**Name: Gary Doherty**

**Student Number: 10366827**

**Module Code: B8IT117**

**Lecturer: Paul Laird**

Library Management System

Documentation

# Standard Requirements

I believe I met all the standard requirements set out in the assignment specification i.e. College, Lecturer, Student, etc.

It also has many overridden methods, multiple overridden ToString() methods, and many overloaded methods.

# Innovation

**GUI:**

I gave my implementation of the *Library Management System* a Windows Form GUI. This took a bit of work, but in the end I’m much happier I did it this way than having a plain console input/ouput alternative. P.s. double-click works on most things in list boxes (except for person’s borrowed books). You can double click on “Overdue Patron” in bottom right list to display particular overdue patron’s information.

**JSON:**

I used a site called JSON-Generator to generate lots of test data for the people and books, and implemented a static class called JsonHelper to parse this data into my objects and fill up the data lists in my program. This could easily be enhanced to store / retrieve persistent data, or read in real functional data from an API.

**Responsive GUI:**

I made my GUI responsive by using interfaces to communicate between the parent and child forms, by sending simple update signals once an action took place in the child, allowing for consistent data display throughout the application.

**Validated input:**

Validate a lot of the input coming in ( not address, name, or phone number ).

**Overdue Book Loans:**

My application tracks due dates, and will display a list of all patrons with overdue books at runtime. While the debug variable is set in College, for testing purposes, the loan duration will become a random time between 10-20 seconds, and as soon as the book becomes overdue, the patron’s name will appear in the Overdue Patrons box on the entry form. I did this by using a Timer thread to check books every second. I could have done this in the BorrowedBooks themselves, and have it update the Form, but didn’t have the time to sort that out.

**College Singleton:**

For this system it made sense to me to make College a singleton object, as there shouldn’t be multiple Colleges for the designated requirements.

**Library Interface:**

I thought it made sense that the college could have multiple Libraries if it scaled, so I made Library an interface. My Borrowed books hold a variable for the Library interface they were borrowed from ( which made it simple to return books ). To scale would only require minor changes to some of Patron’s methods (add more ILibrary args).

**LibraryPatron abstract class:**

I implemented an abstract LibraryPatron class, that inherits from Person, which my Lecturer and Student classes inherit from. This made it easy to work in conjunction with the Library.

**Book class and LoanedBook:Book class**

I made a book class to track information about each book, and a LoanedBook class to keep track of LoanDuration and a clean ability to return a book. The LoanedBook class is used as a list within my LibraryPatron class.

**Enum Student Status:**

It seemed pointless implementing an Undergrad and Postgrad class inheriting from Student, as all that was needed was an int. Thus I used an *Enum GradStatus { Undergrad=5, Postgrad=10 } .* This simply took care of the student book limit problem; presenting their status as a string, or their loan limit as an int in one variable (Status).

# Testing

Functions tested include:

* The *RequestLoan* function of the Library class.
* The *ReturnBook* function of the LibraryPatron class.
* The *ReachedBookLimit* function of the LibraryPatron class (Undergrad, Postgrad, Lecturer)
* The *PeopleWith* function of the Library class, allowing to check what patrons have a certain book.
* And finally the *IsBookAvailable* function of the Library class, checking whether a certain book is available to borrow (whether it’s not taken already, or simply doesn’t exist in the library inventory).

All tests passed with no problem.

# Conclusion

Overall I’m pleased with the results of my application. There’s a lot more I’d love to do with it, but I think enough is enough. I’m very happy with the GUI; again there was a lot more I wanted to put in, but I just didn’t have the time.

I think I covered almost everything I set out to implement, and more, so I am happy with what I accomplished, and I learnt a lot.