

import hexa.org.entity.Inventory;
import hexa.org.entity.Order;
import hexa.org.entity.OrderDetail;
import hexa.org.entity.Payment;
import hexa.org.entity.Product;
import hexa.org.entity.SalesReport;
import hexa.org.exception.*;
```

```
import hexa.org.util.DBConnUtil;

public class ServiceProviderImpl implements ServiceProvider {

    private List<Product> productList = new ArrayList<>();
    private List<Order> orderList = new ArrayList<>();
    private TreeMap<Integer, Inventory> inventoryMap = new
TreeMap<>();
    private List<Payment> paymentList = new ArrayList<>();

    @Override
    public void addProduct(Product product) {
        for (Product p : productList) {
            if (p.getProductId() == product.getProductId() ||
p.getProductName().equalsIgnoreCase(product.getProductName())) {
                System.out.println("Product already exists");
                return;
            }
        }
        productList.add(product);
        System.out.println("Product added successfully");
    }

    @Override
    public void removeProduct(int productId) {
```

```
    for (Product p : productList) {
        if (p.getProductId() == productId) {
            productList.remove(p);
            System.out.println("Product removed");
            return;
        }
    }
    System.out.println("Product not found");
}

@Override
public Product searchProductByName(String name) {
    for (Product p : productList) {
        if (p.getProductName().equalsIgnoreCase(name)) {
            return p;
        }
    }
    return null;
}

@Override
public List<Product> getAllProducts() {
    return productList;
}
```

```
    @Override
    public void addOrder(Order order) throws IncompleteOrderException {
        if (order.getCustomer() == null || order.getOrderDate() == null ||
order.getTotalAmount() < 0) {
            throw new IncompleteOrderException("Incomplete order details.");
        }
        orderList.add(order);
        System.out.println("Order placed successfully");
    }

    @Override
    public void cancelOrder(int orderId) {
        for (Order o : orderList) {
            if (o.getOrderId() == orderId) {
                orderList.remove(o);
                System.out.println("Order Cancelled");
                return;
            }
        }
        System.out.println("Order not found");
    }

    @Override
    public void updateOrderStatus(int orderId, String newStatus) {
        for (Order o : orderList) {
```

```
            if (o.getOrderId() == orderId) {
                o.setOrderStatus(newStatus);
                System.out.println("Order status: " + newStatus);
                return;
            }
        }
        System.out.println("Order not found");
    }

    @Override
    public void sortOrdersByDateAscending() {
        orderList.sort((o1, o2) ->
o1.getOrderDate().compareTo(o2.getOrderDate()));
        System.out.println("Orders sorted by ascending order date:");
        for (Order o : orderList) {
            System.out.println(o);
        }
    }

    @Override
    public void addInventory(Inventory inventory) {
        Product product = inventory.getProduct();
        int productId = product.getProductId();
        inventoryMap.put(productId, inventory);
        System.out.println("Inventory added for " + productId);
```

```
    }

    @Override
    public Inventory getInventory(int productId) {
        return inventoryMap.get(productId);
    }

    @Override
    public void removeInventory(int productId) {
        if (inventoryMap.containsKey(productId)) {
            inventoryMap.remove(productId);
            System.out.println("Inventory removed for Product ID: " +
productId);
        } else {
            System.out.println("No inventory found for Product ID");
        }
    }

    @Override
    public void updateInventoryAfterOrder(int productId, int
orderedQuantity) throws InsufficientStockException {
        Inventory inventory = inventoryMap.get(productId);
        if (inventory != null) {
            int availableStock = inventory.getQuantityInStock();
            if (availableStock >= orderedQuantity) {
```

```
            inventory.setQuantityInStock(availableStock - orderedQuantity);
            System.out.println("Inventory updated for product ID: " +
productId);
        } else {
            throw new InsufficientStockException("Only " + availableStock
+ " items available.");
        }
    } else {
        System.out.println("No inventory found for product ID: " +
productId);
    }
  }

  @Override
  public void findProductByName(String name) {
    boolean flag = false;
    for (Product product : productList) {
      if
(product.getProductName().toLowerCase().contains(name.toLowerCase()))
{
        System.out.println(product);
        flag = true;
      }
    }
    if (!flag) {
```

```
            System.out.println("No product found");
        }
    }

    @Override
    public void recordPayment(Payment payment) throws
PaymentFailedException {
        boolean flag = false;
        for (Order o : orderList) {
            if (o.getOrderId() == payment.getOrderId()) {
                flag = true;
                o.setOrderStatus("Paid");
                break;
            }
        }
        if (flag) {
            paymentList.add(payment);
            System.out.println("Payment recorded");
        } else {
            throw new PaymentFailedException("Cannot record payment.
Order does not exist");
        }
    }

    @Override
```

```
    public void updatePaymentStatus(int paymentId, String newStatus) {
        for (Payment p : paymentList) {
            if (p.getPaymentId() == paymentId) {
                p.setPaymentStatus(newStatus);
                System.out.println("Payment status updated");
                return;
            }
        }
        System.out.println("Payment ID not found.");
    }

    @Override
    public void listAllPayments() {
        if (paymentList.isEmpty()) {
            System.out.println("No payments recorded.");
        } else {
            for (Payment p : paymentList) {
                System.out.println(p);
            }
        }
    }

