CITY BOOK SHOP

Object Oriented Programming

AUGUST 16, 2021

MN RIZNI MOHAMED

JF HDCSE CMU 09 21

ST 20195902

Assignment Cover Sheet

Qualification	Module Number and Title
HD in Computing and Software Engineering	**OOP-** CSE4006
Student Name & No.	**Assessor**
MN RIZNI MOHAMED JF HDCSE CMU 09 21 ST 20195902	Miss Ushamini Kulas

Hand out date	Submission Date
13/06/2021	21/8/2021

Assessment type WRIT1-Coursework	**Duration/Length of** **Assessment Type** 3000 words equivalent	**Weighting of Assessment** 100%

Learner declaration
I certify that the work submitted for this assignment is my own and research sources are fully acknowledged.

Marks Awarded			
First assessor			
IV marks			
Agreed grade			
Signature of the assessor		Date	

FEEDBACK FORM
INTERNATIONAL COLLEGE OF BUSINESS & TECHNOLOGY

Module:	OOP - CSE4006
Student:	MN RIZNI MOHAMED JF HDCSE CMU 09 21 ST 20195902
Assessor:	Miss Ushamini Kulas
Assignment:	Object oriented programming

Strong features of your work:

Areas for improvement:

Marks Awarded:

Acknowledgement

In the first place, I'd want to thank God Almighty and my parents for their encouragement and support. Our sincere gratitude goes out to Miss Ushamini Kulas, who educated us and supervised us throughout this process. It would not have been possible without their invaluable assistance and valuable experience. Apart from that, I'd want to express our gratitude to the ICBT administration for granting us permission and providing us with space to finish our project. Last but not least, I apologize to everyone else who assisted us in many ways to have a nice time.

List of Contents

List of Tables

List of Figures

City Book Shop

1. Introduction

City book shop is the well-known bookshop in Puttalam. There are several kinds books available in affordable price. Due to higher number of customers visit they face difficulties in transaction management. Therefore, they decided to automate the entire transaction system. They come with for create a software for all transaction management process.

2. Objectives of the system

Main objective of this system is automating and computerize entire transaction system to manage customer traffic in efficient manner. For that creating a software which include various features according employee's needs. There are two types of employees working in city book shop. Those are admin and cashier. Each type of employee has different task respective to their job role.

Main requirements:

Cashier	Admin
View all book details	View all book details
Add new book details and category	Add new book details and category
Search book details based on category, Name, Price.	Search book details based on category, Name, Price.
	Create accounts

Table 1 Main requirements

Other requirements:

Cashier	Admin
View Statistics	View Statistics
Login and forget password	Login and forget password
Create, view, search invoice	Create, view, delete, update search invoice
Update, view own account	Update, search, view, delete accounts

Table 2 Other requirements

3. Design of solution

All architectural modules and the communication and flow representation with the external and third-party modules are explicitly defined in a design approach (if any). In the design document specification, the internal design of all modules of the proposed architecture should be fully specified with the greatest possible detail (DDS).

(SDLC - Overview, 2021)

3.1. Use case diagram

The dynamic conduct of a system is represented by a use case diagram. It embodies the functioning of the system through the integration of use cases, actors and their relations. The system/subsystem of an application modeling the tasks, services and functions that are required. It shows the system's high-level capabilities and how the user manages a system as well.

(UML Use Case Diagram - Javatpoint, 2021)

3.1.1. City bookshop use-case diagram

Double click the picture for view in Microsoft Viso for clear visuals

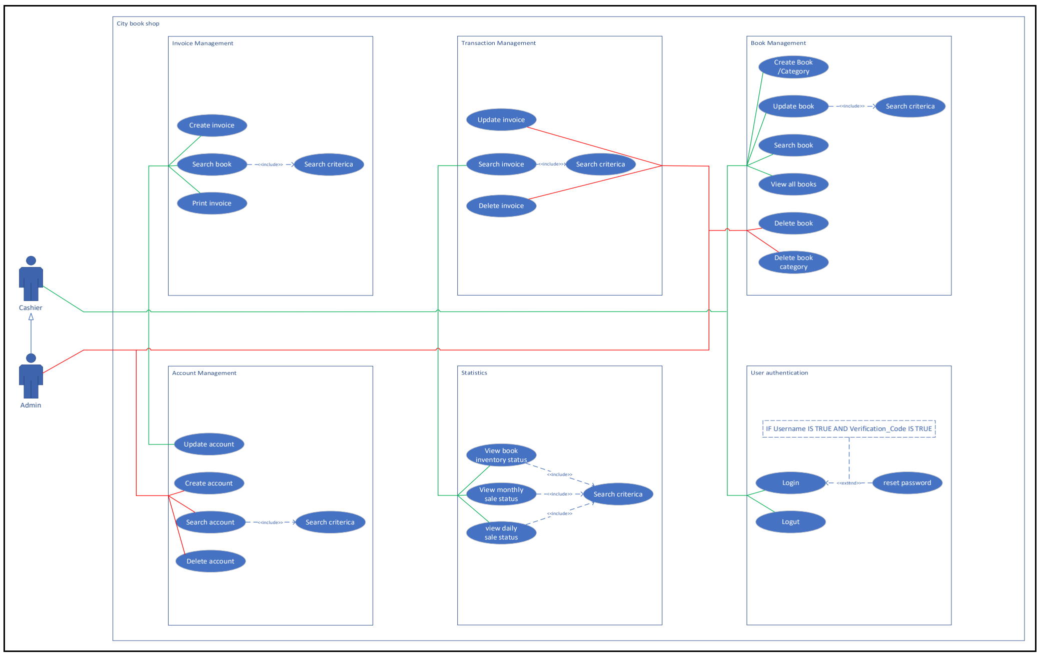


Figure 1 Use-case diagram

Assumptions:

- Admin and cahier have an account
- Admin and cashier have email addresses
- System has proper internet connection

Above use case diagram depicts the subsystems of city bookshop and system features. And also, it shows the capacities of employees according to their job roles. City book shop have 6 subsystems in it for perming their tasks. Those are Invoice management, Transaction management, Book management, Account management, Statistics and finally User authentication. Each subsystem has several functionalities related that system process. Those functionalities listed in below. See table 3.

Subsystem	**Functionalities**
Invoice management	Create an invoice, Search book, print an invoice
Transaction management	Update an invoice, Search an invoice, Delete an invoice
Book management	Create book/category, Update book, Search book, view all books, Delete book, Delete book category
Account management	Update an account, create an account, Search an account, Delete an account
Statistics	View book inventory status, view monthly sale details, view daily status
User authentication	Login, logout, forget password

Table 3 Sub systems and functionalities

In the invoice management admin/cashier can search a book by book's name, book's category, book's unit price. In transaction management admin/cashier can search a invoice by book's category, quantity, invoice's ID, book's name, total price and date. In book management admin/cashier can search a book by book's name, book's category, book's unit price, book's stock quantity and book's ID. In account management admin can search a account by account type, account ID, username, name and email address. In statistics system, admin/cashier can view book inventory details by selecting book category, also they can view monthly sales details by selecting year and can view daily sale's details by selecting a date. Finally, in user authentication if admin or cashier forget their password then can reset their password via reset password protocol. For confirming the user, for their email address an verification code will be send then they have to enter same code as they got via email in the system then they can set new password for their account

A cashier has the access to all features of user authentication system, statistics system, invoice management system. But for the security purpose cashier has access to some features in other systems. In the transaction management cashier has the access to search invoice only. In the book management system cashier has the access to create book/category, update book, search book, view all book details. In the account management cashier has the access to update their own account only

An admin has the full access to the system. So, admin inheriting all the access from cashier and additional to that in transaction management an admin can update or delete the transaction. Like vise, in book management admin can delete the book and book category and in account management an admin can update, delete, search and view any account also an admin can create new type of user accounts

By this use case diagram, it helps to understand the motive of the system and objectives of the system. By this use case we can justify our developed which full filled the requirements or not.

3.2. Class diagram

Unified Modeling Language (UML) class diagrams are static structural diagrams that depict the system's classes, attributes, actions (or methods), and connections between classes.

3.2.1. City bookshop use case diagram

Click the picture while pressing ctrl key for view in clear visuals.

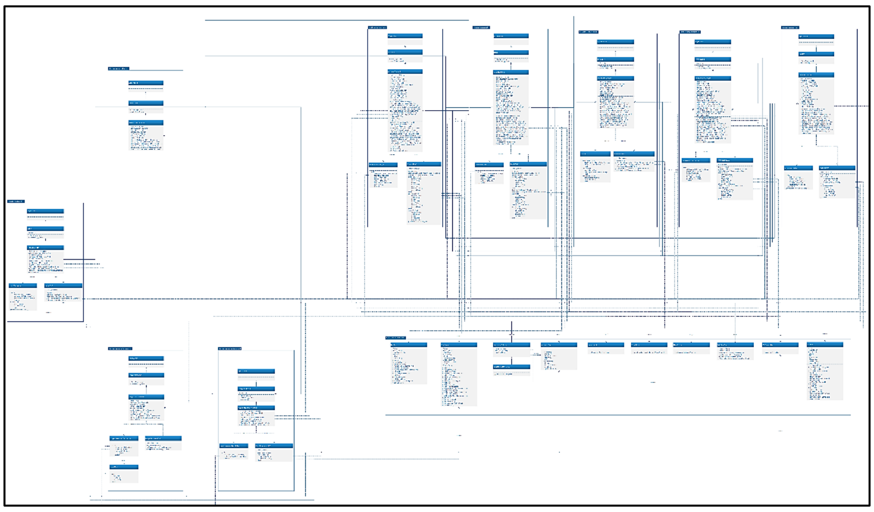


Figure 2 Class diagram

Assumptions:

- All system classes are inherited
- All external libraries such as MySQL connecter, JavaFX are associated

The above class diagram shows, there are 10 packages in this system and each packages consist several classes in it. Those are clearly mentioned in below table.

Package name	**Class name**
rizni.bookshop.login	Application, Login, loginController, LoginDAO, LoginValidation
rizni.bookshop.forgetpassword	Application, Forgetpassword, forgetpassController, ForgetPasswordValidation, ForgetPasswordDAO, SendMail
rizni.bookshop.resetpassword	Application, ResetPassword, resetpasswordController, ResetPasswordValidation, ResetPasswordDAO
rizni.bookshop.dashborad	Application, Dashboard, dashboardController
rizni.bookshop.reuseable	Books, Employee, LoginInDetails, LoginInDetailsDAO, MessageBox, NumberUtils, UserPicture, WinTitleBar, DBConnection, Invoice
rizni.bookshop.invoice	Application, Invoice, invoiceController, InvoiceValidation, InvoiceDAO
rizni.bookshop.transaction	Application, Transaction, TransactionDAO, transactionController, TransactionValidation
rizni.bookshop.statistics	Application, Statistics, statisticsController, Charts, StatisticsDAO
rizni.bookshop.book	Application, Book, bookController, BookValidation, BookDAO
rizni.bookshop.forgetpassword	Application, Account, accountController, AccountValidation, AccountDAO

Table 4 Package name and class name

The program starts with login unit. When user enter correct username and password then it’s moved into the dashboard unit. When user forget the password then it goes to forget password unit for the verification process and then it goes to resetpassword unit for set new password and finally it will go for dashboard unit. There is a reusable unit for the reusable components. Those components have use dependency with other unit's classes. Dashboard, reusable, invoice, transaction, statistics, book, forgetpassword units are have interconnection with themselves which can lead users to go between these units and additionally it leads to login unit too. All the unit's main class is inherited to the Application class and all controller classes are composite to their main class because without main class there is no life cycle for controller classes and also without start function which coming from Application, JavaFX will not start the application. And also, in reusable unit LoggedInUserDAO class inherit LoggedInUser class for access the variables.

By this class diagram, we can clearly get know about what are the classes are need to build this system and how to organize all the class within packages. And also, by this diagram we able to know how to declare the classes and what are the variables and methods need to declare in each class. So, this diagram gives clear picture about declarations.

3.3. Sequence diagram

As the name implies, it shows the order in which items interact with one other in a sequence diagram. Sequence diagrams can also be referred to as event diagrams or event scenarios. In a sequence diagram, the items in a system are described in terms of how they work and in what order. Businessmen and software engineers use these diagrams to document and understand requirements for new and current systems.

(Unified Modeling Language (UML) | Sequence Diagrams - GeeksforGeeks, 2021)

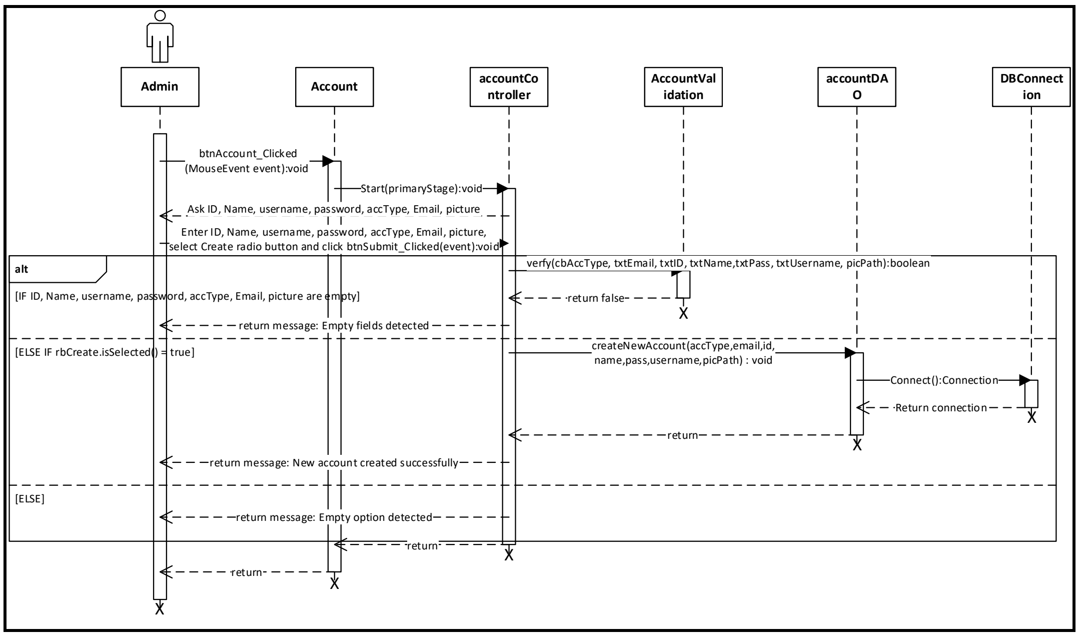
Subsystem	Use case ID	Use case
Account management	3.3.1.	Create account
	3.3.2.	Delete account
	3.3.3.	Search account
	3.3.4.	Update account
Book management	3.3.5.	Create book
	3.3.6.	Delete book
	3.3.7.	Delete book category
	3.3.8.	Search book
	3.3.9.	Update book
	3.3.10.	View all book category
Invoice management	3.3.11.	Create invoice
	3.3.12.	Print invoice

	3.3.13.	Search book
Statistics	3.3.14.	Book inventory
	3.3.15.	Daily sales
	3.3.16.	Monthly sales
Transaction management	3.3.17.	Delete invoice
	3.3.18.	Search invoice
	3.3.19.	Update invoice
User authentication	3.3.20.	Login
	3.3.21.	Logout
	3.3.22.	Reset password

Table 5 Sequence diagram use case specification

3.3.1. Create account

Double click the picture for view in Microsoft Viso for clear visuals



Admin
Account
accountController
AccountValidation
accountDAO
DBConnection
btnAccount_Clicked (MouseEvent event):void
Start(primaryStage):void
Ask ID, Name, username, password, accType, Email, picture
Enter ID, Name, username, password, accType, Email, picture, select Create radio button and click btnSubmit_Clicked(event):void
alt
verfy(cbAccType, txtEmail, txtID, txtName,txtPass, txtUsername, picPath):boolean
[IF ID, Name, username, password, accType, Email, picture are empty]
return false
return message: Empty fields detected
[ELSE IF rbCreate.isSelected() = true]
createNewAccount(accType,email,id, name,pass,username,picPath) : void
Connect():Connection
Return connection
return
return message: New account created successfully
[ELSE]
return message: Empty option detected
return
return

Figure 3 Sequence diagram create account

Assumptions:

- Database is created
- employee, login and role tables are created
- Admin have an account
- Admin logged in correct username and password

The above sequence diagram depicts about creating a new account in the system. Admin can create new type of account too. In first admin need to fill all filed in correct manner. For an example, account ID field cannot contain character values in that case it will throw error message to admin. And also, if any fields seemed to be empty or create option is not selected that time also it will throw error message. All the validating processes are held in AccountValidation class. This help to ensure the data accuracy. All the database query operations are done within AccountDAO class.

3.3.2. Delete account

Double click the picture for view in Microsoft Viso for clear visuals

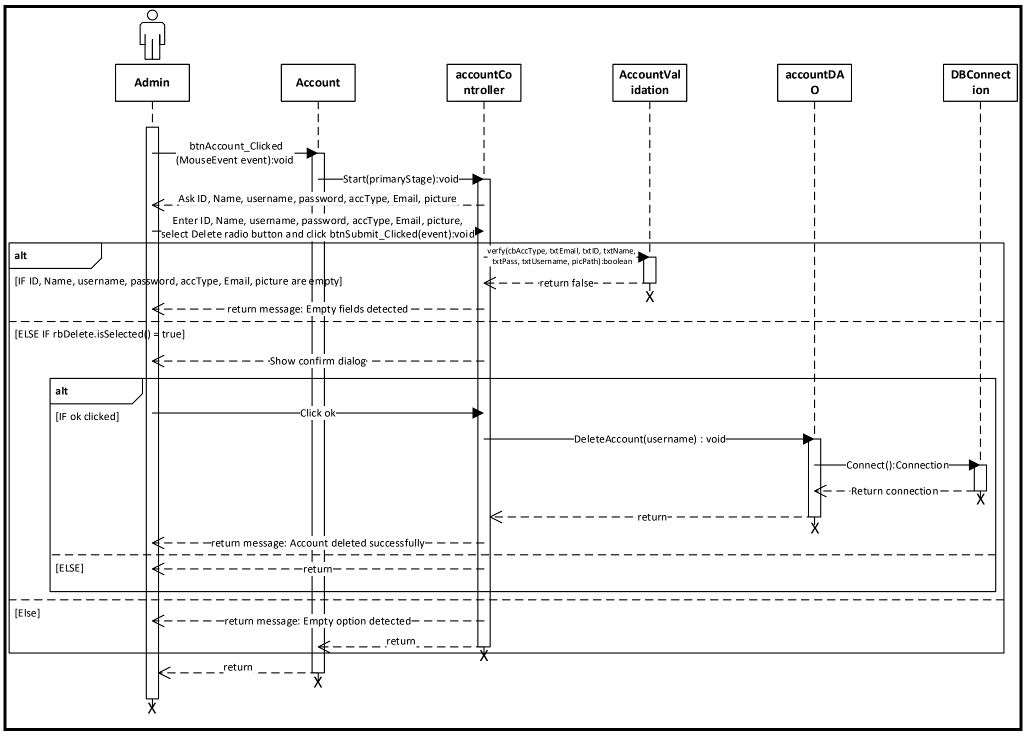

Admin
Account
accountController
AccountValidation
accountDAO
DBConnection
btnAccount_Clicked (MouseEvent event):void
Start(primaryStage):void
Ask ID, Name, username, password, accType, Email, picture
Enter ID, Name, username, password, accType, Email, picture, select Delete radio button and click btnSubmit_Clicked(event):void
alt
verfy(cbAccType, txtEmail, txtID, txtName, txtPass, txtUsername, picPath):boolean
[IF ID, Name, username, password, accType, Email, picture are empty]
return false
return message: Empty fields detected
[ELSE IF rbDelete.isSelected() = true]
Show confirm dialog
alt
[IF ok clicked]
Click ok
DeleteAccount(username) : void
Connect():Connection
Return connection
return
return message: Account deleted successfully
[ELSE]
return
[Else]
return message: Empty option detected
return
return

Figure 4 Sequence diagram - delete account

Assumptions:

- Database is created
- employee, login and role tables are created
- Admin have an account
- Admin logged in correct username and password

The above sequence diagram depicts about deleting an existing account from the system. Admin can only perform this task. In first admin need to fill all filed in correct manner. For an example, account ID field cannot contain character values in that case it will throw error message to admin. And also, if any fields seemed to be empty or delete option is not selected that time also it will throw error message. All the validating processes are held in AccountValidation class. This help to ensure the data accuracy. Also, it will ask confirmation to the admin for a second verification. It reduces the accidental data loss. All the database query operations are done within AccountDAO class.

3.3.3. Search account

Double click the picture for view in Microsoft Viso for clear visuals

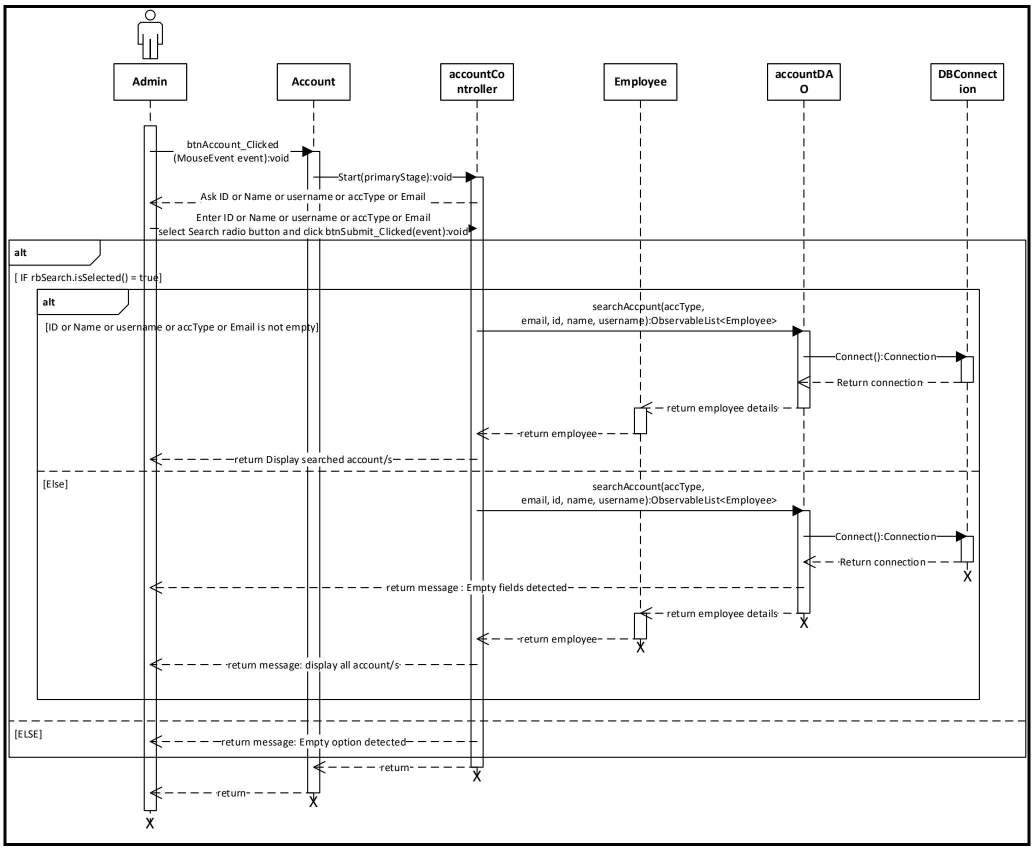

Admin
Account
accountController
Employee
accountDAO
DBConnection
btnAccount_Clicked (MouseEvent event):void
Start(primaryStage):void
Ask ID or Name or username or accType or Email
Enter ID or Name or username or accType or Email
select Search radio button and click btnSubmit_Clicked(event):void
alt
[IF rbSearch.isSelected() = true]
alt
[ID or Name or username or accType or Email is not empty]
searchAccount(accType, email, id, name, username):ObservableList<Employee>
Connect():Connection
Return connection
return employee details
return employee
return Display searched account/s
[Else]
searchAccount(accType, email, id, name, username):ObservableList<Employee>
Connect():Connection
Return connection
return message : Empty fields detected
return employee details
return employee
return message: display all account/s
[ELSE]
return message: Empty option detected
return
return

Figure 5 Sequence diagram - Search account

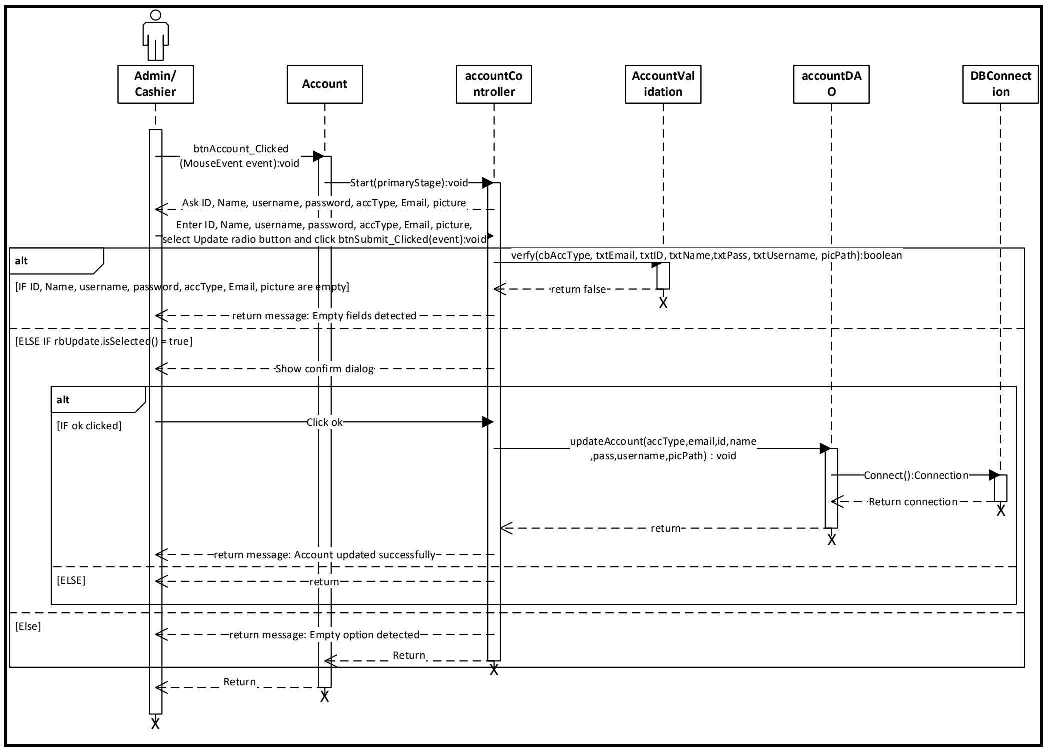
Assumptions:

- Database is created
- employee, login and role tables are created
- Admin have an account
- Admin logged in correct username and password

The above sequence diagram depicts about searching an existing account in the system. Admin can only perform this task. In first admin need to fill all filed in correct manner. For an example, account ID field cannot contain character values in that case it will throw error message. Also, if search option is not selected then it will throw error message. But if any fields seemed to be empty that time it will show all user according to user role permission. When everything is seemed to be correct then it will show searched user account details in the table. All the database query operations are done within AccountDAO class.

3.3.4. Update account

Double click the picture for view in Microsoft Viso for clear visuals



Admin/ Cashier
Account
accountController
AccountValidation
accountDAO
DBConnection
btnAccount_Clicked (MouseEvent event):void
Start(primaryStage):void
Ask ID, Name, username, password, accType, Email, picture
Enter ID, Name, username, password, accType, Email, picture, select Update radio button and click btnSubmit_Clicked(event):void
alt
verfy(cbAccType, txtEmail, txtID, txtName,txtPass, txtUsername, picPath):boolean
[IF ID, Name, username, password, accType, Email, picture are empty]
return false
return message: Empty fields detected
[ELSE IF rbUpdate.isSelected() = true]
Show confirm dialog
alt
[IF ok clicked]
Click ok
updateAccount(accType,email,id,name ,pass,username,picPath) : void
Connect():Connection
Return connection
return
return message: Account updated successfully
[ELSE]
return
[Else]
return message: Empty option detected
Return
Return

Figure 6 Sequence diagram - Update account

Assumptions:

- Database is created
- employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier logged in correct username and password

The above sequence diagram depicts about updating an existing account in the system. Admin or cahier can only perform this task. In first admin or cashier need to fill all filed in correct manner. For an example, account ID field cannot contain character values in that case it will throw error message. And also, if any fields seemed to be empty or update option is not selected that time also it will throw error message. All the validating processes are held in AccountValidation class. This help to ensure the data accuracy. Also, it will ask confirmation to the admin or cashier for a second verification. It reduces the accidental data loss. Cashier can only update their own account but admin can update any account within the system. All the database query operations are done within AccountDAO class.

