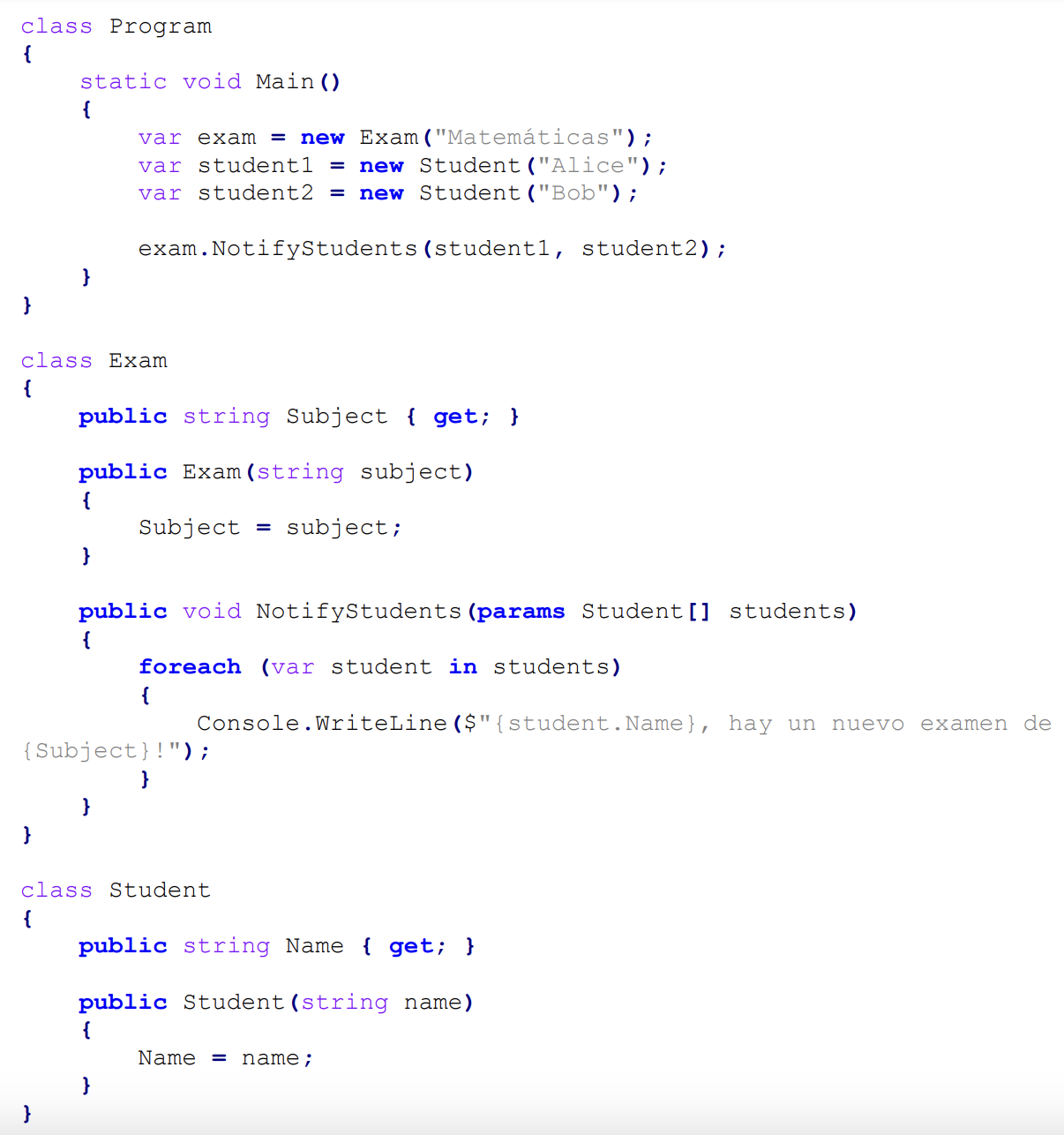
UNIDAD TEMÁTICA 5 – Patrones de diseño– Trabajo de Aplicación 3

