Python Assignment-

Electronic Gadget TechShop

Muskan Saxena-PGET

--Dao Package

CustomersDao.py

```
from Entity.Customers import Customers


class CustomersDAO(Customers):
    def __init__(self):
        super().__init__()

    def perform_customers_actions(self):
        while True:
            print("(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT
0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                self.create_customers_table()
            elif ch == 2:
                print(self.add_customers())
            elif ch == 3:
                print(self.update_customers())
            elif ch == 4:
                print(self.delete_customers())
            elif ch == 5:
                self.select_customers()
            elif ch == 0:
                break
            else:
                print("Invalid choice")

    def create_customers_table(self):
        try:
            create_str = '''CREATE TABLE IF NOT EXISTS Customers (
                customerid INT PRIMARY KEY,
                firstname VARCHAR(50),
                lastname VARCHAR(50),
                email VARCHAR(50),
                phone VARCHAR(10),
                address VARCHAR(50))'''
            self.open()
            self.stmt.execute(create_str)
            print('Customers Table Created successfully.')
        except Exception as e:
```

```
            print(f"Error creating customers table: {e}")
        finally:
            self.close()

    def add_customers(self):
        try:
            self.open()
            customer_id = int(input('Enter Customer ID: '))
            firstname = input('Enter First Name: ')
            lastname = input('Enter Last Name: ')
            email = input('Enter Email: ')
            phone = input('Enter Phone Number: ')
            address = input('Enter Address: ')

            data = [(customer_id, firstname, lastname, email, phone,
address)]
            insert_str = '''INSERT INTO Customers(customerid, firstname,
lastname, email, phone, address)
                            VALUES(%s, %s, %s, %s, %s, %s)'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error adding customer: {e}"
        finally:
            self.close()

    def update_customers(self):
        try:
            self.open()
            customer_id = int(input('Input Customer ID to be Updated: '))
            firstname = input('Enter First Name: ')
            lastname = input('Enter Last Name: ')
            email = input('Enter Email: ')
            phone = input('Enter Phone Number: ')
            address = input('Enter Address: ')

            data = [(firstname, lastname, email, phone, address,
customer_id)]
            update_str = '''UPDATE Customers SET firstname=%s, lastname=%s,
email=%s, phone=%s, address=%s
                            WHERE customerid = %s'''
            self.stmt.executemany(update_str, data)
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error updating customer: {e}"
        finally:
            self.close()

    def delete_customers(self):
        try:
            self.open()
            customer_id = int(input('Input Customer ID to be Deleted: '))
            delete_str = f'''DELETE FROM Customers WHERE customerid =
{customer_id}'''
            self.stmt.execute(delete_str)
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error deleting customer: {e}"
```

```
        finally:
            self.close()

    def select_customers(self):
        try:
            self.open()
            query = "SELECT * FROM Customers"
            self.stmt.execute(query)
            records = self.stmt.fetchall()

            # Displaying the records
            print('Records In Customers Table:')
            for record in records:
                print(record)

        except Exception as e:
            print(f"Error selecting customers: {e}")

        finally:
            self.close()
```

OrderDetailsDao.py

```
from datetime import datetime

from Entity.OrderDetails import Orderdetails


class OrderdetailsDAO(Orderdetails):
    def __init__(self):
        super().__init__()

    def perform_orderdetails_actions(self):
        while True:
            print("(Orderdetails) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE
5.SELECT 0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                self.create_orderdetails_table()
            elif ch == 2:
                print(self.add_orderdetails())
            elif ch == 3:
                print(self.update_orderdetails())
            elif ch == 4:
                print(self.delete_orderdetails())
            elif ch == 5:
                self.select_orderdetails()
            elif ch == 0:
                break
            else:
                print("Invalid choice")

    def create_orderdetails_table(self):
        try:
            create_str = '''CREATE TABLE IF NOT EXISTS Orderdetails (
                orderdetailid INT PRIMARY KEY,
```

```
                orderid INT,
                product_id INT,
                quantity INT,
                FOREIGN KEY(orderid) REFERENCES Orders(orderid) ON DELETE
CASCADE ON UPDATE CASCADE,
                FOREIGN KEY(product_id) REFERENCES Products(product_id) ON
DELETE CASCADE ON UPDATE CASCADE)'''

            self.open()
            self.stmt.execute(create_str)
            print('OrderDetails Table Created successfully.')
        except Exception as e:
            print(f"Error creating orderdetails table: {e}")
        finally:
            self.close()

    def add_orderdetails(self):
        try:
            self.open()
            orderdetail_id = int(input('Enter Order Detail ID: '))
            order_id = int(input('Enter Order ID: '))
            product_id = int(input('Enter Product ID: '))
            quantity = int(input('Enter Quantity: '))

            data = [(orderdetail_id, order_id, product_id, quantity)]
            insert_str = '''INSERT INTO Orderdetails(orderdetailid,
orderid, product_id, quantity)
                            VALUES(%s, %s, %s, %s)'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error adding orderdetails: {e}"
        finally:
            self.close()

    def update_orderdetails(self):
        try:
            self.open()
            orderdetail_id = int(input('Input Order Detail ID to be
Updated: '))
            order_id = int(input('Enter Order ID: '))
            product_id = int(input('Enter Product ID: '))
            quantity = int(input('Enter Quantity: '))

            data = [(order_id, product_id, quantity, orderdetail_id)]
            update_str = '''UPDATE Orderdetails SET orderid=%s,
product_id=%s, quantity=%s
                            WHERE orderdetailid = %s'''
            self.stmt.executemany(update_str, data)
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error updating orderdetails: {e}"
        finally:
            self.close()

    def delete_orderdetails(self):
        try:
            self.open()
            orderdetail_id = int(input('Input Order Detail ID to be
```

```
Deleted: '))
            delete_str = f'''DELETE FROM Orderdetails WHERE orderdetailid =
{orderdetail_id}'''
            self.stmt.execute(delete_str)
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error deleting orderdetails: {e}"
        finally:
            self.close()

    def select_orderdetails(self):
        try:
            select_str = '''SELECT * FROM Orderdetails'''
            self.open()
            self.stmt.execute(select_str)
            records = self.stmt.fetchall()
            print('Records In Orderdetails Table:')
            for i in records:
                print(i)
        except Exception as e:
            print(f"Error selecting orderdetails: {e}")
        finally:
            self.close()
```

