# Analysis

Need to talk about the need to audit and why it is important.

## **The problem background**

Utopia Handmade and Vintage is a retail and clothing company. It is run by my brother Mr. T. Harding and his partner. The business is quite small and they tend to make most of their sales through online third-party websites such as etsy.com and also at stalls at multiple different music festivals. As the business is starting to grow and the company is transitioning from something more of a hobby over to an actual business, my brother has come to the realization that taxes and bookkeeping are legal responsibilities which will become very prevalent in his future. A lot of small business owners tend to use Microsoft Excel or bookkeeping software which can be bought online such as ‘Quickbooks’ and ‘Sage’ however I intend to make a program which can be used (most likely by him) to help assist and record the bookkeeping for the company.

As I personally had a very limited understanding of bookkeeping and accounting I decided that step one would be to do some basic preliminary research into this field. I came to the understanding that double entry bookkeeping is the standard method used in the modern world. Some other things I noted is the need and usefulness of balance sheets and profit and loss statements.

The second step I decided to take was to arrange a meeting with my brother as he would be the end user of the product I intend to build. The two things I was hoping to address with him were his current options for bookkeeping software which he could purchase and use, and the requirements he would need in a bookkeeping software for his business. After some quick research we concluded that although excel was the most cost-efficient option not only did my brother not know anything about using it but it also only allowed cashbook accounting where as he was looking for a double entry bookkeeping system. The other requirements he noted will be mentioned further on.

To conclude I decided that I would start of by creating a very standard double entry bookkeeping system as this would act as a good foundation to add improvements to and tailor it towards the end users more specific requirements. This also helps as it allows for me to make sure the basics are all up and running before getting into a greater level of depth and detail.

## **Research into Accounting**

I came to the realization that I will need to learn a substantial amount more about accounting and bookkeeping to be able to fully take on and produce a suitable product for my client. To do this I have decided to use a very well acclaimed textbook which is used by most schools and colleges in the UK to teach accounting. It is called “Business Accounting 1” by Frank Woods and Alan Sangster. The book starts from the very basics of accounting and provides multiple practice questions and examples which I will be able to use in the testing and development of my project. I am hoping to progress my program as I progress through the textbook so I can develop the program hand in hand with my learning of the subject. This is how I plan to create the foundation for my program as stated earlier.

## **The current system**

The company is currently quite small as I stated earlier and is therefore under the minimum tax bracket. This means that currently they do not have any system in place for bookkeeping as there has been no need for it so far.

## **Clarification of The End User**

Although my client is Utopia Handmade and Vintage I would say that Mr. T Harding will be my sole end user for the system as he is one of the owner’s and the most likely out of the two to be doing the bookkeeping. He will be my first point of contact for information on what he will require from me. Therefore, the system will be catered towards his requirements and he will also be the person I get to test the program once it is in its later stages and past the book level testing.

## **User requirements**

1. The program should be able to perform the basic functions of a double entry bookkeeping system, this includes but is not limited to:

* Allowing the user to create multiple different accounts.
* Allow the user to add sub accounts.
* Allow the user to record transactions between the accounts.
* Use these transactions to form a balance sheet.
* Use transactions to make up a profit and loss statement for a requested time period.
* The program should also have the ability to audit user’s actions.

1. The system must be easy to use, simple to understand and navigate, and display data and information in a way which is tidy, efficient and easy to understand.
2. The system will NOT require any authentication process as It will be a locally run program therefore it will only be accessed from the clients own computing device.
3. The client would like the system to be as aesthetically pleasing as possible; as it will be used for long hours therefore it should not be dull.

Of course, the example I mentioned earlier such as Quickbooks and Sage can do all of these things and more however I believe that as Mr. Harding the end user is new to bookkeeping it would in fact be more beneficial to keep the accounting side to the simplest level possible. One other important thing is that my code will all be free and open source (being shared via GitHub) this allows for other users to further extend to its functionality and add complexity to it if needs be.

# Design

Add more class diagrams showing relations etc. not just inheritance.

## **Why I chose to do an Object-Oriented Project**

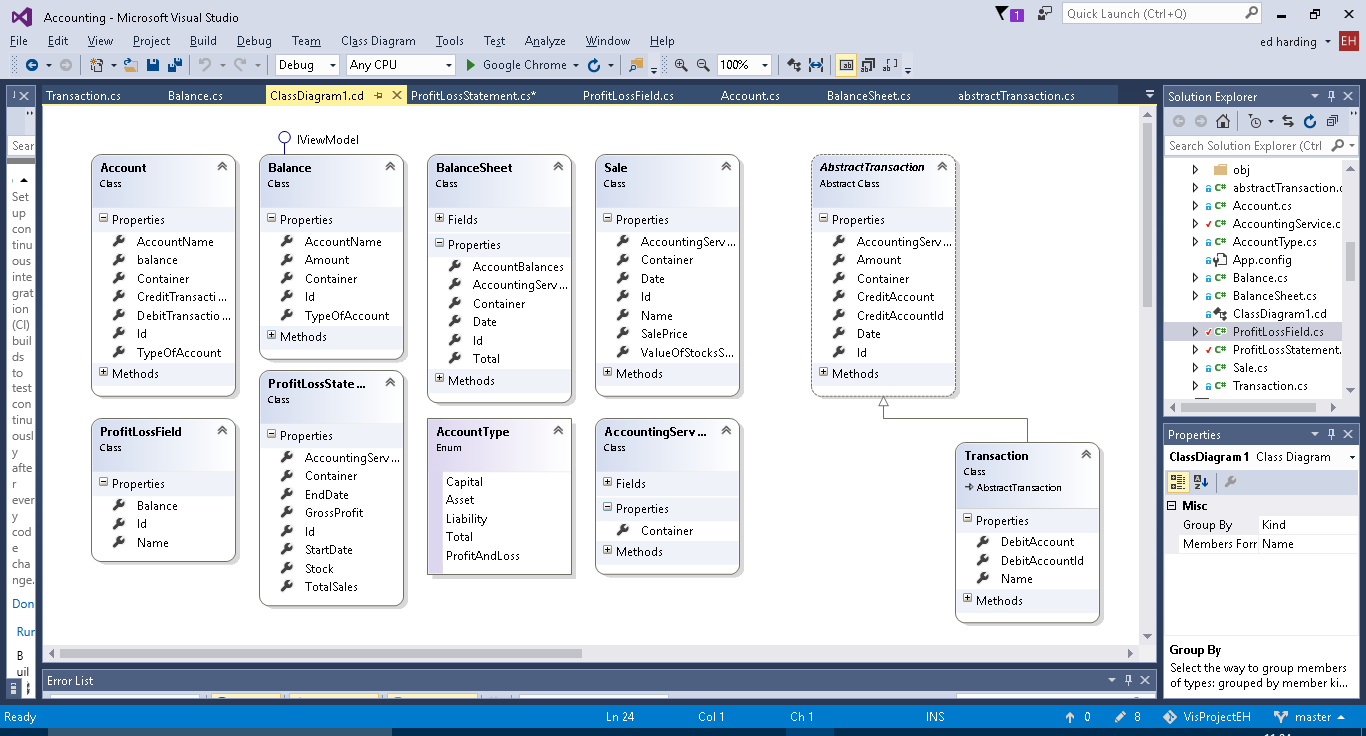
The greatest reason I had for choosing object oriented programming was due to the ability to create user defined objects to be used in my program. After I decided on my project idea I sat down and thought about some of the basic functionality it would need to do and the easiest way to implement it. I 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**

