# Assignment 4 – OOP: Encapsulated Class

~~This is an exercise in file I/O management through an encapsulated class, not an exercise in List or Array management. If you declare a collection such as an array or List outside of a method or as a property, I will delete the object before I start marking. You have to think of a multi-user environment: if you retain your data in memory and just dump it to file periodically, then you will obliterate data added or updated by others … and they will do the same to you.~~

~~An~~ ***~~encapsulated~~*** ~~class contains both the properties and the methods required to instantiate and manage objects of the class. The class you create will save and retrieve objects in a text file. It will not store things in memory. Later, you can replace it with one that uses a relational database table. As long as the class name and its~~ ***~~method signatures~~*** ~~are the same, applications will run & not know the storage mechanism has been replaced.~~

~~This is~~ ***~~abstraction~~*** ~~or~~ ***~~decoupling~~*** ~~… applications become rigid and fragile when they talk directly to (aka~~ *~~tightly coupled~~* ~~to) a specific data storage mechanism.~~

~~Where you see~~ ***~~XX~~***~~, insert your initials instead.~~

1. ~~Create a new Windows project or continue with your MDI main page and menu.~~
2. ~~Add a folder to your project called~~ ***~~XXClasses~~***~~. If you prefer, you can use a separate~~ ***~~XXClassLibrary~~*** ~~project.~~

## XXStock class

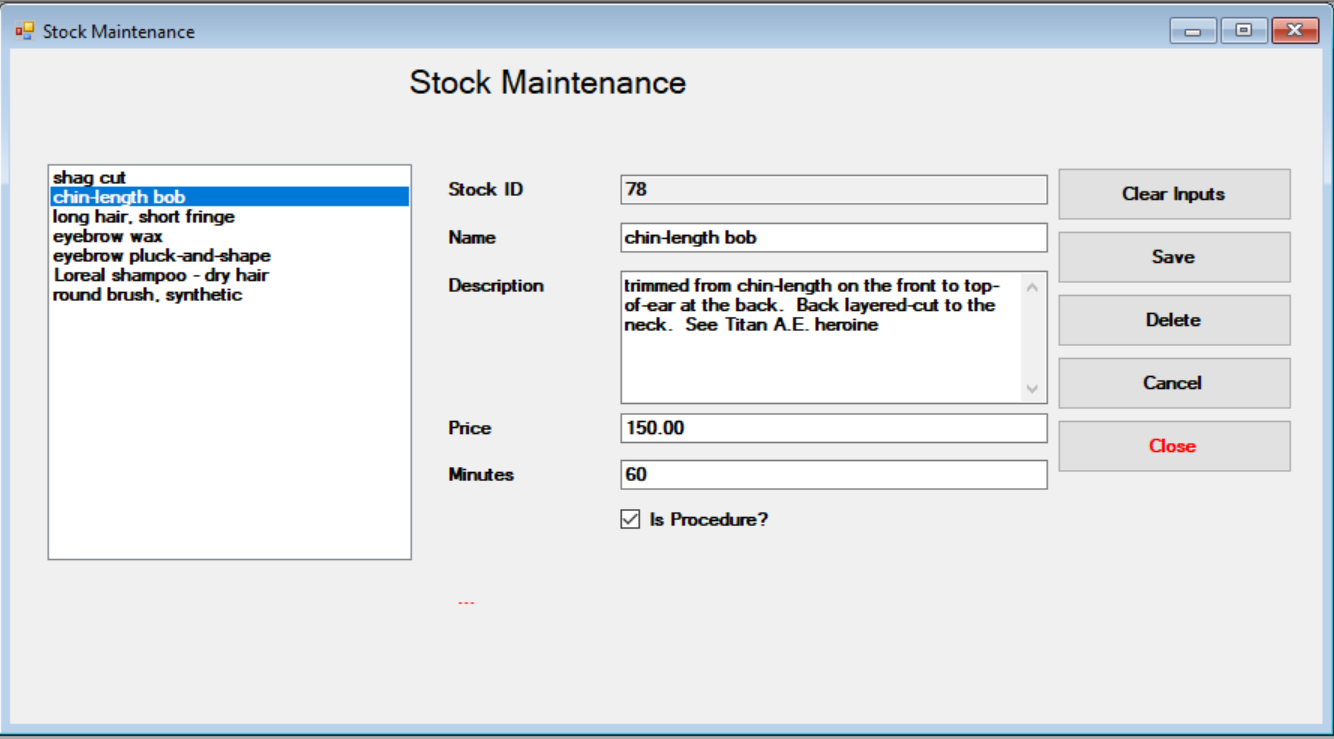
~~This class manages stock items in a beauty shop. Some are procedures, like eyebrow waxing, while others are products, like brushes or conditioners. Each item has a stock price, but if it is a procedure, it also has the number of minutes it will take (so they can tell customers how long the appointment will be).~~

~~Add a class called~~ ***~~XXStock~~*** ~~to your XXClasses folder. The following properties and methods are public instance methods unless stated otherwise:~~

