The observer pattern

START

Team:

Phan Huỳnh Anh Thư - 17095
Võ Công Minh - 10421040

Example:
Milk shop & Clients

Definition:

the subject: the "subject" refers to the object that is being observed by one or more "observers". The subject is typically a core component of the system, The subject maintains a list of its observers and provides methods for adding, removing, and notifying observers

The observer: an "observer" is an object that is being notified when the state of a "subject" object changes.
The observer maintains a reference to the subject and registers itself with the subject to receive notifications.
When the state of the subject changes, it notifies all of its registered observers by calling a method on each observer, typically passing some information about the new state as an argument.

Definition:

The observer pattern: a design pattern that defines a one-to-many relationship between objects, where a change in the state of one object (the "subject") results in notifications being sent to one or more other objects (the "observers").

Implemetation

```cpp
// Subject interface
class Subject {
public:
    virtual void registerObserver(Observer* observer) = 0;
    virtual void removeObserver(Observer* observer) = 0;
    virtual void notifyObservers() = 0;
};

// Concrete subject class (milkshop)
class MilkShop : public Subject {
private:
    vector<Observer*> observers;
    string currentMilkBatch;

public:
    void registerObserver(Observer* observer) {
        observers.push_back(observer);
    }

    void removeObserver(Observer* observer) {
        for (int i = 0; i < observers.size(); i++) {
            if (observers[i] == observer) {
                observers.erase(observers.begin() + i);
                break;
            }
        }
    }

    void notifyObservers() {
        for (int i = 0; i < observers.size(); i++) {
            observers[i]->update(currentMilkBatch);
        }
    }

    void setMilkBatch(string milkBatch) {
        currentMilkBatch = milkBatch;
        notifyObservers();
    }
};
```

Subject interface: an abstract interface that **defines** the operations that must be supported by the subject in order to allow observers to register, unregister, and receive notifications.

Concrete Subject: The subject interface **provides** a way for observers to register and unregister themselves with the subject, as well as receive notifications when the subject's state changes.

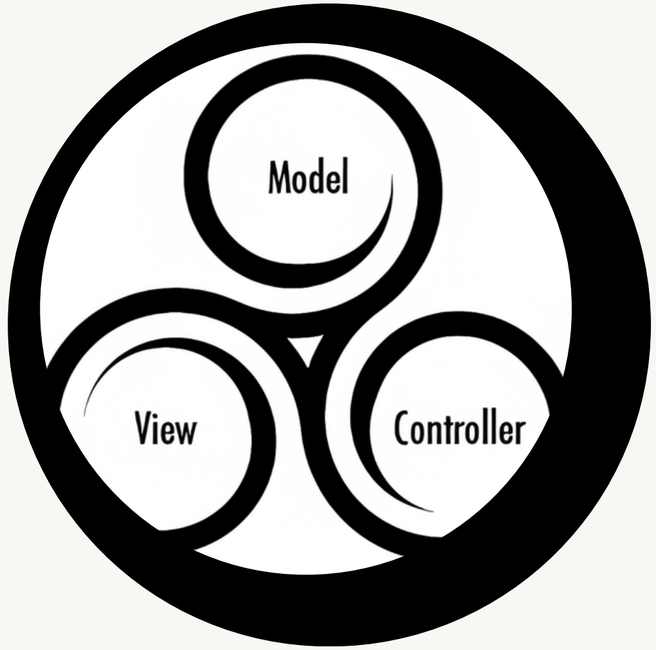
```cpp
// Observer interface
class Observer {
public:
    virtual void update(string milkBatch) = 0;
};
// Concrete observer class (client)
class Client : public Observer {
private:
    string name;

public:
    Client(string name) {
        this->name = name;
    }

    void update(string milkBatch) {
        cout << name << "Notification: " << milkBatch << endl;
    }
};
```

Observer interface:
The observer interface is an abstract interface that **defines** the operations that must be supported by all observers.

Concrete observer: A concrete observer is a **specific implementation** of the observer interface. It receives notifications from the subject and responds to them in a specific way.

REAL-LIFE EXAMPLE

MVC Model



Avantages

1. Separation of concerns
2. Extensibility
3. Maintainability
4. Decoupling

Disavantages

1. Overhead
2. Complexity
3. Timing issues
4. Memory management

THANK YOU!