

**AAPP004-4-2-JP**

**JAVA PROGRAMMING**

**UCDF1905ICT(SE)**

**HAND OUT DATE: 29 DECEMBER 2020**

**HAND IN DATE: 19 MARCH 2021**

**WEIGHTAGE:**  **60%**

**INSTRUCTIONS TO CANDIDATES:**

**1 Submit your assignment at the administrative counter.**

**2 Students are advised to underpin their answers with the use of references (cited using the Harvard Name System of Referencing).**

**3 Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld.**

**4 Cases of plagiarism will be penalized.**

**5 The assignment should be bound in an appropriate style (comb bound or stapled).**

**6 Where the assignment should be submitted in both hardcopy and softcopy, the softcopy of the written assignment and source code (where appropriate) should be on a CD in an envelope / CD cover and attached to the hardcopy.**

**7 You must obtain 50% overall to pass this module.**

**Name: MUHAMMAD FUAD BIN ABDULLAH TP: TP055684**

# **Table of Contents**

[**Table of Contents** 2](#_Toc66957685)

[**1.0** **Sample Output** 4](#_Toc66957686)

[1.1 Login Menu 4](#_Toc66957687)

[1.2 Main Menu 6](#_Toc66957688)

[1.3 Borrowing Menu 8](#_Toc66957689)

[1.4 Renewing Menu 11](#_Toc66957690)

[1.5 Returning Menu 13](#_Toc66957691)

[1.6 Borrowing History Menu 15](#_Toc66957692)

[1.7 Book Management Menu 16](#_Toc66957693)

[1.8 Client Management Menu 20](#_Toc66957694)

[1.9 Register Librarian Menu 23](#_Toc66957695)

[1.10 Application-wide Output 26](#_Toc66957696)

[**2.0** **Sample Code** 29](#_Toc66957697)

[2.1 Encapsulation 29](#_Toc66957698)

[2.1.1 Getter 29](#_Toc66957699)

[2.1.2 Setter 29](#_Toc66957700)

[2.1.3 Access Modifier 30](#_Toc66957701)

[2.2 Abstraction 31](#_Toc66957702)

[2.3 Inheritance 32](#_Toc66957703)

[2.4 Polymorphism 33](#_Toc66957704)

[2.4.1 Method Overloading 33](#_Toc66957705)

[2.4.2 Method Overriding 36](#_Toc66957706)

[2.4.3 super Keyword 37](#_Toc66957707)

[2.5 Constructor 39](#_Toc66957708)

[2.5.1 Constructor Overloading 40](#_Toc66957709)

[2.6 Packages 41](#_Toc66957710)

[2.7 Exception Handling 42](#_Toc66957711)

[2.8 Event Handling 44](#_Toc66957712)

[**3.0** **Additional Features** 46](#_Toc66957713)

[3.1 Registering as Librarian 46](#_Toc66957714)

[3.2 Logging in as Librarian 47](#_Toc66957715)

[3.3 Adding, Updating and Deleting Client Records 48](#_Toc66957716)

[3.4 Adding, Updating and Deleting Book Records 49](#_Toc66957717)

[3.5 Viewing Past Borrowing Records 50](#_Toc66957718)

[**4.0** **Assumptions** 51](#_Toc66957719)

[4.1 System-oriented 51](#_Toc66957720)

[4.2 Business model 51](#_Toc66957721)

[4.3 Miscellaneous 51](#_Toc66957722)

[**5.0** **References** 53](#_Toc66957723)

# **Sample Output**

## Login Menu

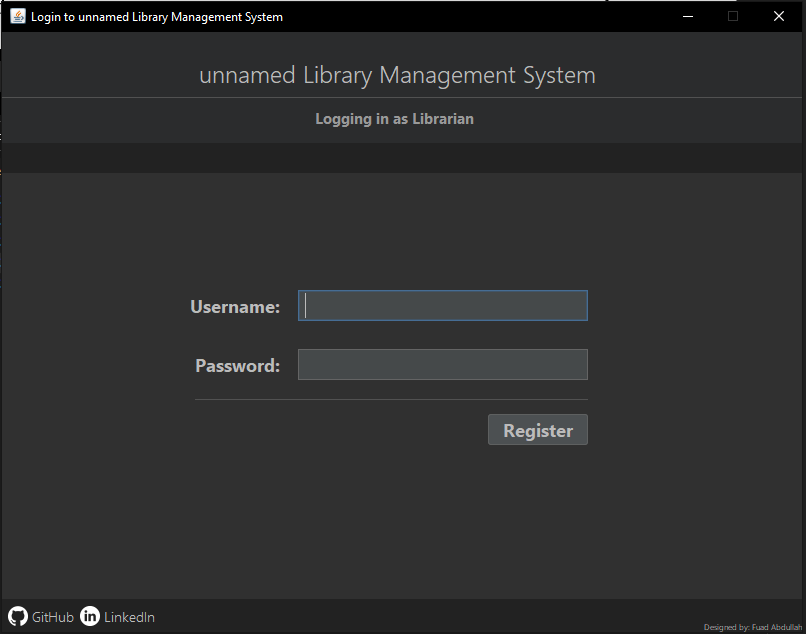


Figure 1: Login menu of unnamed Library Management System.

The form above shows the login interface for a librarian to access the library management system by keying in their username and password. Login button is set to hidden upon start to prevent librarian from entering empty strings as credentials. Login button will show only when both username and password fields are not empty.

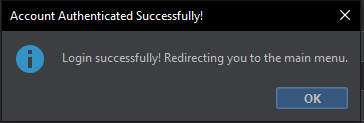


Figure 2: Successful login attempt popup message.

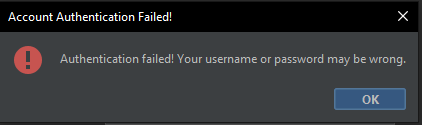


Figure 3: Failed authorization for the login attempt popup message.

In Figure 2, the popup message notifies the librarian of a successful authorization and redirection to the main menu page where the librarian can perform librarian’s tasks. The account will only be authorized if the username is available inside the text file and the inserted password matches the one found in the username record. For the popup message in Figure 3, the librarian is informed of a failed login attempt as the warning message will only appear if the username or the password or both credentials inserted are wrong and have no exact match inside the text file.

## Main Menu

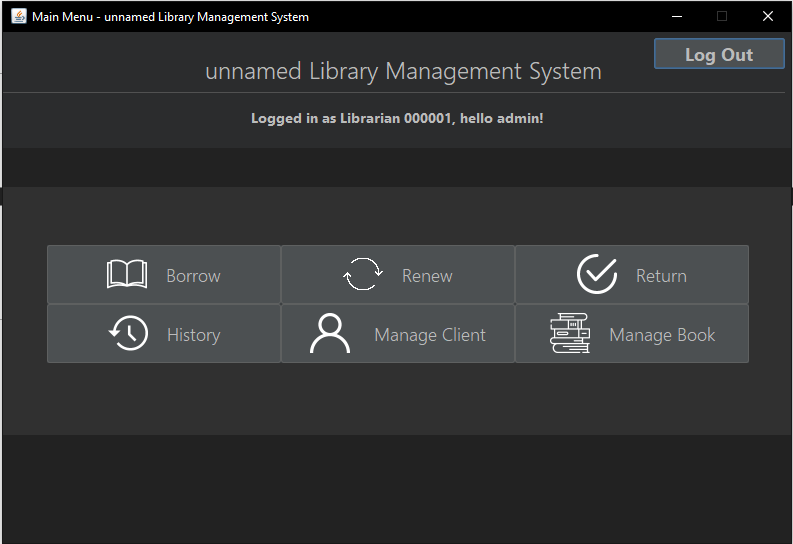


Figure 4: Main menu of unnamed Library Management System

The form above shows the main menu interface for librarian. There are 7 interactable buttons available in the form starting from the top right is a button to log the user out of their account. Button labelled “Borrow” opens the borrowing page, “Renew” opens renew borrowing page, “Return” opens return borrowing page, “History” opens borrowing history page, “Manage Client” opens client management page and lastly “Manage Book” opens book management page.

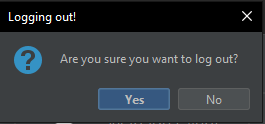


Figure 5: Logout confirmation popup message

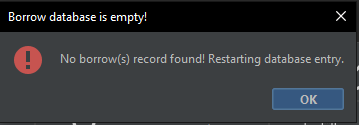


Figure 6: Opening borrowing form with empty database popup message

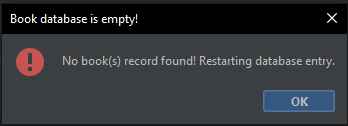


Figure 7: Opening book management form with empty database popup message

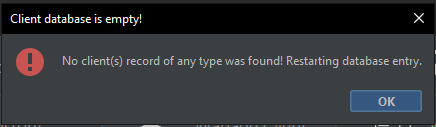


Figure 8: Opening client management form with empty database popup message

In Figure 5, the confirmation dialogue appeared when the librarian attempts to log out of their account. If the librarian presses on “Yes”, the librarian will be returned to the login menu. Otherwise, the librarian will remain on main menu page. A confirmation dialog is used to prevent librarians from accidentally logging out due to false click on the button. For Figure 6, 7 and 8, displays a popup message to notify librarian of an empty borrow, book and client databases and the imminent restart for ID increment from one. Popups regarding the database restart are more likely to happen when the system runs for the first time.

## Borrowing Menu

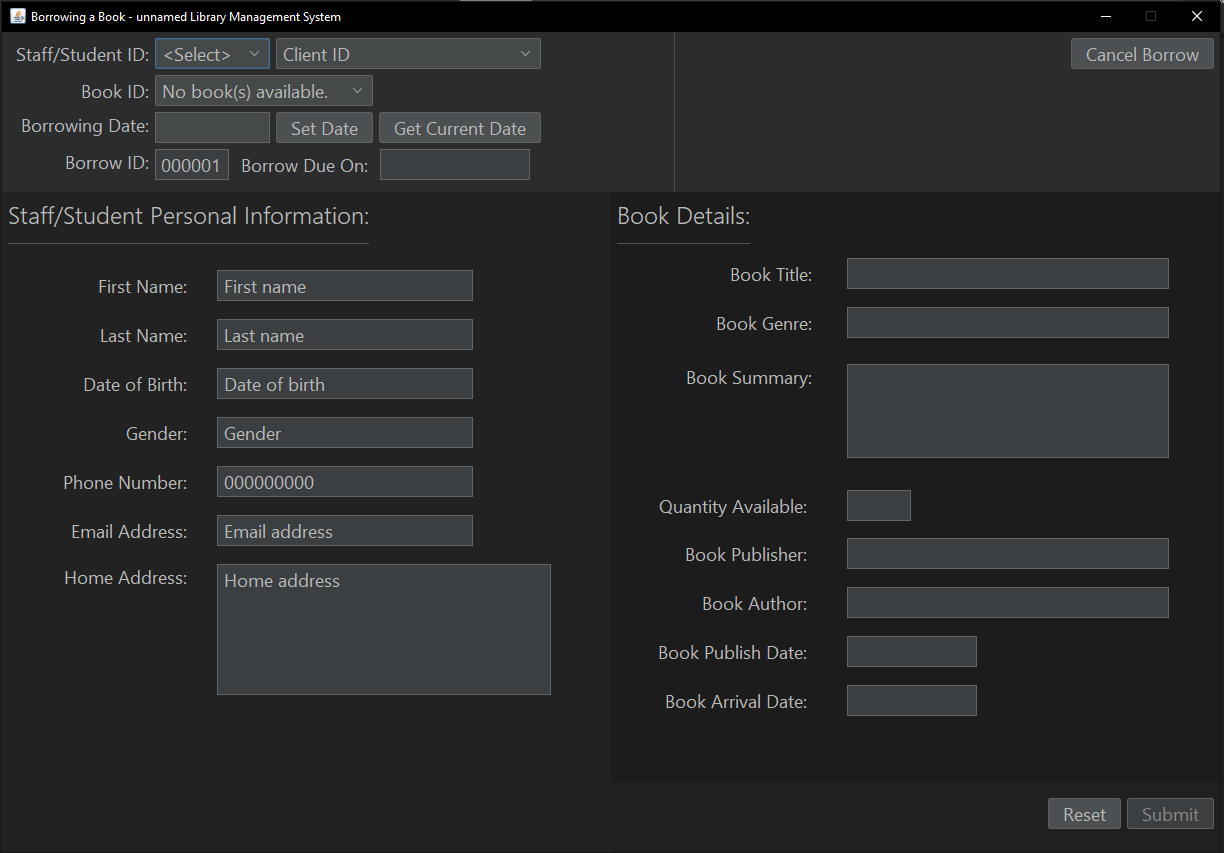


Figure 9: Borrowing form of unnamed Library Management System

The form above shows the borrowing menu page where borrowing session for clients takes place. The librarian is only required to select the client type, client ID, book to be borrowed by the client, and setting the date of borrowing. Borrowing duration is defaulted to 14 days and therefore the due date is set to 2 weeks after borrowing date. Clients who are not registered or books which are not listed inside the system cannot borrow and be borrowed. Once all the required fields are set, the librarian may find “Submit” button enabled for borrowing record creation.

