Consistent update of user interface elements

## Design Problem

The VMCS design, is targeted for two types of users, namely customer, controller, and maintainer. The design achieves to satisfy its targeted users with the help of rich user interface panels. Customer accesses the system via Customer Panel, controller makes use of the Machinery Panel, and maintainer uses the Maintenance Panel.

All three user interface panels display the name, quantity, and/or price of all store items. Store items currently consist if various type of drinks and various denominations of coins.

Consider following use cases –

1. Maintainer:
   1. Updates price of any drink
   2. Transfer all cash from the store
2. Controller:
   1. Update quantity of any denomination of coins
   2. Update quantity of any drinks

All these above activities require to display updated quantities on both Maintenance and Machinery panels. Which means that changes done in maintenance panel must be visible in machinery panel and vice-versa.

With the current design, the update mechanism requires that the Maintenance and Machinery panel know each other and are able to notify whenever there is any change in store items. In other words, they are tightly coupled.

The panels/controllers also makes use of update and refresh mechanisms for the complete panel instead of only updating the changed item. This complicates things future if any requirement changes like, introduction of new store item, say snacks, or addition of a new user interface panel, are to be implemented.

## Candidate design patterns

1. Command PatternIntent:
   1. Encapsulate a request as an object.
   2. Parameterize clients with different requests, queue or log requests.
   3. Support undoable operations.
2. Mediator PatternIntent:
3. Define an object that encapsulates how a set of objects interact.
4. Promote loose coupling by keeping objects from referring to each other explicitly.
5. Vary the interaction independently.

1. Observer PatternIntent:
   1. Define one-to-many dependency between objects.
   2. Notify all dependents when object state changes.

Selected pattern:Observer

## Motivation

There are some panels sharing the quantity of coins and drinks, so if the changes take place in the quantity of coins and drinks, those panels are needed to be informed. The important of the consistent of viewing the data has become obvious. The observer pattern helps the views to be notified when a change occurs in the data object. So there is a change in the quantity of coins or drinks, any panels which are attached with the data will be notified. In the observer pattern, the key objects are subject and observer. In VMCS, the coin item and drink item which contain the information of the quantity will become Subject. So any panels which are interested on those items will become observer. One clear advantage is that there are no modifications on the objects when new panel is introduced or changes in the view. Those store items will send out the notifications without having to know who its observers.

## 

## Implementation

## D:\develop\BitBucket\vmcs_oodp\VMCS\documents\Main.jpg

Class diagram

### Participants

* Subject

- Observable - abstract class

- Knows its Observers. Any number of observers can observe the subject.

- Provides a mechanism to attach and detach observers and to notify changes.

* Concrete Subject

- StoreItem (CashStoreItem, and DrinkStoreItem inherit this role)

- Stores state of interest to ConcreteObserver objects.

- Notifies observers when state changes.

* Observer

- Observer - interface

- Defines an updating interface for objects that should be notified of changes in a subject.

* Concrete Observer

- LabeledDisplay, ButtonItem, and ButtonItemDisplay

- Maintains a consistent reference to a ConcreteSubjects state.

- Implements the Observer updating interface to keep its state consistent with the Subject’s.