CODING CHALLENGE –PYTHON

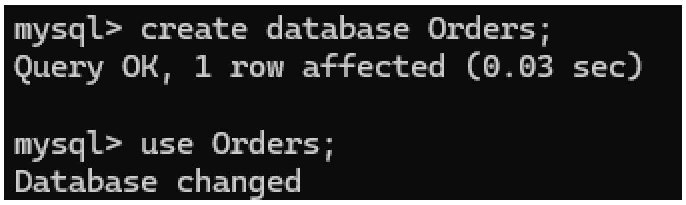
RUTH ROSHINI S

(02/05/2024)

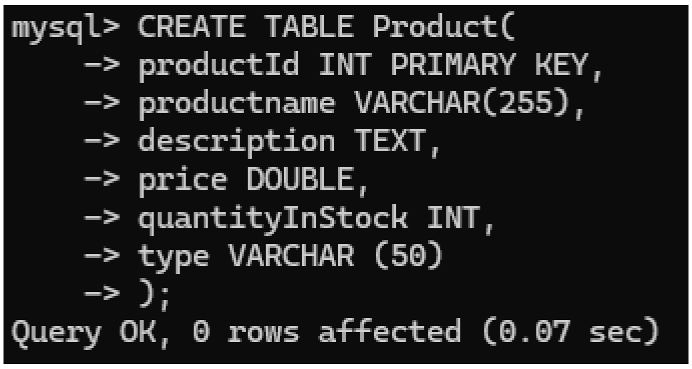
Problem Statement:

Create SQL Schema from the product and user class, use the class attributes for table column names.

```
CREATE TABLE Product(
    -> productId INT PRIMARY KEY,
    -> productname VARCHAR(255),
    -> description TEXT,
    -> price DOUBLE,
    -> quantityInStock INT,
    -> type VARCHAR (50)
    -> );
```



```
mysql> create database Orders;
Query OK, 1 row affected (0.03 sec)

mysql> use Orders;
Database changed
```



```
mysql> CREATE TABLE Product(
    -> productId INT PRIMARY KEY,
    -> productname VARCHAR(255),
    -> description TEXT,
    -> price DOUBLE,
    -> quantityInStock INT,
    -> type VARCHAR (50)
    -> );
Query OK, 0 rows affected (0.07 sec)
```

1. Create a base class called Product with the following attributes:

• productId (int)

• productName (String)

• description (String)

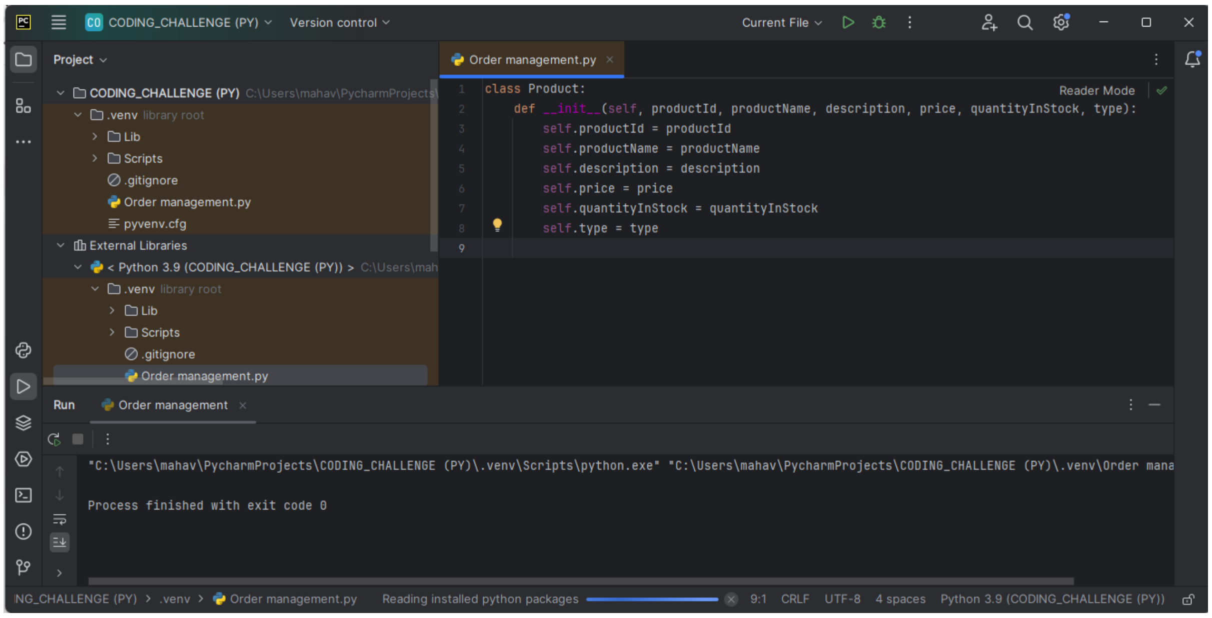
• price (double)

