VENDING MACHINE DESIGN

Designing a Vending Machine Software using a low-level (LLD) approach

Vending Machine

Vending machines have become an essential part of our everyday lives, offering various products, from snacks and beverages to personal care items. While their capability can also appear simple from a user perspective, the low-level design of a vending machine includes complex info to ensure clean operation, robustness, and safety.

Introduction

This project simulates a vending machine with a user-friendly interface, secure transactions, and inventory management.

Here's a breakdown of its functionalities:

- **Product Selection:** Users can see products with IDs, names, and prices.
- **Inventory Control:** You can add various products with their details like name, price, and quantity.
- **Payment Processing:** The system validates cash payments ensuring sufficient funds.
- **Dispensing Mechanism:** Upon successful payment, the chosen product is dispensed.
- **Security Measures:** Transactions are logged for security purposes.

This code provides a foundation for a basic vending machine program, allowing further development for functionalities like:

- Accepting different payment methods (coins, bills, cards)
- Managing change dispensing
- Tracking inventory levels and generating low-stock alerts

Requirements Gathering for Vending Machine

1.1 Functional Requirements for the Vending Machine Functional Requirements generally describe and define features of the end product of a software system and simply focuses on what the end product does. These are the requirements that a system should accomplish or do like calculations, and data manipulations. Functional requirements of the Vending Machine are:

The vending machine has to maintain inventory records. The machine should allow a user to select an item and insert cash. When cash is inserted, the machine should verify that it matches the amount of the chosen item. When an item is unavailable or there is insufficient cash, the machine must display an error. Upon successful verification and transaction, the vending machine shall dispense the chosen item to the user. Ultimately, the user receives the chosen item if all of the previously mentioned stages are successful.

1.2 Non-Functional Requirements for the Vending Machine Non-functional Requirements in software engineering refer to the characteristics of a software system that are not related to specific functionality or behaviour. They describe how the system should perform, rather than what it should do. Non-Functional requirements of the Vending Machine are:

The inventory records and cash verification mechanisms shall operate with high accuracy to ensure correct transactions. The vending machine shall provide timely responses, display errors promptly, and dispensing items efficiently. The user interface shall be designed to be intuitive, making it easy for users to select items, insert cash, and understand error messages. The vending machine shall be reliable, minimizing downtime and ensuring consistent functionality. The system shall incorporate security measures to prevent unauthorized access to inventory records and cash transactions. The vending machine shall be available for use during specified operational hours, minimizing any periods of inactivity for maintenance or other reasons.

Low-Level Design (LLD) of Vending Machine vending-machine-lld

The Low-Level Design (LLD) is a crucial segment in the development of a vending machine system, following the High-Level Design (HLD).

In this segment, the focal point shifts from the conceptual structure to a more unique and granular level of implementation. The LLD digs into the specifics of every aspect, both hardware and software, presenting a roadmap for builders to translate the high-level concepts into a functioning device. It acts as a bridge among the summary illustration of the device inside the HLD and the unique implementation that happens inside the coding phase.

CODE

1.User Interface Class

This class handles user interactions, including selecting products and processing payments

```
class UserInterface:
    def __init__(self, inventory, payment_processor):
        self.inventory = inventory
        self.payment_processor = payment_processor

    def display_products(self):
        for product in self.inventory.get_products():
            print(f"{product.id}: {product.name} - ₹{product.price}")

    def select_product(self, product_id):
        product = self.inventory.get_product_by_id(product_id)
        if product:
            return product
        else:
            print("Product not available.")
            return None
```

```
    def process_payment(self, product):
        amount = float(input("Insert cash amount: "))
        if self.payment_processor.validate_payment(product.price, amount):
            self.payment_processor.complete_transaction(product.price)
            self.inventory.update_stock(product)
            print("Transaction successful. Dispensing product...")
            return True
        else:
            print("Insufficient funds or invalid payment.")
            return False
```

2. Payment Processing Class

This class manages all types of payment methods including cash, bill acceptors, and card readers.

```
class PaymentProcessor:
    def __init__(self):
        self.current_balance = 0.0

    def validate_payment(self, price, amount):
        return amount >= price

    def complete_transaction(self, amount):
        self.current_balance += amount

    def get_balance(self):
        return self.current_balance
```

3.Inventory Management Class

This class manages the inventory, including updating stock levels and checking product availability.

```
class Product:
    def __init__(self, product_id, name, price, quantity):
        self.id = product_id
        self.name = name
        self.price = price
        self.quantity = quantity

class InventoryManagement:
    def __init__(self):
        self.products = []

    def add_product(self, product):
        self.products.append(product)

    def get_products(self):
        return self.products

    def get_product_by_id(self, product_id):
        for product in self.products:
            if product.id == product_id and product.quantity > 0:
                return product
        return None

    def update_stock(self, product):
        product.quantity -= 1
```

4. Dispensing Logic Class

This class is responsible for dispensing products.

```
class DispensingMechanism:
    def dispense(self, product):
        print(f"Dispensing {product.name}")
```

5. Security Measures Class

This class ensures the security of transactions and inventory.

```
class SecurityMeasures:
    def __init__(self):
        self.security_logs = []

    def log_transaction(self, transaction):
        self.security_logs.append(transaction)
        print("Transaction logged for security.")
```

6. Main Vending Machine Class

This class integrates all components and provides a cohesive vending machine system.