InventoryDao.py

```
from Entity.Inventory import Inventory


class InventoryDAO(Inventory):
    def __init__(self):
        super().__init__()

    def perform_inventory_actions(self):
        while True:
            print("(Inventory) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT
0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                self.create_inventory_table()
            elif ch == 2:
                print(self.add_inventory())
            elif ch == 3:
                print(self.update_inventory())
            elif ch == 4:
                print(self.delete_inventory())
            elif ch == 5:
                self.select_inventory()
            elif ch == 0:
                break
            else:
                print("Invalid choice")

    def create_inventory_table(self):
        try:
```

```
            create_str = '''CREATE TABLE IF NOT EXISTS Inventory (
                inventory_id INT PRIMARY KEY,
                product_id INT,
                quantity_in_stock INT,
                last_stock_updated DATE,
                FOREIGN KEY(product_id) REFERENCES Products(product_id) ON
DELETE CASCADE ON UPDATE CASCADE)'''

            self.open()
            self.stmt.execute(create_str)
            print('Inventory Table Created successfully.')
        except Exception as e:
            print(f"Error creating inventory table: {e}")
        finally:
            self.close()

    def add_inventory(self):
        try:
            self.open()
            inventory_id = int(input('Enter Inventory ID: '))
            product_id = int(input('Enter Product ID: '))
            quantity_in_stock = int(input('Enter Quantity In Stock: '))
            last_stock_updated = input('Enter Last Stock Updated (YYYY-MM-
DD): ')

            data = [(inventory_id, product_id, quantity_in_stock,
last_stock_updated)]
            insert_str = '''INSERT INTO Inventory(inventory_id, product_id,
quantity_in_stock, last_stock_updated)
                            VALUES(%s, %s, %s, %s)'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error adding inventory: {e}"
        finally:
            self.close()

    def update_inventory(self):
        try:
            self.open()
            inventory_id = int(input('Input Inventory ID to be Updated: '))
            product_id = int(input('Enter Product ID: '))
            quantity_in_stock = int(input('Enter Quantity In Stock: '))
            last_stock_updated = input('Enter Last Stock Updated (YYYY-MM-
DD): ')

            data = [(inventory_id, product_id, quantity_in_stock,
last_stock_updated)]
            update_str = '''UPDATE Inventory SET product_id=%s,
quantity_in_stock=%s, last_stock_updated=%s
                            WHERE inventory_id = %s'''
            self.stmt.executemany(update_str, data)
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error updating inventory: {e}"
        finally:
            self.close()

    def delete_inventory(self):
```

```
        try:
            self.open()
            inventory_id = int(input('Input Inventory ID to be Deleted: '))
            delete_str = f'''DELETE FROM Inventory WHERE inventory_id =
{inventory_id}'''
            self.stmt.execute(delete_str)
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error deleting inventory: {e}"
        finally:
            self.close()

    def select_inventory(self):
        try:
            select_str = '''SELECT * FROM Inventory'''
            self.open()
            self.stmt.execute(select_str)
            records = self.stmt.fetchall()
            print('Records In Inventory Table:')
            for i in records:
                print(i)
        except Exception as e:
            print(f"Error selecting inventory: {e}")
        finally:
            self.close()

    def GetQuantityInStock(self, product_id):
        pass

    def IsProductAvailable(self, product_id, quantitytocheck):
        pass

    def ListLowStockProducts(self, threshold):
        pass
```

