Bradley De’Ath | DEA18001355

SCDT41 Programming And SOftware FUndamentals: Assignment 1 PORTFOLIO

Table of Contents

[Task 1: The Rail System 2](#_Toc30486369)

[Introduction 2](#_Toc30486370)

[Requirements 2](#_Toc30486371)

[Extra Functionality 2](#_Toc30486372)

[The Structure 2](#_Toc30486373)

[Testing 3](#_Toc30486374)

[Task 2: The Spelling Quiz 4](#_Toc30486375)

[Introduction 4](#_Toc30486376)

[Requirements 4](#_Toc30486377)

[Extra Functionality 4](#_Toc30486378)

[The Structure 5](#_Toc30486379)

[Testing 5](#_Toc30486380)

[Task 3: Office Login System 7](#_Toc30486381)

[Introduction 7](#_Toc30486382)

[Requirements 7](#_Toc30486383)

[Extra Functionality 8](#_Toc30486384)

[The Structure 8](#_Toc30486385)

[Testing 9](#_Toc30486386)

[Task 4: OOP Library System 10](#_Toc30486387)

[Introduction 10](#_Toc30486388)

[Requirements 10](#_Toc30486389)

[Extra Functionality 10](#_Toc30486390)

[The Structure 11](#_Toc30486391)

[Testing 12](#_Toc30486392)

[Raw Code 16](#_Toc30486393)

[Task 1 16](#_Toc30486394)

[Task 2 18](#_Toc30486395)

[Task 3 27](#_Toc30486396)

[Task 4 37](#_Toc30486397)

For this assignment I was tasked to create a selection of software and systems for a variety of scenarios. This portfolio will go over each of the systems I have created as well as any important information about them.

# Task 1: The Rail System

## Introduction

The first task of this assignment was to develop a software to allow staff members of a train station to manage how many customers are booked to ride on the trains that are stopping there. The types of customers that are being entered into the system are split into two categories: those who had pre-booked their tickets before the journey, and those who paid upon arrival at the station. The requirements laid out by the station can be seen below:

## Requirements

|  |  |
| --- | --- |
| **Requirement** | **Description** |
| Records Passengers | The total number of passengers per coach for a journey need to be recorded |
| Pre-Booked Passengers | The number of passengers that paid Pre-Booked per coach needs to be recorded |
| Pay-on-arrival Passengers | The number of passengers that paid on Arrival per coach needs to be recorded |
| Sum of Values | The total number of passengers per coach needs to be calculated |
| Ascending Order | The total number of passengers per coach need to be displayed in order of smallest to largest |

## Extra Functionality

|  |  |
| --- | --- |
| **Additional Feature** | **Description** |
| Login System | Rather than just contain the system for calculating passengers, the code is locked behind a login system to improve security for the train station. |

## The Structure

The main body of this software was designed using “foreach” loops as well as arrays. The data relating to customer counts and the carriage designations are stored in arrays. This is done so that the appropriate information is grouped together and has set sizes – ensuring the proper size is ensured is important for this scenario as otherwise, data relating to a sixth carriage on the train could be entered, even though there may not be one present. For each carriage on the train, the user can enter the customers for each category of booking – and with the “foreach” loops, this is repeated until the total number of carriages have been cycled through.

Once all booking has been entered, the totals for each of the carriages are calculated through taking the sum of the customers from each category. Once the totals have been calculated, each carriage is outputted to the Command Line Interface in ascending order based on the total number of passengers booked.

## Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Test** | **Description** | **Outcome** | **Screenshots** |
| 1 | Login Verification | Entered login data is checked against those stored in the system | Success |  |
| 2 | Pre-Booked Passengers | The user is able to enter the total of Pre-booked passengers for each Coach | Success |  |
| 3 | On-Arrival Passengers | The user is able to enter the total of Pay-on-arrival passengers for each Coach | Success |  |
| 4 | Passenger Totals | The total Passengers for each coach are calculated | Success |  |
| 5 | Each Carriage Used | The software asks the user for inputs until each coach has the needed data | Success | N/A |
| 6 | Carriage Ordered Output | The totals are displayed for each coach, in order from smallest to largest | Success |  |

# Task 2: The Spelling Quiz

## Introduction

The second scenario that I was tasked with designing a program for was the creation of a spelling quiz software for use with school children. The quiz asks the user a series of questions where the answers are commonly misspelt words – requiring the user to spell the word correctly in order to get the correct answer to count. The requirements for this software can be seen below:

## Requirements

|  |  |
| --- | --- |
| **Requirement** | **Description** |
| Outputs Questions | The questions in the quiz are outputted to the interface for the student to answer |
| Checks Student Input | The student answers the question and the correctness of their answer is checked |
| 10 points for correct spelling | If the student is correct, 10 points is added to their total score |
| Point deduction | For each incorrect letter in the student’s answer, a point is taken off their total score |
| Percentage result | At the end of the quiz, the percentage of the student’s score out of the maximum possible is displayed |

## Extra Functionality

|  |  |
| --- | --- |
| **Additional Feature** | **Description** |
| Login System | To access the system, a recognised login (username and password) must be entered and checked |
| View Users | A list of all students can be viewed by Teachers |
| Add Users | New student users can be added to the login system for the software by Teachers |
| Remove Users | Student logins can be removed from the login system for the software by Teachers |
| Teacher Users | Similar to an admin role, teacher logins can manage aspects of the software such as the students and the questions |
| View all Questions | Teachers are able to view a list of all the questions & answers from the quiz |
| Add Questions | Teachers are able to add additional questions to the quiz |
| Remove Questions | Teachers are able to remove questions from the quiz |
| Hints System | Students are able to request a “hint” from the software for a question, where they are given the first character of the answer at the cost of losing a point from their total score |
| Minimum similarity Score | Expanding on the system of losing a point for every incorrect letter in a student’s answer, if they have guessed less than half of the letters correctly then they earn a score of 0 for that question |

## The Structure

The structure of this program is split across a total 5 classes, based on the tasks they perform. The first of these classes is the main ‘program’,. This class is simply used for the main login into the system. As a part of the extra features I have included - the user can login to the system whether they are a teacher or a student. Once the entered login information has been marked correct as it matches a login stored in the system; the program refers the user to the correct class to continue with the software.

The next of the classes is the ‘School Logins’ class; the purpose of this class is the storage and management of the logins used by the system. This class is used when checking the entered login information from the program class and returning whether it matches or not. As well as the storage of the logins, once called by the ‘Teacher Menu’ class, the teacher can add or remove student logins from the system.

The previously mentioned ‘Teacher Menu’ is the class used by those who have logged into the system with a level above student. One using this class, the teachers are able to manage the data stored within the ‘School Logins’ class to view, add and delete the students stored there. As well as this, the data stored within the ‘Spelling List’ class can also be manged, to view, add and delete the questions that are asked to the students.

The ‘Student Menu’ class is where the actual quiz takes place. Once a user with the rank of student has logged into the software, this class is loaded and allows the user to answer the questions. Once all of the questions stored within the ‘Spelling List’ class has been answered, the users score, and percentage is outputted to the screen.

The final class, the ‘Spelling List’ class, is where the questions asked within the quiz are stored and managed. Every time the student has entered their answer to a question – it is compared against the answer stored within the system – with a point being taken off of the maximum score that can be earned from that word. If less than half of the letters entered by the student are correct, the user earns a score of 0 for their answer. On top of the questions stored there, this class is also used to generate the ‘hints’ for each question. The hints are one of the extra functionality parts of the software, where the student can request the first letter of the answer to the question at the cost of one point being taken off their total.

## Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Test** | **Description** | **Outcome** | **Screenshots** |
| 1 | Login Verification | Entered login data is checked against those stored in the system | Success |  |
| 2 | Teacher Menu loads | If a teacher login is entered, the Teacher Menu class is called | Success |  |
| 3 | Student Menu Loads | If a student login is entered, the Student Menu class is called | Success |  |
| 4 | Can add new student | A teacher is able to add a new student login to the system | Success |  |
| 5 | Can view all students | A teacher is able to view a list of all student accounts on the system | Success |  |
| 6 | Can remove a student | A teacher is able to delete a student account from the system | Success |  |
| 7 | Can add a new Question | A teacher is able to add a new question to the quiz | Success |  |
| 8 | Can view all questions | A teacher is able to view all of the questions in the quiz | Success |  |
| 9 | Can delete a question | A teacher is able to delete a question from the quiz | Success |  |
| 10 | The quiz works | Students are able to take the quiz | Success |  |
| 11 | Hints can be requested | Students are able to request a hint for a question on the quiz | Success |  |
| 12 | Scores are calculated | Once the quiz is complete, the student’s score is outputted to the interface | Success |  |
| 13 | Percentages are calculated | The percentage conversion of the student’s score is displayed to the interface | Success |  |

# Task 3: Office Login System

## Introduction

The third task of this assignment was to develop a login management system for use within an organisation. The users need to be able to login to the system using a correct username and password, whereas if a correct combination is used three times in a row: the software is locked and will need to be unlocked by a user with admin privileges. Once the user has successfully logged in, a welcome message needs to be displayed to the user with their correct first and last names.