Overall Class Diagram.

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. Which will have more sub-classes.

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**

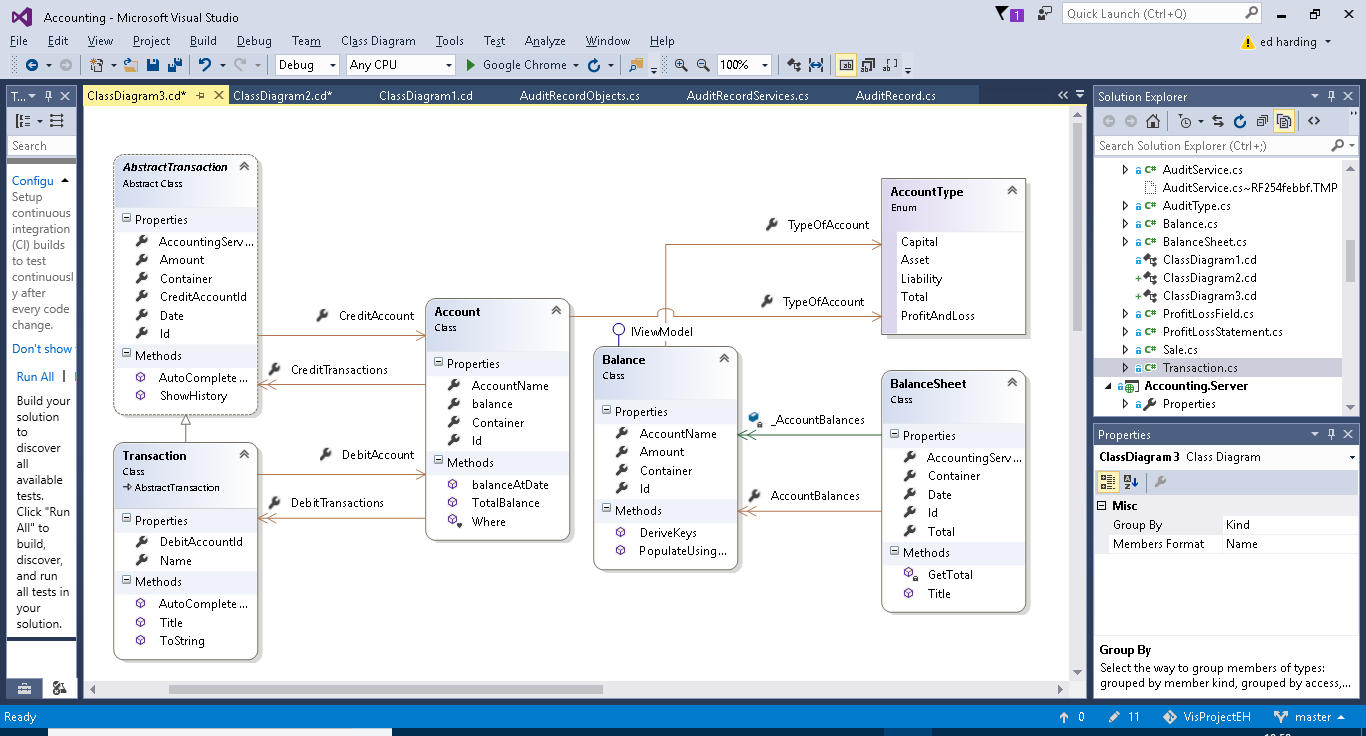
Is used to create the fields used in the profit and loss statement, those being the total value of the stock and the total value for sales made.

Needs renaming to ProfitLossField.