OrdersDao.py

```
from datetime import datetime

from Entity.Orders import Orders


class OrdersDAO(Orders):
    def __init__(self):
        super().__init__()

    def perform_orders_actions(self):
        while True:
            print("(Orders) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT
0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                self.create_orders_table()
            elif ch == 2:
                print(self.add_orders())
            elif ch == 3:
```

```
                print(self.update_orders())
            elif ch == 4:
                print(self.delete_orders())
            elif ch == 5:
                self.select_orders()
            elif ch == 0:
                break
            else:
                print("Invalid choice")

    def create_orders_table(self):
        try:
            create_str = '''CREATE TABLE IF NOT EXISTS Orders (
                orderid INT PRIMARY KEY,
                customerid INT,
                orderdate DATE,
                totalamount FLOAT,
                FOREIGN KEY(customerid) REFERENCES Customers(customerid) ON
DELETE CASCADE ON UPDATE CASCADE)'''

            self.open()
            self.stmt.execute(create_str)
            print('Orders Table Created successfully.')
        except Exception as e:
            print(f"Error creating orders table: {e}")
        finally:
            self.close()

    def add_orders(self):
        try:
            self.open()
            order_id = int(input('Enter Order ID: '))
            customer_id = int(input('Enter Customer ID: '))
            order_date = input('Enter Order Date (YYYY-MM-DD): ')
            total_amount = float(input('Enter Total Amount: '))

            data = [(order_id, customer_id, order_date, total_amount)]
            insert_str = '''INSERT INTO Orders(orderid, customerid,
orderdate, totalamount)
                            VALUES(%s, %s, %s, %s)'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error adding orders: {e}"
        finally:
            self.close()

    def update_orders(self):
        try:
            self.open()
            order_id = int(input('Input Order ID to be Updated: '))
            customer_id = int(input('Enter Customer ID: '))
            order_date = input('Enter Order Date (YYYY-MM-DD): ')
            total_amount = float(input('Enter Total Amount: '))

            data = [(customer_id, order_date, total_amount, order_id)]
            update_str = '''UPDATE Orders SET customerid=%s, orderdate=%s,
totalamount=%s
                            WHERE orderid = %s'''
            self.stmt.executemany(update_str, data)
```

```
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error updating orders: {e}"
        finally:
            self.close()

    def delete_orders(self):
        try:
            self.open()
            order_id = int(input('Input Order ID to be Deleted: '))
            delete_str = f'''DELETE FROM Orders WHERE orderid =
{order_id}'''
            self.stmt.execute(delete_str)
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error deleting orders: {e}"
        finally:
            self.close()

    def select_orders(self):
        try:
            select_str = '''SELECT * FROM Orders'''
            self.open()
            self.stmt.execute(select_str)
            records = self.stmt.fetchall()
            print('Records In Orders Table:')
            for i in records:
                print(i)
        except Exception as e:
            print(f"Error selecting orders: {e}")
        finally:
            self.close()

    def CalculateTotalAmount(self, order_id):
        pass

    def CancelOrder(self, order_id):
        pass
```

ProductsDao.py

```
from Entity.Products import Products

from Entity.Products import Products

class ProductsDAO(Products):
    def __init__(self):
        super().__init__()

    def perform_products_actions(self):
        while True:
```

```
            print("(Products) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT
0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                self.create_products_table()
            elif ch == 2:
                print(self.add_products())
            elif ch == 3:
                print(self.update_products())
            elif ch == 4:
                print(self.delete_products())
            elif ch == 5:
                self.select_products()
            elif ch == 0:
                break
            else:
                print("Invalid choice")

    def create_products_table(self):
        try:
            create_str = '''CREATE TABLE IF NOT EXISTS Products (
            product_id INT PRIMARY KEY,
            product_name VARCHAR(50),
            description VARCHAR(100),
            price FLOAT)'''
            self.open()
            self.stmt.execute(create_str)
            self.close()
            print('Products Table Created successfully.')
        except Exception as e:
            print(f"Error creating products table: {e}")

    def add_products(self):
        try:
            self.open()
            product_id = int(input('Enter Product ID: '))
            product_name = input('Enter Product Name: ')
            description = input('Enter Description: ')
            price = float(input('Enter Price: '))
            data = [(product_id, product_name, description, price)]
            insert_str = '''INSERT INTO Products(product_id, product_name,
description, price)
                            VALUES(%s, %s, %s, %s)'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error adding products: {e}"
        finally:
            self.close()

    def update_products(self):
        try:
            self.open()
            product_id = int(input('Input Product ID to be Updated: '))
            product_name = input('Enter Product Name: ')
            description = input('Enter Description: ')
            price = float(input('Enter Price: '))
            data = [(product_name, description, price, product_id)]
            update_str = '''UPDATE Products SET product_name=%s,
description=%s, price=%s
```

```
                                WHERE product_id = %s'''
                self.stmt.executemany(update_str, data)
                self.conn.commit()
                return True
            except Exception as e:
                return f"Error updating products: {e}"
            finally:
                self.close()

    def delete_products(self):
        try:
            self.open()
            product_id = int(input('Input Product ID to be Deleted: '))
            delete_str = f'''DELETE FROM Products WHERE product_id =
{product_id}'''
            self.stmt.execute(delete_str)
            self.conn.commit()
            return True
        except Exception as e:
            return f"Error deleting products: {e}"
        finally:
            self.close()

    def select_products(self):
        try:
            select_str = '''SELECT * FROM Products'''
            self.open()
            self.stmt.execute(select_str)
            records = self.stmt.fetchall()
            print('Records In Products Table:')
            for i in records:
                print(i)
        except Exception as e:
            print(f"Error selecting products: {e}")
        finally:
            self.close()

    def CalculateSubTotal(self, order_id):
        pass
```

