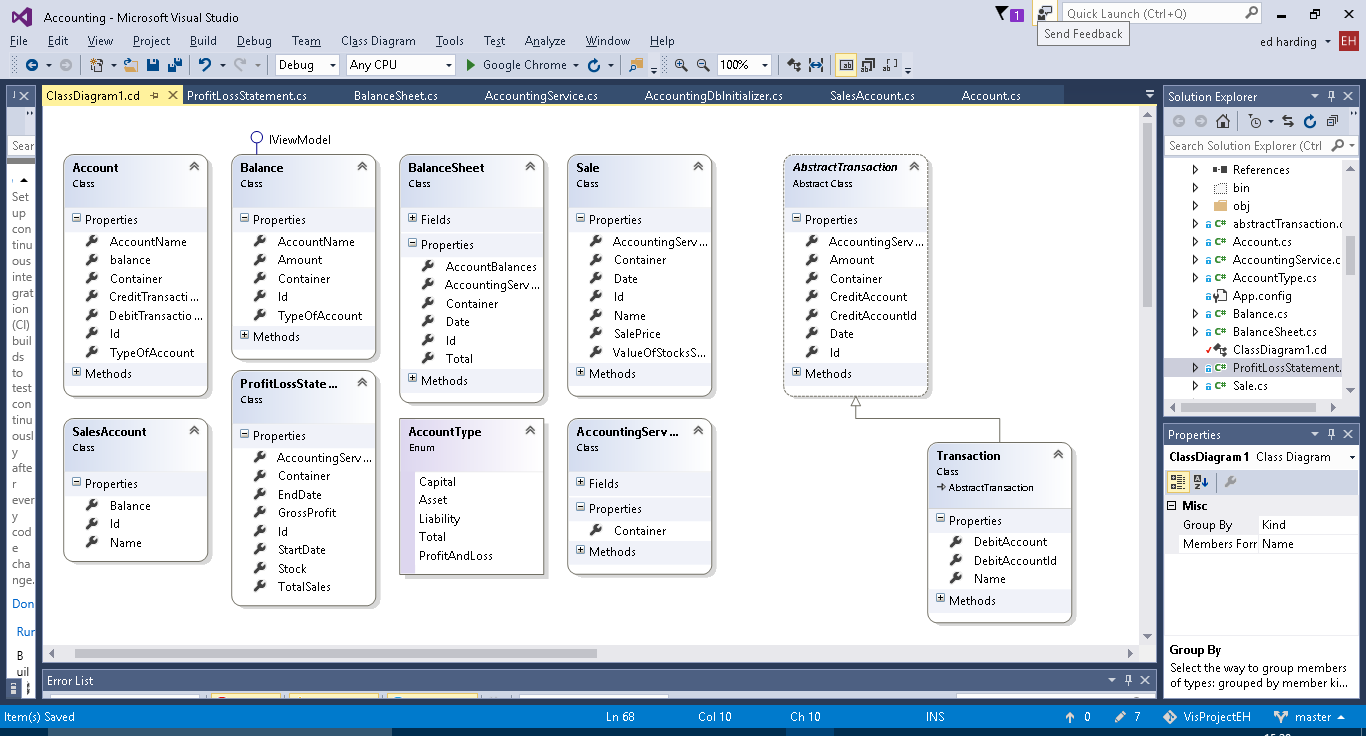
Why I chose to do an Object Oriented Project

The greatest reason I had for choosing object oriented programming was due to the ability to create user defined objects to be used in my program. After I decided on my project idea I sat down and thought about some of the basic functionality it would need to do and the easiest way to implement it. I had already had some experience with OOP and after looking at the basics of an accounting system I noticed that there were lots of clear objects which could be created. The ability to create your own object set properties and give functionality to the objects really stood out as a necessity to me in designing my project.

A more trivial reason for my decision to go with OOP was just for organisation of the code in visual studio. OOP allows for your code to be split up over multiple segments which allows for easy navigation and organisation of your program.

Another big selling point to use polymorphism is inheritance, having the ability to create sub/super classes of your use defined objects is incredibly useful to reduce the repetition of code and functionality, which in turn reduces the likelihood of errors and usually makes them easier to fix if they effect one part of a class hierarchy rather than finding an error in a part of code in an object which is in other objects and then having to individually fix them all.

Class Diagram and Objects Used

Account

The account class is – as it is named – meant to represent an account in a running business. Every business has a list of different accounts which are used to record the movement of money and value of objects owned by the company. For example, a small retail company may have an account for the current cash held by them in store.

The account contains a collection of transactions and a calculated balance from these transactions. As this balance is used in the balance sheets and balance sheets can be made for past dates a function to calculate the balance at a specific date is needed. For both of the above a function to calculate a total balance is needed therefore a function can be made to do this to reduce repetition.

Balance

The Balance class doesn’t really have a business application behind it, it was just necessary to have for functionality purposes to be used as to retrieve and format the account data into the balance sheet.

Balance Sheet

As it is named the balance sheet object is made to create a balance sheet and calculate the balances of all the accounts and make sure that the capital equals the company assets minus liabilities.

A title builder method is implemented this is so that an apt title can be created automatically for the user using the date for which the balance sheet is for. This allows for them to be ordered in the user interface by date for user convenience.