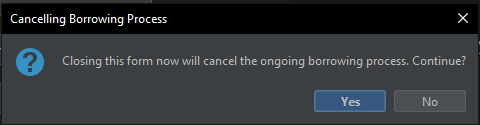


Figure 10: Borrowing cancellation popup message.

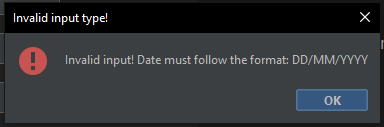


Figure 11: Invalid date format for borrow date popup message.

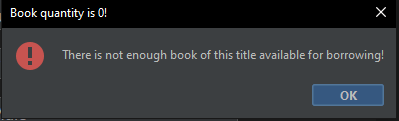


Figure 12: Insufficient number of books available for borrowing popup message.

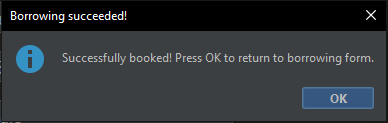


Figure 13: Successful borrowing popup message.

The screenshots above display multiple popups and confirmation dialog to inform the user of specific events that must be handled by the librarian. For confirmation dialog of Figure 10, the system is asking if the librarian actually intended on leaving the borrowing menu and be returned to the main menu page. If the librarian answers “Yes”, they will be redirected back to the main menu page and any ongoing borrowing will be cancelled. Otherwise, nothing will happen if the librarian were to choose “No” as answer. A confirmation dialog is implemented to prevent the librarian from accidentally exiting to main menu while a borrowing process is ongoing. For the popup message in Figure 11, the system is notifying the librarian of an invalid date insertion as it did not follow the specified format of “DD/MM/YYYY”. A try-catch block is written to ensure that if the date fails to parse, this message will appear, and librarian should correct the inserted date format. In Figure 12, the popup message warns the librarian of insufficient number of books available for clients to borrow as the book quantity is zero and client will not be able to proceed with borrowing until they switch to another book. Upon successful borrowing, a popup message seen in Figure 13 will appear after the “Submit” button is pressed by the librarian, indicating a new borrowing record creation.

## Renewing Menu

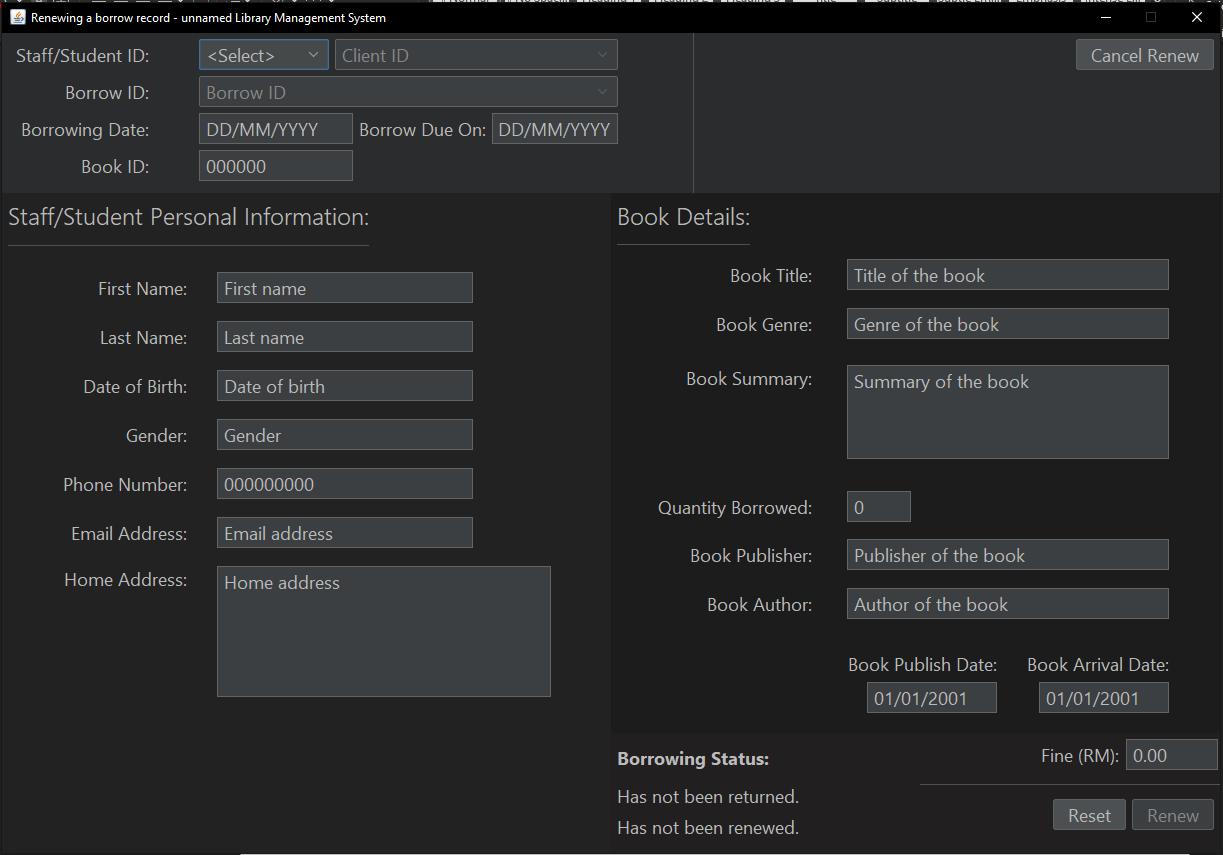


Figure 14: Renewing form of unnamed Library Management System.

The form above shows borrowing renewal interface for librarian-use to clients who would like to extend the duration of their borrowings. Librarian is required to select the client type, client ID and the borrowing ID which the client wants to be renewed from the combo boxes on the top part of the page. A borrowing record can only be renewed once, and the extension duration is capped at 14 days. A client cannot extend an overdue borrowing and must return the book at once after fine is settled.

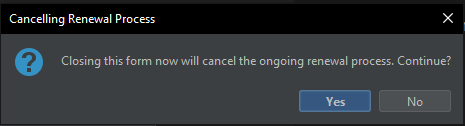


Figure 15: Renewal cancellation popup message.

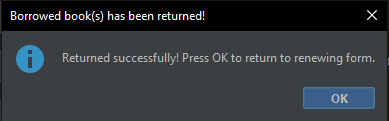


Figure 16: Successful book return popup message.

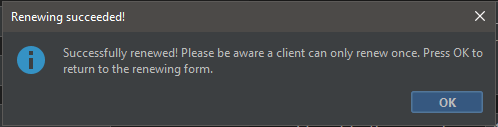


Figure 17: Successful book renew popup message.

In Figure 15, a confirmation dialog is shown to ask for librarian’s decision on whether or not to cancel the renewing process and return to the main menu page. If the librarian selects “Yes”, then they will be redirected to the main menu and any ongoing renewal process will be cancelled. If “No” is selected, then librarian would remain at the main menu page with nothing happening. The confirmation dialog serves as extra precaution for when librarian accidentally presses on the “Cancel Renew” button. For Figure 16, the popup displays a successful return message after “Pay Fine” button is pressed and have completed all of its return procedures. As seen in Figure 17 is a message to notify the librarian of a successful renew along with library policy to limit renewal to once a borrowing record.

## Returning Menu

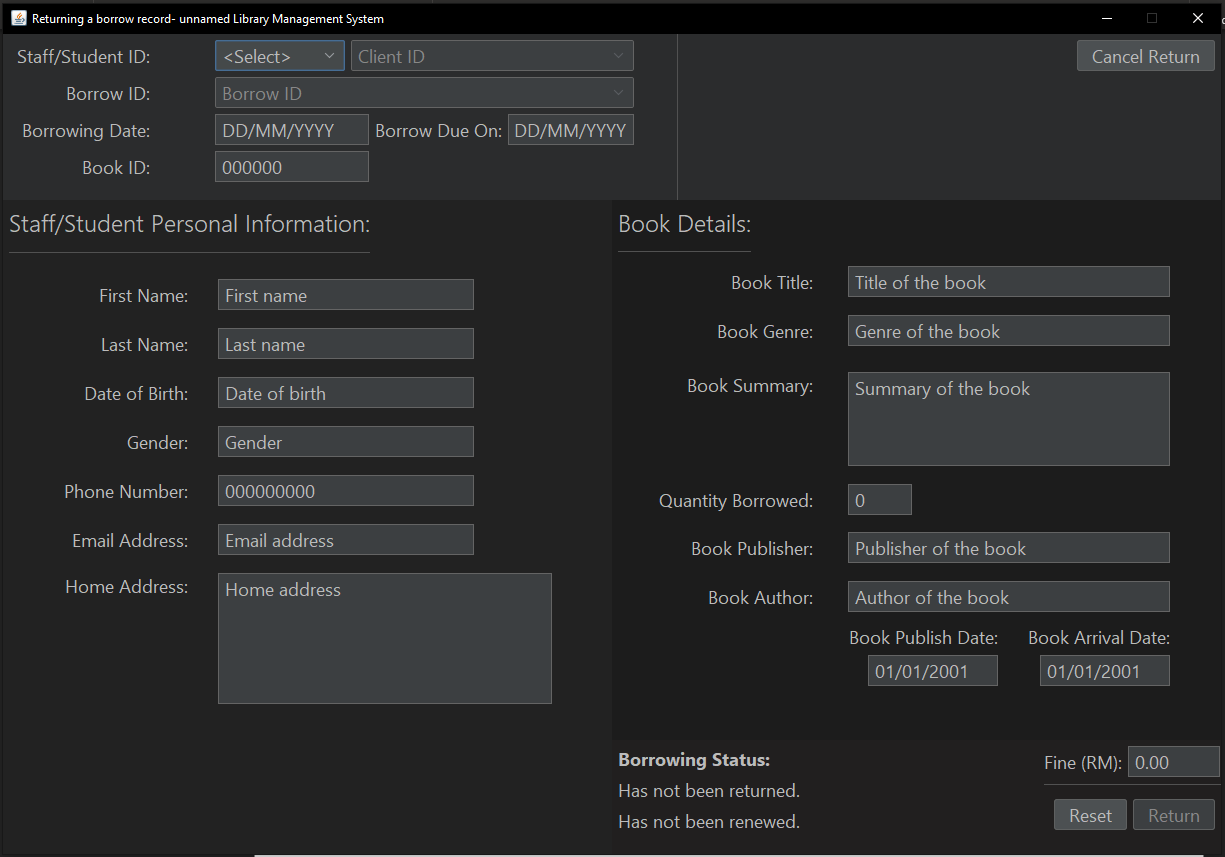


Figure 18: Returning form of unnamed Library Management System.

The form above shows borrowing return interface for librarian to accept clients who are returning their borrowed books. Librarian is required to select the client type, client ID and the borrowing ID which the client wants to be returned from the combo boxes on the top part of the page. A client can only start returning their borrowed books after the 7th day of their borrowing period. An overdue borrowing record requires the client to pay the accumulated amount of fine before continuing with the return process.

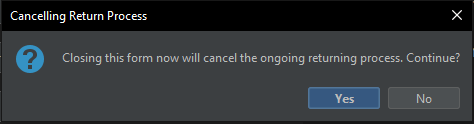


Figure 19: Return cancellation popup message.

