**Vending Machine Design**

Applied the GoF state design pattern to model the states of the Vending machine. There is a state interface called IState that contains a method for every action in the Vending machine (see the flowchart below). The concrete State classes are responsible for the behaviour of the machine when it is in corresponding state. Each concrete state class provides its own implementation for a request. Whenever the request is made on the Context (Vending Machine) it is delegated to the state for handling. There is a **TransactionManager** class created by the **VendingMachine** class which is responsible for the inventory and cash calculations and tracking. The **CalculateChange()** method contains the Coin change algorithm logic.

****

**Flow of Vending Machine** (All Actions of the **IState** interface are marked in the Flow below)

SelectItem();

Insert desired money for the product

Is product available ?

Select Product

No

DispenseItem();

DispenseChange();

Yes

No InsertCoins();

Is Right amount?

Cancel Request

EjectCoin();

Dispense Item & Tender change (if any)

Return amount

Assumptions/Limitations:

1. The steps in the **flowchart** above are assumption for the Vending machine states i.e.

It does not proceed until the user selects an item. May be one alternative could be that user should be able to insert coins first and then select the product (which is a valid workflow)

1. **Algorithm** for coin change currently uses the **greedy** approach (which may not be the best optimal solution). More optimal solution can be achieved using **Dynamic programming** technique.
2. **Unit Tests** have been written for TransactionManager class’s CoinChange algorithm and the WaitUserSelectionState class. Should be written for every class in the assembly such that making code changes is easier.
3. The product prices are integers and not float/double
4. All products are hardcoded with name, price and quantity (defined as constants)
5. All items in the inventory are initialised with the same count initially
6. Given the current functionality, chosen **IState** interface instead of an abstract class because there was no common functionality which could have been refactored into a base abstract class.
7. Currently cannot generate a report of the inventory and current state. But, it’s simple to implement since we have the data.