* **Singleton Pattern (creational)**
* Egear Initialization: khởi tạo biến instance global với từ khóa static final để tạo instance của của class ngay thời điểm được gọi => tạo static method getInstance() trả về biến instance ở trên để đảm bảo chỉ 1 instance của class được tạo ra.
* Lazy Initialization: khởi tạo biến instance global với từ khóa static nhưng sẽ check null ở method getInstance(), nếu instance == null thì gán new instance mới của class.
* ThreadSafe: giống cách lazy nhưng dùng từ khóa synchonized ở method để đảm bảo chỉ 1 thread trong cùng 1 thời điểm được sử dụng instance của class
* Double check locking: giống thread-safe nhưng không dùng synchronized ở khai báo method mà sẽ check trong method, khi instance == null thì dùng synchronized (synchronized(Instance.class){}) rồi check null lần nữa ở trong lock của synchronized rồi mới khởi tạo new instance của class.
* **Factory Pattern (creational)**
* Thuộc nhóm creational, nó chịu trách nhiệm quản lý và trả về những object theo đúng những gì được yêu cầu. Nó giống như việc ta chỉ thấy nguyên liệu tạo sp và sản phẩm được tạo ra trong nhà máy chứ không biết quy trình chi tiết như thế nào.
* Tạo lớp factory để implement 1 method dùng cho việc tạo đối tượng mong muốn mà không cần chỉ định trực tiếp lớp cụ thể của đối tượng
* **Builder Pattern (creational)**
* Khi 1 class có quá nhiều properties và khi cần khởi tạo class thì đôi khi sẽ không cần một vài properties, với mỗi lần khởi tạo khác nhau có các prop khác nhau thì sẽ cần tạo rất nhiều contructor tương ứng => sử dụng pattern này để khắc chế nhược điểm kia, nó giúp tạo object có nhiều properties khi khởi tạo sẽ không cần truyền hết các prop mà chỉ cần truyền những tham số cần thiết tùy theo nhu cầu, có thể mở rộng class trong tương lai mà không ảnh hưởng đến code cũ.
* Các thành phần:

+ Product: là object cần tạo với nhiều thuộc tính khác nhau.

+ Builder Interface: nơi khởi tạo các methods cần cho object.

+ ConcreteBuilder: implement builder interface và cài đặt chi tiết cho các method cần để tạo ra object

+ Client: nơi khởi tạo object.

* **Adapter Pattern (structural pattern)**
* Cho phép các interface không liên quan tới nhau có thể làm việc cùng nhau, là đối tượng giúp kết nối các interface, giữ vai trò trung gian giữa 2 lớp, chuyển đổi interface của 1/n lớp có sẵn thành 1 interface khác => giúp các lớp có interface khác nhau giao tiếp được thông qua interface trung gian, không cần chỉnh sửa code của các lớp hiện tại
* Thành phần:

+ Adaptee: interface kh tương thích và cần được tích hợp vào

+ Adapter: lớp tích hợp, giúp các interface kh tương tích tích hợp được với nhau, chuyển đổi interface cho adaptee và connect adaptee với client

+ Target: interface chứa các method được sử dụng bởi client

+ Client: sử dụng các method trong target

* Cách implement Object adapter – Composition: 1 class mới(Adapter) sẽ reference đến 1/n object của class có sẵn với interface kh tương thích(Adaptee), đồng thời cài đặt interface mà user mong muốn(Target)
* Case hay dùng:

+ Khi không thể kế thừa class A nhưng lại muốn class B có những xử lý giống như class A => cài đặt B như object adapter,

* Developed many functions for the project.
* Coloraboted and discussed with other members of the team to fix bugs, integrate backend-frontend, and design a database.
* Ultiliezed unit test using Junit before merging code and running CI/CD.
* Taked charge of reviewing codes from other members before merging and fix conflicts if have.
* **Observer Pattern( Behaviror)**
* Định nghĩa mối phụ thuộc 1-n giữa các đối tượng để khi mà một đối tượng có sự thay đổi trạng thái, tất cả các thành phần phụ thuộc của nó sẽ được thông báo và cập nhật 1 cách tự động
* Thành phần:

+ Subject(Interface): chứa danh sach các observer, cung cấp method để có thể thêm và loại bỏ observer

+ Observer(Interface): định nghĩa 1 method update() cho các object sẽ được subject thông báo đến khi có sự thay đổi trạng thái

+ ConcreteSubject(Class impl): implement các method của subject, lưu trữ trạng thái danh sách của các ConcreteObsever, gửi thông báo đến các observer của nó khi có sự thay đổi của trạng thái

+ ConcreteObserver(Class impl): implement các object của Observer, lưu trữ trạng thái của subject, thực thi việc cập nhật để giữ cho trạng thái đồng nhất với subject gửi thông báo đến