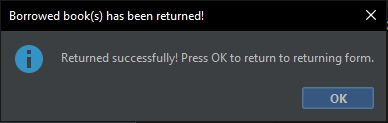


Figure 20: Successful book return popup message.

Based on Figure 19, a confirmation dialog to verify the librarian’s action is displayed before redirecting them back to the main menu. This message uses a confirmation dialog in case librarian changed their mind. In Figure 20, a popup is displayed to inform the librarian of a successful return after “Pay Fine” or “Return” button is pressed and have completed all of its return procedures.

## Borrowing History Menu

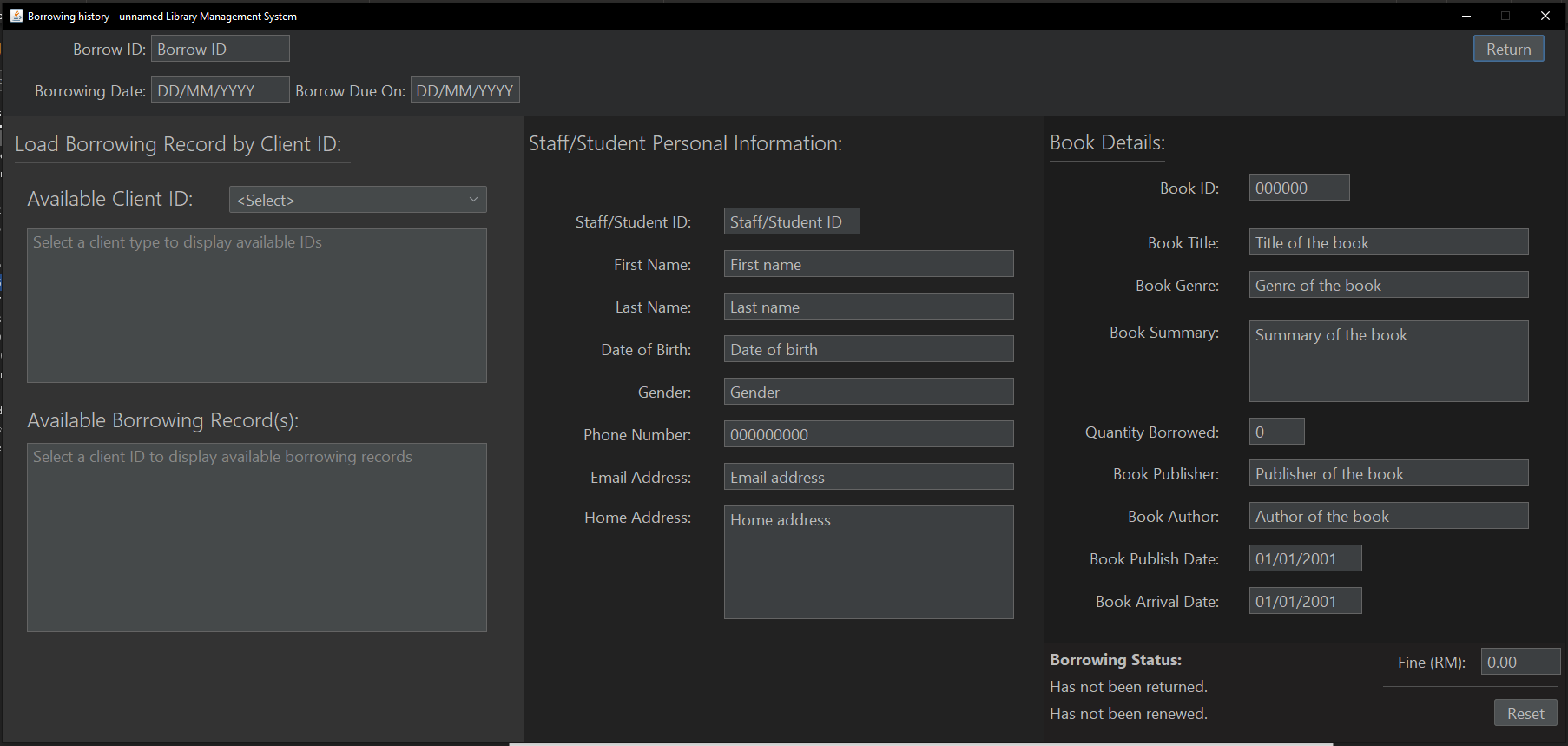


Figure 21: Borrowing history form of unnamed Library Management System.

The form above shows borrowing history interface for the librarian to keep track of client’s borrowing records that are either ongoing or returned. The librarian is required to select the client type, client ID, and available borrowing IDs under the client in order to fully display all relevant information in the fields.

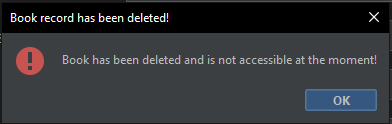


Figure 22: Deleted book popup message.

In the figure above, the librarian is notified of an exception to handle deleted book entries when the system is attempting to display its details in a borrowing history page.

## Book Management Menu

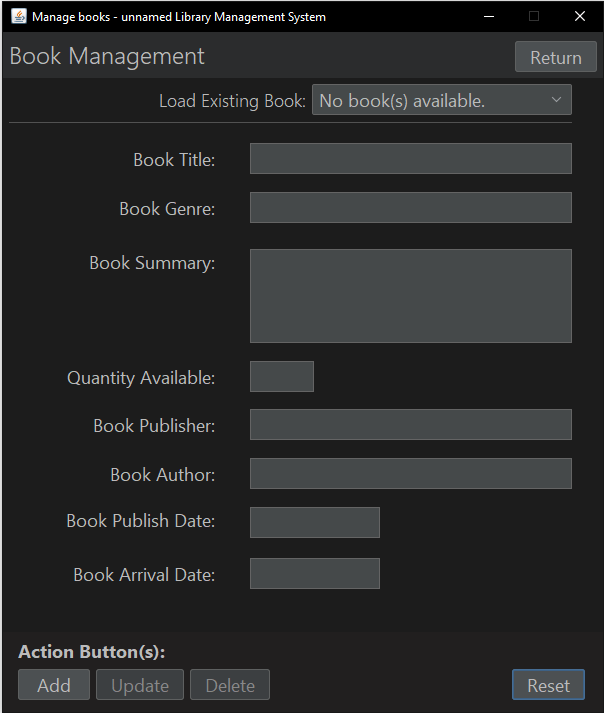


Figure 23: Book management menu of unnamed Library Management System.

The form above shows book management interface for librarian to perform administrative control over new or existing book records through insertion, update or delete actions. Librarian is required to insert book details into related fields when adding a new book, make changes to existing fields when updating a book, or delete the book record to make it inaccessible and unloadable upon system request.

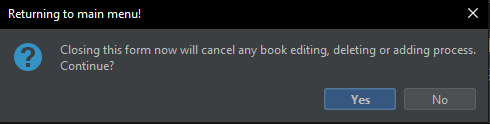


Figure 24: Book management cancellation popup message.

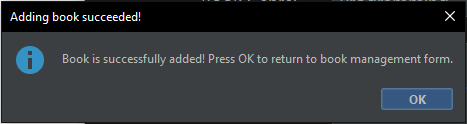


Figure 25: Successful book record insertion popup message.

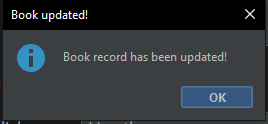


Figure 26: Successful book update popup message.

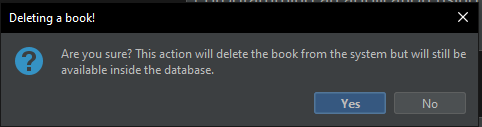


Figure 27: Book deletion confirmation popup message.

In Figure 24, the confirmation box is displayed to the librarian to verify their attempt to cancel the ongoing book management and to return to the main menu. Option-based dialog is provided to allow for librarian to cancel their action. For Figure 25, a message popup is displayed to indicate a successful book insertion into the text file after the “Add” button is pressed. The popup in Figure 26 notifies the librarian of successful update to the selected book record after the “Update” button is pressed. For Figure 27, another confirmation dialog is used but this time is to verify the librarian’s attempt to delete a book from the text file. Same reason as confirmation dialog for return to menu, deletion of book is an important decision to make and therefore it is wise to have an option to cancel the deletion.

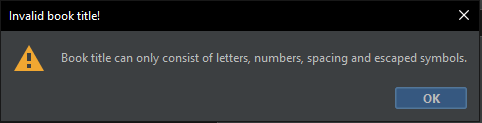


Figure 28: Invalid book title string popup message.

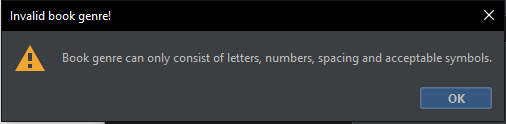


Figure 29: Invalid book genre string popup message.

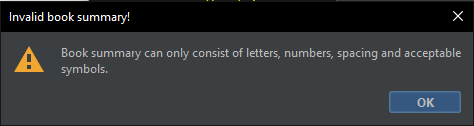


Figure 30: Invalid book summary string popup message.

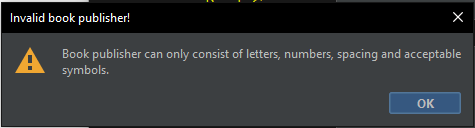


Figure 31: Invalid book publisher string popup message.

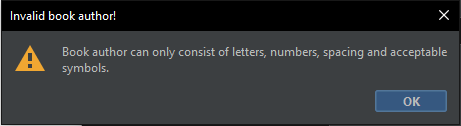


Figure 32: Invalid book author string popup message.

Figures starting from 28 until 32 are popups triggered by document listener’s methods for string validation. The strings are extracted from respective text field and is matched against a regular expression to check and see if it contains any illegal characters. The popups are created from an object class made specially to handle real-time validation as the one demonstrated by the text field exceptions here. Book details validations are exclusive to book management page.

## Client Management Menu

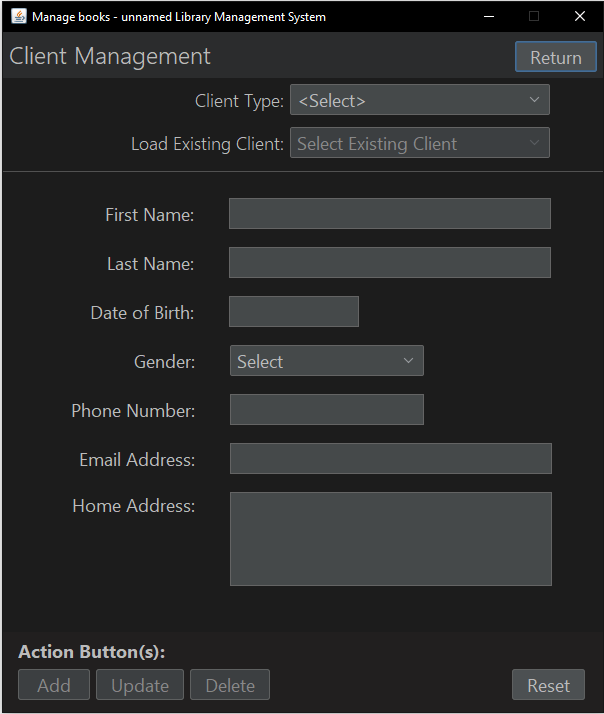


Figure 33: Client management menu of unnamed Library Management System.

The form above shows client management interface for librarian to perform administrative control over new or existing client records through insertion, update or delete actions. Librarian is required to insert client details into related fields when adding a new client, make changes to existing fields when updating a client, or delete the client record to make it inaccessible and unloadable upon system request.

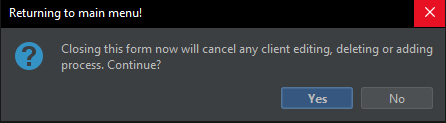


Figure 34: Client management cancellation popup message.

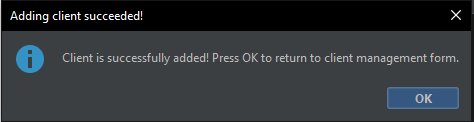


Figure 35 : Successful client record insertion popup message.