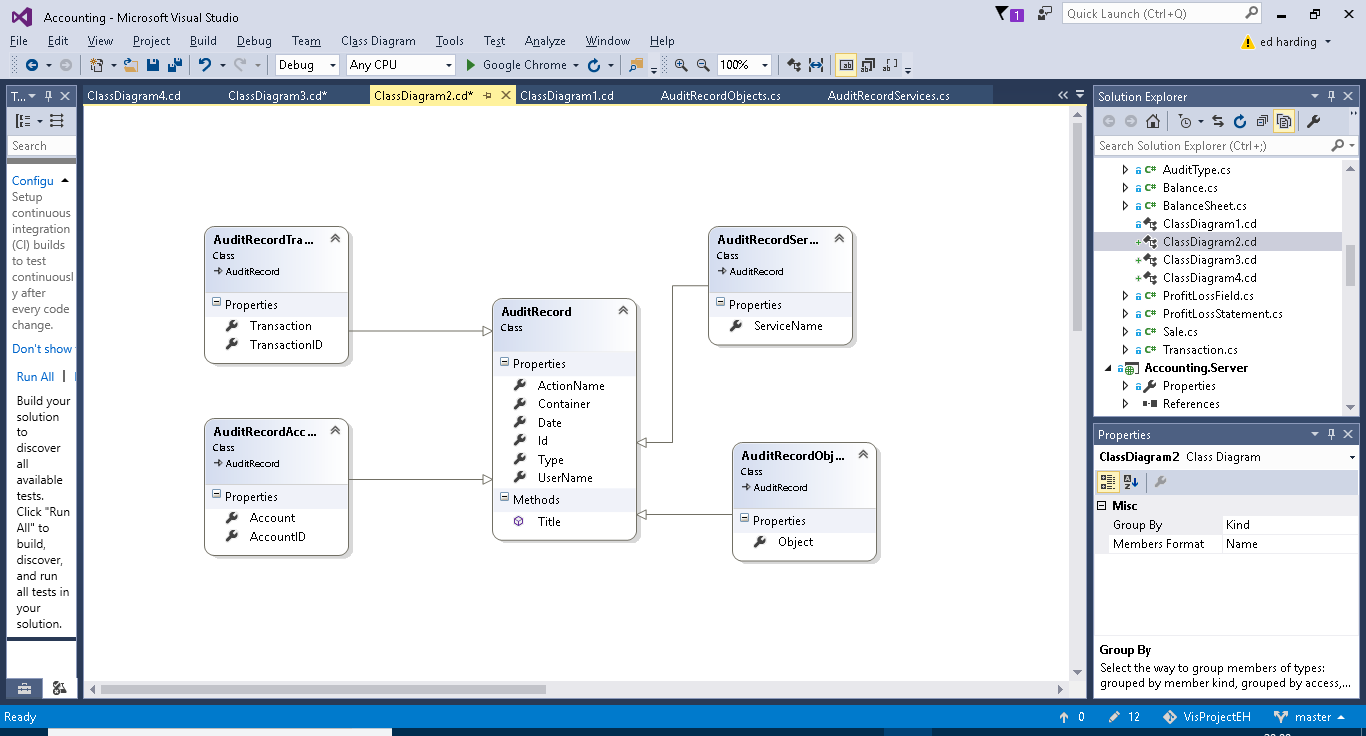
**Profit Loss Statement**

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.

**Additional Class Diagrams**

Below is the class diagram which displays the object associations for the main part of the prgram(that being the transaction and balance sheet functionality)

Below is the class diagram for the classes related to the profit and loss statement functionality of the program.



# Technical Solution

Added iterations, numbered to keep them separate for now. Once they are all fleshed out will combine them together/re-format. Are iterations design or technical solution?

Iterations moved to “Appenix” word document

C# object model, entity framework, SQL Server database, Naked Objects framework, auditing functionality, UI.

Naked Objects Framework allows to use the entity framework and sql server database and creates a ui from them. With services etc.

Also it comes with a built in auditing functionality which I have utilised.

The below (blue) needs revisiting (clinic).

Entity Framework is an integral part of the project as it acts almost like a translator between the business-based objects I have coded and the SQL server database. Entity framework has the ability to format and map user made objects to a database by creating an Entity Data Model from the objects. This therefore reducing a whole work load for the developer. It also allows the developer to perform LINQ queries in C# on the database. This is extremely useful when creating searching methods and such.

I chose to use Naked Objects Framework for multiple reasons. The biggest advantage for using it is its UI. The client side of NOF is brilliant for database orientated projects as the UI will automatically display objects in a tabulated way similar to how it would look in a database. It also separates object methods and adds them to the actions button on the UI. This is all customisable from inside the objects. The UI’s colour and layout can all be customized, this includes individual objects and everything. NOF also contains the ability to set up user profiles using Auth0, this can be done via a developer-built profile system or using something like google accounts this became useful in auditing. Another advantage of NOF is that it comes with a template auditing system. Meaning that the actual integration part of auditing can be set up with relative ease. This is very useful as it all I had to do was create the actual audit recording system, the program would automatically pick up the user’s actions for me. On my end this involved creating the functions to get data from the user’s profile and the action which was performed and manipulate it from there.

