Toronto Metropolitan University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	COE528: Object Oriented Eng Analysis and Design
Course Number:	COE528
Semester/Year (e.g.F2016)	W2025

Instructor:	Boujemaa Guermazi

Assignment/Lab Number:	PROJECT FINAL REPORT
Assignment/Lab Title:	PROJECT FINAL REPORT

Submission Date:	03/30/2025
Due Date:	03/30/2025

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
BILLOO	MUHAMMAD MUBASHIR	500808243	12	JHAMMAD MUBASHIR
Hamza	Dorothy	501153244	12	D.H.
Ramesh	Artthi	501160484	12	R.Artthi

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: https://www.torontomu.ca/content/dam/senate/policies/pol60.pdf

Objective

The main goal of this project was to implement the knowledge and skills acquired throughout the last few weeks in class towards the analysis, design and application of a bookstore application. The initial first two phases of this project are inclined towards the design and implementation of UML diagrams and class diagrams for different parts of this application. Which is then followed by the implementation of the state design pattern in the software design. And finally, the overall system of bookstore application was implemented using NetBeans IDE.

PHASE I – USE CASE DIAGRAM:

Step 1: Identify the Actor

- **Customer**
  The Customer is the primary actor for these use cases. The Customer logs in, views books, purchases books, redeems points, and logs out.

Step 2: Identify the High-Level Use Cases

For the Customer, the main use cases are:

- **Login**
  The Customer logs in using their username and password.

- **View Books**
  This use case allows the Customer to browse available books.

- **Purchase Books**
  This use case allows the Customer to buy books at full price.

- **Redeem Points**
  This use case allows the Customer to use reward points to get a discount on purchases.

- **Logout**
  The Customer logs out of the application.

Step 3: Break Down the "View Books" Use Case

Within the **View Books** use case, the following actions occur:

- **Retrieve Book List**
  The system retrieves the list of books from the stored data.

- **Display Book List**
  The system displays a table containing:

    - **Book Name**
    - **Book Author**

    - **Book Price**

    - **Selection Checkbox**
      For marking books to purchase later.

- **Handle Empty Book List**
  If no books are available, the system displays the message:

    - **“No books available at the moment.”**

- **Handle Errors**
  If an error occurs while retrieving books, the system displays an error message.

- **Book Selection**
  The Customer can mark one or more books using the selection checkboxes.

- **Return to Customer Start Screen**
  If the Customer does not wish the purchase a book, they can return to the **Customer Start Screen** by clicking the [Back] button.

Step 4: Break Down the “Purchase Books” Use Case

Within the **Purchase Books** use case, the following actions occur:

- **Select Books**
  The Customer selects one or more books using the selection checkboxes.

- **Buy Button**
  The Customer clicks the [Buy] button to proceed with the purchase.

- **Calculate Total Cost**
  The system calculates the Total Cost:

    - **TotalCost = Sum of selected book prices**

- **Handle No Books Selected Error**
  If the Customer clicks [Buy] without selecting any books, the system displays:

    - **“Please select at least one book to purchase.”**

- **Process Payment**
  The system deducts the Total Cost from the Customer's payment.

- **Handle Insufficient Funds Error**
  If a payment system is implemented and the Customer does not have enough funds, the system displays:

  - **"Transaction failed: Insufficient funds."**

- **Update Rewards Points**
  The system updates the Customer's points:

  - **For every $1 spent, the Customer earns 10 points.**

- **Check Customer Status**
  The system updates the Customer's status:

  - **Silver "S" (if points <1000).**
  - **Gold "G" (if points >=1000).**

- **Display Customer Cost Screen**
  The system shows a screen displaying:

  - **Total Cost**
  - **Points Earned**
  - **Updated Points Total**
  - **Updated Customer Status**

- **Update Book List**
  The system removes the purchased book(s) from the store's list (assuming there is only one copy of each book).

