Step 1: Requirements Description

Functional Requirements:

- The machine must accept payment (coins, cards).
- The user can select a drink from the menu.
- The machine dispenses the selected drink after successful payment.
- The machine returns change if the amount inserted exceeds the drink's price.
- The machine must notify the user when ingredients are out of stock.

Non-Functional Requirements:

- **Service Speed**: Dispensing a drink should take no more than 1 minute.
- **Reliability**: The system should operate without failures 99% of the time.
- **Ease of Use**: A simple interface with a screen for navigation.
- **Security**: Secure storage of monetary funds.

Step 2: Developing Use Cases

Use Case 1: Buying a Drink

Actor: User

Steps:

1. The user inserts coins/scans a card.
2. The user selects a drink from the menu.
3. The machine checks the availability of ingredients and the sufficiency of payment.
4. If everything is fine, the machine dispenses the drink and returns change (if needed).
5. If the payment is insufficient or ingredients are unavailable, the machine displays an appropriate message.

Use Case 2: Refilling the Machine

Actor: Operator

Steps:

1. The operator opens access to the internal mechanisms.
2. Refills the ingredients (water, coffee, sugar, etc.).
3. Closes the machine.

Step 3: Identification of Objects, Classes, and Relationships

Objects:

- Coffee Machine (CoffeeMachine)
- Drink (Drink)
- User (User)
- Payment System (PaymentSystem)
- Ingredients (Ingredients)

Classes:

1. **CoffeeMachine**

- **Attributes**: currentBalance, drinkMenu, ingredients.
- **Methods**: acceptPayment(), selectDrink(), dispenseDrink(), refillIngredients().

2. **Drink**

- **Attributes**: name, price, ingredientsRequired.
- **Methods**: None.

3. **PaymentSystem**

- **Attributes**: totalBalance.
- **Methods**: acceptPayment(), returnChange().

4. **Ingredients**

- **Attributes**: water, coffeeBeans, milk, sugar.
- **Methods**: useIngredient().

Relationships Between Classes

1. **CoffeeMachine ↔ Drink (Relationship: "uses"):**

The CoffeeMachine class has a list of drinks (drinkMenu) that it offers to users.

Association: 1-to-many (one coffee machine offers multiple drinks).

2. **CoffeeMachine ↔ PaymentSystem (Relationship: "has"):**

The CoffeeMachine class uses the PaymentSystem to handle payments.

Association: 1-to-1 (one coffee machine uses one payment system).

3. **CoffeeMachine ↔ Ingredients (Relationship: "manages"):**

The CoffeeMachine class manages ingredient stocks (water, coffee, sugar, etc.) through the Ingredients object.

Association: 1-to-1.

4. **Drink ↔ Ingredients (Relationship: "requires"):**

The Drink class describes the set of ingredients required to prepare the drink.

Association: 1-to-many (one drink requires multiple ingredients).