Abstract transaction

An Abstract super class for transactions.

Contains an auto complete method for the account to be credited for user convenience.

Transaction

The most common type of transaction made in a double entry bookkeeping system. It debits and credits individual accounts as it should. As you can see from the class diagram it is a sub class of Abstract Transaction.

Transaction also contains a title builder which creates a title consisting of the inputted transaction name and the date of which the transaction occurred, again this is just for user convenience and for data recording purposes. As Transaction is a sub class of abstract transaction it auto inherits the auto complete method for the account to be credited. However, it also contains an additional auto complete for the account to be debited (works the same as the credit one).

Account Type

The accounts in a double entry book keeping system fall into three categories capital, assets and liabilities. I need a way to represent them, so that total balances of each category can be calculated so that the balance can be checked as already mentioned in the Balance Sheet. The easiest way to do this was to create an enum.

Accounting Services

Accounting services doesn’t really represent anything specific in business terms it just adds functions which are useful to multiple different objects and to the user in the user interface, such as being able to add new instances of objects like Account and transactions. It also contains methods for the user such as listing all the accounts, transactions, balance sheets etc.

**Sale**

Is used in the profit and loss side of the program. Represents a sale of goods to a customer, and records the profit or loss made from that sale.

Sale also has a Title builder method which like transaction creates a title from the date of the sale and the name given to the sale.

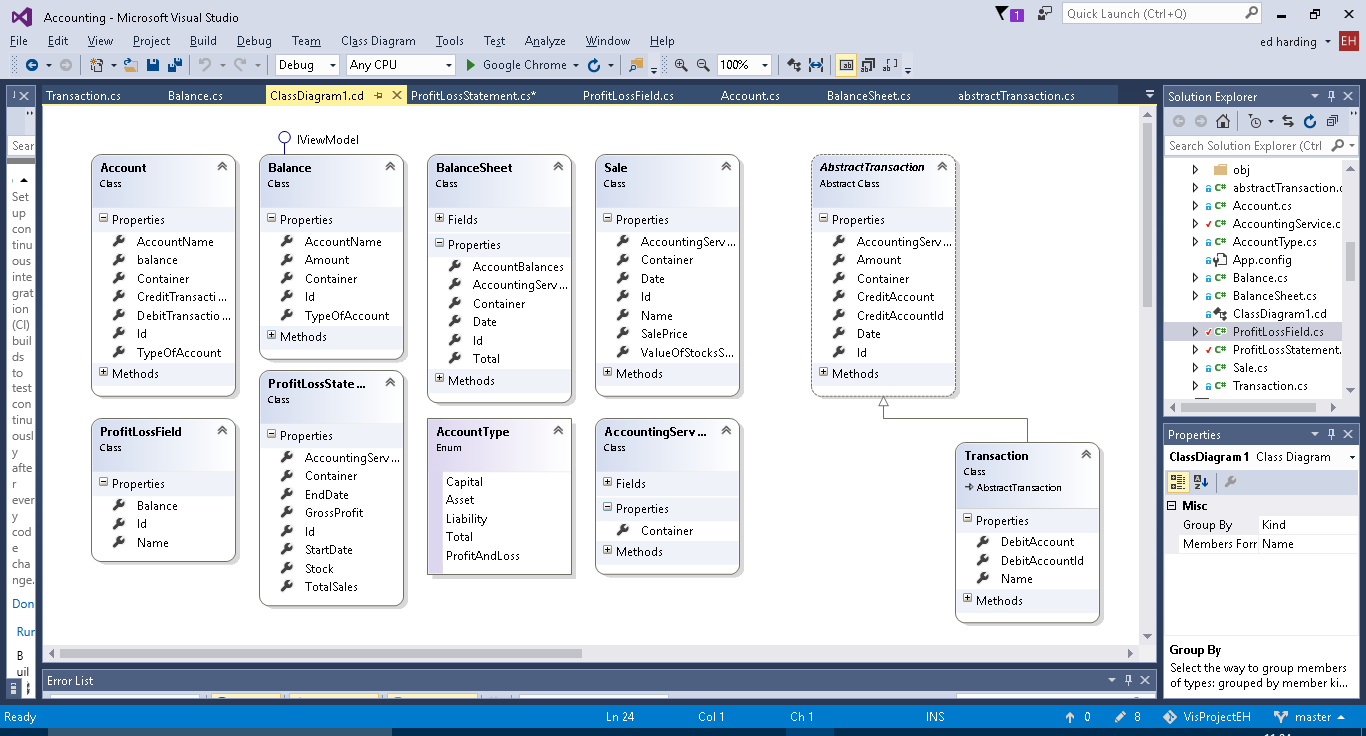
**Sales Account**

Is used to create the fields used in the profit and loss statement, those being the total value of the stock and the total value for sales made.

Needs renaming to ProfitLossField.

**Profit Loss Statement**

As the name suggests this class creates a profit and loss statement for a given time frame. It does this by getting the Sales (from Sale) made in that time frame and the value of the stock (at the latest point of the time frame) and then calculating the amount of stock sold and for what total price it was all sold at.

**Updated Class Diagram**