3.3.5. Create book

Double click the picture for view in Microsoft Viso for clear visuals

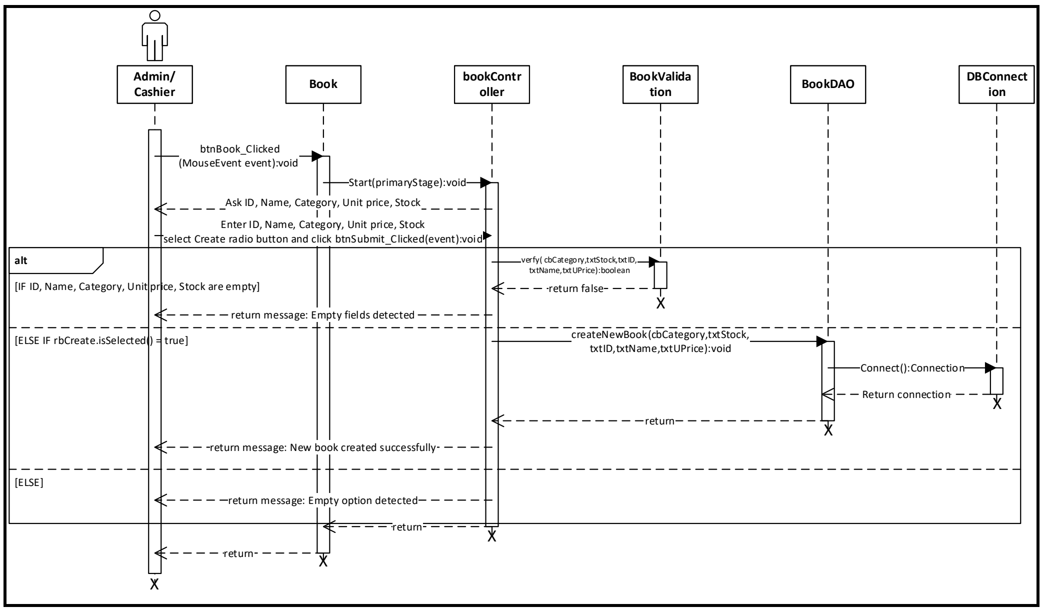

Admin/ Cashier
Book
bookController
BookValidation
BookDAO
DBConnection
btnBook_Clicked (MouseEvent event):void
Start(primaryStage):void
Ask ID, Name, Category, Unit price, Stock
Enter ID, Name, Category, Unit price, Stock select Create radio button and click btnSubmit_Clicked(event):void
alt
verfy(cbCategory,txtStock,txtID, txtName,txtUPrice):boolean
[IF ID, Name, Category, Unit price, Stock are empty]
return false
return message: Empty fields detected
[ELSE IF rbCreate.isSelected() = true]
createNewBook(cbCategory,txtStock, txtID,txtName,txtUPrice):void
Connect():Connection
Return connection
return
return message: New book created successfully
[ELSE]
return message: Empty option detected
return
return
X

Figure 7 Sequence diagram - Create book

Assumptions:

- Database is created
- book, book category, employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier logged in correct username and password

The above sequence diagram depicts about creating a new book in the system. Admin or cashier can create new book category too. In first they need to fill all fields in correct manner. For an example, book ID field cannot contain character values in that case it will throw error message. And also, if any fields seemed to be empty or create option is not selected that time also it will throw error message. All the validating processes are held in BookValidation class. This help to ensure the data accuracy. All the database query operations are done within BookDAO class.

3.3.6. Delete book

Double click the picture for view in Microsoft Viso for clear visuals

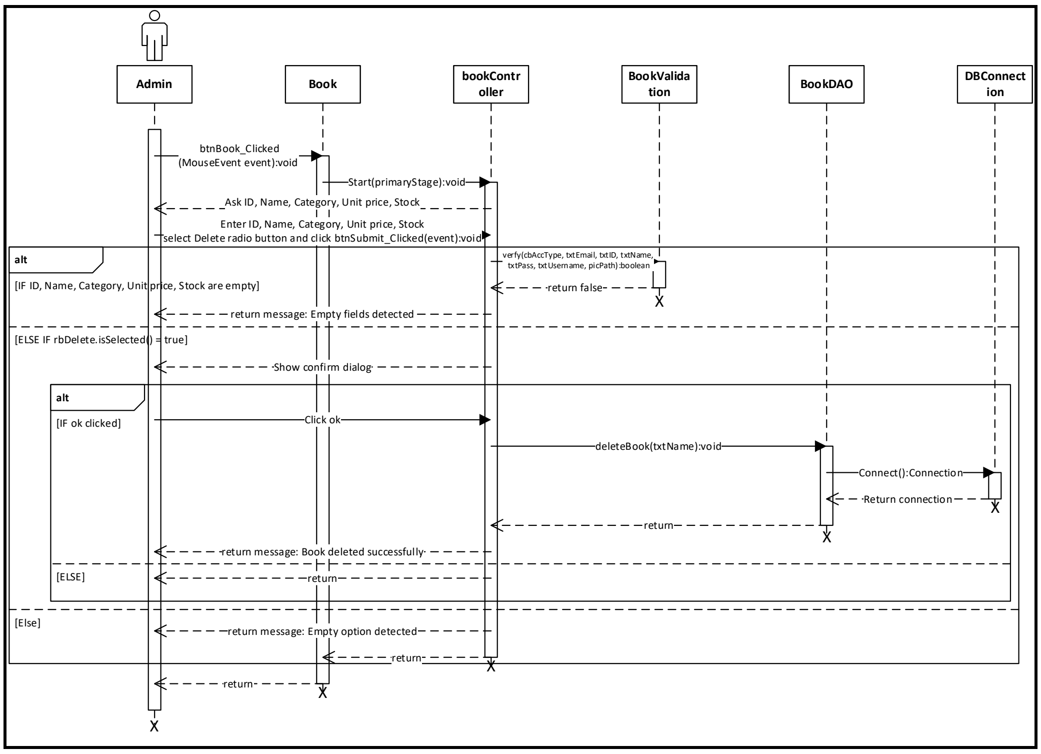

Admin
Book
bookController
BookValidation
BookDAO
DBConnection
btnBook_Clicked (MouseEvent event):void
Start(primaryStage):void
Ask ID, Name, Category, Unit price, Stock
Enter ID, Name, Category, Unit price, Stock select Delete radio button and click btnSubmit_Clicked(event):void
alt
[IF ID, Name, Category, Unit price, Stock are empty]
verfy(cbAccType, txtEmail, txtID, txtName, txtPass, txtUsername, picPath):boolean
return false
return message: Empty fields detected
[ELSE IF rbDelete.isSelected() = true]
Show confirm dialog
alt
[IF ok clicked]
Click ok
deleteBook(txtName):void
Connect():Connection
Return connection
return
return message: Book deleted successfully
[ELSE]
return
[Else]
return message: Empty option detected
return
return

Figure 8 Sequence diagram - Delete book

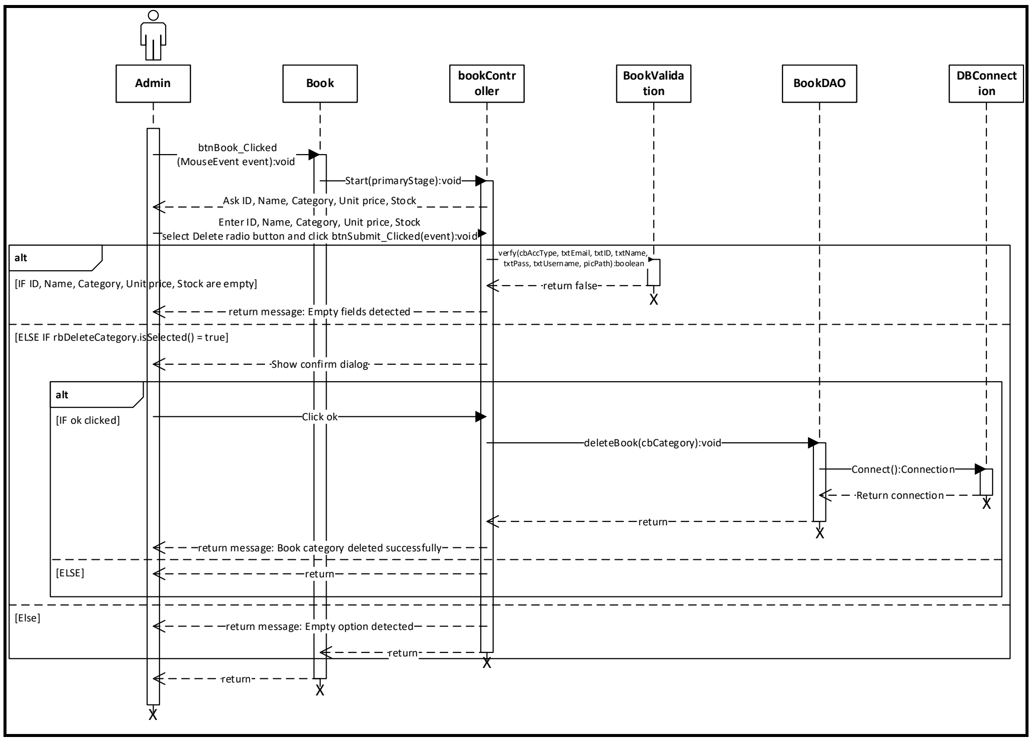
Assumptions:

- Database is created
- book, book category, employee, login and role tables are created
- Admin have an account
- Admin logged in correct username and password
- Some book's details entered in the system

The above sequence diagram depicts about deleting an existing book from the system. Admin can only perform this task. In first admin need to fill all filed in correct manner. For an example, book ID field cannot contain character values in that case it will throw error message. And also, if any fields seemed to be empty or delete option is not selected that time also it will throw error message. All the validating processes are held in BookValidation class. This help to ensure the data accuracy. Also, it will ask confirmation to the admin for a second verification. It reduces the accidental data loss. All the database query operations are done within BookDAO class.

3.3.7. Delete book category

Double click the picture for view in Microsoft Viso for clear visuals



Admin
Book
bookController
BookValidation
BookDAO
DBConnection
btnBook_Clicked (MouseEvent event):void
Start(primaryStage):void
Ask ID, Name, Category, Unit price, Stock
Enter ID, Name, Category, Unit price, Stock
select Delete radio button and click btnSubmit_Clicked(event):void
alt
verfy(cbAccType, txtEmail, txtID, txtName, txtPass, txtUsername, picPath):boolean
[IF ID, Name, Category, Unit price, Stock are empty]
return false
return message: Empty fields detected
[ELSE IF rbDeleteCategory.isSelected() = true]
Show confirm dialog
alt
[IF ok clicked]
Click ok
deleteBook(cbCategory):void
Connect():Connection
Return connection
return
return message: Book category deleted successfully
[ELSE]
return
[Else]
return message: Empty option detected
return
return

Figure 9 Sequential diagram - Delete book category

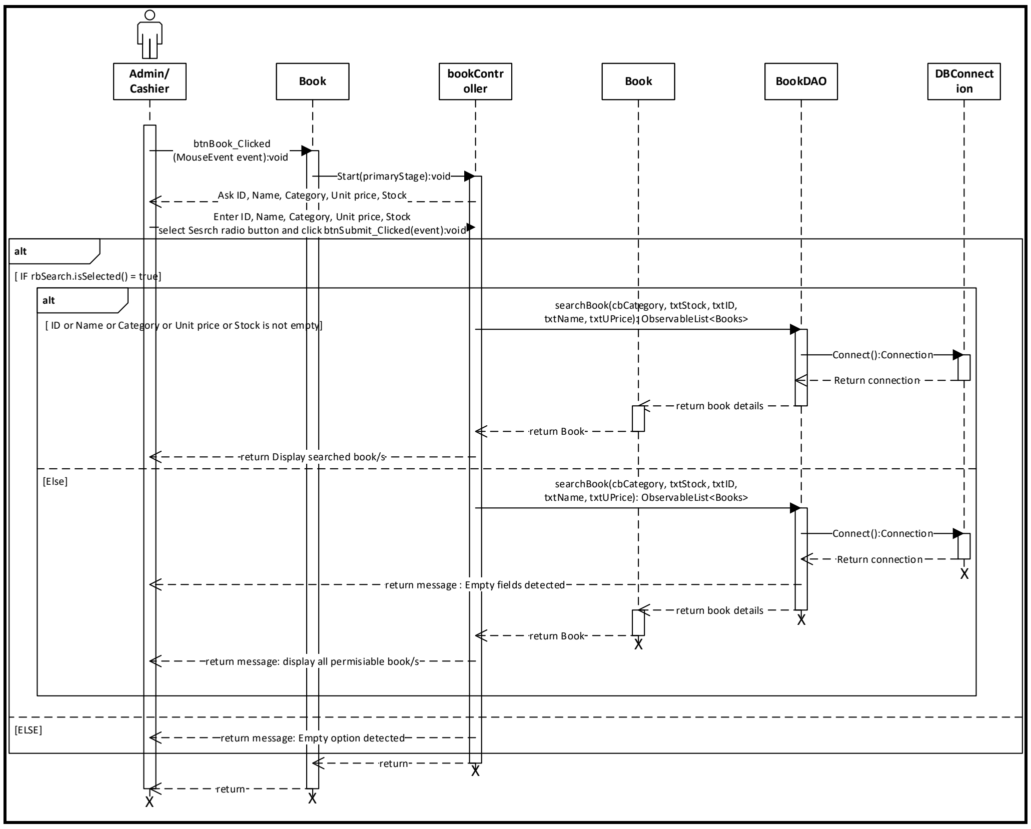
Assumptions:

- Database is created
- book, book category, employee, login and role tables are created
- Admin have an account
- Admin logged in correct username and password
- Some book categorie's details entered in the system

The above sequence diagram depicts about deleting an existing book category from the system. Admin can only perform this task. In first admin need to fill all filed in correct manner. For an example, book ID field cannot contain character values in that case it will throw error message. And also, if any fields seemed to be empty or delete book category option is not selected that time also it will throw error message. All the validating processes are held in BookValidation class. This help to ensure the data accuracy. Also, it will ask confirmation to the admin for a second verification. It reduces the accidental data loss. All the database query operations are done within BookDAO class.

3.3.8. Search book

Double click the picture for view in Microsoft Viso for clear visuals



Admin/ Cashier
Book
bookController
Book
BookDAO
DBConnection
btnBook_Clicked (MouseEvent event):void
Start(primaryStage):void
Ask ID, Name, Category, Unit price, Stock
Enter ID, Name, Category, Unit price, Stock
select Sesrch radio button and click btnSubmit_Clicked(event):void
alt
[IF rbSearch.isSelected() = true]
alt
[ID or Name or Category or Unit price or Stock is not empty]
searchBook(cbCategory, txtStock, txtID, txtName, txtUPrice): ObservableList<Books>
Connect():Connection
Return connection
return book details
return Book
return Display searched book/s
[Else]
searchBook(cbCategory, txtStock, txtID, txtName, txtUPrice): ObservableList<Books>
Connect():Connection
Return connection
return message : Empty fields detected
return book details
return Book
return message: display all permisiable book/s
[ELSE]
return message: Empty option detected
return
return

Figure 10 Sequential diagram - search book

Assumptions:

- Database is created
- book, book category, employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier logged in correct username and password
- Some book’s details entered in the system

The above sequence diagram depicts about searching an existing book in the system. Admin or cashier can perform this task. In first they need to fill all filed in correct manner. For an example, book ID field cannot contain character values in that case it will throw error message. Also, if search option is not selected then it will throw error message. But if any fields seemed to be empty that time it will show all book details. When everything is seemed to be correct then it will show searched user book details in the table. All the database query operations are done within BookDAO class.

3.3.9. Update book

Double click the picture for view in Microsoft Viso for clear visuals

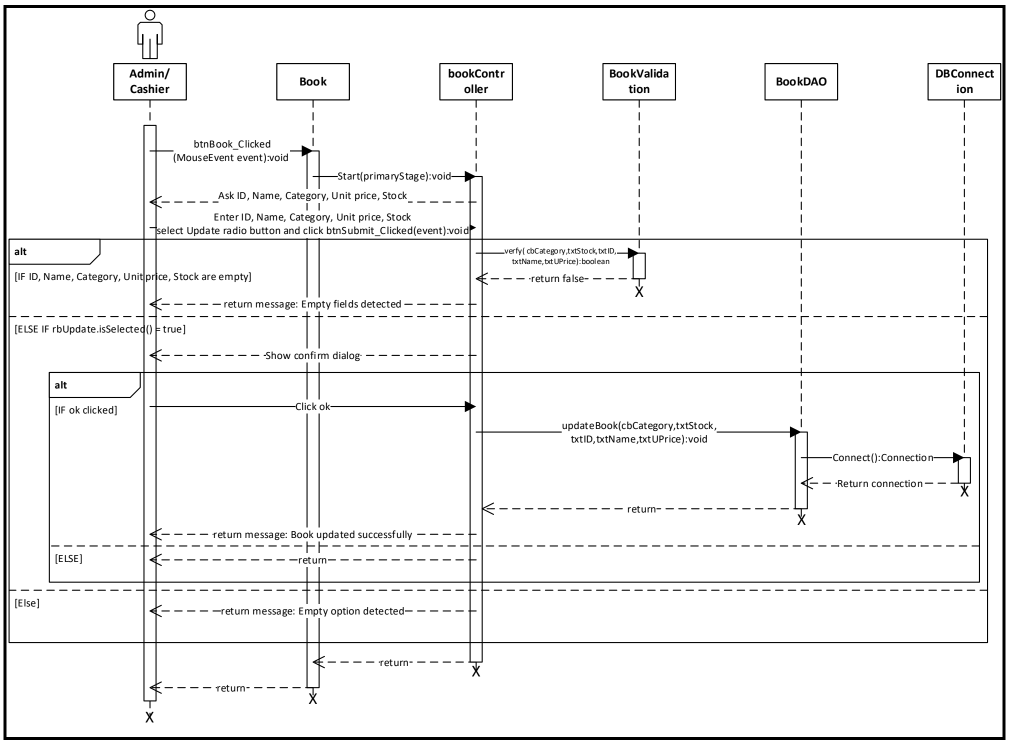

Admin/ Cashier
Book
bookController
BookValidation
BookDAO
DBConnection
btnBook_Clicked (MouseEvent event):void
Start(primaryStage):void
Ask ID, Name, Category, Unit price, Stock
Enter ID, Name, Category, Unit price, Stock select Update radio button and click btnSubmit_Clicked(event):void
alt
[IF ID, Name, Category, Unit price, Stock are empty]
verfy(cbCategory,txtStock,txtID, txtName,txtUPrice):boolean
return false
return message: Empty fields detected
[ELSE IF rbUpdate.isSelected() = true]
Show confirm dialog
alt
[IF ok clicked]
Click ok
updateBook(cbCategory,txtStock, txtID,txtName,txtUPrice):void
Connect():Connection
Return connection
return
return message: Book updated successfully
[ELSE]
return
[Else]
return message: Empty option detected
return
return

Figure 11 Sequence diagram - Update book

Assumptions:

- Database is created
- book, book category, employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier logged in correct username and password
- Some book’s details entered in the system

The above sequence diagram depicts about updating an existing book in the system. Admin or cahier can perform this task. In first admin or cashier need to fill all filed in correct manner. For an example, book ID field cannot contain character values in that case it will throw error message. And also, if any fields seemed to be empty or update option is not selected that time also it will throw error message. All the validating processes are held in BookValidation class. This help to ensure the data accuracy. Also, it will ask confirmation to the admin or cashier for a second verification. It reduces the accidental data loss. All the database query operations are done within BookDAO class.

3.3.10. View all book category

Double click the picture for view in Microsoft Viso for clear visuals

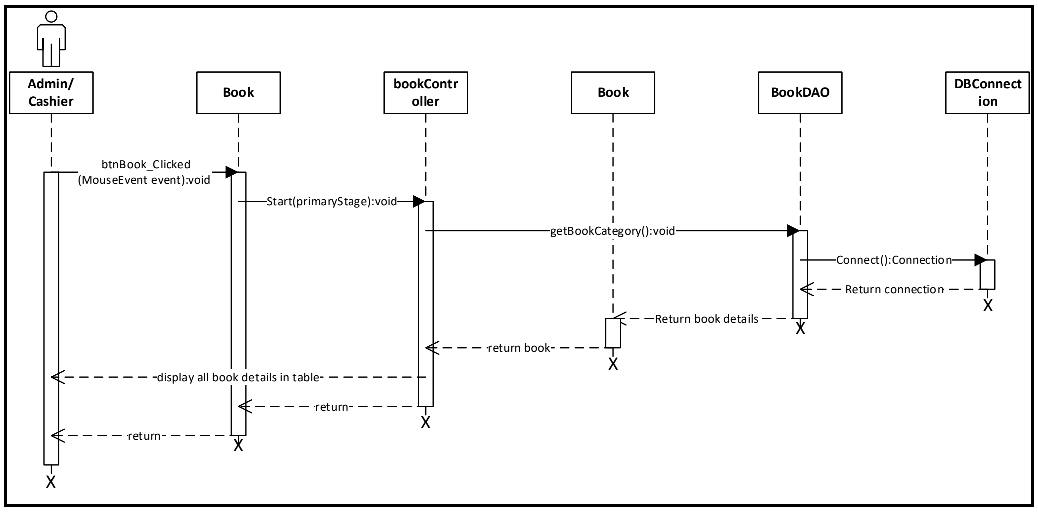


Figure 12 Sequence diagram - view all book details

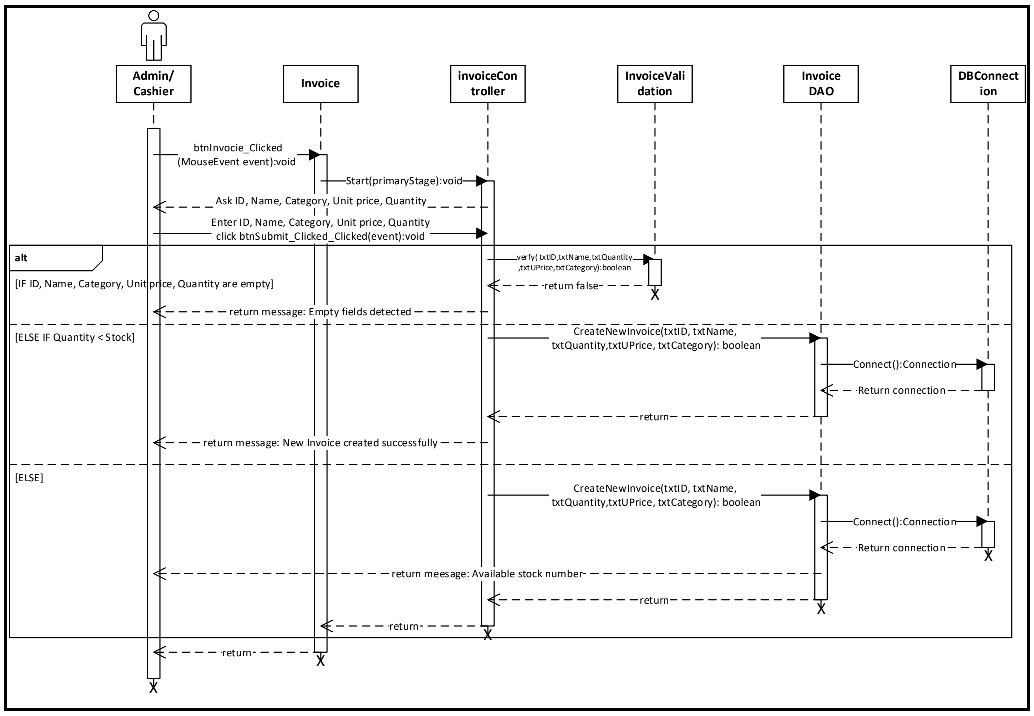
Assumptions:

- Database is created
- book, book category, employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier logged in correct username and password
- Some book's details entered in the system

The above sequence diagram depicts about view all book details in the system. This can be done by cashier or admin. There is no validation process for this task. All available book details will be displayed in the table. All the database query operations are done within BookDAO class.

3.3.11. Create invoice

Double click the picture for view in Microsoft Viso for clear visuals



Admin/ Cashier
Invoice
invoiceCon troller
InvoiceVali dation
Invoice DAO
DBConnect ion
btnInvocie_Clicked (MouseEvent event):void
Start(primaryStage):void
Ask ID, Name, Category, Unit price, Quantity
Enter ID, Name, Category, Unit price, Quantity click btnSubmit_Clicked_Clicked(event):void
alt
verfy(txtID,txtName,txtQuantity ,txtUPrice,txtCategory):boolean
[IF ID, Name, Category, Unit price, Quantity are empty]
return false
return message: Empty fields detected
[ELSE IF Quantity < Stock]
CreateNewInvoice(txtID, txtName, txtQuantity,txtUPrice, txtCategory): boolean
Connect():Connection
Return connection
return
return message: New Invoice created successfully
[ELSE]
CreateNewInvoice(txtID, txtName, txtQuantity,txtUPrice, txtCategory): boolean
Connect():Connection
Return connection
return meesage: Available stock number
return
return
return

Figure 13 Sequence diagram - Create invoice

Assumptions:

- Database is created
- invoice, employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier logged in correct username and password

The above sequence diagram depicts about creating a new invoice in the system. Admin or cashier can perform this task. In first they need to fill all fields in correct manner. For an example, invoice ID field cannot contain character values in that case it will throw error message. And also, if any fields seemed to be empty that time it will throw error message too. All the validating processes are held in InvoiceValidation class. This help to ensure the data accuracy. All the database query operations are done within InvoiceDAO class.

3.3.12. Print invoice

Double click the picture for view in Microsoft Viso for clear visuals

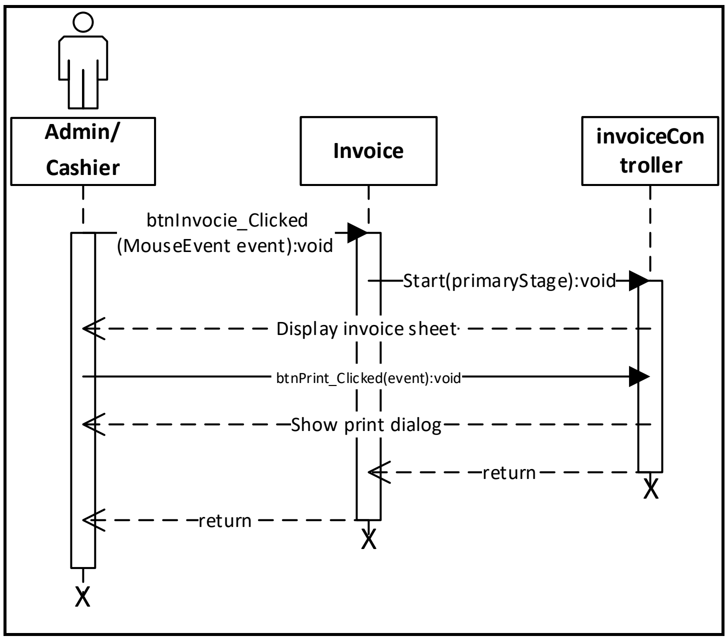


Figure 14 Sequence diagram - Print invoice

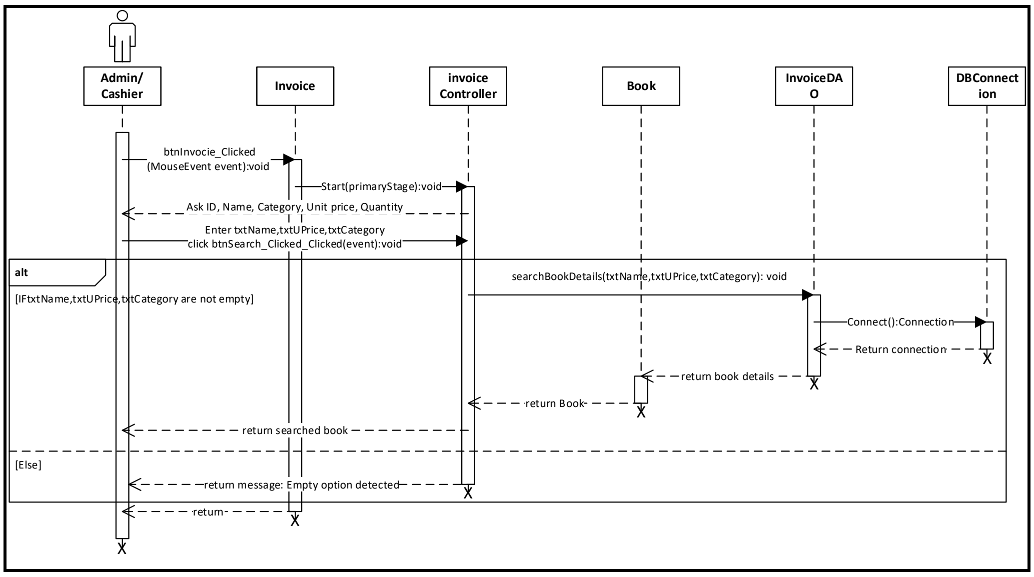
Assumptions:

- Database is created
- invoice, employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier logged in correct username and password
- Printer is configured within the system
- Invoice is created successfully

The above sequence diagram depicts about printing the invoice details by printer via using the system. This can be done by cashier or admin. There is no validation process for this task. All invoice details will display in the window. All the database query operations are done within InvoiceDAO class. In success scenario it will show print dialog and when we click ok then it sent the file to printer que. From printer que it will print the invoice.

