Ammar Rafiqui (218931410)

MD Ahnaf Hyder (218207159)

Rigzin Shawootsang (217242124)

Kevin Dao (218118190)

Yash Dani (218462531)

EECS 3311

Alvine Boaye Belle

2024-03-22

Justification for choosing these six design patterns:

Factory Pattern for User-Client Relation. We know that the Factory Pattern provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created. With how the user-client relationship is set up it creates different types of users based on the type of data passed to it. It can be seen then the Factory Pattern works best for this situation since it works in a similar way in which it returns an instance of one of several possible classes depending on the data provided. The User class represents the different types of users of the library system, the User Factory class is responsible for creating User objects, Client class is an abstract class that all the other user types inherit from and defines methods that users can perform. The other user type classes inherit from the Client class and can implement the methods specific to their user type. For instance, a student might be able to reserve books while a visitor might not have that privilege.

Observer Pattern for Library Management System. This pattern aligns perfectly for this situation since the classes the Library Management System connects to need to be updated regularly on the changes that have happened and we know that the Observer Pattern works best here since it is made for situations when one object changes its state, all its dependents are notified and updated automatically. This means that the LibraryManagementTeam is notified whenever a new item is added, an item is removed, or an item's location is changed. Thus, it can make it easier to maintain and modify the system over time.

State Design Pattern for Borrowed Item and Request. This pattern works here the best since the State Design Pattern allows an object to change its behaviour when the internal state of that object changes particularly where an object depends on its state and its behaviour must be changed during run time depending on its internal state. As such there can be a state class that is the superclass for both of them that changes whether the user wants to know his borrowed item or requests a new item which can only be known during run time as such it works perfectly with the State Design Pattern.

Command Pattern for Newsletter and View Notifications. Can use command pattern for GUI state when using it to view/close newsletters and notifications. Using the NewsletterSystem and NotificationSystem, these classes should be able to execute the commands given to them, such as viewing due dates and viewing newsletters. It will be able to close the associated GUI by closing it, which is another command. The User class will end up performing the commands, which can be logged if need be, as it is not guaranteed that every user will have the same requests.

Proxy Pattern for Payment. This pattern adds a layer of security to provide controlled access and further protection based on the client's access rights. In this case, the protection proxy which is ProxyPayment controls access to the original object which is Payment. ProxyPayment acts as a proxy for the implementation of the PaymentInterface. Which delegates all method calls to the underlying implementation. PaymentImpl provides the implementation of payment processing and stores the payment details.

Visitor Pattern for Physical Items. This pattern allows a flexible and clean way to perform operations on different types of physical items without changing individual classes. The Visitor Pattern enables us to operate on a group that contains similar objects. In this case, the concrete classes/elements are book, magazine and CD which implement the PhysicalItems interface. The RentingVisitor interface defines the visitor class operations. If the logic of operations were to change then rather than changing everything in the item classes, all that needs to be changed is in the visitor implementation.

Discussion about how each requirement can be achieved:

Req1: This request is achieved through the user first going through the userFactory and client class as a user which checks the email and password either denying or giving them a successful registration. If they are not a visitor, they become one of the four types of users, which further requires them to go through the validation process from the LibraryManagementTeam and LibraryManagementSystem classes.

Req2: This request is achieved through the user having become registered giving them the option to through the use case rent a physical item, open an online book, or subscribe to an online university-provided newsletter which happens through the permission of the LibraryManagementTeam class and LibraryManagementSystem class after they have checked the SystemInventory class. The penalty for overdue books and loss of browsing privileges if three items are overdue and should books be overdue for more than 15 days will be labelled lost both become individual methods that are applied by the LibraryManagementTeam class.

Req3: This request is achieved through the PhysicalItemDetails class which manages the currently rented books and checks if they are past the due date or if the due date is approaching within.

Req4: This request is achieved through the registered user choosing the subscribe to newsletter option. The NewsLetterSystem Class satisfies this requirement with the subscribe method implemented from the Newsletter interface under the Command Pattern. Similarly, users have the option to cancel subscriptions, similarly satisfied by the NewsLetterSystem class. The option for payment for the newsletter is made available to the user by applying the proxy pattern, here the payment is processed and stored by the actual PaymentImpl object through the reference of the PaymentProxy class.

Req5: This request is achieved when the user goes through the search for a book through the LibraryManagementSystem class which includes the show for similar recommendations of other book use cases.

Req6: This request is achieved if the user is a faculty member who tracks current teaching courses and previously used book methods which are connected back to the LibraryManagementSystem. It then connects to the NotificationSystem class which notifies when there is a new textbook edition which is sent to the LibraryManagementTeam class if there is a new edition so that they work to procure a new book.

Req7: This request is achieved by the LibraryManagementSystem class as it goes through the add item method. It includes the manage item details method which includes whether the item can be rented or not.

Req8: This request is checked when the LibraryManagementSystem class creates a virtual copy of a textbook for a student to use, it is then removed from the account once the course is over.

Req9: This request is met by users who call upon the request books method, and state what type it is, depending on that type the LibraryManagementSystem class will inform the user if the book is available or that based on their type it may not be prioritized and the class notifies the user.