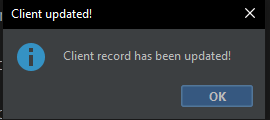


Figure 36: Successful client update popup message.

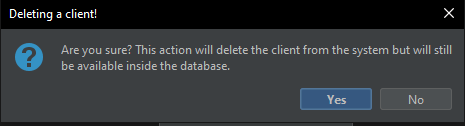


Figure 37: Client deletion confirmation popup message.

Figure 34 displays a confirmation dialog for librarian to reconsider their action on either to be redirected back to the main menu or remain in the client management page. For Figure 35, a popup will show successful client record insertion into client text file message after the “Add” button is pressed. Meanwhile in Figure 36, the popup is displayed after a successful update made to the selected client record over the “Update” button click. Lastly in Figure 37, another confirmation dialog appears when “Delete” button is pressed in order to make sure the librarian is serious with deleting a client record.

## Register Librarian Menu

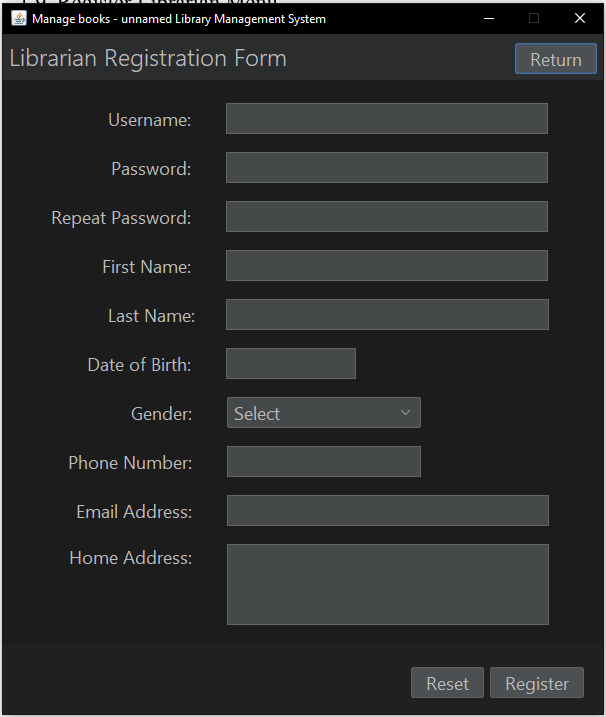


Figure 38: Librarian registration form of unnamed Library Management System.

The form above shows librarian registration interface for librarian to register an authorized account to login to the system. Librarian is required to insert their details into related fields.

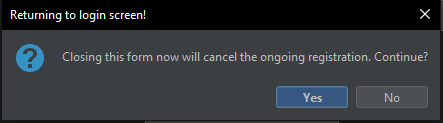


Figure 39: Registration cancellation confirmation popup message.

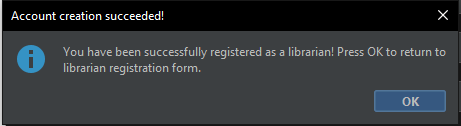


Figure 40: Successful account creation popup message.

Figure 39, a confirmation box to return to the login menu appears and it has two options for librarian to select to act as cautionary measure in case they mistakenly pressed on the button. While in Figure 40, the librarian is notified of a successful account creation with the account becoming available for login once the librarian returns to the login page.

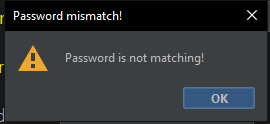


Figure 41: Password mismatch popup message.

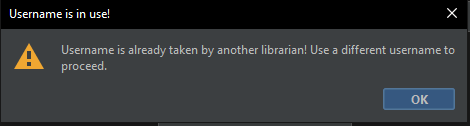


Figure 42: Unavailable username popup message.

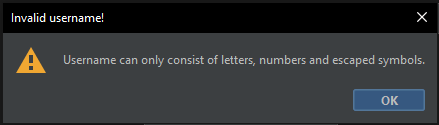


Figure 43: Invalid username string popup message.

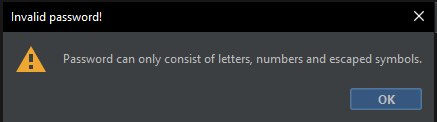


Figure 44: Invalid password string popup message.

Screenshots from Figure 41 to 44 are popups triggered by document listener’s methods for string validation. The strings are extracted from respective text field and is matched against a regular expression to check and see if it contains any illegal characters. The popups are created from an object class made specially to handle real-time validation as the one demonstrated by the text field exceptions here. Username and password validation are exclusive to librarian registration page for the system.

## Application-wide Output

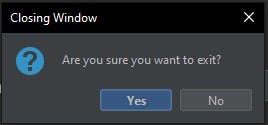


Figure 45: Window closing confirmation popup message.

The figure above is confirmation dialog displayed every time the close button on any form is pressed to warn the librarian of application termination if “Yes” is pressed.

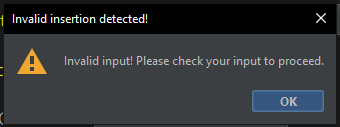


Figure 46: Invalid input string popup message.

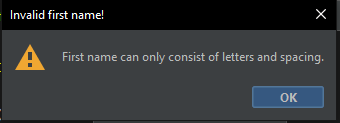


Figure 47: Invalid first name string popup message.

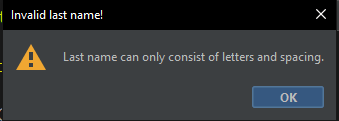


Figure 48: Invalid last name string popup message.

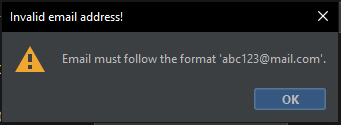


Figure 49: Invalid email address string popup message.

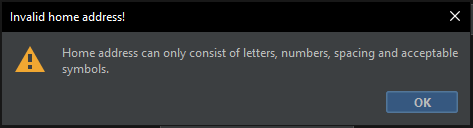


Figure 50: Invalid home address string popup message.

The figures starting from 46 to 50 are string validation popups which are triggered by document event listener with validation instances handling specific text fields. The strings are extracted from respective text field and is matched against a regular expression to check and see if it contains any illegal characters. The popups are created from an object class made specially to handle real-time validation as the one demonstrated by the text field exceptions here. Personal details validations are exclusive to librarian registration and client management page.

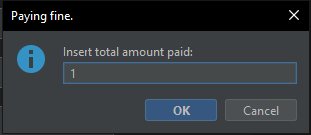


Figure 51: Fine amount input popup message

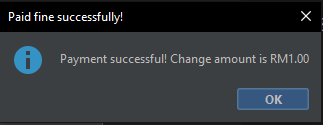


Figure 52: Change amount to the fine paid

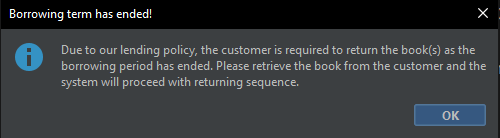


Figure 53: Mandatory return policy popup message

Figures from 51 until 53 are popups during fine settlement process. The input dialog requires the librarian to insert the amount of fine paid by the client over the counter when returning the books. In case of when a client paid the fine amount in excess, another popup message will appear to tell the librarian of exact amount to return to the client as change. Once fine is settled, a popup is displayed to remind the librarian of the library’s policy. The policy covers standard, fined, and overdue returning process. The librarian is required to explain to the client of the policy to let them understand about the library’s standard operating procedure.

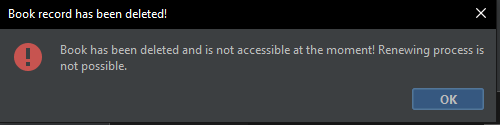


Figure 54: Deleted book popup message

The figure above displays a popup that displays when a deleted book record is being accessed by an ongoing or returned borrowing record and it can be found in renew and returning pages. The system disables loading and display of deleted book record and thus, this popup is shown to inform the librarian of the inability to view what book was borrowed.

# **Sample Code**

## Encapsulation

Encapsulation is a concept under Object-Oriented Programming whereby a method is created to fetch or manipulate a variable which is intended to be hidden or inaccessible by other classes. The methods are often named exactly or consisting of ‘get’ for fetching and ‘set’ for manipulating the variable value.

### 2.1.1 Getter

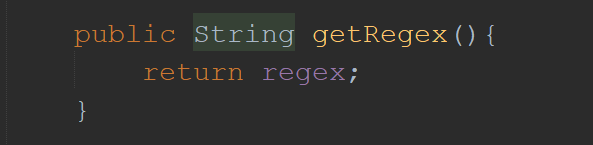


Figure 55: Implementation of getter for regular expression variable

In the code snippet above, getRegex() is an encapsulation method that is written to return a String data type of variable regex when called through the instance of the class. The regex variable stores regular expression string which will be of use during validation phase.

### 2.1.2 Setter

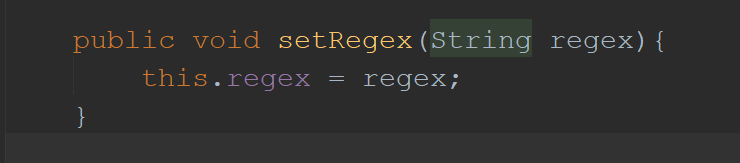


Figure 56: Implementation of setter for regular expression variable

In contrary to getRegex(), setRegex() is another method used to define a new value to the regex variable. In this case, there is no return value as the purpose of the method is only to set the variable to a new value but a String parameter is expected when calling this method through the instantiated class.

### 2.1.3 Access Modifier

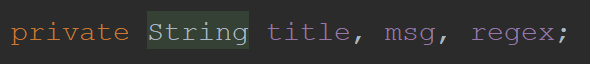


Figure 57: Private variables found in abstract class unnamedStringValidation

From the code above, a private access-level is assigned to the variables title, msg and regex. Private access-level is a type of access modifiers available to provide control to the class. When paired with the concept of encapsulation, the variables cannot be written nor read by classes other than the one they are currently in unless by using getters and setters.

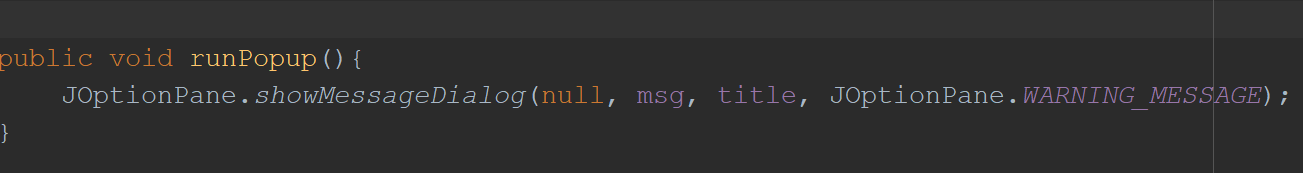


Figure 58: Public method found in abstract class unnamedStringValidation

The method above is an example of an element that is set to public as to allow for other classes of the same project to access and execute the function of the method when a subclass of the abstract class is instantiated. Public access-level has no strict restriction as those found in private and protected access-levels.