- **Logout or Return to Customer Start Screen**
  The Customer can either log out or return to the Customer Start Screen.

Step 5: Break Down the "Redeem Points and Buy" Use Case

Within the **Redeem Points and Buy** use case, the following actions occur:

- **Point Redemption**
  The Customer checks the [Redeem Points] checkbox.

- **Calculate Total Cost (After Points Redemption)**
  The system applies the redemption rule:

    - **100 points = $1 discount**

    - **All available points are used to reduce Total Cost**

    - **The Total Cost must not go below $0.**

- **Handle Not Enough Points Error**
  If the Customer has fewer than 100 points, the system displays:

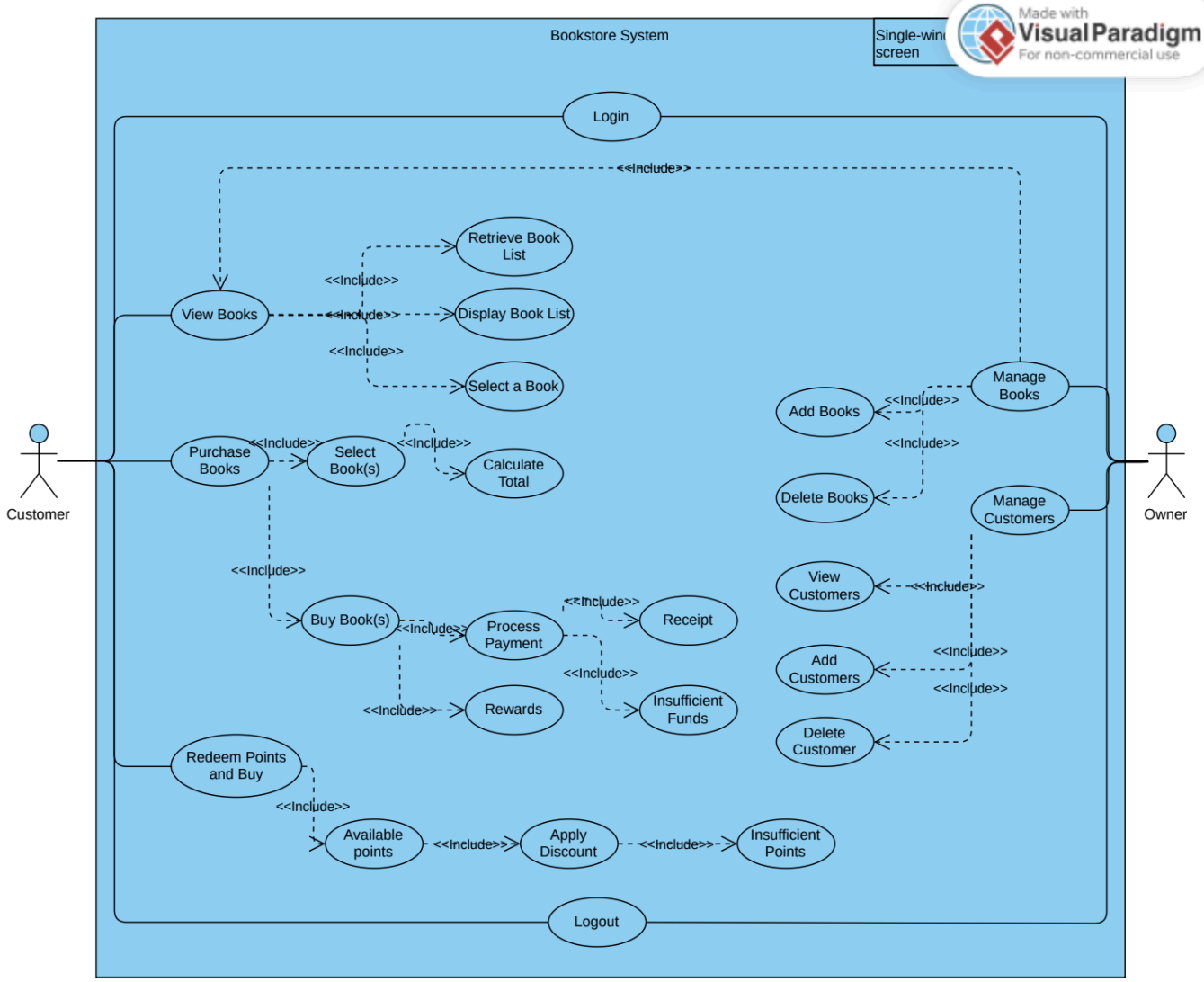
    - **“You need at least 100 points to redeem.”**