TechShopMethods.py

```
from dao.OrdersDAO import OrdersDAO
from dao.ProductsDAO import ProductsDAO
from dao.OrderDetailsDAO import OrderdetailsDAO
from dao.InventoryDAO import InventoryDAO

class TechShopMethods(OrdersDAO, ProductsDAO, OrderdetailsDAO,
InventoryDAO):
    def __init__(self):
        super().__init__()

    # Calculate Total Orders
    def CalculateTotalAmount(self, orderid):
        try:
            self.open()
            query = "SELECT SUM(TotalAmount) FROM Orders WHERE orderid =
```

```
%s"
            self.stmt.execute(query, (orderid,))
            total_amount = self.stmt.fetchone()[0]
            return total_amount if total_amount else 0

        except Exception as e:
            return f"Error: {e}"

        finally:
            self.close()


    def CancelOrder(self, orderid):
        try:
            self.open()
            self.stmt.execute(f"SELECT * FROM Orders WHERE orderid =
{orderid}")
            existing_order = self.stmt.fetchone()

            if not existing_order:
                return "Order not found."

            return "Order canceled successfully."

        except Exception as e:
            return f"Error: {e}"

        finally:
            self.close()

    # Calculate Sub Total
    def CalculateSubtotal(self, orderid, quantity):
        try:
            self.open()
            query = "SELECT Price FROM Products WHERE ProductID = %s"
            self.stmt.execute(query, (orderid,))
            self.price = self.stmt.fetchone()[0] if self.stmt.rowcount > 0
else 0
            subtotal = quantity * self.price
            return subtotal

        except Exception as e:
            print(f"Error calculating subtotal: {e}")
            return 0

        finally:
            self.close()

    # Get Quantity In Stock
    def GetQuantityInStock(self, product_id):
        try:
            self.open()
            self.stmt.execute("SELECT QuantityInStock FROM Inventory WHERE
ProductID = %s", (product_id,))
            quantity_in_stock = self.stmt.fetchone()[0] if
self.stmt.rowcount > 0 else 0
            return quantity_in_stock

        except Exception as e:
            print(f"Error getting quantity in stock: {e}")
            return 0
```

```
        finally:
            self.close()

    # Is Product Available
    def IsProductAvailable(self, product_id, quantitytocheck):
        try:
            self.open()
            query = "SELECT QuantityInStock FROM Inventory WHERE ProductID
= %s"
            self.stmt.execute(query, (product_id,))
            quantity_in_stock = self.stmt.fetchone()[0] if
self.stmt.rowcount > 0 else 0
            return quantity_in_stock >= quantitytocheck

        except Exception as e:
            print(f"Error checking product availability: {e}")
            return False

        finally:
            self.close()

    # List Low Stock products
    def ListLowStockProducts(self, threshold):
        try:
            self.open()
            query = "SELECT Product_ID, ProductName, Quantity_In_Stock FROM
Products INNER JOIN Inventory ON Products.Product_ID = Inventory.Product_ID
WHERE Quantity_In_Stock < %s"
            self.stmt.execute(query, (threshold,))
            low_stock_products = self.stmt.fetchall()
            return low_stock_products

        except Exception as e:
            print(f"Error listing low stock products: {e}")
            return []

        finally:
            self.close()

    # List Out Of Stock Products
    def ListOutOfStockProducts(self):
        try:
            self.open()
            query = "SELECT Product_ID, ProductName FROM Products WHERE
Product_ID NOT IN (SELECT Product_ID FROM Inventory WHERE Quantity_In_Stock
> 0)"
            self.stmt.execute(query)
            out_of_stock_products = self.stmt.fetchall()
            return out_of_stock_products

        except Exception as e:
            print(f"Error listing out-of-stock products: {e}")
            return []

        finally:
            self.close()
```

--Entity Package

Customers.py

```
from util.DBConnUtil import DBConnection


class Customers(DBConnection):
    def __init__(self):
        super().__init__()
        self._customerid = 0
        self._firstname = ''
        self._lastname = ''
        self._email = ''
        self._phone = ''
        self._address = ''

    @property
    def customerid(self):
        return self._customerid

    @customerid.setter
    def customerid(self, value):
        self._customerid = value

    @property
    def firstname(self):
        return self._firstname

    @firstname.setter
    def firstname(self, value):
        self._firstname = value

    @property
    def lastname(self):
        return self._lastname

    @lastname.setter
    def lastname(self, value):
        self._lastname = value

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

    @email.setter
    def email(self, value):
        self._email = value

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

    @phone.setter
    def phone(self, value):
        self._phone = value
```

```
    @property
    def address(self):
        return self._address

    @address.setter
    def address(self, value):
        self._address = value

    def __str__(self):
        return f'Customer ID: {self._customerid} First Name:
{self._firstname} Last Name: {self._lastname}\n' \
               f'Email: {self._email} Phone: {self._phone} Address:
{self._address}'
```

Orders.py

```
from Entity.Customers import Customers

class Orders(Customers):
    def __init__(self):
        super().__init__()
        self._orderid = 0
        self._customerid = 0  # Composition relationship with Customer
class
        self._orderdate = ''
        self._totalamount = 0.0

    @property
    def orderid(self):
        return self._orderid

    @orderid.setter
    def orderid(self, value):
        self._orderid = value

    @property
    def customerid(self):
        return self._customerid

    @customerid.setter
    def customerid(self, value):
        self._customerid = value

    @property
    def orderdate(self):
        return self._orderdate

    @orderdate.setter
    def orderdate(self, value):
        self._orderdate = value

    @property
    def totalamount(self):
        return self._totalamount

    @totalamount.setter
    def totalamount(self, value):
        self._totalamount = value
```

```
    def __str__(self):
        return f'Order ID: {self._orderid} Customer ID:
{self._customerid}\n' \
               f'Order Date: {self._orderdate} Total Amount:
{self._totalamount}'
```