## Abstraction

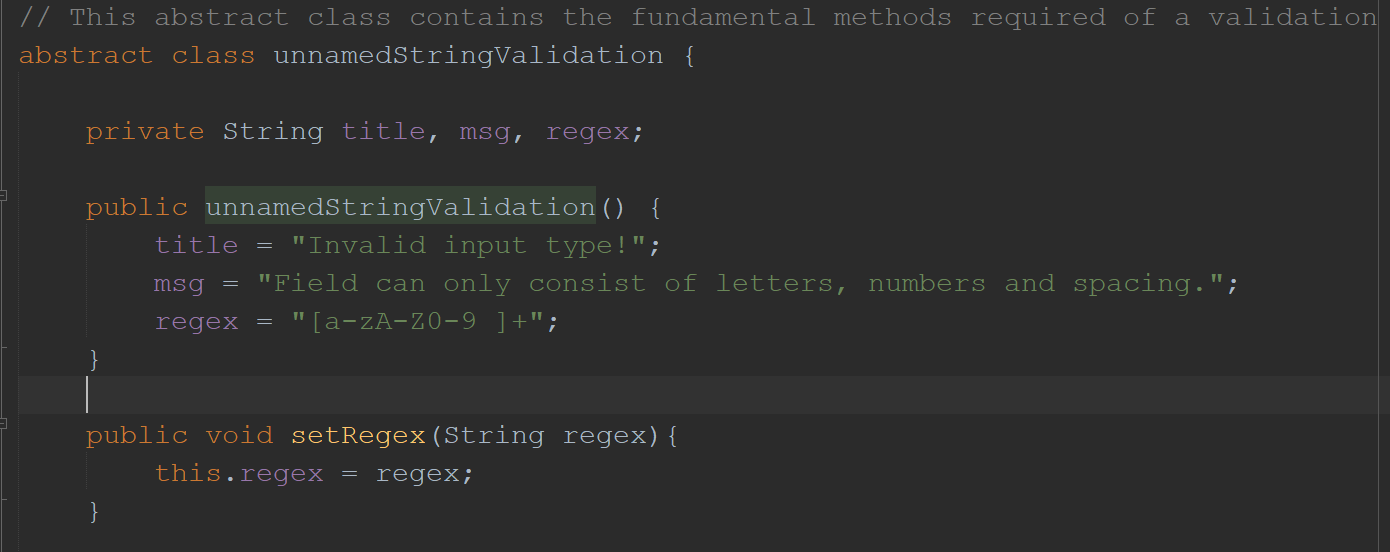


Figure 59: Abstract class unnamedStringValidation with constructor and encapsulation methods

Abstraction is another Object-Oriented Programming concept that focuses on concealing back-end implementations and only display the functionalities of the class. Simply put, an abstraction hides the way a thing is implemented, allowing the user or developer to only care about its purpose and ways to apply the class. Abstraction can be implemented as a class or an interface.

In the code snippet above, unnamedStringValidation is declared as an abstract class to contain all the necessary methods including getters and setters to the private variables found within the class. When a class is too general and must never be instantiated by any other classes due to its broad applicability, then it should be declared as an abstract class. In the case of the validation class above, its abstract declaration is fitting as the purpose of the methods found inside the class is too general and can be specialised through inheritance to subclasses.

## Inheritance

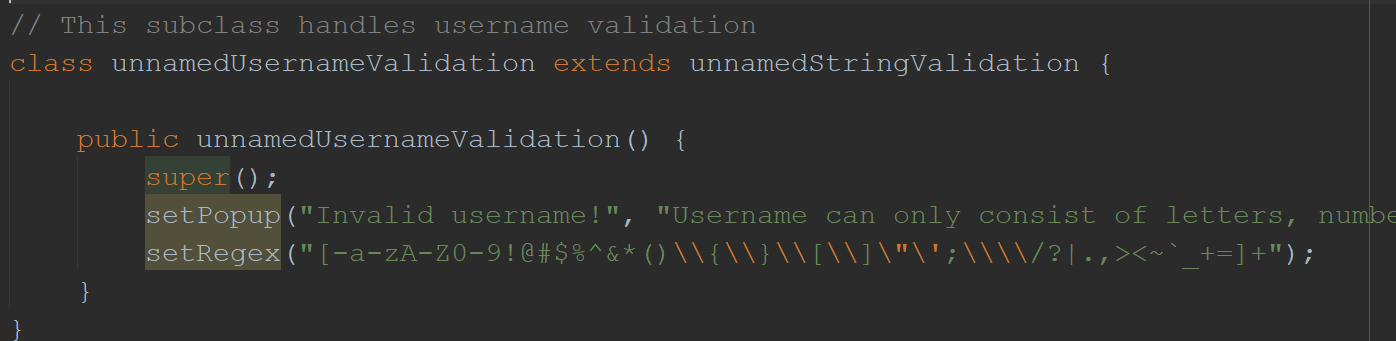


Figure 60: Inheritance of a subclass from abstract class unnamedStringValidation

Inheritance is another core concept of Object-Oriented Programming in Java which is often depicted by many as Parent-Child relation between classes with a class acting as parent class for many child classes to inherit from. As the relationship states, the child class will inherit all functionalities and characteristics of a parent class. The relationship can be materialized by creating and extending a new class from one of the existing classes. Advantages of inheriting from a parent class allow for reusability of methods and variables as well as being able to override a specific part of the class with different functionality while retaining the rest.

For the library system, the concept is applied to validation of client, book, and librarian information fields to ensure invalid characters are filtered out before the record can be inserted into the text file. In the example above, unnamedUsernameValidation is a subclass to the abstract class unnamedStringValidation and inside the class, there is nothing but the public constructor with super(), setPopup(), and setRegex() methods being called to initialize username specialization when the class is instantiated.

## Polymorphism

Polymorphism is an Object-Oriented Programming concept that revolves around the idea of having a method to accept different input from its parameters or being overridden with different functionalities entirely. In simpler terms, it allows for a method to have different settings to fit various kinds of implementation criteria.

### 2.4.1 Method Overloading

Method overloading is a concept under polymorphism where a class contains methods of the same name but with different parameters. This enables different input parameters in terms of data type, object, or sequence to be accepted and processed by the function written inside the method.

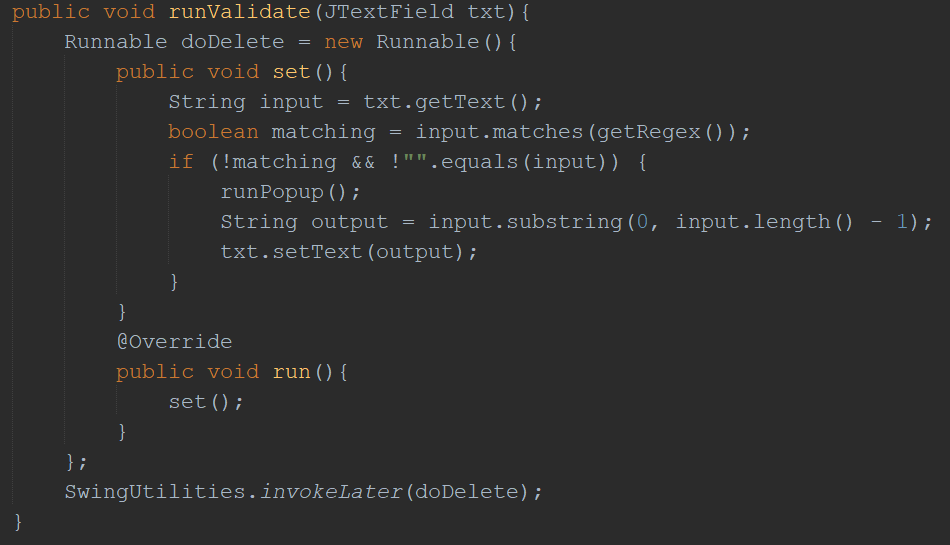


Figure 61: Initial method runValidate() in abstract class unnamedStringValidation with JTextField as argument

In the example above, the method runValidate() is found inside abstract class unnamedStringValidation. The functionality of this method is to validate the strings inserted into a text field by comparing the strings to the regular expression defined for the instance. In the case of runValidate(), the method is set to accept input parameter in the form of JTextField object. However due to the abstract properties of the class, the method can only be used through an instantiated subclass.

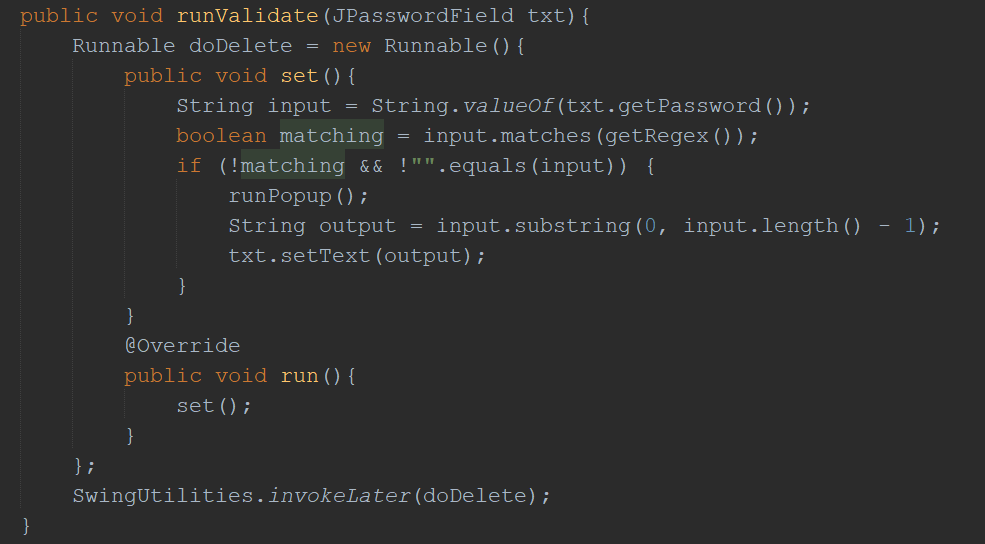


Figure 62: Another method runValidate() in subclass unnamedPasswordValidation with JPasswordField as argument

The screenshot above is the method runValidate() with different input parameter defined. This method is found inside unnamedPasswordValidation, which is a subclass to the abstract class unnamedStringValidation. In the parameter section of the method, it was found out that the object JPasswordField takes the position of the previous JTextField object. Looking at the context, the subclass, once instantiated, will have two methods of the same name however with different input parameters available for use.

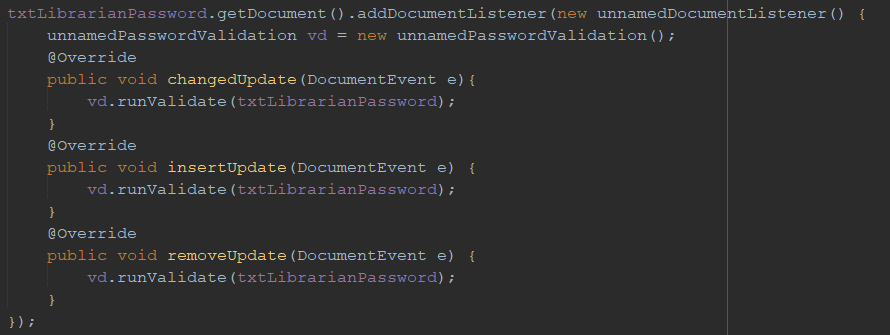


Figure 63: Overloaded runValidate() through instantiated class unnamedPasswordValidation with txtLibrarianPassword JPasswordField as its parameter.

Code snippet above is an example of application scenario for unnamedPasswordValidation subclass through anonymous class unnamedDocumentListener’s document event listener methods. Inside the anonymous class, unnamedPasswordValidation is instantiated and named as ‘vd’ and subsequently used in all three of the methods to trigger validation upon string insertion. Note that txtLibrarianPassword is a JPasswordField type of object and no error is triggered as the method is available inside the subclass along with the inherited JTextField parameter method. The IDE will automatically determine which method is suitable for use by comparing the overloaded parameter to all methods of the same name available in the class. If no such method with such parameter exists, an error is displayed for the developer to take note and act accordingly.

### 2.4.2 Method Overriding

Opposite to the idea of method overloading and as the name states, method overriding is another concept under polymorphism where a method with the same name and parameter is overridden with new codes partially or completely. An overridden method is usually found in subclasses and denoted by a ‘@Override’ notation above the method. The perk of overriding a method is that it enables the subclass an enhanced control over the method to suit the application situation.