• quantityInStock (int)

• type (String) [Electronics/Clothing]

Code:

```
class Product:
    def __init__(self, productId, productName, description, price,
quantityInStock, type):
        self.productId = productId
        self.productName = productName
        self.description = description
        self.price = price
        self.quantityInStock = quantityInStock
        self.type = type
```



```
class Product:
    def __init__(self, productId, productName, description, price, quantityInStock, type):
        self.productId = productId
        self.productName = productName
        self.description = description
        self.price = price
        self.quantityInStock = quantityInStock
        self.type = type
```

```
"C:\Users\mahav\PycharmProjects\CODING_CHALLENGE (PY)\.venv\Scripts\python.exe" "C:\Users\mahav\PycharmProjects\CODING_CHALLENGE (PY)\.venv\Order mana

Process finished with exit code 0
```

2. Implement constructors, getters, and setters for the Product class.

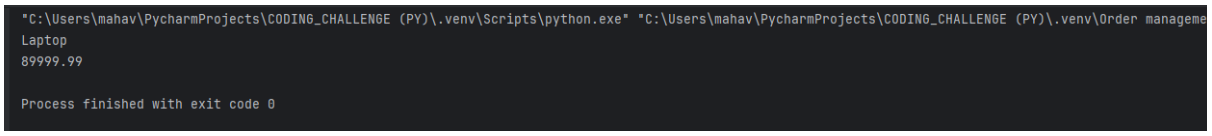
Code:

```
class Product:
    def __init__(self, productId, productName, description, price,
quantityInStock, type):
        self.productId = productId
        self.productName = productName
        self.description = description
        self.price = price
        self.quantityInStock = quantityInStock
        self.type = type

    # Getters
    def getProductId(self):
        return self.productId

    def getProductName(self):
        return self.productName

    def getDescription(self):
        return self.description

    def getPrice(self):
        return self.price

    def getQuantityInStock(self):
        return self.quantityInStock

    def getType(self):
        return self.type

    # Setters
    def setProductId(self, productId):
        self.productId = productId

    def setProductName(self, productName):
        self.productName = productName

    def setDescription(self, description):
        self.description = description

    def setPrice(self, price):
        self.price = price

    def setQuantityInStock(self, quantityInStock):
        self.quantityInStock = quantityInStock

    def setType(self, type):
        self.type = type
```

Implementing:

```
product= Product (1, "Laptop",  "High-performance laptop", 999.99, 10,
"Electronics")

print(product.getProductName())

product.setPrice (89999.99)

print(product.getPrice())
```

Output:



```
"C:\Users\mahav\PycharmProjects\CODING_CHALLENGE (PY)\.venv\Scripts\python.exe" "C:\Users\mahav\PycharmProjects\CODING_CHALLENGE (PY)\.venv\Order manageme
Laptop
89999.99

Process finished with exit code 0
```

3. Create a subclass Electronics that inherits from Product. Add attributes specific to electronics products, such as:

• brand (String)

• warrantyPeriod (int)

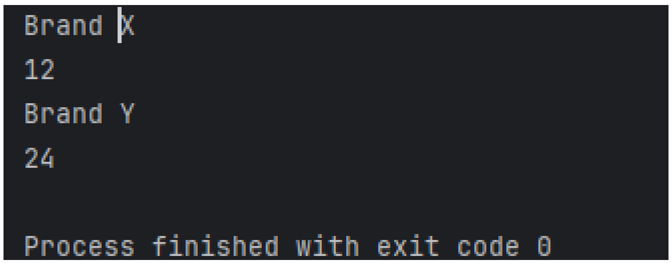
Code:

```
class Electronics(Product):
    def __init__(self, productId, productName, description, price,
quantityInStock, type, brand, warrantyPeriod):
        super().__init__(productId, productName, description, price,
quantityInStock, type)
        self.brand = brand
        self.warrantyPeriod = warrantyPeriod

    # Getter method for brand
    def getBrand(self):
        return self.brand

    # Setter method for brand
    def setBrand(self, brand):
        self.brand = brand

    # Getter method for warrantyPeriod
    def getWarrantyPeriod(self):
        return self.warrantyPeriod

    # Setter method for warrantyPeriod
    def setWarrantyPeriod(self, warrantyPeriod):
        self.warrantyPeriod = warrantyPeriod
```