    @Override
    public void addOrderDetail(OrderDetail orderDetail) throws
IncompleteOrderException, InsufficientStockException {
```

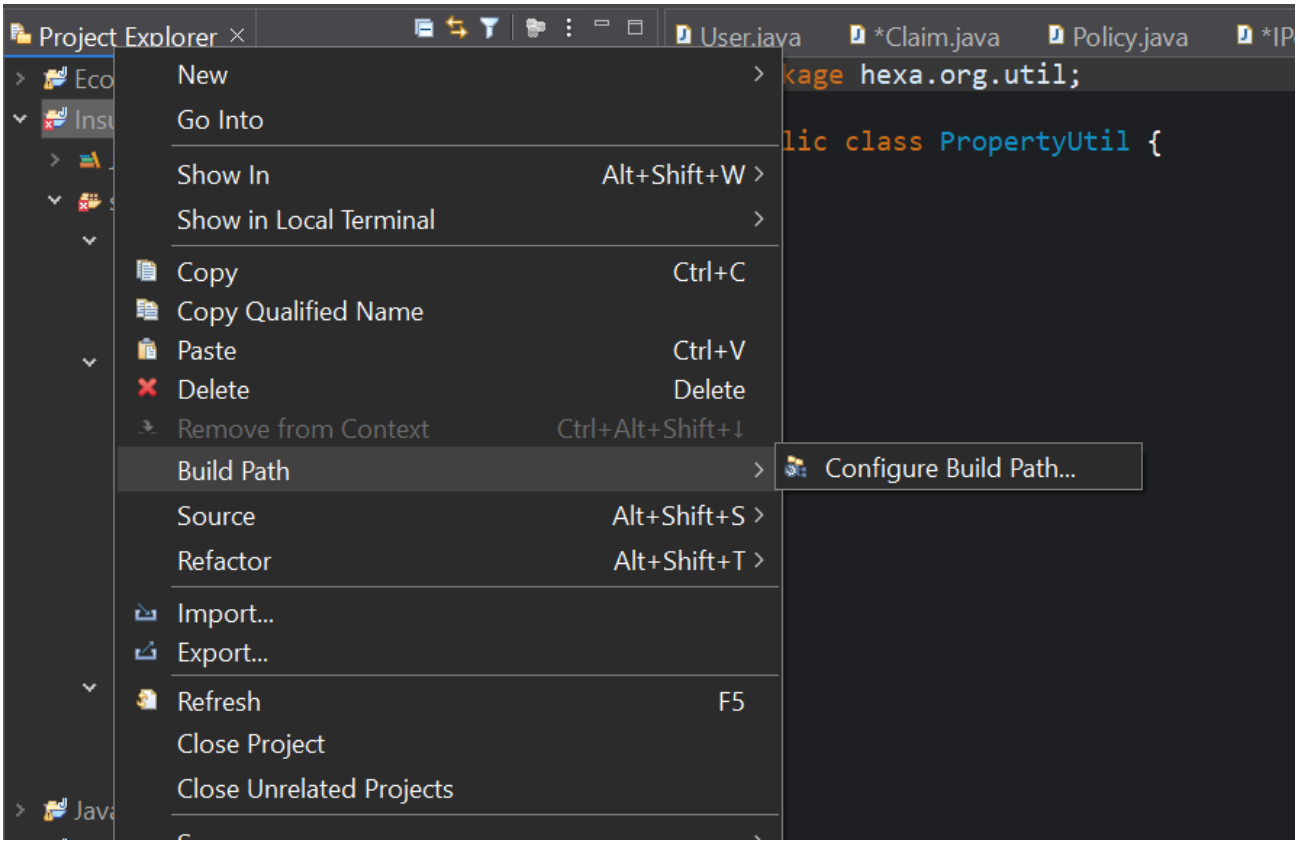
```
        if (orderDetail.getProduct() == null) {
            throw new IncompleteOrderException("Order detail missing
product reference.");
        }

        int productId = orderDetail.getProduct().getProductId();
        int orderedQty = orderDetail.getQuantity();
        Inventory inventory = inventoryMap.get(productId);

        if (inventory != null) {
            if (inventory.getQuantityInStock() >= orderedQty) {
                updateInventoryAfterOrder(productId, orderedQty);
                System.out.println("Order detail added");
            } else {
                throw new InsufficientStockException("Cannot add order detail.
Insufficient stock.");
            }
        } else {
            throw new IncompleteOrderException("Cannot add order detail.
Product not in inventory.");
        }
    }



    @Override
```

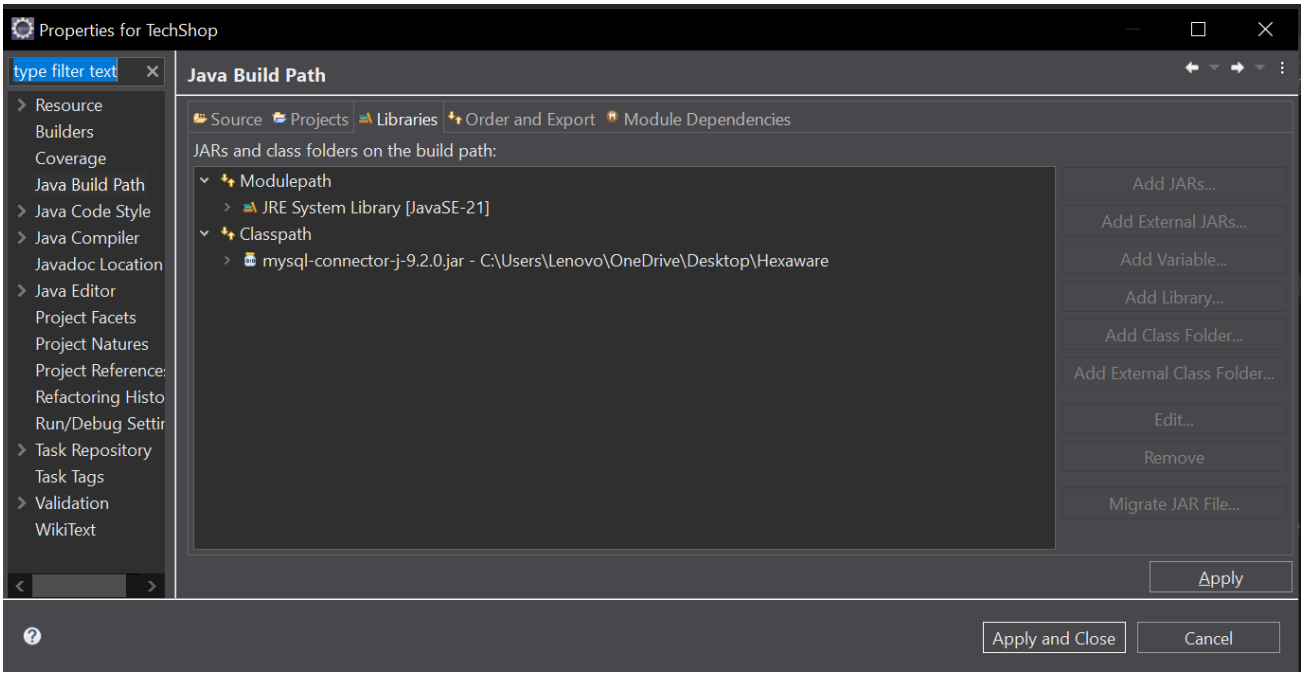
```
public String getOrderStatus(int orderId) {
    String orderStatus = null;

    String sql = "SELECT status FROM orders WHERE orderId = ?";

    try (Connection con = DBConnUtil.getConnection()) {
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setInt(1, orderId);

