COE528 (W 2021)
Project

Duration: four weeks.

Objectives

- Analyze, design and implement a bookstore application.
- Design with UML diagrams.
- Use state design pattern in the software design.
- Implement the system.

This project **must** be done in **group**.

The project **must be demonstrated**. During demonstration, **each group member** will be asked questions by the TA **orally**, and **marked individually** for the **oral part**. Each member will be required to turn on video and show the student card at the start of question-answering session of the member.

Items to be delivered are as follows: (Deadlines are posted in D2L. Please refer to the section "Submitting your project" of this document on how to submit the deliverables)

- **Phase-1 submission (by 1st week):** Use Case Diagram in a file named useCaseDiagram.pdf.
- **Phase-2 submission (by 2nd week):** Class Diagram in a file named classDiagram.pdf.
- **Phase-Final submission (by 4th week) :** create a zip file named ***phaseFinal.zip***. This file should contain:
  - *useCaseDiagram.pdf* (updated if necessary)
  - *classDiagram.pdf* (updated if necessary). **Indicate the classes that are participating in the State Design Pattern.**
  - *report.pdf* (**describe** a use-case following the format discussed in class, **describe** the rationale behind using the State Design Pattern)
  - netbeans project folder (containing the implementation)

Problem Description

You will develop a simple Book Store Application (app). The app must be graphical user interface (GUI) based. The app should be a single-window GUI, that is, ONLY one window should be available to the user of the app (analogous to a web browser). This implies that if a user of the app clicks a button to obtain a new screen, the last screen of the app should be replaced by the new screen, in the SAME window (i.e. multiple windows MUST NOT get opened while using the app).

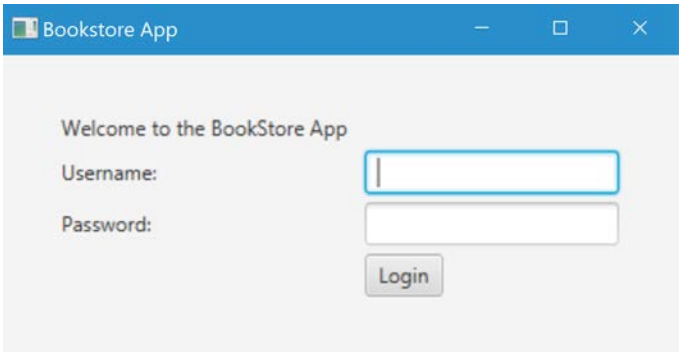
Whenever a user of the app clicks the [x] button (i.e. the exit button) at the top-right of the app window, all the relevant data currently present in the app should be written in two relevant files **books.txt** and **customers.txt**.

To keep the implementation simple, it is suggested that every time the [x] button is clicked, the old data in books.txt and customers.txt **should be overwritten** by the current data present in the relevant data structure(s) of the app. It is also suggested that every time the app is started, the data from the two files books.txt and customers.txt **should be loaded** in the data structure(s) of the app.

There are two kinds of users of this application: Owner and Customer. Here we will **assume** there is only one Owner and zero or more Customers who uses the application.

The app starts with a *login-screen*. The *login-screen* has three GUI items: a username field, a password field, and the button **[login]**. The owner's username is **admin** and password is **admin**.

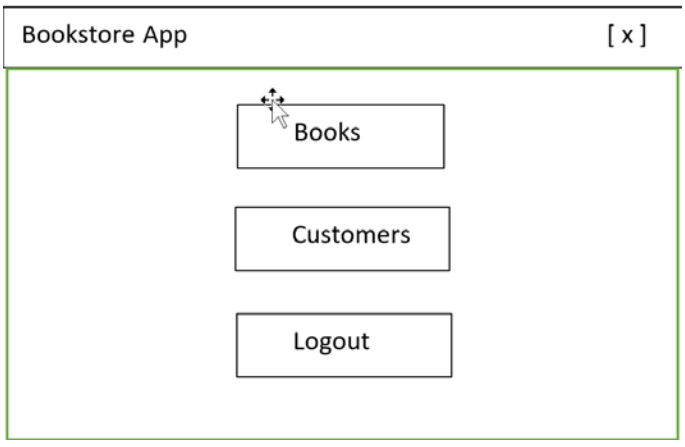
A sample login screen is:



How the owner will use the BookStore app:

When the owner enters her username and password, and clicks button **[login]**, a new screen replaces the *login-screen*. We call this new screen *owner-start-screen*. The *owner-start-screen* has three buttons: **[Books]**, **[Customers]**, **[Logout]**.