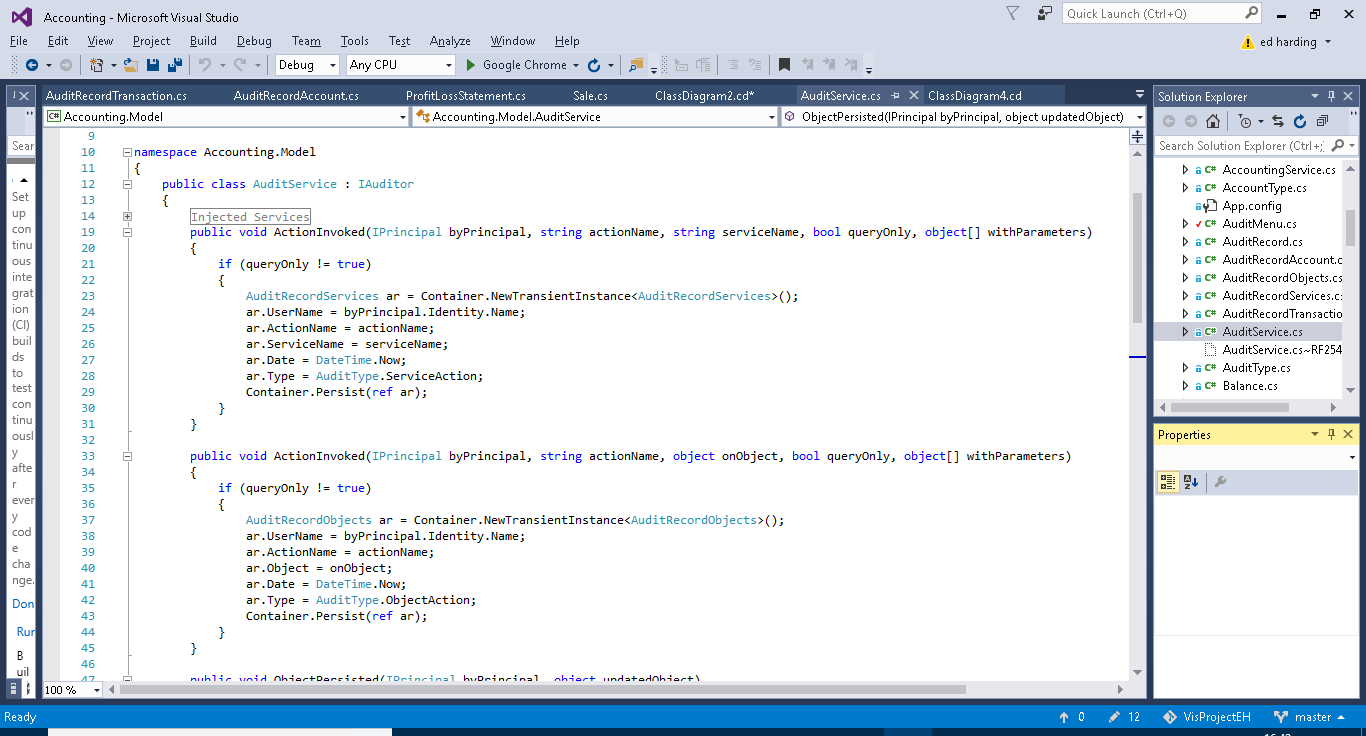
## **Functionality**

### **Auditing**

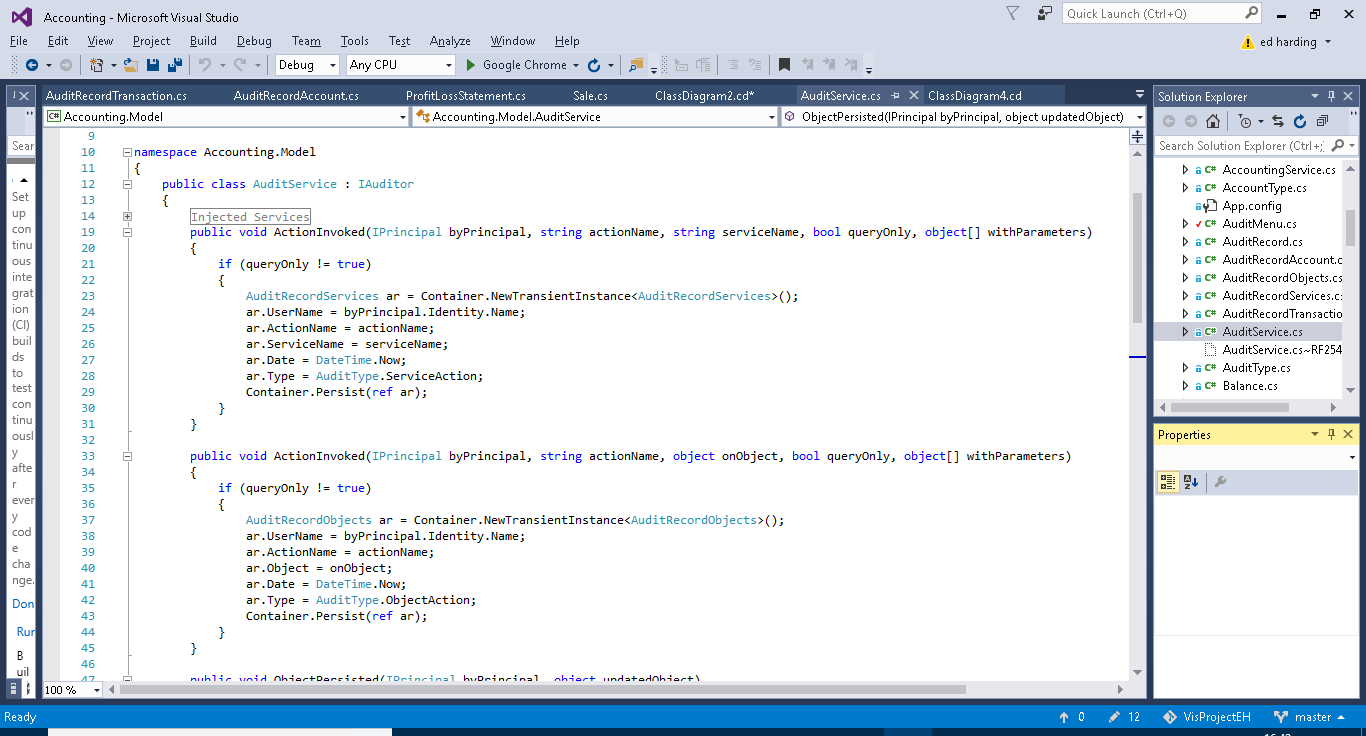
Auditing is one of the more complex capabilities of my program. My auditing code can be split into 3 sections (I will go into more detail of each below). The AuditService which detects when actions have been perfomed by a user, the AuditRecord which is used to save information about the action and user and finally the AuditMenu which is the UI end of the audit capability, it is used to display the audit records to a user.