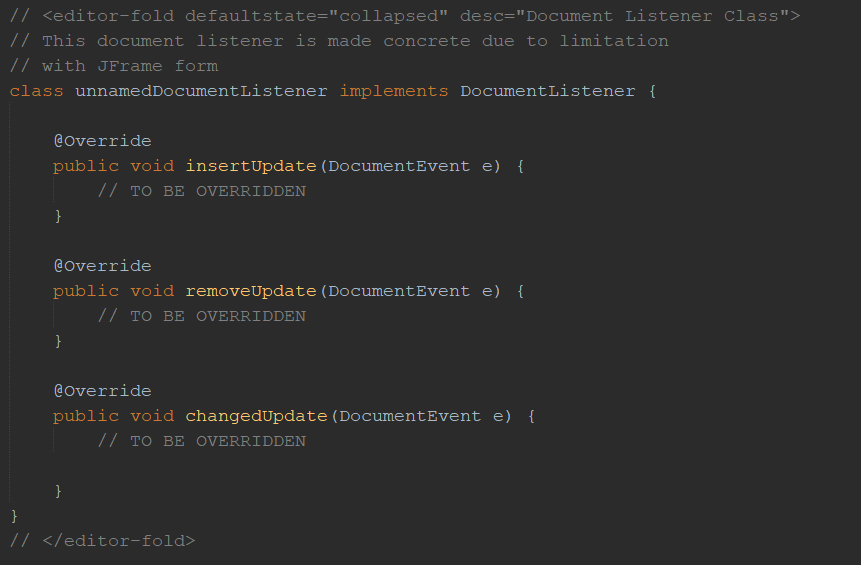


Figure 64: Document listener concrete class unnamedDocumentListener with all of its mandatory methods declared.

Although this step is unnecessary for some, a concrete class was written for the abstract event listener class that triggers every time a text field has changes made to it. In the case of creating a real-time validation, unnamedDocumentListener class contains three mandatory methods that fires when the text field it is attached to has string insertion, string removal, or properties changes. Since the class has no other practical use besides for detecting live changes, the method was not populated with any codes or methods and will be overridden through an anonymous class during an actual application.

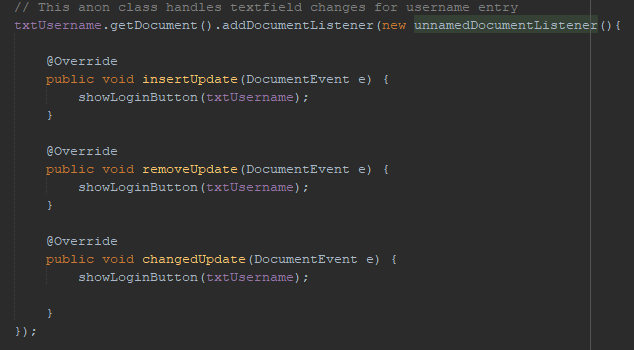


Figure 65: Anonymous instance of unnamedDocumentListener attached to txtUsername text field with its methods overridden.

As displayed in the code excerpt above, an anonymous class was instantiated to attach the concrete class unnamedDocumentListener to txtUsername JTextField. Inside the anonymous instance, the three methods have been overridden with different code and in this example, all three has a method called showLoginButton() overloaded with txtUsername JTextField object as its parameter. Besides the clear difference between the original unnamedDocumentListener and the one in the anonymous instance, the methods have the required ‘@Override’ notation shown above it. Now whenever the document listener fires, one of the three methods will have its content executed and for the system, it would be a real-time validation to check if the user has keyed-in their username or not.

### 2.4.3 super Keyword

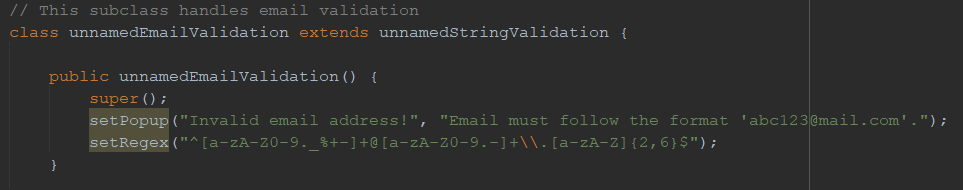


Figure 66: Default constructor of unnamedEmailValidation class with super() constructor.

The screenshot above is a demonstration to the usage of super keyword which, when invoked, refers to the parent class of the child class. There are plenty of ways to use the keyword primarily when a subclass is attempting to refer to anything from a parent’s class. For example, a super can be used to refer to the method found inside a parent class or the parent class itself. In the screenshot, setPopup is a short hand of super.setPopup() which is another way to invoke the setter method. Notice that there is a super() written before the setters, the super() is a keyword used to invoke the parent’s constructor and in this case, it is to load default values to the private access-level variables of unnamedStringValidation abstract class.

## Constructor

A constructor is an element of Object-Oriented Programming that is identical to a method though with a key difference as there are two types of constructors, default and parameterized. A default constructor is a term used to describe a constructor with no parameter set and will be called when the class is instantiated without parameters while a parameterized constructor has parameters set and is called when the instantiated class has relevant parameters. Common similarity between a default and a parameterized constructor is that both must have the class name as its constructor name. For instance, a class Motorcycle will have a default constructor with the name Motorcycle() and an optional parameterized constructor Motorcycle(String brand).

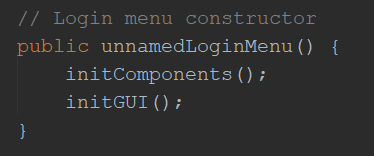


Figure 67: Constructor of login menu with auxiliary methods to load upon run

The code snippet above is taken from a JForm which displays login interface for librarians of unnamed Library. Due to the form being a class, it has a default constructor that is called when the form is instantiated for display. Inside, methods to initialise objects such as labels, inputs and other behavioural mechanics are included.

### Constructor Overloading

An overloaded constructor is similar to an overloaded method through its parameter variables declaration or insertion. Major difference between an overloaded method and constructor would be that a constructor cannot be called directly and as freely as a method because a constructor is only called during the instantiation process of the class.

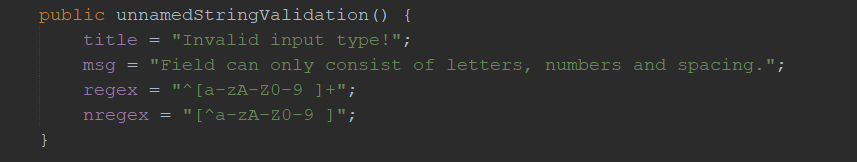


Figure 68: Default constructor in abstract class unnamedStringValidation

Screenshot above depicts the default constructor of unnamedStringValidation. When the class is instantiated, this constructor will be called by default if no parameters were stated. But, this class can never be instantiated as it was declared abstract and thus, the constructor can be called only when it is invoked by using the super() keyword in the default constructor of a subclass.

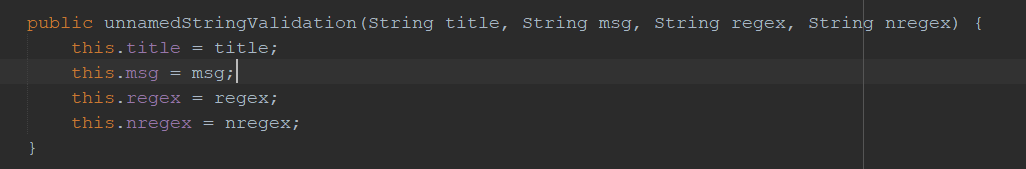


Figure 69: Parameterized constructor overloaded with four string inputs.

The code snippet above displays a constructor with its parameter overloaded with string variables to accept custom title, message, regular expression and excluded regular expression. As seen from previous screenshots, unnamedStringValidation is the name of the class and it has a default constructor. This constructor can be called during the class instantiation by inserting all four variables, as opposed to empty parameter which will automatically call the default constructor.

## Packages



Figure 70: User-defined package of the system.

A package is a collection of classes, sub-classes and interfaces which can be classified into two types and those are user-defined and Java built-in package. In the screenshot above, unnamedlibrary is the name of the package where the whole system is running on. If the line were to be removed from the file, the system will not be able to run at all as references used by the code can no longer be found. As for Java built-in packages, there are plenty of them available which are only referenceable by the code if the packages are imported or written in the line of code which requires it explicitly.

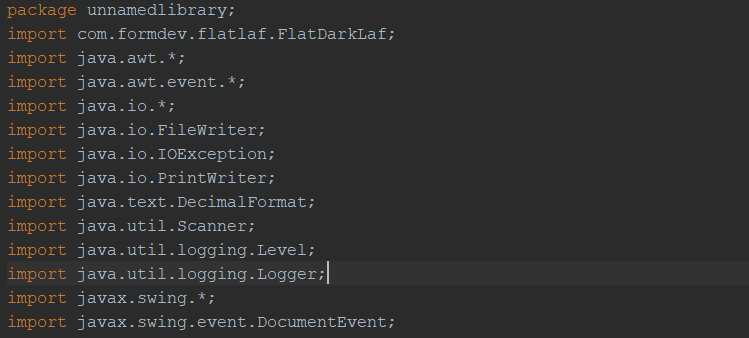


Figure 71: List of imports made by unnamedClientMenu JForm file.

The list of imports as seen in the figure above are referenced by the codes of this particular file. In the case of when an import is unused, the line would be highlighted to notify the developer of redundant or unused import, and it can be deleted depending on the IDE used. To give an example of the packages used by this file, java.awt.\* package is imported to use GUI-related objects such as JTextField while java.awt.event.\* is for event handling methods. Besides built-in packages, the system is also implementing an external package, or libraries as people referred to it online, which is the first import, com.formdev.flatlaf.FlatDarkLaf to provide a unique dark mode Look and Feel to the system.

## Exception Handling

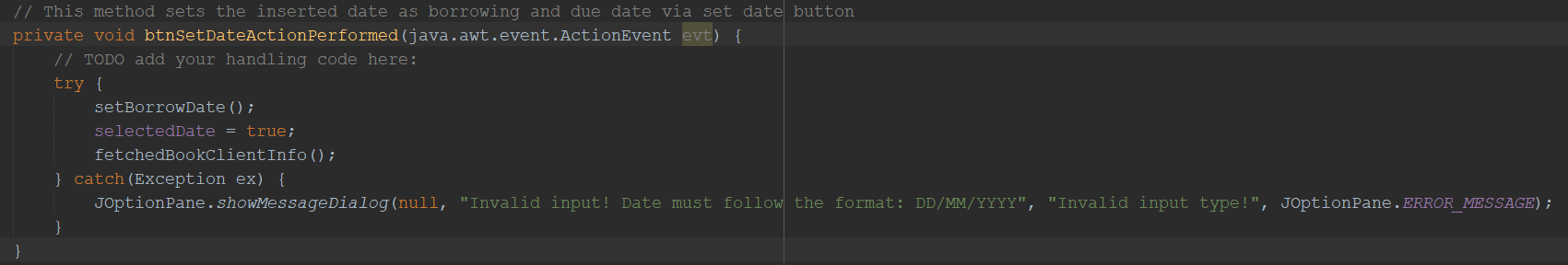


Figure 72: A try-catch block to attempt parsing borrowing date.

In Java, exception handling is a crucial implementation that any developer must use to ensure a smooth user experience when using the application. Say if an application were to have a number that must be divided by 0, clearly this is impossible for human to calculate and the same applies to a computer. When this happens, a runtime exception, specifically arithmetic exception, is thrown to notify the user or developer of the impossibility for the division to produce an answer. In that case, a developer must have an oversight and produce a countermeasure to prevent a thrown exception from breaking user experience. The simplest solution to the impossible division would be to create a popup and tell the user that division by 0 is not possible and stop the calculation from resuming. The same principle is applied in the screenshot above because a Date object cannot parse an empty or incorrectly formatted string. Therefore, a try-catch statement is encapsulating that particular block of code to protect it from faulty parse. If a parse with null or incorrectly formatted string happens, the method stops processing the lines and trigger the catch statement which is to notify the librarian of invalid format for date string.

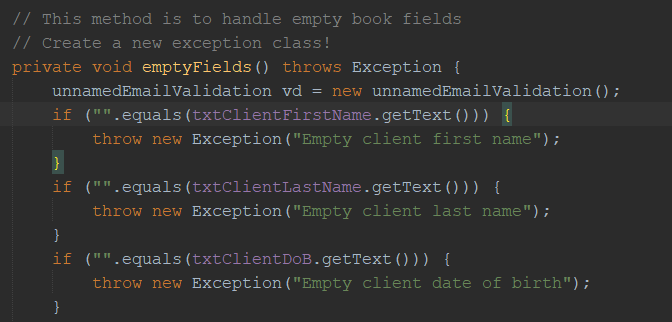


Figure 73: Method which throws exception when the if-statements are true.