        ResultSet rs = ps.executeQuery();

        if (rs.next()) {
            orderStatus = rs.getString("status");
        } else {
            System.out.println("Order not found.");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return orderStatus;
}

@Override
public List<SalesReport> getSalesByProduct() {
```

```
        List<SalesReport> salesReports = new ArrayList<>();
        String query = "SELECT p.ProductName, SUM(od.Quantity) AS
totalQuantity, SUM(od.Quantity * p.Price) AS totalSales " +
                "FROM orderdetails od " +
                "JOIN products p ON od.ProductID = p.ProductID " +
                "GROUP BY p.ProductID";
        try (Connection con = DBConnUtil.getConnection();
             Statement stmt = con.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {
            while (rs.next()) {
                String productName = rs.getString("ProductName");
                int totalQuantity = rs.getInt("totalQuantity");
                double totalSales = rs.getDouble("totalSales");

                SalesReport report = new SalesReport(productName,
totalQuantity, totalSales);
                salesReports.add(report);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
        return salesReports;
    }
}
```

# Task 7: Database Connectivity

1. **To enable Java to communicate with the MySQL database via JDBC, the MySQL JDBC driver (mysql-connector-j-9.2.0.jar) must be added to the classpath.**



2. **Connecting the application to the SQL database:**

Writing code to establish a connection to your SQL database.

Create a utility class DBConnection in a package util with a static variable connection of Type Connection and a static method getConnection() which returns connection.Connection properties supplied in the connection string should be read from a property file.

Create a utility class PropertyUtil which contains a static method named getPropertyString() which reads a property file containing connection details like hostname, dbname, username, password, port number and returns a connection string.

**DBConnUtil.java**

```
package hexa.org.util;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnUtil {
private static final String fileName="db.properties";
public static Connection getConnection() {
Connection con=null;
String connString=null;
try {
connString=DBPropertyUtil.getPropertyString(fileName);
}catch(IOException e) {
System.out.println("Connection String Creation Failed...");
e.printStackTrace();
}
if(connString!=null) {
try {
con=DriverManager.getConnection(connString);
}catch(SQLException e) {
```

```
System.out.println("Error While Establishing DBConnection...");
e.printStackTrace();
}
}
return con;
}

}
```

**DBPropertyUtil.java**

```
package hexa.org.util;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class DBPropertyUtil {
public static String getPropertyString(String fileName)throws
IOException{
String connStr=null;
Properties props=new Properties();
FileInputStream fis=new FileInputStream(fileName);
props.load(fis);
String user=props.getProperty("username");
String password=props.getProperty("password");
String protocol=props.getProperty("protocol");
String system=props.getProperty("system");
String database=props.getProperty("dbname");
String port=props.getProperty("port");
connStr=protocol+"//"+system+":"+port+"/"+database+"?user="+user+"&p
assword="+password;
```

```
return connStr;
}

}

}
```

**db.properties**

```
protocol=jdbc:mysql:
system=localhost
port=3306
dbname=techshop
username=root
password=*********
```

**Implement classes for Customers, Products, Orders, OrderDetails, Inventory with properties, constructors, and methods for CRUD (Create, Read, Update, Delete) operations.**

1. Implement a registration form and database connectivity to insert new customer records. Ensure proper data validation and error handling for duplicate email addresses.
7. Implement a user profile management feature with database connectivity to allow customers to update their account details. Ensure data validation and integrity.

**CustomerDAO.java**

```
package hexa.org.dao;

import hexa.org.entity.Customer;

import hexa.org.exception.InvalidDataException;

import java.sql.SQLException;


public interface CustomerDAO {

        void registerCustomer(Customer customer) throws InvalidDataException;

   void updateCustomerDetails(int customerId, String newEmail, String newPhone)
```

```
throws InvalidDataException;
}
```

**CustomerDAOImpl.java**

```
package hexa.org.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import hexa.org.entity.Customer;
import hexa.org.exception.InvalidDataException;
import hexa.org.util.DBConnUtil;

public class CustomerDAOImpl implements CustomerDAO {

	@Override
	public void registerCustomer(Customer customer) throws InvalidDataException
{
		String insertQuery = "INSERT INTO customers (FirstName, LastName,
Email, Phone, Address) VALUES (?, ?, ?, ?, ?)";

		try (Connection con = DBConnUtil.getConnection()) {
			PreparedStatement pstmt = con.prepareStatement(insertQuery,
PreparedStatement.RETURN_GENERATED_KEYS);
			pstmt.setString(1, customer.getFirstName());
			pstmt.setString(2, customer.getLastName());
			pstmt.setString(3, customer.getEmail());
```

```
            pstmt.setString(4, customer.getPhone());
            pstmt.setString(5, customer.getAddress());

            int rowsInserted = pstmt.executeUpdate();
            if (rowsInserted > 0) {
                ResultSet generatedKeys = pstmt.getGeneratedKeys();
                if (generatedKeys.next()) {
                    int generatedCustomerId = generatedKeys.getInt(1);
                    customer.setCustomerId(generatedCustomerId);
                    System.out.println("Customer registered successfully with ID: " +
generatedCustomerId);
                }
            } else {
                throw new InvalidDataException("Customer registration failed. No rows
were inserted.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            throw new InvalidDataException("An error occurred while registering the
customer: " + e.getMessage());
        }
    }

    @Override
    public void updateCustomerDetails(int customerId, String newEmail, String
newPhone) throws InvalidDataException {
        Connection con = DBConnUtil.getConnection();
        if (con != null) {
```

```
        try {
            String checkEmailQuery = "SELECT COUNT(*) FROM customers
WHERE Email = ?";
            PreparedStatement checkEmailStmt =
con.prepareStatement(checkEmailQuery);
            checkEmailStmt.setString(1, newEmail);
            var resultSet = checkEmailStmt.executeQuery();
            if (resultSet.next() && resultSet.getInt(1) > 0) {
                throw new InvalidDataException("Email is already taken by another
customer.");
            }

            String updateQuery = "UPDATE customers SET Email = ?, Phone = ?
WHERE CustomerID = ?";
            PreparedStatement pstmt = con.prepareStatement(updateQuery);
            pstmt.setString(1, newEmail);
            pstmt.setString(2, newPhone);
            pstmt.setInt(3, customerId);

            int rowsUpdated = pstmt.executeUpdate();
            if (rowsUpdated > 0) {
                System.out.println("Customer account updated successfully.");
            } else {
                System.out.println("Customer not found.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (InvalidDataException e) {
```