3.3.13. Search book

Double click the picture for view in Microsoft Viso for clear visuals



Admin/ Cashier
Invoice
invoice Controller
Book
InvoiceDAO
DBConnection
btnInvocie_Clicked (MouseEvent event):void
Start(primaryStage):void
Ask ID, Name, Category, Unit price, Quantity
Enter txtName,txtUPrice,txtCategory click btnSearch_Clicked_Clicked(event):void
alt
[IFtxtName,txtUPrice,txtCategory are not empty]
searchBookDetails(txtName,txtUPrice,txtCategory): void
Connect():Connection
Return connection
return book details
return Book
return searched book
[Else]
return message: Empty option detected
return

Figure 15 Sequence diagram - Search book

Assumptions:

- Database is created
- Invoice, book, book category, employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier logged in correct username and password
- Some book’s details entered in the system

The above sequence diagram depicts about searching an existing book in the system for create an invoice. Admin or cashier can perform this task. In first they need to fill all filed in correct manner. For an example, if entered wrong book name in that case it will throw error message. also, if any fields seemed to be empty that time it will throw error message. When everything is seemed to be correct then it will show searched user book details in the table. All the database query operations are done within InvocieDAO class.

3.3.14. Book inventory

Double click the picture for view in Microsoft Viso for clear visuals

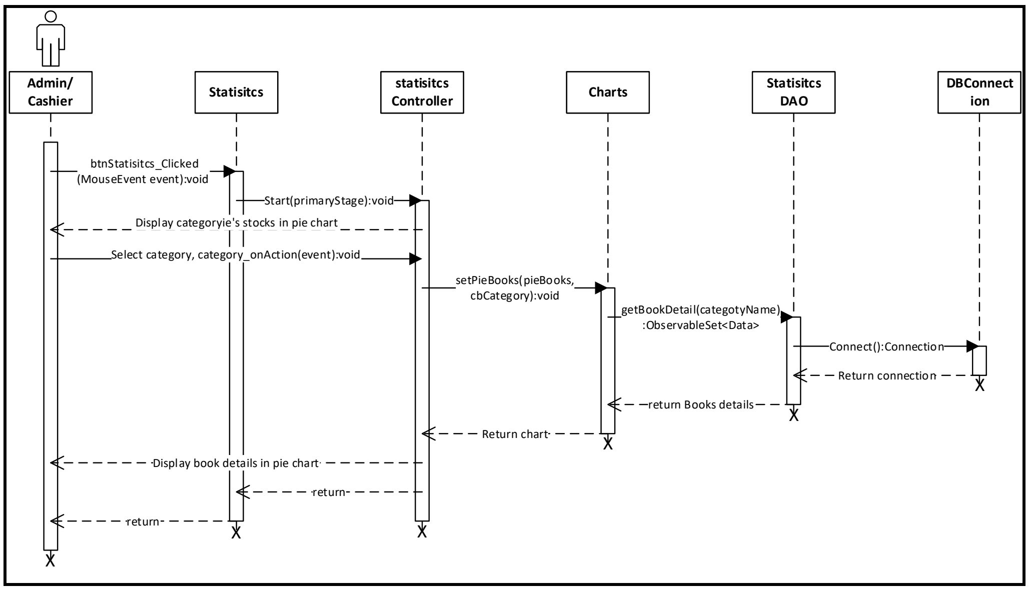

Admin/ Cashier
Statisitcs
statisitcs Controller
Charts
Statisitcs DAO
DBConnect ion
btnStatisitcs_Clicked (MouseEvent event):void
Start(primaryStage):void
Display categoryie's stocks in pie chart
Select category, category_onAction(event):void
setPieBooks(pieBooks, cbCategory):void
getBookDetail(categotyName) :ObservableSet<Data>
Connect():Connection
Return connection
return Books details
Return chart
Display book details in pie chart
return
return

Figure 16 Sequence diagram - Book inventory

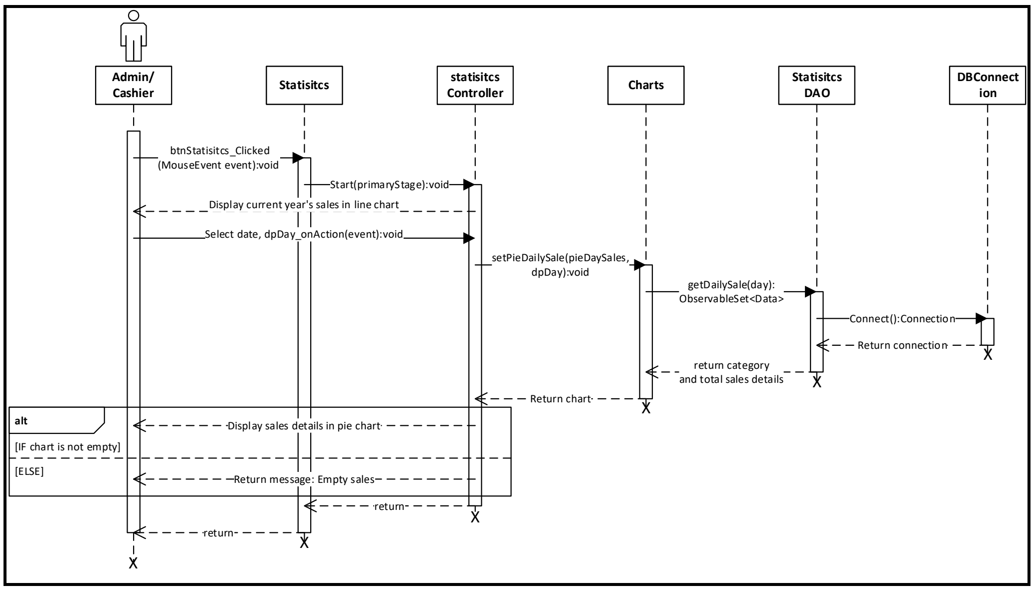
Assumptions:

- Database is created
- Book, book category, employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier logged in correct username and password
- Some book's details entered in the system

The above sequence diagram depicts about view book inventory details. Admin or cashier can perform this task. In the statistics page, initially all book categories and total number of books inside each category are will display in a pie chart. But admin or cashier can select a specific category for view books of particular book category. All the database query operations are done within StatisticsDAO class.

3.3.15. Daily sales

Double click the picture for view in Microsoft Viso for clear visuals



Admin/ Cashier
Statisitcs
statisitcs Controller
Charts
Statisitcs DAO
DBConnect ion
btnStatisitcs_Clicked (MouseEvent event):void
Start(primaryStage):void
Display current year's sales in line chart
Select date, dpDay_onAction(event):void
setPieDailySale(pieDaySales, dpDay):void
getDailySale(day): ObservableSet<Data>
Connect():Connection
Return connection
return category and total sales details
Return chart
alt
[IF chart is not empty]
Display sales details in pie chart
[ELSE]
Return message: Empty sales
return
return

Figure 17 Sequence diagram - Daily sales

Assumptions:

- Database is created
- Invoice, book, book category, employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier logged in correct username and password
- Some book's details entered in the system
- Some invoices are made by admin or cashier

The above sequence diagram depicts about view daily details. Admin or cashier can perform this task. In the statistics page, initially all book categories and sum of sold book's price inside each category are will display in a pie chart. But admin or cashier can select a specific day for view sold books and sum of total payment of particular day. All the database query operations are done within StatisticsDAO class.

3.3.16. Monthly sales

Double click the picture for view in Microsoft Viso for clear visuals

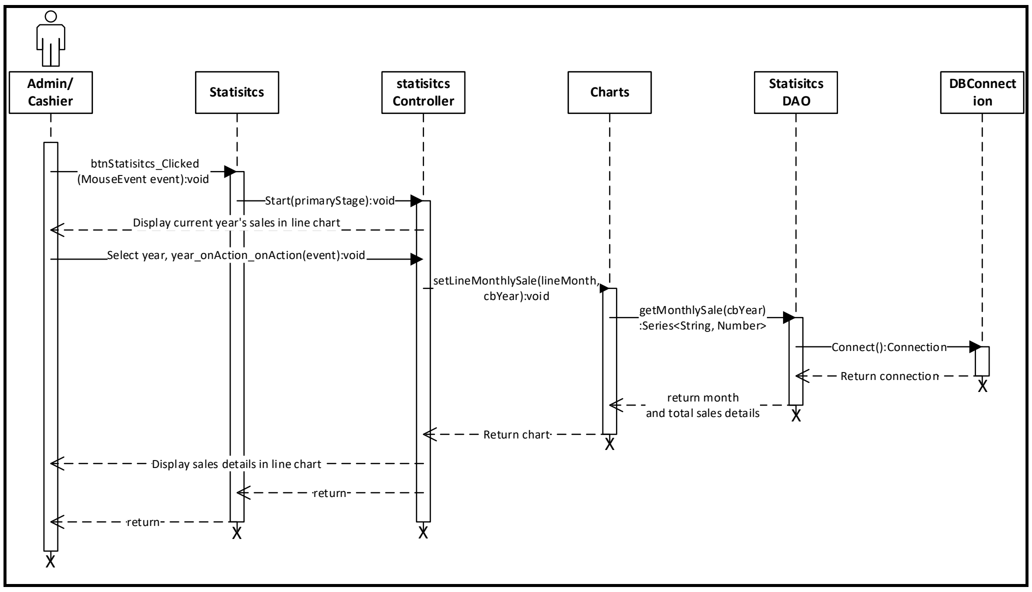

Admin/ Cashier
Statisitcs
statisitcs Controller
Charts
Statisitcs DAO
DBConnect ion
btnStatisitcs_Clicked (MouseEvent event):void
Start(primaryStage):void
Display current year's sales in line chart
Select year, year_onAction_onAction(event):void
setLineMonthlySale(lineMonth, cbYear):void
getMonthlySale(cbYear) :Series<String, Number>
Connect():Connection
Return connection
return month and total sales details
Return chart
Display sales details in line chart
return
return

Figure 18 Sequence diagram - Monthly sales

Assumptions:

- Database is created
- Invoice, book, book category, employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier logged in correct username and password
- Some book’s details entered in the system
- Some invoices are made by admin or cashier

The above sequence diagram depicts about view monthly details. Admin or cashier can perform this task. In the statistics page, initially sum of sold book’s price in each month of current year will display in a line chart. But admin or cashier can select a specific year for view details of particular year. All the database query operations are done within StatisticsDAO class.

3.3.17. Delete invoice

Double click the picture for view in Microsoft Viso for clear visuals

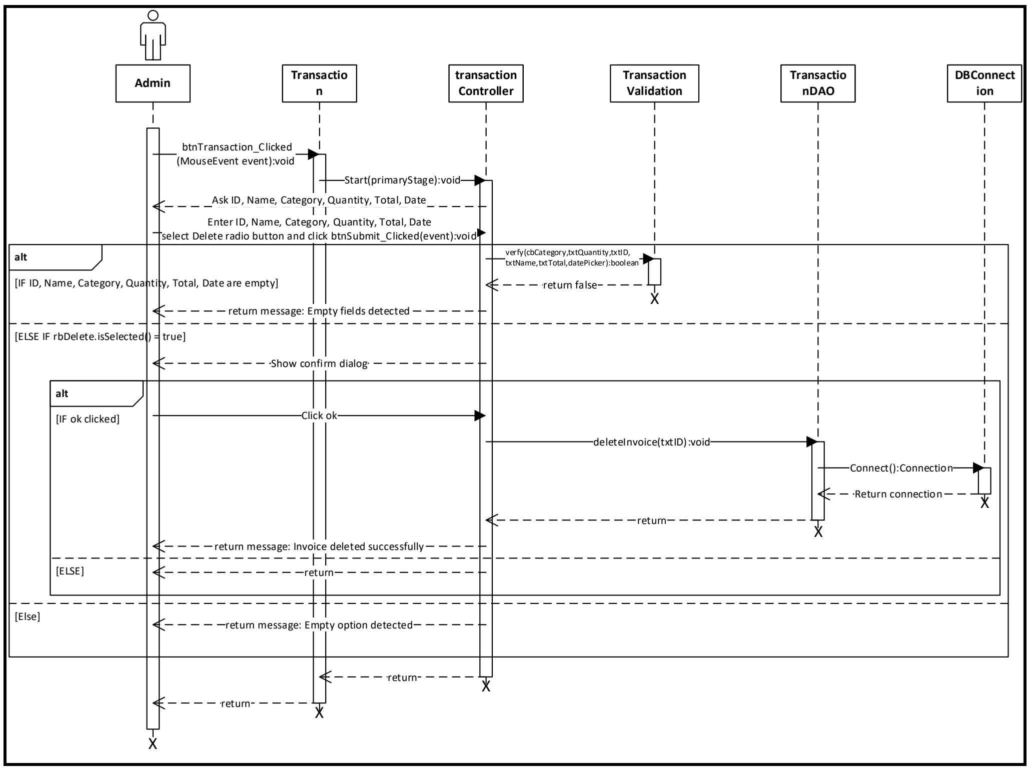

Admin
Transaction
transaction Controller
Transaction Validation
TransactionDAO
DBConnection
btnTransaction_Clicked (MouseEvent event):void
Start(primaryStage):void
Ask ID, Name, Category, Quantity, Total, Date
Enter ID, Name, Category, Quantity, Total, Date
select Delete radio button and click btnSubmit_Clicked(event):void
alt
verfy(cbCategory,txtQuantity,txtID, txtName,txtTotal,datePicker):boolean
[IF ID, Name, Category, Quantity, Total, Date are empty]
return false
return message: Empty fields detected
[ELSE IF rbDelete.isSelected() = true]
Show confirm dialog
alt
[IF ok clicked]
Click ok
deleteInvoice(txtID):void
Connect():Connection
Return connection
return
return message: Invoice deleted successfully
[ELSE]
return
[Else]
return message: Empty option detected
return
return

Figure 19 Sequence diagram - Delete invoice

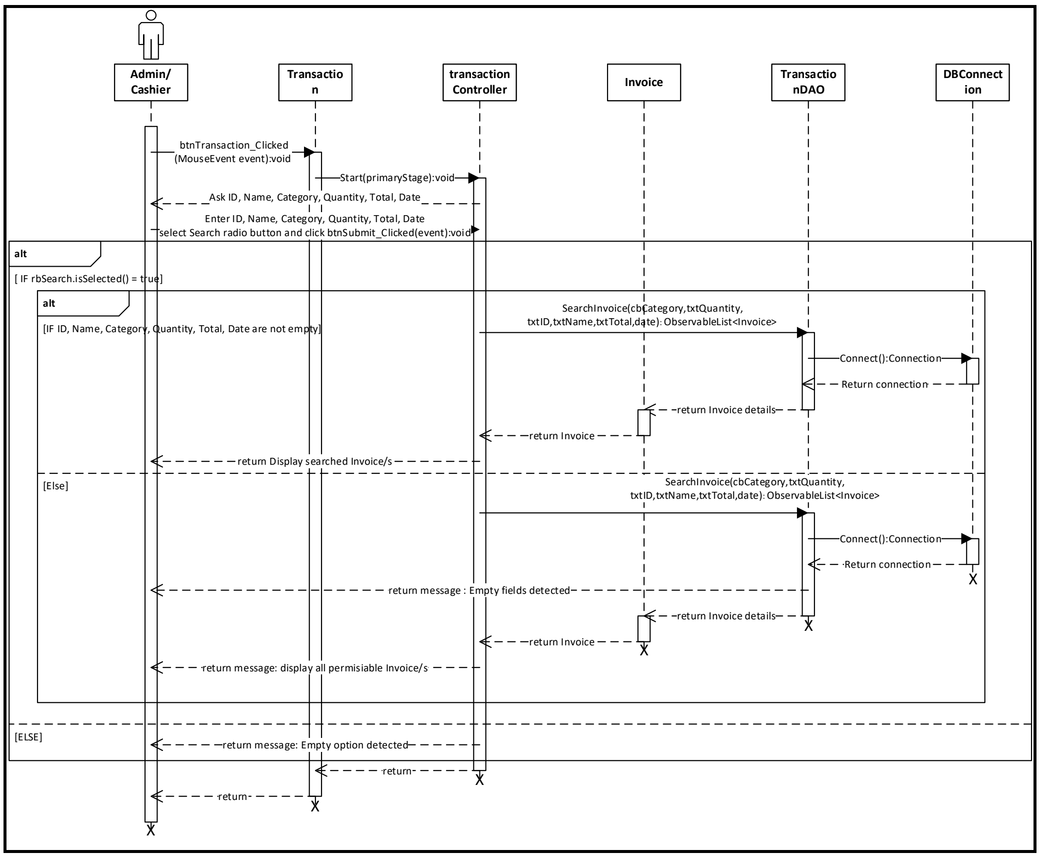
Assumptions:

- Database is created
- Invoice, employee, login and role tables are created
- Admin have an account
- Admin logged in correct username and password
- Some invoice details entered in the system

The above sequence diagram depicts about deleting an existing invocie from the system. Admin can only perform this task. In first admin need to fill all filed in correct manner. For an example, invcoie ID field cannot contain character values in that case it will throw error message. And also, if any fields seemed to be empty or delete option is not selected that time also it will throw error message. All the validating processes are held in InvoiceValidation class. This help to ensure the data accuracy. Also, it will ask confirmation to the admin for a second verification. It reduces the accidental data loss. All the database query operations are done within InvoiceDAO class.

3.3.18. Search invoice

Double click the picture for view in Microsoft Viso for clear visuals



Admin/ Cashier
Transactio n
transaction Controller
Invoice
Transactio nDAO
DBConnect ion
btnTransaction_Clicked (MouseEvent event):void
Start(primaryStage):void
Ask ID, Name, Category, Quantity, Total, Date
Enter ID, Name, Category, Quantity, Total, Date
select Search radio button and click btnSubmit_Clicked(event):void
alt
[IF rbSearch.isSelected() = true]
alt
[IF ID, Name, Category, Quantity, Total, Date are not empty]
SearchInvoice(cbCategory,txtQuantity, txtID,txtName,txtTotal,date): ObservableList<Invoice>
Connect():Connection
Return connection
return Invoice details
return Invoice
return Display searched Invoice/s
[Else]
SearchInvoice(cbCategory,txtQuantity, txtID,txtName,txtTotal,date): ObservableList<Invoice>
Connect():Connection
Return connection
return message : Empty fields detected
return Invoice details
return Invoice
return message: display all permisiable Invoice/s
[ELSE]
return message: Empty option detected
return
return

Figure 20 Sequence diagram - Search invoice

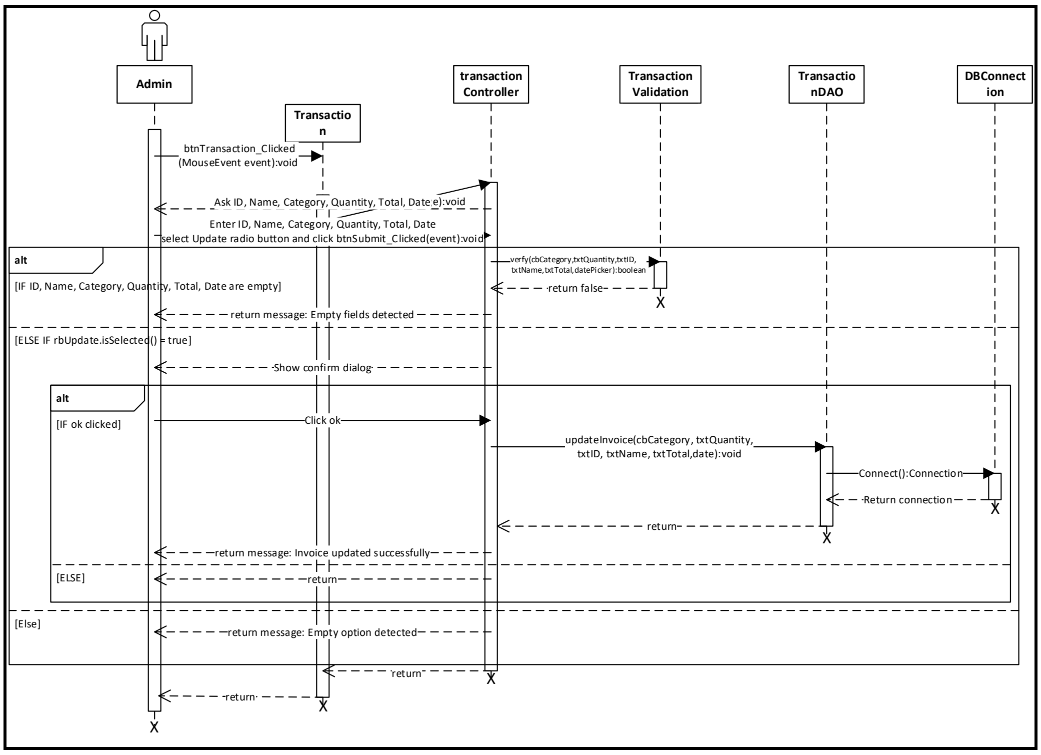
Assumptions:

- Database is created
- Invoice, employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier logged in correct username and password
- Some invoice details entered in the system

The above sequence diagram depicts about searching an existing invoice in the system for create an invoice. Admin or cashier can perform this task. In first they need to fill all filed in correct manner. For an example, if entered wrong book invoice ID in that case it will throw error message. also, if any fields seemed to be empty or search option is not selected that time it will throw error message. When everything is seemed to be correct then it will show searched invoice details in the table. All the database query operations are done within TransactionDAO class.

3.3.19. Update invoice

Double click the picture for view in Microsoft Viso for clear visuals



Admin
Transactio n
transaction Controller
Transaction Validation
Transactio nDAO
DBConnect ion
btnTransaction_Clicked (MouseEvent event):void
Ask ID, Name, Category, Quantity, Total, Date e):void
Enter ID, Name, Category, Quantity, Total, Date select Update radio button and click btnSubmit_Clicked(event):void
alt
[IF ID, Name, Category, Quantity, Total, Date are empty]
verfy(cbCategory,txtQuantity,txtID, txtName,txtTotal,datePicker):boolean
return false
return message: Empty fields detected
[ELSE IF rbUpdate.isSelected() = true]
Show confirm dialog
alt
[IF ok clicked]
Click ok
updateInvoice(cbCategory, txtQuantity, txtID, txtName, txtTotal,date):void
Connect():Connection
Return connection
return
return message: Invoice updated successfully
[ELSE]
return
[Else]
return message: Empty option detected
return
return

Figure 21 Sequence diagram - Update invoice

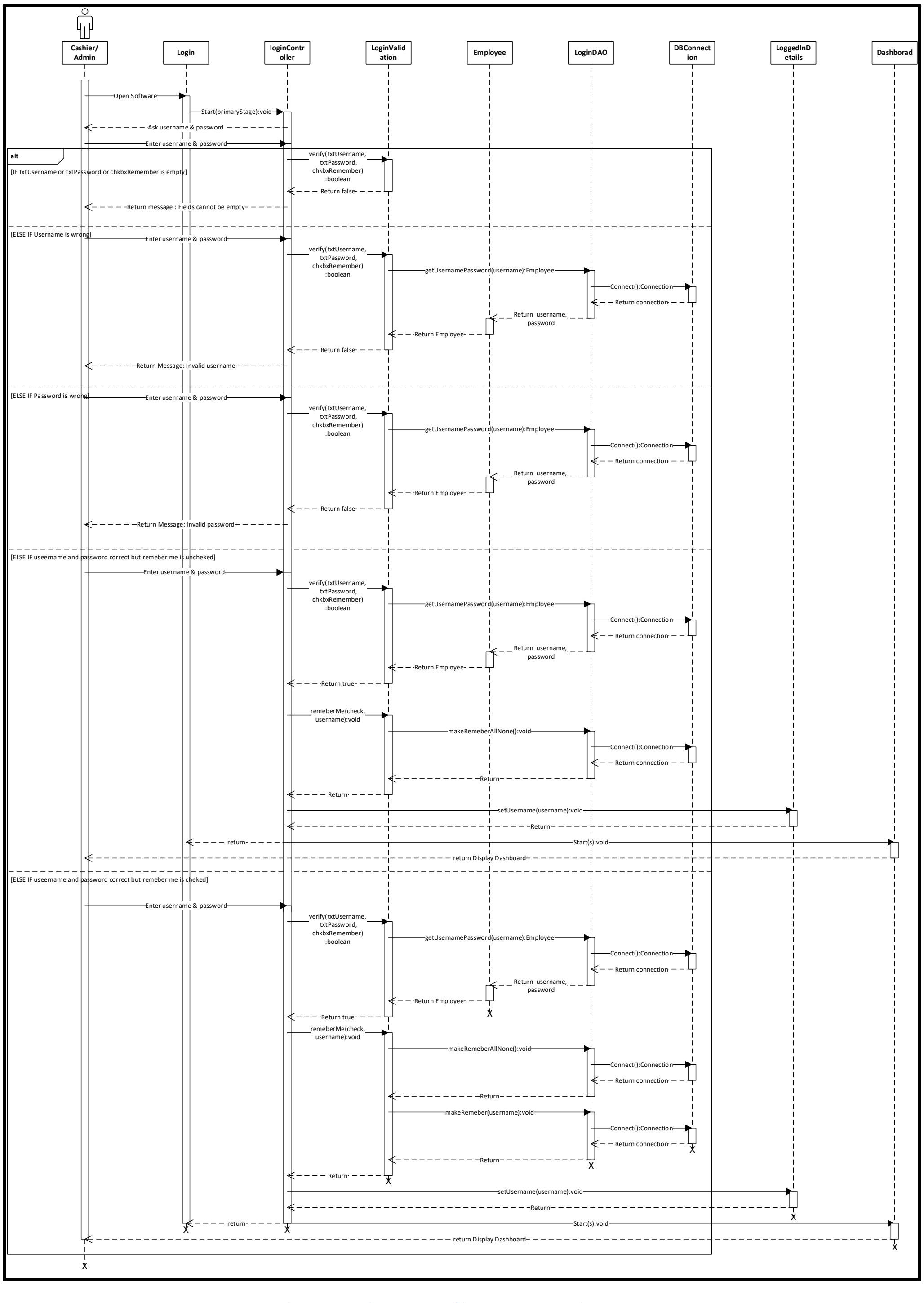
Assumptions:

- Database is created
- Inventory, book, book category, employee, login and role tables are created
- Admin have an account
- Admin logged in correct username and password
- Some invoice details entered in the system

The above sequence diagram depicts about updating an existing invocie in the system. Admin can only perform this task. In first admin or cashier need to fill all filed in correct manner. For an example, Invoice ID field cannot contain character values in that case it will throw error message. And also, if any fields seemed to be empty or update option is not selected that time also it will throw error message. All the validating processes are held in TransactionValidation class. This help to ensure the data accuracy. Also, it will ask confirmation to the admin for a second verification. It reduces the accidental data loss. All the database query operations are done within TransactionDAO class.

3.3.20. Login

Double click the picture for view in Microsoft Viso for clear visuals



Cashier/ Admin
Login
loginController
LoginValidation
Employee
LoginDAO
DBConnection
LoggedInDetails
Dashborad
Open Software
Start(primaryStage):void
Ask username & password
Enter username & password
alt
[IF txtUsername or txtPassword or chkbxRemember is empty]
verify(txtUsername, txtPassword, chkbxRemember) :boolean
Return false
Return message : Fields cannot be empty
[ELSE IF Username is wrong]
getUsernamePassword(username):Employee
Connect():Connection
Return connection
Return username, password
Return Employee
Return Message : Invalid username
[ELSE IF Password is wrong]
Return Message : Invalid password
[ELSE IF useername and password correct but remeber me is uncheked]
Return true
remeberMe(check, username):void
makeRemeberAllNone():void
Return
setUsername(username):void
return
Start(s):void
return Display Dashboard
[ELSE IF useername and password correct but remeber me is cheked]
makeRemeber(username):void

Figure 22 Sequence diagram – Login

Assumptions:

- Database is created
- employee, login and role tables are created
- Admin or cashier have an account

The above sequence diagram depicts about login into the system. Admin or cashier can perform this task. When user name is not identified in the database or entered password is not matched with database or any empty fields found then it will throw an error message. All validations are done LoginValidation class. All the database query operations are done within LoginDAO class.