Products.py

```
from util.DBConnUtil import DBConnection

class Products(DBConnection):
    def __init__(self):
        super().__init__()
        self.product_id = 0
        self.product_name = ''
        self.description = ''
        self.price = 0.0

    def add_product(self):
        try:
            self.open()
            data = [(self.product_id, self.product_name, self.description,
self.price)]
            insert_str = '''INSERT INTO Products(product_id, product_name,
description, price)
                            VALUES(%s, %s, %s, %s)'''
            self.stmt.executemany(insert_str, data)
            self.conn.commit()
            return True
        except Exception as e:
            print(f"Error adding product: {e}")
            return False
        finally:
            self.close()



    def __str__(self):
        return f'Product ID: {self.product_id} Product Name:
{self.product_name}\n' \
               f'Description: {self.description} Price: {self.price}'
```

Orderdetails.py

```
from Entity.Orders import Orders
from Entity.Products import Products

class Orderdetails(Orders, Products):
    def __init__(self):
        super().__init__()
        self._orderdetailid = 0
        self._orderid = 0  # Composition relationship with Orders class
        self._productid = 0  # Composition relationship with Products class
        self._quantity = 0

    @property
    def orderdetailid(self):
        return self._orderdetailid

    @orderdetailid.setter
    def orderdetailid(self, value):
        self._orderdetailid = value

    @property
    def orderid(self):
        return self._orderid

    @orderid.setter
    def orderid(self, value):
        self._orderid = value

    @property
    def productid(self):
        return self._productid

    @productid.setter
    def productid(self, value):
        self._productid = value

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

    @quantity.setter
    def quantity(self, value):
        self._quantity = value

    def __str__(self):
        return f'Order Detail ID: {self._orderdetailid} Order ID:
{self._orderid}\n' \
               f'Product ID: {self._productid} Quantity: {self._quantity}'
```

Inventory.py

```
from Entity.Products import Products

class Inventory(Products):
    def __init__(self):
        super().__init__()
        self._inventoryid = 0
        self._productid = 0  # Composition relationship with Products Class
        self._quantityinstock = 0
        self._laststockupdated = ''

    @property
    def inventory_id(self):
        return self._inventoryid

    @inventory_id.setter
    def inventory_id(self, value):
        self._inventoryid = value

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

    @product_id.setter
    def product_id(self, value):
        self._productid = value

    @property
    def quantity_in_stock(self):
        return self._quantityinstock

    @quantity_in_stock.setter
    def quantity_in_stock(self, value):
        self._quantityinstock = value

    @property
    def last_stock_updated(self):
        return self._laststockupdated

    @last_stock_updated.setter
    def last_stock_updated(self, value):
        self._laststockupdated = value

    def __str__(self):
        return f'Inventory ID: {self._inventoryid} Product ID: {self._productid}\n' \
```

```
                f'Quantity In Stock: {self._quantityinstock} Last Stock Updated:
        {self._laststockupdated}'
```

--Exceptions Package

AuthenticationException.py

```
class AuthenticationException(Exception):
    def __init__(self, message="Authentication failed."):
        self.message = message
        super().__init__(self.message)
class AuthorizationException(Exception):
    def __init__(self, message="Authorization error."):
        self.message = message
        super().__init__(self.message)
```

ConcurrencyException.py

```
class ConcurrencyException(Exception):
    def __init__(self, message="Concurrency error."):
        self.message = message
        super().__init__(self.message)
```

DatabaseAccessException.py

```
class DatabaseAccessException(Exception):
    def __init__(self, message="Database access error."):
        self.message = message
        super().__init__(self.message)
```

FileIOException.py

```
class FileIOException(Exception):
    def __init__(self, message="File I/O error."):
        self.message = message
        super().__init__(self.message)
```

IncompleteOrderException.py

```
class IncompleteOrderException(Exception):
    def __init__(self, message="Incomplete order."):
        self.message = message
        super().__init__(self.message)
```

InsufficientStockException.py

```
class InsufficientStockException(Exception):
    def __init__(self, message="Insufficient stock."):
        self.message = message
        super().__init__(self.message)
```

InvalidDataException.py

```
class InvalidDataException(Exception):
    def __init__(self, message="Invalid Data."):
        self.message = message
        super().__init__(self.message)
```

PaymentFailedException.py

```
class PaymentFailedException(Exception):
    def __init__(self, message="Payment failed."):
        self.message = message
        super().__init__(self.message)
```

