**Sales Tax Application**

The design on the Application can be refined more depending on the timelines available for Solution design. While designing this solution SOLID principles have been followed, apart from that I have kept following things in mind

* **Domain Modeling**   
  How do you create a good model of the solution?   
  What objects do you create?

How will they solve the requirements?

* **Separation of concerns, loose coupling, high cohesion**

How do you separate out the parts of the design that have different concerns or rates of change and how do you relate them?

How do you keep your design flexible and current?

**Design Pattern Approach**

I have used two different kinds of design pattern for the design of Sales Tax Problem. Below are the Design patterns:

**Abstract Factory Design Pattern**

The Product is an Abstract Super Class and different kinds of items like Book Cooler, Medicine, Food, etc. will inherit from the Product class. It will give the facility to extend the new Product type in future.

Here, main reason to separate class for each product type is that other attributes can be added in the future by just extending the Product class.

**Advantages**

1. If new Product will be added, then there will be no need to change the existing client code.

**Strategy Design Pattern**

Tax Calculation has been designed using Strategy Pattern.

So, below classes will be involved,

1. Strategy - ITaxCalculator
2. Concrete Strategy - LocalTaxCalculator
3. Context - Biller

**Advantages**

1. We can add new algorithm for Tax Calculation in the feature easily and algorithm can be selected at run time.
2. The actual creation of a TaxCalculator will be delegated to Factory method.

Based on the above factors/considerations the Solution structure looks like below..

**SalesTax.Client**

Project represents the Console Client Application which consumes the SalesTax solution and provides the output .

**SalesTax.Domain**

This is the heart of the Application, it represents all our Domain entities, entities relationship and solution implementation.

**SalesTax.Domain.Tests**

This project constitutes the basic isolated unit test covering the SalesTax.Domain functionality. I have written very basic unit test out of time constraint .Please note the Unit testing can be refined and extended further by using

* Autofac for mocking your dependencies (Service Calls, DB Calls etc)
* Writing Integration test
* Extending the Unit test coverage by writing comprehensive isolated and Integration tests.

**How to run the Application**

* Unzip the zip file in a windows folder
* Open the SalesTax.sln file
* Run the Solution from Visual Studio and you should see the results.