3.3.21. Logout

Double click the picture for view in Microsoft Viso for clear visuals

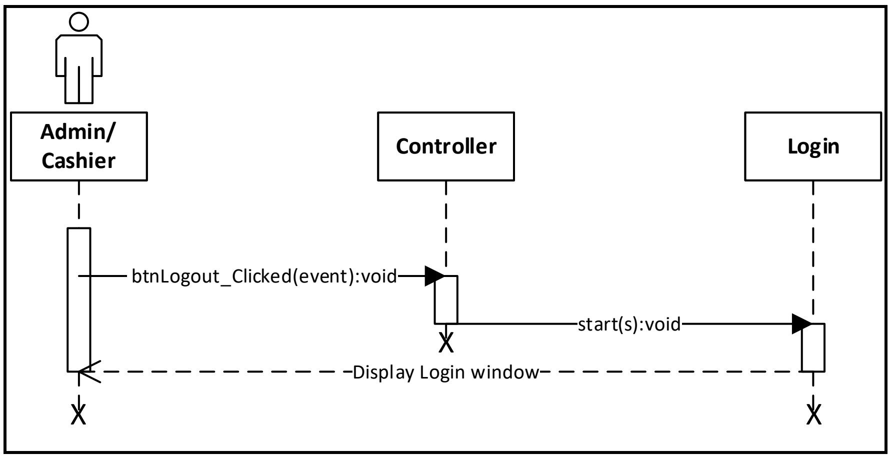


Figure 23 Sequence diagram – logout

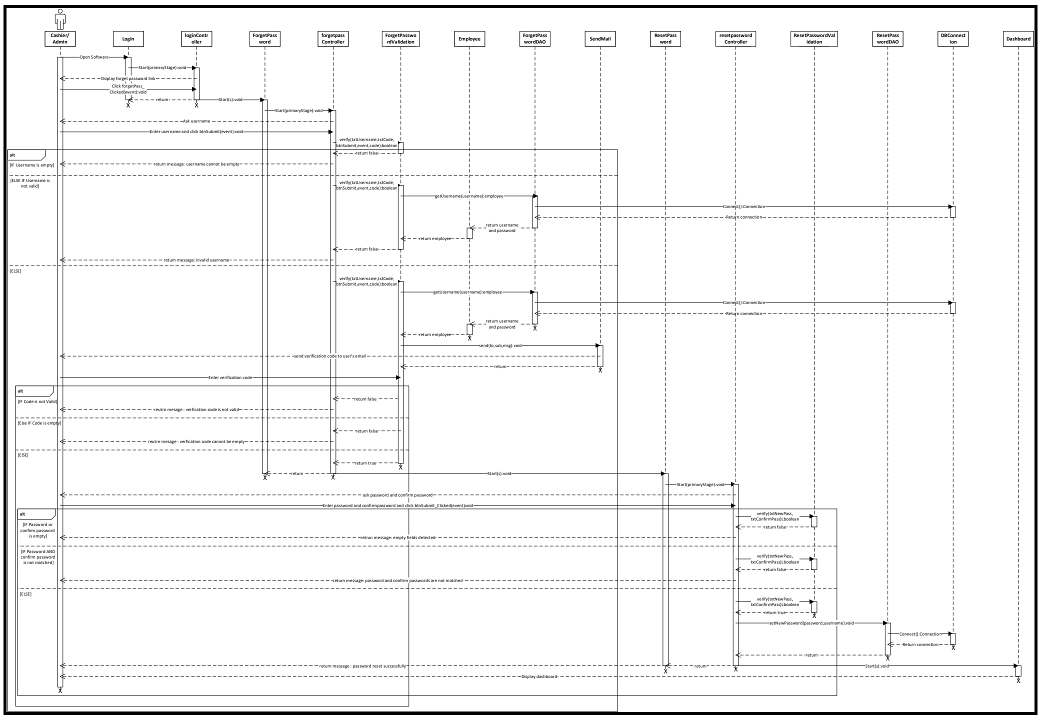
Assumptions:

- Database is created
- employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier login with correct username and password

The above sequence diagram depicts about logout from the system. Admin or cashier can perform this task. This operation can perform in dashboard unit or book management unit or invoice management unit or transaction unit or account management unit or statistics unit. When admin or cashier click the logout button then it will redirect them into login page.

3.3.22. Reset password

Double click the picture for view in Microsoft Viso for clear visuals



Assumptions:

- Database is created
- employee, login and role tables are created
- Admin or cashier have an account
- Admin or cashier know username but forget their password
- Admin and cashier have email address

The above sequence diagram depicts about resetting admin or cashier password. Admin or cashier can perform this task. This process travel through three units. First, it’s starts from login unit. When user forget their password then they will click forget password link in the login page. That link direct them into forget password unit. In this unit, user will be verified for ensure the security. For that user need to enter user name in the user field then it will be matched with database for verify the user. Then a verification code will be sent to user’s email address. Then user need to enter the code into the system same as they got via email. If any of them not matched then error message will be popup. When everything is seemed to be success then it will direct the user into the reset password unit. In the reset password unit, it will ask for new password but for twice to reduce mistyping. In forget password unit all verification done by within ForegetPasswordValidation class and database query processes are done by within ForgetPasswordDAO. In reset password unit all verification done by within ResetPasswordValidation class and database query processes are done by within ResetPasswordDAO.

4. Development

The actual product development process begins in the development stage. Here, according to the Design Document Specification (DDS), computer code is created. It is possible to generate code quickly and easily if the design is comprehensive and well-organized.

(SDLC - Overview, 2021)

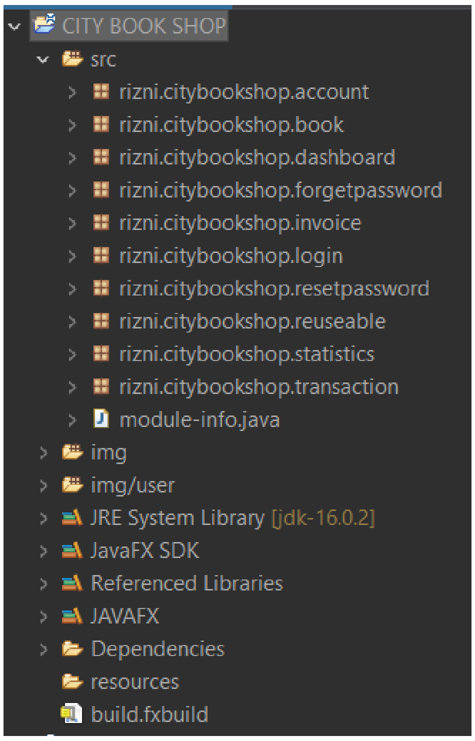
4.1. Development environment

The following table categorizes the hardware, software and libraries combinations that were utilized during system development.

Hardware	Software	Libraries
System Model - Aspire A515-54G	Microsoft Windows 10 Home Single Language	Activation V1.1
RAM 8.00 GB	Eclipse Version: 2021-06 (4.20.0) Build id: 20210612-2011	fontawesomefx-commons V9.1.2
1TB HDD + 256 SSD	Java V16.0.2	fontawesomefx-fontawesome V4.7.0-9.1.2
Nvidia MX-250 2GB	JavaFX V16	Javafx Mail V1.6.2
	Scene builder	Jfoenix V9.0.4
		mysql-connector-java V8.0.22
		JRE System library
		JAVAFX
		JavaFX SDK

Table 6 Hardware, software and libraries

4.2. Module structure of the system

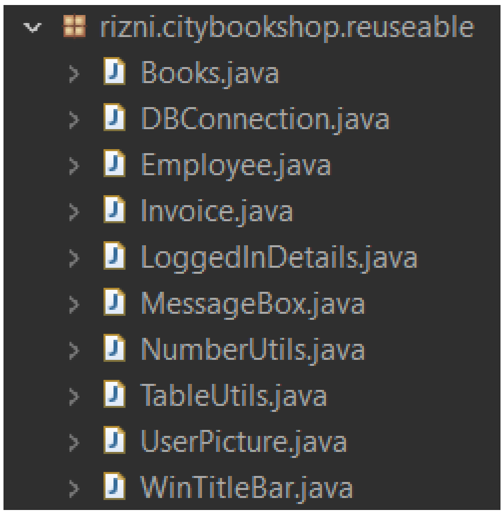


```
CITY BOOK SHOP
  src
    rizni.citybookshop.account
    rizni.citybookshop.book
    rizni.citybookshop.dashboard
    rizni.citybookshop.forgetpassword
    rizni.citybookshop.invoice
    rizni.citybookshop.login
    rizni.citybookshop.resetpassword
    rizni.citybookshop.reuseable
    rizni.citybookshop.statistics
    rizni.citybookshop.transaction
    module-info.java
  img
  img/user
  JRE System Library [jdk-16.0.2]
  JavaFX SDK
  Referenced Libraries
  JAVAFX
  Dependencies
  resources
  build.fxbuild
```

Figure 24 Module structure of the system

4.3. Reusable components

Reusable components are created inside the Rizni.citybookshop.reuseable package for reduce code repetition and enhance the system performance. Below figure is illustrate the reusable components.



```
rizni.citybookshop.reuseable
  Books.java
  DBConnection.java
  Employee.java
  Invoice.java
  LoggedInDetails.java
  MessageBox.java
  NumberUtils.java
  TableUtils.java
  UserPicture.java
  WinTitleBar.java
```

Figure 25 Reusable components

4.4. Object oriented programming

In this context, an object is a real-world item such as a pen or a chair or a table or a computer or a watch. In Object-Oriented Programming, classes and objects are used to create a program. Software development and maintenance are simplified by the following concepts: objects, classes, inheritance, encapsulation, polymorphism and abstraction.

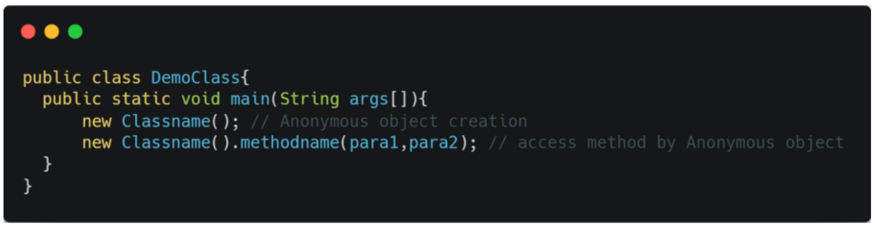
4.4.1. Object

Objects are instances of classes created with specific data. An object occupies some space in heap memory until its life time is over. Object can be two categories

1. Anonymous object

Anonymous objects don't have any variable name to call it again. It doesn't have stack memory allocation; it uses only heap memory for the allocation.

General syntax:



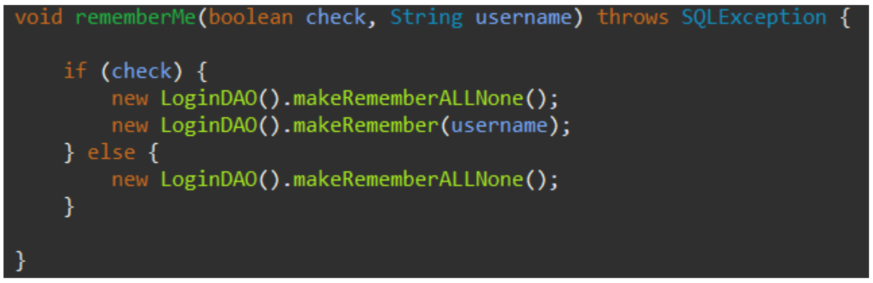
```
public class DemoClass{
  public static void main(String args[]){
      new Classname(); // Anonymous object creation
      new Classname().methodname(para1,para2); // access method by Anonymous object
  }
}
```

Figure 26 Anonymous object – general syntax

Used in the system:

Almost 90% of objects are created as anonymous object for utilize memory management. Because when particular task is finished then that object eligible for garbage collection (GC)

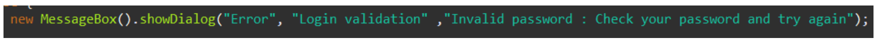
Example 1: there is a method rememberMe developed for remember the user's username and password when they tick the remember me check box in the login page. It designed for increase the user experience. In that method to access database methods it is created as anonymous object. See figure 27 below



```java
void rememberMe(boolean check, String username) throws SQLException {

    if (check) {
        new LoginDAO().makeRememberALLNone();
        new LoginDAO().makeRemember(username);
    } else {
        new LoginDAO().makeRememberALLNone();
    }

}
```

Figure 27 Anonymous object - example 1

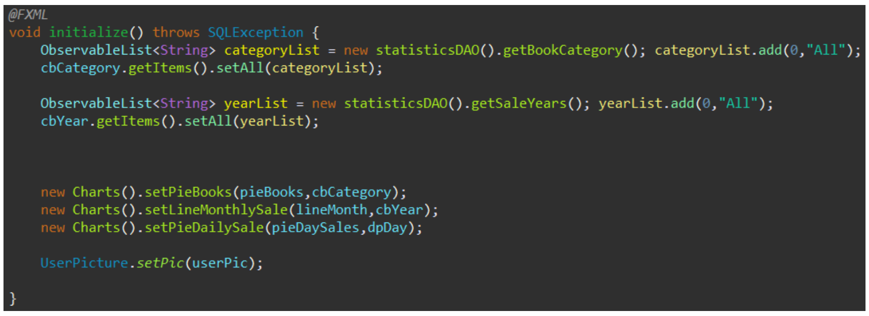
Example 2: For showing the messages to the user, there is a class created for messageBox and that contain a method called showDialog. In the entire system to call that showDialog method for displaying a message; An anonymous object creation is used. See figure 28.



```java
new MessageBox().showDialog("Error", "Login validation" ,"Invalid password : Check your password and try again");
```

Figure 28 Anonymous object - example 2

Example 3: In statisticsController, for set data for charts anonymous object is created from Chart class inside initialize() method.



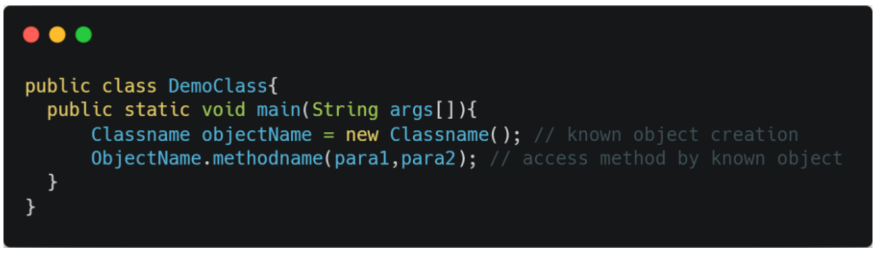
```java
@FXML
void initialize() throws SQLException {
    ObservableList<String> categoryList = new statisticsDAO().getBookCategory(); categoryList.add(0,"All");
    cbCategory.getItems().setAll(categoryList);

    ObservableList<String> yearList = new statisticsDAO().getSaleYears(); yearList.add(0,"All");
    cbYear.getItems().setAll(yearList);



    new Charts().setPieBooks(pieBooks,cbCategory);
    new Charts().setLineMonthlySale(lineMonth,cbYear);
    new Charts().setPieDailySale(pieDaySales,dpDay);

    UserPicture.setPic(userPic);

}
```

Figure 29 Anonymous object - example 3

2. Known object

    Known objects can refer multiple times. It consumes heap memory for the object creation and also it creates stack memory by allocating a variable for it. So, by calling that variable name this kind of objects can recall anytime. Until the system is ending this kind of objects cannot be eligible for garbage collection (GC).

General syntax:



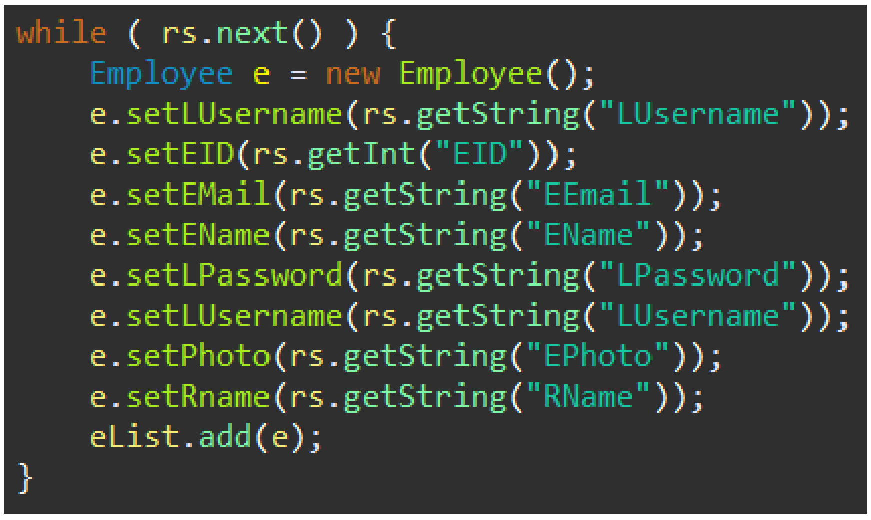
```
public class DemoClass{
  public static void main(String args[]){
      Classname objectName = new Classname(); // known object creation
      ObjectName.methodname(para1,para2); // access method by known object
  }
}
```

Figure 30 Known object - general syntax

Used in the system:

This kind of objects are mostly developed in query process in the database accessing objects. Because this kind of objects is useful for retrieve data from particular entity's tuple and store it into the collection framework.

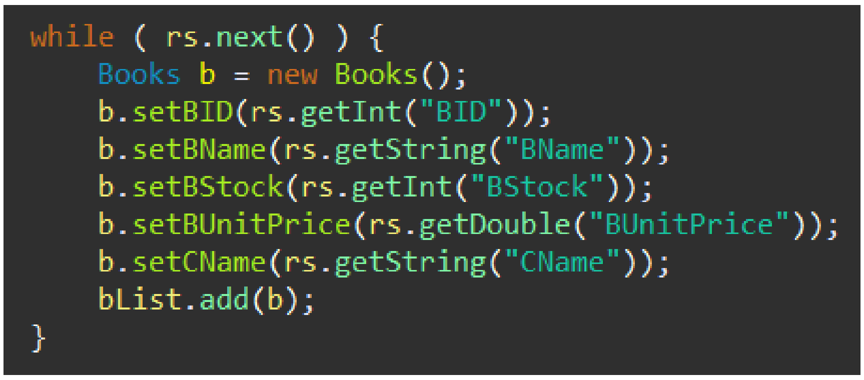
Example 1: Inside the while loop of getEmployeeDetails from AccountDAO Class, there is an object created as known object and assigned that into a name 'e'. then assigned all the retrieved values from database to that object and added to an employee list namely 'eList'.



```
while ( rs.next() ) {
    Employee e = new Employee();
    e.setLUsername(rs.getString("LUsername"));
    e.setEID(rs.getInt("EID"));
    e.setEMail(rs.getString("EEmail"));
    e.setEName(rs.getString("EName"));
    e.setLPassword(rs.getString("LPassword"));
    e.setLUsername(rs.getString("LUsername"));
    e.setPhoto(rs.getString("EPhoto"));
    e.setRname(rs.getString("RName"));
    eList.add(e);
}
```

Figure 31 Known object - example 1

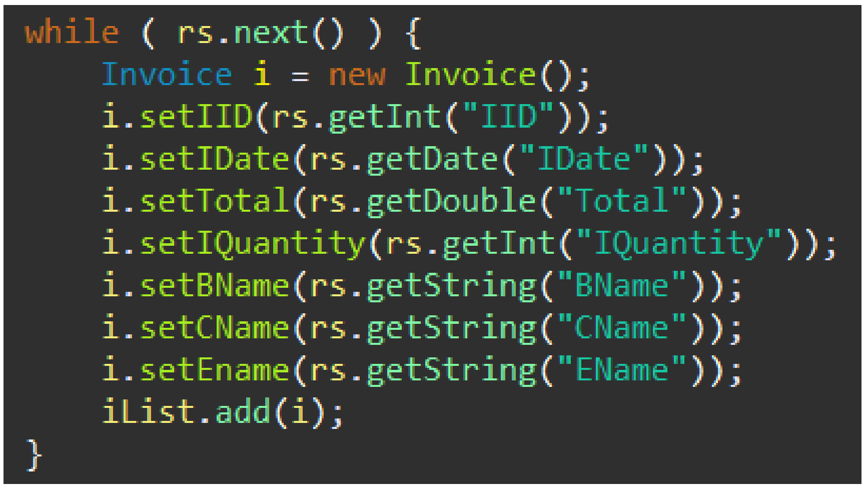
Example 2: Inside the while loop of getBooksDetails from BookDAO Class, there is an object created as known object and assigned that into a name 'b'. then assigned all the retrieved values from database to that object and added to a book list namely 'bList'.



```
while ( rs.next() ) {
    Books b = new Books();
    b.setBID(rs.getInt("BID"));
    b.setBName(rs.getString("BName"));
    b.setBStock(rs.getInt("BStock"));
    b.setBUnitPrice(rs.getDouble("BUnitPrice"));
    b.setCName(rs.getString("CName"));
    bList.add(b);
}
```

Figure 32 Known object - example 2

Example 3: Inside the while loop of getInvoiceDetails from InvoiceDAO Class, there is an object created as known object and assigned that into a name 'i'. then assigned all the retrieved values from database to that object and added to an invoice list namely 'iList'.



```
while ( rs.next() ) {
    Invoice i = new Invoice();
    i.setIID(rs.getInt("IID"));
    i.setIDate(rs.getDate("IDate"));
    i.setTotal(rs.getDouble("Total"));
    i.setIQuantity(rs.getInt("IQuantity"));
    i.setBName(rs.getString("BName"));
    i.setCName(rs.getString("CName"));
    i.setEname(rs.getString("EName"));
    iList.add(i);
}
```

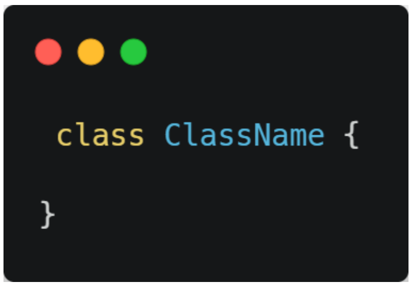
4.4.2. Class

Alternatively, a class can serve as a blueprint from which an individual object can be created. Also, class doesn't occupy any spaces in RAM. There are 3 types of classes are there.

1. Regular class

   Regular classes are offer multiple object creation and inheritance

   General syntax:



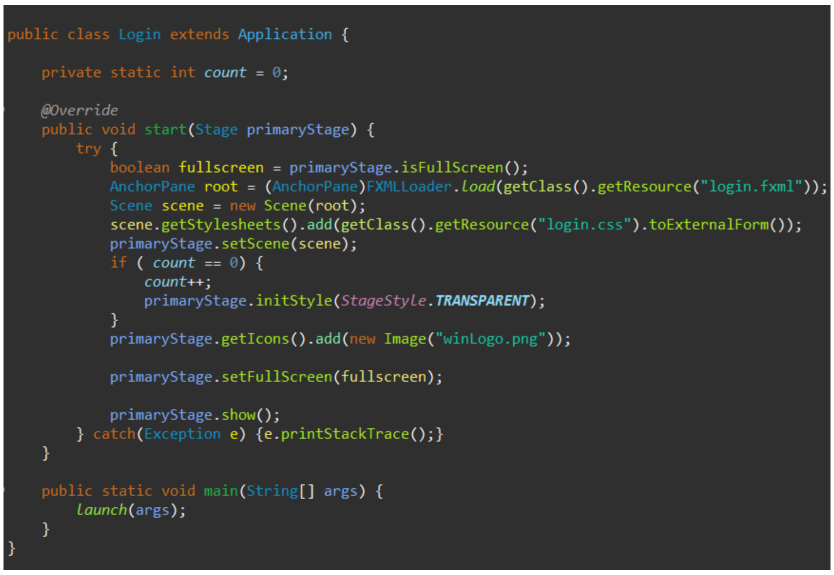
```
class ClassName {

}
```

Figure 33 Regular class - General syntax

Used in the system:

All classes are created as regular class in this system

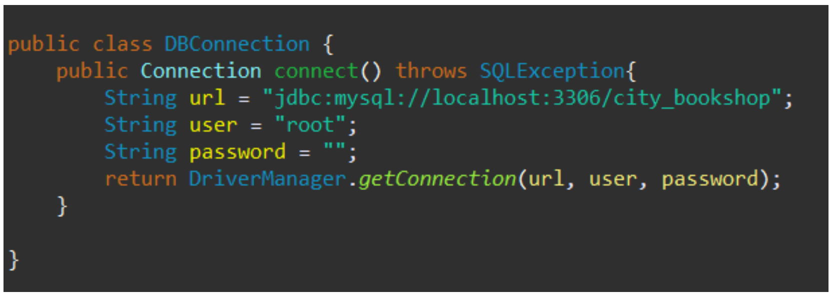
Example 1: Login class



```
public class Login extends Application {

    private static int count = 0;

    @Override
    public void start(Stage primaryStage) {
        try {
            boolean fullscreen = primaryStage.isFullScreen();
            AnchorPane root = (AnchorPane)FXMLLoader.load(getClass().getResource("login.fxml"));
            Scene scene = new Scene(root);
            scene.getStylesheets().add(getClass().getResource("login.css").toExternalForm());
            primaryStage.setScene(scene);
            if ( count == 0) {
                count++;
                primaryStage.initStyle(StageStyle.TRANSPARENT);
            }
            primaryStage.getIcons().add(new Image("winLogo.png"));

            primaryStage.setFullScreen(fullscreen);

            primaryStage.show();
        } catch(Exception e) {e.printStackTrace();}
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Figure 34 Login class – example 1

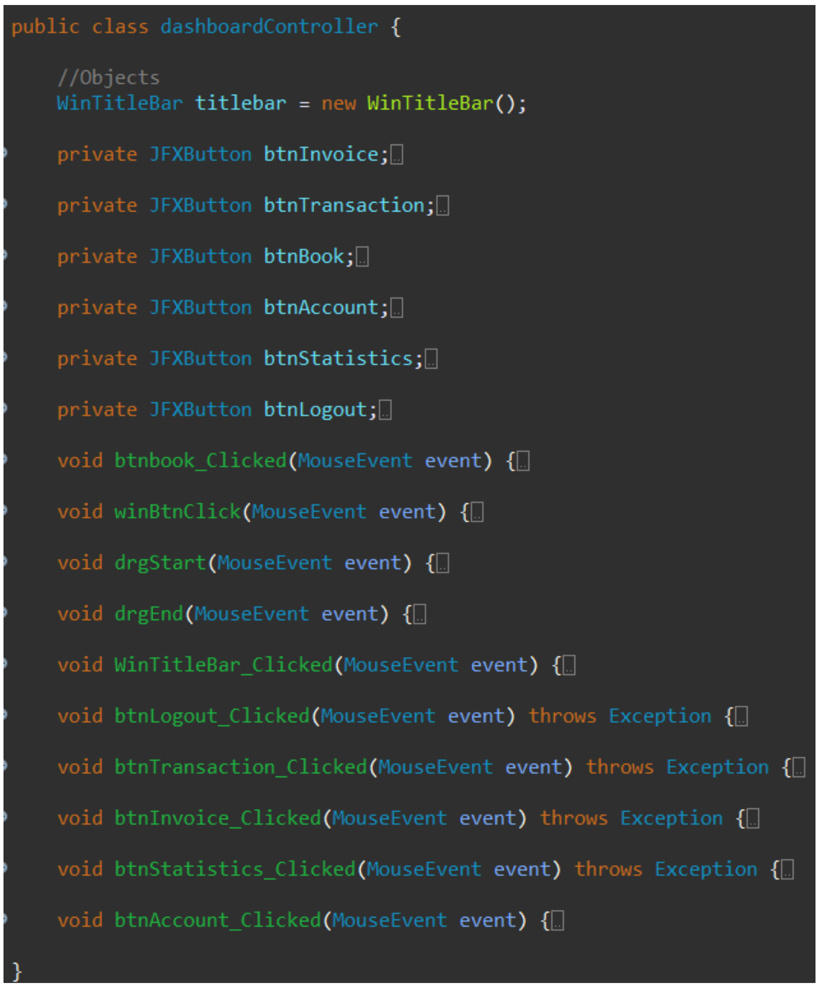
Example 2: DBConnection



```
public class DBConnection {
    public Connection connect() throws SQLException{
        String url = "jdbc:mysql://localhost:3306/city_bookshop";
        String user = "root";
        String password = "";
        return DriverManager.getConnection(url, user, password);
    }

}
```

Figure 35 DBConnection - example 2

Example 3: dashboardController



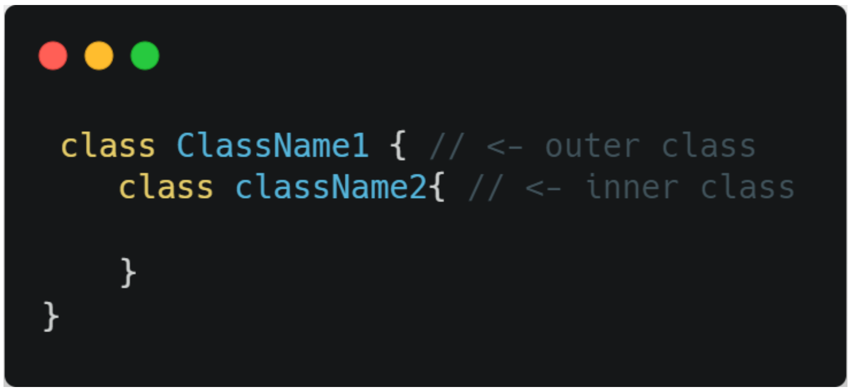
```
public class dashboardController {