Implementing:

```
electronics_product = Electronics(1, "Laptop", "High-performance laptop",
999.99, 10, "Electronics", "Brand X", 12)

# Using getter methods
print(electronics_product.getBrand())
print(electronics_product.getWarrantyPeriod())

# Using setter methods to modify attributes
electronics_product.setBrand("Brand Y")
electronics_product.setWarrantyPeriod(24)

print(electronics_product.getBrand())
print(electronics_product.getWarrantyPeriod())
```

Output:



```
Brand X
12
Brand Y
24

Process finished with exit code 0
```

4. Create a subclass Clothing that also inherits from Product. Add attributes specific to clothing products, such as:

• size (String)

• color (String)

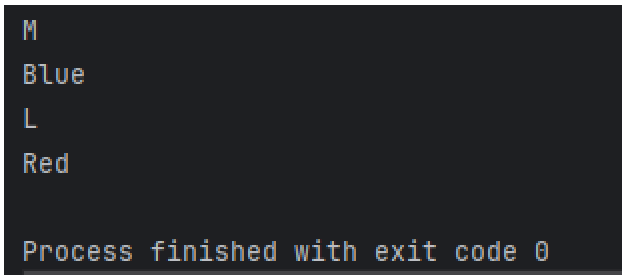
Code:

```
class Clothing(Product):
    def __init__(self, productId, productName, description, price,
quantityInStock, type, size, color):
        super().__init__(productId, productName, description, price,
quantityInStock, type)
        self.size = size
        self.color = color

    # Getter method for size
    def getSize(self):
        return self.size

    # Setter method for size
    def setSize(self, size):
        self.size = size

    # Getter method for color
    def getColor(self):
        return self.color
    # Setter method for color
    def setColor(self, color):
        self.color = color
```

Implementing:

```
clothing_product = Clothing(1, "T-shirt", "Casual cotton t-shirt", 29.99,
50, "Clothing", "M", "Blue")

# Using getter methods
print(clothing_product.getSize())
print(clothing_product.getColor())
# Using setter methods to modify attributes
clothing_product.setSize("L")
clothing_product.setColor("Red")

print(clothing_product.getSize())
print(clothing_product.getColor())
```

Output:



```
M
Blue
L
Red

Process finished with exit code 0
```

5. Create a User class with attributes:

• userId (int)

• username (String)

• password (String)

• role (String) // "Admin" or "User"

Code:

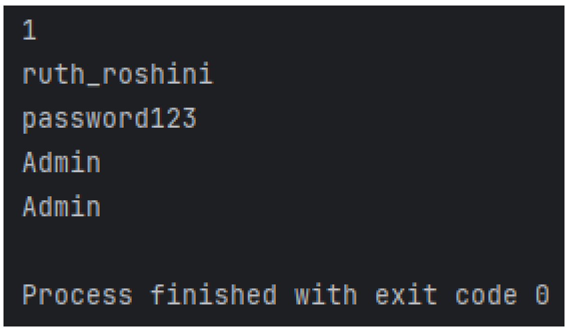
```
class User:
    def __init__(self, userId, username, password, role):
        self.userId = userId
        self.username = username
        self.password = password
        self.role = role

    # Getter method for userId
    def getUserId(self):
        return self.userId
```

```python
    # Setter method for userId
    def setUserId(self, userId):
        self.userId = userId

    # Getter method for username
    def getUsername(self):
        return self.username

    # Setter method for username
    def setUsername(self, username):
        self.username = username

    # Getter method for password
    def getPassword(self):
        return self.password

    # Setter method for password
    def setPassword(self, password):
        self.password = password

    # Getter method for role
    def getRole(self):
        return self.role

    # Setter method for role
    def setRole(self, role):
        self.role = role
```

Implementing:

```python
user = User(1, "ruth_roshini", "password123", "Admin")

# Using getter methods
print(user.getUserId())
print(user.getUsername())
print(user.getPassword())
print(user.getRole())

# Using setter methods to modify attributes
user.setRole("Admin")

print(user.getRole())
```

Output:



```
1
ruth_roshini
password123
Admin
Admin

Process finished with exit code 0
```

6. Define an interface/abstract class named IOrderManagementRepository with methods for:

• createOrder(User user, list of products): check the user as already present in database to create order or create user (store in database) and create order.

