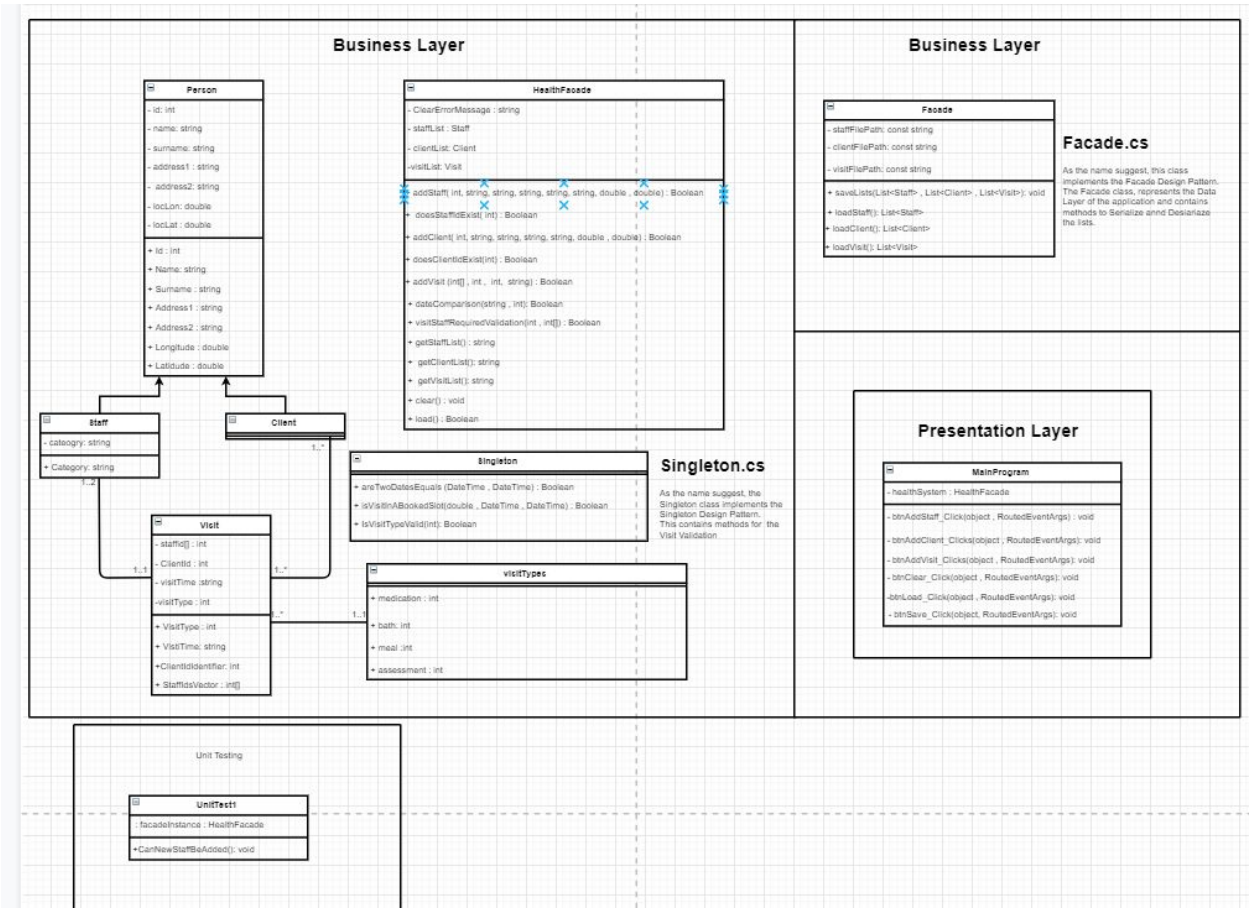


3-layer architecture

The three level / 3 tier architecture, is a client-server architecture logically splitted in three different levels:

- **Business Layer:** This layer contains all the code about for the logic and behavior of the application.
- **Presentation Layer**: it is also known as GUI layer. Thee presentation layer contains usually all the code for the interaction between the user and the application.
- **Data Layer:** This link provides a link to a data store, that can be a generic file (.txt , .bin etc..) , a spreadsheet , a web service or a Database.

Use the three layer architecture can help us from different point of view. 3 Layer architecture:

1. Allow to work contemporary on the three layers
2. Allows to modify one of three layer or all of them at the same time, without affect in any way

the other layers.