#### Audit Service

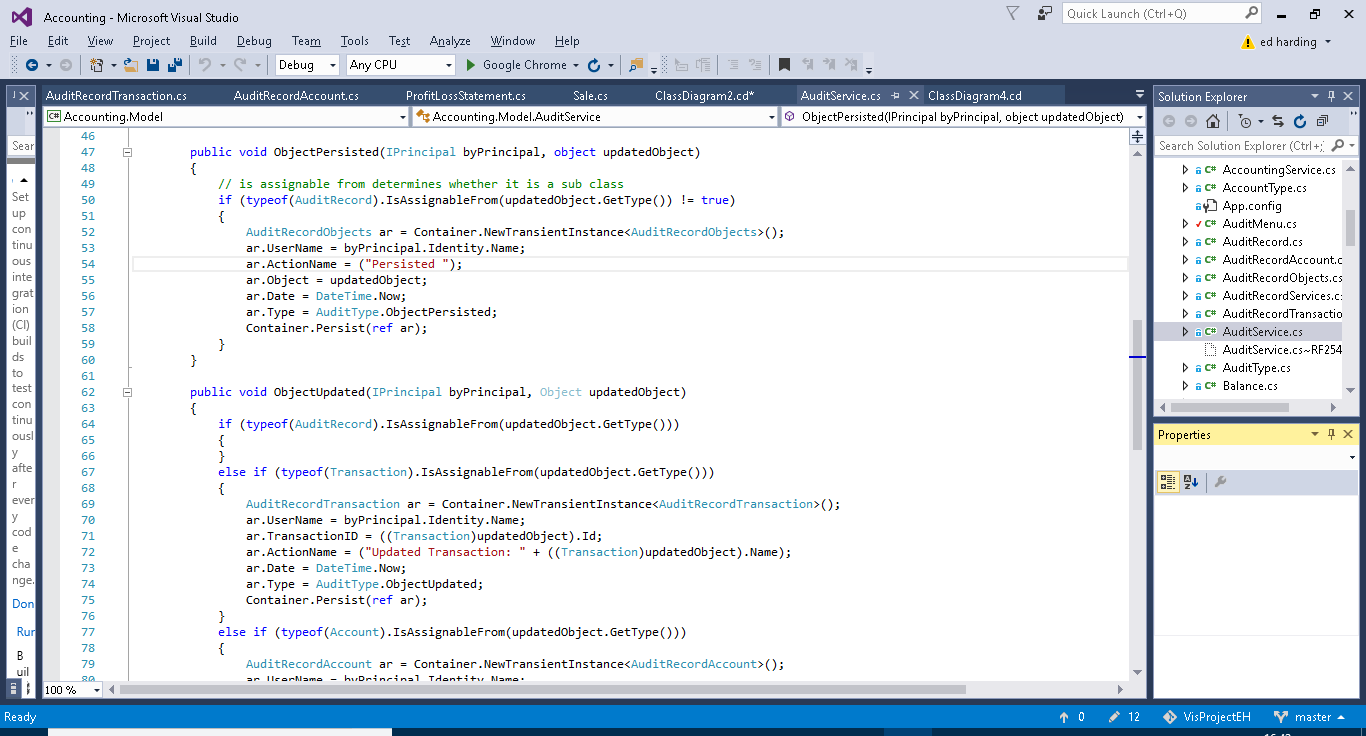
The Audit service constists of 4 methods each will be called under different circumstances.



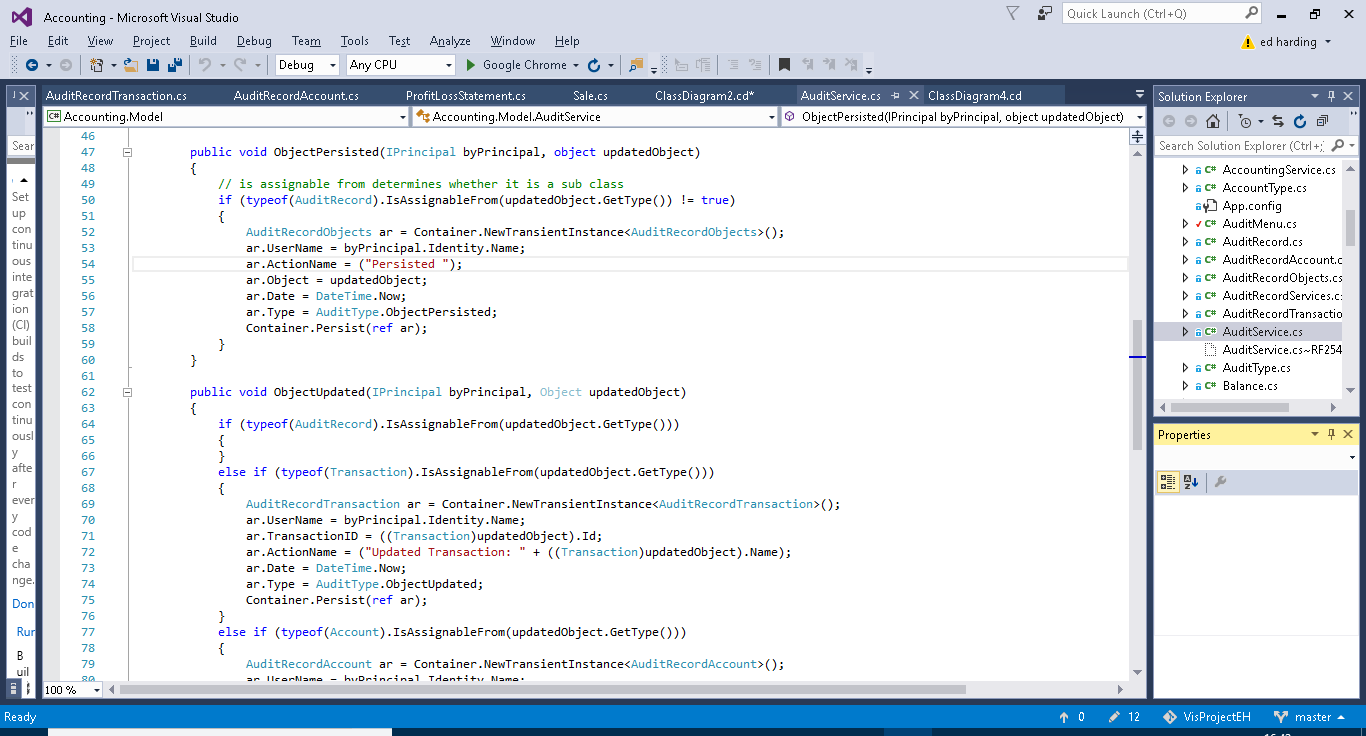
The first method (above) is called when a service action (e.g.) is executed by the user. The if statement checks to see if the action was a query only action, if it is nothing will happen as queries don’t necessarily need to be recorded as nothing critical is happening to the data. If the action isn’t a query action the method will create a new instance of AuditRecordService (AuditRecords are explained further on in this section). It then fills out the fields from AuditRecordService and persists the new instance. The username is obtained from the byPrincipal parameter. The username comes from the Auth0 log in system. AuditType is an enum I created for querying purposes you will be able to see this in use later on.