## Requirements

|  |  |
| --- | --- |
| **Requirement** | **Description** |
| Allows Username & Password Input | The system must allow for the user to input both a username and password into the interface |
| Only allow access to certain users | Only allow access to the software if the entered username and password combination match those stored within the system |
| Output error messages | If any incorrect information is entered, an appropriate error message must be displayed to the user |
| 3 login attempts before lockout | If the incorrect login is entered 3 times sequentially then the user is locked out of the system until the system is restarted or they are unlocked by an admin |
| Login attempts display | The total number of login attempts remaining should be displayed to the user |
| Welcome message | Once the login is successful, the user should be greeted with a welcome message containing their First and Last name |

## Extra Functionality

|  |  |
| --- | --- |
| **Additional Feature** | **Description** |
| View all Users | Admin users are able to view a list of all registered users on the system |
| Add a new User | Admins are able to add new users to the system, identifying: First Name, Surname, Username, Password, whether they are an Admin |
| Remove a User | Admins are able to remove existing users from the system |
| Password Change | Users are able to change their own password once logged in |

## The Structure

This software is made up of a total of 5 classes, with each containing methods that relate to their purpose. The first of these classes is the main ‘program’ class and is used for the main initial login into the system. Once the user has logged in and the login details have been checked against those stored in the system, the user’s name is displayed within a welcome message - the employee is then directed to the correct class for their login type. If the user enters an incorrect login three times in a row, then the system becomes locked, and nothing else can loaded until the login details of a user with admin privileges is entered. This is done through the ‘Admin Override’ class, where the option to enter admin details is run until a correct login is entered.

If an admin user logs into the system, then they are loaded into the ‘Admin User’ class, which contains all the necessary methods to manage the other users and systems on the software. From this class, all employees stored in the ‘User Logins’ class can be viewed in a display to the CLI. As well as this, users can be both added and deleted to the system – along with any important information relating to them such as their names and whether they have access to admin privileges.

If a regular employee logs into the system, then they are loaded into the ‘User’ class, which contains the options needed by regular employees using the system. From this class the user is able to change their password that is associated with the account. This was added as part of my extra functionality, and once the new password has been entered and confirmed (it must be entered twice and both must match), the password associated with the logged in account is changed in the ‘User Logins’ class.

The previously mentioned ‘User Logins’ class is where all of the data relating to employee logins are stored, as well as where the different methods to edit said data run from. The dictionaries and arrays within this class are used to store the following important information relating to users on the system:

* Account Username
* Account Password
* Employee First Name
* Employee Surname
* Employee Privileges (Whether they are a regular user or an admin)

## Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Test** | **Description** | **Outcome** | **Screenshots** |
| 1 | Login Verification | The entered login information is checked against those stored in the system | Success |  |
| 2 | Welcome Message | Upon login, the user’s name is displayed within a welcome message | Success |  |
| 3 | Login Attempts | The user has three attempts to login before an admin is needed – these attempts are listed | Success |  |
| 4 | Admin Unlock | The admin is able to override the locked software | Success |  |
| 5 | Admin Menu loads | Once an admin logs in, the admin menu is loaded | Success |  |
| 6 | Employee Menu loads | Once a regular user logs in, the user menu is loaded | Success |  |
| 7 | View all users | The admin is able to view an output of all users and their details | Success |  |
| 8 | Add new user | The admin is able to add a new user to the system | Success |  |
| 9 | Delete a user | The admin is able to delete a user from the system | Success |  |
| 10 | Password Change | The user is able to change their own password | Success |  |

# Task 4: OOP Library System

## Introduction

The scenario for this task of the assignment was to create a system that allows for the management of the operations that go on within a library business. The first of these is for the user to be able to view existing, and add additional, customers to the library system. As well as the customers, both loans and books need to be managed in the same way. These requirements can be seen below:

## Requirements

|  |  |
| --- | --- |
| **Requirement** | **Description** |
| Add and Retrieve Customer Details | The admin needs to be able to access and add to the assortment of customer details stored in the system |
| Add and Retrieve Book Details | The admin needs to be able to access and add to the assortment of customer details stored in the system |
| Loan Books to Customers | The admin needs to be able to create new loans to loan out books to customers |
| Retrieve Loan Details | The admin needs to be able to view all active loans in the system |

## Extra Functionality

|  |  |
| --- | --- |
| **Additional Feature** | **Description** |
| Remove Books | The Admin has the ability to remove books from the Library System |
| Remove Customers | The Admin has the ability to remove customers from the Library Systems |
| Remove Loans | The Admin has the ability to manually remove loans from the system |
| Loan Status | When displaying all books, the Admin can see whether the books is currently loaned out to a customer |
| Return Date | When a book is loaned out to a customer, a date for it to be returned is generated. |
| Unique IDs | Each Loan and Customer have their own 5-digit unique ID assigned at creation |

## The Structure

The structure of this software is made up of a total of 6 classes, with each containing the appropriate data and methods relating to their purposes. The first of these classes, the main ‘program’ class, is used for the initial login system. As part of my extra functionality for this system, I added the ability for logins to be saved and used to gain access to the methods within – adding a layer of security for the library. Once the entered login information is found to match those stored in the ‘Login Details’ class, a welcome message containing the employee’s name is displayed and the user is redirected to the ‘Admin Control’ class.

This ‘Admin Control’ class is the main menu for allowing the user to manage the different aspects of the library system. From this menu the user has the option to do load the menus relating to the following aspects:

* Customers
* Books
* Loans
* Logins

The ‘Customer Details’ class is where all of the data relating to customers are stored as well as the options for employees to manage it. From this class, a list of all current customers can be displayed to the interface – along with the options to delete them or add additional. The data stored in this class is the information that is entered when registering a customer for the first time, these are:

* Customer First Name
* Customer Surname
* Customer’s Data of Birth
* Join Year (current year is added to show when customer was registered)

The ‘Book Details’ class, similar to the ‘Customer Details’ class, is where all of the data relating to customers are stored as well as the options for employees to manage it. From this class, a list of all books available in the library are stored, as well as whether it has already been loaned out to a customer. The admin has the option to output all of the books to the interface – along with the options to delete them or add additional. The data that is displayed is as follows:

* Book Name
* Author
* Publication Year
* Whether it currently loaned

For the ‘Login Details’ class, all information and functions relating to user accounts are managed here. The login information that is checked against upon log in is stored here, and the admin has the option to add or remove users that are stored here. However, unlike with the other tasks in this assignment, all of the users on the system have the level of admin, and so all would have the option to master delete other accounts. To help maintain security and integrity with this software, both the username and password are needed for an account to be removed.

The final class, the ‘Loan Details’ class, is arguably the most complicated of the classes. It contains the data and methods relating to the loans managed by the library, the data that is stored within the loans are as follows:

* The Book being loaned
* The first name of the customer taking out the loan
* The surname of the customer taking out the loan
* The employee that is authorising the loan
* The date of the loan
* The date the loan needs to be returned
* The newly generated reference ID for the loan

The information relating to the customers and books for the loans are pulled directly from the classes that store them, with the return date of the loan being generated by adding 3 weeks to the current date. Along with adding these new loans, a list of all active loans and the ability to manually delete selected loans are available to the users.

## Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Test** | **Description** | **Outcome** | **Screenshots** |
| 1 | Login Verification | The entered login information is checked against the logins stored within the system | Success |  |
| 2 | Welcome Message | A welcome message is displayed to the user with the employee’s name | Success |  |
| 3 | Loads Admin Menu | The Admin Menu is loaded after successful login | Success |  |
| 4 | Loads Customer Menu | The Customer menu is loaded after being selected from the Admin menu | Success |  |
| 5 | View all customers | A list of all customers & their information is displayed | Success |  |
| 6 | Add new customer | The user can add a new customer to the system | Success |  |
| 7 | Remove a customer | The user can remove a customer from the system | Success |  |
| 8 | Loads Book Menu | The Book menu is loaded after being selected from the Admin menu | Success |  |
| 9 | View All Books | A list of all books & their information is displayed | Success |  |
| 10 | Loaned Status | Whether or not each book is currently loaned is displayed | Success | N/A |
| 11 | Add new book | The user can add a new book to the system | Success |  |
| 12 | Remove a book | The user can remove a book from the system | Success |  |
| 13 | Loads Loan Menu | The Loan menu is loaded after being selected from the Admin menu | Success |  |
| 14 | Create new loan | A new loan can be created by the user | Success |  |
| 15 | Chose book to loan | A book from the Books class can be used in the loan | Success | N/A |
| 16 | Choose customer for loan | A customer from the Customers class can be used in the loan | Success | N/A |
| 17 | Detects user for loan | The current employee user is automatically detected and added to the Loan information | Success | N/A |
| 18 | Sets loan out date | The current date is automatically used and added to the loan information | Success | N/A |
| 19 | Sets loan return date | A date 3 weeks from the current date is calculated and added to the loan information | Success | N/A |
| 20 | Generates Loan ID | A random 5-character ID is generated and added to the Loan information | Success | N/A |
| 21 | View all Loans | A list of all loans & their data is outputted for the user | Success |  |
| 22 | Delete a Loan | The user can delete a loan from the system based on its unique ID | Success |  |
| 23 | Loads Login Menu | The Login Menu is loaded after being selected from the Admin Menu | Success |  |
| 24 | Add new User | The user can add a new Employee Login to the system | Success |  |
| 25 | Delete a User | The user can delete an employee login from the system | Success |  |
| 26 | Customer error cancels Loan | If the chosen customer cannot be found in the customer class, the Loan request is cancelled | Success |  |

# Raw Code

## Task 1

class Program

{

public static Dictionary<string, string> loginInput = new Dictionary<string, string>();

static void Main(string[] args)

{

bool loginState = false;

Console.WriteLine(Environment.NewLine + "Welcome to the Passenger Management System | Please Log in" + Environment.NewLine);

do

{

loginInput.Clear();

Console.Write("Username: ");

string usernameInput = Console.ReadLine().ToUpper(); //Stores input as user login

Console.Write("Password: ");

string passwordInput = Console.ReadLine(); //Stores input as user password

Console.WriteLine("");

loginInput.Add(usernameInput, passwordInput); //passes inputs to method to check them against those stored in the system

loginState = LoginDetails.loginCheck(loginInput); //passes inputs to method to check them against those stored in the system

} while (loginState == false); //Loops until recognised Login is entered

Console.WriteLine("Logging In..." + Environment.NewLine);

CoachDetails.coachPassengers();

Console.ReadLine();

}

}

class CoachDetails

{

public static void coachPassengers()

{

int x = 0;

int[,] Passengers = new int[3, 5] { { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 } }; //A multi-dimensional array for storing passengers: The first row contains pre-booked passengers, the second contains passengers that paid on arrival and the third is the two previous added together (The total)

string[] Coaches = new string[5] { "A", "B", "C", "D", "E" }; //Array containing the designations for each coach of the train

Dictionary<string, int> Filled\_Coaches = new Dictionary<string, int>();

for (int loop\_1 = 0; loop\_1 < 5; loop\_1++)

{

Console.Write("Enter the number of passengers pre-booked for Coach {0} : ", Coaches[x]); //cycles through until all coaches have been checked

string User\_Input = Console.ReadLine();

Passengers[0, x] = int.Parse(User\_Input); //Stores input as number of passengers that pre-booked for current coach

x = x + 1;

Console.WriteLine(Environment.NewLine);

}

x = 0;

for (int loop\_2 = 0; loop\_2 < 5; loop\_2++)

{

Console.Write("Enter the number of passengers that paid on arrival for Coach {0} : ", Coaches[x]); //cycles through until all coaches have been checked

Passengers[1, x] = int.Parse(Console.ReadLine()); //Stores input as number of passengers that paid on arrival for current coach

Passengers[2, x] = Passengers[0, x] + Passengers[1, x]; //calculates passenger total for current coach by adding two previous inputs for said coach together

Filled\_Coaches.Add(Coaches[x], Passengers[2, x]); //stores passenger total

x = x + 1;

Console.WriteLine(Environment.NewLine);

}

foreach (KeyValuePair<string, int> coaches in Filled\_Coaches.OrderBy(key => key.Value)) //generates new key order based on value amount

{

Console.WriteLine("Coach: {0} | Passengers: {1}", coaches.Key, coaches.Value); //displays all coaches and their passenger totals in ascending order

}

Console.ReadLine();

}

}

class LoginDetails

{

static Dictionary<string, string> loginData = new Dictionary<string, string>() //Storage of all user login data

{

{"USERNAME","password"}

};

public static bool loginCheck(Dictionary<string, string> loginInput) //method to check entered login information is recognised

{

foreach (var login in loginData) //check every login in the system

{

if (loginInput.Keys.ElementAt(0) == login.Key & loginInput.Values.ElementAt(0) == login.Value) //if they match, return that it is correct

{

return true;

}

}

Console.WriteLine("Error | Unrecognised Login Details | Try Again"); //if no match can be found, return error message

return false;

}

}

## Task 2

class Program

{

static void Main(string[] args)

{

bool loopBreak = false;

Console.WriteLine("Welcome to the Spelling Quiz"); //Welcomes the user to the system

Console.WriteLine("Please Log in to use the system");

do

{

Console.WriteLine("Please enter your Username");

string usernameInput = Console.ReadLine().ToUpper(); //Saves the user's entered Username

Console.WriteLine("Please enter your Password");

string passwordInput = Console.ReadLine(); //Saves the user's entered Password

List<string> loginInput = new List<string>(2)

{

usernameInput,passwordInput //Saves both of the user inputs as a List

};

string userRole = SchoolLogins.detailsCheck(loginInput); //Passes the inputs to a method to check if they are recognised

if (userRole == "Teacher") //If the user is recognised and is a teacher

{

List<string> teacherName = new List<string>();

teacherName = SchoolLogins.userName();

Console.WriteLine("Welcome: {0} {1}", teacherName.ElementAt(0), teacherName.ElementAt(1) + Environment.NewLine); //Personalised Welcome Message

TeacherMenu.teacherMenu(); //Loads the Teacher Menu for the Software

}

else if (userRole == "Student") //If the user is recognised and is a student

{

List<string> studentName = new List<string>();

studentName = SchoolLogins.userName();

Console.WriteLine("Welcome: {0} {1}", studentName.ElementAt(0), studentName.ElementAt(1)); //Personalised Weclome Message

StudentMenu.startQuiz(); //Loads the Quiz

}

else //If the user is unrecognised

{

loopBreak = false; //Resets the Loop

Console.WriteLine("Error | User not found | Please try Again" + Environment.NewLine);

}

} while (loopBreak == false);

}

}

class SchoolLogins

{

static Dictionary<string, string> userLogins = new Dictionary<string, string>() //Storage of the Login details of the users

{

{"EXAMPLESTUDENT","password"},

{"EXAMPLETEACHER","password"}

};

static Dictionary<string, string> userNames = new Dictionary<string, string>() //Storage of the Names of the users

{

{"Example","Student"},

{"Mr.","Example"}

};

static List<bool> userIsTeacher = new List<bool>() //Bool to signify whether each user is a student or teacher

{

false, true

};

static int userCounter = 0;

public static string detailsCheck(List<string> details) //Checks the user input from program to see if they are recognised

{

bool loopBreak = false;

do

{

if(userCounter == userLogins.Count) //If every recognised login has been checked, return that there was no match

{

loopBreak = true; //end the loop

userCounter = 0;

}

else if(details.ElementAt(0) == userLogins.Keys.ElementAt(userCounter) & details.ElementAt(1) == userLogins.Values.ElementAt(userCounter)){ //If both the username and password match

switch (userIsTeacher.ElementAt(userCounter))

{

case true: //If the user is a teacher, return that to the 'program' class

return "Teacher";

case false: //If the user is a student, return that to the 'program' class

return "Student";

}

}

else

{

userCounter = userCounter + 1; //increase the placement in the dictonary to check, increments by 1 each time

}

} while (loopBreak == false);

return null;

}

public static List<string> userName()

{

List<string> currentUser = new List<string>()

{

userNames.Keys.ElementAt(userCounter), userNames.Values.ElementAt(userCounter) //Calls the full name of the curent user for display in the welcome message

};

return currentUser;

}

public static void studentList() //outputs a list of all students

{

int x = 0;

do

{

if (userIsTeacher.ElementAt(x) == false) //If the user is not a student

{

Console.WriteLine("{0} , {1} | {2}", userNames.Values.ElementAt(x), userNames.Keys.ElementAt(x), userLogins.Keys.ElementAt(x)); //Display all students, with their names and username

x = x + 1; //location increases, increments by 1

}

else

{

x = x + 1;

}

} while (x != userLogins.Keys.Count); //Runs until all stored user details are displayed

}

public static bool addStudent() //The method to add new students to the system

{

bool loopBreak = false;

string firstName;

string secondName;

string userName;

string password;

do

{

Console.WriteLine("Enter the Student's First Name");

firstName = Console.ReadLine().ToUpper(); //Saves the input as the new student's first name

if (firstName.Length < 1)

{

Console.WriteLine("Error | No Input Detected | Try Again"); //If nothing is entered, display an error

return false;

}

Console.WriteLine("Enter the Student's Surname");

secondName = Console.ReadLine().ToUpper(); //Saves the input as the new student's surname

if (secondName.Length < 1)

{

Console.WriteLine("Error | No Input Detected | Try Again"); //If nothing is entered, display an error

return false;

}

Console.WriteLine("Enter the Student's new Username");

userName = Console.ReadLine().ToUpper(); //Saves the input as the new student's username

if (userName.Length < 1)

{

Console.WriteLine("Error | No Input Detected | Try Again"); //If nothing is entered, display an error

return false;

}

Console.WriteLine("Enter the Student's new Password");

password = Console.ReadLine(); //Saves the input as the new student's password

if (password.Length < 1)

{

Console.WriteLine("Error | No Input Detected | Try Again"); //If nothing is entered, display an error

return false;

}

loopBreak = true;

} while (loopBreak == false);

userNames.Add(firstName, secondName); //Saves the entered Username and Password to the system

userLogins.Add(userName, password); //Saves the entered First and Last Name to the system

userIsTeacher.Add(false); //Saves that the new user is a student

return true;

}

public static bool removeStudent() //method used to remove students from the system

{

bool loopBreak = false;

int x = 0;

Console.WriteLine("Enter a username to delete from the system");

string choice = Console.ReadLine().ToUpper(); //Saves the input as the desired username to delete

do

{

if (x == userLogins.Keys.Count) //If every user has been checked without a match

{

Console.WriteLine("Error | Could not find requested User"); //return an error and delete no users

return false;

}

else if (choice == userLogins.Keys.ElementAt(x)) //if a user has been found with a matching username

{

userLogins.Remove(choice);

choice = userNames.Keys.ElementAt(x);

userNames.Remove(choice);

bool toRemove = userIsTeacher.ElementAt(x);

userIsTeacher.Remove(toRemove); //remove all details from the system for that user

loopBreak = true;

}

else

{

x = x + 1;

}

} while (loopBreak == false);

return true;

}

}

class SpellingList

{

static Dictionary<string, string> quizWords = new Dictionary<string, string>() //Storage off all quiz Questions & Answers

{

{"The second month in the year is:","FEBRUARY"},

{"When you have made your mind up, you have:","DECIDED"},

{"If something is not boring, it is:","INTERESTING"},

{"Not all the time but:","SOMETIMES"},

{"Not the same but:","DIFFERENT"}

};

static List<char> quizHints = new List<char>(); //Stores the Hints for the questions

public static void generateHints()

{

//Generates hints by taking the first letter of each word

int z = 0;

do

{

char newHint = quizWords.Values.ElementAt(z)[0];

quizHints.Add(newHint);

z = z + 1;

} while (z != quizWords.Keys.Count);

}

static int questionNumber = 0;

static int studentScore = 0;

public static bool addWord() //Method to add a new Question

{

Console.WriteLine("Enter the question to be Asked");

string question = Console.ReadLine(); //Sets Teacher input as new question to be asked

if (question.Length < 1) //If nothing is entered, Display error message

{

Console.WriteLine("No Input detected | Request Failed | Please Try Again");

return false;

}

Console.WriteLine("Enter the Answer to the above Question");

string answer = Console.ReadLine().ToUpper(); //Sets Teacher Input as new answer for the question

if (answer.Length < 1) //If nothing is entered, Display error message

{

Console.WriteLine("No Input detected | Request Failed | Please Try Again");

return false;

}

quizWords.Add(question, answer); //Adds the entered Question & Answer to the quiz

quizHints.Add(answer[0]); //Adds the Question's hint to the list of hints

return true;

}

public static bool removeWord() //Method to remove a Question

{

int check = 0;

Console.WriteLine("Enter a word to delete from the quiz");

string choice = Console.ReadLine().ToUpper(); //Stores Teacher Input as word to remove

do

{

if(choice == quizWords.Values.ElementAt(check)) //If the entered word is found in the system

{

string removeQuestion = quizWords.Keys.ElementAt(check);

quizWords.Remove(removeQuestion); //remove the question and answer

return true;

}

check++;

} while (check != quizWords.Keys.Count);

return false;

}

public static void quizOutput() //method to output all questions and answers

{

int count = 0;

do

{

Console.WriteLine("{0} {1}", quizWords.Keys.ElementAt(count), quizWords.Values.ElementAt(count)); //Display all Questions and Answers in the system

count++;

} while (count != quizWords.Keys.Count);

}

public static List<double> studentGuess() //Method for checking student's guess against stored answer

{

SpellingList.generateHints(); //calls the method to generate hints

bool hintUsed = false;

do

{

if (hintUsed == true) //If student has requested hint

{

Console.WriteLine("First Letter Hint: {0}", quizHints.ElementAt(questionNumber)); //display the first letter of the current answer

}

Console.WriteLine(quizWords.Keys.ElementAt(questionNumber)); //Asks the next question

if (hintUsed == false) //If user hasn't already asked for a hint for this question

{ Console.WriteLine("Type 'Hint' to get a hint | Warning! Doing so will cost you a point of your score"); } //Let the user know that they can request a hint

string studentAnswer = Console.ReadLine().ToUpper();

string correctAnswer = quizWords.Values.ElementAt(questionNumber);

int characterLetter = 0;

int wordScore = 10; //Sets maximum score possible for each word

int wrongScore = 0;

if (studentAnswer == "HINT")

{

hintUsed = true;

studentScore = studentScore - 1; //If hint has been used, deduct a point from the student's total score

}

else

{

char[] guessCharacters;

guessCharacters = studentAnswer.ToCharArray(0, studentAnswer.Length); //Seperates each character of student's answer into a character array

char[] answerCharacters;

answerCharacters = correctAnswer.ToCharArray(0, correctAnswer.Length); //Seperates each chatacter of the correct answer into a charatcer array

do

{

if (guessCharacters.ElementAt(characterLetter) == answerCharacters.ElementAt(characterLetter)) //if the current letter from each char array match

{

characterLetter++; //move onto next letter

}

else if (guessCharacters.ElementAt(characterLetter) != answerCharacters.ElementAt(characterLetter)) //if the two letters don't match

{

characterLetter++; //move onto next letter

wrongScore++; //increase the count of incorrect letters

}

} while (characterLetter != studentAnswer.Length);

Console.WriteLine("The correct answer is: {0}", correctAnswer + Environment.NewLine); //Display the correct answer for the question

if (wrongScore >= (correctAnswer.Length / 2) | studentAnswer.Length <= (correctAnswer.Length / 2)) //If the number of incorrect letters is at least half the length of the current correct answer

{

wrongScore = 10; //inncorect letters count is set to max

}

wordScore = wordScore - wrongScore; //score for this word is calculated through deducting a point for each incorrect letter from the maximum total score

studentScore = studentScore + wordScore; //add score for this word to the student's total score

questionNumber++;

hintUsed = false; //resets whether a hint has been used

};

} while (questionNumber != quizWords.Keys.Count); //runs until all questions have been asked

List<double> calculatedScore = new List<double>();

calculatedScore = SpellingList.scoreCalculation(studentScore); //passes the score of the student to the method for calculating the percentage

return calculatedScore;

}

public static List<double>scoreCalculation(int totalScored) //method for calculating offical student score

{

double canScore = quizWords.Keys.Count \* 10;

List<double> scoreDecimal = new List<double>();

scoreDecimal.Add(totalScored);

scoreDecimal.Add (totalScored / canScore); //converts student score for use in Student class

return scoreDecimal;

}

}

class TeacherMenu

{

static List<string> teacherOptions = new List<string>() //Stores all possible options for the menu choice

{

"V","S","R","A","Q","D","X"

};

public static void teacherMenu()

{

bool loopBreak = false;

do

{

Console.WriteLine(""); //Lists out all options for the teacher Menu

Console.WriteLine("What would you like to do?" + Environment.NewLine);

Console.WriteLine("Press V to: view all of the students currently enrolled");

Console.WriteLine("Press S to: enroll an additional student");

Console.WriteLine("Press R to: remove a student from the class");

Console.WriteLine("Press A to: view all questions & answers");

Console.WriteLine("Press Q to: add another question to the quiz");

Console.WriteLine("Press D to: remove a question from the quiz");

Console.WriteLine("Press X to: exit the software" + Environment.NewLine);

string choice = Console.ReadLine().ToUpper(); //Stores the user's choice

choice = teacherChoice(choice);

if (choice == "V")

{

SchoolLogins.studentList(); //Call the method to view all users

}

else if (choice == "S")

{

SchoolLogins.addStudent(); //Call the methods to add a user

}

else if (choice == "R")

{

SchoolLogins.removeStudent(); //Calls the methods to remove a student

}

else if (choice == "A")

{

SpellingList.quizOutput(); //Calls the method to view all questions and correct answers

}

else if (choice == "Q")

{

SpellingList.addWord(); //Calls the method to a question & answer

}

else if (choice == "D")

{

SpellingList.removeWord(); //Calls the method to remove a question & answer

}

else if (choice == "X")

{

System.Environment.Exit(1); //Closes the software

}

else

{

Console.WriteLine("Unknown Input Detected | Try again" + Environment.NewLine); //If the user's choice doesn't match, return an error

}

} while (loopBreak == false);

}

static string teacherChoice(string choice) //checks the user input and converts it to a format that can be checked in the above method

{

int y = 0;

bool loopBreak = false;

do

{

if (y == teacherOptions.Count)

{

loopBreak = true;

}

else if (choice == teacherOptions.ElementAt(y))

{

return choice;

}

else

{

y = y + 1;

}

} while (loopBreak == false);

return null;

}

}

class StudentMenu

{

public static void startQuiz() //Method to run the spelling quiz

{

Console.WriteLine("Welcome to the School Spelling Quiz" + Environment.NewLine + "Press Any Key to begin" + Environment.NewLine); //Welcomes the user

Console.ReadLine();

List<double> decimalScore = SpellingList.studentGuess(); //Calls the method to run through each question in the quiz

Console.WriteLine("");

Console.WriteLine("You got A Scrore of {0} ({1}%)", decimalScore.ElementAt(0), decimalScore.ElementAt(1) \* 100); //Displays the users score as a total and a percentage

Console.ReadLine();

System.Environment.Exit(1);

}

}

## Task 3

class Program

{

private static void Main(string[] args)

{

bool loopBreak = false;

bool adminOverride = false;

string check = "";

string loginError;

Console.WriteLine("Welcome to the Company System"); //Welcomes the user

do

{

Console.WriteLine("Please Enter your Username");

string usernameInput = Console.ReadLine().ToUpper(); //stores user input of username

Console.WriteLine("Please Enter your Password");

string passwordInput = Console.ReadLine(); //stores user input of password

List<string> loginInput = new List<string>(2)

{

usernameInput, passwordInput //stores inputted login in list

};

bool loginMatch = UserLogins.detailsCheck(loginInput); //passes login info to method for validation

bool adminCheck = false;

List<string> userDetails = new List<string>(2);

if (loginMatch == true) //if login successful

{

userDetails = UserLogins.retrieveName(userDetails); //retrives user's name for welcome message from method

adminCheck = UserLogins.checkAdmin(check); //retrieves user's admin status from method

}

if (loginMatch == true & adminCheck == true) //if user is logged in and an admin

{

loopBreak = true;

Console.WriteLine("Welcome Admin: {0} {1}", userDetails.ElementAt(0), userDetails.ElementAt(1)); //personalised welcome message

AdminUser.adminMenu(); //Loads Admin menu

}

else if(loginMatch == true) //if user is logged in but not an admin

{

Console.WriteLine("Welcome {0} {1}", userDetails.ElementAt(0), userDetails.ElementAt(1)); //personalised welcome message

User.userMenu(); //LOads Regular User Menu

loopBreak = true;

}

else if(UserLogins.loginAttempts == 0) //If user has not logged in and has used all login attempts

{

loginError = UserLogins.attemptsCheck(check); //calls error to display

Console.WriteLine("");

Console.WriteLine(loginError + Environment.NewLine);

loopBreak = true;

adminOverride = true;

}

else

{

loginError = UserLogins.attemptsCheck(check); //if user has not logged in and has login attempts remaining

Console.WriteLine(loginError); //displays error

Console.WriteLine("");

loopBreak = false;

}

} while(loopBreak == false);

do //admin override

{

Console.WriteLine("Enter Admin Username");

string adminUsername = Console.ReadLine().ToUpper(); //entered Admin Login

Console.WriteLine("Enter Admin Password");

string adminPassword = Console.ReadLine(); //entered Admin Password

List<string> adminInput = new List<string>(2)

{

adminUsername,adminPassword //stores entered login as list

};

AdminOverride.adminAuthourisation(adminInput); //checks login list against recognised Admin logins

} while (adminOverride == true);

Console.ReadLine();

}

}

class AdminUser

{

static List<string> adminOptions = new List<string>() //All Possbile answers to Admin Menu

{

"!","V","A","R","X"

};

public static void adminMenu()

{

bool loopBreak = false;

do

{

Console.WriteLine(""); //Displays list of options for Admin

Console.WriteLine("What would you like to do?");

Console.WriteLine("Press V to view user details | Press A to add users to the system |");

Console.WriteLine("Press R to remove a user from the system | Press X to exit the system|");

string userChoice = Console.ReadLine().ToUpper(); //saves choice

string choice = adminOptions.ElementAt(adminChoice(userChoice)); //passes choice to method to check against possible answers

if (choice == "!")

{

Console.WriteLine("Unknown Input Detected | Try again"); //Asks user to re-enter choice

}

else if (choice == "V")

{

UserLogins.userOutput(); //View all User information

}

else if (choice == "A")

{

UserLogins.addUsers(); //Add a new user to the system

}

else if (choice == "R")

{

UserLogins.removeUsers(); //Remove a user from the system

}

else if (choice == "X")

{

System.Environment.Exit(1); //Closes the software

}

} while (loopBreak == false);

}

static int adminChoice(string choice) //compares entered admin choice against possible ones stored in the system

{

int y = 0;

bool loopBreak = false;

do

{

if (y == adminOptions.Count)

{

loopBreak = true;

}

else if (choice == adminOptions.ElementAt(y))

{

return y;

}

else

{

y = y + 1;

}

} while (loopBreak == false);

return 0;

}

}

class AdminOverride

{

public static void adminAuthourisation(List<string> loginCheck)

{

if (UserLogins.authorisedCheck(loginCheck) == true) //If admin login is recognised

{

Console.WriteLine("Admin Override Successful");

Console.WriteLine("Press Enter to Close the Software");

Console.ReadLine();

System.Environment.Exit(1); //Exit the software

}

else

{

Console.WriteLine("Unrecognised Admin Login | Try Again" + Environment.NewLine); //Ask user to re-enter admin Login

}

}

}

class UserLogins

{

static Dictionary<string, string> Employee\_Login = new Dictionary<string, string>() //Stores all user login information

{

{"JOHNSM22738","D7y6a"},

{"JANEDO98786","i&acN"},

{"BRYNW56655","GgjN6"},

{"NESSAJ25255","3KsyX"}

};

static Dictionary<string, string> Employee\_Names = new Dictionary<string, string>() //Stores all user personal details

{

{"John", "Smith"},

{"Jane", "Doe" },

{"Bryn", "Williams" },

{ "Nessa", "Jenkins"}

};

static List<bool> Employee\_Admin = new List<bool>(2) //stores whether each user is an admin

{

true, false, false, false

};

public static int x;

public static int loginAttempts = 3;

public static bool detailsCheck(List<string> test)

{

x = 0;

bool userMatch = false;

bool loopBreak = false;

do

{

if (x == Employee\_Login.Count & (loginAttempts == 3 | loginAttempts == 2 | loginAttempts == 1)) //If login is unrecognised and user has attempts remaining

{

loginAttempts = loginAttempts - 1; //deduct one login attempt

x = 0;

userMatch = false;

loopBreak = true;

}

else if (test.ElementAt(0) == Employee\_Login.Keys.ElementAt(x) & test.ElementAt(1) == Employee\_Login.Values.ElementAt(x)) //If login is recognised

{

loginAttempts = 3;

userMatch = true;

loopBreak = true;

}

else

{

x = x + 1; //itterate place in dictonary by one

}

} while (loopBreak == false);

return userMatch;

}

public static string attemptsCheck(string logins)

{

if (loginAttempts >= 2) //2 attempts remaining

{

logins = $"Incorrect Login Details | {loginAttempts} attempts remaining";

return logins;

}

else if (loginAttempts == 1) //1 attempt remaining

{

logins = "Incorrect Login Details | Warning! - Only 1 attempt remaining before the device will need to be unlocked by an admin";

return logins;

}

else if (loginAttempts == 0) //No attempts remaining

{

logins = "Inncorrect Login Details" + Environment.NewLine + "Maximum number of login attempts reached | Device will need to be unlocked by an admin";

return logins;

}

else

{

return null;

}

}

public static List<string> retrieveName(List<string> name) //Method to retrieve name for current user

{

List<string> employeeName = new List<string>(2)

{

Employee\_Names.Keys.ElementAt(x),

Employee\_Names.Values.ElementAt(x)

};

return employeeName;

}

public static bool checkAdmin(string check) //checks whether current user is an Admin

{

bool isAdmin = false;

if (Employee\_Admin.ElementAt(x) == true)

{

isAdmin = true;

return isAdmin;

}

else

{

isAdmin = false;

return isAdmin;

}

}

public static bool authorisedCheck(List<string> check)

{

bool loopBreak = false;

do

{

if (x == Employee\_Login.Count)

{

x = 0;

return false;

}

else if (check.ElementAt(0) == Employee\_Login.Keys.ElementAt(x) & check.ElementAt(1) == Employee\_Login.Values.ElementAt(x) & Employee\_Admin.ElementAt(x) == true)

{

x = 0;

loopBreak = true;

}

else

{

x = x + 1;

}

} while (loopBreak == false);

return true;

}

public static bool addUsers() //Method to add new user to the system

{

bool loopBreak = false;

string fnameInput;

string snameInput;

string unameInput;

string passwordInput;

string adminInput;

bool adminTrue = false;

do

{

Console.WriteLine("Enter the First Name of the new user");

fnameInput = Console.ReadLine().ToUpper(); //Stores input as new first name

if(fnameInput.Length < 1) //If nothing is typed, display error

{

Console.WriteLine("No input detected, Try again");

return false;

}

Console.WriteLine("Enter the Surname of the new user");

snameInput = Console.ReadLine().ToUpper(); //Stores input as new surname

if (snameInput.Length < 1) //If nothing is typed, display error

{

Console.WriteLine("No input detected, Try again");

return false;

}

Console.WriteLine("Enter the Desired Username of the new user");

unameInput = Console.ReadLine().ToUpper(); //Stores input as new Username

if (unameInput.Length < 1) //If nothing is typed, display error

{

Console.WriteLine("No input detected, Try again");

return false;

}

Console.WriteLine("Enter the Desired Password of the new user");

passwordInput = Console.ReadLine(); //Stores input as new Password

if (passwordInput.Length < 1) //If nothing is typed, display error

{

Console.WriteLine("No input detected, Try again");

return false;

}

Console.WriteLine("Is this user an Admin? (Y/N)");

adminInput = Console.ReadLine().ToUpper(); //Stores whether new user is an admin

if(adminInput == "Y")

{

adminTrue = true;

}

else

{

adminTrue = false;

}

loopBreak = true;

} while (loopBreak == false);

Employee\_Login.Add(unameInput, passwordInput); //Adds all new information as a user on the system

Employee\_Names.Add(fnameInput, snameInput);

Employee\_Admin.Add(adminTrue);

return true;

}

public static bool removeUsers() //Method to remove a user from the system

{

bool loopBreak = false;

int y = 0;

Console.WriteLine("Enter a username to delete from the system");

string choice = Console.ReadLine().ToUpper(); //stores

do

{

if (y == Employee\_Login.Keys.Count)

{

Console.WriteLine("Error | Could not find User"); //if all users have been checked, return an eror

return false;

}

else if (choice == Employee\_Login.Keys.ElementAt(y)) //if a match is found

{

string toRemove = Employee\_Login.Keys.ElementAt(y); //remove that user

Employee\_Login.Remove(toRemove);

toRemove = Employee\_Names.Keys.ElementAt(y);

Employee\_Names.Remove(toRemove);

bool toRemove2 = Employee\_Admin.ElementAt(y);

Employee\_Admin.Remove(toRemove2);

loopBreak = true;

}

else

{

y = y + 1; //iterate place in dictionary by 1

}

} while (loopBreak == false);

return true;

}

public static void userOutput() //Method to display all users on the system

{

int z = 0;

do

{

Console.WriteLine("{0} - {1} - {2} - {3} - Admin: {4}", Employee\_Login.Keys.ElementAt(z), Employee\_Login.Values.ElementAt(z), Employee\_Names.Keys.ElementAt(z), Employee\_Names.Values.ElementAt(z), Employee\_Admin.ElementAt(z));

z = z + 1;

} while (z != (Employee\_Login.Keys.Count));

}

public static void changePassword(string oldPassword) //Method to allow users to change their password

{

bool loopBreak = false;

if (oldPassword == Employee\_Login.Values.ElementAt(x)) //if current password is entered correctly

{

do

{

Console.WriteLine("Please Enter your new Password");

string newPassword1 = Console.ReadLine(); //enter new password

Console.WriteLine("Please Confirm your new Password");

string newPassword2 = Console.ReadLine(); //confirm new password

if (newPassword1 == newPassword2)

{

string currentUser = Employee\_Login.Keys.ElementAt(x);

Employee\_Login[currentUser] = newPassword1;

Console.WriteLine("Password Updated Successfully"); //update password

loopBreak = true;

}

else

{

Console.WriteLine("Those passwords did not match, please try again"); //give error - ask user to try again

Console.WriteLine("");

loopBreak = false;

}

} while (loopBreak == false);

}

}

}

class User

{

static List<string> userOptions = new List<string>() //All options for User Menu

{

"!","X","P"

};

public static void userMenu()

{

bool loopBreak = false;

do

{

Console.WriteLine("What would you like to do?");

Console.WriteLine("Press X to Log out | Press P to change your password");

string userAction = Console.ReadLine().ToUpper(); //User's choice

string choice = userOptions.ElementAt(userChoice(userAction));

if (choice == "!")

{

Console.WriteLine("Unknown Input Detected | Try again"); //if unrecognised input, ask user to try again

}

else if (choice == "X")

{

System.Environment.Exit(1); //Exit the software

}

else if (choice == "P") //Reset user's password

{

Console.WriteLine("Please enter your current password");

string oldPassword = Console.ReadLine();

UserLogins.changePassword(oldPassword); //passes current password input to method to update password

}

} while (loopBreak == false);

}

static int userChoice(string choice) //converts user choce input into refrence in options list

{

int y = 0;

bool loopBreak = false;

do

{

if (y == userOptions.Count)

{

loopBreak = true;

}

else if (choice == userOptions.ElementAt(y))

{

return y;

}

else

{

y = y + 1;

}

} while (loopBreak == false);

return 0;

}

}

## Task 4

class Program

{

public static Dictionary<string, string> loginInput = new Dictionary<string, string>();

static void Main(string[] args)

{

bool loginState = false; //Welcomes the user to the system

Console.WriteLine(Environment.NewLine + "Welcome to the Library Internal System | Please Log in" + Environment.NewLine);

do

{

loginInput.Clear();

Console.Write("Username: ");

string usernameInput = Console.ReadLine().ToUpper(); //Stores input as user login

Console.Write("Password: ");

string passwordInput = Console.ReadLine(); //Stores input as user password

Console.WriteLine("");

loginInput.Add(usernameInput, passwordInput); //passes inputs to method to check them against those stored in the system

loginState = LoginDetails.loginCheck(loginInput); //passes inputs to method to check them against those stored in the system

} while (loginState == false); //Loops until recognised Login is entered

Console.WriteLine("Welcome Admin" + Environment.NewLine); //Personalised Welcome Message

AdminControl.adminMenu(); //Loads user into Admin Menu

Console.ReadLine();

}

}

class CustomerDetails

{

static Dictionary<string, string[]> customerRecords = new Dictionary<string, string[]>(); //Stores all customer records in the system

static bool constantMenu = false;

public static void customerMenu()

{

do

{ //Displays all options for the Admin relating to customers

Console.WriteLine(Environment.NewLine + "------------------------" + Environment.NewLine + "CUSTOMER MANAGEMENT MENU" + Environment.NewLine);

Console.WriteLine("Type V to | View All Customers" + Environment.NewLine + "Type A to | Register a Customer" + Environment.NewLine + "Type D to | Delete a Customer" + Environment.NewLine + "Type B to | Go Back" + Environment.NewLine + "Type X to | Exit the Software");

Console.WriteLine(Environment.NewLine);

Console.Write("What would you like to do? ");

string userChoice = Console.ReadLine().ToUpper(); //Saves Admin choice in the menu

int j = menuChoice(userChoice);

switch (j) //Checks Which Menu option was selected

{

case 1:

viewCustomers(); //Output List of all customers and their details

break;

case 2:

addCustomer(); //Add a new customer to the system

break;

case 3:

removeCustomer(); //Remove a new customer to the system

break;

case 4:

AdminControl.adminMenu(); //Return to the main Admin Menu

break;

case 5:

Environment.Exit(1); //Close the Software

break;

}

} while (constantMenu == false);

}

static int menuChoice(string choice)

{

Dictionary<int, string> menuChoices = new Dictionary<int, string>() //Checks User choice against possible choices in the system

{

{1,"V"},{2,"A"},{3,"D"},{4,"B"},{5,"X"}

};

foreach (var option in menuChoices)

{

if (choice == option.Value)

{

return option.Key;

}

}

return 0;

}

static void viewCustomers() //Method to output a list of all customers and their details

{

foreach (var customer in customerRecords) //cycles until all customers have been displayed

{

Console.WriteLine(Environment.NewLine + "Customer ID: {0} | Name: {2}, {1} | D.O.B: {3} | Customer Since: {4}", customer.Key, customer.Value[0],customer.Value[1], customer.Value[2],customer.Value[3]);

}

}

static bool addCustomer() //Method to add a new customer to the system

{

Console.WriteLine("Please enter the new Customer's details below:");

Console.Write("First Name: ");

string firstName = Console.ReadLine().ToUpper(); //Stores input as new customer's first name

Console.Write("Surname: ");

string surname = Console.ReadLine().ToUpper(); //Stores input as new customer's surname name

Console.WriteLine("Date of Birth (dd/mm/yyyy): ");

string birthdate = Console.ReadLine(); //Stores input as new customer's date of birth

string joinYear = Convert.ToString(DateTime.Now.Year); //Sets Join year as current year

string newID = Guid.NewGuid().ToString();

char[] idCharacters = newID.Take(5).ToArray(); //generates random unique 5 digit ID for customer

newID = new string(idCharacters);

string[] newCustomer = new string[4] {firstName, surname, birthdate, joinYear}; //Adds inputs as new customer object

foreach(var existing in customerRecords)

{

if(newCustomer[0] == existing.Value[0] & newCustomer[1] == existing.Value[1]) //If customer already exists in the system, display error and abort the method

{

Console.WriteLine(Environment.NewLine + "Error | Customer is already registered" + Environment.NewLine);

return false;

}

}

customerRecords.Add(newID, newCustomer); //Add new customer to the system

Console.WriteLine(Environment.NewLine + "Customer Succesfully Added" + Environment.NewLine);

return true;

}

static bool removeCustomer() //Method to remove a customer from the system

{

Console.WriteLine("Enter the details of the customer to be Removed:");

Console.Write("First Name: ");

string firstName = Console.ReadLine().ToUpper(); //First name of customer to be removed

Console.Write("Surname: ");

string surname = Console.ReadLine().ToUpper(); //Surname of customer to be removed

foreach(var customer in customerRecords) //check every customer stored on the system

{

if (firstName == customer.Value[0] & surname == customer.Value[1]) //If entered information matches a customer

{

string toRemove = customer.Key;

customerRecords.Remove(toRemove); //Remove the customer

Console.WriteLine(Environment.NewLine + "Customer Succesfully Removed" + Environment.NewLine);

return true;

}

}

Console.WriteLine(Environment.NewLine + "Error | Couldn't Find Customer" + Environment.NewLine); //If customer cannot be found, display error

return false;

}

public static bool lookupCustomer(string[] customerCheck) //Method to find customer to use in Loan

{

foreach (var existing in customerRecords)

{

if (customerCheck[0] == existing.Value[0] & customerCheck[1] == existing.Value[1]) //If customer is found, return that it does to the Loan Details CLass

{

return true;

}

}

return false;

}

}

class LoginDetails

{

static Dictionary<string, string> loginData = new Dictionary<string, string>() //Storage of all user login data

{

{"ADMIN","password"}

};

static List<string> loginNames = new List<string>()

{

{"Admin"} //storage of all user's names

};

static bool constantMenu = false;

public static bool loginCheck(Dictionary<string, string> loginInput) //method to check entered login information is recognised

{

foreach (var login in loginData) //check every login in the system

{

if (loginInput.Keys.ElementAt(0) == login.Key & loginInput.Values.ElementAt(0) == login.Value) //if they match, return that it is correct

{

return true;

}

}

Console.WriteLine("Error | Unrecognised Login Details | Try Again"); //if no match can be found, return error message

return false;

}

public static void loginMenu()

{

do

{ //Display all options for admin relating to Login management

Console.WriteLine(Environment.NewLine + "---------------------" + Environment.NewLine + "LOGIN MANAGEMENT MENU" + Environment.NewLine);

Console.WriteLine("Type A to | Add a Login" + Environment.NewLine + "Type D to | Delete a Login" + Environment.NewLine + "Type B to | Go Back" + Environment.NewLine + "Type X to | Exit the Software");

Console.WriteLine(Environment.NewLine);

Console.Write("What would you like to do? ");

string userChoice = Console.ReadLine().ToUpper(); //saved Admin choice

int i = menuChoice(userChoice);

switch (i) //checks which menu choice the admin has chosen

{

case 1:

addLogin(); //Add a new login to the system

break;

case 2:

removeLogin(); //Remove a login from the system

break;

case 3:

AdminControl.adminMenu(); //Return to the main Admin Menu

break;

case 4:

Environment.Exit(1); //Close the software

break;

}

} while (constantMenu == false);

}

static int menuChoice(string choice) //Method to check if admin choice for menu is valid and recognised

{

Dictionary<int, string> menuChoices = new Dictionary<int, string>()

{

{1,"A"},{2,"D"},{3,"B"},{4,"X"}

};

foreach (var option in menuChoices)

{

if (choice == option.Value)

{

return option.Key;

}

}

return 0;

}

static bool addLogin() //Method to add a new login to the system

{

Console.WriteLine("Enter the Information for the new User");

Console.Write("First Name: ");

string firstName = Console.ReadLine().ToUpper();

if (firstName.Length < 1) //If nothing is entered, return an error

{

Console.WriteLine("Error | No Input Detected | Try Again");

return false;

}

Console.Write("Username: ");

string username = Console.ReadLine().ToUpper();

if (username.Length < 1)

{

Console.WriteLine("Error | No Input Detected | Try Again");

return false;

}

Console.Write("Password: ");

string password = Console.ReadLine();

if (password.Length < 1)

{

Console.WriteLine("Error | No Input Detected | Try Again");

return false;

}

loginData.Add(username, password);

loginNames.Add(firstName); //Add new user to the system

Console.WriteLine(Environment.NewLine + "User Succesfully Added" + Environment.NewLine);

return true;

}

static bool removeLogin() //Method to remove a login from the system

{

int x = 0;

Console.WriteLine("Enter the details of the user to be removed");

Console.Write("Username: ");

string username = Console.ReadLine().ToUpper();

Console.Write("Password: ");

string password = Console.ReadLine();

foreach (var user in loginData)

{

if (username == user.Key & password == user.Value) //If entered username and password match a login on the system

{

string toRemove = user.Key;

loginData.Remove(toRemove);

loginNames.RemoveAt(x); //Remove the login

Console.WriteLine(Environment.NewLine + "User Succesfully Removed" + Environment.NewLine);

return true;

}

x++;

}

Console.WriteLine(Environment.NewLine + "Error | Couldn't Find User" + Environment.NewLine); //if no match can be found, display an error

return false;

}

public static string currentUser() //Method to retrive name of the current logged in user

{

string[] currentLogin = new string[2] { Program.loginInput.Keys.ElementAt(0),Program.loginInput.Values.ElementAt(0)};

return currentLogin[0];

}

}

class AdminControl

{

static bool constantMenu = false;

public static void adminMenu()

{

do

{

Console.WriteLine(Environment.NewLine + "----------" + Environment.NewLine + "ADMIN MENU" + Environment.NewLine);

Console.WriteLine("Type C to | Manage Customers" + Environment.NewLine + "Type B to | Manage Books" + Environment.NewLine + "Type L to | Manage Loans" + Environment.NewLine + "Type U to | Manage Logins" + Environment.NewLine + "Type X to | Exit the Software" + Environment.NewLine);

Console.Write("What would you like to do? "); //Displays Admin options to the user

string userChoice = Console.ReadLine().ToUpper(); //stores admin choice

int j = adminChoice(userChoice);

switch (j) //checks options for admin menu

{

case 1:

CustomerDetails.customerMenu(); //Loads Class to manage customers

break;

case 2:

BookDetails.bookMenu(); //Loads Class to manage books

break;

case 3:

LoanDetails.loanMenu(); //Loads Class to manage loans

break;

case 4:

LoginDetails.loginMenu(); //Loads class to manage Logins

break;

case 5:

Environment.Exit(1); //Exit the software

break;

}

} while (constantMenu == false);

}

static int adminChoice(string choice)

{

Dictionary<int,string> adminChoices = new Dictionary<int, string>()

{

{1,"C"},{2,"B"},{3,"L"},{4,"U"},{5,"X"} //Checks Admin choice against possible accepted options

};

foreach (var option in adminChoices)

{

if (choice == option.Value)

{

return option.Key;

}

}

return 0;

}

}

class BookDetails

{

static Dictionary<string[], bool> bookRecords = new Dictionary<string[], bool>();

static bool constantMenu = false;

public static void bookMenu()

{

do

{ //Displays all options available to the Admin relating to managing books

Console.WriteLine(Environment.NewLine + "--------------------" + Environment.NewLine + "BOOK MANAGEMENT MENU" + Environment.NewLine);

Console.WriteLine("Type V to | View All Books" + Environment.NewLine + "Type A to | Add a Book" + Environment.NewLine + "Type D to | Delete a Book" + Environment.NewLine + "Type B to | Go Back" + Environment.NewLine + "Type X to | Exit the Software");

