Design Pattern:

1.
A university system requires a single shared virtual guard mama who would be able to do student enrollment, grade calculation, and updates. Creating multiple instances of the connection pool would waste resources and lead to conflicts. You must ensure only one instance of the connection pool exists throughout the application.

Solution: Singleton. Class-> Guard mama

2.
The university's IT department is implementing a centralized notification system where the Proctors would be able to send out Emails to students. The system must ensure only one instance is created while sending the sms, as multiple instances could lead to duplicate notifications or race conditions when managing resources like SMTP connections or SMS gateways. The university is also planning to keep a new feature for students to swim in the swimming pool.

Solution: Singleton -> proctor class
And Observer-> celeb-> proctor
fan-> student

```
public class Proctor{//Celeb

//For singleton

        private static Proctor obj=Null;
        private Proctor(){

    }

        public static Proctor getobj(){
                if (obj==Null){
                        obj=new Proctor();
                        return obj;
                }
                else{
                        return obj
                }
        }



//For Observer

        private list <Student> students= new ArrayList<Students>();
        private String mail_to_send;

        String getEmail(){//getstate
```

```
                return mail_to_send;
        }
        void notify_Student_By_Mail(){//notify
                for each (Student s: students){
                        s.seeEmail();
                }
        }

        void send_email_to_Students(String new_email){ //changestate
                mail_to_send=new_email;
                notify_Student_By_Mail();

        }


}

public class Student{//Fan
        private list<Proctor> proctors =new ArrayList <Proctor>();
        String seeEmail(Proctor P){//seeState
                P.getEmail();
        }


//Basic Add,remove

        void addProctor(Proctor p){
                p.addStudent(this);
                proctors.add(p);
        }
        void removeProctor(Proctor p){
                p.removeStudent(this);
                proctors.remove(p);
        }


//Extra methods
        void Swim_in_pool(){
                print('Swimming');
        }

}
```

3.

The university's IT department is implementing a centralized notification system where the Proctors would be able to send out SMS and email to students and faculties.

Solution: Observer-> celeb-> proctor
fan-> student, faculty

```
public class Proctor{//Celeb



//For Observer

        private list <Student> students= new ArrayList<Students>();
        private list <Faculty> faculties= new ArrayList<Faculty>();

        private String mail_to_send;
        private String sms_to_send;

        String getEmail(){//getstate
                return mail_to_send;
        }
         String getSms(){//getstate
                return smsl_to_send;
        }

        void notify_Student_By_Mail(){//notify
                for each (Student s: students){
                        s.seeEmail();
                }
        }
        void notify_Student_By_Sms(){//notify
                for each (Student s: students){
                        s.seeSms();
                }
        }
        void notify_Faculty_By_Mail(){//notify
                for each (Faculty f: faculties){
                        f.seeEmail();
                }
        }
        void notify_Faculty_By_Sms(){//notify
                for each (Faculty f: faculties){
                        f.seeSms();
                }
        }


        void send_email_to_Students(String new_email){ //changestate
                mail_to_send=new_email;
                notify_Student_By_Mail();

        }
        void send_Sms_to_Students(String new_sms){ //changestate
                sms_to_send=new_Sms;
                notify_Student_By_Sms();

        }
```

```
        void send_Sms_to_faculties(String new_sms){ //changestate
                sms_to_send=new_Sms;
                notify_faculties_By_Sms();

        }
        void send_email_to_Faculties(String new_email){ //changestate
                mail_to_send=new_email;
                notify_Faculties_By_Mail();

        }

//Write the basic add remove operation here for both faculty and student


}

public class Student{//Fan
        private list<Proctor> proctors =new ArrayList <Proctor>();
        String seeEmail(Proctor P){//seeState
                P.getEmail();
        }
        String seeSms(Proctor P){//seeState
                P.getSms();
        }


//Basic Add,remove here



}
public class Faculties{//Fan
        private list<Proctor> proctors =new ArrayList <Proctor>();
        String seeEmail(Proctor P){//seeState
                P.getEmail();
        }
        String seeSms(Proctor P){//seeState
                P.getSms();
        }


//Basic Add,remove here



}
```