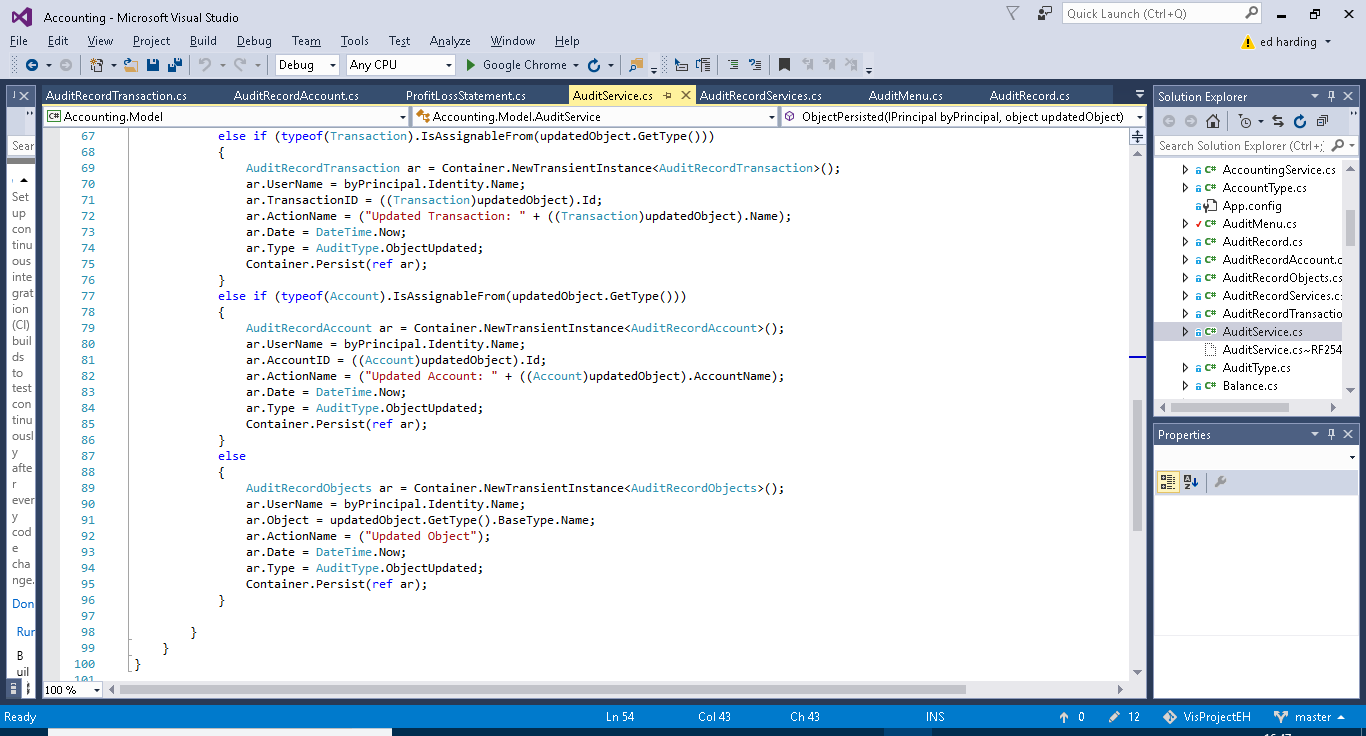


The second method is very similar to the first however it acts upon actions performed on objects (e.g.). Like the first method it checks whether the action is query only or not, it then creates a new instance of AuditRecordObjects, fills out the fields and persists the new instance.