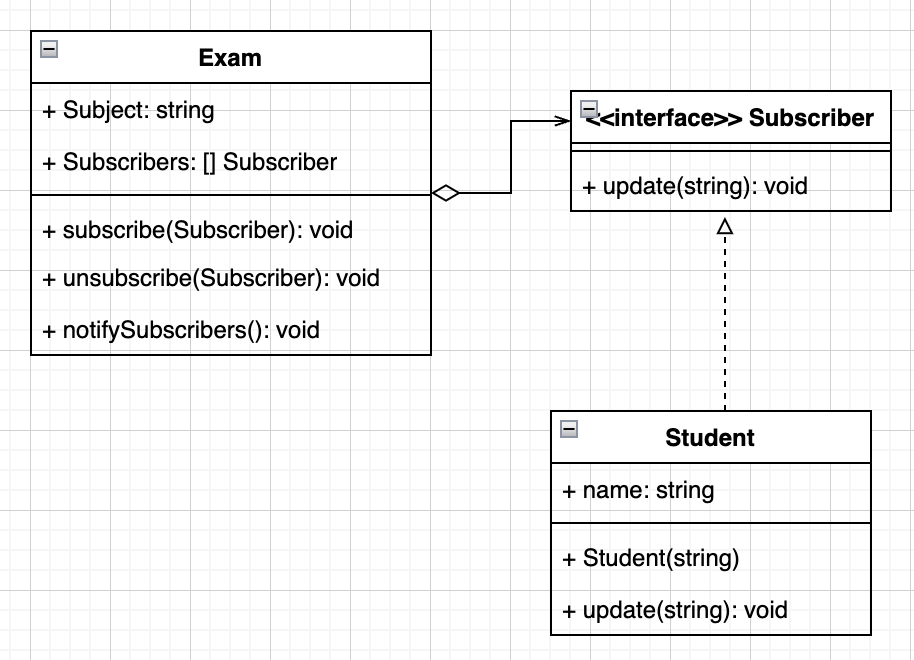
Para cada uno de los siguientes ejercicios, en equipo:

1. Determine que patrón puede resolver el problema de una forma más eficiente.
2. Agregue las clases, interfaces, métodos que considere necesarios para remediar la situación.