Req10: This request is handled by the Publishers class use which implements the method to buy discounted items and is afterward handled by the LibraryManagementTeam class which connects back to the User class and includes the method for handling the payment options which connects to the Payment class.

Req11: This request is achieved through the App class which connects all classes allowing it to manage and store system data CSV files.

Justification for component decomposition and interactions:

Command Pattern:

The User interface represents the system's users. It interacts with the UserAccount connector to manage their UserSettings and with the NotificationSystem component, which receives notifications from the LibraryManagementSystem interface. The UserAccount connector stores the user's data and authenticates the user. The ManageSubscriptions component handles users' requests to subscribe to or unsubscribe from newsletters. It interacts with the PaymentSystem and NewsletterCatalog components. The NewsletterCatalog component stores information about available newsletters. The PaymentSystem component handles payments for newsletter subscriptions. It interacts with the User and Payment interfaces. The Payment interface receives the user's payment details.

The newsletter/notification system of the library app is designed to provide newsletters to users in an efficient manner while also providing timely alerts and news. Central to this subsystem is the NotificationSystem, which communicates various notifications to users. It operates together with the UserNewsletterActions component, enabling users to interact with the newsletter content. The NewsletterCatalog provides a complete list of available newsletters, allowing users to browse and select newsletters that interest them. This catalogue is directly linked to the ManageSubscriptions component, where users can personalize their subscriptions. With these components in place, the system can provide a better user experience overall. Moreover, the NewsletterSystem, along with the NewsletterInterface makes possible the creation, update, and removal of newsletters. This ensures that the newsletter content is dynamic and up-to-date for the user base. The system also incorporates UserSettings with newsletter subscription settings. This integration allows users to manage their information and preferences in a main setting, simplifying the user experience. Interactions among these components are carefully organized to provide a flow of information from the library management system to the user. For example, the process from subscribing to newsletters in ManageSubscriptions to receiving notifications through the NotificationSystem shows the subsystem's efficiency and user-centered design. This structured approach to managing newsletters and notifications highlights the system's ability to provide specific needs for fast communication and a personalized experience.

Proxy Pattern:

The User interface represents the system's users. It interacts with the PaymentSystem component to process payments. The PaymentOption component represents the different payment options available (e.g., credit card, debit card). The PaymentMethod component processes the user's payment details. It interacts with the PaymentSystem to validate the payment information.

The subsystem classifier for the payment system using the proxy pattern is an important part of the library app. Each specialized component is used to handle different important tasks of the transaction process. Due to the proxy pattern, components such as ValidatePayment and SecurePayment help optimize the system making it more efficient and streamlining the transaction process. These components also make sure the system is safe from fraud with an importance on security, making sure each transaction is valid. The SubscriptionManager integrates with these security measures, managing user subscriptions smoothly, enhancing the user experience and system efficiency. PaymentOption and PaymentMethod modules provide flexible interfaces for users to select and validate their payment choices, bringing ease of use and adaptability to various payment methods. The interactions between these components streamline the payment process for both the user experience and the library system allowing for modular updates and maintenance without disrupting the overall system quality. With these components in place, handling user requests to processing payments in the backend allows the system to adapt to security and ease of user experience.

Factory Pattern:

The Client interface interacts with the user Factory component to request user objects which then interacts with the LibraryManagementSystem interface. After the user has been validated they go through the database and then they are finally created.

The subsystem classifier for the client registration system using the factory pattern is an important part of the library app, allowing for security and flexibility. Each component within this system is important in managing the registration process with efficiency. The UserFactory component serves as a central point for creating user profiles, which simplifies the management of different types of users. This component helps in keeping the user creation process modular but consistent. The UserRegistration component, alongside AccountDetails, is key in processing user information. This ensures that all necessary data is securely stored in the system. Furthermore, the ValidateUser component checks the validity of user inputs, preventing errors and false data from being in the system, which allows the library team to maintain a reliable user database. The NotificationSystem is integrated to provide immediate feedback to users upon successful registration or in the event of errors. Interactions among these components are meticulously designed to ensure a smooth and secure registration process.

Visitor Pattern:

The User interface represents the system's users. It interacts with the LibraryManagementSystem interface to check for item availability and then interacts with the Payment interface, which checks the item, and authorizes and validates the payment afterwards. The Library Store Page component contains the library items like CDs, magazines and books under the PhysicalItems component and is managed by the LibraryDatabase.

State Pattern:

The User interface represents the system's users. Which contains the UserAccount component that interacts with the LibraryDatabase component that checks for item availability through its respective component. The UserAccount component can interact with

the ItemBorrowDate component to update the user on an item's due date. The rest of what it does is the same as what is said in the visitor pattern.