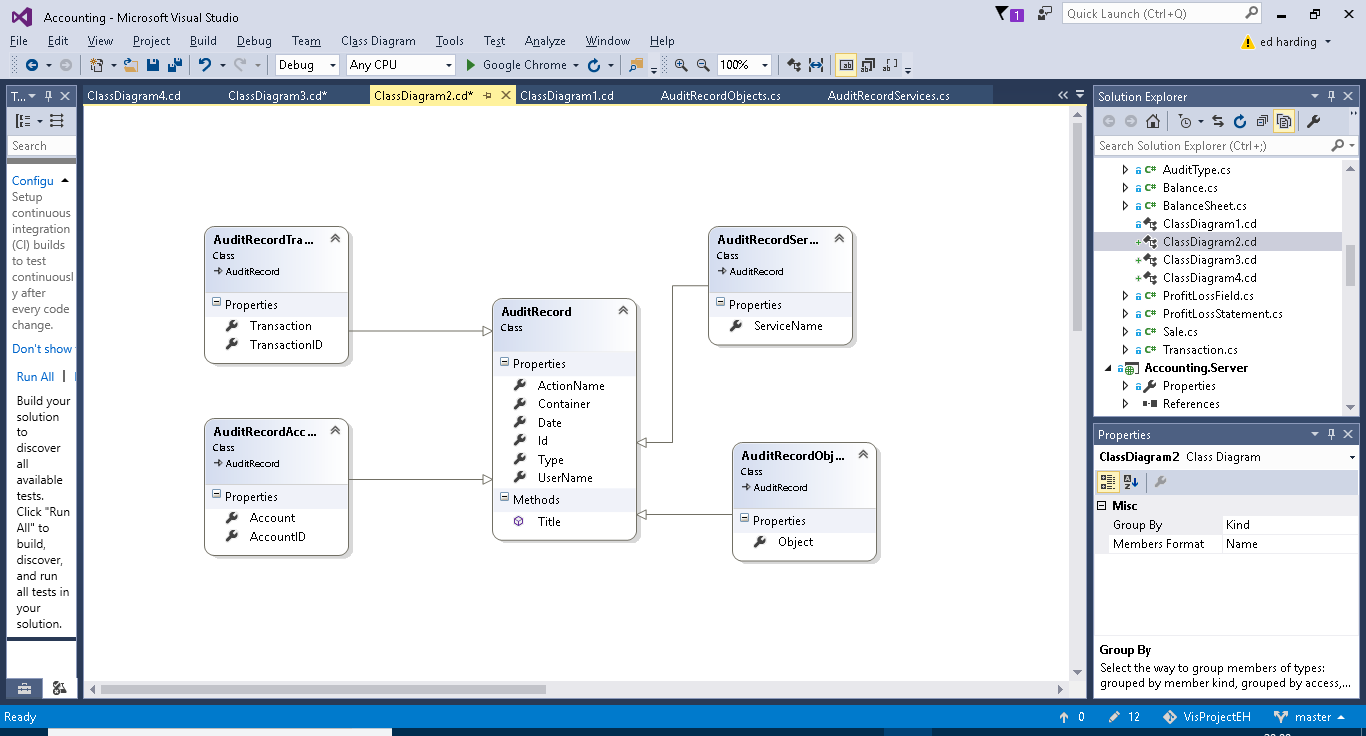
The third method is for auditing persisted objects. The first if statement is very important, its purpose is to assert whether or not the object which has been persisted is a sub class of AuditRecord (e.g. AuditRecordService), if it is then the method does nothing. This is important as without this if statement this method will be called whenever any AuditRecord sub class is peristed such as in the previous methods. As this method itself persists a subclass of AuditRecord it then will get stuck into a loop and a stack overflow error would occur. If the persisted object is not a sub class of Audit Record then the method will create, fill out and persist a new instance of AuditRecordObjects. The action name is not passed in through a parameter as in previous methods and is therefore declared as “persisted” by the method.