- **Process Payment after Discount**
  The system deducts the discounted Total Cost from the Customer’s payment.

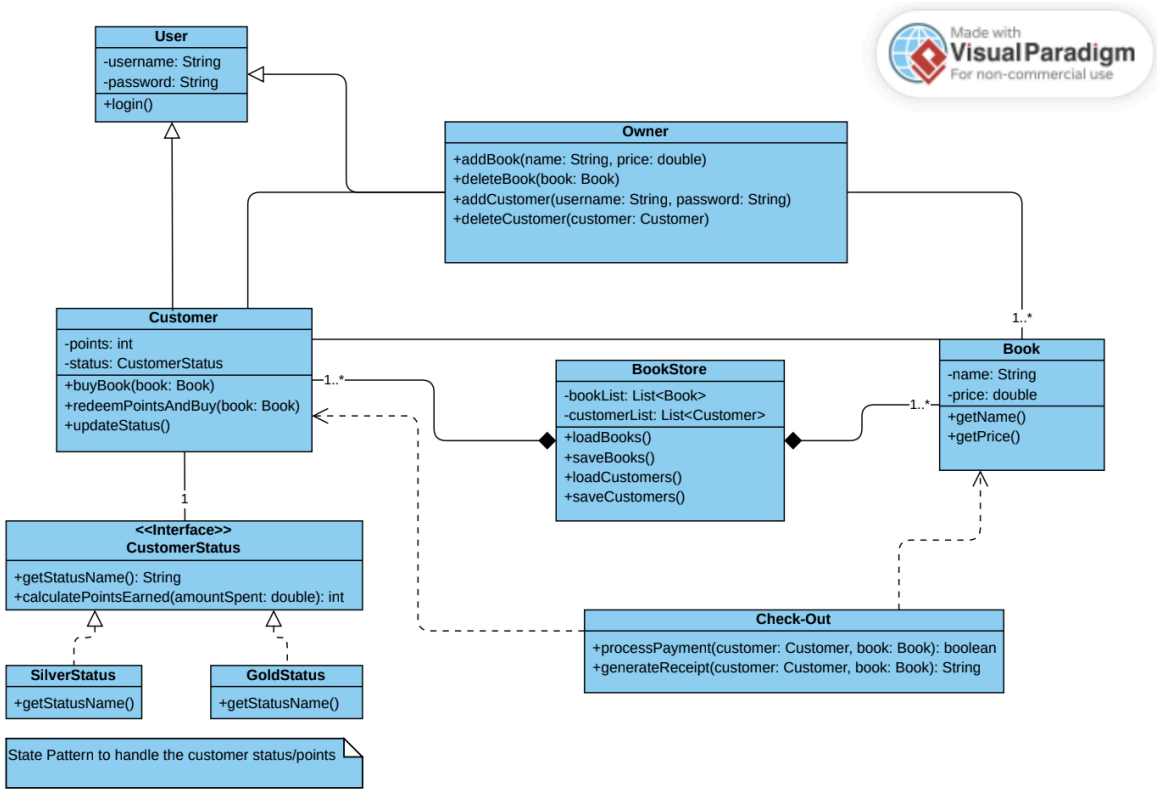
- **Update Remaining Points**
  The system updates the Customer’s remaining points after redemption.