For the screenshot above, a private access-level method is created and set to void as it only triggered to run multiple series of if-statements. Only difference with a normal method is that this method has the keyword “throws Exception” included in its declaration. This keyword creates a method that allows for developer to explicitly cause an exception to be thrown when it is called. As for the library system, the method is meant to check if one or more of the fields are empty and an exception will be thrown. This method is not working on its own and it requires another method with a try-catch statement to properly catch the exception thrown.

## Event Handling

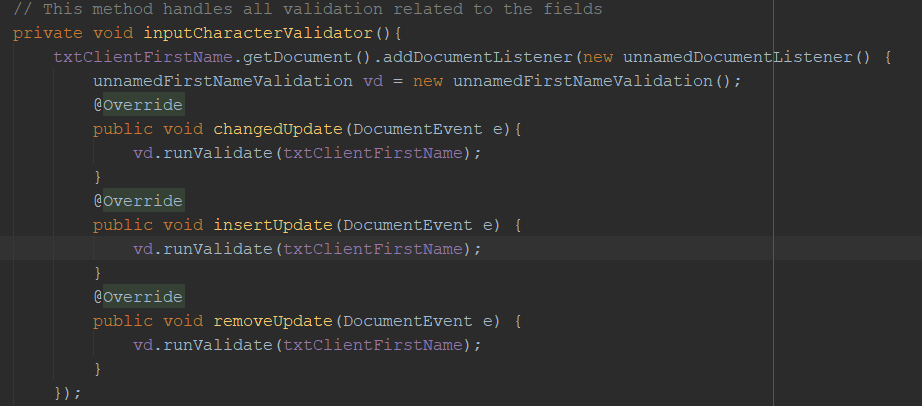


Figure 74: Anonymous instance of unnamedDocumentListener class to perform validation on txtClientFirstName text field.

In Java, an object may have its properties or states changing from one occurrence to another. These changes can be “listened” for by the system and if needed, trigger an event through the event handler. To give an illustration on what is event, mouse clicking the close button of a window frame is an event that can be listened through an interface called WindowListener. As for the screenshot above, similar concept is applied but instead of listening for a window event, it is listening for input changes event inside a text field. The JTextField object named txtClientFirstName is attached a special listener called DocumentListener class via anonymous class.

Anonymous class is another kind of implementation for event handling which does not require all of its methods to be overridden by the child class. Therefore, the anonymous class above is allowing for the unnamedDocumentListener instance to listen for input changes of txtClientFirstName and fire off one of the methods inside of it when triggered. This style of execution allows for what is called real-time validation as the vd.runValidate() method is set to look for illegal characters from the input string and delete them after notifying the librarian.

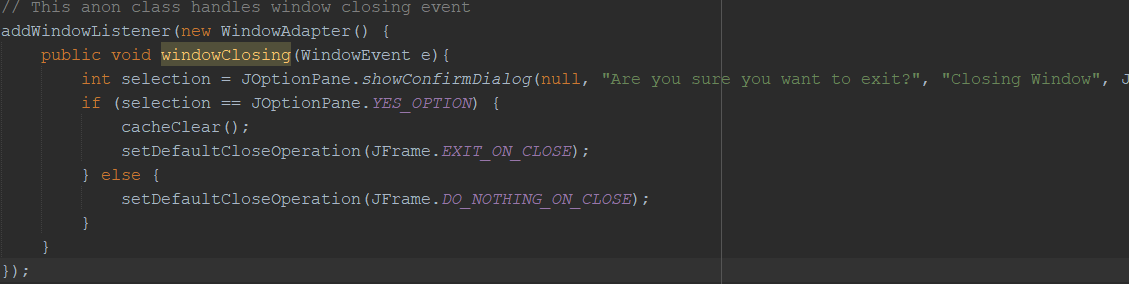


Figure 75: Anonymous instance of window adapter class to only listen to window closing event.

For the figure above, it is another example of anonymous class implementation that handles a WindowListener interface which is to check for events around window frame behaviour. For instance, WindowListener can be triggered when the window is about to be closed, maximized, minimized and many more. An anonymous class was used in this case because only the method windowClosing is required for this WindowListener since the system only wants to listen and act when the window is closing and inheriting other, unused methods found in WindowEvent class is pointless and redundant. In the system’s case, a popup is set to appear when the close button is pressed, giving the librarian a chance to cancel the close action or proceed with terminating the application depending on the option chosen.

# **Additional Features**

## Registering as Librarian

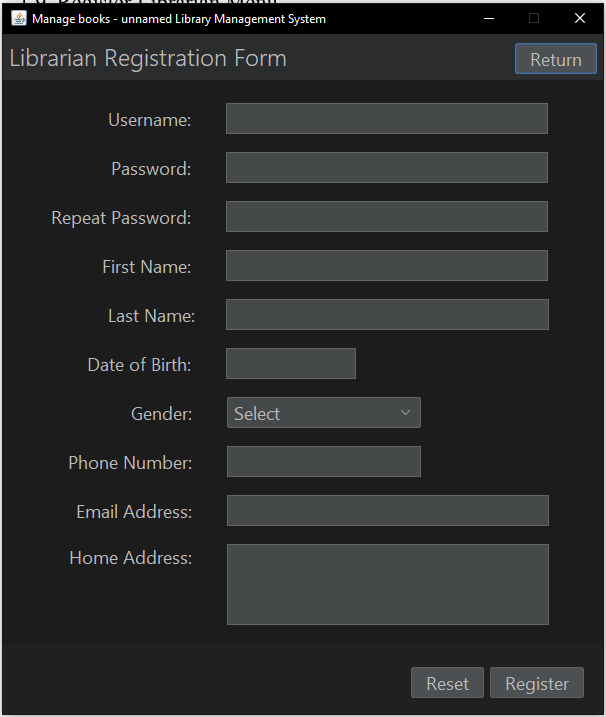


Figure 76: Librarian registration form of unnamed Library Management System

The form above is a feature to allow for librarians to register an account which can be used to login to the system and assume librarian privilege. The librarian is required to fill-in their details into respective input fields.

## Logging in as Librarian

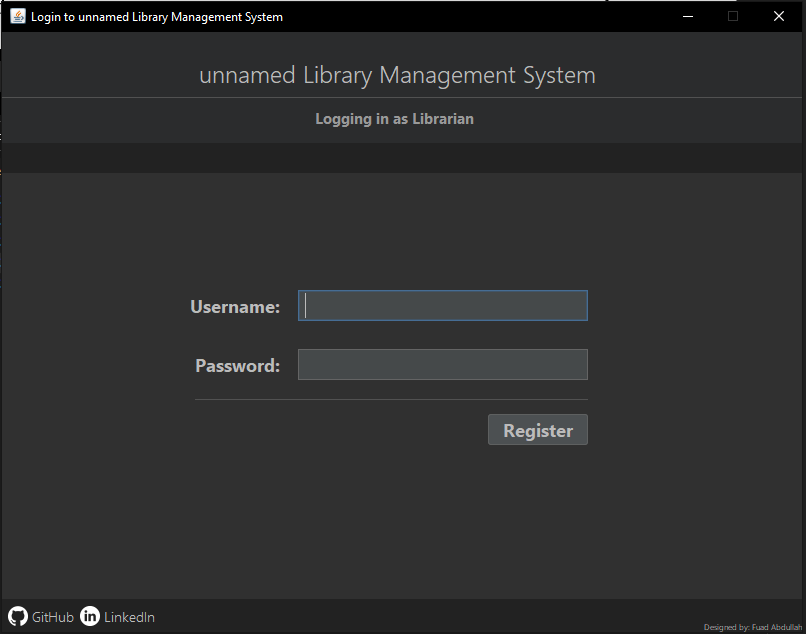


Figure 77: Login menu of unnamed Library Management System

The form above is the login menu a librarian must use to authorize themselves by using the credentials provided to them after a complete registration is performed.

## Adding, Updating and Deleting Client Records

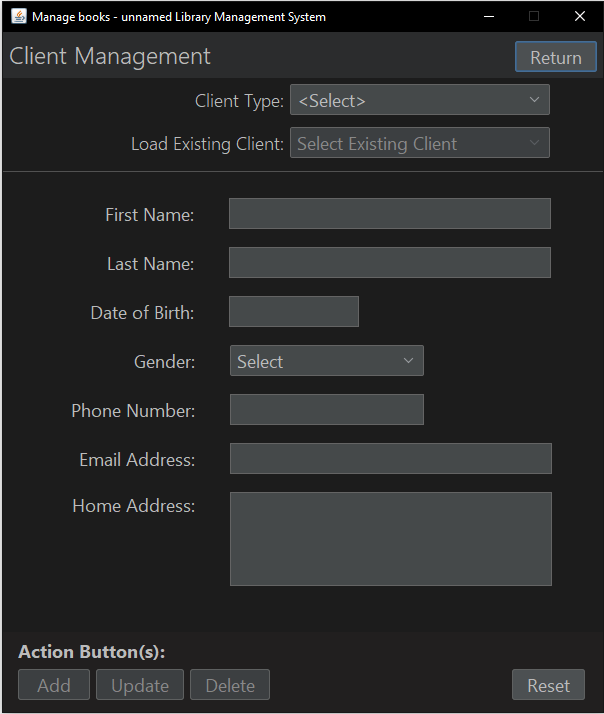


Figure 78: Client management menu of unnamed Library Management System

The form above is client management menu where a librarian is permitted to perform administrative actions to an existing or new client record including add, update, and delete.

## Adding, Updating and Deleting Book Records

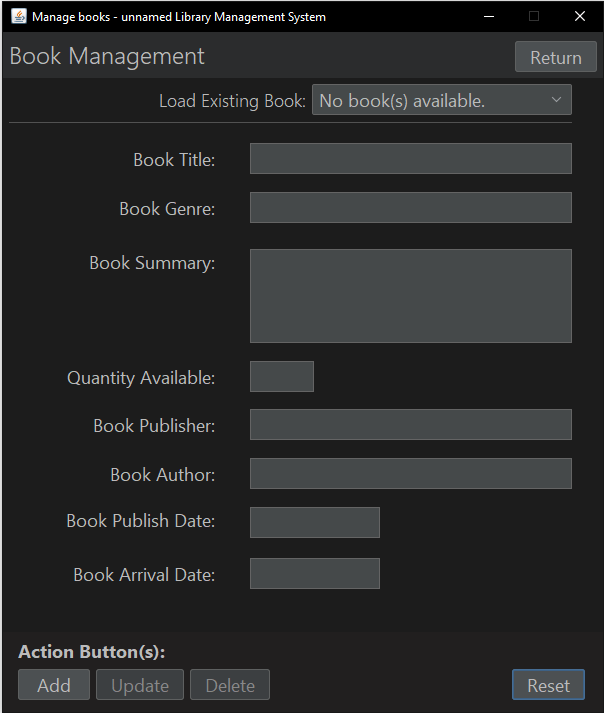


Figure 79: Book management menu of unnamed Library Management System

The form above is book management menu where a librarian is permitted to perform administrative actions to an existing or new book record including add, update, and delete.