    //Objects
    WinTitleBar titlebar = new WinTitleBar();

    private JFXButton btnInvoice;

    private JFXButton btnTransaction;

    private JFXButton btnBook;

    private JFXButton btnAccount;

    private JFXButton btnStatistics;

    private JFXButton btnLogout;

    void btnbook_Clicked(MouseEvent event) {

    void winBtnClick(MouseEvent event) {

    void drgStart(MouseEvent event) {

    void drgEnd(MouseEvent event) {

    void WinTitleBar_Clicked(MouseEvent event) {

    void btnLogout_Clicked(MouseEvent event) throws Exception {

    void btnTransaction_Clicked(MouseEvent event) throws Exception {

    void btnInvoice_Clicked(MouseEvent event) throws Exception {

    void btnStatistics_Clicked(MouseEvent event) throws Exception {

    void btnAccount_Clicked(MouseEvent event) {

}
```

Figure 36 dashboardController - example 3

2. Inner class

   When a class created inside a class, then that class is known as inner class

   General syntax:



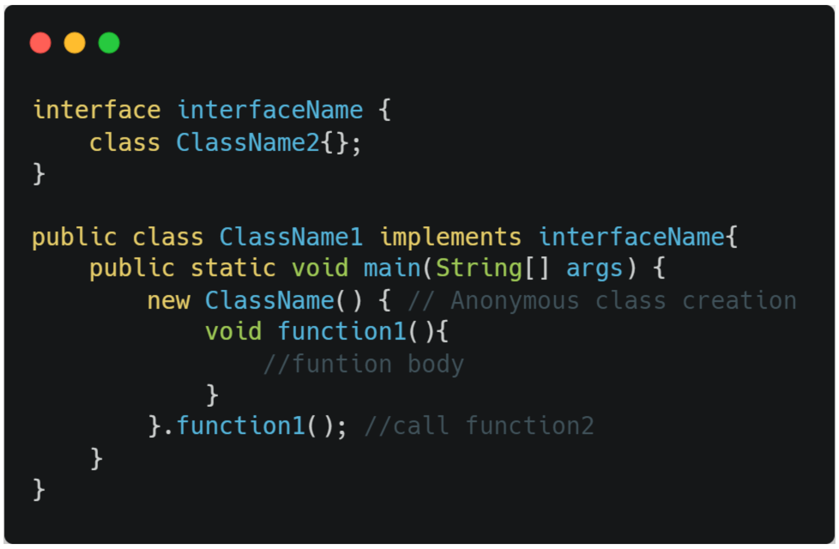
```
class ClassName1 { // <- outer class
    class className2{ // <- inner class

    }
}
```

Figure 37 Inner class - general syntax

3. Anonymous class

Anonymous classes cannot call again because they don't have a name. this kind of class are created for change implementation of the class to various situations. For implement this kind of class there is an already a abstract class need for it. In a simple word anonymous classes are used give implementation for abstract classes in different scenarios,

General Syntax:



```java
interface interfaceName {
    class ClassName2{};
}

public class ClassName1 implements interfaceName{
    public static void main(String[] args) {
        new ClassName() { // Anonymous class creation
            void function1(){
                //funtion body
            }
        }.function1(); //call function2
    }
}
```

Figure 38 Anonymous class - general syntax

4.4.3. Inheritance

Inheritance is a technique in Java that allows one object to inherit all of its parent object's attributes and actions. It is possible to build new Java classes that are based on existing ones. It is possible to reuse methods and properties from the parent class when you inherit from it.

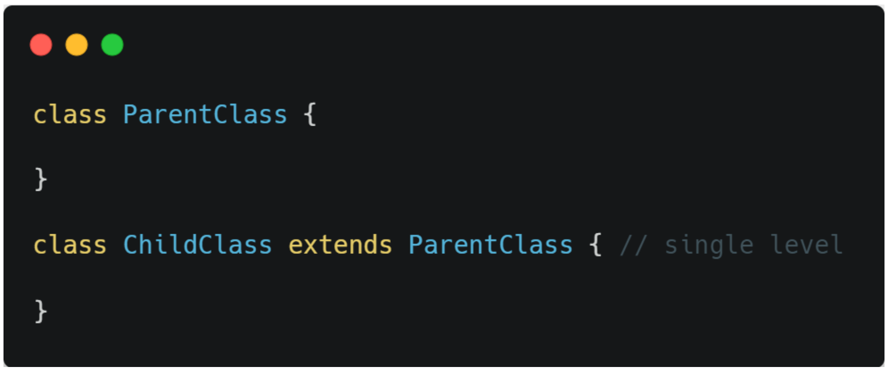
(Inheritance in Java - Javatpoint, 2021)

There are mainly 4 types of inheritance are there:

1. Single level

Single level inheritance is, one class inherit another class. It means there will be one parent class and one child class.

General syntax:

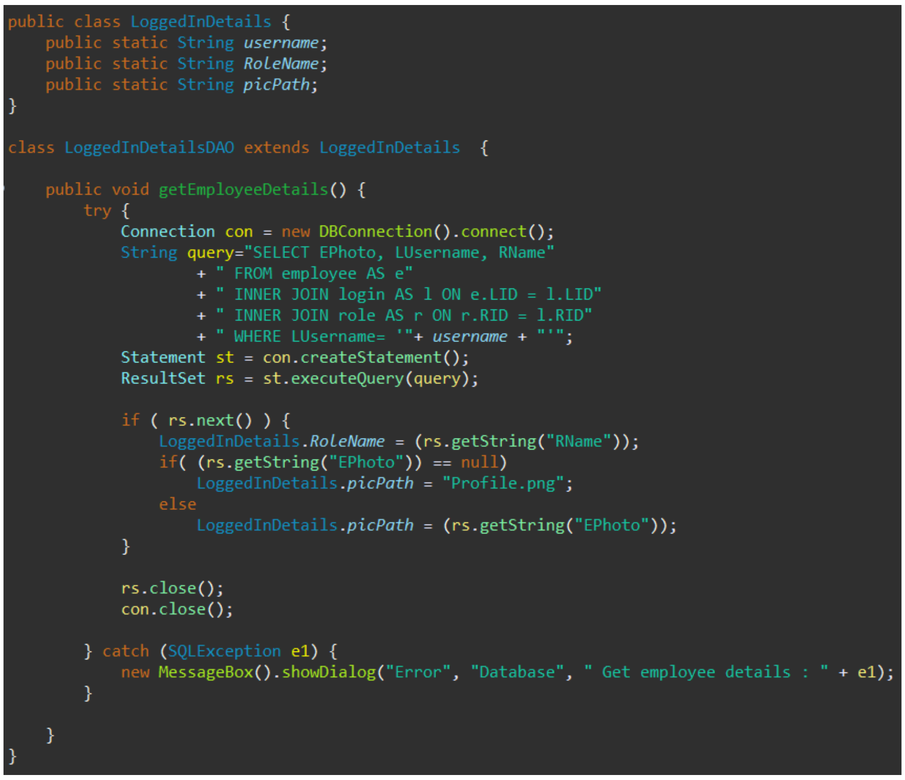


```
class ParentClass {

}

class ChildClass extends ParentClass { // single level

}
```

Figure 39 Single level inheritance - general syntax

Used in the system:

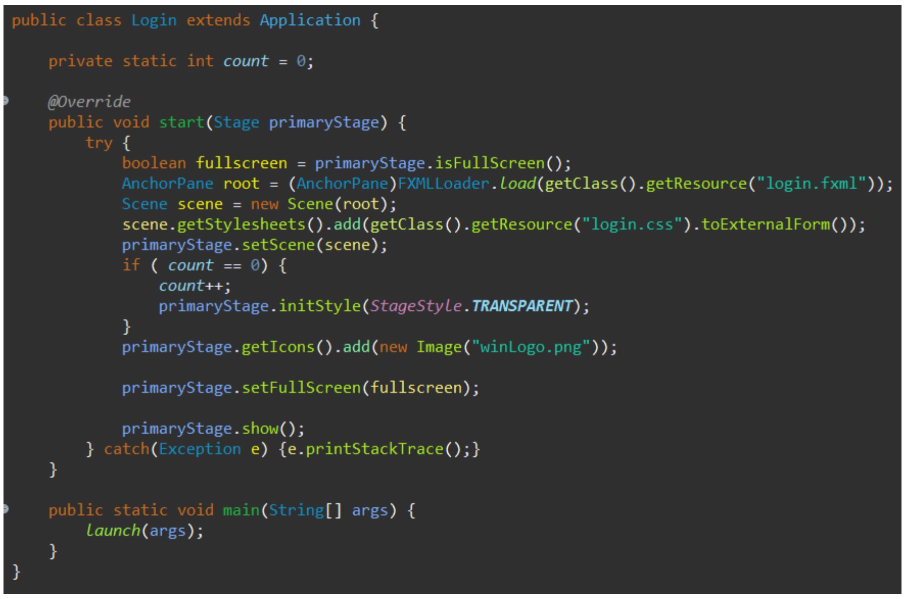
Example 1: In LoggedInDetails Class, LoggedInDetailsDAO class inheriting LoggedInDetails class at single level



```
public class LoggedInDetails {
    public static String username;
    public static String RoleName;
    public static String picPath;
}

class LoggedInDetailsDAO extends LoggedInDetails  {

    public void getEmployeeDetails() {
        try {
            Connection con = new DBConnection().connect();
            String query="SELECT EPhoto, LUsername, RName"
                    + " FROM employee AS e"
                    + " INNER JOIN login AS l ON e.LID = l.LID"
                    + " INNER JOIN role AS r ON r.RID = l.RID"
                    + " WHERE LUsername= '"+ username + "'";
            Statement st = con.createStatement();
            ResultSet rs = st.executeQuery(query);

            if ( rs.next() ) {
                LoggedInDetails.RoleName = (rs.getString("RName"));
                if( (rs.getString("EPhoto")) == null)
                    LoggedInDetails.picPath = "Profile.png";
                else
                    LoggedInDetails.picPath = (rs.getString("EPhoto"));
            }

            rs.close();
            con.close();

        } catch (SQLException e1) {
            new MessageBox().showDialog("Error", "Database", " Get employee details : " + e1);
        }

    }
}
```

Figure 40 Single level inheritance - Example 1

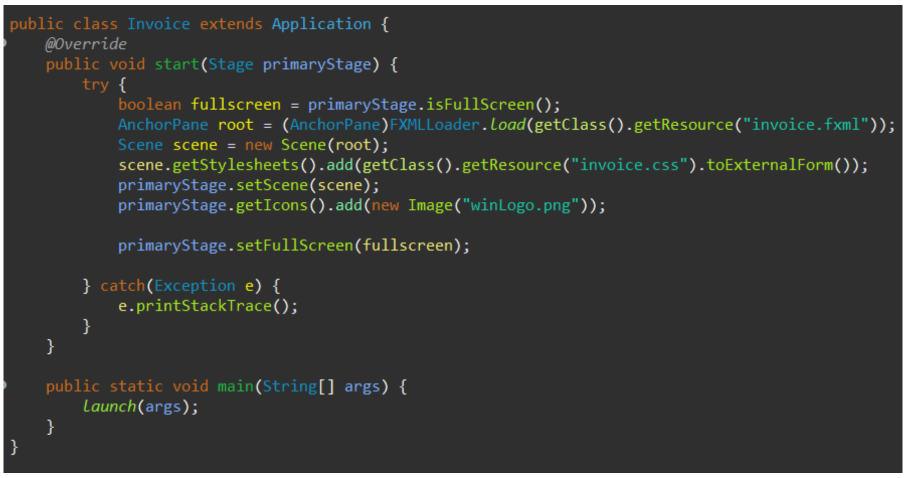
Example 2: In Login class. Login class inheriting Application class at single level



```
public class Login extends Application {

    private static int count = 0;

    @Override
    public void start(Stage primaryStage) {
        try {
            boolean fullscreen = primaryStage.isFullScreen();
            AnchorPane root = (AnchorPane)FXMLLoader.load(getClass().getResource("login.fxml"));
            Scene scene = new Scene(root);
            scene.getStylesheets().add(getClass().getResource("login.css").toExternalForm());
            primaryStage.setScene(scene);
            if ( count == 0) {
                count++;
                primaryStage.initStyle(StageStyle.TRANSPARENT);
            }
            primaryStage.getIcons().add(new Image("winLogo.png"));

            primaryStage.setFullScreen(fullscreen);

            primaryStage.show();
        } catch(Exception e) {e.printStackTrace();}
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Figure 41 Single level inheritance - example 2

Example 3: In Invoice class, Invoice class inheriting Application at single level



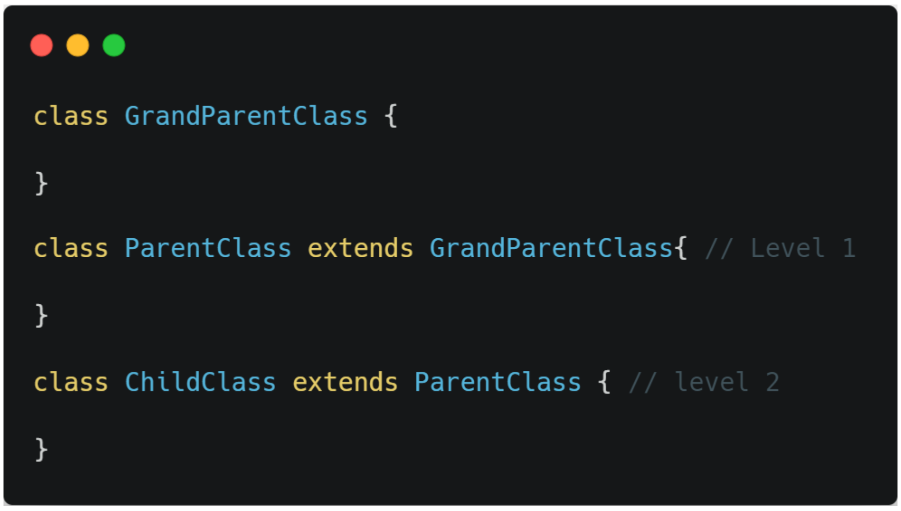
```
public class Invoice extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            boolean fullscreen = primaryStage.isFullScreen();
            AnchorPane root = (AnchorPane)FXMLLoader.load(getClass().getResource("invoice.fxml"));
            Scene scene = new Scene(root);
            scene.getStylesheets().add(getClass().getResource("invoice.css").toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.getIcons().add(new Image("winLogo.png"));

            primaryStage.setFullScreen(fullscreen);

        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Figure 42 Single level Inheritance - example 3

2. Multi-level

    Multi-level inheritance is, one class inherit another class and another class inherit the inherited class. It means there will be one grandparent class, one parent class and one child class

General syntax:



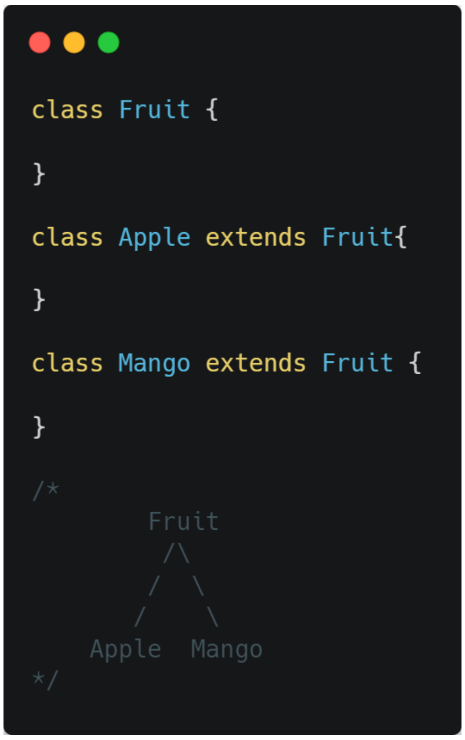
```
class GrandParentClass {

}

class ParentClass extends GrandParentClass{ // Level 1

}

class ChildClass extends ParentClass { // level 2

}
```

Figure 43 multi-level inheritance - general syntax

3. Hierarchical

   Hierarchical inheritance means two classes inherits the same class

   General syntax:



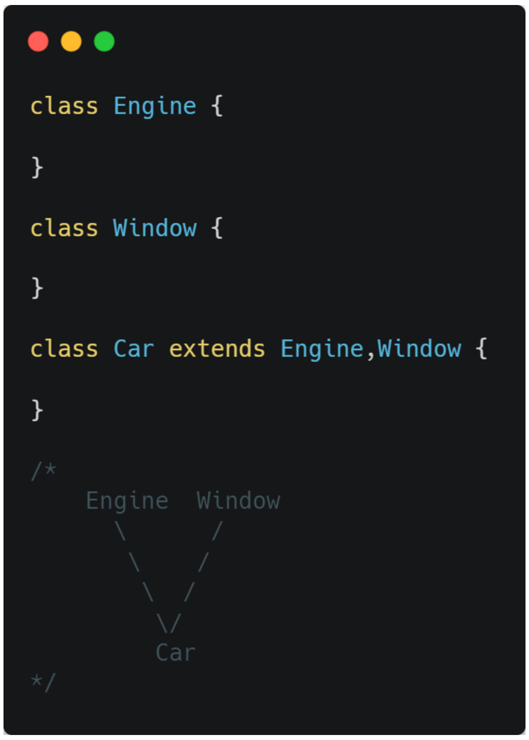
```
class Fruit {

}

class Apple extends Fruit{

}

class Mango extends Fruit {

}

/*
        Fruit
         /\
        /  \
       /    \
    Apple  Mango
*/
```

Figure 44 Hierarchical inheritance - general syntax

4. Multiple

Multiple inheritance means, a class inherits more than one class

General syntax;



```java
class Engine {

}

class Window {

}

class Car extends Engine,Window {

}

/*
    Engine   Window
       \       /
        \     /
         \   /
          \/
          Car
*/
```

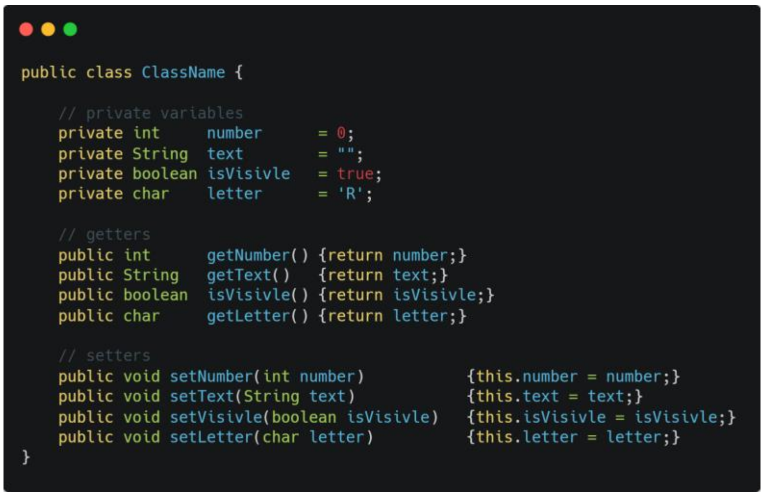
Figure 45 Multiple inherence - general syntax

4.4.4. Encapsulation

Data (variables) and code (methods) are encapsulated in Java as a single entity. Classes that implement this technique can hide their variables and only provide access to them through the methods of their own class.

(Java - Encapsulation, 2021)

General syntax:

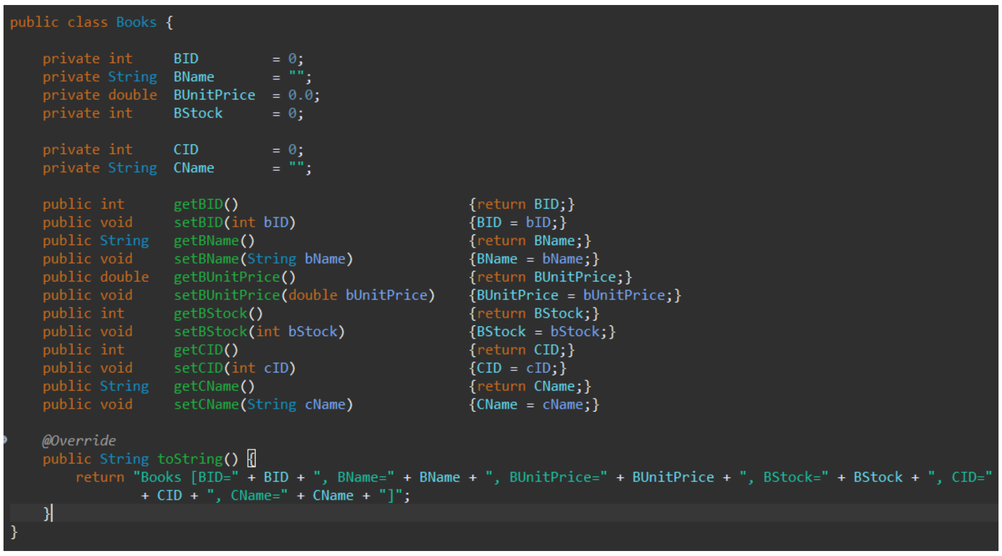


```java
public class ClassName {

    // private variables
    private int     number      = 0;
    private String  text        = "";
    private boolean isVisivle   = true;
    private char    letter      = 'R';

    // getters
    public int      getNumber() {return number;}
    public String   getText()   {return text;}
    public boolean  isVisivle() {return isVisivle;}
    public char     getLetter() {return letter;}

    // setters
    public void setNumber(int number)           {this.number = number;}
    public void setText(String text)            {this.text = text;}
    public void setVisivle(boolean isVisivle)   {this.isVisivle = isVisivle;}
    public void setLetter(char letter)          {this.letter = letter;}
}
```

Figure 46 Encapsulation - general syntax

Used in the system:

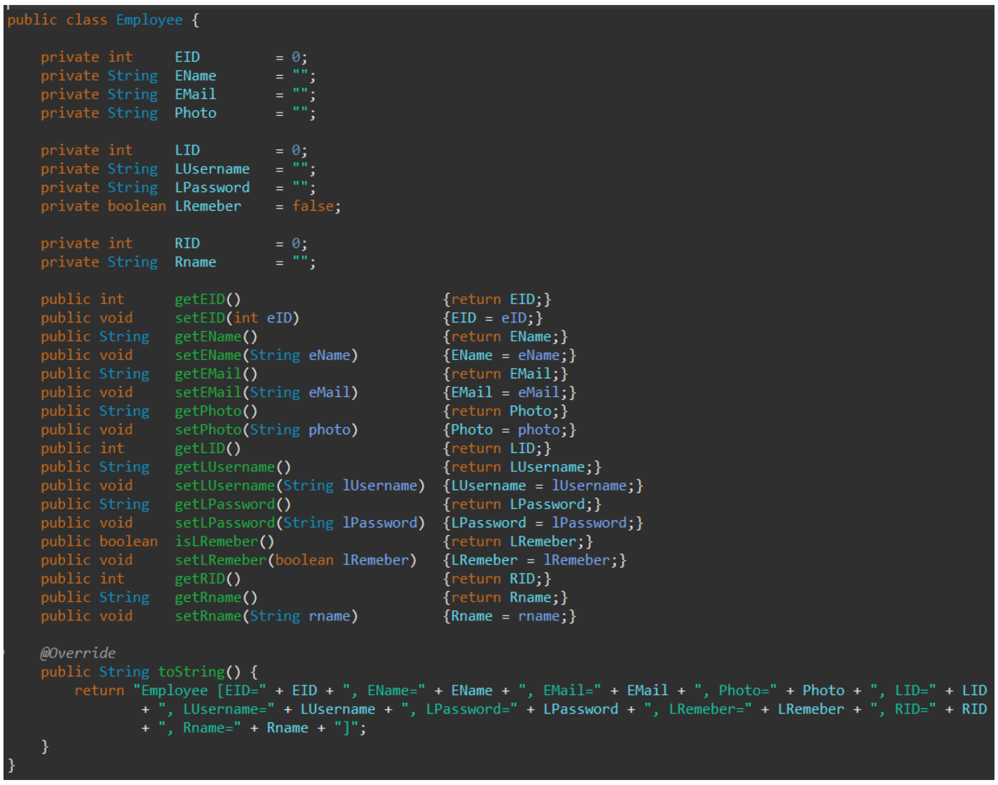
Example 1: Book class



```java
public class Books {

    private int     BID         = 0;
    private String  BName       = "";
    private double  BUnitPrice  = 0.0;
    private int     BStock      = 0;

    private int     CID         = 0;
    private String  CName       = "";

    public int      getBID()                                {return BID;}
    public void     setBID(int bID)                         {BID = bID;}
    public String   getBName()                              {return BName;}
    public void     setBName(String bName)                  {BName = bName;}
    public double   getBUnitPrice()                         {return BUnitPrice;}
    public void     setBUnitPrice(double bUnitPrice)        {BUnitPrice = bUnitPrice;}
    public int      getBStock()                             {return BStock;}
    public void     setBStock(int bStock)                   {BStock = bStock;}
    public int      getCID()                                {return CID;}
    public void     setCID(int cID)                         {CID = cID;}
    public String   getCName()                              {return CName;}
    public void     setCName(String cName)                  {CName = cName;}

    @Override
    public String toString() {
        return "Books [BID=" + BID + ", BName=" + BName + ", BUnitPrice=" + BUnitPrice + ", BStock=" + BStock + ", CID="
                + CID + ", CName=" + CName + "]";
    }
}
```

Figure 47 Encapsulation - example 1

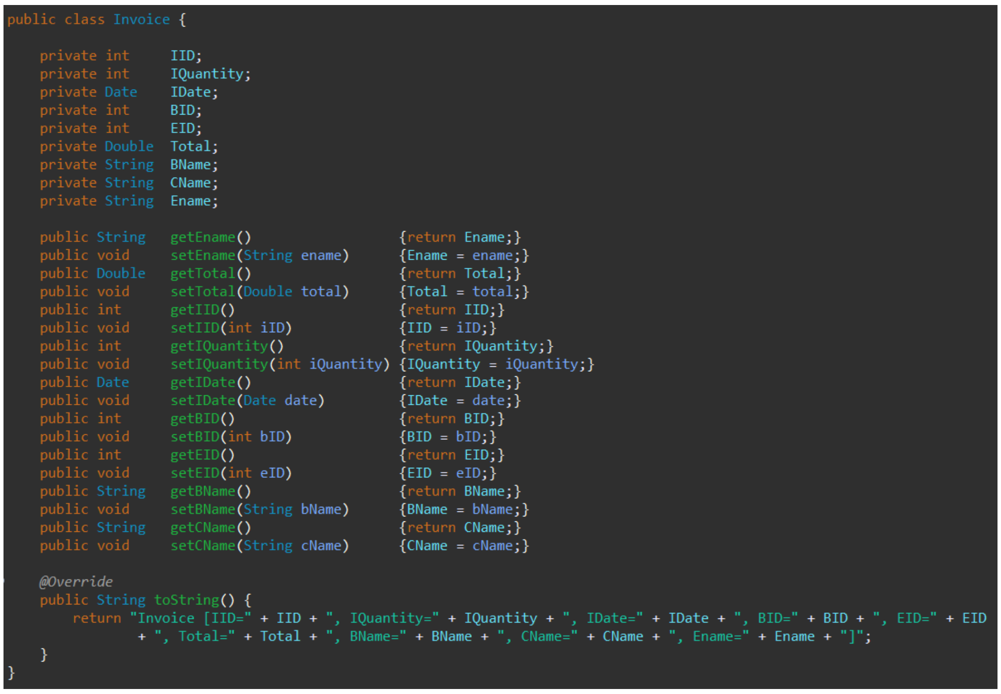
Example 2: Employee class



```java
public class Employee {

    private int     EID         = 0;
    private String  EName       = "";
    private String  EMail       = "";
    private String  Photo       = "";

    private int     LID         = 0;
    private String  LUsername   = "";
    private String  LPassword   = "";
    private boolean LRemeber    = false;

    private int     RID         = 0;
    private String  Rname       = "";

    public int      getEID()                                {return EID;}
    public void     setEID(int eID)                         {EID = eID;}
    public String   getEName()                              {return EName;}
    public void     setEName(String eName)                  {EName = eName;}
    public String   getEMail()                              {return EMail;}
    public void     setEMail(String eMail)                  {EMail = eMail;}
    public String   getPhoto()                              {return Photo;}
    public void     setPhoto(String photo)                  {Photo = photo;}
    public int      getLID()                                {return LID;}
    public String   getLUsername()                          {return LUsername;}
    public void     setLUsername(String lUsername)          {LUsername = lUsername;}
    public String   getLPassword()                          {return LPassword;}
    public void     setLPassword(String lPassword)          {LPassword = lPassword;}
    public boolean  isLRemeber()                            {return LRemeber;}
    public void     setLRemeber(boolean lRemeber)           {LRemeber = lRemeber;}
    public int      getRID()                                {return RID;}
    public String   getRname()                              {return Rname;}
    public void     setRname(String rname)                  {Rname = rname;}

    @Override
    public String toString() {
        return "Employee [EID=" + EID + ", EName=" + EName + ", EMail=" + EMail + ", Photo=" + Photo + ", LID=" + LID
                + ", LUsername=" + LUsername + ", LPassword=" + LPassword + ", LRemeber=" + LRemeber + ", RID=" + RID
                + ", Rname=" + Rname + "]";
    }
}
```

Figure 48 Encapsulation - example 2

Example 3:



```
public class Invoice {

    private int     IID;
    private int     IQuantity;
    private Date    IDate;
    private int     BID;
    private int     EID;
    private Double  Total;
    private String  BName;
    private String  CName;
    private String  Ename;

    public String   getEname()                   {return Ename;}
    public void     setEname(String ename)       {Ename = ename;}
    public Double   getTotal()                   {return Total;}
    public void     setTotal(Double total)       {Total = total;}
    public int      getIID()                     {return IID;}
    public void     setIID(int iID)              {IID = iID;}
    public int      getIQuantity()               {return IQuantity;}
    public void     setIQuantity(int iQuantity) {IQuantity = iQuantity;}
    public Date     getIDate()                   {return IDate;}
    public void     setIDate(Date date)          {IDate = date;}
    public int      getBID()                     {return BID;}
    public void     setBID(int bID)              {BID = bID;}
    public int      getEID()                     {return EID;}
    public void     setEID(int eID)              {EID = eID;}
    public String   getBName()                   {return BName;}
    public void     setBName(String bName)       {BName = bName;}
    public String   getCName()                   {return CName;}
    public void     setCName(String cName)       {CName = cName;}