• cancelOrder(int userId, int orderId): check the userid and orderId already present in database and cancel the order. if any userId or orderId not present in database throw exception corresponding UserNotFound or OrderNotFound exception

• createProduct(User user, Product product): check the admin user as already present in database and create product and store in database.

• createUser(User user): create user and store in database for further development.

• getAllProducts(): return all product list from the database.

• getOrderByUser(User user): return all product ordered by specific user from database.

Code:

```
from abc import ABC, abstractmethod

class IOrderManagementRepository(ABC):
    @abstractmethod
    def createOrder(self, user, products):
        pass

    @abstractmethod
    def cancelOrder(self, userId, orderId):
        pass

    @abstractmethod
    def createProduct(self, user, product):
        pass

    @abstractmethod
    def createUser(self, user):
        pass

    @abstractmethod
    def getAllProducts(self):
        pass

    @abstractmethod
    def getOrderByUser(self, user):
        pass
```

7. Implement the IOrderManagementRepository interface/abstractclass in a class called OrderProcessor. This class will be responsible for managing orders.

Code:

```python
class OrderProcessor(IOrderManagementRepository):
    def __init__(self):
        pass

    def createOrder(self, user, products):
        pass

    def cancelOrder(self, userId, orderId):
        pass

    def createProduct(self, user, product):
        pass

    def createUser(self, user):
        pass

    def getAllProducts(self):
        pass

    def getOrderByUser(self, user):
        pass
```

8. Create DBUtil class and add the following method.

• static getDBConn():Connection Establish a connection to the database and return

database Connection

Code:

```python
import mysql.connector

class DBUtil:
    @staticmethod
    def getDBConn():
        # Establish connection to the database
        try:
            connection = mysql.connector.connect(
                host="localhost",
                user="root",
                password="Roshini2002*",
                database="3306"
            )
            print("Database connection established.")
            return connection
        except mysql.connector.Error as err:
            print("Error: ", err)
```

9. Create OrderManagement main class and perform following operation:

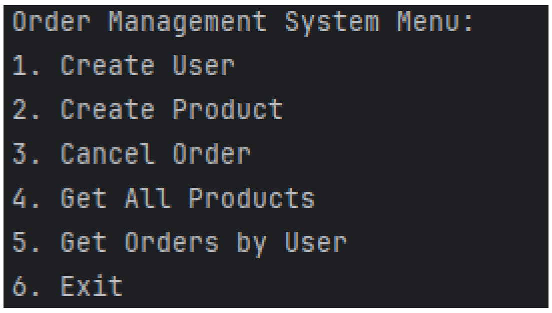
• main method to simulate the loan management system. Allow the user to interact with the system by entering choice from menu such as "createUser", "createProduct",

"cancelOrder", "getAllProducts", "getOrderbyUser", "exit".

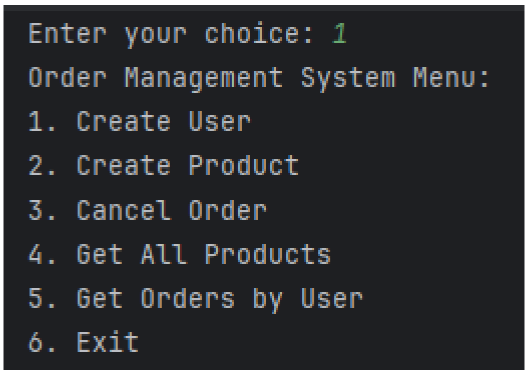
Code:

```
class OrderManagement:
    def __init__(self):
        self.orderProcessor = OrderProcessor()  # Initialize OrderProcessor
instance

    def display_menu(self):
        print("Order Management System Menu:")
        print("1. Create User")
        print("2. Create Product")
        print("3. Cancel Order")
        print("4. Get All Products")
        print("5. Get Orders by User")
        print("6. Exit")

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

            if choice == "1":
               pass
            elif choice == "2":
                 pass
            elif choice == "3":
                 pass
            elif choice == "4":
                pass
            elif choice == "5":
                pass
            elif choice == "6":
                print("Exiting Order Management System.")
                break
            else:
                print("Invalid choice. Please enter a number between 1 and
6.")


if __name__ == "__main__":
    orderManagement = OrderManagement()
    orderManagement.start()
```