--Main Package

MainModule.py

```
from dao.TechShopMethods import TechShopMethods
from dao.CustomersDAO import CustomersDAO
from dao.OrdersDAO import OrdersDAO
from dao.ProductsDAO import ProductsDAO
from dao.OrderDetailsDAO import OrderdetailsDAO
from dao.InventoryDAO import InventoryDAO
from util.DBConnUtil import DBConnection
from exception.PaymentFailedException import PaymentFailedException
from exception.InvalidDataException import InvalidDataException
from exception.InsufficientStockException import InsufficientStockException
from exception.IncompleteOrderException import IncompleteOrderException
from exception.FileIOException import FileIOException
from exception.DatabaseAccessException import DatabaseAccessException
from exception.ConcurrencyException import ConcurrencyException
from exception.AuthorizationException import AuthorizationException
from exception.AuthenticationException import AuthenticationException


def main():
    dbconnection = DBConnection()

    try:
        dbconnection.open()
        print("--Database Is Connected:--")
    except Exception as e:
```

```
        print(e)

    try:
        print("=" * 30)
        print("Electronic Gadget TechShop")
        print("=" * 30)
        print("Welcome to TechShop!")

        electronic_gadget_techshop = TechShopMethods()

        while True:
            print("1.Customers 2.Orders 3.Products 4.OrderDetails
5.Inventory 0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:

                u = CustomersDAO()
                u.perform_customers_actions()
            elif ch == 2:

                c = OrdersDAO()
                c.perform_orders_actions()
            elif ch == 3:

                e = ProductsDAO()
                e.perform_products_actions()
            elif ch == 4:

                l = OrderdetailsDAO()
                l.perform_orderdetails_actions()
            elif ch == 5:

                cc = InventoryDAO()
                cc.perform_inventory_actions()
            elif ch == 0:
                break
            else:
                print("Invalid choice")

        while True:
            print("=" * 10)
            print("---MENU---")
            print("=" * 10)

print("1.CalculateTotalAmount\n2.CancelOrder\n3.CalculateSubTotal\n4.IsProd
uctAvailable\n5.ListLowStockProducts\n0.EXIT")
            ch = int(input("Enter choice: "))
            if ch == 1:
                orderid = int(input('Enter Order ID of the Order to get
TotalAmount: '))

print(electronic_gadget_techshop.CalculateTotalAmount(orderid))
            elif ch == 2:
                orderid = int(input('Enter Order ID of the Order to be
Cancelled: '))
                print(electronic_gadget_techshop.CancelOrder(orderid))
            elif ch == 3:
                product_id = int(input('Enter Product ID to check if the
product is in stock: '))
                quantity = int(input('Enter Quantity: '))
```

```
print(electronic_gadget_techshop.CalculateSubtotal(product_id, quantity))
            elif ch == 4:
                product_id = int(input('Enter Product ID to check if the
product is available: '))
                quantity_to_check = int(input('Enter Quantity to check if
the product is available: '))

print(electronic_gadget_techshop.IsProductAvailable(product_id,
quantity_to_check))
            elif ch == 5:
                threshold_quantity = int(input('Enter Threshold Quantity:
'))

print(electronic_gadget_techshop.ListLowStockProducts(threshold_quantity))
            elif ch == 0:
                break
            else:
                print("Invalid choice")
    except AuthenticationException as e:
        print(e)

    except AuthorizationException as e:
        print(e)

    except ConcurrencyException as e:
        print(e)

    except DatabaseAccessException as e:
        print(e)

    except FileIOException as e:
        print(e)

    except IncompleteOrderException as e:
        print(e)

    except InsufficientStockException as e:
        print(e)

    except InvalidDataException as e:
        print(e)

    except PaymentFailedException as e:
        print(e)

    except Exception as e:
        print(e)
    finally:
        dbconnection.close()
        print("Thank you for visiting TechShop!")
        print("--Connection Is Closed:--")


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

--Util Package

DBConnUtil.py

```
import sys
import mysql.connector as sql
from util.DBPropertyUtil import PropertyUtil


class DBConnection:
    def open(self):
        try:

            connection_properties=PropertyUtil.getConnectionString()
            self.conn=sql.connect(**connection_properties)
            self.stmt=self.conn.cursor()
        except Exception as e:
            print(str(e) + '--Database Is Not Connected:--')
            sys.exit(1)

    def close(self):
        self.conn.close()
```

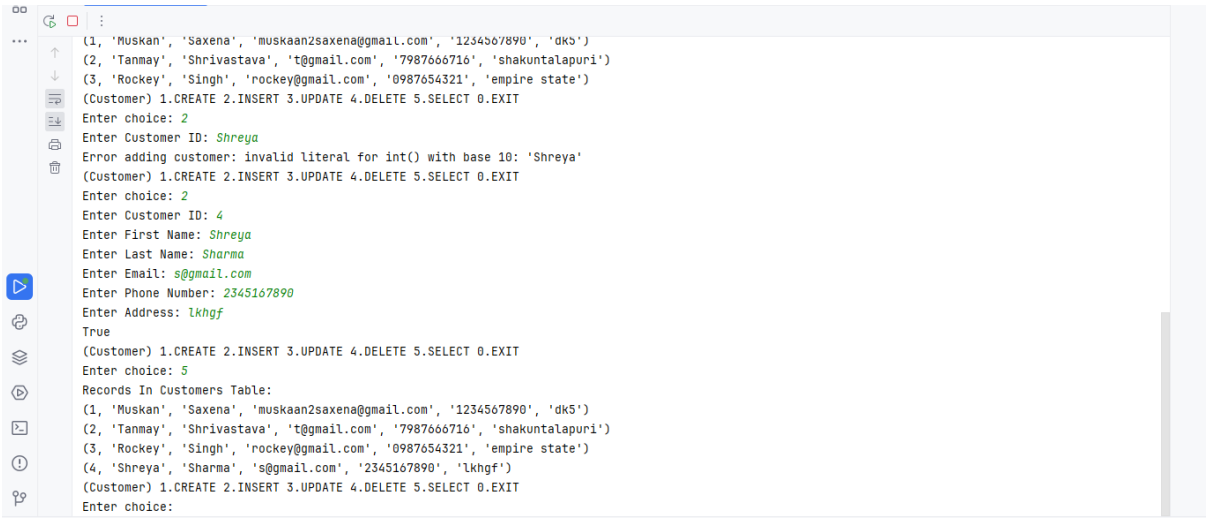
DBConnUtil.py