- **Return to Check Customer Status**

Return to the **Check Customer Status** action in the **Purchase Books Use Case**.


Bookstore System
Single-win
screen
Login
<<Include>>
Retrieve Book List
<<Include>>
View Books
<<Include>>
Display Book List
<<Include>>
Select a Book
Customer
Purchase Books
<<Include>>
Select Book(s)
<<Include>>
Calculate Total
<<Include>>
Buy Book(s)
<<Include>>
Process Payment
<<Include>>
Receipt
<<Include>>
Rewards
<<Include>>
Insufficient Funds
Redeem Points and Buy
<<Include>>
Available points
<<Include>>
Apply Discount
<<Include>>
Insufficient Points
Logout
Manage Books
<<Include>>
Add Books
<<Include>>
Delete Books
Manage Customers
<<Include>>
View Customers
<<Include>>
Add Customers
<<Include>>
Delete Customer
Owner
Made with
Visual Paradigm
For non-commercial use

PHASE II – CLASS DIAGRAM:



FINAL IMPLEMENTATION – NETBEANS

MAIN PROGRAM

The main program's functionality essentially controls the overall bookstore application and GUI using JavaFX. It is responsible for major functions such as application startup, login logic, customer data storage and management and User Interface transitions for both the admin and customer.

```
private ArrayList<Customer> customers = new ArrayList<>();

private final String CUSTOMERS_FILE = "customers.txt";

@Override
public void start(Stage primaryStage) {
    this.primaryStage = primaryStage;

    // 1) Load customers from file
    loadCustomersFromFile();

    // Show the login scene
    showLoginScene();

    primaryStage.setTitle("Bookstore App");
    primaryStage.show();
}

@Override
public void stop() {
    storeCustomersToFile();
}
```

Primary Stage variable has been defined at the very beginning of the code, throughout this project the overall design has been done using JavaFX, the primary stage variable is a part of JavaFX. Next our code involves definition for an ArrayList for customers to store the data of customers in our customers.txt file. Upon the beginning we load the customer from the file to check if we have an existing customer in the file, moving forward we have design a login screen.

```
// ------------------ LOGIN SCENE ------------------
private void showLoginScene() {
    GridPane grid = new GridPane();
    grid.setAlignment(Pos.CENTER);
    grid.setHgap(10);
    grid.setVgap(10);
    grid.setPadding(new Insets(25));

    Label sceneTitle = new Label("Welcome to the BookStore App");
    grid.add(sceneTitle, 0, 0, 2, 1);

    Label userLabel = new Label("Username:");
    grid.add(userLabel, 0, 1);

    TextField userField = new TextField();
    grid.add(userField, 1, 1);

    Label passLabel = new Label("Password:");
    grid.add(passLabel, 0, 2);

    PasswordField passField = new PasswordField();
    grid.add(passField, 1, 2);
```

We have designed the GUI for login screen for admin and customers login and logout functionality, the backend functionality involves the taking in of logins including the username and password and its validation followed by the welcome screen.

```
Button loginBtn = new Button("Login");
grid.add(loginBtn, 1, 3);

Label messageLabel = new Label();
grid.add(messageLabel, 1, 4);

loginBtn.setOnAction(e -> {
    String username = userField.getText();
    String password = passField.getText();

    // Check if admin
    if (username.equals("admin") && password.equals("admin")) {
        showOwnerStartScene();
        return;
    }

    // Otherwise, check customers
    for (Customer c : customers) {
        if (c.getUsername().equals(username) && c.getPassword().equals(password)) {
            showCustomerStartScene(c);
            return;
        }
    }

    // No match
    messageLabel.setText("Invalid credentials. Try again!");
});
```