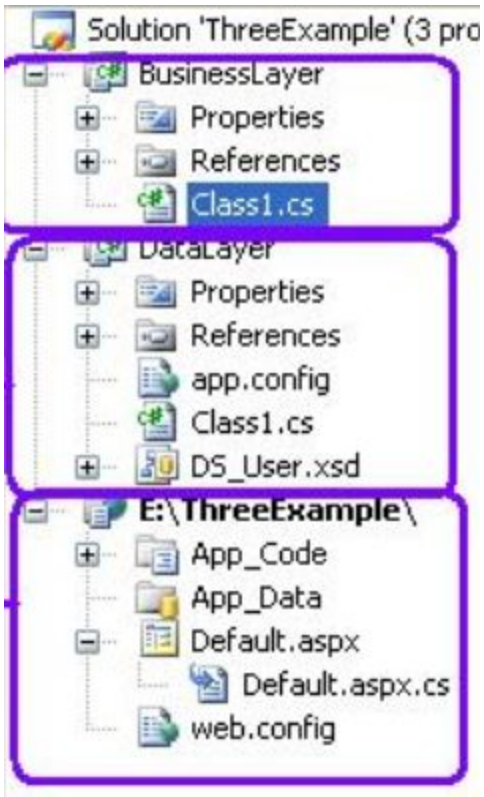
3. Provides an extremely easy way to maintain the code base, managing the presentation

code and

business / logic code separately.

The implementation of the 3 Layer architecture is normally done in conjunction with Design Patterns: As Façade , Decorator, Singleton etc..

Watching out solution explorer, a project that uses the 3 Layer Architecture can have the following structure:



In a real scenario, building a 3 Layer Architecture project with C# .Net, in the following layers we could have these classes:

- **Presentation Layer:** It could contain a .xaml file for the GUI and a classes for the event management for all / most of the GUI elements.
- **Business /logic layer:** contains classes for the logic part implementation of the application.
- **Data Layer:** class for the Serialization/ Derealization of the objects or any relevant data. In thi layer, we could have .txt , .bin, .xml, json files (we are using file for the Data Storage), or our DataSet Container.

Sources:

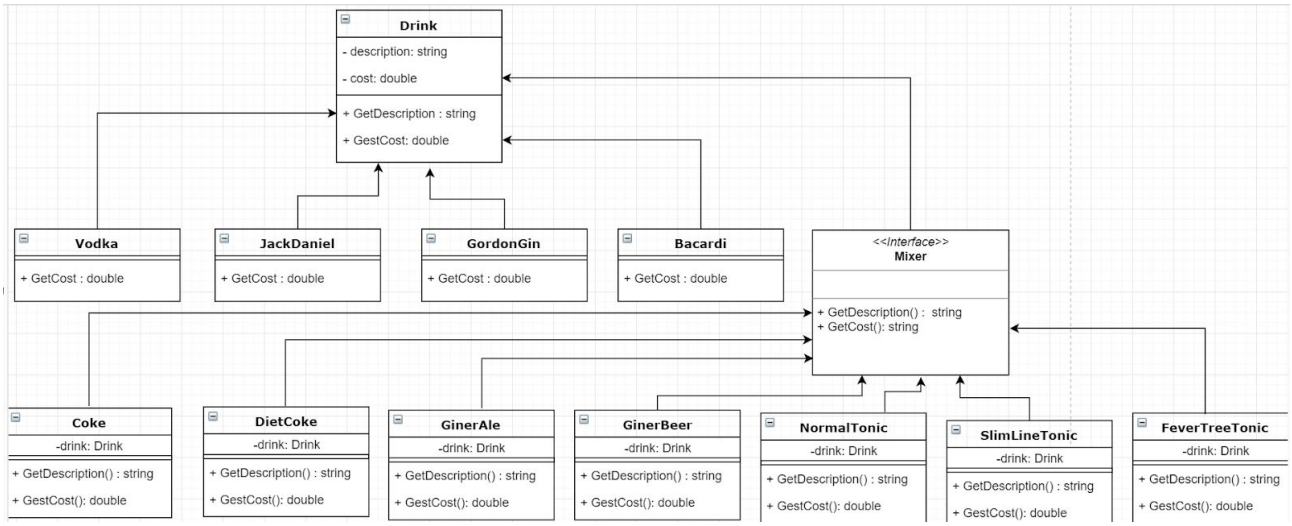
- Architecture Presentation – Provided on Moodle by Dt. Neil Urquhart
- 3 Layer Architecture - www.izenda.com
- Picture from onlineBuzz.com

Decorator Design Pattern

The Decorator Pattern is based on the open-closed principle. This Design Pattern provides a valid alternative to subclassing for extending behavior. It involves a set of decorator classes that are used to wrap concrete components.

Of course, with this Design Pattern we can wrap a component with an unlimited number of decorators.

Decorator is a Design Pattern that allows to add dynamically a behavior to an object individually, without affecting the behavior of other objects from the same class.



Example created By: Davide Pollicino

In according with the Diagram above:

- Drink is out abstract component class
- There four different concrete components, called: Vodka, JackDaniel , GordonGin and Bacardi.
- Mixer is going to be out abstract decorator
- Coke, DietCoke , GingerAle , GingerBeer, NormalTonic , SlimlineTonic and FeverTreeTonic are our concrete decorators.

Sources: ima.udg.edu