EJERCICIO 1

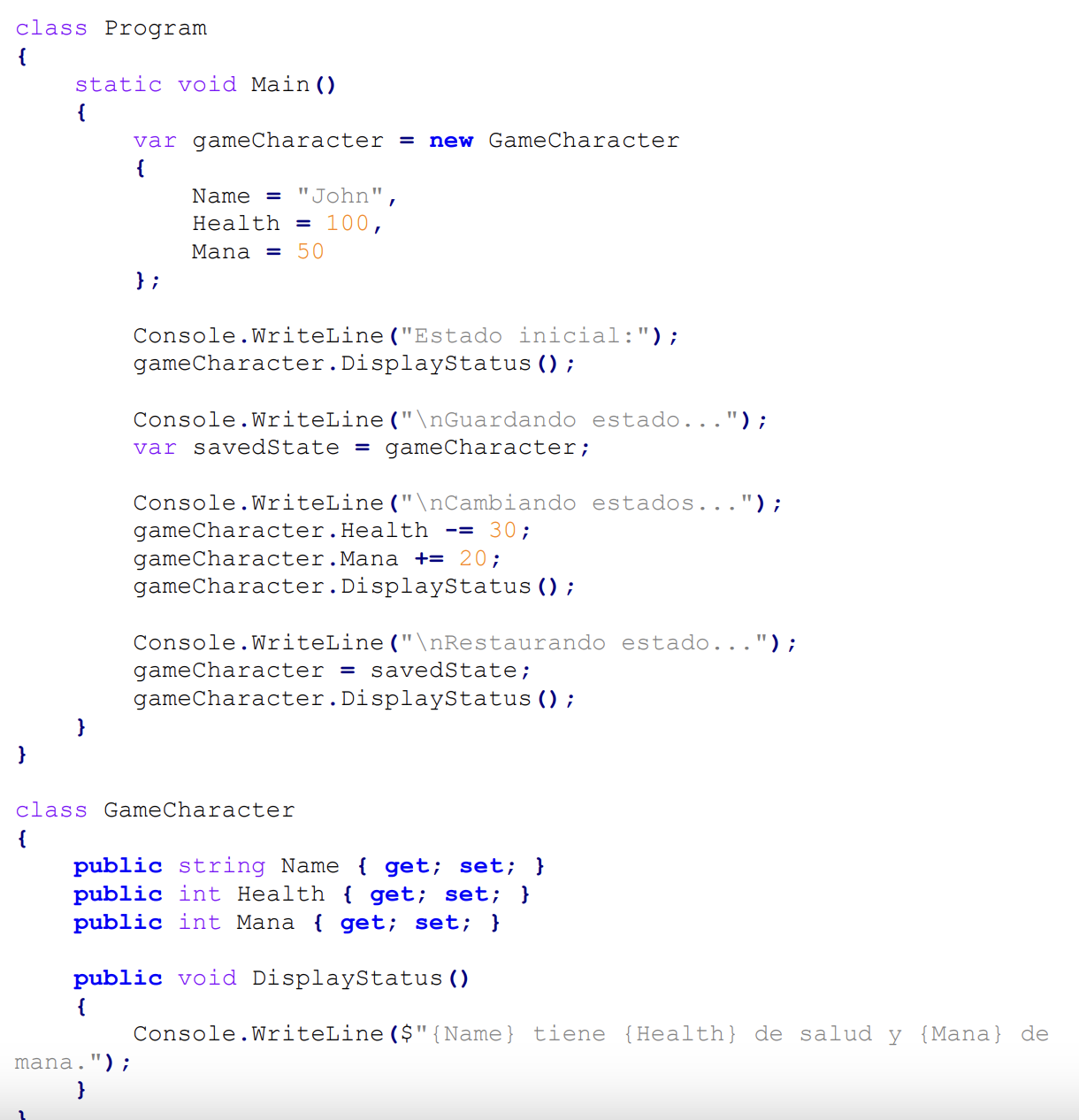


El patrón que resuelve el problema es observer ya que permite notificar al objeto Estudiantes y en base a eso actualizar su estado si corresponde.

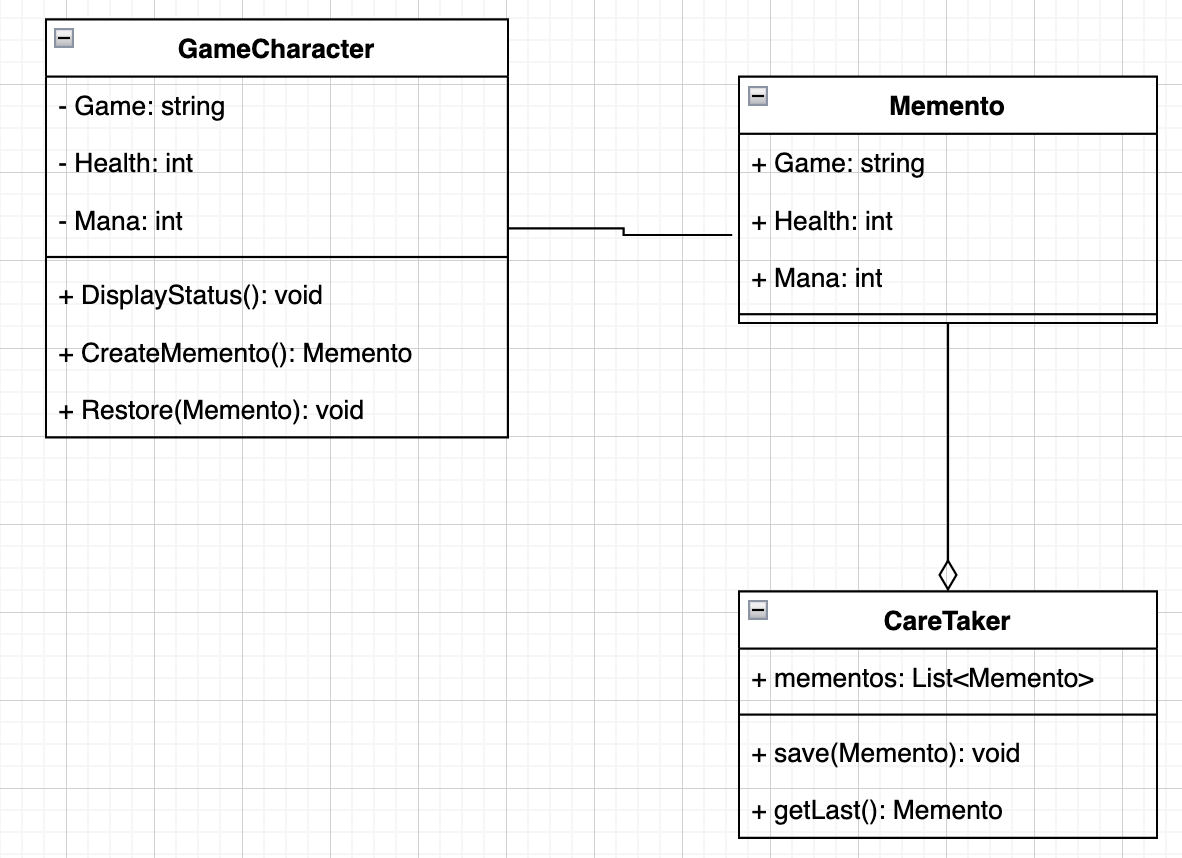


| public interface Subscriber  {  void Update(string subject);  }  public class Student : Subscriber  {  private string name;  public Student(string name)  {  this.name = name;  }  public void Update(string subject)  {  Console.WriteLine($"Student {name} received message: {subject}");  }  }  public class Exam  {  private List<Subscriber> subscribers = new List<Subscriber>();  private string subject;  public void Subscribe(Subscriber subscriber)  {  subscribers.Add(subscriber);  }  public void RemoveSubscriber(Subscriber subscriber)  {  subscribers.Remove(subscriber);  }  public void NotifySubscribers()  {  foreach (Subscriber subscriber in subscribers)  {  subscriber.Update(subject);  }  }  }  public class Program  {  public static void Main()  {  Exam exam = new Exam();  Student student1 = new Student("Alice");  Student student2 = new Student("Bob");    exam.Subscribe(student1);  exam.Subscribe(student2);    exam.NotifySubscribers();  }  } |
| --- |

EJERCICIO 2



El patrón que resuelve es memento ya que se desea guardar estados, cambiarlos y restaurarlos.