The login screen for admin has a validation implementation of username and password set as “admin” upon validation the user could essentially log in as a admin. In the other case, if the customer logs in using its credentials the credentials are validated and if validated the log in operation is completed if not, the programs shows a message of “Invalid Credentials, Try Again”


```
// ---------------- LOAD CUSTOMERS ----------------
private void loadCustomersFromFile() {
    File file = new File(CUSTOMERS_FILE);
    if (!file.exists()) {
        System.out.println("No existing customers.txt file found. Starting fresh.");
        return; // No file yet, so skip loading
    }

    try (BufferedReader br = new BufferedReader(new FileReader(file))) {
        String line;
        customers.clear(); // Clear any existing data
        while ((line = br.readLine()) != null) {
            // Each line: username:password:points
            String[] parts = line.split(":");
            if (parts.length == 3) {
                String username = parts[0];
                String password = parts[1];
                int points = Integer.parseInt(parts[2]);

                Customer c = new Customer(username, password);
                c.setPoints(points);
                customers.add(c);
            }
        }
        System.out.println("Customers loaded successfully from " + CUSTOMERS_FILE);
```

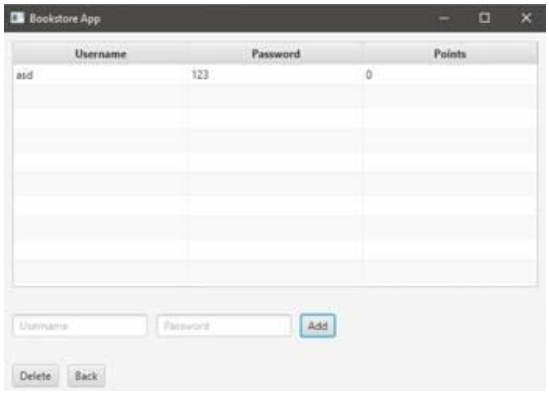
This part is essentially responsible to load the customer file, if the file is not found the message is shown as "No existing customers.txt file found. Starting fresh."

```
// ------------------ OWNER SCENE ------------------
private void showOwnerStartScene() {
    Label welcomeLabel = new Label("Welcome, Owner!");
    Button customersBtn = new Button("Customers");
    Button logoutBtn = new Button("Logout");

    customersBtn.setOnAction(e -> showManageCustomersScene());
    logoutBtn.setOnAction(e -> showLoginScene());

    VBox layout = new VBox(10, welcomeLabel, customersBtn, logoutBtn);
    layout.setPadding(new Insets(20));
    primaryStage.setScene(new Scene(layout, 400, 300));
}

// ------------------ MANAGE CUSTOMERS SCENE ------------------
private void showManageCustomersScene() {
TableView<Customer> table = new TableView<>();
table.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);

// Columns
TableColumn<Customer, String> colUsername = new TableColumn<>("Username");
colUsername.setCellValueFactory(new PropertyValueFactory<>("username"));
```

Then we have the user interface and GUI setup for the Owner Start Screen, this primarily consist of different buttons such as the Customers and Logout along with the Welcome, Owner message screen.

```
// ------------------ MANAGE CUSTOMERS SCENE ------------------
private void showManageCustomersScene() {
TableView<Customer> table = new TableView<>();
table.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);

// Columns
TableColumn<Customer, String> colUsername = new TableColumn<>("Username");
colUsername.setCellValueFactory(new PropertyValueFactory<>("username"));

TableColumn<Customer, String> colPassword = new TableColumn<>("Password");
colPassword.setCellValueFactory(new PropertyValueFactory<>("password"));

TableColumn<Customer, Integer> colPoints = new TableColumn<>("Points");
colPoints.setCellValueFactory(new PropertyValueFactory<>("points"));

table.getColumns().addAll(colUsername, colPassword, colPoints);

    ObservableList<Customer> data = FXCollections.observableArrayList(getCustomers());
table.setItems(data);

// Input fields
TextField userField = new TextField();
userField.setPromptText("Username");
TextField passField = new TextField();
passField.setPromptText("Password");
```

Then we have the manage customers screen designed. The backend of this part is essentially capable of storing and displaying the username and password of the customers along with the points this essentially shows us the customer database and gives the user the ability to add and delete customer credentials as needed.