4.You are the lead developer at a company called *SmartLife Solutions*, which is building a smart home automation system. The goal is to create a unified platform that allows homeowners to control all their

smart devices (like lights and fans) from a single app. However, the devices you want to integrate come from different manufacturers, each with its own API and device interface.

```
// Target Interface
interface SmartLife {
    void turnOnLight();
    void turnOffLight();
    void turnOnFan();
    void turnOffFan();
    void adjustFanSpeed();
}

// Adaptee 1
Class Light {
    void turnOnLight(){
            sout("TurnedOn")
    }
    void turnOffFLight(){
            sout("TurnedOff")
    }
}

// Adaptee 2:
class Fan{
    void turnOnFan(){
            sout("TurnedOn")
    }
    void turnOffFan(){
            sout("TurnedOff")
    }
    void adjustFanSpeed(){
            sout("Adjusted")
    }
}

}

// Adapter
class SmartLifeAdapter implements SmartLife{



    light=new Light();
    fan=new Fan();

    void turnOnLight(){
        light.turnOnLight();
    }
    void turnOffLight(){
        light.turnOffLight();

    }
    void turnOnFan(){
```

```
        fan.turnOnFan();
    }
    void turnOffFan(){
        fan.turnOffFan();
    }
    void adjustFanSpeed(){
        fan.adjustFanSpeed();
    }


}
```

5.Creating and implementing a robust e-commerce system for the "Global Market Hub" was a challenging task. The company advertised the position of Lead Software Architect, looking for someone with the expertise to unify multiple existing payment gateways into a single user-friendly system. You were hired to spearhead this transformation.

The project required you to integrate various payment services, such as PayPal, Stripe, and a legacy bank API, each with its own unique interface, into a single, unified payment processing module. Customers should be able to choose their preferred payment method seamlessly during checkout without worrying about the underlying complexities. Additionally, the system needed to be scalable so that new payment methods could be added in the future without disrupting existing functionality.

```
// Target Interface
interface AllInOnePaymentProcessor {
    void processPayment(double amount);
}

// Adaptee 1: PayPal Payment Service
class PayPalService {
    public void makePayment(double amount) {
        System.out.println("Processing payment of $" + amount + " through PayPal.");
    }
}

// Adaptee 2: Stripe Payment Service
class StripeService {
    public void chargeAmount(double amount) {
        System.out.println("Charging $" + amount + " through Stripe.");
    }
}

// Adaptee 3: Legacy Bank Payment Service
class LegacyBankAPI {
    public void executeTransaction(double amount) {
```

```
        System.out.println("Executing transaction of $" + amount + " through Legacy Bank.");
    }
}

// Adapter
class AllInOnePaymentProcessortAdapter implements AllInOnePaymentProcessor {
    public TypeBasedObjectCreation(String serviceType) {
        if (serviceType.equalse("PayPal")) {
            paymentService = new PayPalService();
        } else if (serviceType.equals("Stripe")) {
            paymentService = new StripeService();
        } else if (serviceType.equals("LegacyBank")) {
            paymentService = new LegacyBankAPI();
        } else {
            throw new IllegalArgumentException("Unsupported payment service type.");
        }
    }


    @Override
    public void processPayment(double amount) {
        if (paymentService instanceof PayPalService) {
                paymentService.makePayment(amount);
        } else if (paymentService instanceof StripeService) {
                paymentService.chargeAmount(amount);
        } else if (paymentService instanceof LegacyBankAPI) {
                paymentService.executeTransaction(amount);
        } else {
                System.out.println("Unsupported payment service.");
        }
    }
}
```