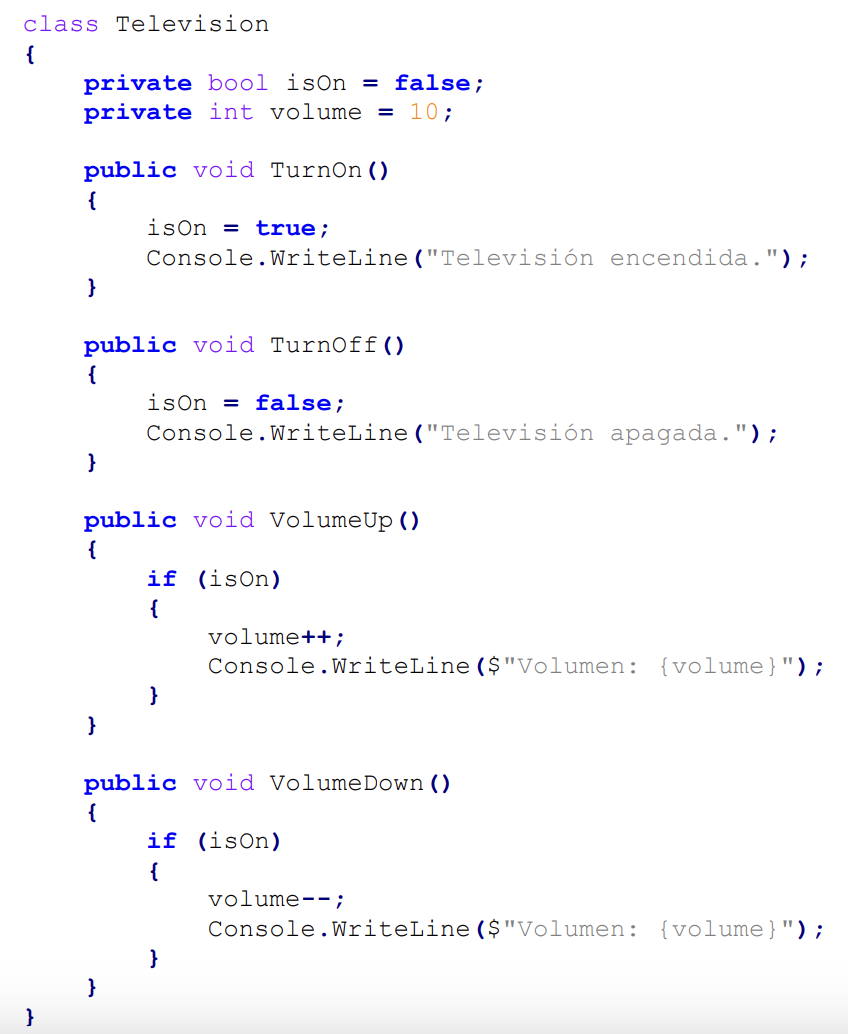


| using System;  using System.Collections.Generic;  using System.Linq;  class Program  {  static void Main()  {  var gameCharacter = new GameCharacter  {  Name = "John",  Health = 100,  Mana = 50  };  Console.WriteLine("Estado inicial:");  gameCharacter.DisplayStatus();  var caretaker = new Caretaker();  // Guardar el memento  caretaker.AddMemento(gameCharacter.CreateMemento());  Console.WriteLine("\nCambiando estados...");  gameCharacter.Health -= 30;  gameCharacter.Mana += 20;  gameCharacter.DisplayStatus();  // Restaurar el memento  gameCharacter.RestoreMemento(caretaker.GetLast());  gameCharacter.DisplayStatus();  }  }  class GameCharacter  {  public string Name { get; set; }  public int Health { get; set; }  public int Mana { get; set; }  public void DisplayStatus()  {  Console.WriteLine($"{Name} tiene {Health} de salud y {Mana} de mana.");  }  public Memento CreateMemento()  {  return new Memento(Name, Health, Mana);  }  public void RestoreMemento(Memento memento)  {  Name = memento.Name;  Health = memento.Health;  Mana = memento.Mana;  }  }  class Memento  {  public string Name { get; }  public int Health { get; }  public int Mana { get; }  public Memento(string name, int health, int mana)  {  Name = name;  Health = health;  Mana = mana;  }  }  class Caretaker  {  private List<Memento> Mementos;  public Caretaker()  {  Mementos = new List<Memento>();  }  public void AddMemento(Memento memento)  {  Mementos.Add(memento);  }  public Memento GetLast()  {  return Mementos.Last();  }  } |
| --- |