    @Override
    public String toString() {
        return "Invoice [IID=" + IID + ", IQuantity=" + IQuantity + ", IDate=" + IDate + ", BID=" + BID + ", EID=" + EID
                + ", Total=" + Total + ", BName=" + BName + ", CName=" + CName + ", Ename=" + Ename + "]";
    }
}
```

Figure 49 Encapsulation - example 3

4.4.5. Polymorphism

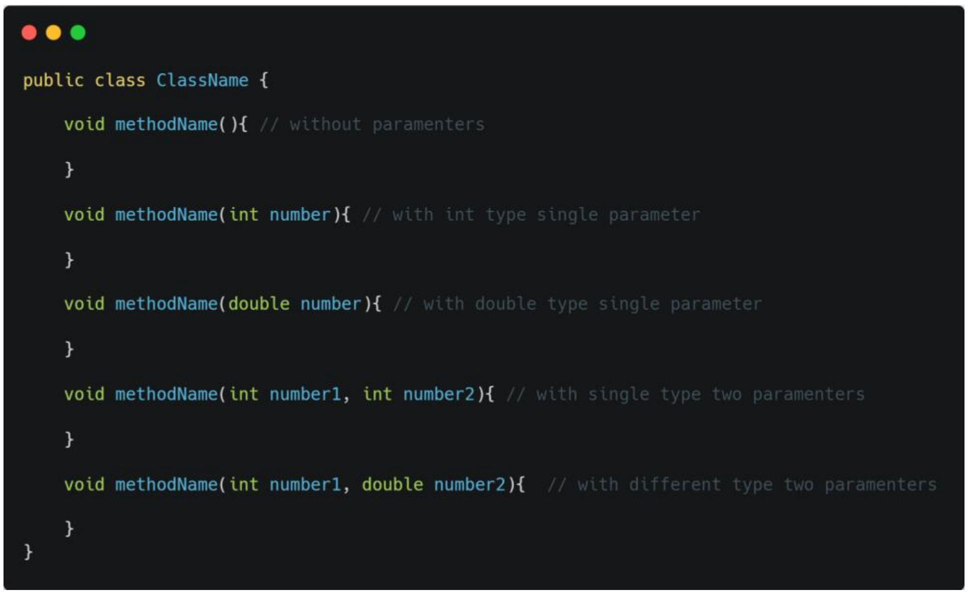
In Java, polymorphism refers to an object's capacity to take on several forms. To put it another way, polymorphism in Java enables us to accomplish the same operation in a variety of ways.

(Polymorphism in Java and Java Polymorphism Examples, 2021)

4.4.5.1. Method overloading

Method overloading is a Java feature that allows us to define many methods with the same name in the same class, each of which performs differently. The term Overloaded Methods refers to methods that have more than one instance of the same name in the same class.

General syntax:

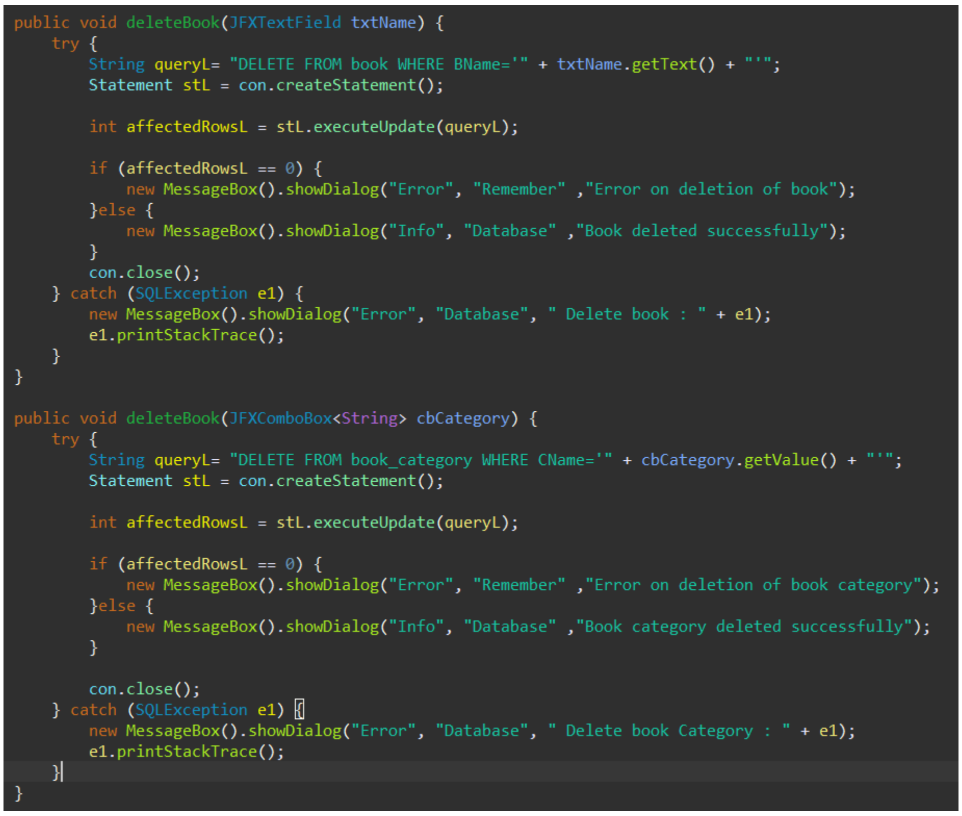


```java
public class ClassName {

    void methodName( ){ // without paramenters

    }

    void methodName(int number){ // with int type single parameter

    }

    void methodName(double number){ // with double type single parameter

    }

    void methodName(int number1, int number2){ // with single type two paramenters

    }

    void methodName(int number1, double number2){  // with different type two paramenters

    }
}
```

Figure 50 Method overloading - general syntax

Used in the system:

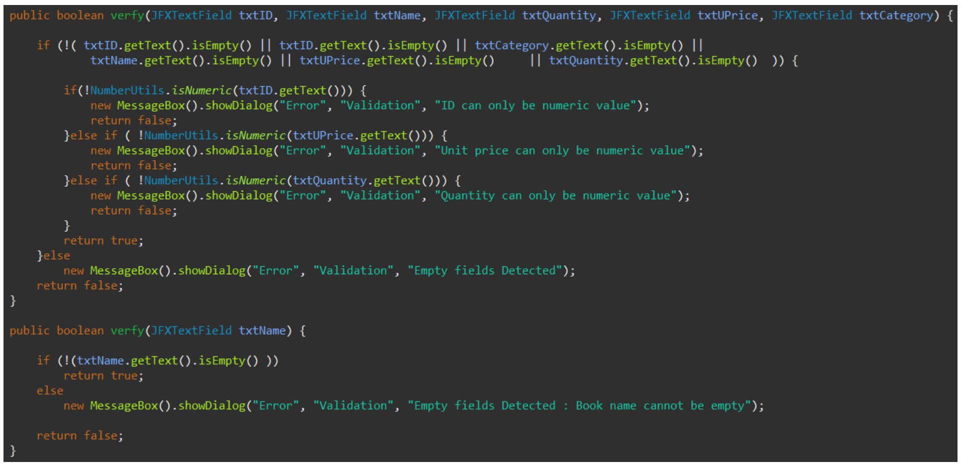
Example 1: In BookDAO class, deleteBook method overloaded with different type single parameter



```java
public void deleteBook(JFXTextField txtName) {
    try {
        String queryL= "DELETE FROM book WHERE BName='" + txtName.getText() + "'";
        Statement stL = con.createStatement();

        int affectedRowsL = stL.executeUpdate(queryL);

        if (affectedRowsL == 0) {
            new MessageBox().showDialog("Error", "Remember" ,"Error on deletion of book");
        }else {
            new MessageBox().showDialog("Info", "Database" ,"Book deleted successfully");
        }
        con.close();
    } catch (SQLException e1) {
        new MessageBox().showDialog("Error", "Database", " Delete book : " + e1);
        e1.printStackTrace();
    }
}

public void deleteBook(JFXComboBox<String> cbCategory) {
    try {
        String queryL= "DELETE FROM book_category WHERE CName='" + cbCategory.getValue() + "'";
        Statement stL = con.createStatement();

        int affectedRowsL = stL.executeUpdate(queryL);

        if (affectedRowsL == 0) {
            new MessageBox().showDialog("Error", "Remember" ,"Error on deletion of book category");
        }else {
            new MessageBox().showDialog("Info", "Database" ,"Book category deleted successfully");
        }

        con.close();
    } catch (SQLException e1) {
        new MessageBox().showDialog("Error", "Database", " Delete book Category : " + e1);
        e1.printStackTrace();
    }
}
```

Figure 51 Method overloading - example 1

Example 2: in Invoice validation class, verify method overloaded by multiple parameters varies.



```
public boolean verfy(JFXTextField txtID, JFXTextField txtName, JFXTextField txtQuantity, JFXTextField txtUPrice, JFXTextField txtCategory) {

    if (!( txtID.getText().isEmpty() || txtID.getText().isEmpty() || txtCategory.getText().isEmpty() ||
            txtName.getText().isEmpty() || txtUPrice.getText().isEmpty()     || txtQuantity.getText().isEmpty()  )) {

        if(!NumberUtils.isNumeric(txtID.getText())) {
            new MessageBox().showDialog("Error", "Validation", "ID can only be numeric value");
            return false;
        }else if ( !NumberUtils.isNumeric(txtUPrice.getText())) {
            new MessageBox().showDialog("Error", "Validation", "Unit price can only be numeric value");
            return false;
        }else if ( !NumberUtils.isNumeric(txtQuantity.getText())) {
            new MessageBox().showDialog("Error", "Validation", "Quantity can only be numeric value");
            return false;
        }
        return true;
    }else
        new MessageBox().showDialog("Error", "Validation", "Empty fields Detected");
    return false;
}

public boolean verfy(JFXTextField txtName) {

    if (!(txtName.getText().isEmpty() ))
        return true;
    else
        new MessageBox().showDialog("Error", "Validation", "Empty fields Detected : Book name cannot be empty");

    return false;
}
```

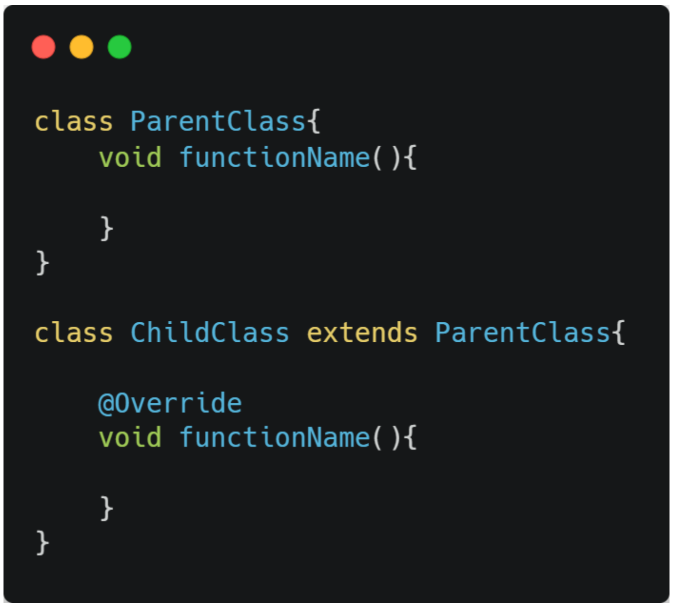
Figure 52 Method overloading - example 2

4.4.5.2. Method overriding

Overriding is a feature that allows a subclass or child class to offer a customized implementation of a method that is already available from one of its super-classes or parent classes. Subclasses are said to override super-classes when a method in a subclass has the same parameters, same name, or same signature, and also, same return type as a method in its superclass.

(Overriding in Java - GeeksforGeeks, 2021)

General syntax:

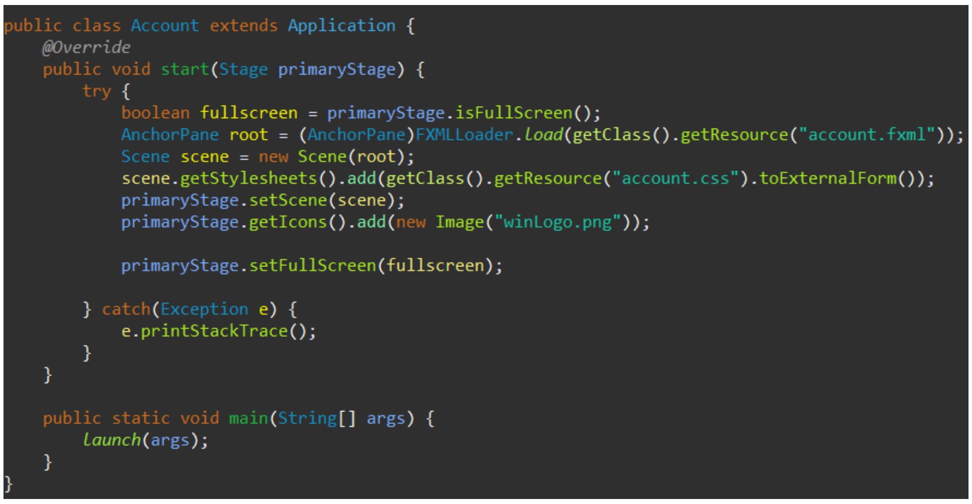


```
class ParentClass{
    void functionName(){

    }
}

class ChildClass extends ParentClass{

    @Override
    void functionName(){

    }
}
```

Figure 53 Method overriding - General syntax

Used in the system:

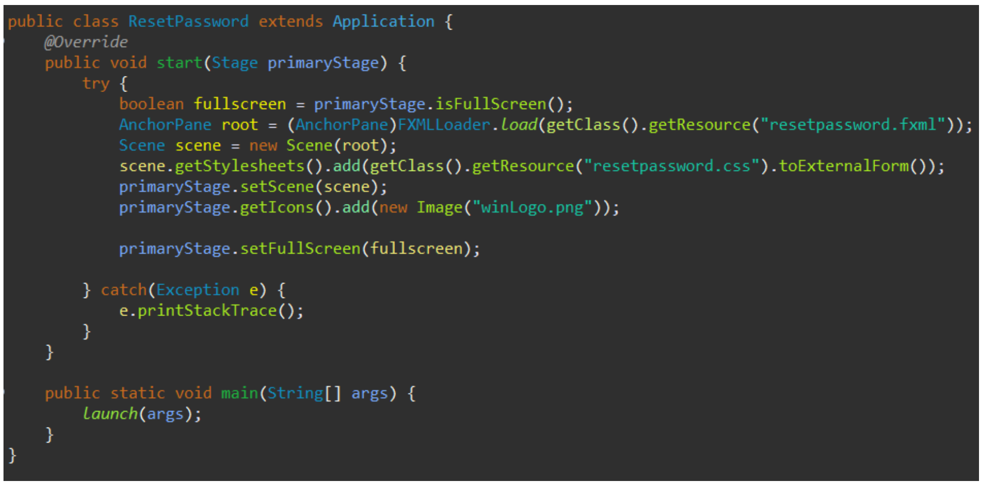
Example 1: Account class overriding start class from Application class



```
public class Account extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            boolean fullscreen = primaryStage.isFullScreen();
            AnchorPane root = (AnchorPane)FXMLLoader.load(getClass().getResource("account.fxml"));
            Scene scene = new Scene(root);
            scene.getStylesheets().add(getClass().getResource("account.css").toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.getIcons().add(new Image("winLogo.png"));

            primaryStage.setFullScreen(fullscreen);

        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Figure 54 Method overriding - example 1

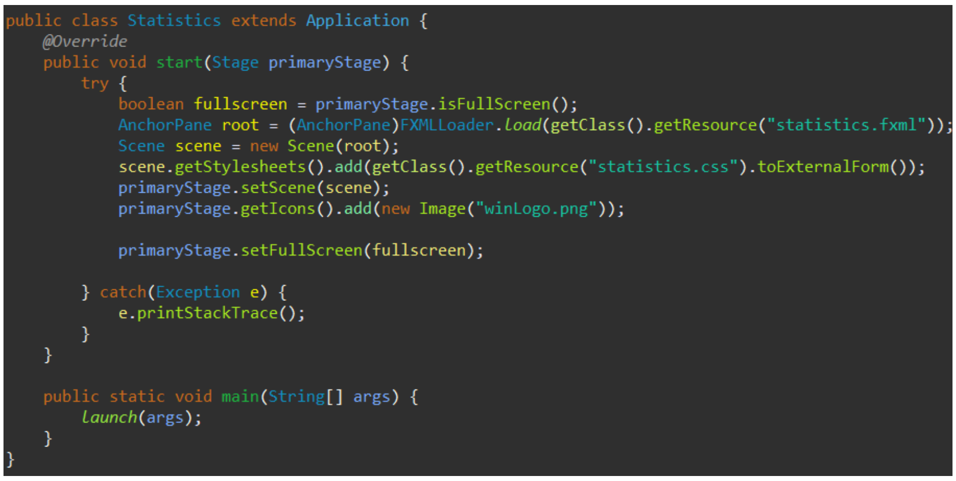
Example 2: RestPassword class overriding start class from Application class



```
public class ResetPassword extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            boolean fullscreen = primaryStage.isFullScreen();
            AnchorPane root = (AnchorPane)FXMLLoader.load(getClass().getResource("resetpassword.fxml"));
            Scene scene = new Scene(root);
            scene.getStylesheets().add(getClass().getResource("resetpassword.css").toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.getIcons().add(new Image("winLogo.png"));

            primaryStage.setFullScreen(fullscreen);

        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Figure 55 Method overriding - example 2

Example 3: Statistics class overriding start class from Application class



```
public class Statistics extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            boolean fullscreen = primaryStage.isFullScreen();
            AnchorPane root = (AnchorPane)FXMLLoader.load(getClass().getResource("statistics.fxml"));
            Scene scene = new Scene(root);
            scene.getStylesheets().add(getClass().getResource("statistics.css").toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.getIcons().add(new Image("winLogo.png"));

            primaryStage.setFullScreen(fullscreen);

        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

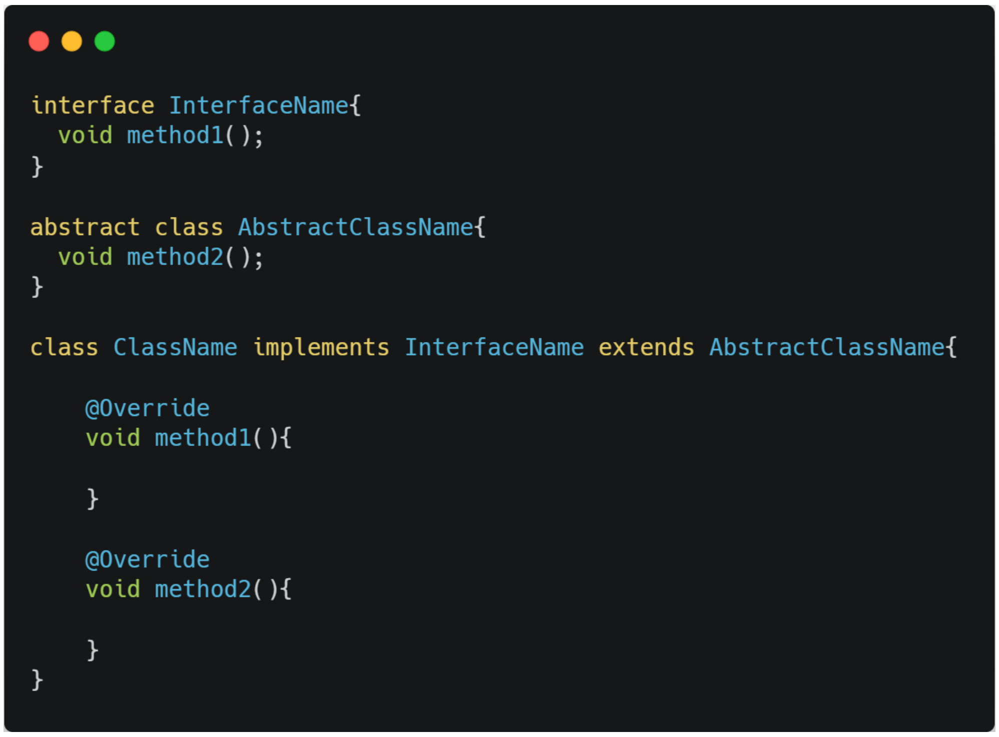
Figure 56 Method overriding - example 3

4.4.6. Abstraction

Abstraction is the technique of concealing some facts from the user and only giving them what they need to know. It is possible to create abstraction by using abstract classes or interfaces

(Java Abstraction, 2021)

General syntax:

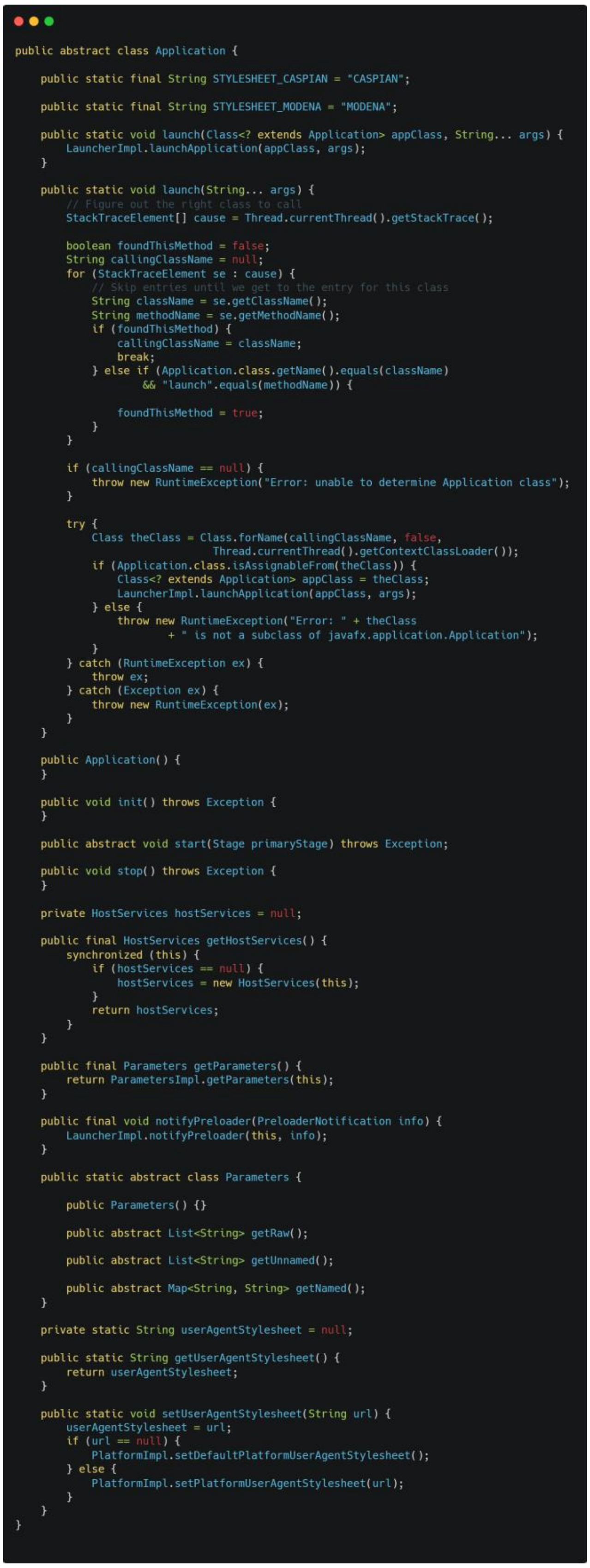


```
interface InterfaceName{
  void method1();
}

abstract class AbstractClassName{
  void method2();
}

class ClassName implements InterfaceName extends AbstractClassName{

    @Override
    void method1(){

    }

    @Override
    void method2(){

    }
}
```

Figure 57 Abstraction - General syntaxe

Used in the system:

Example 1: Application class



```java
public abstract class Application {

    public static final String STYLESHEET_CASPIAN = "CASPIAN";

    public static final String STYLESHEET_MODENA = "MODENA";

    public static void launch(Class<? extends Application> appClass, String... args) {
        LauncherImpl.launchApplication(appClass, args);
    }

    public static void launch(String... args) {
        // Figure out the right class to call
        StackTraceElement[] cause = Thread.currentThread().getStackTrace();

        boolean foundThisMethod = false;
        String callingClassName = null;
        for (StackTraceElement se : cause) {
            // Skip entries until we get to the entry for this class
            String className = se.getClassName();
            String methodName = se.getMethodName();
            if (foundThisMethod) {
                callingClassName = className;
                break;
            } else if (Application.class.getName().equals(className)
                    && "launch".equals(methodName)) {

                foundThisMethod = true;
            }
        }

        if (callingClassName == null) {
            throw new RuntimeException("Error: unable to determine Application class");
        }

        try {
            Class theClass = Class.forName(callingClassName, false,
                               Thread.currentThread().getContextClassLoader());
            if (Application.class.isAssignableFrom(theClass)) {
                Class<? extends Application> appClass = theClass;
                LauncherImpl.launchApplication(appClass, args);
            } else {
                throw new RuntimeException("Error: " + theClass
                        + " is not a subclass of javafx.application.Application");
            }
        } catch (RuntimeException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new RuntimeException(ex);
        }
    }

    public Application() {
    }

    public void init() throws Exception {
    }

    public abstract void start(Stage primaryStage) throws Exception;

    public void stop() throws Exception {
    }

    private HostServices hostServices = null;

    public final HostServices getHostServices() {
        synchronized (this) {
            if (hostServices == null) {
                hostServices = new HostServices(this);
            }
            return hostServices;
        }
    }

    public final Parameters getParameters() {
        return ParametersImpl.getParameters(this);
    }

    public final void notifyPreloader(PreloaderNotification info) {
        LauncherImpl.notifyPreloader(this, info);
    }

    public static abstract class Parameters {

        public Parameters() {}

        public abstract List<String> getRaw();

        public abstract List<String> getUnnamed();

        public abstract Map<String, String> getNamed();
    }

    private static String userAgentStylesheet = null;

    public static String getUserAgentStylesheet() {
        return userAgentStylesheet;
    }

    public static void setUserAgentStylesheet(String url) {
        userAgentStylesheet = url;
        if (url == null) {
            PlatformImpl.setDefaultPlatformUserAgentStylesheet();
        } else {
            PlatformImpl.setPlatformUserAgentStylesheet(url);
        }
    }
}
```

Figure 58 Abstraction - example 1

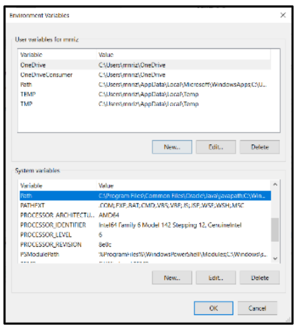
5. User manual

5.1. Prerequisites

To run this application there are two basic requirements are there. The entire system developed based on JavaFX. So, JavaFX SDK is need to start the application. Like vise the system used citybook_shop database which created in MySQL database software for data management. Due that MySQL database software and citybook_shop database is needed

Steps for the install prerequisites:

- JavaFX SDK V16
    1. Click https://gluonhq.com/download/javafx-16-sdk-windows/ this links to download JavaFX SDK V16.
    2. Go to download location and extract the file to C drive.
    3. In Windows 10, type in start search bar as "Edit the system environment variables" and click it.
    4. Search path in system variable list and click to edit

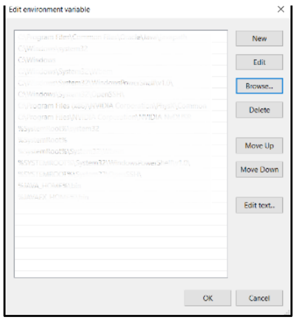


Variable	Value
OneDrive	C:\Users\mnriz\OneDrive
OneDriveConsumer	C:\Users\mnriz\OneDrive
Path	C:\Users\mnriz\AppData\Local\Microsoft\WindowsApps;C:\U...
TEMP	C:\Users\mnriz\AppData\Local\Temp
TMP	C:\Users\mnriz\AppData\Local\Temp

Variable	Value
Path	C:\Program Files\Common Files\Oracle\Java\javapath;C:\Win...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTU...	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 142 Stepping 12, GenuineIntel
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	8e0c
PSModulePath	%ProgramFiles%\WindowsPowerShell\Modules;C:\Windows\s...