```
            throw e;
        }
    }
  }
}
```

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

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

**ProductDAO.java**

```
package hexa.org.dao;
import hexa.org.entity.Product;
import hexa.org.exception.InvalidDataException;
import java.util.List;

public interface ProductDAO {
        void updateProduct(Product product) throws InvalidDataException;
        void addProductToDatabase(Product product);
        void removeProductFromDatabase(int productId);

   List<Product> searchProducts(String searchTerm);
   List<Product> getRecommendations(String category);
   Product getProductById(int productId);
}
```

**ProductDAOImpl.java**

```
package hexa.org.dao;

import hexa.org.entity.Product;
import hexa.org.exception.InvalidDataException;
import hexa.org.util.DBConnUtil;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class ProductDAOImpl implements ProductDAO {

   public void addProductToDatabase(Product product) {
       String query = "INSERT INTO products (ProductName, Description, Price,
Category) VALUES (?, ?, ?, ?)";

       try (Connection con = DBConnUtil.getConnection();
            PreparedStatement pstmt = con.prepareStatement(query)) {

           pstmt.setString(1, product.getProductName());
           pstmt.setString(2, product.getDescription());
           pstmt.setDouble(3, product.getPrice());
           pstmt.setString(4, product.getCategory());

           int rowsInserted = pstmt.executeUpdate();

           if (rowsInserted > 0) {
               System.out.println("Product added to the database successfully.");
           } else {
```

```
            System.out.println("Failed to add product to the database.");
        }

    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("Error while adding product: " + e.getMessage());
    }
}

public void removeProductFromDatabase(int productId) {
    String query = "DELETE FROM products WHERE ProductID = ?";

    try (Connection con = DBConnUtil.getConnection();
         PreparedStatement pstmt = con.prepareStatement(query)) {

        pstmt.setInt(1, productId);

        int rowsDeleted = pstmt.executeUpdate();

        if (rowsDeleted > 0) {
            System.out.println("Product removed from the database successfully.");
        } else {
            System.out.println("Product not found in the database.");
        }

    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("Error while removing product: " + e.getMessage());
```

```
    }
  }

      @Override
        public void updateProduct(Product product) throws InvalidDataException {
          Connection con = DBConnUtil.getConnection();
          if (con != null) {
            try {
              String updateQuery = "UPDATE products SET ProductName = ?,
Description = ?, Price = ?, Category = ? WHERE ProductID = ?";
              PreparedStatement pstmt = con.prepareStatement(updateQuery);
              pstmt.setString(1, product.getProductName());
              pstmt.setString(2, product.getDescription());
              pstmt.setDouble(3, product.getPrice());
              pstmt.setString(4, product.getCategory());
              pstmt.setInt(5, product.getProductId());

              int rowsUpdated = pstmt.executeUpdate();
              if (rowsUpdated > 0) {
                System.out.println("Product updated successfully.");
              } else {
                throw new InvalidDataException("Product update failed, product
not found.");
              }

            } catch (SQLException e) {
              System.out.println("Error while updating product.");
              e.printStackTrace();
```

```
                } catch (InvalidDataException e) {
                    throw e;
                }
            }
        }

    @Override
    public List<Product> searchProducts(String searchTerm) {
        List<Product> productList = new ArrayList<>();
        String query = "SELECT * FROM products WHERE ProductName LIKE ? OR
Category LIKE ?";

        try (Connection con = DBConnUtil.getConnection();
             PreparedStatement ps = con.prepareStatement(query)) {

            ps.setString(1, "%" + searchTerm + "%");
            ps.setString(2, "%" + searchTerm + "%");

            try (ResultSet rs = ps.executeQuery()) {
                while (rs.next()) {
                    Product product = new Product(
                        rs.getInt("ProductID"),
                        rs.getString("ProductName"),
                        rs.getString("Description"),
                        rs.getDouble("Price"),
                        rs.getString("Category")
                    );
                    productList.add(product);
```

```
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return productList;
}

@Override
public List<Product> getRecommendations(String category) {
    List<Product> productList = new ArrayList<>();
    String query = "SELECT * FROM products WHERE Category LIKE ?";

    try (Connection con = DBConnUtil.getConnection();
         PreparedStatement ps = con.prepareStatement(query)) {

        ps.setString(1, "%" + category + "%");

        try (ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                Product product = new Product(
                    rs.getInt("ProductID"),
                    rs.getString("ProductName"),
                    rs.getString("Description"),
                    rs.getDouble("Price"),
                    rs.getString("Category")
                );
                productList.add(product);
```

```
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return productList;
}

public Product getProductById(int productId) {
    Product product = null;
    String query = "SELECT * FROM products WHERE ProductID = ?";
    try (Connection con = DBConnUtil.getConnection();
         PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setInt(1, productId);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            product = new Product(
                rs.getInt("ProductID"),
                rs.getString("ProductName"),
                rs.getString("Description"),
                rs.getDouble("Price"),
                rs.getString("Category")
            );
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return product;
```

```
    }
}
```

**3.**Implement an order processing system. Use database connectivity to record customer orders, update product quantities in inventory, and calculate order totals.
4.Develop a feature that allows users to view the status of their orders. Implement database connectivity to retrieve and display order status information.

**OrderDAO.java**

```
package hexa.org.dao;
import hexa.org.entity.Order;
import hexa.org.exception.InsufficientStockException;
public interface OrderDAO {

    void addOrder(Order order) throws InsufficientStockException;
    void cancelOrder(int orderId);
    double calculateOrderTotal(Order order);
}
```

**OrderDAOImpl.java**

```
package hexa.org.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import hexa.org.entity.Order;
import hexa.org.entity.OrderDetail;
import hexa.org.exception.InsufficientStockException;
import hexa.org.util.DBConnUtil;
```

```
public class OrderDAOImpl {

    public void addOrder(Order order) throws InsufficientStockException {
        try (Connection con = DBConnUtil.getConnection()) {
            String insertOrderQuery = "INSERT INTO orders (CustomerID, OrderDate,
TotalAmount, Status) VALUES (?, ?, ?, ?)";
            PreparedStatement pstmt = con.prepareStatement(insertOrderQuery);
            pstmt.setInt(1, order.getCustomer().getCustomerId());
            pstmt.setString(2, order.getOrderDate().toString());
            pstmt.setDouble(3, order.getTotalAmount());
            pstmt.setString(4, order.getOrderStatus());
            int rowsInserted = pstmt.executeUpdate();

            if (rowsInserted > 0) {
                System.out.println("Order placed successfully!");
                updateInventory(order);
            } else {
                System.out.println("Order placement failed.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void updateInventory(Order order) throws InsufficientStockException {
        try (Connection con = DBConnUtil.getConnection()) {
            for (OrderDetail orderDetail : order.getOrderDetails()) {
                int productId = orderDetail.getProduct().getProductId();
```