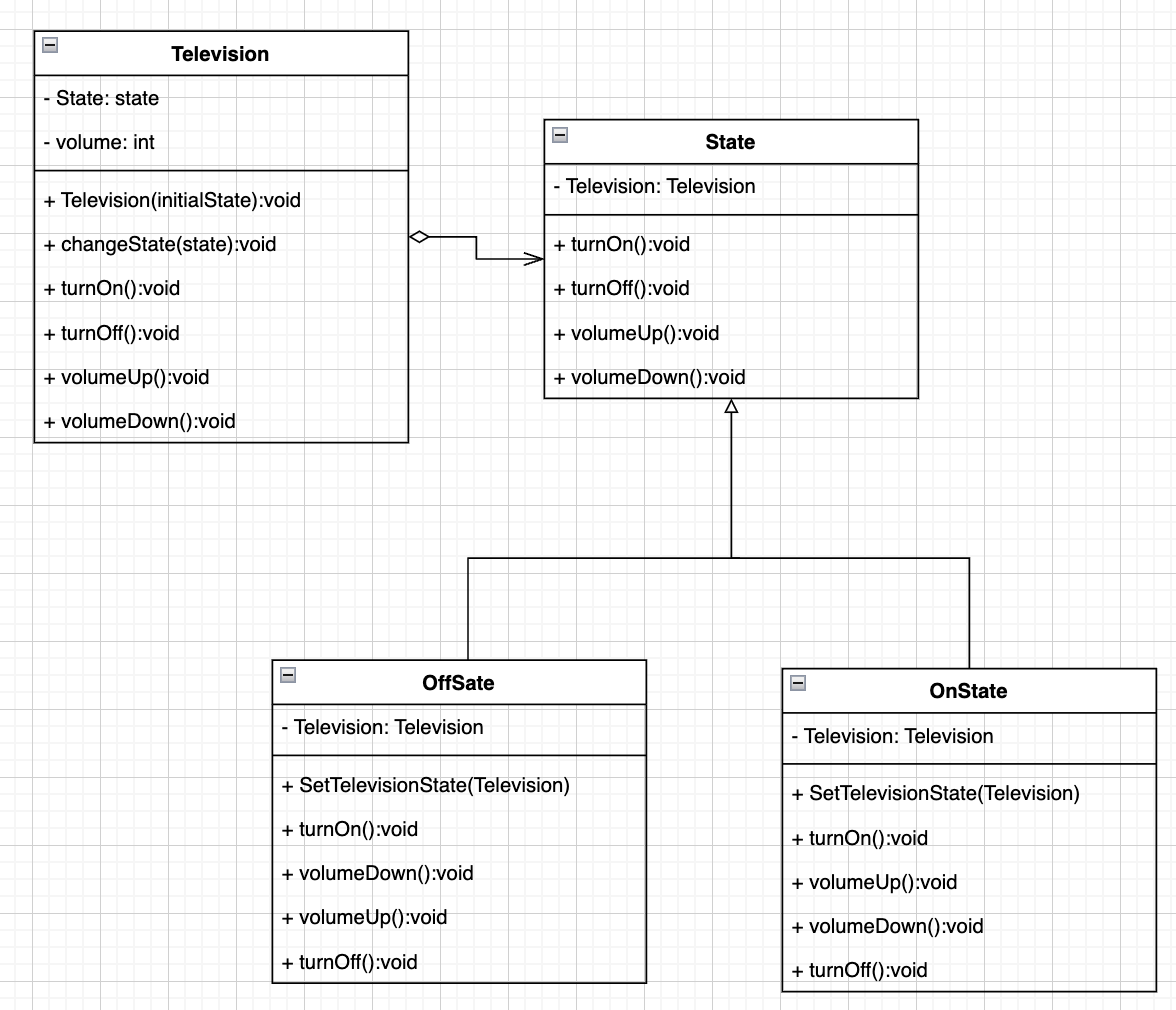
EJERCICIO 3

EJERCICIO 4



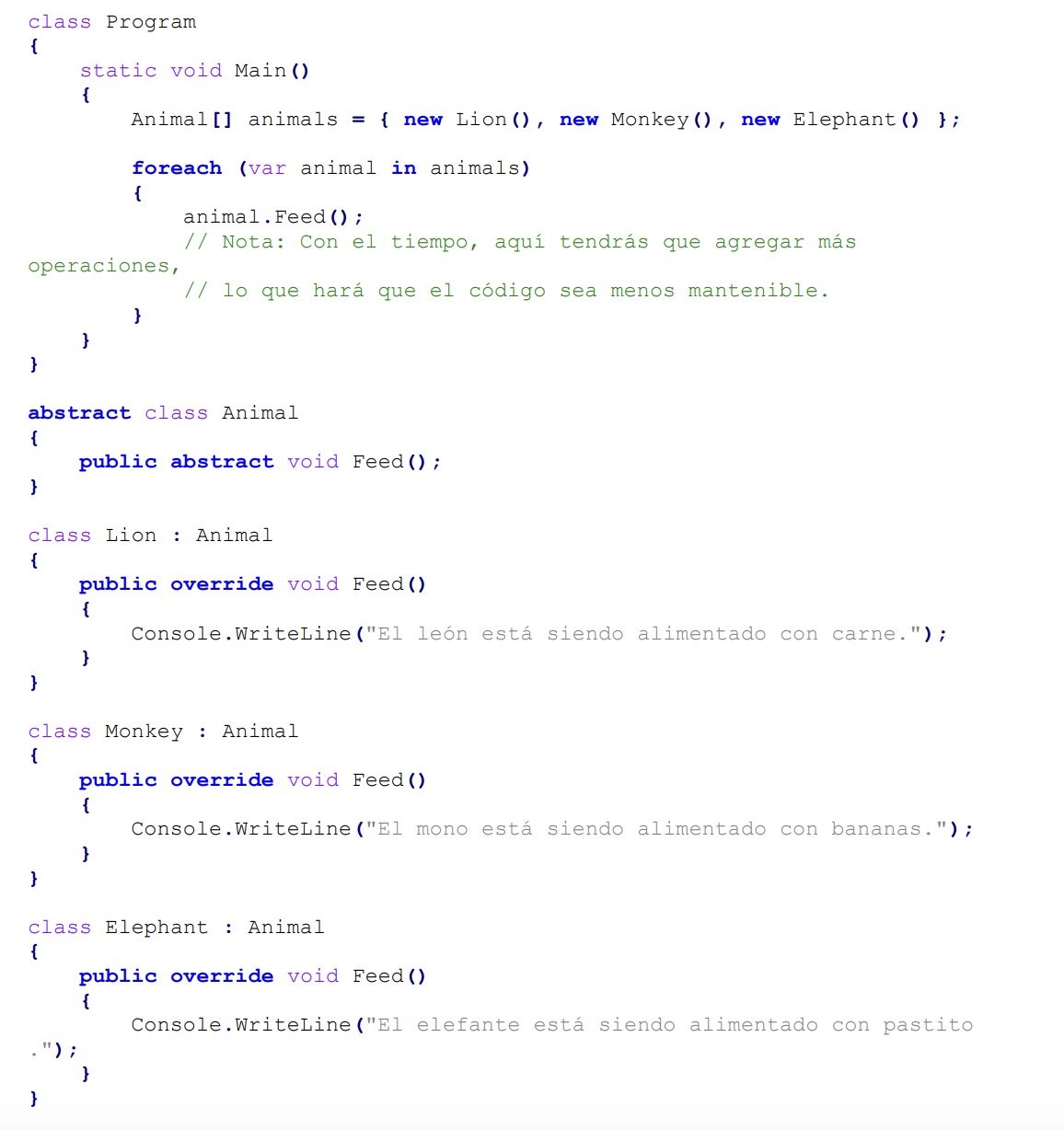


Es State porque en base a condiciones cambia el estado del objeto.