Figure 59 Environment variables

    5. Click browse and direct to "c:\ \javafx-sdk-16 \bin" and click ok then click ok and then click



```
C:\Program Files\Common Files\Oracle\Java\javapath
C:\Windows\system32
C:\Windows
C:\Windows\System32\Wbem
C:\Windows\System32\WindowsPowerShell\v1.0\
C:\Windows\System32\OpenSSH\
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common
C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
%JAVA_HOME%\bin
%JAVAFX_HOME%\bin
```

Figure 60 Edit environment variables

- MySQL
    1. Click https://sourceforge.net/projects/wampserver/files/latest/download this links to download WAMP Server
    2. After download, click the setup.exe and run it

3. Click next to all in the setup wizard but in one place there will be option selection will be display. In that moment select PHP last version and MySQL last version. And install it
4. After installation click the WAMP server and run it
5. Open a web browser and type 127.0.0.1. in the URL bar
6. Click phpMyAdmin
7. Type root for username and give blank for password. Select MySql in database selection
8. Click Go

- citybook_shop
    1. Login in PhpMyAdmin with MySQL
    2. Click import tab
    3. Click choose file
    4. Browse to city book shop installation folder and go to database folder and select citybook_shop.sql
    5. Click Go

5.2. Login page

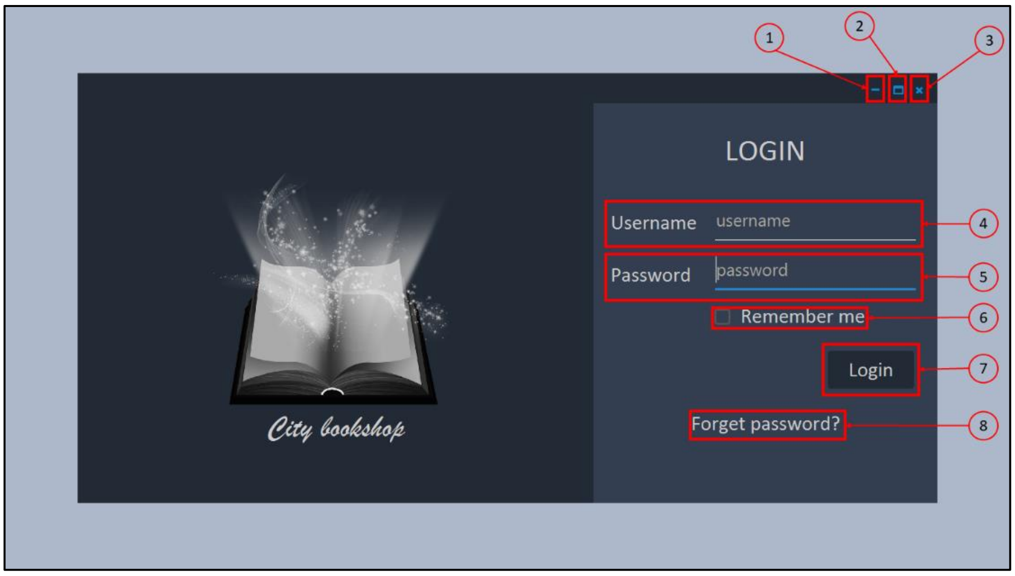


Figure 61 Login page

Login page is the first page when the application is start to run. In this page use need to enter correct username and password or else there will be error message will be thrown via a message box. There are two optional programs namely Remember me and Forget password for remember the user’s login details for next login and reset the password when user forget their password respectively.

Components of Login page

1. Minimize – Minimize the application
2. Maximize – Maximize the application
3. Close – Close the application
4. Username – Enter correct username
5. Password – Enter correct password
6. Remember me – When you click this system will remember your login details. So, in next login you don’t need to enter username and password.
7. Login – Click this button for login in
8. Forget password – Click this when you forget your password. It will direct you to forget password page

Events of Login page

1. Username field or password field is empty – Error Message: Empty fields detected
2. Username is wrong – Error message: Username is wrong
3. Username is Correct but password is wrong – Error message: Wrong password
4. Username and password are correct – Open Dashboard page

5.3. Forget password page

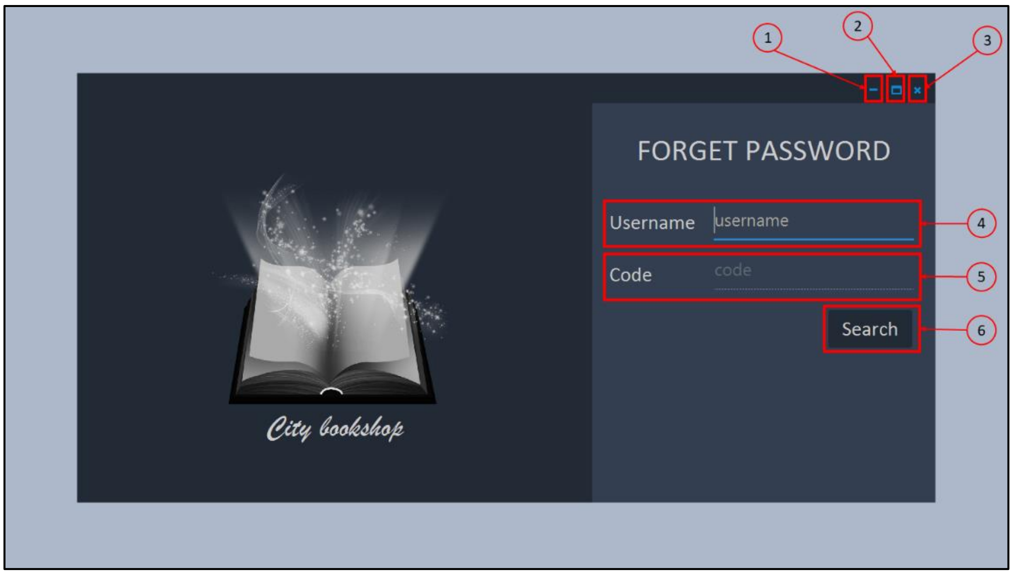


Figure 62 Forget password page

The motive of this page is to verify the user. First user needs to enter username in the username field and click search button. Then system will check whether username is correct or not. When username is correct then a verification code will send to a email address which they entered in to the system. Then user need to enter the verification code in filed as same as they got via mail. If that code is matched then it will direct them into reset password page

Components of Forget password page

1. Minimize – Minimize the application
2. Maximize – Maximize the application
3. Close – Close the application
4. Username – Enter correct username and click search
5. Code – Enter the verification code that you get from your email address
6. Submit (Search will change into submit) – Click submit, it will direct you to the reset password page

Events of Forget password page

1. Username field or Code field is empty – Error message: Empty fields detected
2. Username is wrong – Error message: Invalid username
3. Code is wrong – Error message: Invalid code

4. Code is Correct – Direct to reset password page

5.4. Reset password page

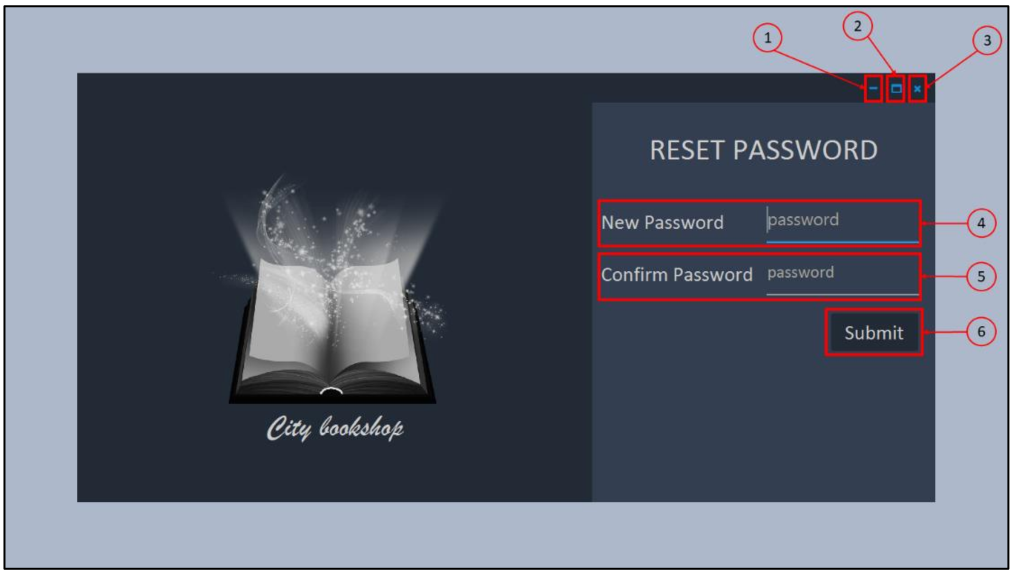


Figure 63 Reset password page

The motive of reset password page is to set new password for user account which verified from forget password page. User need to enter a password in the new password field and also, they need to enter same password in confirm password. This reduces the mistype passwords. Finally, when user click the submit button then it set the new password to that user account and direct them to dashboard page

Components of reset password page

1. Minimize – Minimize the application
2. Maximize – Maximize the application
3. Close – Close the application
4. New password – Enter your new password
5. Confirm password – Re-enter your new password
6. Submit – Click submit. It will set the new password and direct you to the dashboard window

Events of reset password page

1. New password field or confirm password is empty - Error message: Empty fields detected
2. New password and confirm password are matched – Set new password and direct to dashboard

5.5. Dashboard page

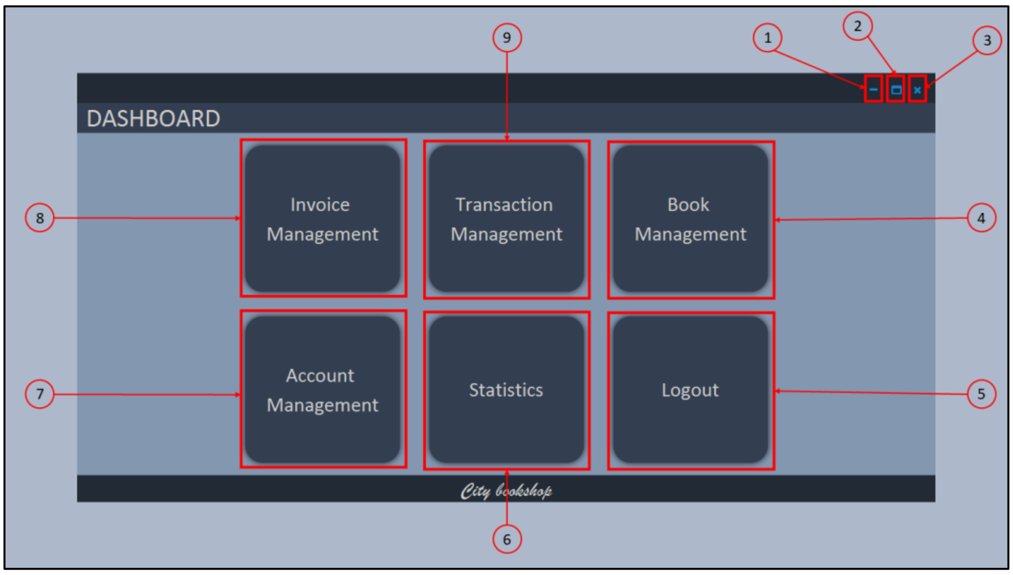


Figure 64 Dashboard page

The motive of dashboard page is to direct the user to their destination by displaying all the available tasks

Components of reset password page

1. Minimize – Minimize the application
2. Maximize – Maximize the application
3. Close – Close the application
4. Book management – direct the user to book management page
5. Logout – direct the user to login page
6. Statistics – direct the user to statistics page
7. Account management – Direct the user to account management page
8. Invoice management – Direct the user to invoice management page
9. Transaction management – Direct the user to transaction management page

Event of reset password page

1. Direct the user to their destination

5.6. Invoice management page

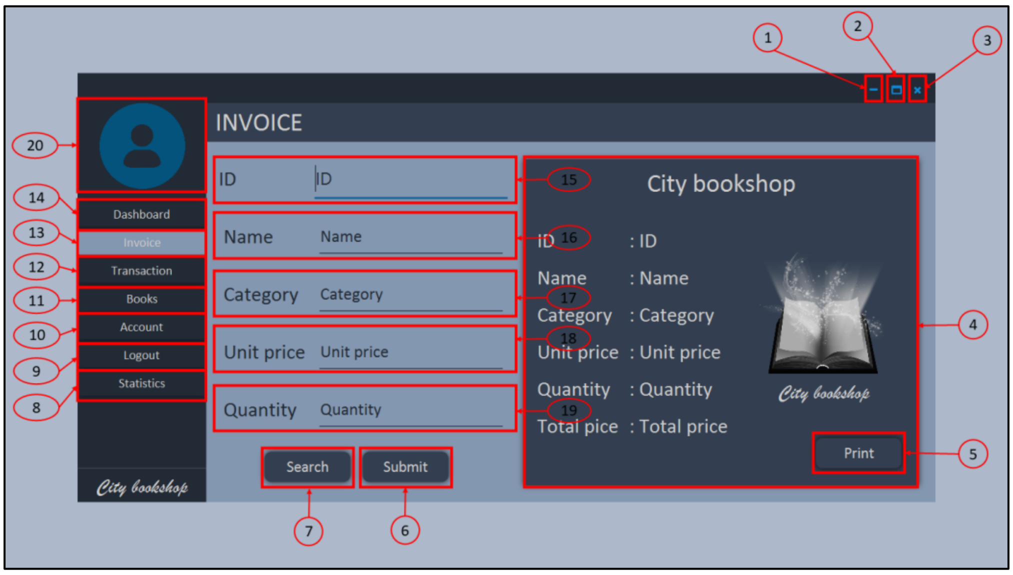


Figure 65 Invoice management page

The motive of the page is creating invoices. User need to give ID and quantity themselves. The book details can get by give a name of the book in the book field and click search button. It automatically fills the category and unit price; user don't need to enter full name of the book in book field; they can get the book name by enter the book's starting word and click search button. And also, there is side panel in left for navigation to other pages as in dashboard page

Components of invoice management page

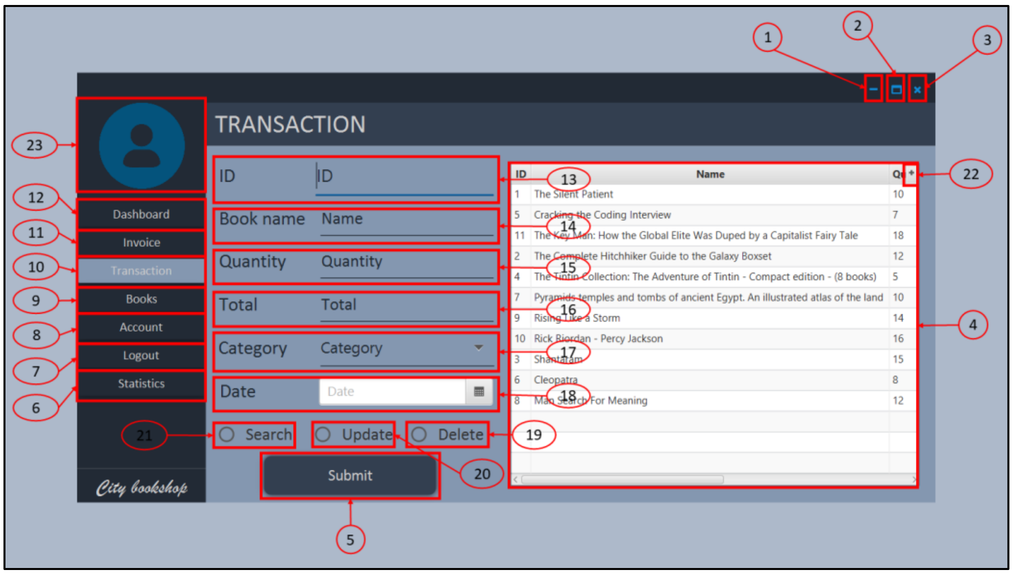
1. Minimize – Minimize the application
2. Maximize – Maximize the application
3. Close – Close the application
4. Invoice sheet – Printable area for invoice receipt
5. Print – Print the invoice receipt
6. Submit – Create invoice and prepare invoice receipt
7. Search – Search a book
8. Statistics – direct the user to statistics page
9. Logout – direct the user to login page

10. Account management – Direct the user to account management page
11. Book management – direct the user to book management page
12. Transaction management – Direct the user to transaction management page
13. Invoice management – Direct the user to invoice management page
14. Dashboard – Direct the user to dashboard page
15. ID – Enter Invoice ID
16. Name – Enter book's name
17. Category – Enter book's category
18. Unit price – Enter book's unit price
19. Quantity – Enter quantity of books
20. User picture – Displays user's picture

Events of invoice management page

1. Direct the user to their destination
2. Search book
3. Create invoice
4. Prepare invoice receipt
5. Print invoice receipt
6. Display user's picture

5.7. Transaction management page



ID	Name	Qt
1	The Silent Patient	10
5	Cracking the Coding Interview	7
11	The Key Man: How the Global Elite Was Duped by a Capitalist Fairy Tale	18
2	The Complete Hitchhiker Guide to the Galaxy Boxset	12
4	The Tintin Collection: The Adventure of Tintin - Compact edition - (8 books)	5
7	Pyramids temples and tombs of ancient Egypt. An illustrated atlas of the land	10
9	Rising Like a Storm	14
10	Rick Biordan - Percy Jackson	16
3	Shantaram	15
6	Cleopatra	8
8	Man Search For Meaning	12

Figure 66 Transaction management page

The motive of transaction management page is to monitoring and alternate the invoice details. When user arrive to the transaction management page initially all invoice details will appear in the table. When user select a row then that row's consecutive details automatically set to their text fields. Cashier can search an invoice. But admin can update, delete and search an invoice. And also, there is side panel in left for navigation to other pages as in dashboard page

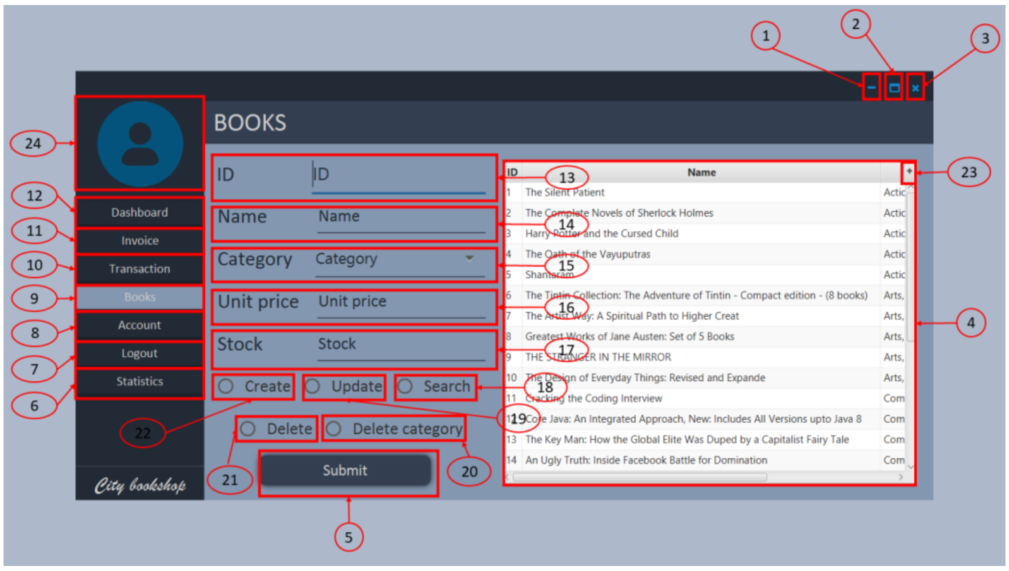
Components of transaction management page

1. Minimize – Minimize the application
2. Maximize – Maximize the application
3. Close – Close the application
4. Invoice table – Show all invoice details
5. Submit – Execute the task
6. Statistics – direct the user to statistics page
7. Logout – direct the user to login page
8. Account management – Direct the user to account management page
9. Book management – direct the user to book management page
10. Transaction management – Direct the user to transaction management page
11. Invoice management – Direct the user to invoice management page
12. Dashboard – Direct the user to dashboard page
13. ID – Enter Invoice ID
14. Book name – Enter book's name
15. Quantity – Enter quantity of books
16. Total – Total price of payment
17. Category – Enter book's category
18. Date – Invoice created date
19. Delete – Option for delete an invoice
20. Update – Option for update an invoice
21. Search – option for search an invoice
22. Table optional button – Customize the displaying table
23. User picture – Displays user's picture

Events of transaction management page

1. View all transaction
2. Search an invoice
3. Delete an invoice
4. Update an invoice
5. Customize table view details
6. Direct the user to their destination
7. Display user’s picture

5.8. Book management page



ID	Name	
1	The Silent Patient	Actic
2	The Complete Novels of Sherlock Holmes	Actic
3	Harry Potter and the Cursed Child	Actic
4	The Oath of the Vayuputras	Actic
5	Shantaram	Actic
6	The Tintin Collection: The Adventure of Tintin - Compact edition - (8 books)	Arts,
7	The Artist Way: A Spiritual Path to Higher Creat	Arts,
8	Greatest Works of Jane Austen: Set of 5 Books	Arts,
9	THE STRANGER IN THE MIRROR	Arts,
10	The Design of Everyday Things: Revised and Expande	Arts,
11	Cracking the Coding Interview	Com
12	Core Java: An Integrated Approach, New: Includes All Versions upto Java 8	Com
13	The Key Man: How the Global Elite Was Duped by a Capitalist Fairy Tale	Com
14	An Ugly Truth: Inside Facebook Battle for Domination	Com

Figure 67 Book management page

The motive of book management page is to monitoring, create and alternate the book details. When user arrive to the book management page initially all book details will appear in the table. When user select a row then that row’s consecutive details automatically set to their text fields. Cashier can search, create and update the books. But admin can delete the category, delete the books, search, create and update the books. And also, there is side panel in left for navigation to other pages as in dashboard page

Components of book management page

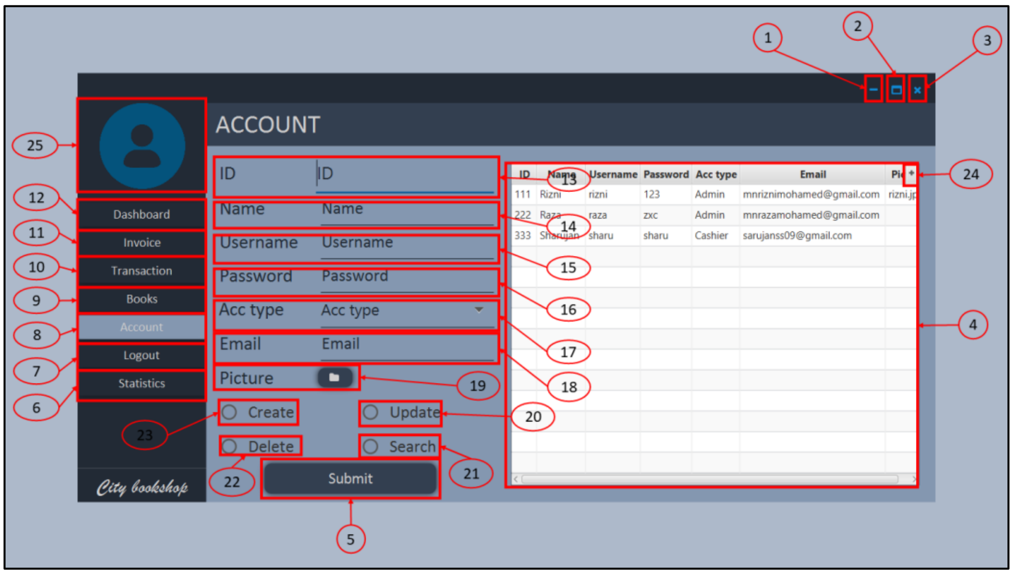
1. Minimize – Minimize the application
2. Maximize – Maximize the application
3. Close – Close the application
4. book table – Show all book details
5. Submit – Execute the task
6. Statistics – direct the user to statistics page
7. Logout – direct the user to login page
8. Account management – Direct the user to account management page
9. Book management – direct the user to book management page
10. Transaction management – Direct the user to transaction management page
11. Invoice management – Direct the user to invoice management page
12. Dashboard – Direct the user to dashboard page
13. ID – Enter book ID
14. Book name – Enter book’s name
15. Category – Enter book’s category
16. Unit price – Enter unit price of the book
17. Stock – Enter books stock count
18. Search – option for search a book
19. Update – Option for update a book
20. Delete category – Option for delete book category
21. Delete – Option for delete a book
22. Create – option for create a book
23. Table optional button – Customize the displaying table
24. User picture – Displays user’s picture

Events of book management page

1. View all books
2. Search a book
3. Delete a book
4. Delete a book category
5. Create a book

6. Update a book
7. Customize table view details
8. Direct the user to their destination
9. Display user's picture

5.9. Account management page



ID	Name	Username	Password	Acc type	Email	Pi
111	Rizni	rizni	123	Admin	mnriznimohamed@gmail.com	rizni.jp
222	Raza	raza	zxc	Admin	mnrazamohamed@gmail.com	
333	Sharujan	sharu	sharu	Cashier	sarujanss09@gmail.com	

Figure 68 Account management page

The motive of account management page is to monitoring, create and alternate the accounts. When cashier arrive to the account management page initially. their account details will appear in the table. But when admin arrives, initially all account details will appear in the table. When user select a row then that row's consecutive details automatically set to their text fields. Cashier can only update their account. But admin can delete, update, search and create any account. And also, there is side panel in left for navigation to other pages as in dashboard page

Components of account management page

1. Minimize – Minimize the application
2. Maximize – Maximize the application
3. Close – Close the application
4. Account table – Show all account details
5. Submit – Execute the task

6. Statistics – direct the user to statistics page
7. Logout – direct the user to login page
8. Account management – Direct the user to account management page
9. Book management – direct the user to book management page
10. Transaction management – Direct the user to transaction management page
11. Invoice management – Direct the user to invoice management page
12. Dashboard – Direct the user to dashboard page
13. ID – Enter account ID
14. Account name – Enter account name
15. Username – Enter username for their account
16. Password – Enter password for their account
17. Email – Enter email address for forget password process
18. Picture – Select picture for account picture
19. Update – Option for update an account
20. Search – option for search an account
21. Delete – Option for delete an account
22. Create – option for create an account
23. Table optional button – Customize the displaying table
24. User picture – Displays user’s picture

Events of account management page

1. View account/s
2. Search an account
3. Delete an account
4. Create an account
5. Update an account
6. Customize table view details
7. Direct the user to their destination
8. Display user’s picture

5.10. Statistics page

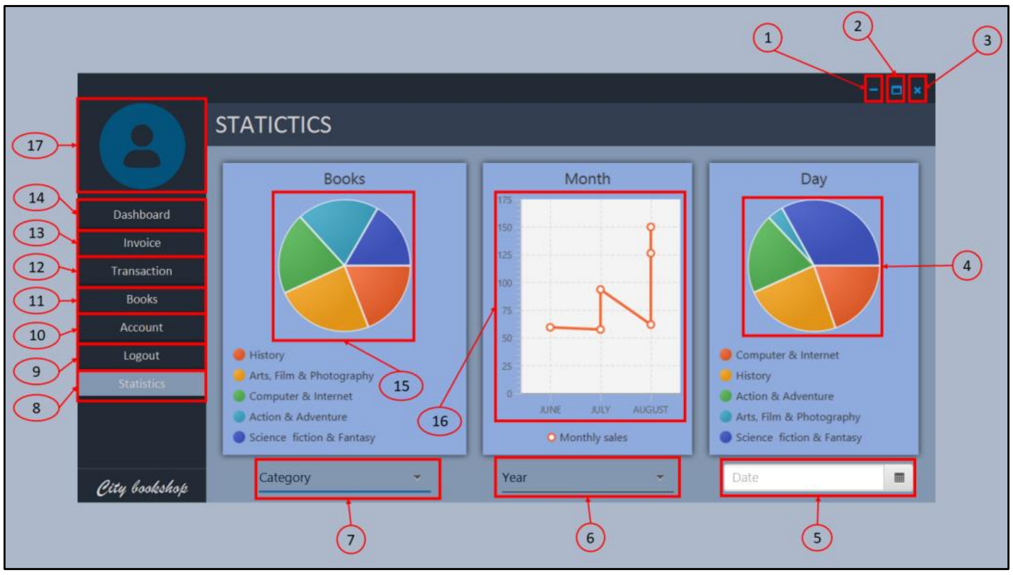


Figure 69 Statistics page

The motive of statistics page is to monitoring book inventory and sales progress. There are three sections inside this page. Those are book inventory unit which displays in pic chart, Monthly sales of year which displays in a line a chart and daily sales which displays in a pie chart, when user arrives, initially, the available number of total books in each category will display in the book's pie chart; the number total sales in each month of current year will display in the month line chart; the number of sales going on a day in each book category will display in a day's pie chart. By selecting categories under the book's pie chart user can view available books and its quantity of a category. By selecting year under the month's line chart user can view total sales of each month of particular year. By pic a date under day's pie chart user can view what are the books sold in that particular day with total prices. There is side panel in left for navigation to other pages as in dashboard page

