**Scenario:**

You're tasked with developing a system for an online store. The basic functionalities of this system include:

* Users can register and log in.
  + Users may have distinct roles, such as a Customer or an Owner.
* Customers can browse products, add them to their cart, and finalize their purchases.
* The Owner has the ability to add new products or modify existing ones in the store's inventory.
* An integrated inventory management system checks for the availability of products.
* An advanced payment processing system is responsible for securely handling all transactions.

**Step 1: Analyzing the Code (~20 minutes)**

Your first task is to critically analyze the provided 'GadgetStore' code. Your main objectives during this analysis are:

* Identify areas in the code that might be violating any of the SOLID principles.
* Use Booch's metrics to evaluate the quality and complexity of the codebase. Is the system maintainable? Is it modular? How coupled are the components?

*Tip: Note down your findings. These will be crucial when you proceed to the next step.*

**Step 2: Code Refinement (~20 – 30 minutes)**

Using the insights and observations from your analysis in Step 1:

* Propose an updated version of the code. Your goal is to modify the code such that it adheres to the SOLID principles more effectively.
* Apply Booch's metrics to improve the system's design, enhancing its modularity, maintainability, and reducing unnecessary complexities.

*Tip: Work in teams or pairs. Collaboration often brings multiple perspectives to a problem, which can lead to more robust solutions.*

**Workshop Deliverables:**

At the end of this workshop, you should:

* Understand SOLID principles and Booch's metrics and their importance in real-world scenarios.
* Have a refined *'GadgetStore'* that covers the principles of SOLID and aligns with Booch's metrics.
* Be able to articulate the changes made, the reasoning behind them, and the benefits that the changes bring.

*Note: Please remember that there's rarely a "one-size-fits-all" solution in software design. The goal of this exercise is to practice applying these principles and metrics to real-world scenarios, not necessarily to reach a singular "correct" solution.*

|  |  |
| --- | --- |
| **SOLID Principles**   * **Single Responsibility Principle** – A class should have only one reason to change, and have a single purpose. * **Open/Closed Principle** – Classes should be open for extension but closed for modification. * **Liskov Substitution Principle** – Subtypes must be substitutable for their base types without altering the correctness of the program. * **Interface Segregation Principle** – No client should be forced to depend on interfaces they don’t use. * **Dependency Inversion Principle** – High-level modules should not depend on low-level modules. Both should depend on abstractions (interfaces). | **Booch’s metrics**   * **Cohesion** – Aim to make responsibilities of a module or class closely related to each other. * **Coupling** – Aim for weak interconnection between classes and between components. * **Primitiveness** – Aim for functions that are composed only by fundamental/elementary operations. * **Sufficiency** – Aim to ensure that the code does the bare minimum it’s supposed to. * **Completeness** – Aim to ensure that classes capture enough details to be re-usable. |

1 **import** **java.util.\***;

2 **public** **class** **GadgetStore** {

3 **enum** UserType {

4 CUSTOMER,

5 OWNER;

6 }

7

8 **static** **class** **User** {

9 String username, password, email;

10 UserType userType;

11 ShoppingCart cart;

12

13 **public** void register(String username, String password, String email, UserType userType) {

14 **this**.username = username;

15 **this**.password = password;

16 **this**.email = email;

17 **this**.userType = userType;

18 **if** (**this**.userType == UserType.CUSTOMER) {

19 **this**.cart = **new** ShoppingCart();

20 }

21 *// save user to database logic*

22 }

23 **public** boolean login(String username, String password) {

24 *// check against database logic*

25 **return** **true**;

26 }

27 **public** UserType getUserType() {

28 *// fetch userType from database*

29 **return** **this**.userType;

30 } }

31

32 **static** **class** **Product** {

33 String name;

34 double price;

35 int stock;

36

37 **public** Product(String name, double price, int stock) {

38 **this**.name = name;

39 **this**.price = price;

40 **this**.stock = stock;

41 }

42 **public** boolean isAvailable(int desired) {

43 **return** stock >= desired;

44 }

45 **public** void updateProductInformation(String name, double price, int stock) {

46 **this**.name = name;

47 **this**.price = price;

48 **this**.stock = stock;

49 } }

50

51 **static** **class** **ShoppingCart** {

52 List<Product> products = **new** ArrayList<>();

53

54 **public** void addProduct(Product product) {

55 **if** (product.isAvailable(1)) {

56 products.add(product);

57 }

58 }

59 **public** void checkout(String paymentType) {

60 double totalAmount = 0.0;

61 **for** (Product product : products) {

62 totalAmount += product.price;

63 }

64 **if** (paymentType.equals("CreditCard")) {

65 *// process credit card payment*

66 } **else** **if** (paymentType.equals("PayPal")) {

67 *// process PayPal payment*

68 }

69 *// Deduct stock from the products*

70 } }

71

72

73 **public** **static** void main(String[] args) {

74 User user = **new** User();

75 user.register("johnDoe", "securePass", "john@email.com", UserType.CUSTOMER);

76 **if** (user.login("johnDoe", "securePass")) {

77 Product phone = **new** Product("Smartphone", 299.99, 10);

78 **if** (user.getUserType() == UserType.CUSTOMER) {

79 user.cart.addProduct(phone);

80 user.cart.checkout("CreditCard");

81 } **else** **if** (user.getUserType() == UserType.OWNER) {

82 phone.updateProductInformation("Advanced Smartphone", 399.99, 20);

83 } } } }