CUSTOMER

```java
package coe528bookstore;

public class Customer {
    private String username;
    private String password;
    private int points;

    public Customer(String username, String password) {
        this.username = username;
        this.password = password;
        this.points = 0;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public int getPoints() {
        return points;
    }

    public void setPoints(int points) {
        this.points = points;
    }
}
```

The customer class has been designed in the bookstore system to store customer login information and points balance and to provide access to mutators and accessors for these fields from the main program. This class primarily functions as a simple Plain Old Java Object covering the user identity and points tracking system internal mechanism. The main use of this class is to store core data entities for customer-related applications in our program. In this class we have defined attributes such as username, password, points and we have defined the accessors such as getUsername, getPassword and getPoints. Our main constructor is public Customer(String username, String password) This initializes a new customer object with a username and password.

BOOKS

```java
 * @author artthiramesh
 */
import javafx.beans.property.BooleanProperty;
import javafx.beans.property.SimpleBooleanProperty;
import javafx.scene.control.CheckBox;

public class Books {


    private final double price;
    private final String title;
    private final String author;
    private BooleanProperty selected = new SimpleBooleanProperty(false);

    public Books(String bookTitle, String authorOfBook, double priceOfBooks){

        this.price = priceOfBooks;
        this.title = bookTitle;
        this.author = authorOfBook;
       //  this.selectedBooks = new CheckBox();
    }

    public String getTitle(){
        return title;
    }

    public String getAuthor(){
        return author;
    }

    public double getPrice(){
        return price;
    }

    public BooleanProperty selectedProperty(){
        return selected;
    }

    public boolean isSelectedBooks(){
        return selected.get();
    }

    public void setSelectedBooks(boolean selected){
        this.selected.set(selected);
    }

}
```

The book class represents the books available in the bookstore, which holds all the necessary information such as the title, author and price of the book. The class has a getter method for the book's details, along with getter and setter methods to help during checkout. This feature is possible with a Boolean property, which helps to see if the book has been selected or not.

CUSTOMER STATUS

```java
public interface CustomerStatus {
    String getStatusName();
    int calculatePointsEarned(double amountSpent);
}
```

```java
public class SilverStatus implements CustomerStatus {
    @Override
    public String getStatusName() {
        return "Silver";
}
  @Override
    public int calculatePointsEarned(double amountSpent) {
        return (int) (amountSpent * 10);  // Earn 10 points per $1 spent
    }
}
```

```java
public class GoldStatus implements CustomerStatus {
    @Override
    public String getStatusName() {
        return "Gold";
    }

    @Override
    public int calculatePointsEarned(double amountSpent) {
        return (int) (amountSpent * 15);  // Earn 15 points per $1 spent (bonus for Gold)
    }
}
```

The customer status uses a state design pattern which dynamically manages the customers status as either silver or gold. The customer status is an interface that defines the methods to be implemented by SilverStatus and GoldStatus classes. The SilverStatus and GoldStatus classes contain the logic to determine the points earned with the calculatePointsEarned() method.

Conclusion:

Based on our project implementation the design and implementation of our bookstore application with GUI has been successfully implemented using Netbeans IDE. In this project, we were able to successfully utilize the knowledge and skills we have gained during in class learning for the core concepts of Java from the basic fundamentals concepts to advanced java implementation along with the GUI design. In this project, initially we had defined the conceptual working mechanism of the bookstore application using a simplified UML diagram and class diagram and moving towards the final stage of implementation the back end programming and the front end user interface GUI was designed using Netbeans. Overall, we have successfully implemented a bookstore application that has the capability of logging in, logging out for the admin and customers, with backend username and password validation along with the functionality to add, delete and view customers from the database while keeping a record of the points as well. This project was very successful in terms of implementation and has given us the opportunity to confidently implement the basic to advanced level functionality within Java.