Refactoring

1.

```
class Library {
    private List<Book> books;

    public Library() {
        books = new ArrayList<>();
    }
```

```
public void calculatePriceForGoldCustomers(Book book, Customer customer) {
   sout('Yo Gold Customer")
    if (book.price > 1000 || customer.age>20 || book.author==customer.author) {
        return price * 0.70;
    } else {
        return price;
    }

}
public void calculatePriceForSilverCustomers(Book book, Customer customer) {
   sout("Yo silver candidate")
   If (book.price==500){
        sout('wow')
    }
    if (book.price > 1000 || customer.age>30 || book.author==customer.author) {
        return price * 0.80;
    } else {
        return price;
    }

}
```

Solution:

Duplication->Extract Method
Long Condition-> Extract method

```
class Library {
   private List<Book> books;

   public Library() {
      books = new ArrayList<>();
   }


   public void calculatePriceForGoldCustomers(Book book, Customer customer) {
      sout('Yo Gold Customer")
      return applyDiscount(Book book, Customer customer, 20 ,  .70)

   }
   public void calculatePriceForSilverCustomers(Book book, Customer customer) {
      sout("Yo silver candidate")
      If (book.price==500){
           sout('wow')
       }
      return applyDiscount(Book book, Customer customer, 30 ,  .80)
   }
```

```
    public void applyDiscount(Book book, Customer customer,age_limit,charge_rate) {

        if (isDiscountApplcable(Book book, Customer customer,age_limit)) {
            return price * charge_rate;
        } else {
            return price;
        }

    }
    public void isDiscountApplicable(Book book, Customer customer,age_limit) {

        if (book.price > 1000 || customer.age>age_limit  || book.author==customer.name) {
            return True;
        } else {
            return False;
        }

    }
```

2.

```
class Order {
    private Customer customer;
    private double totalPrice;

    public Order(Customer customer, double totalPrice) {
        this.customer = customer;
        this.totalPrice = totalPrice;
    }

    public double calculateDiscountRate() {

        if (customer.glp() > 100) {
            return  0.1;
        }
        return 0;
    }
}

class Customer {
    private String name;
    private int loyaltyPoints;
    public void PriceRecommendation(double saree_price,double shirt_price,double
panjabi_price,double hat_price){

        //Calculating recommendation rate
        saree_price=2*5*shirt_price;
        panjabi_price=saree_price+hat_price
        Recommendation_rate=panjabi_price*100;
```

```
        Sout("Hooray. done. ",Recommendation_rate);

    }

    public Customer(String name, int loyaltyPoints) {
        this.name = name;
        this.loyaltyPoints = loyaltyPoints;
    }

    public int glp() {
        return loyaltyPoints;
    }
}
```

Solution: Feature Envy-> Move field
Inappropriate naming -> Proper naming
Long Parameter-> create class
Comment->Extract method

```
class Order {
    private Customer customer;
    private double totalPrice;

    public Order(Customer customer, double totalPrice) {
        this.customer = customer;
        this.totalPrice = totalPrice;
    }

}
class RecommendedPrice{
    double saree;
    double shirt;
    double panjabi;
    double hat;

    public RecommendedPrice(double saree,double shirt,double panjabi,double hat){
         this.saree=saree;
         this.shirt=shirt;
         this.panjabi=panjabi;
         this.hat=hat;

    }

}


class Customer {
    private String name;
```

```
    private int loyaltyPoints;

    public Customer(String name, int loyaltyPoints) {
        this.name = name;
        this.loyaltyPoints = loyaltyPoints;
    }
    public void PriceRecommendation(RecommendedPrice){

        Recommendaton_rate=calculate_recommendation_rate(RecommendedPrice)

        Sout("Hooray. done. ",Recommendaton_rate);

    }

    public double calculate_recommendation_rate(RecommendedPrice){
        RecommendedPrice.saree=2*5*RecommendedPrice.shirt;
        RecommendedPrice.panjabi=RecommendedPrice.saree+RecommendedPrice.hat
        Recommendation_rate=RecommendedPrice.panjabi*100;

        return  Recommendation_rate;

    }


    public int getLoyaltyPoints() {
        return loyaltyPoints;
    }

    public double calculateDiscountRate(Customer customer, double totalPrice) {

        if (customer.getLoyaltyPoints() > 100) {
            return  0.1;
        }
        return 0;
    }
}
```