A sample *owner-start-screen* is:



- When the owner clicks the button **[Books]**, she sees a screen having three parts, a top part, a middle part, and a bottom part. We will refer to this screen as *owner-books-screen*. The top part of *owner-books-screen* contains a table. The table has two columns with heading **Book Name** and **Book Price**, from left to right. A row in the table contains the name of a book in the **Book Name** column, and its price in the **Book Price** column. It is **assumed** that the owner has only one copy of a book, and hence the information about a book in a row implies just one copy of that book. The middle part of *owner-books-screen* contains three GUI items: two textfields **Name** and **Price**; one button [**Add**]. When the owner enters a book's name and its price in the **Name** and the **Price** fields respectively, and then clicks the [**Add**] button, a new row containing the book's information gets entered in the table. The bottom part of the *owner-books-screen* contains a [**Delete**] button and a [**Back**] button. When the owner selects a row in the table, and then clicks the [**Delete**] button, the selected row should get deleted from the table and consequently the book information corresponding to the deleted row gets removed from the bookstore. If the owner clicks the [**Back**] button, she should be taken back to the *owner-start-screen*.

- When the owner clicks the button **[Customers]**, she sees a screen having three parts, a top part, a middle part, and a bottom part. We will refer to this screen as *owner-customers-screen*. The top part of *owner-customers-screen* contains a table. The table has three columns with heading **Username**, **Password, Points**, from left to right. They refer to a customer's username, password, and the points the customer has earned by buying books. The middle part of *owner-customers-screen* contains three GUI items: two textfields Username and Password; one button [**Add**]. The owner can register a customer to the bookstore as follows: When the owner enters a customer's username and password in the Username and the Password fields respectively, and then clicks the [**Add**] button, a new row containing the customer's information gets entered in the table. The **Points** for this newly added row should be **0**, that is, initially the customer has **0** points. The bottom part of the *owner-customers-screen* contains a [**Delete**] button and a [**Back**] button. When the owner selects a row in the table, and then clicks the [**Delete**] button, the selected row should get deleted from the table and consequently the customer information corresponding to the deleted row gets removed from the bookstore. If the owner clicks the [**Back**] button, she should be taken back to the *owner-start-screen*.

- When the owner clicks the button **[Logout]**, she should be taken back to the *login-screen*.

How a registered customer will use the BookStore app:

When a customer, say Jane (previously added by the owner) enters her username and password, and clicks button **[login]**, a new screen replaces the *login-screen*. We call this new screen *customer-start-screen*. The *customer-start-screen* has three parts, a top part, middle part, and a bottom part.

- The top part shows the message "Welcome Jane. You have **P** points. Your status is **S**". Here, P is the number of points Jane currently has. S is one of the two status, either **Gold** or **Silver,** that Jane has. A customer who has points less than 1000 has status **silver**. A customer who has points 1000 or above has status **gold**.

- The middle part of *customer-start-screen* contains a table. The table has three columns. From left to right, the first column has heading **Book Name**, the second column has heading **Book Price**, the third column has the heading **Select**. A row in the table contains the name of a book in the **Book Name** column, its price in the **Book Price** column, and a **checkbox** in the **Select** column. The **checkbox** can be either checked or unchecked.

- The bottom part of the *customer-start-screen* contains three buttons, [**Buy**], [**Redeem points and Buy**], [**Logout**]. If the customer clicks the [**Logout**] button, she should be taken back to the *login-screen*.

