Design patterns applied:

**Strategy Pattern** was implemented for the logic of updating each inventory day. This was necessary to get rid of messy client code. The user now creates an instance of the *InventoryUpdater,* which runs the entire report of the inventory based on the number of days passed to it. The operations are abstracted away from the client. Additionally, override functions built on the base *ItemQualityUpdater* class determine the Sell-In and Quality handling. This makes adding new rules easier for the user: only new classes derived from the base class need to be added to the program.

**Iterator Pattern** was implemented to update the inventory for a selected number of days. The class *ConcreteAggregate* holds a list of all of the days that have been updated. The iterator class implements methods that allow us to traverse the entire collection of days and pull any records. This reduces the record making to just two lines within the client: the user declared a new *ConcreteAggregate* instance, then passes it to the update function from the *InventoryUpdater* instance.

SOLID principles applied:

Single-Responsibility Principle was applied in the *InventoryUpdater* class. The functions which alter the inventory have been delegated to the *ItemQualityUpdater* class and their inherited implementations.

Open-Closed Principle was applied in the UpdateQuality method: since each item is handled differently based on it’s name, an interface was implemented to handle different quality changes. Now, each call of UpdateQuality method creates a new instance of *ItemNameStrategy*, and the name is passed directly into the interface. Additional rules may be added to the interface without disturbing the integrity of the *IntentoryUpdater* class.