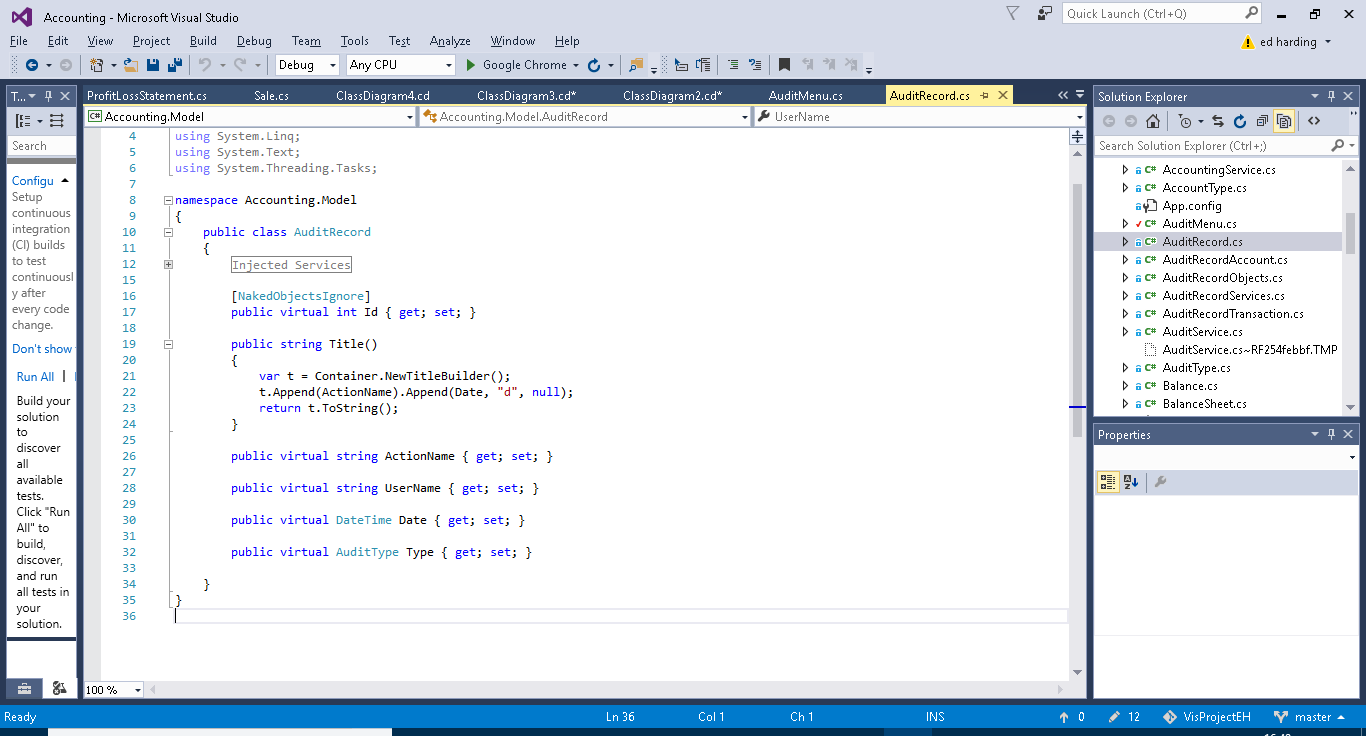


Similiarly to the previous method the first if case checks that the updated object isn’t a sub class of Audit Record, again this is to aoid stack overflow errors as described before. The next three if cases are to establish if the updated object is a transaction, account or other object. This is because transaction and account have their own subclasses of AuditRecord. As before a new instance of the required AuditRecord subclass will be created the fields initialised and then the instance will be persisted. The advantage of having Account and transaction records separate from one another and other objects is so that the name of the account or transaction can be added to the action name for userbility purposes and for querying advantages.

#### Audit Record

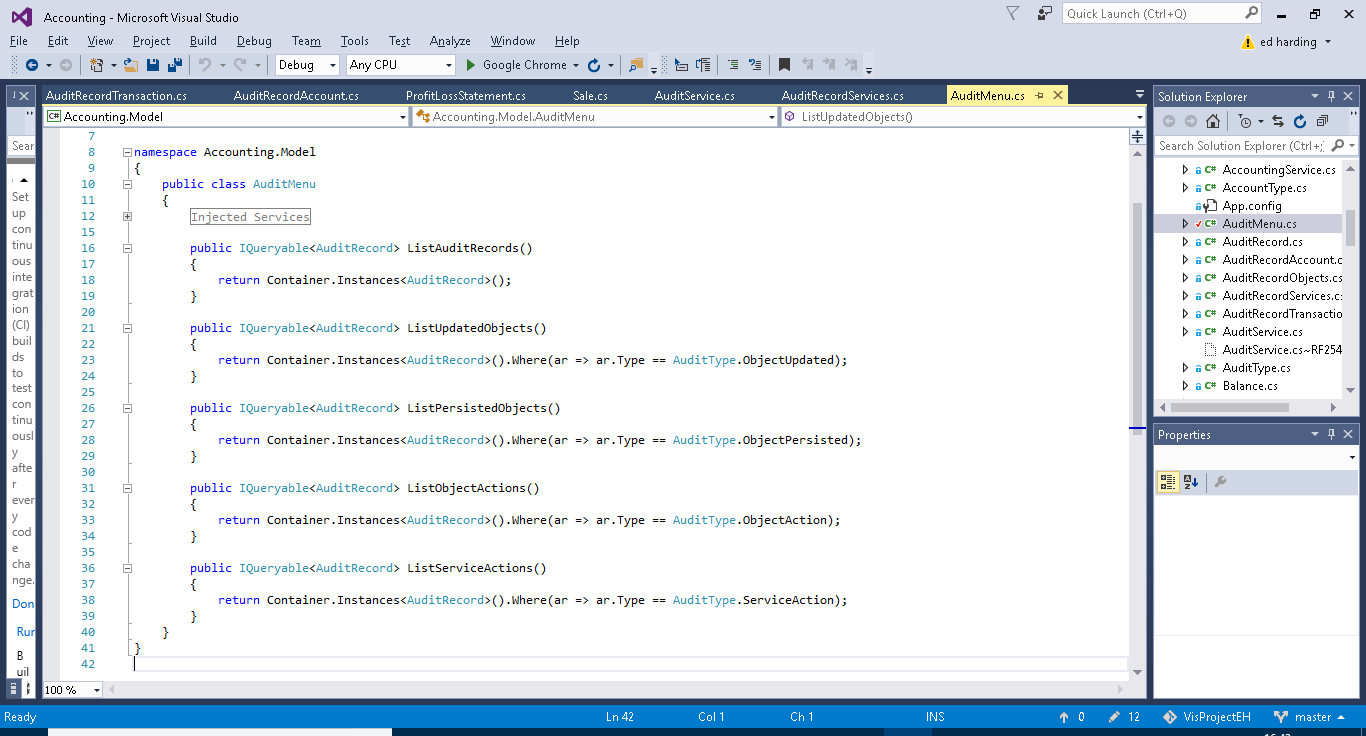


Above is a class diagram showing the four subclasses of account record. These were used in the methods in Audit Record (previous section).



This is the super class Audit Record. It contains the Username ActionName and Date fields, very typical and useful information for someone auditing the system. The AuditType field is for querying purposes. And the Title Builder is for usability purposes i.e. making the name of each instance of AuditRecord more presentable and convenient for the user to find what they need as it combines the date and the action name. The subclasses each have one or two additional fields specific to themselves.

#### Audit Menu



The audit menus sole purpose is to provide pre-built query methods on the entirety of saved AuditRecord. These methods just make it easier for the user to find what they want. The first method just outputs an entire list of all AuditRecord (this being all the instances of the subclasses of AuditRecord). The second method Lists all the instances of AuditRecord which contain information on Updated Objects. The third method Lists all the instances of AuditRecord which contain information on Persisted Objects. The fourth method lists all the instances of AuditRecord which contain information on executed object actions. The fourth method lists all the instances of AuditRecord which contain information on executed service actions.

### **Profit and Loss Statement**

# Testing

* Test Table and results, explain results especially if not expected result.
* Test/demonstration video, should start scripting soon, will help pick up on anything I need to fix or add to the program.

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Input | Expected Outcome | Actual Outcome |
| Create and save an account. |  |  |  |
| Create and save a Transaction |  |  |  |
| Create a balance sheet. |  |  |  |
| Etc. |  |  |  |

# Evaluation

* Did it meet end user requirements.
* What I could do better/improve with more time.
* What it should have if it was a complete business level program.