If the customer wants to buy one or more books, she should click the checkbox(es) present on the relevant row(s) of the book(s) in the *customer-start-screen*, and should click either the button [**Buy**], or [**Redeem points and Buy**].

When the customer clicks either the button [**Buy**], or [**Redeem points and Buy**], a new screen replaces the *customer-start-screen*. We call this new screen *customer-cost-screen*. This screen has three GUI items from top to bottom.

- The top item is the message **Total Cost: TC**. Here TC is the total transaction cost. It is **assumed** that the total cost of books is just the sum of the cost of individual books. There is no tax to be paid by the customer.

- The middle item is the message **Points: P, Status: S**. P is the current number of points of the customer. S is the current status of the customer.

- The bottom item is a [**Logout**] button. If the customer clicks the [**Logout**] button, she should be taken back to the *login-screen*.

For the case [**Redeem points and Buy**], the **TC** should be the transaction cost after the relevant amount of existing points of the customer has been redeemed. Here it is **assumed** that when possible, all the accumulated points must be redeemed. **Further assumption:** Number of points can be 0 or positive. It is assumed that for every 1 CAD spent by the customer, she earns 10 points. It is also assumed that for every 100 points redeemed by a customer, only 1 CAD is deducted from her transaction cost. The transaction cost **must not** be less than 0.

An example to explain accumulation and redemption of points, and update of customer's status is as follows:

If we assume that there are four books in the store whose prices are 50, 100, 200, 500. Let us assume that one customer has been added previously in the store by the owner.

- When the customer logs in for the first time, the points P should be **zero**, and the status S should be **Silver**. If the customer clicks button [**Buy**] for buying two books whose prices are 200, and 500 then TC should be 700, and customer's points P should become 7000 and status S should become **Gold**.

- In the next login session, if the same customer clicks [**Redeem points and Buy**] for buying the book with price 50, then TC should be 0, and customer's points P should become 2000. Customer's status S stays at **Gold.**

- In the next login session, if the same customer again clicks [**Redeem points and Buy**] to buy the book with price 100, then TC should be 80, and customer's points P should become 800. The customer's status should change to **Silver**.

NOTE:

- **The GUI of the application has to be developed using JavaFX. JavaFX Scene Builder may be used, if necessary.**
- **The tables can be created using the class TableView<S>. The API for TableView<S> can be found in the following webpage: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TableView.html**
- **Use Visual Paradigm to draw UML diagrams. Then copy the UML diagrams into a document (such as MS Word document) and convert the document into a pdf file.**

Submitting your project

Deadline: See announcement in D2L for deadline.

Phase-1 submission:

1. **Upload your *useCaseDiagram.pdf* file on D2L through the "Assessment" > "Assignments" > "Project-Phase-1"** dropbox**.**

Phase-2 submission:

1. **Upload your *classDiagram.pdf* file on D2L through the "Assessment" > "Assignments" > "Project-Phase-2"** dropbox**.**

Project-Phase-Final submission:

1. Create a folder. Name it as Group#_coe528_project. Example: If the group name is Group 15, the name of folder should be Group15_coe528_ project.

2. Copy your ***phaseFinal.zip*** file (See page-1 for the content of this zip file) in the folder Group#_coe528_project.

3. Copy the duly filled and signed cover page Standard Assignment/Lab Cover Page in the folder Group#_coe528_project.

4. Compress the folder Group#_coe528_project as a single zip file that is named according to the following rule: **Group#_coe528_project.zip. (**Example: Group15_coe528_project.zip.)

5. **Upload the file Group#_coe528_project.zip on D2L through the "Assessment" > "Assignments" > " Project-Phase-Final "** dropbox**.**

Within the deadline, you may re-submit (i.e. re-upload) the aforementioned zip file. However, note that your latest submission will always overwrite your previous submission.

Submission by email is NOT accepted.

NOTE: The implementation **must be done using Netbeans**. **If a group does not include Netbeans project folder in its submission, the group will receive a ZERO mark. If the code does not compile, the group will receive a ZERO mark.**