```
            int orderedQuantity = orderDetail.getQuantity();

            String query = "SELECT QuantityInStock FROM inventory WHERE
ProductID = ?";
            PreparedStatement pstmt = con.prepareStatement(query);
            pstmt.setInt(1, productId);
            var rs = pstmt.executeQuery();

            if (rs.next()) {
               int currentStock = rs.getInt("QuantityInStock");
               if (currentStock >= orderedQuantity) {
                  String updateQuery = "UPDATE inventory SET QuantityInStock = ?
WHERE ProductID = ?";
                  PreparedStatement updateStmt = con.prepareStatement(updateQuery);
                  updateStmt.setInt(1, currentStock - orderedQuantity);
                  updateStmt.setInt(2, productId);
                  updateStmt.executeUpdate();
               } else {
                  throw new InsufficientStockException("Not enough stock for product:
" + productId);
                  }
            } else {
               throw new InsufficientStockException("Product not found in inventory: "
+ productId);
            }
         }
      } catch (SQLException e) {
         e.printStackTrace();
```

```
        }
    }

    public void cancelOrder(int orderId) {
        try (Connection con = DBConnUtil.getConnection()) {
            String deleteOrderQuery = "DELETE FROM orders WHERE OrderID = ?";
            PreparedStatement pstmt = con.prepareStatement(deleteOrderQuery);
            pstmt.setInt(1, orderId);
            int rowsDeleted = pstmt.executeUpdate();

            if (rowsDeleted > 0) {
                System.out.println("Order canceled successfully.");
                restoreInventory(orderId, con);
            } else {
                System.out.println("Order not found.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private void restoreInventory(int orderId, Connection con) {
        try {
            String getOrderDetailsQuery = "SELECT od.ProductID, od.Quantity, p.Price
FROM orderdetails od JOIN products p ON od.ProductID = p.ProductID WHERE
od.OrderID = ?";
            PreparedStatement pstmt = con.prepareStatement(getOrderDetailsQuery);
            pstmt.setInt(1, orderId);
```

```
            var rs = pstmt.executeQuery();

            while (rs.next()) {
                int productId = rs.getInt("ProductID");
                int quantity = rs.getInt("Quantity");
                double price = rs.getDouble("Price");

                String updateInventoryQuery = "UPDATE inventory SET QuantityInStock
= QuantityInStock + ? WHERE ProductID = ?";
                PreparedStatement updateStmt =
con.prepareStatement(updateInventoryQuery);
                updateStmt.setInt(1, quantity);
                updateStmt.setInt(2, productId);
                updateStmt.executeUpdate();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

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

**PaymentDAO.java**

```
package hexa.org.dao;
import java.sql.SQLException;
import hexa.org.entity.Payment;
import hexa.org.exception.InvalidDataException;
import hexa.org.exception.PaymentFailedException;

public interface PaymentDAO {
	void recordPayment(Payment payment) throws InvalidDataException,
PaymentFailedException;
	void updatePaymentStatus(int paymentId, String newStatus) throws
SQLException;
}
```

**PaymentDAOImpl.java**

```
package hexa.org.dao;
import hexa.org.entity.Payment;
import hexa.org.exception.InvalidDataException;
import hexa.org.exception.PaymentFailedException;
import hexa.org.util.DBConnUtil;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class PaymentDAOImpl implements PaymentDAO {

   @Override
```

```
   public void recordPayment(Payment payment) throws InvalidDataException,
PaymentFailedException {
       Connection con = DBConnUtil.getConnection();

       if (con != null) {
          String insertQuery = "INSERT INTO payments (OrderID, Amount,
PaymentStatus) VALUES (?, ?, ?)";

          try (PreparedStatement ps = con.prepareStatement(insertQuery)) {
             ps.setInt(1, payment.getOrderId());
             ps.setDouble(2, payment.getAmount());
             ps.setString(3, payment.getPaymentStatus());

             int rowsAffected = ps.executeUpdate();

             if (rowsAffected > 0) {
                System.out.println("Payment recorded successfully.");
             } else {
                throw new PaymentFailedException("Payment transaction failed, unable
to record.");
             }
          } catch (SQLException e) {
             System.out.println("Error executing payment record insertion: " +
e.getMessage());
             e.printStackTrace();
             throw new PaymentFailedException("Payment process failed due to SQL
error.");
          } finally {
```

```
            try {
                con.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    } else {
        System.out.println("Database connection failed.");
        throw new PaymentFailedException("Payment process failed due to
connection issues.");
    }
  }

  public void updatePaymentStatus(int paymentId, String newStatus) throws
SQLException {
    Connection con = DBConnUtil.getConnection();
    if (con != null) {
        String updateQuery = "UPDATE payments SET PaymentStatus = ? WHERE
PaymentID = ?";
        try (PreparedStatement ps = con.prepareStatement(updateQuery)) {
            ps.setString(1, newStatus);
            ps.setInt(2, paymentId);
            int rowsUpdated = ps.executeUpdate();

            if (rowsUpdated > 0) {
                System.out.println("Payment status updated successfully.");
            } else {
                System.out.println("Payment not found or update failed.");
```

```
            }
        }
    } else {
        System.out.println("Database connection failed.");
    }
  }
}
```

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

**InventoryDAO.java**

```
package hexa.org.dao;
import hexa.org.entity.Inventory;
public interface InventoryDAO {
        void addInventory(Inventory inventory);
   void updateInventory(int productId, int quantity);
   void removeInventory(int productId);
}
```