```
class PropertyUtil:
    connection_properties= None

    @staticmethod
    def getConnectionString():
        if PropertyUtil.connection_properties is None:
            host='localhost'
            database='techshop_db'
            user='root'
            password='Muskan20'

PropertyUtil.connection_properties={'host':host,'database':database,'user':
user,'password':password}
        return  PropertyUtil.connection_properties
```

--Relevant Output Snippets

```
"C:\Users\Muskan Saxena\PycharmProjects\TechShop\.venv\Scripts\python.exe" "C:\Users\Muskan Saxena\PycharmProjects\TechShop\main\MainModule.py"
--Database Is Connected:--
==============================
Electronic Gadget TechShop
==============================
Welcome to TechShop!
1.Customers 2.Orders 3.Products 4.OrderDetails 5.Inventory 0.EXIT
Enter choice: 1
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: 1
Customers Table Created successfully.
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: 2
Enter Customer ID: 1
Enter First Name: Muskan
Enter Last Name: Saxena
Enter Email: muskaan2saxena@gmail.com
Enter Phone Number: 7987666716
Enter Address: DK 5
True
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: 3
Input Customer ID to be Updated: 1
Enter First Name: Tanmay
Enter Last Name: Shrivastava
```

TechShop > dao > CustomersDAO.py 107:1 CRLF UTF-8 4 spaces Python 3.12 (TechShop)



```
(1, 'Muskan', 'Saxena', 'muskaan2saxena@gmail.com', '1234567890', 'dk5')
(2, 'Tanmay', 'Shrivastava', 't@gmail.com', '7987666716', 'shakuntalapuri')
(3, 'Rockey', 'Singh', 'rockey@gmail.com', '0987654321', 'empire state')
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: 2
Enter Customer ID: Shreya
Error adding customer: invalid literal for int() with base 10: 'Shreya'
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: 2
Enter Customer ID: 4
Enter First Name: Shreya
Enter Last Name: Sharma
Enter Email: s@gmail.com
Enter Phone Number: 2345167890
Enter Address: lkhgf
True
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: 5
Records In Customers Table:
(1, 'Muskan', 'Saxena', 'muskaan2saxena@gmail.com', '1234567890', 'dk5')
(2, 'Tanmay', 'Shrivastava', 't@gmail.com', '7987666716', 'shakuntalapuri')
(3, 'Rockey', 'Singh', 'rockey@gmail.com', '0987654321', 'empire state')
(4, 'Shreya', 'Sharma', 's@gmail.com', '2345167890', 'lkhgf')
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice:
```

```
Enter choice: 2
Enter Order ID: 1
Enter Customer ID: 1
Enter Order Date (YYYY-MM-DD): 2024-02-10
Enter Total Amount: 5000
True
(Orders) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: 2
Enter Order ID: 2
Enter Customer ID: 2
Enter Order Date (YYYY-MM-DD): 2024-03-01
Enter Total Amount: 1000
True
```

TechShop > Entity > Products.py 31:1 CRLF UTF-8 4 spaces Python 3.12 (TechShop)

```
Electronic Gadget TechShop
==============================
Welcome to TechShop!
1.Customers 2.Orders 3.Products 4.OrderDetails 5.Inventory 0.EXIT
Enter choice: 1
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: 2
Enter Customer ID: 1
Enter First Name: Tanmay
Enter Last Name: Shrivastava
Enter Email: t@gmail.com
Enter Phone Number: 6263582103
Enter Address: dk5
Error adding customer: 1062 (23000): Duplicate entry '1' for key 'customers.PRIMARY'
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice:
```

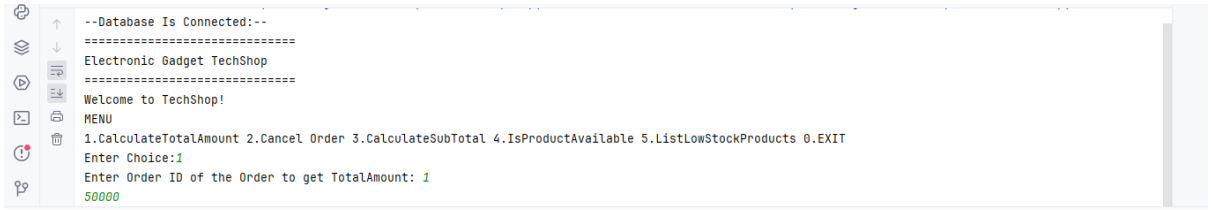
TechShop > dao > TechShopMethods.py 90:1 CRLF UTF-8 4 spaces Python 3.12 (TechShop)

