Approach -

The main business logic of updating quantity which is based on the Trade Type and Account Type is handled using Strategy Design Pattern.

As Strategy Design Patterns suggests us to separate out the behavior , I’ve created two classes which handle the way quantity will behave . Quantity of the Instrument will either Increase or Decrease in the current scenario .  
Therefore 2 classes **IncreaseQuantityStrategy** and **DecreaseQuantityStrategy**  have been created .

I’ve created an AccountFactory class which returns a new instance of IAccount or EAccount based on the input. AccountFactory holds a accountType map which is used to create instances as per the input

I’ve created 2 trade classes BuyTrade and SellTrade . Each will have its own UpdateQuantity Strategy based on Account type .

AccountType will update quantity based on the strategy provided to it while the trades are applied to the positons.

Each Position class has a Set<Accounts> as every position has an Internal & External Account.

Once all the start of day positions is loaded in the positionMap and the InputTransactions are loaded in the List<Trade> listOfTrade.

Each trade is then applied on the position and also delta is computed.

After this all the positions are written to the file .