**InventoryDAOImpl.java**

```
package hexa.org.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import hexa.org.entity.Inventory;
import hexa.org.util.DBConnUtil;

public class InventoryDAOImpl implements InventoryDAO {
```

```
    public void addInventory(Inventory inventory) {
        Connection con = DBConnUtil.getConnection();
        if (con != null) {
            try {
                String query = "INSERT INTO inventory (ProductID, QuantityInStock,
LastStockUpdate) VALUES (?, ?, ?)";
                PreparedStatement stmt = con.prepareStatement(query);
                stmt.setInt(1, inventory.getProduct().getProductId());
                stmt.setInt(2, inventory.getQuantityInStock());
                stmt.setTimestamp(3,
java.sql.Timestamp.valueOf(inventory.getLastStockUpdate()));
                stmt.executeUpdate();
                System.out.println("Inventory added successfully!");
            } catch (SQLException e) {
                System.out.println("Error adding inventory: " + e.getMessage());
                e.printStackTrace();
            }
        }
    }

    public void removeInventory(int productId) {
        Connection con = DBConnUtil.getConnection();
        if (con != null) {
            try {
                String query = "DELETE FROM inventory WHERE ProductID = ?";
                PreparedStatement stmt = con.prepareStatement(query);
                stmt.setInt(1, productId);
                stmt.executeUpdate();
```

```
            } catch (SQLException e) {
                System.out.println("Error removing inventory: " + e.getMessage());
                e.printStackTrace();
            }
        }
    }

    public void updateInventory(int productId, int quantity) {
        Connection con = DBConnUtil.getConnection();
        if (con != null) {
            try {
                String query = "UPDATE inventory SET QuantityInStock = ? WHERE
ProductID = ?";
                PreparedStatement stmt = con.prepareStatement(query);
                stmt.setInt(1, quantity);
                stmt.setInt(2, productId);
                stmt.executeUpdate();
            } catch (SQLException e) {
                System.out.println("Error updating inventory: " + e.getMessage());
                e.printStackTrace();
            }
        }
    }
}
```

