Github link for project
https://github.com/kanishk062001/oop-project

Analysis of oop project

This project loosely follows the **command pattern**.
Here Lms class uses Course class and User class . Then Student and admin class inherits User class .Both of them use their respective service class to manage users. StudentService and AdminService class implements Service class which in turn uses Course class to manage their respective Objects. Here the command interface is Service interface.Request class is a Course class. Studentservice and AdminService are the concrete classes using Service Interface.The command invoker class is the User class. This project is loosely based on this as User does not directly commands respective service classes but in turn, it is it's derived classes Student and Admin class which uses it. Also request class(Course) is also coupled with Topic class which interferes with the command pattern.

1. This project is lacking in terms of encapsulation as most of the attributes of class are public.Also each attribute does not have its getter/setter methods for accessing attributes. In the main program also objects are directly calling attributes of class, which makes the program exposed to outside.
2. This project strongly favours composition over inheritance as the Course class is composed of Topic class. Also the User class is composed of Course objects. Even the service classes of each type were composed of its related user type which helps the user

class for performing various functions.In the whole project there were only 2 cases of inheritance but several cases of composition. This fact is evident from the uml class diagram. This composition allows accessing methods much more efficiently.

3. The objects of classes were loosely coupled as no class was completely contained in another. This reduces the risk that a change made within one element might create an unanticipated impact on the other elements. Such as topic class is loosely coupled with the course class but the Course class is tightly coupled with User class. A slight change in course class could completely affect the user class but a slight modification to topic class performs a negligible action on the course class.
4. Classes should be open for extension and closed for modification.The main benefit of this approach is that an interface introduces an additional level of abstraction which enables loose coupling. The implementations of an interface are independent of each other and don’t need to share any code. If you consider it beneficial that two implementations of an interface share some code, you can either use inheritance or composition.This project does not follow this approach as it was not known to me.