The subsystem for renting and returning physical items in the library app is built around the State and Visitor Pattern, allowing for items to transition smoothly between various states such as available, reserved, and checked out, enabling a dynamic and responsive rental process. At the core of this subsystem, the ItemState interface defines the possible states of physical items. This design allows for an item's state to change dynamically and fulfill the different requirements of each state. The LibraryDatabase is key for keeping track of all item details and availability, working with ItemManagement to handle item checkouts, returns, and reservations accurately. Users can easily search and manage rentals through the library HomePage and UserAccount, supported by features like ViewItems for browsing and CheckAvailability for checking item status. Transactions for rentals are securely managed by the ProcessTransactions component alongside PaymentMethod, ValidatePayment, and AuthorizePayment, ensuring safe and smooth payment processes. Overall, this subsystem allows the library app to manage item rentals and returns while being user-friendly.

Observer Pattern:

The User interface represents the system's users. It interacts with the LibraryManagementSystem interface to check for notifications and then interacts with the LibraryStorePage component to rent out items. The Library Store Page component contains the library items like DVDs and books under the PhysicalItems component and payment is managed by the Payment interface for any item that is purchased from the library. Both Publishers and LibraryManagementTeams manage notifications alongside LibraryManagementSystem and they manage the LibraryDatabase component that oversees all items and hands out special deals for any item.

The subsystem focusing on collaborations between the library management team and publishers to offer special deals on items employs the Observer Pattern, making sure that all users are timely notified about new deals. At the core of this subsystem are the Publishers and SpecialDeals components, where publishers can input information about new deals and promotions. The LibraryManagementTeam acts as the observer, monitoring these inputs to organize and manage the deals that will be made available to the library users. The NotificationSystem is an important component that uses the Observer Pattern to send information about special deals to users. Through components like SendNotification and UpdateNotificationAlerts, the system ensures that users are kept in the loop regarding available offers. The system integrates with the LibraryDatabase to keep a record of all special deals, alongside the inventory details of items on offer. User interaction with the subsystem is facilitated through interfaces like the Homepage and ViewNotification, where users can discover and learn more about current special deals. Additionally, UserSettings allows users to customize their notification preferences, ensuring they receive alerts about deals that match their interests. The InventoryManagement component works alongside ItemAvailability to monitor the stock levels of items on promotion, ensuring that the library can fulfill user demand for these items.

Overall component diagram:

The library app's use of subsystems and components designed to manage a wide range of library operations, from user registration and authentication to notifications, searches, recommendations, and inventory management. With the subsystems in place, the library app can provide a smooth user-friendly experience while being efficient and modular for updates. **Client Registration and User Authentication, t**hese subsystems form the starting point for users into the library app. The Client Registration subsystem handles the initial user sign-up process, validating user information and categorizing user types. Once registered, the User Authentication subsystem takes over to manage logins and session validations, allowing access to the app's features. **Notification System**, integrated closely with User Authentication and User Account subsystems, this system is responsible for managing and sending notifications to users, keeping them informed about library updates, news, and alerts. **Search and Recommendation**, this subsystem interacts directly with the User Session component, utilizing user data and search history to provide tailored recommendations and search results. It enhances the user experience by making the shopping experience more personalized, leveraging the Library Management and Collaborations subsystem to include special deals in recommendations. **User Account Management**, this crucial subsystem allows users to manage their settings, including notification preferences, subscription to newsletters, and overview of fees. It pulls information from the Payment System for financial transactions and the Newsletter System for subscription management, ensuring users have control over their library transactions. **Library Store and Inventory**, the Library Store subsystem stores the browsing, selection, and transaction processes for library items, directly interfacing with the Inventory subsystem to check item availability and manage rentals or purchases. It works with the Payment System for financial transactions, ensuring a smooth checkout process for users. **Library Management and Collaborations**, this subsystem works to organize new collaborations or special deals on items. Publishers and library staff can hold special offers, ensuring that the library's store remains interesting and offers valuable items to users. **Payment System** handles all transactions within the app, from user rentals and purchases in the Library Store to managing user charges in the UserAccount subsystem. It ensures secure processing of payments, ensuring safe and smooth payment processes.

Name of the team member	**Tasks completed by the team member**	**Participation of the team member**
Ammar Rafiqui	• Updated the class diagram with three new design patterns and wrote their justification (Factory Design Pattern, Observer Design Pattern, State Design Pattern)	20% (for a team of 5)

	• Wrote about how each requirement can be achieved for the class diagram • Wrote the Justification for component interactions • Wrote the code for the App class and all the classes underneath the package UserData and SystemInventory	
MD Ahnaf Hyder	• Created component diagrams for the overall class diagram and the 6 design patterns subsystems. • Formatted the written report. • Wrote justification breakdown of component diagrams, their subsystems and their interactions.	20% (for a team of 5)
Rigzin Shawootsang	• Added two new design patterns, visitor and proxy and wrote their justifications. • Worked on code for packages PaymentSystem, ItemsData, Renting Visitor and Renting Visitor Inquiry.	20% (for a team of 5)
Kevin Dao	• Adjusted class diagram with design pattern • Added design pattern command pattern with justification • Worked on code on packages MenuOptions, NewsletterSystem	20% (for a team of 5)

	and NotificationSystem	
Yash Dani	• Worked on all design patterns applied in the class diagram to • eliminate errors, contributed in justifications for the • diagram, prepared the DOM and implemented the • skeleton of the complete project, implemented the • GUI, made the presentation video.	20% (for a team of 5)