1. ~~Add the following properties:~~
   1. ~~Integer~~ ***~~StockId~~***
   2. ~~Strings~~ ***~~Name~~*** ~~and~~ ***~~Description~~***
   3. ~~Double~~ ***~~Price~~***
   4. ~~Integer~~ ***~~Minutes~~***
   5. ~~Boolean~~ ***~~IsProcedure~~***
2. ~~Add a global class-level string variable called~~ *~~stockPath~~* ~~and initialise it to “Stock.txt”.~~ ~~Where will this file end up being stored?~~
3. ~~Override the~~ ***~~ToString()~~*** ~~method that your class inherited from Object:~~
   1. ~~Return a single string containing all the object’s properties, delineated so the string can be parsed back into an XXStock object later~~.
4. ~~Add a class (static) method called~~ ***~~XXParse~~*** ~~that accepts a string parameter and returns an object of the XXStock class. The string is a record from the stock file containing a stock object’s properties.~~
5. ~~Add the following~~ *~~class~~* ~~“Get” methods, to retrieve XXStock object(s) from the file:~~
   1. ***~~XXGetStocks~~*** ~~has no parameters and returns a List<XXStock> containing all stocks on file.~~
   2. ***~~XXGetByStockId~~*** ~~accepts an Integer parameter and returns the XXStock object with a~~ *~~StockId~~* ~~equal to the given integer … null if not found.~~
   3. ***~~XXGetByDescription~~*** ~~accepts a string parameter and returns a List<XXStock> of all Stocks containing the given string phrase in either their Name or Description. The list can be empty if none are found.~~
6. ~~Add an instance method called~~ ***~~XXEdit~~*** ~~that performs the following validations on the current object (ie: the method has no parameters). If there are any error or exception messages, throw them all at once in an exception.~~
   1. ~~Initialize an empty string to collect all the error messages. Each error message should include the property name in the message and end with a new-line character~~, ~~so errors display one line per error.~~
   2. ~~Trim all strings, converting nulls into an empty string.~~
   3. ~~If the~~ *~~StockId~~* ~~is zero, then you’re adding a new record … otherwise, it’s an update and the given~~ *~~StockId~~* ~~must be on file. The Add method in the class will assign a unique value.~~
   4. *~~Name~~* ~~and~~ *~~Description~~* ~~are required, and cannot be just spaces~~ … please don’t edit their contents.
   5. *~~Name~~* ~~must be unique on file. The user cannot add a duplicate name,~~ ~~nor can they update an existing record to create a duplicate. They should be able to update a record without having to change its name~~.
   6. *~~Price~~* ~~and~~ *~~Minutes~~* ~~cannot be less than zero.~~
   7. ~~If it is a procedure,~~ *~~Minutes~~* ~~must be greater then zero. Otherwise,~~ *~~Minutes~~* ~~must be zero.~~
7. ~~Add an instance method called~~ ***~~XXAdd~~*** ~~that accepts an XXStock object and, if it passes all edits:~~ 
   1. ~~It assigns a unique StockId to the object and adds the record to the file.~~
      1. ~~Since this is a reference variable, it’s a pass-by-reference, so you should be able to use its stockId in the form~~
8. ~~Add an instance method called~~ ***~~XXUpdate~~*** ~~that accepts an XXStock object and replaces the current record with the same StockId, if the updated record passes the edits.~~
9. ~~Add a class method called~~ ***~~XXDelete~~*** ~~that accepts a string and removes the record with the given StockId from the file.~~

## XXStockMaintenance

1. ~~Create the following form to perform Create, Retrieve, Update and Delete (CRUD) maintenance on the Stock table, through your XXStock class.~~ 
   1. ~~Don’t bother with browse/modify modes … you have enough work.~~
   2. ~~The stockId is read-only: Add creates a unique one and Update does not permit the user to change it.~~
2. ~~The~~ ***~~ListBox~~*** ~~on the left is used to navigate through the Stock records:~~
   1. ~~It is loaded with all stock records on file … it displays the StockName and returns the StockId as the ListBox’s value.~~ ~~The Stock names are displayed in alphabetical order~~.
   2. ~~When the user selects a Stock, that Stock’s properties are loaded to the input areas.~~
   3. ~~When the user successfully adds a new Stock or updates an existing Stock:~~ 
      1. ~~The ListBox needs to be reloaded and resorted, and the modified/new record selected.~~
   4. ~~When the user deletes an existing record:~~
      1. ~~The ListBox needs to be reloaded, and have the record after the one deleted selected (or the last record, if the user deleted the prior last record).~~
   5. ~~When a record is selected in the ListBox, this should shout “this is an update!”. Hopefully someone’s listening.~~
3. ~~The~~ ***~~Clear Inputs~~*** ~~button does exactly that, readying the input areas for a new record. It also signals “this is a new record”. StockId should be set to zero for a new item.~~
4. ~~The~~ ***~~Save~~*** ~~button creates a new XXStock object, loads it from the input areas and passes it to the object’s Add or Update method.~~
   1. ~~I recommend you check for numbers and Booleans before stuffing them into the object’s properties.~~
   2. ~~Error messages and Exceptions are displayed in a red label or a rich text box, one line per message.~~
5. ~~The~~ ***~~Cancel~~*** ~~button returns the input areas to their values before the user cleared or modified them. What remembers the original StockId, whether it was an update or add that was cancelled?~~



Stock id -> locked when clear inputs clicked, it goes back to 0 or empty string (and it means add)