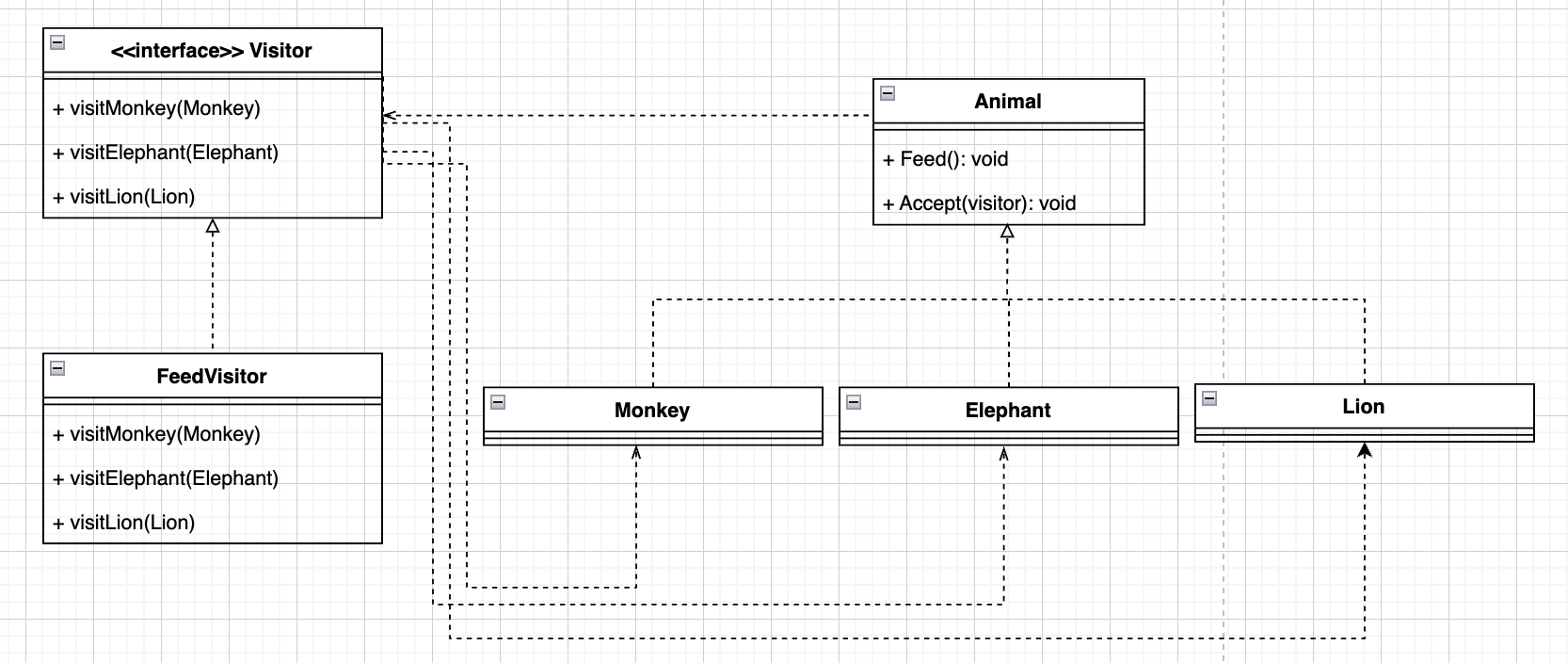


| public class Television {  private State state;  private int volume;  public Television() {  this.state = new OffState(this);  this.volume = 0;  }  public void changeState(State state) {  this.state = state;  }  public void turnOn() {  this.state.turnOn();  }  public void turnOff() {  this.state.turnOff();  }  public void volumeUp() {  this.state.volumeUp();  }  public void volumeDown() {  this.state.volumeDown();  }  }  abstract class State {  protected Television tv;  public State(Television tv) {  this.tv = tv;  }  public abstract void turnOn();  public abstract void turnOff();  public abstract void volumeUp();  public abstract void volumeDown();  }  class OnState extends State {  @Override  public void turnOn() {}  @Override  public void turnOff() {  tv.changeState(new OffState(tv));  }  @Override  public void volumeUp() {  tv.volume++;  }  @Override  public void volumeDown() {  tv.volume--;  }  }  class OffState extends State {  public OffState(Television tv) {  super(tv);  }  @Override  public void turnOn() {  tv.changeState(new OnState(tv));  }  @Override  public void turnOff() {}  @Override  public void volumeUp() {}  @Override  public void volumeDown() {}  }  public class Program {  public static void main(String[] args) {  Television tv = new Television();  tv.turnOn();  tv.volumeUp();  tv.turnOff();  }  } |
| --- |