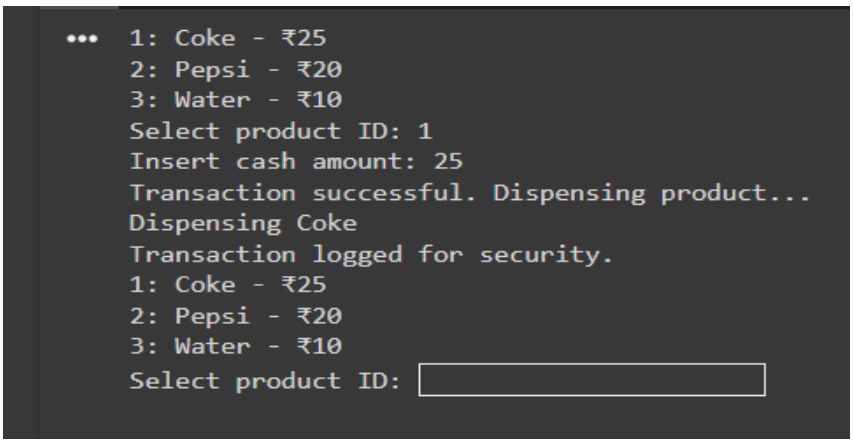
```
class VendingMachine:
    def __init__(self):
        self.inventory = InventoryManagement()
        self.payment_processor = PaymentProcessor()
        self.user_interface = UserInterface(self.inventory, self.payment_processor)
        self.dispensing_mechanism = DispensingMechanism()
        self.security_measures = SecurityMeasures()

    def add_product_to_inventory(self, product):
        self.inventory.add_product(product)

    def start(self):
        while True:
            self.user_interface.display_products()
            product_id = int(input("Select product ID: "))
            product = self.user_interface.select_product(product_id)
            if product:
                if self.user_interface.process_payment(product):
                    self.dispensing_mechanism.dispense(product)
                    self.security_measures.log_transaction(f"Dispensed {product.name} for ${product.price}")
                else:
                    print("Transaction failed.")
            else:
                print("Invalid product selection.")
```

7. Running the Vending Machine

To use the vending machine, you would create an instance of VendingMachine, add products to the inventory, and start the machine.

```
if __name__ == "__main__":
    vm = VendingMachine()
    vm.add_product_to_inventory(Product(1, "Coke", 25, 10))
    vm.add_product_to_inventory(Product(2, "Pepsi", 20, 10))
    vm.add_product_to_inventory(Product(3, "Water", 10, 20))
    vm.start()
```

OUTPUT



Advantages of the Vending Machine Project:

1. **Modularity and Extensibility**:
    - The design is modular, with separate classes handling different responsibilities, making it easy to extend functionality. For example, adding new payment methods or inventory features can be done by extending existing classes or adding new ones.
2. **Maintainability**:
    - The low-level design approach allows for easier maintenance. Each class has a well-defined responsibility, making it easier to locate and fix issues or add new features without affecting the entire system.
3. **Security**:
    - The inclusion of a security measures class ensures that all transactions are logged, providing a robust mechanism for tracking and preventing unauthorized access or fraud.
4. **User-Friendly Interface**:
    - The user interface class provides a simple and intuitive interaction for users, which can be crucial in a real-world scenario where ease of use is important.
5. **Scalability**:
    - The design can be scaled to support more complex vending machines, including those with multiple payment methods, larger inventories, and more sophisticated dispensing mechanisms.
6. **Separation of Concerns**:
    - The system follows the principle of separation of concerns, where each class handles a specific part of the vending machine's functionality, making the system more organized and easier to manage.

Disadvantages of the Vending Machine Project:

1. **Initial Complexity**:

- The low-level design may be complex for beginners or for smaller vending machine projects where such detailed architecture might be overkill.

2. **Limited Payment Processing**:
    - The current implementation only supports cash payments. Extending this to handle coins, bills, or card payments would require additional development effort.
3. **Error Handling**:
    - While the design includes basic error handling (e.g., insufficient funds), more robust error handling mechanisms might be needed in a real-world application, especially for edge cases like hardware malfunctions or unexpected inputs.
4. **Inventory Management Simplicity**:
    - The inventory management system is basic and does not include advanced features like low-stock alerts or expiry date management, which could be important in real-world vending machines.
5. **Lack of Real-Time Updates**:
    - The current system does not support real-time updates for inventory or sales data, which could be a limitation for larger vending machine networks that require centralized monitoring and control.
6. **No Support for Change Dispensing**:
    - The system does not manage change dispensing, which is an essential feature for vending machines that accept cash. This would require additional logic to calculate and dispense the correct change.

This project provides a solid foundation for a vending machine software system, with clear areas for improvement and expansion based on specific use cases or requirements.