**mainModule.java**

```
package hexa.org.app;

import java.sql.SQLException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

import hexa.org.dao.ServiceProviderImpl;
import hexa.org.entity.Customer;
import hexa.org.entity.Inventory;
import hexa.org.entity.Order;
import hexa.org.entity.OrderDetail;
import hexa.org.entity.Payment;
import hexa.org.entity.Product;
import hexa.org.entity.SalesReport;
import hexa.org.exception.InsufficientStockException;
import hexa.org.exception.InvalidDataException;
import hexa.org.exception.PaymentFailedException;
import hexa.org.dao.CustomerDAOImpl;
import hexa.org.dao.InventoryDAOImpl;
import hexa.org.dao.OrderDAOImpl;
import hexa.org.dao.PaymentDAOImpl;
import hexa.org.dao.ProductDAOImpl;
```

```
public class MainModule {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ServiceProviderImpl serviceProvider = new ServiceProviderImpl();
        boolean flag = true;

        while (flag) {
            System.out.println("Welcome to the TECHSHOP Management System!");
            System.out.println("Choose a category:");
            System.out.println("1. Customer Management");
            System.out.println("2. Product Management");
            System.out.println("3. Order Management");
            System.out.println("4. Payment Management");
            System.out.println("5. Inventory Management");
            System.out.println("6. Generate Report");
            System.out.println("7. Exit");

            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    customerManagement(scanner, serviceProvider);
                    break;
                case 2:
                    productManagement(scanner, serviceProvider);
                    break;
                case 3:
```

```
                orderManagement(scanner, serviceProvider);
                break;
            case 4:
                paymentManagement(scanner, serviceProvider);
                break;
            case 5:
                inventoryManagement(scanner, serviceProvider);
                break;
            case 6:
                generateReport(scanner, serviceProvider);
                break;
            case 7:
                flag = false;
                System.out.println("Exiting system...");
                break;
            default:
                System.out.println("Invalid choice! Please try again.");
          }
       }

       scanner.close();
    }

 // Customer Management Methods
    public static void customerManagement(Scanner scanner, ServiceProviderImpl
serviceProvider) {
       System.out.println("Customer Management:");
       System.out.println("1. Add Customer");
```

```
        System.out.println("2. Update Customer");
        System.out.println("3. Go Back");

        CustomerDAOImpl customerDAO = new CustomerDAOImpl();

        int choice = scanner.nextInt();
        scanner.nextLine();

        switch (choice) {
            case 1:
                System.out.println("Enter customer details: ");
                System.out.print("First Name: ");
                String firstName = scanner.nextLine();
                System.out.print("Last Name: ");
                String lastName = scanner.nextLine();
                System.out.print("Email: ");
                String email = scanner.nextLine();
                System.out.print("Phone: ");
                String phone = scanner.nextLine();
                System.out.print("Address: ");
                String address = scanner.nextLine();

                Customer newCustomer = new Customer(firstName, lastName, email,
phone, address);
                try {
                    customerDAO.registerCustomer(newCustomer);
                } catch (InvalidDataException e) {
                    System.out.println(e.getMessage());
```

```
            }
            break;

        case 2:
            System.out.println("Enter customer ID to update: ");
            int customerId = scanner.nextInt();
            scanner.nextLine();
            System.out.println("Enter new email: ");
            String newEmail = scanner.nextLine();
            System.out.println("Enter new phone: ");
            String newPhone = scanner.nextLine();

            try {
                customerDAO.updateCustomerDetails(customerId, newEmail,
newPhone);
            } catch (InvalidDataException e) {
                System.out.println(e.getMessage());
            }
            break;

        case 3:
            break;
        default:
            System.out.println("Invalid choice! Please try again.");
      }
   }

   // Product Management Methods
```

```
    public static void productManagement(Scanner scanner, ServiceProviderImpl
serviceProvider) {
        System.out.println("Product Management:");
        System.out.println("1. Add Product");
        System.out.println("2. Remove Product");
        System.out.println("3. Update Product");
        System.out.println("4. Go Back");

        ProductDAOImpl productDAO = new ProductDAOImpl();

        int choice = scanner.nextInt();
        scanner.nextLine();

        switch (choice) {
        case 1:
            System.out.println("Enter product details: ");
            System.out.print("Product Name: ");
            String productName = scanner.nextLine();
            System.out.print("Description: ");
            String description = scanner.nextLine();
            System.out.print("Price: ");
            double price = scanner.nextDouble();
            scanner.nextLine();
            System.out.print("Category: ");
            String category = scanner.nextLine();

            Product newProduct = new Product(0, productName, description, price,
category);
```

```
            productDAO.addProductToDatabase(newProduct);
            break;

        case 2:
            System.out.println("Enter product ID to remove: ");
            int productIdToRemove = scanner.nextInt();

            productDAO.removeProductFromDatabase(productIdToRemove);
            break;
            case 3:
                System.out.println("Enter product ID to update: ");
                int productIdToUpdate = scanner.nextInt();
                scanner.nextLine();

                System.out.println("Enter new product name: ");
                String updatedProductName = scanner.nextLine();
                System.out.println("Enter new description: ");
                String updatedDescription = scanner.nextLine();
                System.out.println("Enter new price: ");
                double updatedPrice = scanner.nextDouble();
                scanner.nextLine();
                System.out.println("Enter new category: ");
                String updatedCategory = scanner.nextLine();

                Product updatedProduct = new Product(productIdToUpdate,
updatedProductName, updatedDescription, updatedPrice, updatedCategory);
```

```
            try {
                productDAO.updateProduct(updatedProduct);
            } catch (InvalidDataException e) {
                System.out.println("Error updating product: " + e.getMessage());
            }
            break;

        case 4:
            break;

        default:
            System.out.println("Invalid choice! Please try again.");
      }
   }
   // Order Management Methods
   public static void orderManagement(Scanner scanner, ServiceProviderImpl
serviceProvider) {
      System.out.println("Order Management:");
      System.out.println("1. Add Order");
      System.out.println("2. Cancel Order");
      System.out.println("3. Go Back");

      int choice = scanner.nextInt();
      scanner.nextLine();

      OrderDAOImpl orderDAO = new OrderDAOImpl();
      CustomerDAOImpl customerDAO = new CustomerDAOImpl();
      ProductDAOImpl productDAO = new ProductDAOImpl();
```

```
        switch (choice) {
        case 1:
            System.out.print("Enter customer details:\nFirst Name: ");
            String firstName = scanner.nextLine();
            System.out.print("Last Name: ");
            String lastName = scanner.nextLine();
            System.out.print("Email: ");
            String email = scanner.nextLine();
            System.out.print("Phone: ");
            String phone = scanner.nextLine();
            System.out.print("Address: ");
            String address = scanner.nextLine();

            Customer newCustomer = new Customer(firstName, lastName, email, phone,
address);

            try {
                customerDAO.registerCustomer(newCustomer);
                int customerId = newCustomer.getCustomerId();
                System.out.println("Customer registered successfully with ID: " +
customerId);

                System.out.print("Order Date (yyyy-mm-dd): ");
                String orderDateStr = scanner.nextLine();

                List<OrderDetail> orderDetails = new ArrayList<>();
                double totalAmount = 0.0;
```

```
System.out.print("Enter number of items in the order: ");
int numItems = scanner.nextInt();
scanner.nextLine();
for (int i = 0; i < numItems; i++) {
    System.out.println("Enter details for item " + (i + 1));
    System.out.print("Product ID: ");
    int productId = scanner.nextInt();

    Product product = productDAO.getProductById(productId);
    if (product == null) {
        System.out.println("Product not found for ID: " + productId);
        continue;
    }

    System.out.print("Quantity: ");
    int quantity = scanner.nextInt();
    scanner.nextLine();

    OrderDetail orderDetail = new OrderDetail(product, quantity);

    orderDetails.add(orderDetail);
    totalAmount += product.getPrice() * quantity;
}

String orderStatus = "Pending";

Order newOrder = new Order(0, newCustomer,
```

```
LocalDateTime.parse(orderDateStr + "T00:00:00"), totalAmount, orderStatus);
            newOrder.setOrderDetails(orderDetails);
            try {
                orderDAO.addOrder(newOrder);
                System.out.println("Order placed successfully!");
            } catch (InsufficientStockException e) {
                System.out.println("Error: " + e.getMessage());
            }
        } catch (InvalidDataException e) {
            System.out.println("Error: " + e.getMessage());
            return;
        }
        break;



        case 2:
            System.out.print("Enter order ID to cancel: ");
            int orderIdToCancel = scanner.nextInt();
            orderDAO.cancelOrder(orderIdToCancel);
            System.out.println("Order canceled successfully!");
            break;

        case 3:
            break;

        default:
            System.out.println("Invalid choice! Please try again.");
    }
```

```
    }

    // Payment Management Methods
    public static void paymentManagement(Scanner scanner, ServiceProviderImpl
serviceProvider) {
        System.out.println("Payment Management:");
        System.out.println("1. Record Payment");
        System.out.println("2. Update Payment Status");
        System.out.println("3. Go Back");

        int choice = scanner.nextInt();
        scanner.nextLine();

        PaymentDAOImpl paymentDAO = new PaymentDAOImpl();

        switch (choice) {
            case 1:
                System.out.println("Enter payment details:");
                System.out.print("Order ID: ");
                int orderId = scanner.nextInt();
                System.out.print("Amount: ");
                double amount = scanner.nextDouble();
                scanner.nextLine();
                System.out.print("Payment Status (e.g., Completed, Failed): ");
                String paymentStatus = scanner.nextLine();

                Payment payment = new Payment(0, orderId, amount, paymentStatus);
```

```
            try {
                paymentDAO.recordPayment(payment);
            } catch (InvalidDataException | PaymentFailedException e) {
                System.out.println(e.getMessage());
            }
            break;

        case 2:
            System.out.println("Enter payment ID to update: ");
            int paymentId = scanner.nextInt();
            scanner.nextLine();
            System.out.println("Enter new payment status (e.g., Completed, Failed): ");
            String newPaymentStatus = scanner.nextLine();

            try {
                paymentDAO.updatePaymentStatus(paymentId, newPaymentStatus);
            } catch (SQLException e) {
                System.out.println("Error updating payment status: " + e.getMessage());
            }
            break;

        case 3:
            break;

        default:
            System.out.println("Invalid choice! Please try again.");
    }
}
```

```
    // Inventory Management Methods
    public static void inventoryManagement(Scanner scanner, ServiceProviderImpl
serviceProvider) {
        System.out.println("Inventory Management:");
        System.out.println("1. Add Inventory");
        System.out.println("2. Remove Inventory");
        System.out.println("3. Update Inventory");
        System.out.println("4. Go Back");

        int choice = scanner.nextInt();
        scanner.nextLine();

        InventoryDAOImpl inventoryDAO = new InventoryDAOImpl();

        switch (choice) {
        case 1:
          System.out.println("Enter product details:");
          System.out.print("Product ID: ");
          int productId = scanner.nextInt();
          System.out.print("Quantity in Stock: ");
          int quantityInStock = scanner.nextInt();
          scanner.nextLine();

          Product product = new Product(productId, "Default Product Name", "Default
Description", 0.0, "Default Category");
          LocalDateTime currentTime = LocalDateTime.now();
          String lastStockUpdateStr = scanner.nextLine();
```

```
        LocalDateTime lastStockUpdate = (lastStockUpdateStr != null
&& !lastStockUpdateStr.trim().isEmpty())
            ? LocalDateTime.parse(lastStockUpdateStr,
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"))
            : currentTime;

        Inventory inventory = new Inventory(0, product, quantityInStock,
lastStockUpdate);
        inventoryDAO.addInventory(inventory);
        break;

          case 2:
              System.out.println("Enter product ID to remove inventory: ");
              int productIdToRemove = scanner.nextInt();
              inventoryDAO.removeInventory(productIdToRemove); // Remove
inventory for the given product ID
              break;

          case 3:
              System.out.println("Enter product ID to update inventory: ");
              int productIdToUpdate = scanner.nextInt();
              System.out.println("Enter new quantity: ");
              int newQuantity = scanner.nextInt();

              inventoryDAO.updateInventory(productIdToUpdate, newQuantity);
              break;
```

```
            case 4:
                break;

            default:
                System.out.println("Invalid choice! Please try again.");
        }
    }

    // Report Generation Methods
    public static void generateReport(Scanner scanner, ServiceProviderImpl
serviceProvider) {
        System.out.println("Generate Report:");
        System.out.println("1. Sales Report by Product");
        System.out.println("2. Go Back");

        int choice = scanner.nextInt();

        switch (choice) {
            case 1:
                System.out.println("Generating Sales Report...");

                List<SalesReport> salesReports = serviceProvider.getSalesByProduct();

                if (salesReports != null && !salesReports.isEmpty()) {
                    System.out.println("Sales Report by Product:");
                    System.out.println("Product Name | Total Quantity Sold | Total Sales");
                    for (SalesReport report : salesReports) {
                        System.out.println(report.getProductName() + " | " +
```

```
report.getTotalQuantity() + " | " + report.getTotalSales());
                }
            } else {
                System.out.println("No sales data available.");
            }
            break;

        case 2:
                break;

        default:
            System.out.println("Invalid choice! Please try again.");
      }
   }

}
```