```
--Database Is Connected:--
==============================
Electronic Gadget TechShop
==============================
Welcome to TechShop!
MENU
1.CalculateTotalAmount 2.Cancel Order 3.CalculateSubTotal 4.IsProductAvailable 5.ListLowStockProducts 0.EXIT
Enter Choice:2
Enter Order ID of the Order to be Cancelled: 2
Order canceled successfully.
```

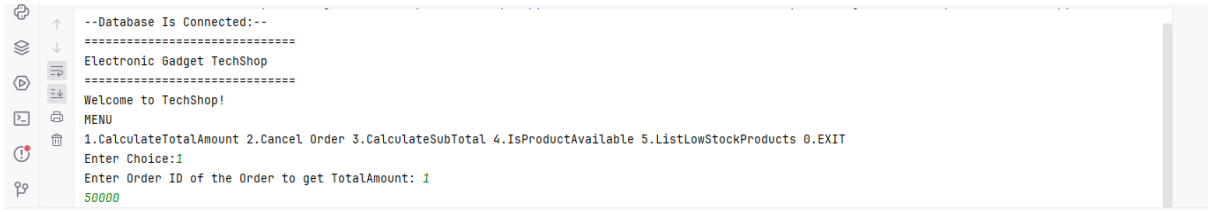
```
Electronic Gadget TechShop
==============================
Welcome to TechShop!
1.Customers 2.Orders 3.Products 4.OrderDetails 5.Inventory 0.EXIT
Enter choice: 1
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: 2
Enter Customer ID: 1
Enter First Name: Tanmay
Enter Last Name: Shrivastava
Enter Email: t@gmail.com
Enter Phone Number: 6263582103
Enter Address: dk5
Error adding customer: 1062 (23000): Duplicate entry '1' for key 'customers.PRIMARY'
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice:
```

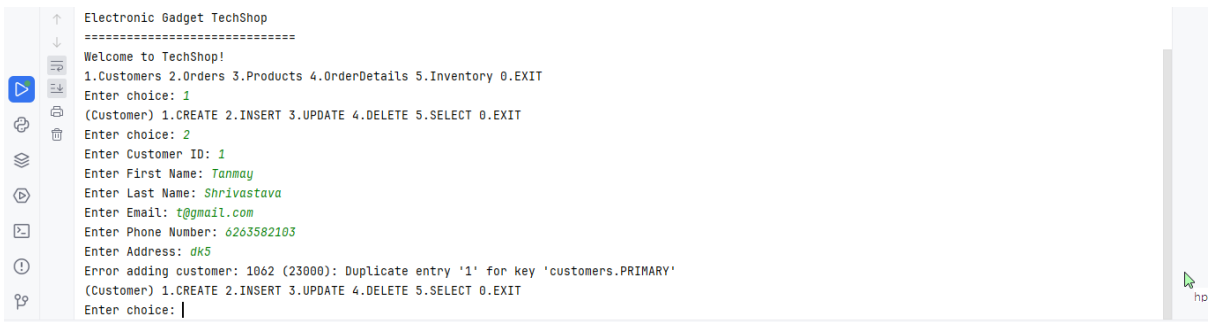
```
==============================
Electronic Gadget TechShop
==============================
Welcome to TechShop!
MENU
1.CalculateTotalAmount 2.Cancel Order 3.CalculateSubTotal 4.IsProductAvailable 5.ListLowStockProducts 0.EXIT
Enter Choice:4
Enter Product ID to check if the product is available: 1
Enter Quantity to check if the product is available: 20
Yes
```


Electronic Gadget TechShop
==========
Welcome to TechShop!
1.Customers 2.Orders 3.Products 4.OrderDetails 5.Inventory 0.EXIT
Enter choice: 1
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: 2
Enter Customer ID: 1
Enter First Name: Tanmay
Enter Last Name: Shrivastava
Enter Email: t@gmail.com
Enter Phone Number: 6263582103
Enter Address: dk5
Error adding customer: 1062 (23000): Duplicate entry '1' for key 'customers.PRIMARY'
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice:
TechShop > dao > TechShopMethods.py
90:1 CRLF UTF-8 4 spaces Python 3.12 (TechShop)


--Database Is Connected:--
==========
Electronic Gadget TechShop
==========
Welcome to TechShop!
MENU
1.CalculateTotalAmount 2.Cancel Order 3.CalculateSubTotal 4.IsProductAvailable 5.ListLowStockProducts 0.EXIT
Enter Choice:1
Enter Order ID of the Order to get TotalAmount: 1
50000


Products Table Created successfully.
(Products) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: 2
Enter Product ID: 9
Enter Product Name: phone
Enter Description: for calling
Enter Price: 20000
True
(Products) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: 3
Input Product ID to be Updated: 9
Enter Product Name: mobile phone
Enter Description: for calling
Enter Price: 25000
True
(Products) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT


--Database Is Connected:--
==========
Electronic Gadget TechShop
==========
Welcome to TechShop!
MENU
1.CalculateTotalAmount 2.Cancel Order 3.CalculateSubTotal 4.IsProductAvailable 5.ListLowStockProducts 0.EXIT
Enter Choice:1
Enter Order ID of the Order to get TotalAmount: 1
50000


Electronic Gadget TechShop
==========
Welcome to TechShop!
1.Customers 2.Orders 3.Products 4.OrderDetails 5.Inventory 0.EXIT
Enter choice: 1
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: 2
Enter Customer ID: 1
Enter First Name: Tanmay
Enter Last Name: Shrivastava
Enter Email: t@gmail.com
Enter Phone Number: 6263582103
Enter Address: dk5
Error adding customer: 1062 (23000): Duplicate entry '1' for key 'customers.PRIMARY'
(Customer) 1.CREATE 2.INSERT 3.UPDATE 4.DELETE 5.SELECT 0.EXIT
Enter choice: