Toby bluck

BLU16274423

Word Count

6522 Words

SCDT41\_CW1

Programming Portfolio

# Task 1 –

## Decomposition –

A booking system, which allows for:

Pre-booked passengers.

Pay on arrival passengers.

Total passengers.

Record all the pre book and pay on arrival and display them as a total for each coach.

## Requirements –

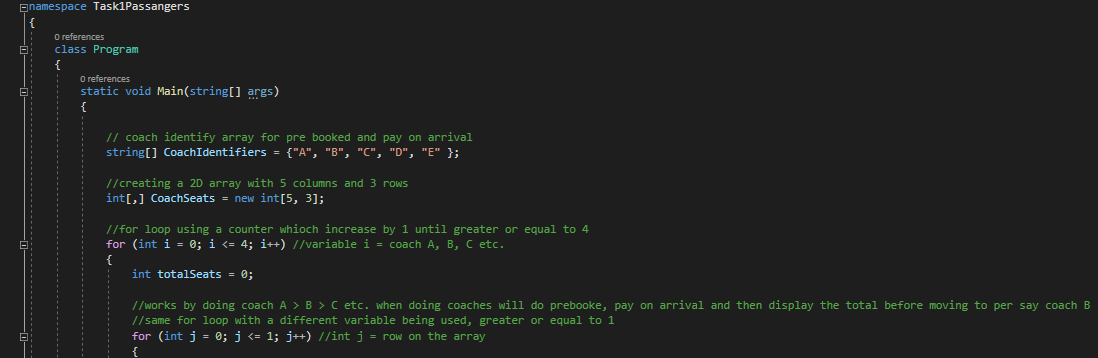
Be able to enter the number of passengers for pay on arrival and pre booked per coach.

Go in sequential order for coaches.

Take the 2 values and add them together to give you the total amount.

Have an appropriate output.

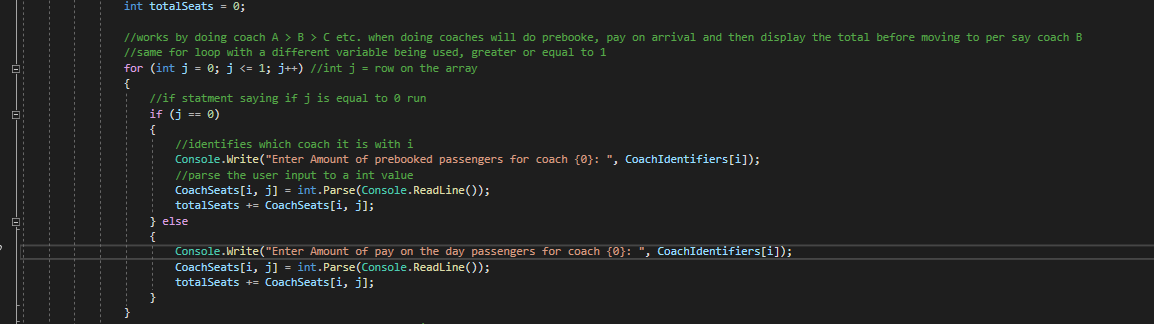
## Screenshots & Annotations



In this task, an array was created using a string, this was called CoachIdentifiers, which stored the values of the five coaches from: A -B. A 2D array was then created specifying the use of integer values called CoachSeats. This 2D array has 6 columns and 3 rows. To create a image of it, it would be similar to this.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

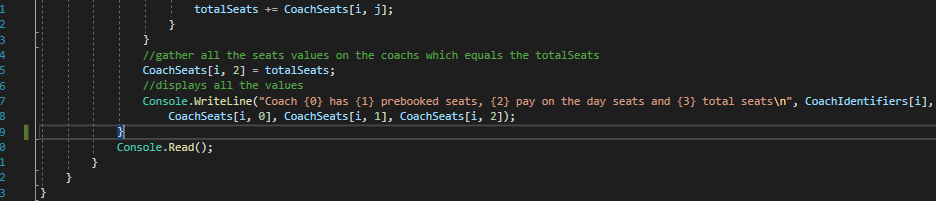
The for loop is then used with a local variable called “I” which allows it to function as a counter and count the letters in the coach identifier. The totalSeats has been declared at 0, if it started at 1 or another number it would create an offset when pre book and pay on arrival passengers are added up. The for loop nested within the if statement is used for the row selection, this is fundamentally the same as the if statement only with a different local variable. This declares the local variable as j making it unique but also, it only goes less than or equal (<=) to 1.



Nested in the for loop, there is an ‘if’ statement which condition is (j == 0) which means that if j has the value of 0, run this if it doesn’t then run the ‘else’ statement. When the ‘if’ statement runs, it writes a line on the console asking to enter the amount of passenger for pre booked. With the coachIdentifiers variable which will display A – E depending how many loops it has made.

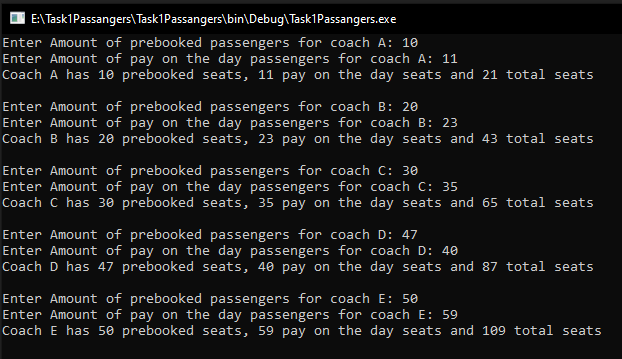
coachSeats[I,j] is the used to parse a console.readline which turns the input of a user into an integer value, the ‘I’ defines which coachIdnetifier it will fall under and ‘j’ to store the inputted value. totalSeats += CoachSeats[I,j] is equivalent to: totalSeats = totalSeats + coachSeats[I,j] which means it adds what has just been inputted.

After that it switches to the ‘else’ statement which as ‘j’ would be +1 with the for statement due to ‘j++’. Within this it runs the amount of pay on the day passengers with the coachIdentifier being displayed. It then parses the inputted value and stores in the coachSeats[I,j] the same as the ‘if’ statement. This also adds the totalSeats as well.



After the it has added the total seats from the loop it declares that the totalSeats is to be stored in the array at position [I,2]. Then a writeline is used to display the: coach it was, the amount of pre booked seats, the amount of pay on arrival seats and then the total amount of seats.

### The application running outputs this:



# Task 2

## Decomposition –

A spelling quiz:

Questions to show.

Answers to show.

Check student inputs.

## Requirements –

Display the questions to the students.

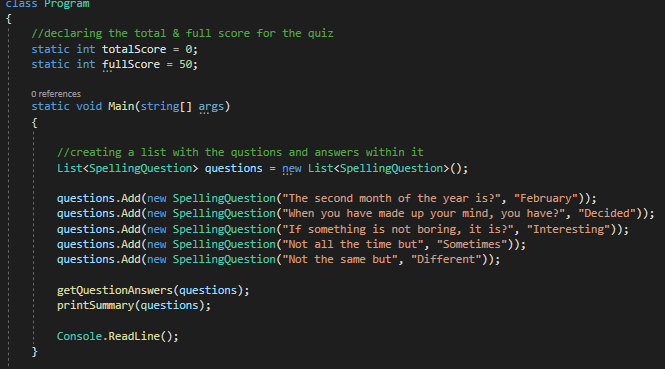
Record student input for answers.

Compare answers against correct answers.

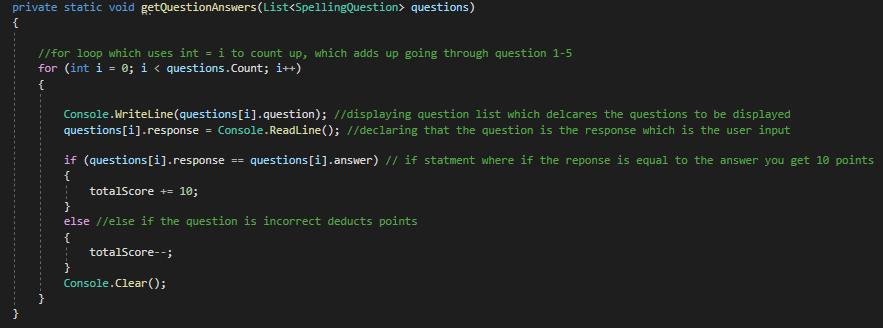
Correct answers give +10 points.

Incorrect answers remove 1 point per incorrect letter.

## Screenshots & Annotations

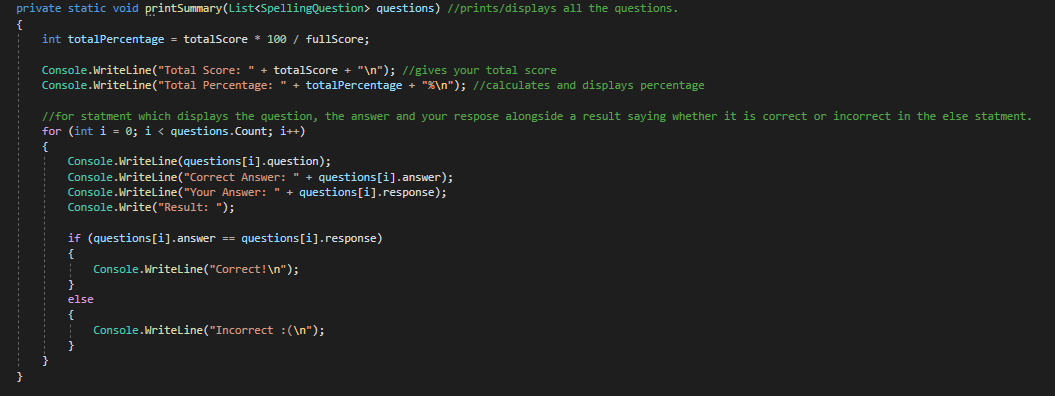


In this program, there are two int values specified in the class program to be used throughout the program. A list was created to store the questions and answers to the questions with the name SpellingQuestions. getQuestionAnswers is in the main args for it to run, this helps keep everything neat and organised and run smoothly. Same applies for printSummary.

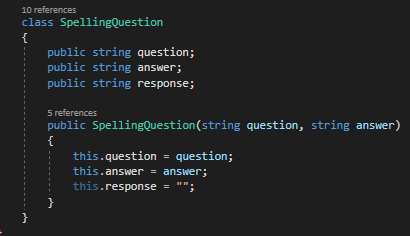


getQuestionAnswers is derived from the list questions. Within this, there is a ‘for’ loop which is used to count the questions, the I int allows it to could up using i++ to the question.Count. Within the ‘for’ loop a writeline is used to display the questions by using (questions[i].questions) the ‘questions[i] is for the counting of the questions and the .questions is the actual question itself to be displayed.

Applying the same method with questions[i] a response is also being recorded by user input by the readline. Nested in the ‘for’ loop there is an ‘if’ statement in which the condition is that if the response (user input) is equivalent to the answers being stored then, the totalScore += 10 which adds 10 points for the scoring system. If the response does not equal the answer then totalScore – which for each letter that is not correct it removes 1 as it is a decrement operator(--). After the question is complete it clears the CLI and a new question is then displayed as ‘i++’ runs again.

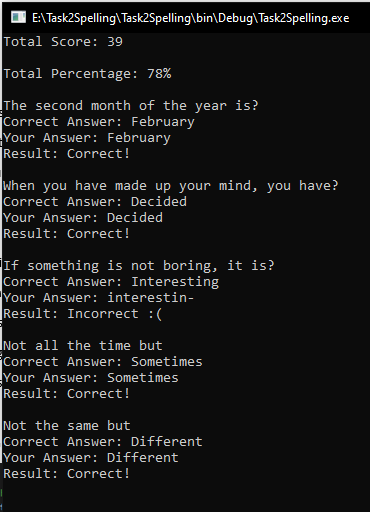


printSummary is used to display the final results. Within this the local variable totalPercentage is declared with its calculations which will return the value of the score as an int value. Writeline is then used to show the total score the user has achieved and then start a new line. totalPercentage is then used to show the percentage achieved after the score, which also starts a new line after. Using the same conditions for the ‘for’ statement as in getQuestionAnswers, this is used to display the question, the user response and then the answer. The result below will run an ‘if’ statement where if the response is equal to (==) the answer then in the result put Correct! If it is not equal to it puts Incorrect ☹.



This is the class for the spellingQuestions where the values are stated and then parameters are being declared.

### Output



# Task 3

## Decomposition –

Login system.

Username and password.

Allow authenticated users.

3 attempts to login

Error for incorrect credentials

## Requirements –

A set number of users with user types

Admin can add or remove users

Admin can block or unblock users

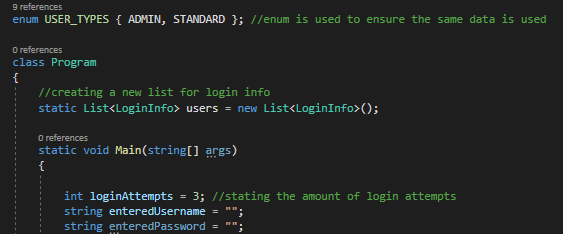
3 login attempts before being locked out, admin unblock

The attempts to be displayed

Catch errors with incorrect credentials

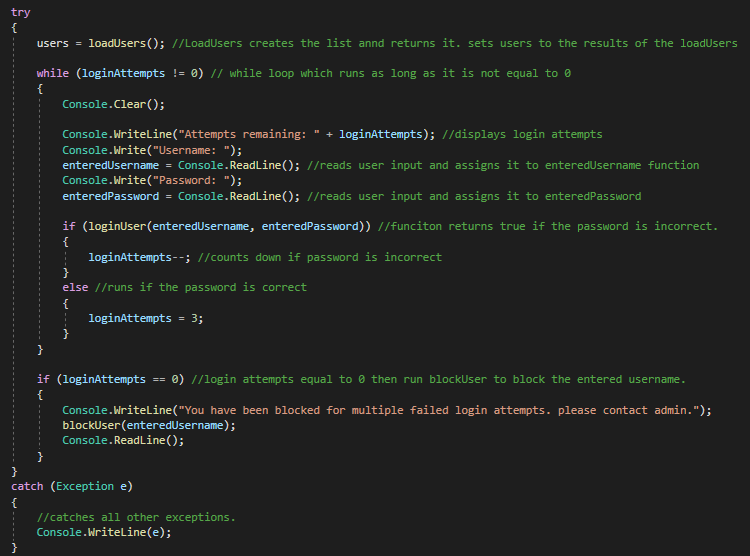
Welcome message when logged in

## Screenshots & Annotations



First of we declare a enumeration called USER\_TYPES which allows us to pick whether they are an admin or standard user, this was chosen to ensure that the data used is consistent within the program and then within the class Program a new list is made called users from the LoginInfo class.

Within the main args we declare loginAttempts equals 3 and that the enterUsername/Password hold nothing.

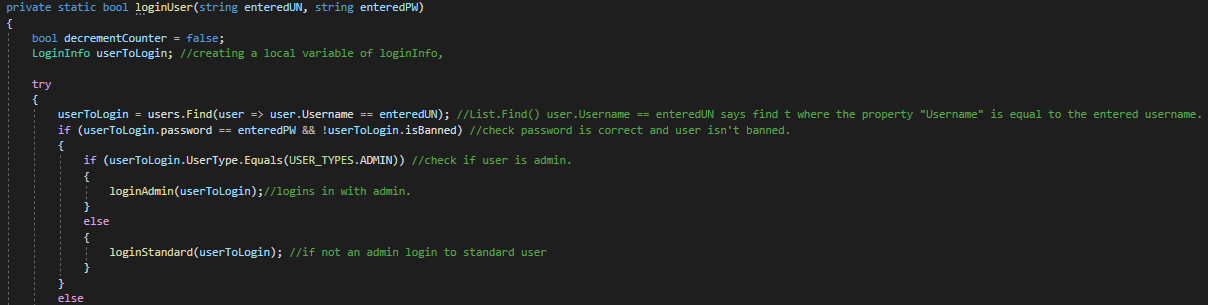


Here we say users = load users which is calling on the static function loadUsers which is used to populate the users list with login information. This allows for users to equal the results of loadUsers.

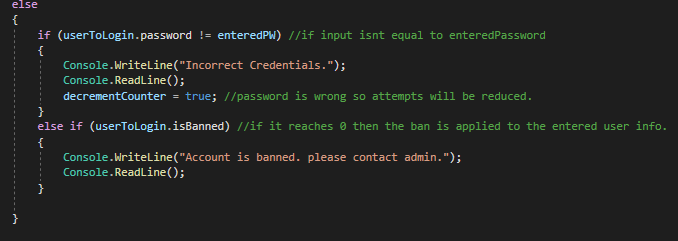
A while loop then runs with the condition that as long as lognAttempts are not equal (!=) 0 it will allow for the loop to run. Console is then cleared, and then the login attempts is displayed to the user showing the amount remaining dependant on whether they have tried to login. Entered Username/Password are both readlines allowing for user input. An if statement runs in that where its condition is run userLogin for enteredUsername/Password which returns true if the password is incorrect. If it comes back true the login attempts is reduced by one meaning there would be 2 attempts left before being blocked. If the password come back correct the loginAttempos goes back to its original value.

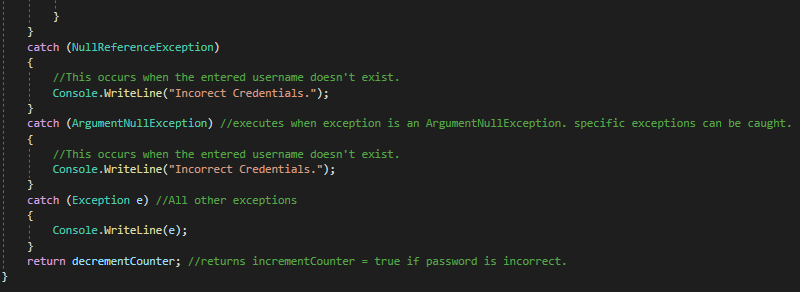
The ‘if’ statement that is not nested within the while loop, sets a condition in which that when the loginAttempts are equal to 0, it blocks the user by running the block user method, with the enterUsername being identified and applying it to that.

Furthermore, the ‘catch’ exception allows for all other exceptions to be caught.

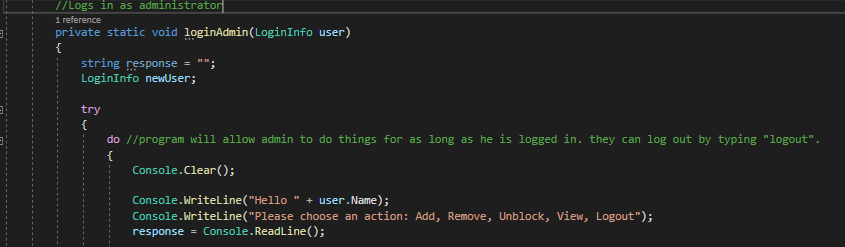


This is the loginUser method and what it does, it specifies a bool which is called decrementCounter which is set to false and using LoginInfo as a local variable called userToLogin. UserToLogin is called which should equal users which uses loadUsers, to find a user with searching the user details and when the username is equal to the enteredUSername, the checks what type of user is logging in using a ‘if’ statement, it condition is that if the userToLogin userType equals the USER\_TYPE ADMIN, if that is correct then it runs loginAdmin with the userToLogin. If they do not have the user type ADMIN then the ‘else’ statement runs and the loginStandard is used for the userToLogin. This is the method to validate the username entered.

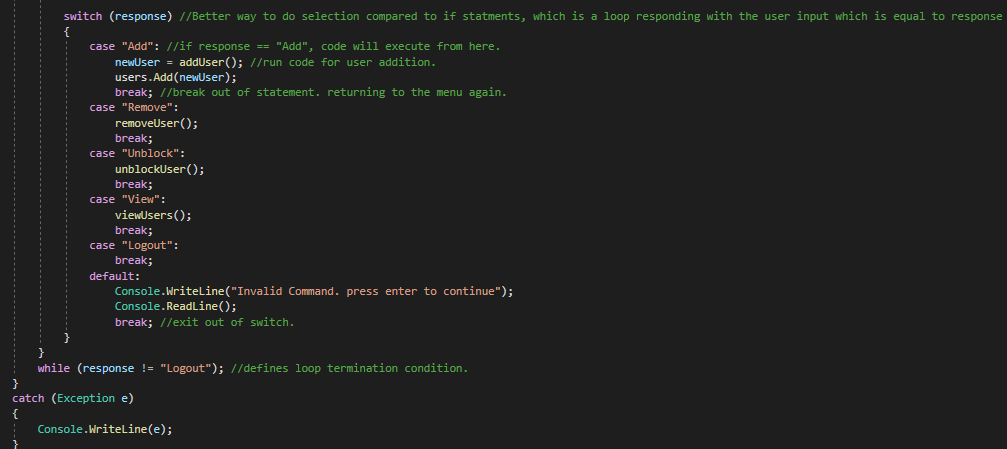
After the username has been verified it moves to the ‘else’ statement this is used for the checking and validation of the entered password. The ‘if’ statement runs the condition userToLogin.password is not equal to enteredPassword, run the if statement which displays that incorrect credentials have been entered with a readline and switches the decrement counter to true which causes the login attempts to be reduced. The ‘else if’ statement will only run when the previous condition is not met, which means if the password was correct it would then run this statement with the isBanned condition banning the userToLogin.



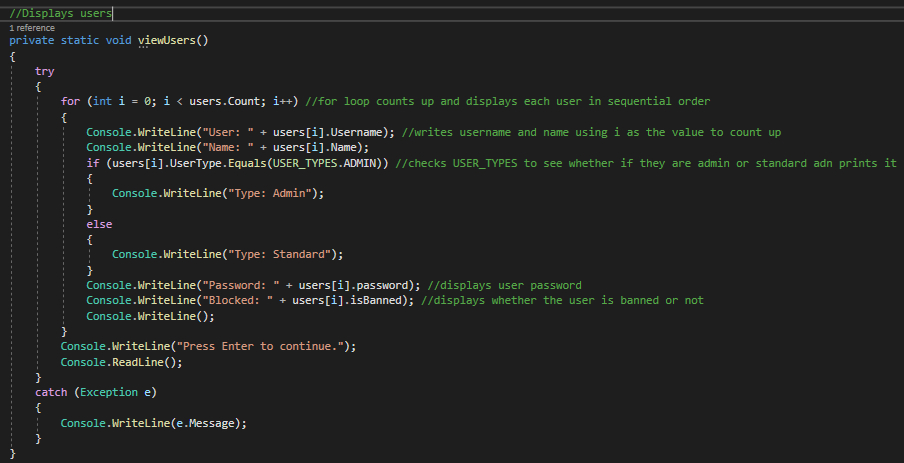
These are the error detection being used, NullReferenceException is thrown when trying to access a value which is null. . The Exception e is used to collect any other exceptions being thrown out and writes them.



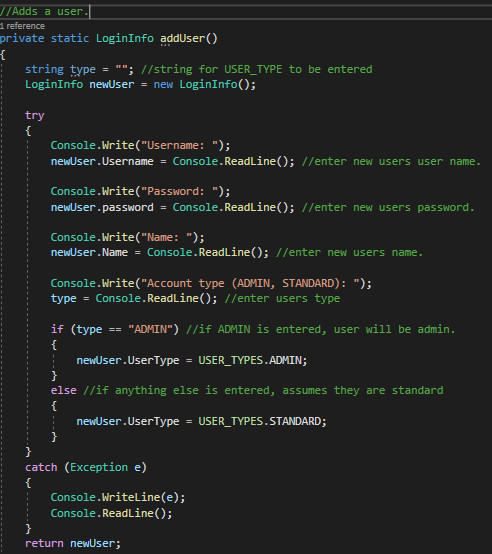
The login admin method is ran when the user input is equal to the admin login info, it is encased in a ‘do while’ loop, which will keep looping until condition are met to terminate it. The console is cleared and then displayed is the name of the admin and another line is written telling you the options to choose from. The response is a local variable for this where it equals user input.



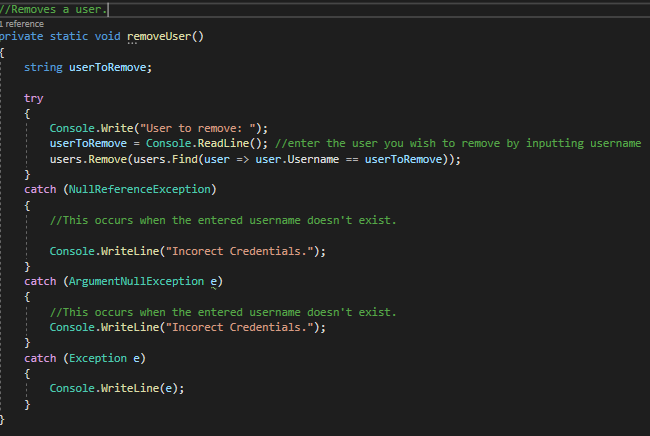
A ‘switch’ statement is then used, which is functionally the same as using ‘if else’ but is much more effective for when there is more than 3-4 choices. ‘switch’ statement is much more readable as well for larger menus, this uses the condition response as a condition. The switch selections are called a case, the first one specifies that if the response is equal to “Remove” then it will run the remoceUser method then a break which uses ‘break’ to break out of the loop which will loop it back to the menu again as it is encased in a ‘do while’ loop. Default is there to make sure that there is a backup for an invalid command or misspelt word such as if the user inputted “Adddd”. The ‘while’ specified the condition in which to stop to loop if the ‘response’ is not equal to “Logout” it will keep looping.



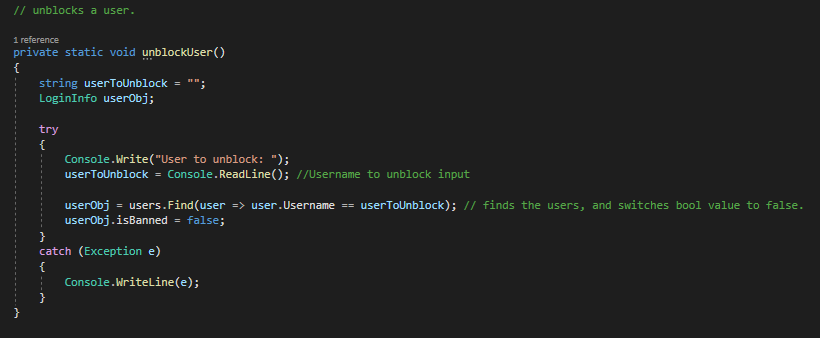
viewUsers uses a for statement with I to count up and display the users in a sequential order, it displays the user name and then the name using I to sequentially go through the users. Then a ‘if’ statement is used to display what user type they are, the condition set for the ‘if’ statement is that is the user.userType equals ADMIN then it displays Admin. If it does not meet that condition it displays Standard. Then it continues to show the password and whether the user is named or not. After that it displays a message to “press enter to continue” with a readline to record the input.



This is used to add a new user to the program a string is declared which is used for the USER\_TYPE and specifying that a newUser is equal to new LoginInfo. It then writes username, password, name and account type which uses newUser.---- = Readline which records the user input. Once the user type is enter a ‘if’ statement is used to apply the correct user type, the condition used is that if the input is equal to “ADMIN” apply the “ADMIN USER\_TYPE” else, they will be set to a “STANDARD” user type and then return the newUser.

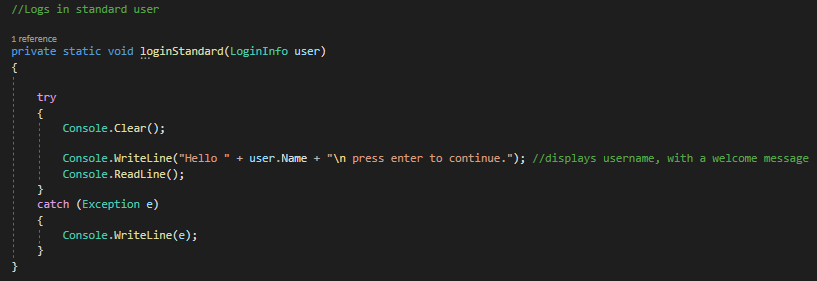


RemoveUser is there to remove users, userToRemove has been made as a string. When this runs it asks what user to remove then userToRemove is set to readline which is the user input. once the input has been recorded it compares it runs users.Remove then the condition runs which matches up the userToRemove to the username, if that is equal to the username they get removed from the users. Error handling is used in this.

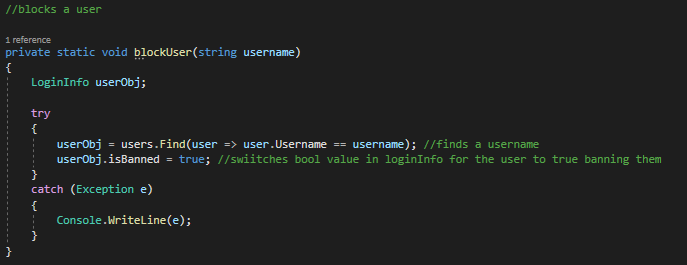


UnblockUsers is used to remove a that has been applied to a user. userToUnblock is a string format which is going to be used to store user input and then spcifiying LoginInfo as userObj.

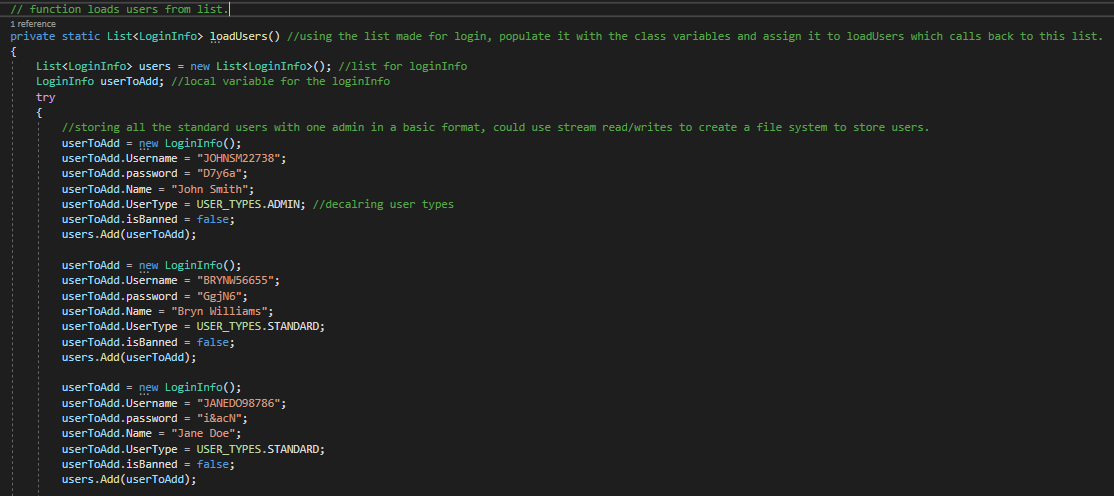
When thus runs it displays a message asking what user should be unblocked. userToUnblock then stores the user input and then, users.Find is used to find the username by comparing the user.Usernames which are stored in the users if the Username is equal to userToUnblock it then applies to the userObj. userObj is then used to check whether they are banned as it is storing the value and then changes the isBanned bool type from true to false.



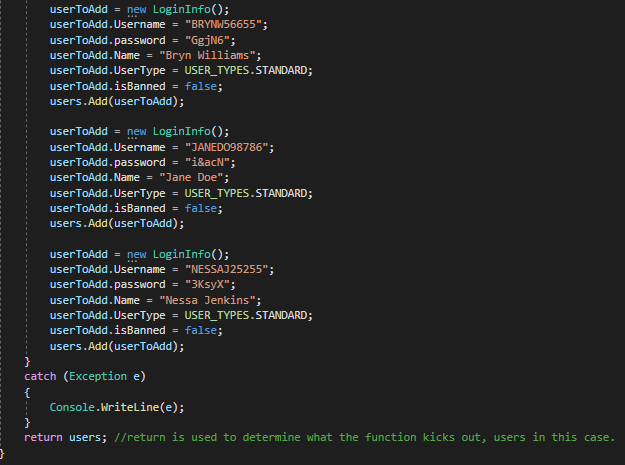
this is what runs if the logs into a standard use, it clears the console and then writes the username with a welcome message.



This is used when the admin wants to block a user. It uses userObj to store a value from loginInfo. it then finds a username and if that is equal to the users.Username, it then switches the isBanned bool to a value of true, this is the method that is ran when the loginAttempts == 0.

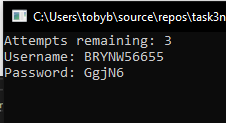


This is what users use when users = loadUser. Its stored as a list and uses usersToAdd as a local variable, here is where the new predefined users are stored for the programs, with the declaration of the username, password, name, userType and isBanned

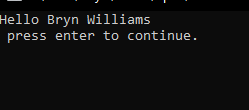


There is basic error handling to catch any exception and return kicks out the users after this.

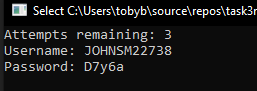
### Logging as a standard user

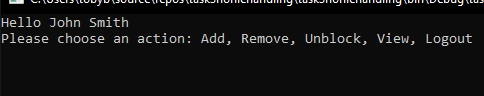


### Welcome screen

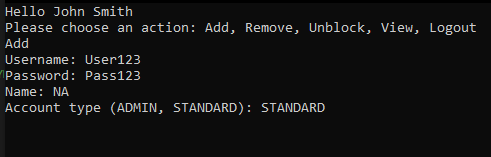


### Admin login

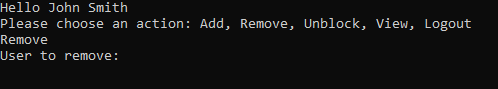




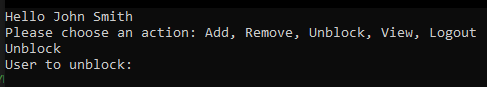
### Add



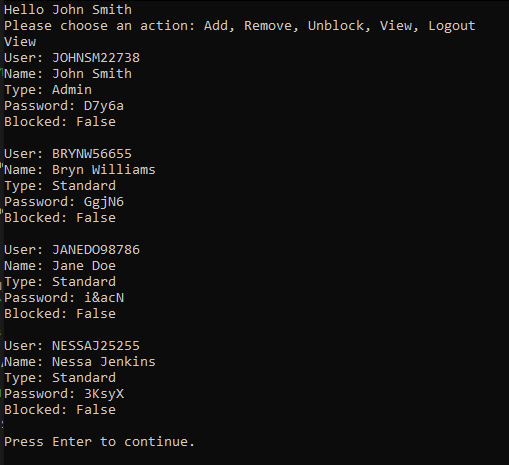
### Remove



### Unblock



### View



# Task 4

## Decomposition

Admin can manage

Customers

Books

Loans

Admin should; add and retrieve customer details, book details, loan books out, retrieve loans

## Requirements

Admins can

Add a book with a unique ID

Delete a book with a unique ID

Find a book with a unique ID

Add a Customer with a unique ID

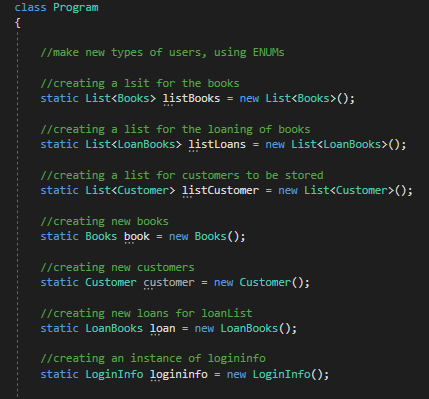
Delete a Customer with a unique ID

Find a customer with a unique ID

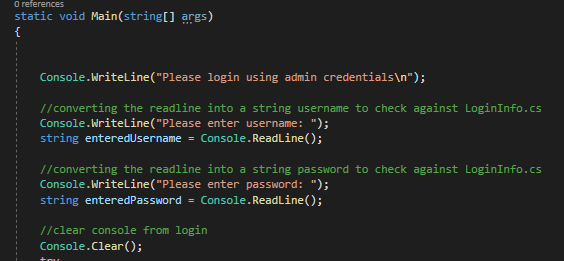
Add a loan with a customer and book

Find a loan with a unique ID

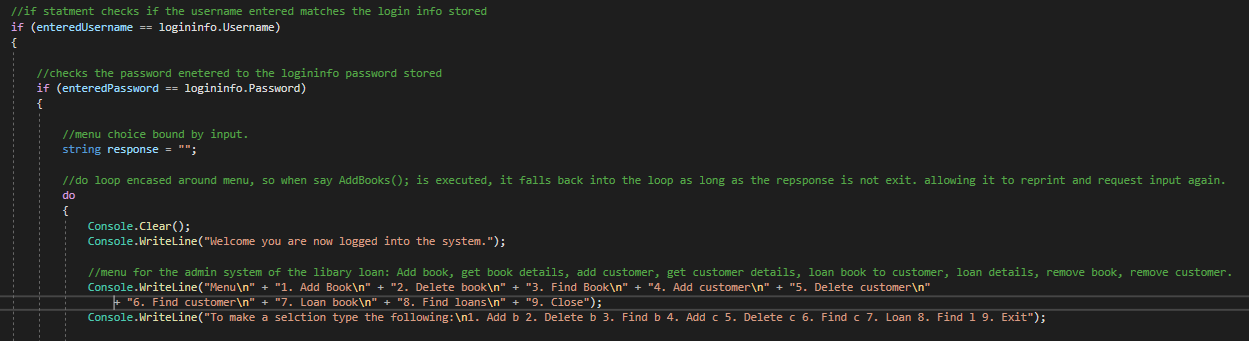
## Screenshots and annotations



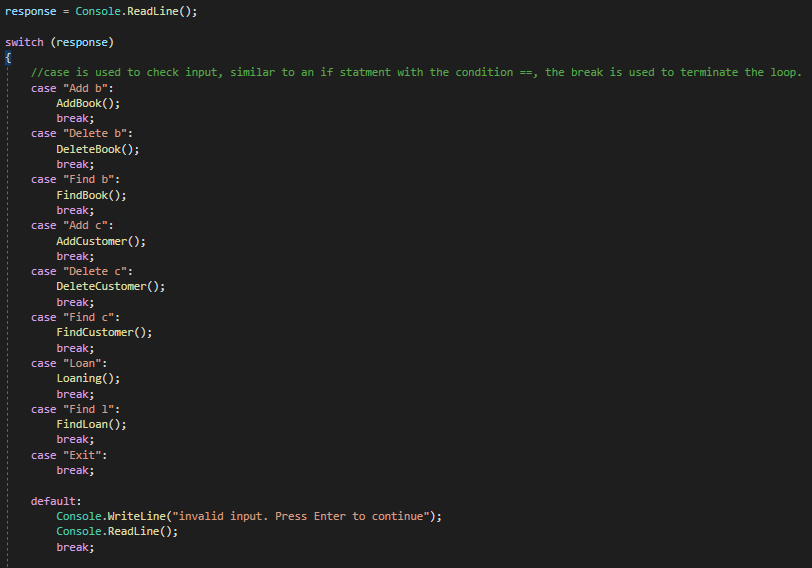
This is creating new lists and instances



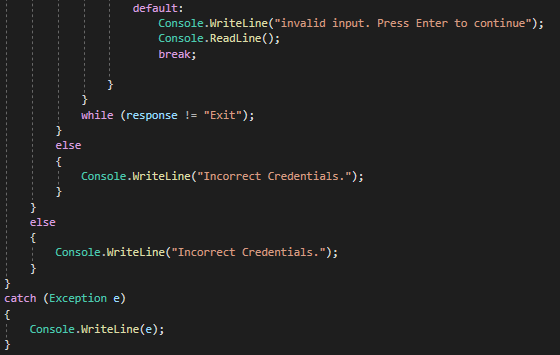
The start of the program give you a welcome message asking to login with admin credentials, it then asks for a username, a sting called enteredUsername is used to store user input and the same is done for the password. When enter is pressed the console is cleared.



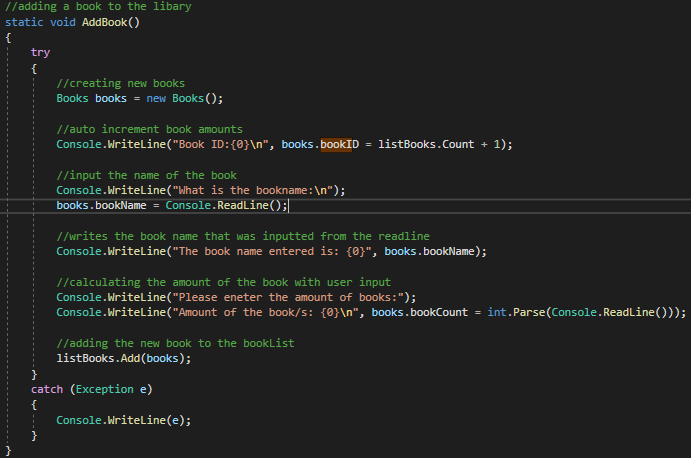
It then runs an if statement with the condition that the enteredUsername is equal to the logininfo.Username where the admins password is stored, if that works moves on to the enteredPassword, which should be equal to logininfo.Password. if that is accepted it continues on.



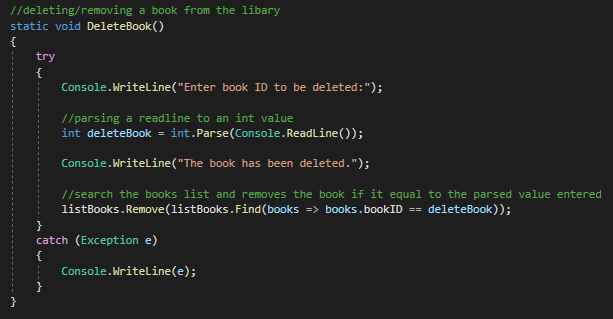
Response is declared and equals a readline which is the user input. A switch statement is then used for the menu with the condition depending on the response variable which is the user input. Typing “Add b” will run the AddBook method which adds a book. Exit will use break to terminate the program.



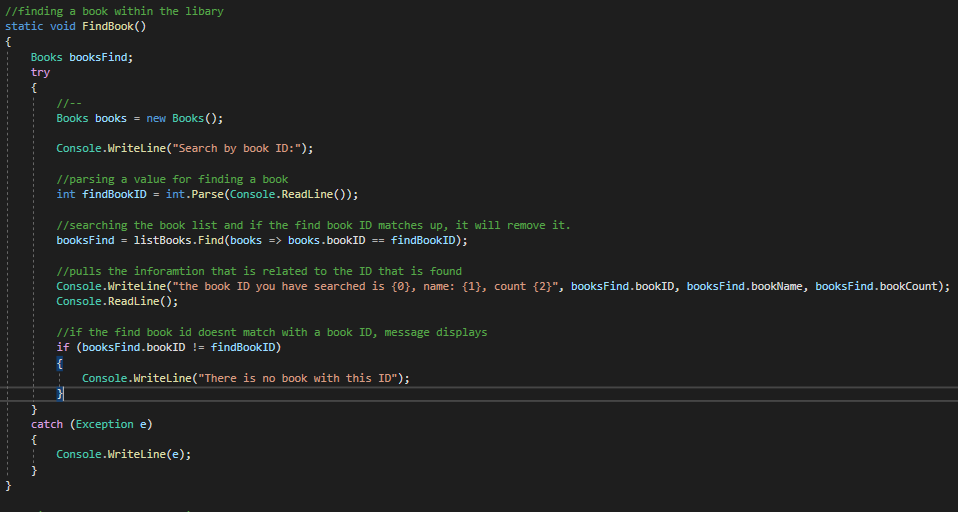
The reason why “Exit” will terminate the loop is because its nested in a ‘do while’ loop and while response isn’t equal to Exit it will continue with the do loop.



This is the add book method books is used as a variable to make a new book then uses bookID equals listBook.Count + 1 as a value increase to make a unique identifier. Then the book name is asked and books.bookName is equal to the user input through a readline. Then it displays the name for the book. The amount of books is then asked to be specified which is a parsed user input. Then the book is added to the listBooks to be pulled when finding them.

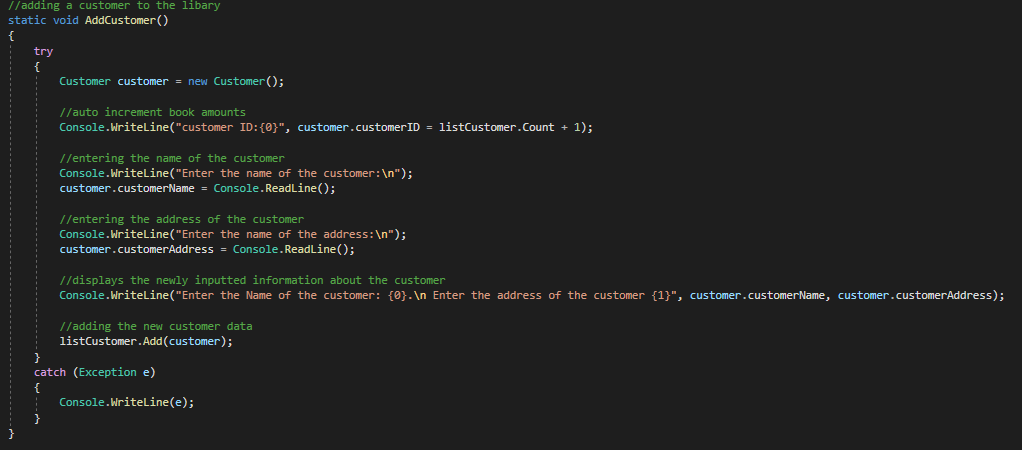


This is to delete a book that has been made, this asks you to enter an id for a book to be deleted, the value is parsed and the stored into deleteBooks. The process to delete it is remove function is added to the listBooks, then uses the find function and compares bookID to the value stored in deleteBook, if the values match the entry is removed.

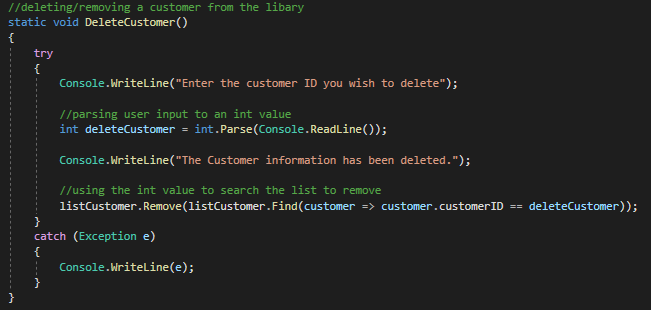


Booksfind is used as a local variable for Books. The writeline asks for the book ID,

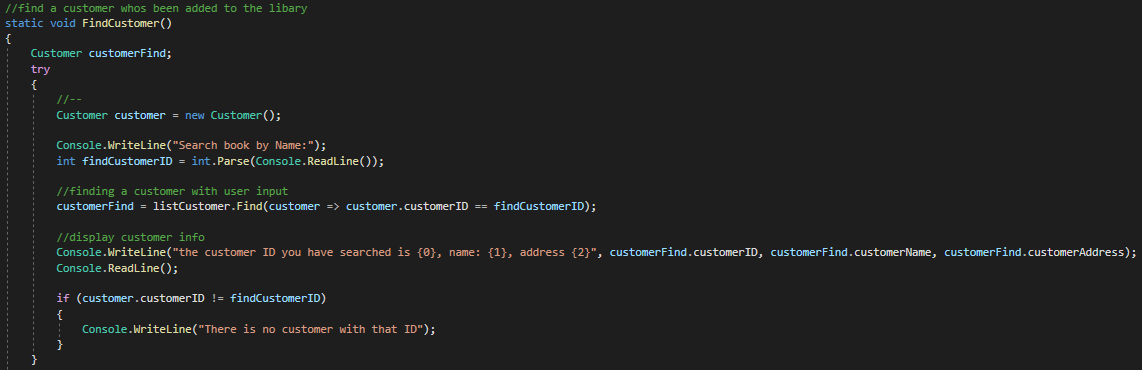
findBookID is then used to store the parsed user input from the readline. We use the booksFind = listBooks.Find to search for the books and within the ‘()’ if states that if the bookID is equal to findBookID, then a writeline is used to display the values it finds. The if statement is used when the findBookID is not equal to BookID and throws a message saying that there is no book with that ID. Basic exception catching for debugging.



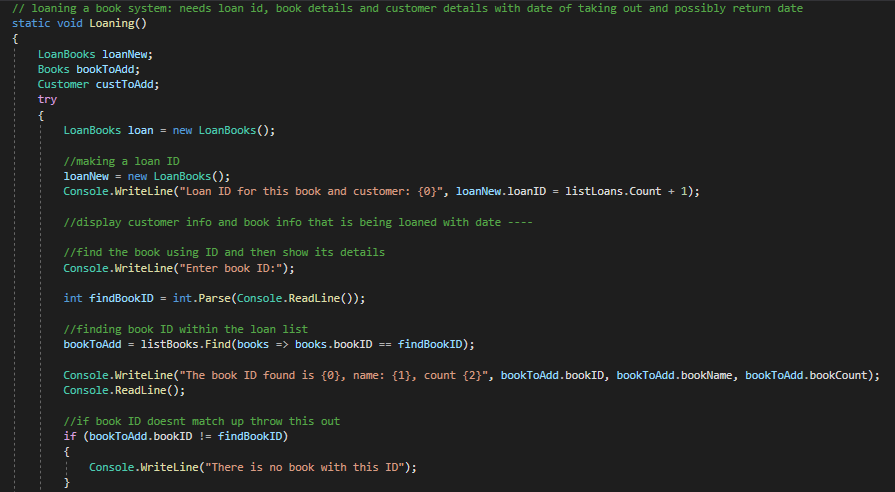
AddCustomer functions the same as AddBook, however there is still a unique ID but it requires the customer name and address which is then stored in the listCustomer.



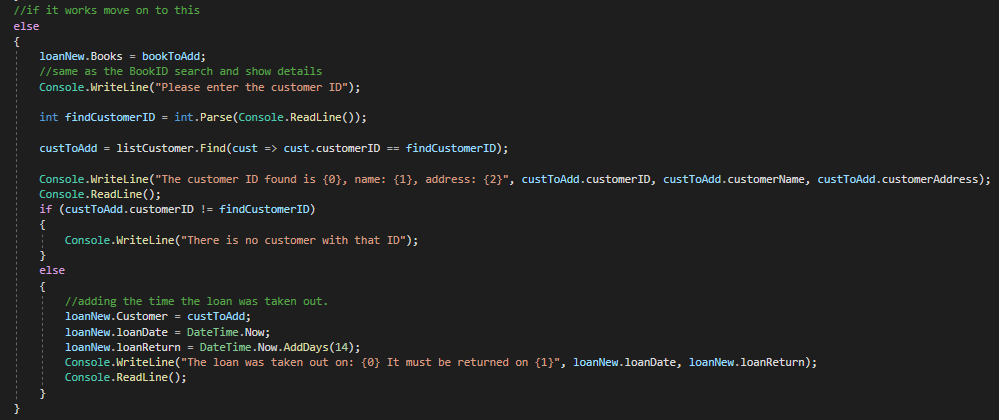
This is functionally the same as the DeleteBook, the message it writes when the customer is deleted is different.



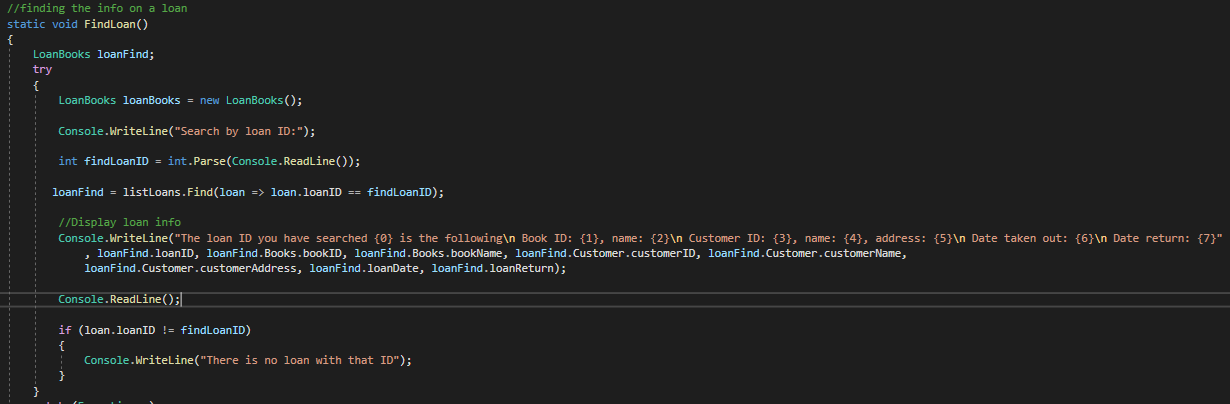
This is the same as FindBook, just with the customer information being displayed through the search of their unique ID.



This is the loaning system that has been made, which calls on loanBooks, Books, Customers to get their information without this a loan cannot be made. We use findBookID to search the list of books, and with that it is then stored into bookToAdd and then the information for it is then displayed. An if statement is run where the bookID is not equal to findBookID it display, there is no book with this ID.

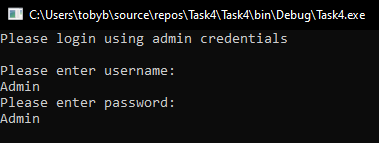


If the if statement matches up it moves onto the else statement which deals with the customers and the date in which the loan is taken out and returned. We use loanNew.Books = booksToAdd which will store the values to add in the loan list. Then enter customer ID is displayed, the user input is pared and set as findCustomerID. custToAdd is used to find the ID within the listCustomer, equal to findCustomerID. It then writes the customer information out, however if it is not equal to the customerID it says there is no customer with that ID. After the customer has been validated, it will move onto adding the customer to the loanNew.Customer. then the dateTime.Now is recorded and the returnDate uses DateTime.Now.AddDays(14) which will add 2 weeks onto current time.

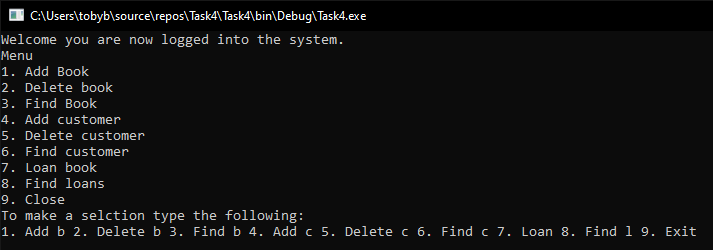


This is functionally the same as find books, with a different local variable, however it searches with the loan ID, and when it does it prints the, loanID, bookID, bookName, customerID, customerName, customerAddress, loanDate and the loanReturn. An ‘if’ statement is used to show if there is no loanID matching findLoanID.

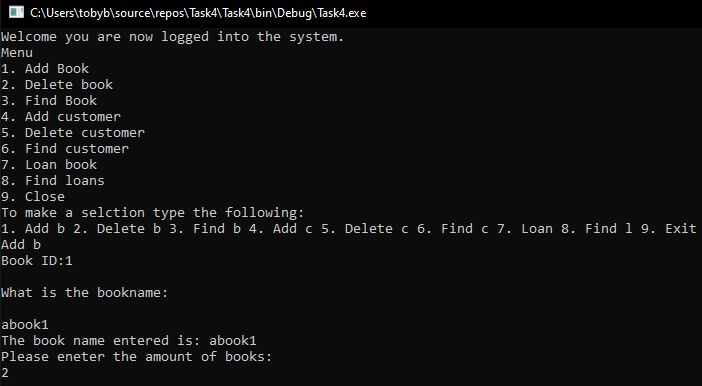
### Login



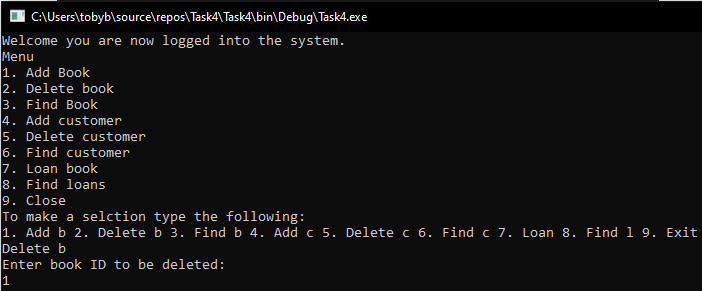
### Menu

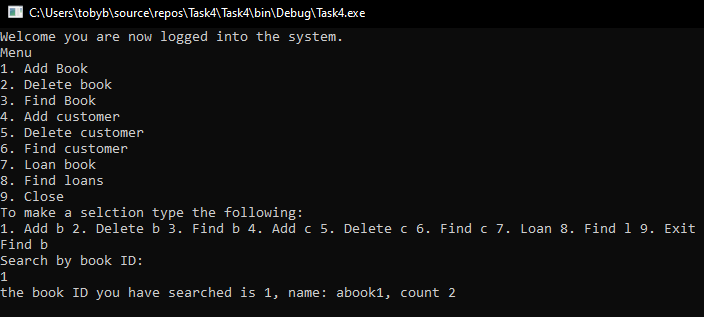


### Add book

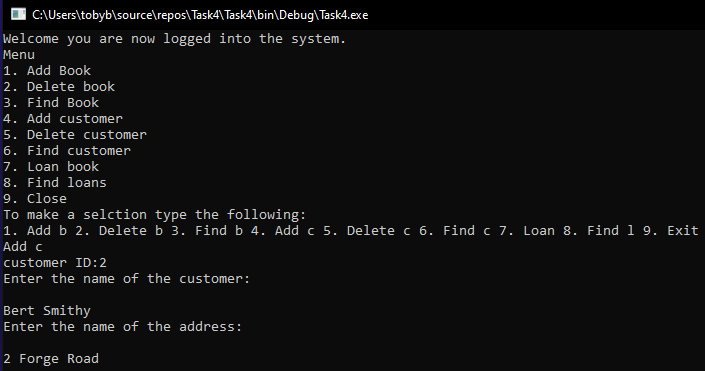


### Delete book

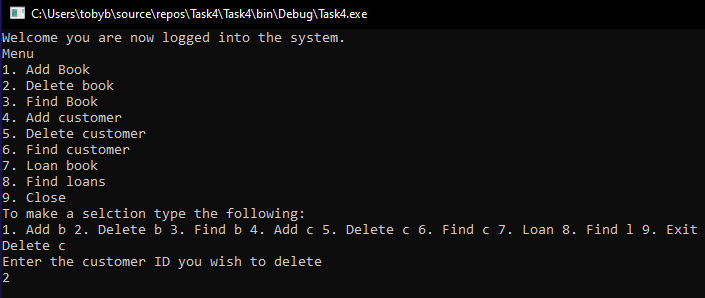


Find book 

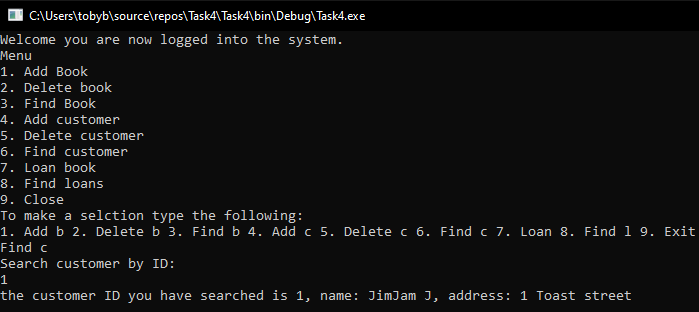
### Add customer



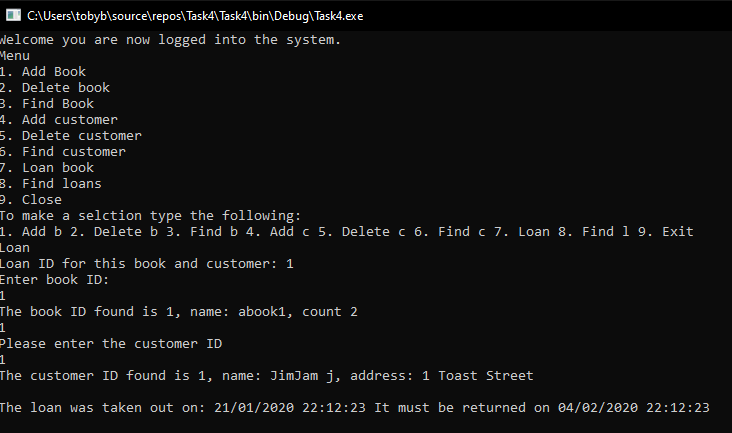
### Delete Customer



### Find Customer



### Loan



Find Loan

# Program code -

## Task 1 –

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task1Passangers

{

class Program

{

static void Main(string[] args)

{

// coach identify array for pre booked and pay on arrival

string[] CoachIdentifiers = {"A", "B", "C", "D", "E" };

//creating a 2D array with 5 columns and 3 rows

int[,] CoachSeats = new int[5, 3];

//for loop using a counter whioch increase by 1 until greater or equal to 4

for (int i = 0; i <= 4; i++) //variable i = coach A, B, C etc.

{

int totalSeats = 0;

//works by doing coach A > B > C etc. when doing coaches will do prebooke, pay on arrival and then display the total before moving to per say coach B

//same for loop with a different variable being used, greater or equal to 1

for (int j = 0; j <= 1; j++) //int j = row on the array

{

//if statment saying if j is equal to 0 run

if (j == 0)

{

//identifies which coach it is with i

Console.Write("Enter Amount of prebooked passengers for coach {0}: ", CoachIdentifiers[i]);

//parse the user input to a int value

CoachSeats[i, j] = int.Parse(Console.ReadLine());

totalSeats += CoachSeats[i, j];

} else

{

Console.Write("Enter Amount of pay on the day passengers for coach {0}: ", CoachIdentifiers[i]);

CoachSeats[i, j] = int.Parse(Console.ReadLine());

totalSeats += CoachSeats[i, j];

}

}

//gather all the seats values on the coachs which equals the totalSeats

CoachSeats[i, 2] = totalSeats;

//displays all the values

Console.WriteLine("Coach {0} has {1} prebooked seats, {2} pay on the day seats and {3} total seats\n", CoachIdentifiers[i],

CoachSeats[i, 0], CoachSeats[i, 1], CoachSeats[i, 2]);

}

Console.Read();

}

}

}

## Task 2

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task2Spelling

{

class Program

{

//declaring the total & full score for the quiz

static int totalScore = 0;

static int fullScore = 50;

static void Main(string[] args)

{

//creating a list with the qustions and answers within it

List<SpellingQuestion> questions = new List<SpellingQuestion>();

questions.Add(new SpellingQuestion("The second month of the year is?", "February"));

questions.Add(new SpellingQuestion("When you have made up your mind, you have?", "Decided"));

questions.Add(new SpellingQuestion("If something is not boring, it is?", "Interesting"));

questions.Add(new SpellingQuestion("Not all the time but", "Sometimes"));

questions.Add(new SpellingQuestion("Not the same but", "Different"));

getQuestionAnswers(questions);

printSummary(questions);

Console.ReadLine();

}

private static void getQuestionAnswers(List<SpellingQuestion> questions)

{

//for loop which uses int = i to count up, which adds up going through question 1-5

for (int i = 0; i < questions.Count; i++)

{

Console.WriteLine(questions[i].question); //displaying question list which delcares the questions to be displayed

questions[i].response = Console.ReadLine(); //declaring that the question is the response which is the user input

if (questions[i].response == questions[i].answer) // if statment where if the reponse is equal to the answer you get 10 points

{

totalScore += 10;

}

else //else if the question is incorrect deducts points

{

totalScore--;

}

Console.Clear();

}

}

private static void printSummary(List<SpellingQuestion> questions) //prints/displays all the questions.

{

int totalPercentage = totalScore \* 100 / fullScore;

Console.WriteLine("Total Score: " + totalScore + "\n"); //gives your total score

Console.WriteLine("Total Percentage: " + totalPercentage + "%\n"); //calculates and displays percentage

//for statment which displays the question, the answer and your respose alongside a result saying whether it is correct or incorrect in the else statment.

for (int i = 0; i < questions.Count; i++)

{

Console.WriteLine(questions[i].question);

Console.WriteLine("Correct Answer: " + questions[i].answer);

Console.WriteLine("Your Answer: " + questions[i].response);

Console.Write("Result: ");

if (questions[i].answer == questions[i].response)

{

Console.WriteLine("Correct!\n");

}

else

{

Console.WriteLine("Incorrect :(\n");

}

}

}

}

class SpellingQuestion

{

public string question;

public string answer;

public string response;

public SpellingQuestion(string question, string answer)

{

this.question = question;

this.answer = answer;

this.response = "";

}

}

}

## Task 3

### LoginInfo.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task3LoginSystem

{

class LoginInfo

{

public string Name = "";

public string Username = "";

public Enum UserType = USER\_TYPES.STANDARD;

public string password = "";

public bool isBanned = false;

public LoginInfo()

{

}

}

}

### Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace Task3LoginSystem

{

enum USER\_TYPES { ADMIN, STANDARD }; //enum is used to ensure the same data is used

class Program

{

//creating a new list for login info

static List<LoginInfo> users = new List<LoginInfo>();

static void Main(string[] args)

{

int loginAttempts = 3; //stating the amount of login attempts

string enteredUsername = "";

string enteredPassword = "";

try

{

users = loadUsers(); //LoadUsers creates the list annd returns it. sets users to the results of the loadUsers

while (loginAttempts != 0) // while loop which runs as long as it is not equal to 0

{

Console.Clear();

Console.WriteLine("Attempts remaining: " + loginAttempts); //displays login attempts

Console.Write("Username: ");

enteredUsername = Console.ReadLine(); //reads user input and assigns it to enteredUsername function

Console.Write("Password: ");

enteredPassword = Console.ReadLine(); //reads user input and assigns it to enteredPassword

if (loginUser(enteredUsername, enteredPassword)) //funciton returns true if the password is incorrect.

{

loginAttempts--; //counts down if password is incorrect

}

else //runs if the password is correct

{

loginAttempts = 3;

}

}

if (loginAttempts == 0) //login attempts equal to 0 then run blockUser to block the entered username.

{

Console.WriteLine("You have been blocked for multiple failed login attempts. please contact admin.");

blockUser(enteredUsername);

Console.ReadLine();

}

}

catch (Exception e)

{

//catches all other exceptions.

Console.WriteLine(e);

}

}

//validates entered credentials.

//users - reference to the users list.

private static bool loginUser(string enteredUN, string enteredPW)

{

bool decrementCounter = false;

LoginInfo userToLogin; //creating a local variable of loginInfo,

try

{

userToLogin = users.Find(user => user.Username == enteredUN); //List.Find() user.Username == enteredUN says find t where the property "Username" is equal to the entered username.

if (userToLogin.password == enteredPW && !userToLogin.isBanned) //check password is correct and user isn't banned.

{

if (userToLogin.UserType.Equals(USER\_TYPES.ADMIN)) //check if user is admin.

{

loginAdmin(userToLogin);//logins in with admin.

}

else

{

loginStandard(userToLogin); //if not an admin login to standard user

}

}

else

{

if (userToLogin.password != enteredPW) //if input isnt equal to enteredPassword

{

Console.WriteLine("Incorrect Credentials.");

Console.ReadLine();

decrementCounter = true; //password is wrong so attempts will be reduced.

}

else if (userToLogin.isBanned) //if it reaches 0 then the ban is applied to the entered user info.

{

Console.WriteLine("Account is banned. please contact admin.");

Console.ReadLine();

}

}

}

catch (NullReferenceException)

{

//This occurs when the entered username doesn't exist.

Console.WriteLine("Incorect Credentials.");

}

catch (ArgumentNullException) //executes when exception is an ArgumentNullException. specific exceptions can be caught.

{

//This occurs when the entered username doesn't exist.

Console.WriteLine("Incorrect Credentials.");

}

catch (Exception e) //All other exceptions

{

Console.WriteLine(e);

}

return decrementCounter; //returns incrementCounter = true if password is incorrect. return is used to determine what the function kicks out, decrementCounter in this case.

}

//Logs in as administrator

private static void loginAdmin(LoginInfo user)

{

string response = "";

LoginInfo newUser;

try

{

do //program will allow admin to do things for as long as he is logged in. they can log out by typing "logout".

{

Console.Clear();

Console.WriteLine("Hello " + user.Name);

Console.WriteLine("Please choose an action: Add, Remove, Unblock, View, Logout");

response = Console.ReadLine();

switch (response) //Better way to do selection compared to if statments, which is a loop responding with the user input which is equal to response

{

case "Add": //if response == "Add", code will execute from here.

newUser = addUser(); //run code for user addition.

users.Add(newUser);

break; //break out of statement. returning to the menu again.

case "Remove":

removeUser();

break;

case "Unblock":

unblockUser();

break;

case "View":

viewUsers();

break;

case "Logout":

break;

default:

Console.WriteLine("Invalid Command. press enter to continue");

Console.ReadLine();

break; //exit out of switch.

}

}

while (response != "Logout"); //defines loop termination condition.

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

//Displays users

private static void viewUsers()

{

try

{

for (int i = 0; i < users.Count; i++) //for loop counts up and displays each user in sequential order

{

Console.WriteLine("User: " + users[i].Username); //writes username and name using i as the value to count up

Console.WriteLine("Name: " + users[i].Name);

if (users[i].UserType.Equals(USER\_TYPES.ADMIN)) //checks USER\_TYPES to see whether if they are admin or standard adn prints it

{

Console.WriteLine("Type: Admin");

}

else

{

Console.WriteLine("Type: Standard");

}

Console.WriteLine("Password: " + users[i].password); //displays user password

Console.WriteLine("Blocked: " + users[i].isBanned); //displays whether the user is banned or not

Console.WriteLine();

}

Console.WriteLine("Press Enter to continue.");

Console.ReadLine();

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

//Adds a user.

private static LoginInfo addUser()

{

string type = ""; //string for USER\_TYPE to be entered

LoginInfo newUser = new LoginInfo();

try

{

Console.Write("Username: ");

newUser.Username = Console.ReadLine(); //enter new users user name.

Console.Write("Password: ");

newUser.password = Console.ReadLine(); //enter new users password.

Console.Write("Name: ");

newUser.Name = Console.ReadLine(); //enter new users name.

Console.Write("Account type (ADMIN, STANDARD): ");

type = Console.ReadLine(); //enter users type

if (type == "ADMIN") //if ADMIN is entered, user will be admin.

{

newUser.UserType = USER\_TYPES.ADMIN;

}

else //if anything else is entered, assumes they are standard

{

newUser.UserType = USER\_TYPES.STANDARD;

}

}

catch (Exception e)

{

Console.WriteLine(e);

Console.ReadLine();

}

return newUser;

}

//Removes a user.

private static void removeUser()

{

string userToRemove;

try

{

Console.Write("User to remove: ");

userToRemove = Console.ReadLine(); //enter the user you wish to remove by inputting username

users.Remove(users.Find(user => user.Username == userToRemove));

}

catch (NullReferenceException)

{

//This occurs when the entered username doesn't exist.

Console.WriteLine("Incorect Credentials.");

}

catch (ArgumentNullException e)

{

//This occurs when the entered username doesn't exist.

Console.WriteLine("Incorect Credentials.");

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

// unblocks a user.

private static void unblockUser()

{

string userToUnblock = "";

LoginInfo userObj;

try

{

Console.Write("User to unblock: ");

userToUnblock = Console.ReadLine(); //Username to unblock input

userObj = users.Find(user => user.Username == userToUnblock); // finds the users, and switches bool value to false.

userObj.isBanned = false;

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

//Logs in standard user

private static void loginStandard(LoginInfo user)

{

try

{

Console.Clear();

Console.WriteLine("Hello " + user.Name + "\n press enter to continue."); //displays username, with a welcome message

Console.ReadLine();

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

//blocks a user

private static void blockUser(string username)

{

LoginInfo userObj;

try

{

userObj = users.Find(user => user.Username == username); //finds a username

userObj.isBanned = true; //swiitches bool value in loginInfo for the user to true banning them

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

// function loads users from list.

private static List<LoginInfo> loadUsers() //using the list made for login, populate it with the class variables and assign it to loadUsers which calls back to this list.

{

List<LoginInfo> users = new List<LoginInfo>(); //list for loginInfo

LoginInfo userToAdd; //local variable for the loginInfo

try

{

//storing all the standard users with one admin in a basic format, could use stream read/writes to create a file system to store users.

userToAdd = new LoginInfo();

userToAdd.Username = "JOHNSM22738";

userToAdd.password = "D7y6a";

userToAdd.Name = "John Smith";

userToAdd.UserType = USER\_TYPES.ADMIN; //decalring user types

userToAdd.isBanned = false;

users.Add(userToAdd);

userToAdd = new LoginInfo();

userToAdd.Username = "BRYNW56655";

userToAdd.password = "GgjN6";

userToAdd.Name = "Bryn Williams";

userToAdd.UserType = USER\_TYPES.STANDARD;

userToAdd.isBanned = false;

users.Add(userToAdd);

userToAdd = new LoginInfo();

userToAdd.Username = "JANEDO98786";

userToAdd.password = "i&acN";

userToAdd.Name = "Jane Doe";

userToAdd.UserType = USER\_TYPES.STANDARD;

userToAdd.isBanned = false;

users.Add(userToAdd);

userToAdd = new LoginInfo();

userToAdd.Username = "NESSAJ25255";

userToAdd.password = "3KsyX";

userToAdd.Name = "Nessa Jenkins";

userToAdd.UserType = USER\_TYPES.STANDARD;

userToAdd.isBanned = false;

users.Add(userToAdd);

}

catch (Exception e)

{

Console.WriteLine(e);

}

return users; //return is used to determine what the function kicks out, users in this case.

}

}

}

## Task 4

### LoginInfo.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task4

{

class LoginInfo

{

public string Username = "Admin";

public string Password = "Admin";

public LoginInfo()

{

}

}

}

### LoanBooks.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task4

{

class LoanBooks

{

public DateTime loanDate;

public DateTime loanReturn;

public int loanAmount;

public int loanID;

public Customer Customer;

public Books Books;

}

}

### Books.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task4

{

class Books

{

public int bookID;

public string bookName;

public int bookCount;

public int bookF;

}

}

### Customer.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task4

{

class Customer

{

public int customerID;

public string customerName;

public string customerAddress;

}

}

### Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task4

{

class Program

{

//make new types of users, using ENUMs

//creating a lsit for the books

static List<Books> listBooks = new List<Books>();

//creating a list for the loaning of books

static List<LoanBooks> listLoans = new List<LoanBooks>();

//creating a list for customers to be stored

static List<Customer> listCustomer = new List<Customer>();

//creating new books

static Books book = new Books();

//creating new customers

static Customer customer = new Customer();

//creating new loans for loanList

static LoanBooks loan = new LoanBooks();

//creating an instance of logininfo

static LoginInfo logininfo = new LoginInfo();

static void Main(string[] args)

{

Console.WriteLine("Please login using admin credentials\n");

//converting the readline into a string username to check against LoginInfo.cs

Console.WriteLine("Please enter username: ");

string enteredUsername = Console.ReadLine();

//converting the readline into a string password to check against LoginInfo.cs

Console.WriteLine("Please enter password: ");

string enteredPassword = Console.ReadLine();

//clear console from login

Console.Clear();

try

{

//if statment checks if the username entered matches the login info stored

if (enteredUsername == logininfo.Username)

{

//checks the password enetered to the logininfo password stored

if (enteredPassword == logininfo.Password)

{

//menu choice bound by input.

string response = "";

//do loop encased around menu, so when say AddBooks(); is executed, it falls back into the loop as long as the repsponse is not exit. allowing it to reprint and request input again.

do

{

Console.Clear();

Console.WriteLine("Welcome you are now logged into the system.");

//menu for the admin system of the libary loan: Add book, get book details, add customer, get customer details, loan book to customer, loan details, remove book, remove customer.

Console.WriteLine("Menu\n" + "1. Add Book\n" + "2. Delete book\n" + "3. Find Book\n" + "4. Add customer\n" + "5. Delete customer\n"

+ "6. Find customer\n" + "7. Loan book\n" + "8. Find loans\n" + "9. Close");

Console.WriteLine("To make a selction type the following:\n1. Add b 2. Delete b 3. Find b 4. Add c 5. Delete c 6. Find c 7. Loan 8. Find l 9. Exit");

response = Console.ReadLine();

switch (response)

{

//case is used to check input, similar to an if statment with the condition ==, the break is used to terminate the loop.

case "Add b":

AddBook();

break;

case "Delete b":

DeleteBook();

break;

case "Find b":

FindBook();

break;

case "Add c":

AddCustomer();

break;

case "Delete c":

DeleteCustomer();

break;

case "Find c":

FindCustomer();

break;

case "Loan":

Loaning();

break;

case "Find l":

FindLoan();

break;

case "Exit":

break;

default:

Console.WriteLine("invalid input. Press Enter to continue");

Console.ReadLine();

break;

}

}

while (response != "Exit");

}

else

{

Console.WriteLine("Incorrect Credentials.");

}

}

else

{

Console.WriteLine("Incorrect Credentials.");

}

}

catch (Exception e)

{

Console.WriteLine(e);

}

//adding a book to the libary

static void AddBook()

{

try

{

//creating new books

Books books = new Books();

//auto increment book amounts

Console.WriteLine("Book ID:{0}\n", books.bookID = listBooks.Count + 1);

//input the name of the book

Console.WriteLine("What is the bookname:\n");

books.bookName = Console.ReadLine();

//writes the book name that was inputted from the readline

Console.WriteLine("The book name entered is: {0}", books.bookName);

//calculating the amount of the book with user input

Console.WriteLine("Please eneter the amount of books:");

Console.WriteLine("Amount of the book/s: {0}\n", books.bookCount = int.Parse(Console.ReadLine()));

//adding the new book to the bookList

listBooks.Add(books);

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

//deleting/removing a book from the libary

static void DeleteBook()

{

try

{

Console.WriteLine("Enter book ID to be deleted:");

//parsing a readline to an int value

int deleteBook = int.Parse(Console.ReadLine());

Console.WriteLine("The book has been deleted.");

//search the books list and removes the book if it equal to the parsed value entered

listBooks.Remove(listBooks.Find(books => books.bookID == deleteBook));

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

//finding a book within the libary

static void FindBook()

{

Books booksFind;

try

{

//--

Books books = new Books();

Console.WriteLine("Search by book ID:");

//parsing a value for finding a book

int findBookID = int.Parse(Console.ReadLine());

//searching the book list and if the find book ID matches up, it will remove it.

booksFind = listBooks.Find(books => books.bookID == findBookID);

//pulls the inforamtion that is related to the ID that is found

Console.WriteLine("the book ID you have searched is {0}, name: {1}, count {2}", booksFind.bookID, booksFind.bookName, booksFind.bookCount);

Console.ReadLine();

//if the find book id doesnt match with a book ID, message displays

if (booksFind.bookID != findBookID)

{

Console.WriteLine("There is no book with this ID");

}

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

//adding a customer to the libary

static void AddCustomer()

{

try

{

Customer customer = new Customer();

//auto increment book amounts

Console.WriteLine("customer ID:{0}", customer.customerID = listCustomer.Count + 1);

//entering the name of the customer

Console.WriteLine("Enter the name of the customer:\n");

customer.customerName = Console.ReadLine();

//entering the address of the customer

Console.WriteLine("Enter the name of the address:\n");

customer.customerAddress = Console.ReadLine();

//displays the newly inputted information about the customer

Console.WriteLine("Enter the Name of the customer: {0}.\n Enter the address of the customer {1}", customer.customerName, customer.customerAddress);

//adding the new customer data

listCustomer.Add(customer);

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

//deleting/removing a customer from the libary

static void DeleteCustomer()

{

try

{

Console.WriteLine("Enter the customer ID you wish to delete");

//parsing user input to an int value

int deleteCustomer = int.Parse(Console.ReadLine());

Console.WriteLine("The Customer information has been deleted.");

//using the int value to search the list to remove

listCustomer.Remove(listCustomer.Find(customer => customer.customerID == deleteCustomer));

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

//find a customer whos been added to the libary

static void FindCustomer()

{

Customer customerFind;

try

{

//--

Customer customer = new Customer();

Console.WriteLine("Search customer by ID:");

int findCustomerID = int.Parse(Console.ReadLine());

//finding a customer with user input

customerFind = listCustomer.Find(customer => customer.customerID == findCustomerID);

//display customer info

Console.WriteLine("the customer ID you have searched is {0}, name: {1}, address: {2}", customerFind.customerID, customerFind.customerName, customerFind.customerAddress);

Console.ReadLine();

if (customer.customerID != findCustomerID)

{

Console.WriteLine("There is no customer with that ID");

}

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

// loaning a book system: needs loan id, book details and customer details with date of taking out and possibly return date

static void Loaning()

{

LoanBooks loanNew;

Books bookToAdd;

Customer custToAdd;

try

{

LoanBooks loan = new LoanBooks();

//making a loan ID

loanNew = new LoanBooks();

Console.WriteLine("Loan ID for this book and customer: {0}", loanNew.loanID = listLoans.Count + 1);

//display customer info and book info that is being loaned with date ----

//find the book using ID and then show its details

Console.WriteLine("Enter book ID:");

int findBookID = int.Parse(Console.ReadLine());

//finding book ID within the loan list

bookToAdd = listBooks.Find(books => books.bookID == findBookID);

Console.WriteLine("The book ID found is {0}, name: {1}, count {2}", bookToAdd.bookID, bookToAdd.bookName, bookToAdd.bookCount);

Console.ReadLine();

//if book ID doesnt match up throw this out

if (bookToAdd.bookID != findBookID)

{

Console.WriteLine("There is no book with this ID");

}

//if it works move on to this

else

{

loanNew.Books = bookToAdd;

//same as the BookID search and show details

Console.WriteLine("Please enter the customer ID");

int findCustomerID = int.Parse(Console.ReadLine());

custToAdd = listCustomer.Find(cust => cust.customerID == findCustomerID);

Console.WriteLine("The customer ID found is {0}, name: {1}, address: {2}", custToAdd.customerID, custToAdd.customerName, custToAdd.customerAddress);

Console.ReadLine();

if (custToAdd.customerID != findCustomerID)

{

Console.WriteLine("There is no customer with that ID");

}

else

{

//adding the time the loan was taken out.

loanNew.Customer = custToAdd;

loanNew.loanDate = DateTime.Now;

Console.WriteLine("The loan was taken out on: {0} It must be returned on {1}", loanNew.loanDate, loanNew.loanReturn);

Console.ReadLine();

listLoans.Add(loan);

}

}

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

//finding the info on a loan

static void FindLoan()

{

LoanBooks loanFind;

try

{

LoanBooks loanBooks = new LoanBooks();

Console.WriteLine("Search by loan ID:");

int findLoanID = int.Parse(Console.ReadLine());

loanFind = listLoans.Find(loan => loan.loanID == findLoanID);

//Display loan info

Console.WriteLine("Loan ID: {1}, Loan Date: {2}", loanFind.loanID, loanFind.loanDate);

Console.WriteLine("Book ID: {0} Book Name: {1}", loanFind.Books.bookID, loanFind.Books.bookName);

Console.WriteLine("Customer ID: {0}, Customer Name: {1}, Customer Address: {2}", loanFind.Customer.customerID, loanFind.Customer.customerName, loanFind.Customer.customerAddress);

Console.ReadLine();

if (loan.loanID != findLoanID)

{

Console.WriteLine("There is no loan with that ID", loanFind.loanID, loanFind.loanDate, loanFind.loanReturn);

}

}

catch (Exception e)

{

Console.WriteLine(e);

Console.ReadLine();

}

}

Console.Read();

}

}

}