# Description of Design Problem

* StoreController has dependency to DrinkPropertyLoader, CashPropertyLoader,CashStore and DrinkStore
* StoreControlelr implemented both Cash and Drink related methods:
  + Cash:
    - getTotalCash()
    - giveChange()
    - storeCoins()
    - transferall()
  + Drink
    - dispenseDrink()
    - setPrice()
  + Common methods
    - saveProperties()
    - initializeStores()
* Anything changed in Store/Cash store, there will be impact to StoreController.

# Candidate Design Patterns Considered

* Strategy pattern

# Pattern Chosen

When design follows strategy pattern, the algorithmic behaviour is separated out and implemented in separate interfaces and encapsulating subclasses. This decouples the behaviour and the class that uses this behaviour. The behaviour can be changed without breaking the classes that use it.

# Participants

  The classes and objects participating in this pattern are:

* **StoreController**

This class has been made as abstract class. Implements only common method across both Cash and Drink store

* **CashStoreController**

This is concert controller to handle cash store specific methods.

* **DrinkStoreController**

This is concert controller to handle drink store specific methods.

* **Client** 
  + MainController

# Diagrams

## Class Diagram

Before



Figure 1: Class Diagram for Strategy Pattern

## After



## Sequence Diagram

Before



Figure 2: Sequence Diagram for old design

After



Figure 3: Sequence Diagram for new design (Strategy Pattern)