METHOD FOR IMPLEMENTING LIBRARY MANAGEMENT SYSTEM

1. Library class:

* Singleton pattern – This ensures that the library class should have only one instance throughout the application
* This pattern ensures that only a single instance of the class is created and is globally accessible.
* We declare the Library instance as private which prevents from using it outside the class
* List<User> - A list of users in the library
* List<Course> - A list of courses used in the library
* getInstance() – Provides access to single instance which creates one if there isn’t any.
* addUser(user:User) – Adds a new user to the library’s user list

1. User class:

* The Abstract factory pattern can be used to create different types of User objects – Student, Librarian, Teacher using a factory method
* The notify(message: String) method can be part of the **Observer Pattern.** Users (observers) are notified when certain events occur in the system (e.g., overdue books, updates from the library).
* name: The name of the user.
* course: The course that the user is associated with.
* role: The role of the user (e.g., STUDENT, LIBRARIAN, TEACHER).
* getName() : Returns the name of the user
* getCourse() : Returns the course of the user
* getRole() : Returns the role of the user
* notify(message:String) :Notifies the user about the event (Books which are overdue or arrival of new books)

1. UserFactory class:

* The UserFactory class is responsible for creating different types of users based on their roles and departments.
* It abstracts the instantiation of users so that the main method doesn’t know about the specific user details

1. Course class:

* No design pattern is required as it is a simple POJO class (Plain Old Java Object) with basic attributes and getter methods

1. UserRole Enum:

* Enum pattern can be used over here where each role is constant.

1. PaymentContext:

* The **Strategy Pattern** is applied in the PaymentContext class. The payment behavior (how the payment is processed) is abstracted through the **PaymentStrategy** interface.
* paymentStrategy : It acts as a reference to the payment strategy interface which is used to perform the actual payment processing.
* setPaymentStrategy : It is used to set the payment strategy based on the current contect
* pay() :Assigns the payment process

1. PaymentStrategy:

* It has no attributes
* It has pay() method to process payment.
* The PaymentStrategy makes sure the payment strategies – creditcard,online and cash payments are implemented.