Components of statistics page

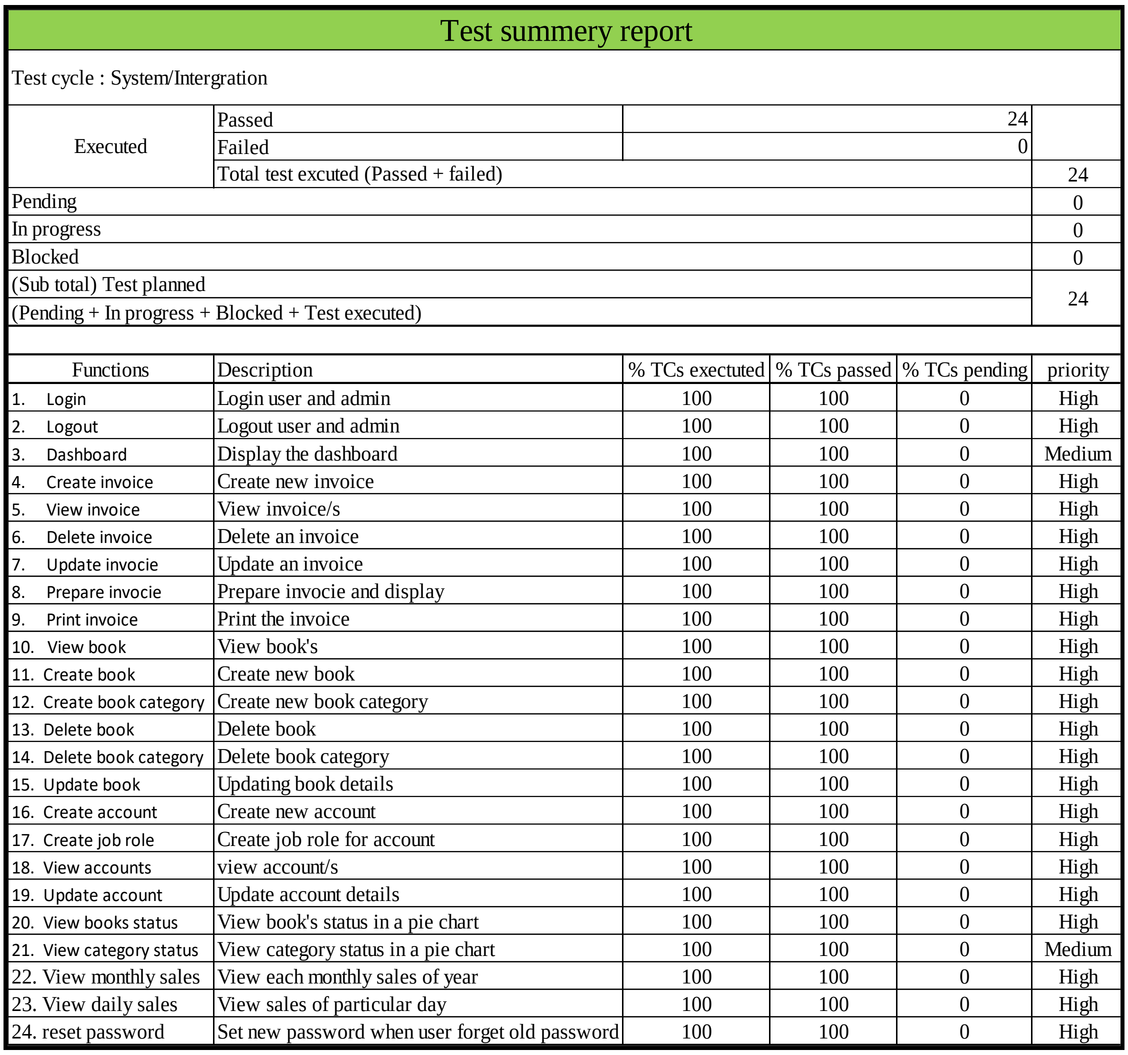
1. Minimize – Minimize the application
2. Maximize – Maximize the application
3. Close – Close the application
4. Day pie chart – Click to refresh the data
5. Date picker – Pic a particular date for view sales of particular day
6. Year – Select a year for view sales of each month in a particular year

7. Category – Select a category for view books availability of particular category
8. Statistics – direct the user to statistics page
9. Logout – direct the user to login page
10. Account management – Direct the user to account management page
11. Book management – direct the user to book management page
12. Transaction management – Direct the user to transaction management page
13. Invoice management – Direct the user to invoice management page
14. Dashboard – Direct the user to dashboard page
15. Book pie chart – Click to refresh the data
16. Month line chart – Click to refresh the data
17. User picture – Displays user’s picture

Events of statistics page

1. View available quantities of each category
2. View available quantities of each book
3. View monthly sales of each month
4. View daily sales
5. Get most profitable month of a year
6. Get most profitable book
7. Direct the user to their destination
8. Display user’s picture

6.Test summery report



Test summery report			
Test cycle : System/Intergration			
Executed	Passed	24	
	Failed	0	
	Total test excuted (Passed + failed)		24
Pending			0
In progress			0
Blocked			0
(Sub total) Test planned (Pending + In progress + Blocked + Test executed)			24

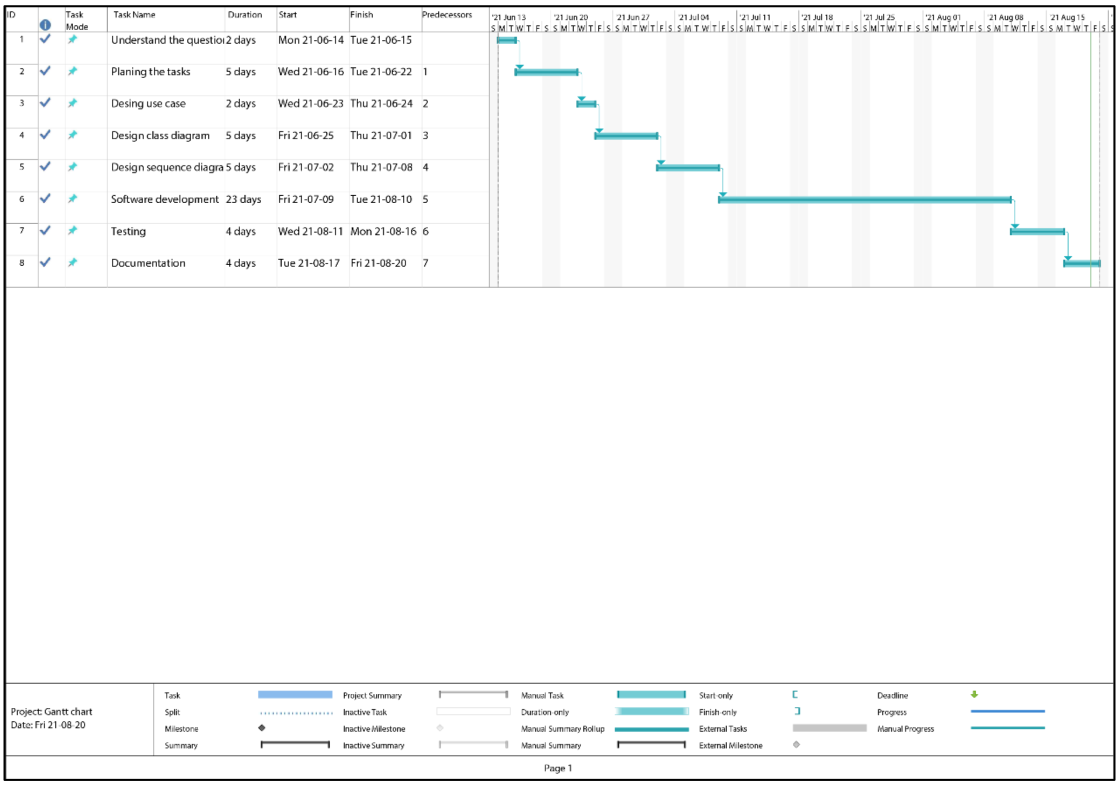
Functions	Description	% TCs exectuted	% TCs passed	% TCs pending	priority
1. Login	Login user and admin	100	100	0	High
2. Logout	Logout user and admin	100	100	0	High
3. Dashboard	Display the dashboard	100	100	0	Medium
4. Create invoice	Create new invoice	100	100	0	High
5. View invoice	View invoice/s	100	100	0	High
6. Delete invoice	Delete an invoice	100	100	0	High
7. Update invocie	Update an invoice	100	100	0	High
8. Prepare invocie	Prepare invocie and display	100	100	0	High
9. Print invoice	Print the invoice	100	100	0	High
10. View book	View book's	100	100	0	High
11. Create book	Create new book	100	100	0	High
12. Create book category	Create new book category	100	100	0	High
13. Delete book	Delete book	100	100	0	High
14. Delete book category	Delete book category	100	100	0	High
15. Update book	Updating book details	100	100	0	High
16. Create account	Create new account	100	100	0	High
17. Create job role	Create job role for account	100	100	0	High
18. View accounts	view account/s	100	100	0	High
19. Update account	Update account details	100	100	0	High
20. View books status	View book's status in a pie chart	100	100	0	High
21. View category status	View category status in a pie chart	100	100	0	Medium
22. View monthly sales	View each monthly sales of year	100	100	0	High
23. View daily sales	View sales of particular day	100	100	0	High
24. reset password	Set new password when user forget old password	100	100	0	High

Figure 70 Test summery report

7. Future recommendation

- In statistics page, highlight the sales made on day in the date picker of daily sales
- Make compatible continue compatible in maximize window
- Create new job roles with new permission areas
- Send verification codes to mobile phones
- Convert this application to cloud based system for remote access
- Enhance print view
- Convert possible validation process to database by using CHECK statement
- Make inter modularity between all packages for create security, privacy and reduce repeated codes

Gantt chart

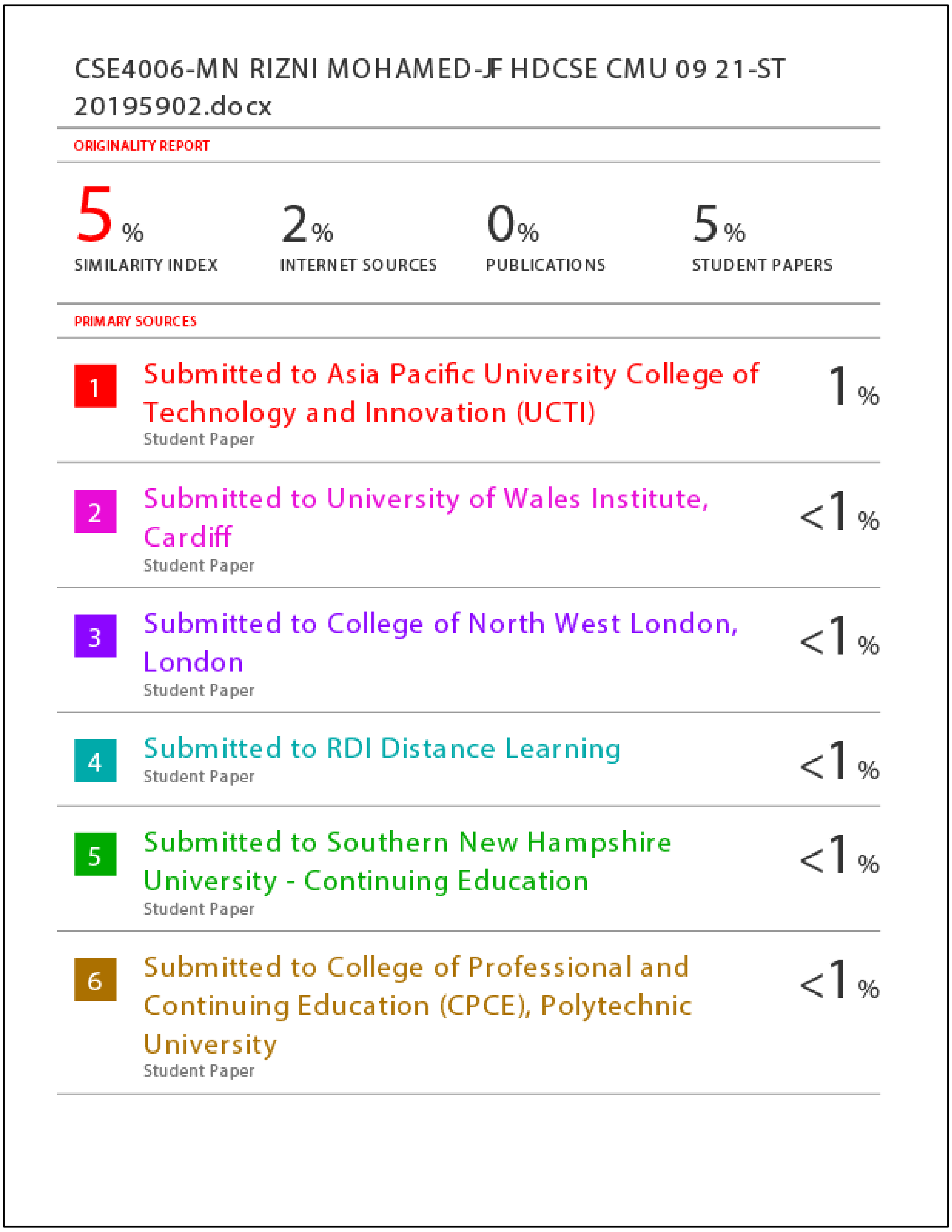


ID	ⓘ	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1	✓	📌	Understand the questio	2 days	Mon 21-06-14	Tue 21-06-15	
2	✓	📌	Planing the tasks	5 days	Wed 21-06-16	Tue 21-06-22	1
3	✓	📌	Desing use case	2 days	Wed 21-06-23	Thu 21-06-24	2
4	✓	📌	Design class diagram	5 days	Fri 21-06-25	Thu 21-07-01	3
5	✓	📌	Design sequence diagra	5 days	Fri 21-07-02	Thu 21-07-08	4
6	✓	📌	Software development	23 days	Fri 21-07-09	Tue 21-08-10	5
7	✓	📌	Testing	4 days	Wed 21-08-11	Mon 21-08-16	6
8	✓	📌	Documentation	4 days	Tue 21-08-17	Fri 21-08-20	7

Project: Gantt chart
Date: Fri 21-08-20

Figure 71 Gantt chart

Plagiarism report



CSE4006-MN RIZNI MOHAMED-JF HDCSE CMU 09 21-ST 20195902.docx

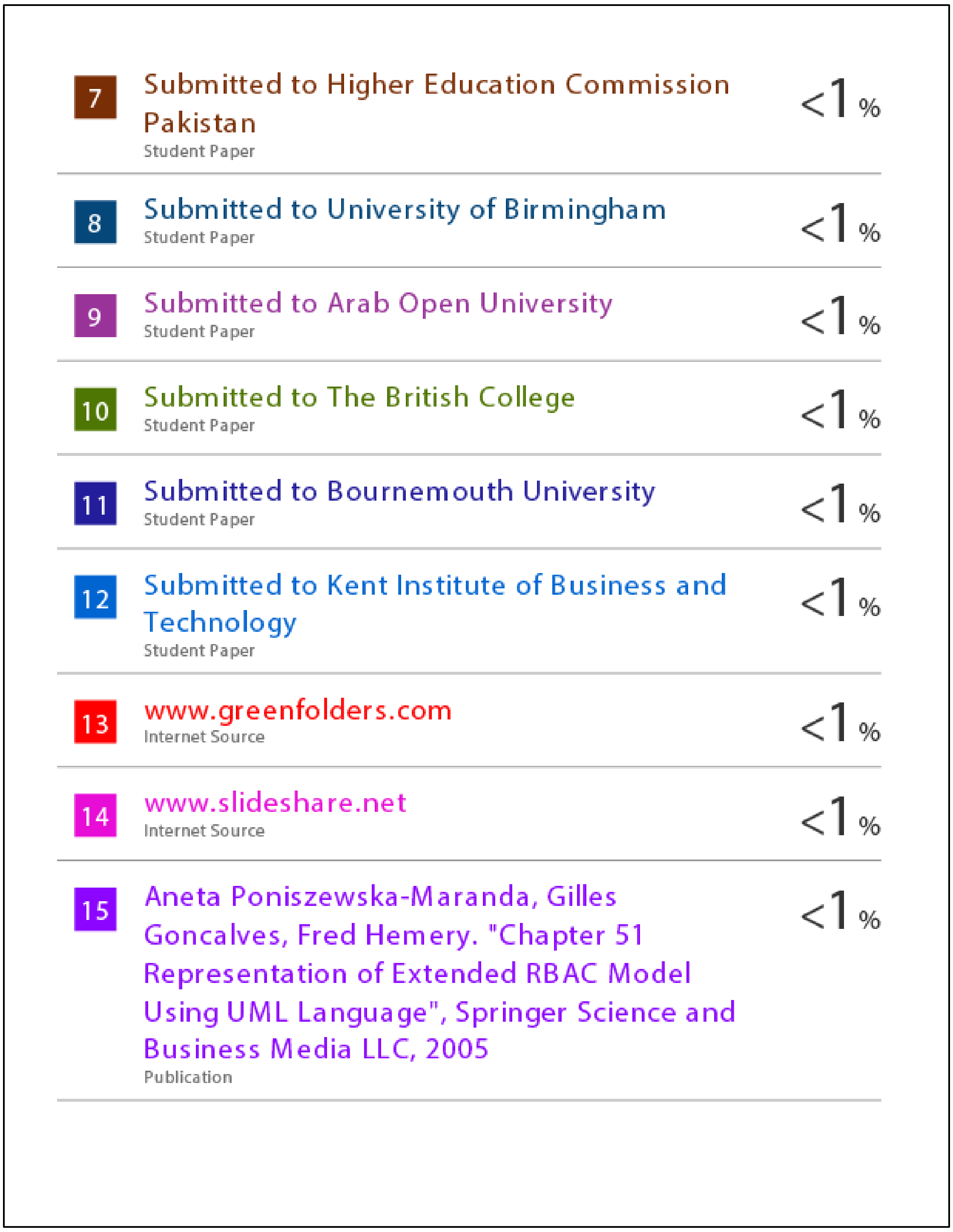
ORIGINALITY REPORT

5%	2%	0%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

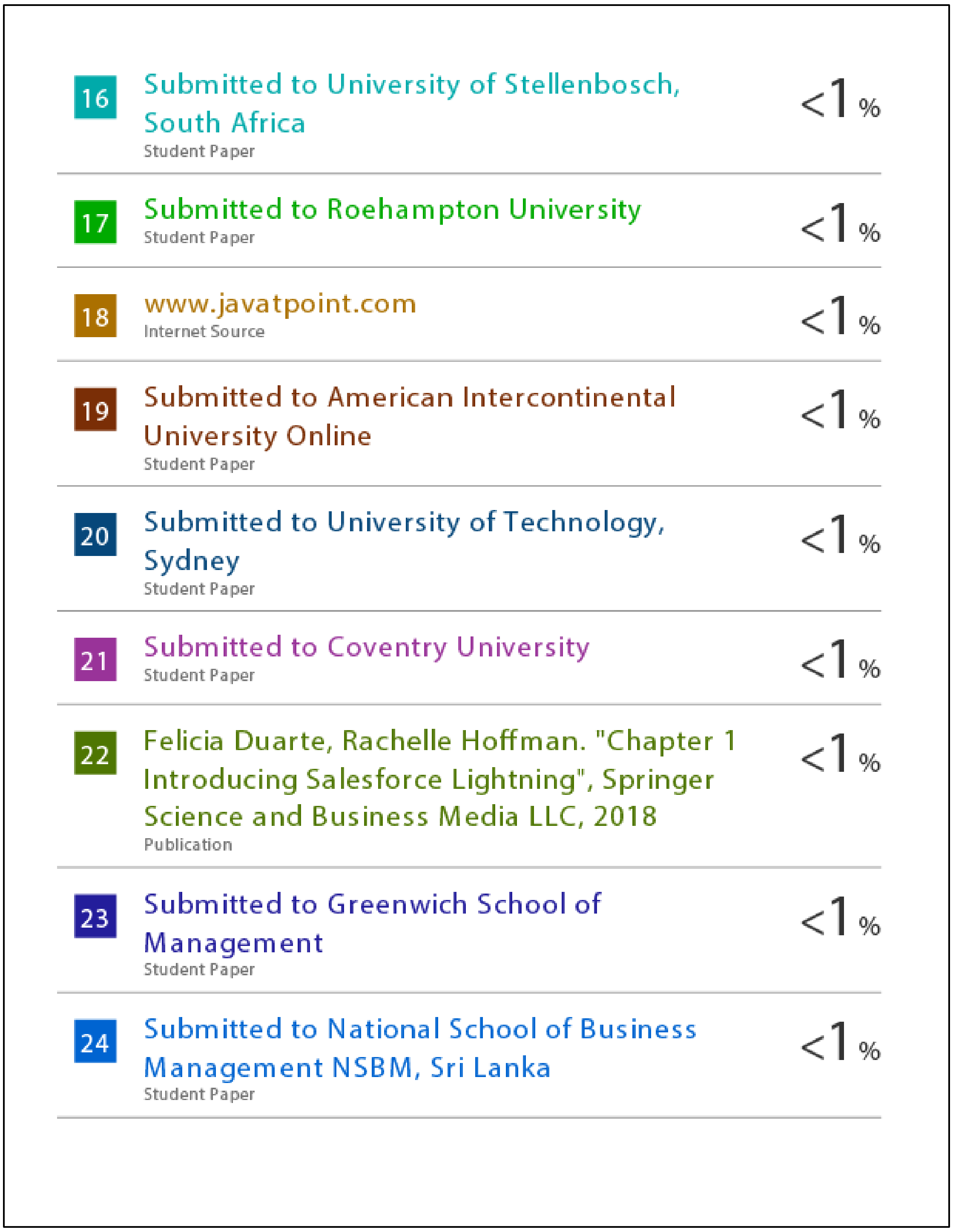
1	Submitted to Asia Pacific University College of Technology and Innovation (UCTI) Student Paper	1%
2	Submitted to University of Wales Institute, Cardiff Student Paper	<1%
3	Submitted to College of North West London, London Student Paper	<1%
4	Submitted to RDI Distance Learning Student Paper	<1%
5	Submitted to Southern New Hampshire University - Continuing Education Student Paper	<1%
6	Submitted to College of Professional and Continuing Education (CPCE), Polytechnic University Student Paper	<1%

Figure 72 Plagiarism report - page 1



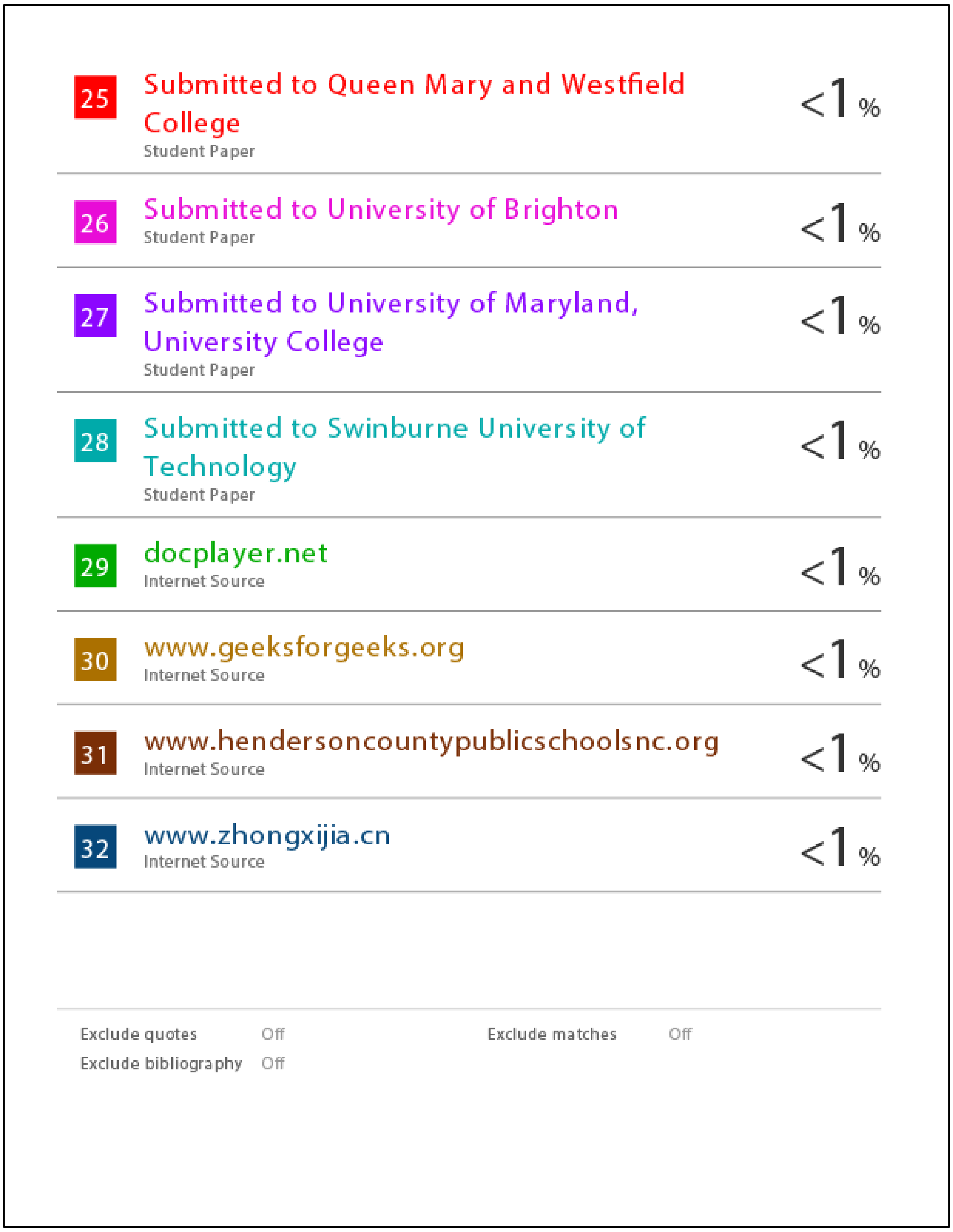
7	Submitted to Higher Education Commission Pakistan Student Paper	<1 %
8	Submitted to University of Birmingham Student Paper	<1 %
9	Submitted to Arab Open University Student Paper	<1 %
10	Submitted to The British College Student Paper	<1 %
11	Submitted to Bournemouth University Student Paper	<1 %
12	Submitted to Kent Institute of Business and Technology Student Paper	<1 %
13	www.greenfolders.com Internet Source	<1 %
14	www.slideshare.net Internet Source	<1 %
15	Aneta Poniszewska-Maranda, Gilles Goncalves, Fred Hemery. "Chapter 51 Representation of Extended RBAC Model Using UML Language", Springer Science and Business Media LLC, 2005 Publication	<1 %

Figure 73 Plagiarism report - page 2



16	Submitted to University of Stellenbosch, South Africa Student Paper	<1 %
17	Submitted to Roehampton University Student Paper	<1 %
18	www.javatpoint.com Internet Source	<1 %
19	Submitted to American Intercontinental University Online Student Paper	<1 %
20	Submitted to University of Technology, Sydney Student Paper	<1 %
21	Submitted to Coventry University Student Paper	<1 %
22	Felicia Duarte, Rachelle Hoffman. "Chapter 1 Introducing Salesforce Lightning", Springer Science and Business Media LLC, 2018 Publication	<1 %
23	Submitted to Greenwich School of Management Student Paper	<1 %
24	Submitted to National School of Business Management NSBM, Sri Lanka Student Paper	<1 %

Figure 74 Plagiarism report - page 3



25	Submitted to Queen Mary and Westfield College Student Paper	<1 %
26	Submitted to University of Brighton Student Paper	<1 %
27	Submitted to University of Maryland, University College Student Paper	<1 %
28	Submitted to Swinburne University of Technology Student Paper	<1 %
29	docplayer.net Internet Source	<1 %
30	www.geeksforgeeks.org Internet Source	<1 %
31	www.hendersoncountypublicschoolsnc.org Internet Source	<1 %
32	www.zhongxijia.cn Internet Source	<1 %

Exclude quotes Off
Exclude bibliography Off
Exclude matches Off

Figure 75 Plagiarism report - page 4

Reference

- Tutorialspoint.com. 2021. SDLC - Overview. [online] Available at: <https://www.tutorialspoint.com/sdlc/sdlc_overview.htm> [Accessed 17 August 2021].
- www.javatpoint.com. 2021. UML Use Case Diagram - Javatpoint. [online] Available at: <https://www.javatpoint.com/uml-use-case-diagram> [Accessed 17 August 2021].
- GeeksforGeeks. 2021. Unified Modeling Language (UML) | Sequence Diagrams - GeeksforGeeks. [online] Available at: <https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/> [Accessed 17 August 2021].
- Tutorialspoint.com. 2021. SDLC - Overview. [online] Available at: <https://www.tutorialspoint.com/sdlc/sdlc_overview.htm> [Accessed 18 August 2021].
- GeeksforGeeks. 2021. Overriding in Java - GeeksforGeeks. [online] Available at: <https://www.geeksforgeeks.org/overriding-in-java/> [Accessed 18 August 2021].
- GreatLearning Blog: Free Resources what Matters to shape your Career!. 2021. Polymorphism in Java and Java Polymorphism Examples. [online] Available at: <https://www.mygreatlearning.com/blog/polymorphism-in-java/> [Accessed 18 August 2021].
- Tutorialspoint.com. 2021. Java - Encapsulation. [online] Available at: <https://www.tutorialspoint.com/java/java_encapsulation.htm> [Accessed 18 August 2021].
- Tutorialspoint.com. 2021. SDLC - Overview. [online] Available at: <https://www.tutorialspoint.com/sdlc/sdlc_overview.htm> [Accessed 18 August 2021].
- W3schools.com. 2021. Java Abstraction. [online] Available at: <https://www.w3schools.com/java/java_abstract.asp> [Accessed 18 August 2021].
- www.javatpoint.com. 2021. Inheritance in Java - Javatpoint. [online] Available at: <https://www.javatpoint.com/inheritance-in-java> [Accessed 18 August 2021].