Console.WriteLine(Environment.NewLine);

Console.Write("What would you like to do? ");

string userChoice = Console.ReadLine().ToUpper(); //Stores user's choice

int i = menuChoice(userChoice);

switch (i) //Checks which option was chosen by the user

{

case 1:

viewBooks(); //Method to Display a list of all books in the system

break;

case 2:

addBook(); //Method to add a new book to the system

break;

case 3:

removeBook(); //Method to remove a book from the system

break;

case 4:

AdminControl.adminMenu(); //Return to the Admin Menu

break;

case 5:

Environment.Exit(1); //Close the Software

break;

}

} while (constantMenu == false);

}

static int menuChoice(string choice)

{

Dictionary<int, string> menuChoices = new Dictionary<int, string>() //Checks if user input matches a valid menu choice

{

{1,"V"},{2,"A"},{3,"D"},{4,"B"},{5,"X"}

};

foreach (var option in menuChoices)

{

if (choice == option.Value)

{

return option.Key;

}

}

return 0;

}

static void viewBooks() //Method to display all Books on the system and their information

{

foreach (var book in bookRecords.ToList())

{

isLoaned(book.Key[0]); //Checks if each book is currently loaned out to a customer

Console.WriteLine(Environment.NewLine + "Book: {0} | Author: {1} | Published: {2} | Currently Loaned: {3}", book.Key[0], book.Key[1], book.Key[2], book.Value);

}

}

static bool addBook() //Metod to add a new book to the system

{

Console.WriteLine("Please enter Book's details below:");

Console.Write("Book Name: ");

string bookName = Console.ReadLine().ToUpper();

Console.Write("Author Name: ");

string authorName = Console.ReadLine().ToUpper();

Console.Write("Year of Publication: ");

string pubYear = Console.ReadLine();

foreach(var book in bookRecords)

{

if(bookName == book.Key[0] & authorName == book.Key[1]) //Checks if new book already exists in the system, if so it returns an error

{

Console.WriteLine(Environment.NewLine + "Error | Book is already registered" + Environment.NewLine);

return false;

}

}

string[] newBook = new string[3] {bookName, authorName, pubYear}; //Adds new book to the system

bookRecords.Add(newBook,false);

Console.WriteLine(Environment.NewLine + "Book Succesfully Added" + Environment.NewLine);

return true;

}

static bool removeBook() //Method to remove a book from the system

{

Console.WriteLine("Enter the details of the book to be Removed:");

Console.Write("Book Name: ");

string bookName = Console.ReadLine().ToUpper();

Console.Write("Author: ");

string authorName = Console.ReadLine().ToUpper();

foreach (var book in bookRecords) //Checks if entered details match a book in the system

{

if (bookName == book.Key[0] & authorName == book.Key[1]) //If book has a matchs

{

string[] toRemove = book.Key;

bookRecords.Remove(toRemove); //Remove the book

Console.WriteLine(Environment.NewLine + "Book Succesfully Removed" + Environment.NewLine);

return true;

}

}

Console.WriteLine(Environment.NewLine + "Error | Couldn't Find Book" + Environment.NewLine); //If no match can be found, return an error

return false;

}

public static bool isLoaned(string bookCheck) //Method to check Loaned status of books

{

bool alreadyLoaned = false;

foreach (var book in bookRecords.ToList())

{

if (bookCheck == book.Key[0])

{

alreadyLoaned = LoanDetails.isLoaned(bookCheck); //Checks if book matches a book in the list of loaned books in the Loan Details Class

switch (alreadyLoaned)

{

case true:

bookRecords[book.Key] = true; //Set that the book is already loaned

return true;

case false:

bookRecords[book.Key] = false; //Set that the book is not currently loaned

break;

}

}

}

return false;

}

}

class LoanDetails

{

static Dictionary<string, string[]> loanRecords = new Dictionary<string, string[]>(); //Storage of all loan details

static List<string> loanedBooks = new List<string>(); //List of all currently loaned Books

static bool constantMenu = false;

public static void loanMenu()

{

do

{ //Presents user with all the optons available for managing books

Console.WriteLine(Environment.NewLine + "--------------------" + Environment.NewLine + "LOAN MANAGEMENT MENU" + Environment.NewLine);

Console.WriteLine("Type V to | View All Active Loans" + Environment.NewLine + "Type A to | Loan out a Book" + Environment.NewLine + "Type D to | Delete a Loan" + Environment.NewLine + "Type B to | Go Back" + Environment.NewLine + "Type X to | Exit the Software");

Console.WriteLine(Environment.NewLine);

Console.Write("What would you like to do? ");

string userChoice = Console.ReadLine().ToUpper();//Saves user choice

int i = menuChoice(userChoice);

switch (i) //checks user choice against available options

{

case 1:

viewLoans(); //Output details of all loans

break;

case 2:

addLoan(); //Loan out a book to a customer

break;

case 3:

removeLoan(); //Manually delete a loan

break;

case 4:

AdminControl.adminMenu(); //Return to main Admin Menu

break;

case 5:

Environment.Exit(1); //Exit the Software

break;

}

} while (constantMenu == false);

}

static int menuChoice(string choice) //Method to check user choice is valid and matches a choice in the system

{

Dictionary<int, string> menuChoices = new Dictionary<int, string>()

{

{1,"V"},{2,"A"},{3,"D"},{4,"B"},{5,"X"}

};

foreach (var option in menuChoices)

{

if (choice == option.Value)

{

return option.Key;

}

}

return 0;

}

static void viewLoans() //Method to output all loans and their details

{

foreach(var loan in loanRecords) //Runs until all loans have been outputted

{

Console.WriteLine(Environment.NewLine + "| Reference ID: {0} | Book: {1} | Customer: {2}, {3} | Employee: {4} " + Environment.NewLine + "| Loaned: {5} | Return Due: {6}",loan.Key,loan.Value[0],loan.Value[1],loan.Value[2],loan.Value[3],loan.Value[4],loan.Value[5] + Environment.NewLine);

}

}

static bool addLoan() //Method to add a new loan to the system

{

Console.Write("Book being Loaned: ");

string loanBook = Console.ReadLine().ToUpper(); //Book to be loaned

bool loaned = BookDetails.isLoaned(loanBook); //checks if book is currently loaned

string[] loanCustomer = new string[2];

Console.WriteLine("Who is Loaning the book?");

Console.Write("First Name: ");

loanCustomer[0] = Console.ReadLine().ToUpper();

Console.Write("Surname: ");

loanCustomer[1] = Console.ReadLine().ToUpper();

bool customerMatch = CustomerDetails.lookupCustomer(loanCustomer); //Checks if customer is registered on the system

string currentEmployee = LoginDetails.currentUser(); //Saves current user as employee loaning the book

DateTime loanOut;

loanOut = DateTime.Now; //Sets current date as when the book is loaned

string loanoutString = loanOut.ToShortDateString();

DateTime loanIn = loanOut.AddDays(21); //Generates a date for when the book needs to be returned

string loaninString = loanIn.ToShortDateString();

string newID = Guid.NewGuid().ToString(); //Generates a unique 5 digit ID for the loan as refrence

char[] idCharacters = newID.Take(5).ToArray();

newID = new string(idCharacters);

if (loaned) //if the book is already loaned

{

Console.WriteLine("Error | Book is Already Loaned"); //Display error message and abort the method

return false;

}

if (customerMatch == false) //If loan customer is not registered on the system

{

Console.WriteLine("Error | Customer not Registered"); //Display error message and abort the loan

return false;

}

string[] loanString = new string[] {loanBook, loanCustomer[1], loanCustomer[0], currentEmployee, loanoutString, loaninString};

loanRecords.Add(newID, loanString); //Add new loan to the system

loanedBooks.Add(loanBook); //Add loaned book to list of books currently loaned

Console.WriteLine("Book Succesfully Loaned");

return true;

}

static bool removeLoan() //Method to remove a loan from the system

{

Console.Write("Enter Loan Reference ID: ");

string searchID = Console.ReadLine();

foreach (var loan in loanRecords)

{

if (searchID == loan.Key) //If entered unique ID matches a loan on the system

{

loanedBooks.Remove(loan.Value[1]); //set loaned book as returned

loanRecords.Remove(searchID); //remove loan from the system

Console.WriteLine(Environment.NewLine + "Loan Succesfully Removed" + Environment.NewLine);

return true;

}

}

Console.WriteLine(Environment.NewLine + "Error | Couldn't Find Loan" + Environment.NewLine); //if Unique ID match cannot be found, display error message

return false;

}

public static bool isLoaned(string bookCheck) //Method to check if book is currently loaned

{

foreach (var book in loanedBooks) //checks every book in list of loaned books

{

if (bookCheck == book) //If searched boom is matched, return that it is loaned

{

return true;

}

}

return false; //otherwise, return that it is available

}

};