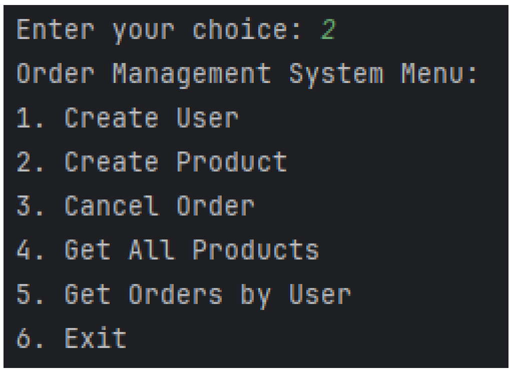
Output:



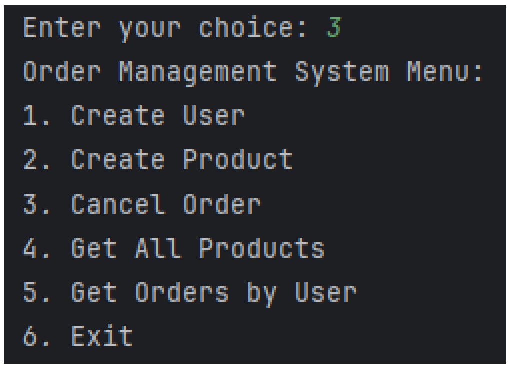
```
Order Management System Menu:
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Orders by User
6. Exit
```



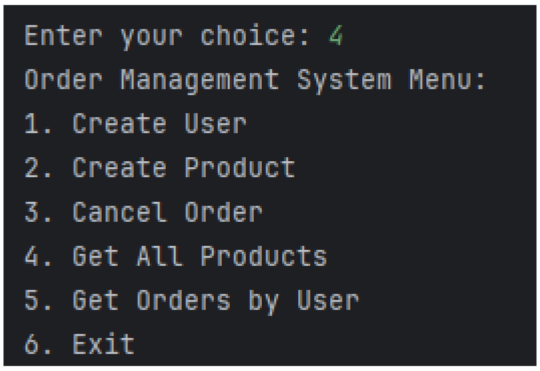
```
Enter your choice: 1
Order Management System Menu:
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Orders by User
6. Exit
```



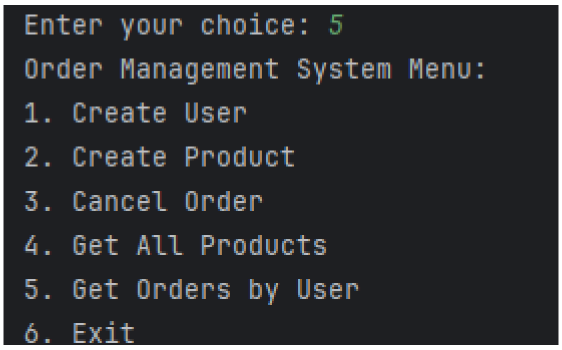
```
Enter your choice: 2
Order Management System Menu:
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Orders by User
6. Exit
```



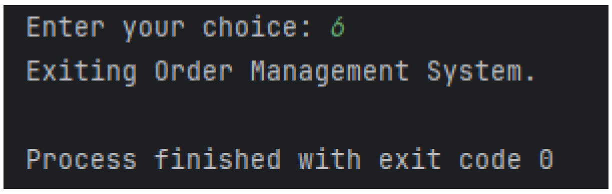
```
Enter your choice: 3
Order Management System Menu:
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Orders by User
6. Exit
```



```
Enter your choice: 4
Order Management System Menu:
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Orders by User
6. Exit
```



```
Enter your choice: 5
Order Management System Menu:
1. Create User
2. Create Product
3. Cancel Order
4. Get All Products
5. Get Orders by User
6. Exit
```



```
Enter your choice: 6
Exiting Order Management System.

Process finished with exit code 0
```