# OUTPUT:

## Customer Management:

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
1
Customer Management:
1. Add Customer
2. Update Customer
3. Go Back
1
Enter customer details:
First Name: Silas
Last Name: Evan
Email: sil@email.com
Phone: 8973662788
Address: tetra street
Customer registered successfully with ID: 23
```

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
1
Customer Management:
1. Add Customer
2. Update Customer
3. Go Back
2
Enter customer ID to update:
23
Enter new email:
silas@email.com
Enter new phone:
787463667
Customer account updated successfully.
```

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
1
Customer Management:
1. Add Customer
2. Update Customer
3. Go Back
3
```

## Product Management:

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
2
Product Management:
1. Add Product
2. Remove Product
3. Update Product
4. Go Back
1
Enter product details:
Product Name: Headphones
Description: Bluetooth Headphones Blue
Price: 799.00
Category: Peripherals
Product added to the database successfully.
```

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
2
Product Management:
1. Add Product
2. Remove Product
3. Update Product
4. Go Back
2
Enter product ID to remove:
14
Product removed from the database successfully.
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
2
Product Management:
1. Add Product
2. Remove Product
3. Update Product
4. Go Back
3
Enter product ID to update:
12
Enter new product name:
Blue Headphones
Enter new description:
Bluetooth Headphones
Enter new price:
899.00
Enter new category:
Peripherals
Product updated successfully.
```

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
2
Product Management:
1. Add Product
2. Remove Product
3. Update Product
4. Go Back
4
```

## Order Management:

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
3
Order Management:
1. Add Order
2. Cancel Order
3. Go Back
1
Enter customer details:
First Name: sandy
Last Name: serilda
Email: sandy@email.com
Phone: 678765468
Address: yerst street
Customer registered successfully with ID: 24
Customer registered successfully with ID: 24
Order Date (yyyy-mm-dd): 2025-04-14
Enter number of items in the order: 1
Enter details for item 1
Product ID: 4
Quantity: 1
Order placed successfully!
```

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
3
Order Management:
1. Add Order
2. Cancel Order
3. Go Back
2
Enter order ID to cancel: 18
Order canceled successfully.
Order canceled successfully!
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
3
Order Management:
1. Add Order
2. Cancel Order
3. Go Back
3
```

## PaymentManagement:

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
4
Payment Management:
1. Record Payment
2. Update Payment Status
3. Go Back
1
Enter payment details:
Order ID: 1
Amount: 8999.00
Payment Status (e.g., Completed, Failed): Completed
Payment recorded successfully.
```

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
4
Payment Management:
1. Record Payment
2. Update Payment Status
3. Go Back
2
Enter payment ID to update:
9
Enter new payment status (e.g., Completed, Failed):
pending
Payment status updated successfully.
```

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
4
Payment Management:
1. Record Payment
2. Update Payment Status
3. Go Back
3
```

## Inventory Management:

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
5
Inventory Management:
1. Add Inventory
2. Remove Inventory
3. Update Inventory
4. Go Back
1
Enter product details:
Product ID: 3
Quantity in Stock: 44

Inventory added successfully!
```

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
5
Inventory Management:
1. Add Inventory
2. Remove Inventory
3. Update Inventory
4. Go Back
2
Enter product ID to remove inventory:
3
```

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
5
Inventory Management:
1. Add Inventory
2. Remove Inventory
3. Update Inventory
4. Go Back
3
Enter product ID to update inventory:
4
Enter new quantity:
34
```

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
5
Inventory Management:
1. Add Inventory
2. Remove Inventory
3. Update Inventory
4. Go Back
4
```

## Generate Report:

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
6
Generate Report:
1. Sales Report by Product
2. Go Back
1
Generating Sales Report...
Sales Report by Product:
Product Name | Total Quantity Sold | Total Sales
bello | 3 | 1200.0
Smartphone | 1 | 8799.99
Smartwatch | 2 | 43999.98
Monitor | 1 | 38500.0
Keyboard | 1 | 1099.99
Mouse | 2 | 1099.98
External Hard Drive | 1 | 3299.99
Speaker | 1 | 879.99
Gaming Mouse | 2 | 2999.98
```

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
6
Generate Report:
1. Sales Report by Product
2. Go Back
2
```

## Exit:

```
Welcome to the TECHSHOP Management System!
Choose a category:
1. Customer Management
2. Product Management
3. Order Management
4. Payment Management
5. Inventory Management
6. Generate Report
7. Exit
7
Exiting system...
```

GitHub Link: https://github.com/sherinsandra03/TechShop