## Viewing Past Borrowing Records

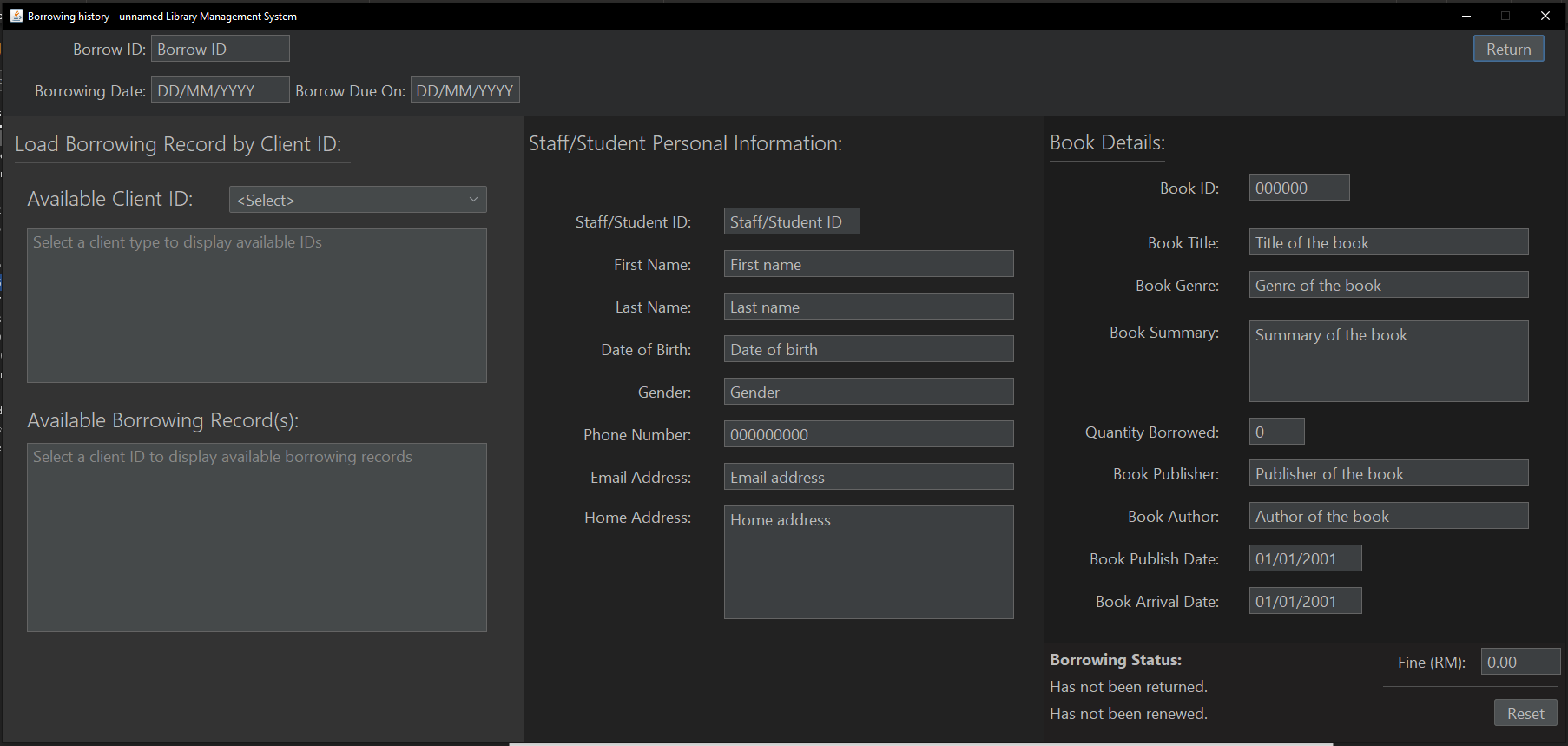


Figure 80: Borrowing history menu of unnamed Library Management System

The form above is where records of past and ongoing borrowing by clients are displayed through the use of client and borrowing ID.

# **Assumptions**

## System-oriented

* The system only caters to a single login-able user type which is the librarian.
* The system management is decentralized, allowing for any librarian to assume administrative privilege of adding, deleting, and updating book and client records.
* The system does not have a centralized administration system for librarian’s accounts management due to absence of administrator role.
* The system displays all of its forms in fixed-size view whereby intentional resize is not possible.
* The system can be terminated at any time by clicking on close button in the window’s frame.
* The system has no countermeasure against accidental application termination in which it led to the loss of information from inside the text files.
* The system has no backup measures for its text files.
* The system shall notify the user of the system of imminent database restart in the case of when no client, book or librarian record available.

## Business model

* The library only offers books as borrowable items.
* A borrowed book is subjected to 14 days (2 weeks) borrowing period by default.
* A borrowed book cannot have its borrowing period changed from its default.
* A borrowed book can be turned in starting from 7th day of the borrowing period.
* A borrowed book can only be renewed its borrowing period for another 14 days (2 weeks) once.
* A borrowed book that has passed its due date is not eligible for renewal and must be returned once fine is cleared.
* A borrowed book that has fine must have its fine cleared prior to returning.
* Only a single book of the same title can be borrowed per borrowing record.
* Fines are only payable over the counter at the library.

## Miscellaneous

* A librarian is required to read the policy of book return to the client during returning process.
* A deleted client record will prevent a client with ongoing borrowing from returning the borrowed book.
* A deleted book record will prevent renewal of ongoing borrowing.
* A deleted book record still allows for book return.
* A deleted book record will have its information unavailable for display when requested by an ongoing borrowing record.
* A librarian must be aware of the system’s limitation when dealing with deleted book.
* A deleted client record will prevent a client with ongoing borrowing from returning the borrowed book.
* A deleted client record will prevent renewal of ongoing borrowing.
* A deleted client record will prevent book return.
* A deleted client record will have its information unavailable for display in any of the pages’ combo boxes.
* A librarian must be aware of the system’s limitation when dealing with deleted client.
* A librarian should not delete a book or client record when there is an ongoing borrowing to prevent accessibility and servicing issues.
* A librarian may delete a book or client record at any time, but the consequences of deletion is irreversible without explicit access to the text files.
* A renewed borrowing record will have its initial borrowing date overridden with the new borrowing date.
* A renewed borrowing record will have its initial due date overridden with the new due date.

# **References**

FormDev.com, n.a.. *FlatLaf - Flat Look and Feel.* [Online] Available at: <https://www.formdev.com/flatlaf/> [Accessed 12 March 2021].

sanbhat, 2013. *Java - Check if JTextField is empty or not.* [Online] Available at: <https://stackoverflow.com/questions/17132452/java-check-if-jtextfield-is-empty-or-not> [Accessed 12 March 2021].

Hovercraft Full Of Eels, 2012. *Java equivalent to C# TextBox TextChanged event.* [Online] Available at: <https://stackoverflow.com/questions/8721197/java-equivalent-to-c-sharp-textbox-textchanged-event> [Accessed 12 March 2021].

Scopin, S., 2012. *java documentlistener.* [Online] Available at: <https://stackoverflow.com/questions/10757555/java-documentlistener> [Accessed 12 March 2021].

ibrabelware, 2012. *JLabel hyperlink to open browser at correct URL.* [Online] Available at: <https://stackoverflow.com/questions/8669350/jlabel-hyperlink-to-open-browser-at-correct-url> [Accessed 12 March 2021].

JasCav, 2011. *Programmatically clicking a GUI button in Java Swing.* [Online] Available at: <https://stackoverflow.com/questions/5109274/programmatically-clicking-a-gui-button-in-java-swing> [Accessed 12 March 2021].

Freepik.com, n.d.. *Graphic resources for everyone.* [Online] Available at: <https://www.freepik.com/> [Accessed 12 March 2021].

hoapham, 2018. *How to define a relative path in java.* [Online] Available at:  [https://stackoverflow.com/questions/14209085/how-to-define-a-relative-path-in-java/](https://www.freepik.com/) [Accessed 12 March 2021].

dogbane, 2011. *Writing to a file in Java: Cannot find the path specified.* [Online] Available at:  [https://stackoverflow.com/questions/7095034/writing-to-a-file-in-java-cannot-find-the-path-specified](https://www.freepik.com/) [Accessed 12 March 2021].

Landup, D., n.d., *How to Get Current Date and Time in Java.* [Online] Available at: <https://stackabuse.com/how-to-get-current-date-and-time-in-java/> [Accessed 12 March 2021].

Singh, C., n.d., *Java – Add days to Date.* [Online] Available at: <https://beginnersbook.com/2017/10/java-add-days-to-date/> [Accessed 12 March 2021].

W3Schools.com, n.d., *Java Delete Files.* [Online] Available at: <https://www.w3schools.com/java/java_files_delete.asp> [Accessed 12 March 2021].

Pierre, 2009, *Rename a file using Java.* [Online] Available at: <https://stackoverflow.com/questions/1158777/rename-a-file-using-java> [Accessed 12 March 2021].

Gurung, B., 2012, *How to remove single character from a String.* [Online] Available at: <https://stackoverflow.com/questions/13386107/how-to-remove-single-character-from-a-string> [Accessed 12 March 2021].

Raja, 2019, *How to compare two dates in Java?* [Online] Available at: <https://www.tutorialspoint.com/how-to-compare-two-dates-in-java> [Accessed 12 March 2021].

Peters, M., 2010, *Empty an array in Java / processing.* [Online] Available at: <https://stackoverflow.com/questions/4208655/empty-an-array-in-java-processing> [Accessed 12 March 2021].

Skeet, J., 2011, *How do I get a Date without time in Java?* [Online] Available at: <https://stackoverflow.com/questions/5050170/how-do-i-get-a-date-without-time-in-java> [Accessed 12 March 2021].

Joda.org, 2021, *Pattern-based formatting.* [Online] Available at: <https://www.joda.org/joda-time/key_format.html> [Accessed 12 March 2021].

Joda.org, 2021, *Available Time Zones.* [Online] Available at: <https://www.joda.org/joda-time/timezones.html> [Accessed 12 March 2021].

DeltaLima, 2013, *How to format Joda-Time DateTime to only mm/dd/yyyy?* [Online] Available at: <https://stackoverflow.com/questions/20331163/how-to-format-joda-time-datetime-to-only-mm-dd-yyyy/20331243> [Accessed 12 March 2021].

Gay, H., 2010, *Joda time : How to convert String to LocalDate?* [Online] Available at: <https://stackoverflow.com/questions/2721614/joda-time-how-to-convert-string-to-localdate> [Accessed 12 March 2021].

Joda.org, 2021, *Why Joda Time?* [Online] Available at: <https://www.joda.org/joda-time/> [Accessed 12 March 2021].

Henley, A., 2013, *how to read last line in a text file using java [duplicate].* [Online] Available at: <https://stackoverflow.com/questions/17509781/how-to-read-last-line-in-a-text-file-using-java/17509803> [Accessed 12 March 2021].

Jain, R., 2013, *An elegant way of initialising an empty String array in Java.* [Online] Available at: <https://stackoverflow.com/questions/14446539/an-elegant-way-of-initialising-an-empty-string-array-in-java> [Accessed 12 March 2021].

Handcoded.co.uk, n.d., *Repeat Text Generator.* [Online] Available at: <https://www.handcoded.co.uk/php/app-1/> [Accessed 12 March 2021].

Kasyap, K., 2019, *When does a Java Array Throws a NullPointerException?* [Online] Available at: <https://www.tutorialspoint.com/when-does-a-java-array-throws-a-nullpointerexception> [Accessed 12 March 2021].

Singh, C., n.d., *How to convert a char array to a string in Java?* [Online] Available at: <https://beginnersbook.com/2014/06/how-to-convert-a-char-array-to-a-string-in-java/> [Accessed 12 March 2021].

CBC\_NS, 2013, *How to create a Jar file in Netbeans.* [Online] Available at: <https://stackoverflow.com/questions/9681876/how-to-create-a-jar-file-in-netbeans> [Accessed 12 March 2021].

mkyong, 2020, *How to create directory in Java.* [Online] Available at: <https://mkyong.com/java/how-to-create-directory-in-java/> [Accessed 12 March 2021].

moeTi, 2013, *Getting 'Attempt to mutate notification' exception.* [Online] Available at: <https://stackoverflow.com/questions/15206586/getting-attempt-to-mutate-notification-exception> [Accessed 12 March 2021].

slackwing, 2013, *Regular expression to allow spaces between words.* [Online] Available at: <https://stackoverflow.com/questions/15472764/regular-expression-to-allow-spaces-between-words> [Accessed 12 March 2021].

Buberel, J., 2011, *Java regex email.* [Online] Available at: <https://stackoverflow.com/questions/8204680/java-regex-email> [Accessed 12 March 2021].

W3School.com, n.d., *Java Abstraction.* [Online] Available at: <https://www.w3schools.com/java/java_abstract.asp> [Accessed 12 March 2021].

W3School.com, n.d., *Java Encapsulation.* [Online] Available at: <https://www.w3schools.com/java/java_encapsulation.asp> [Accessed 12 March 2021].

seow, 2018, *JTextArea pressing enter adds unnecessary new line.* [Online] Available at: <https://stackoverflow.com/questions/19569302/jtextarea-pressing-enter-adds-unnecessary-new-line> [Accessed 13 March 2021].

rtheunissen, 2013, *remove all special characters in java [duplicate].* [Online] Available at: <https://stackoverflow.com/questions/14361556/remove-all-special-characters-in-java> [Accessed 13 March 2021].