EJERCICIO 5

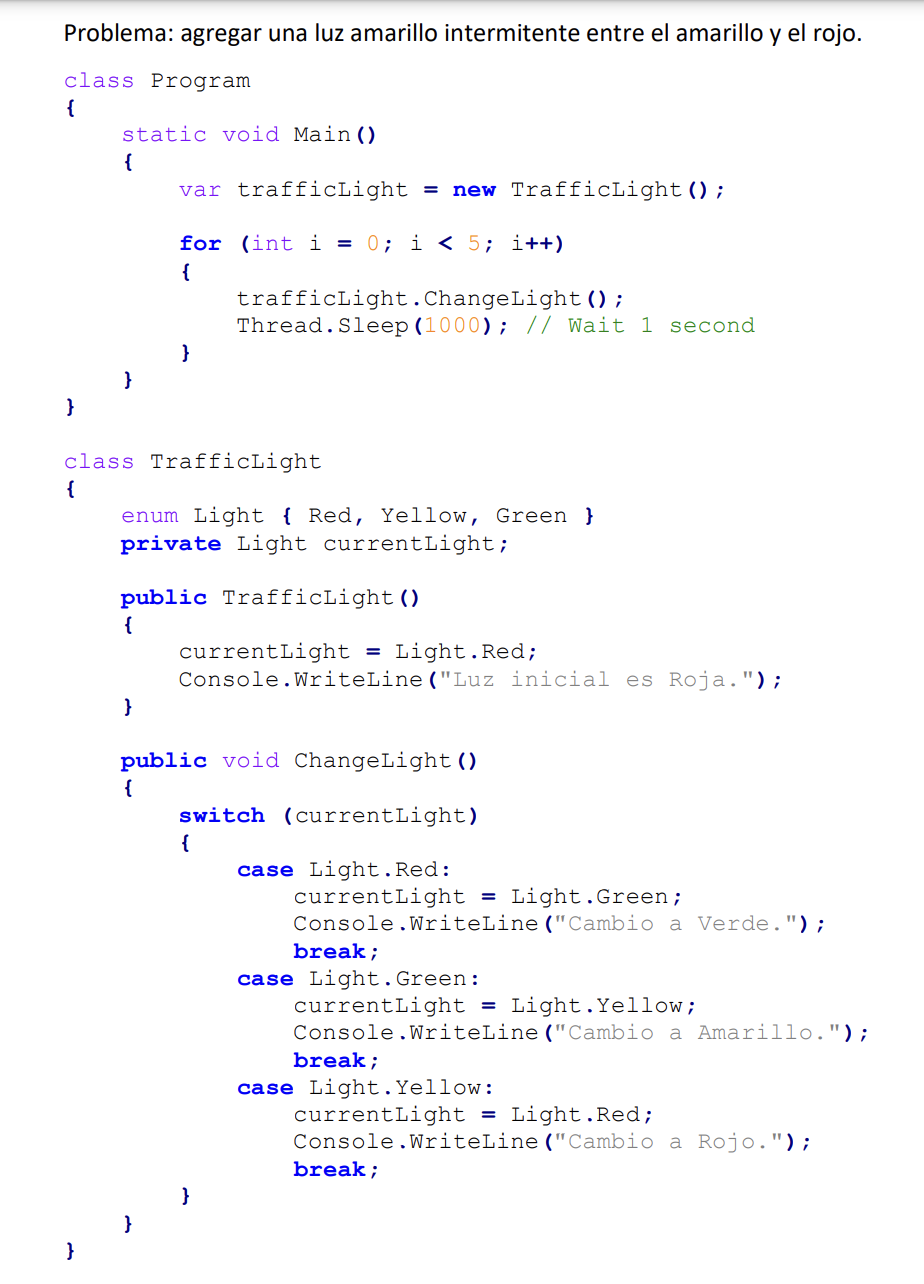


El patrón que resuelve es Visitor ya que este permite agregar funcionalidades sin tener que modificar Animal. Permite separar los algoritmos de los objetos en donde operan.



| public interface Visitor {  void visitMonkey(Monkey monkey);  void visitElephant(Elephant elephant);  void visitLion(Lion lion);  }  public class FeedVisitor implements Visitor {  public void visitMonkey(Monkey monkey) {  monkey.feed();  }  public void visitElephant(Elephant elephant) {  elephant.feed();  }  public void visitLion(Lion lion) {  lion.feed();  }  }  public class WashVisitor implements Visitor {  public void visitMonkey(Monkey monkey) {  System.out.println("Monkey wash");  }  public void visitElephant(Elephant elephant) {  System.out.println("Elephant wash");  }  public void visitLion(Lion lion) {  System.out.println("Lion wash");  }  }  abstract class Animal {  public abstract void accept(Visitor visitor);  public abstract void feed();  }  public class Monkey extends Animal {  public void accept(Visitor visitor) {  visitor.visitMonkey(this);  }  public void feed() {  System.out.println("Monkey feed");  }  }  public class Elephant extends Animal {  public void accept(Visitor visitor) {  visitor.visitElephant(this);  }  public void feed() {  System.out.println("Elephant feed");  }  }  public class Lion extends Animal {  public void accept(Visitor visitor) {  visitor.visitLion(this);  }  public void feed() {  System.out.println("Lion feed");  }  }  public class Program {  public static void main(String[] args) {  Animal[] animals = new Animal[] {  new Monkey(),  new Elephant(),  new Lion()  };  Visitor visitor1 = new FeedVisitor();  Visitor visitor2 = new WashVisitor();  for (Animal animal : animals) {  animal.accept(visitor1);  animal.accept(visitor2);  }  }  } |
| --- |

EJERCICIO 6



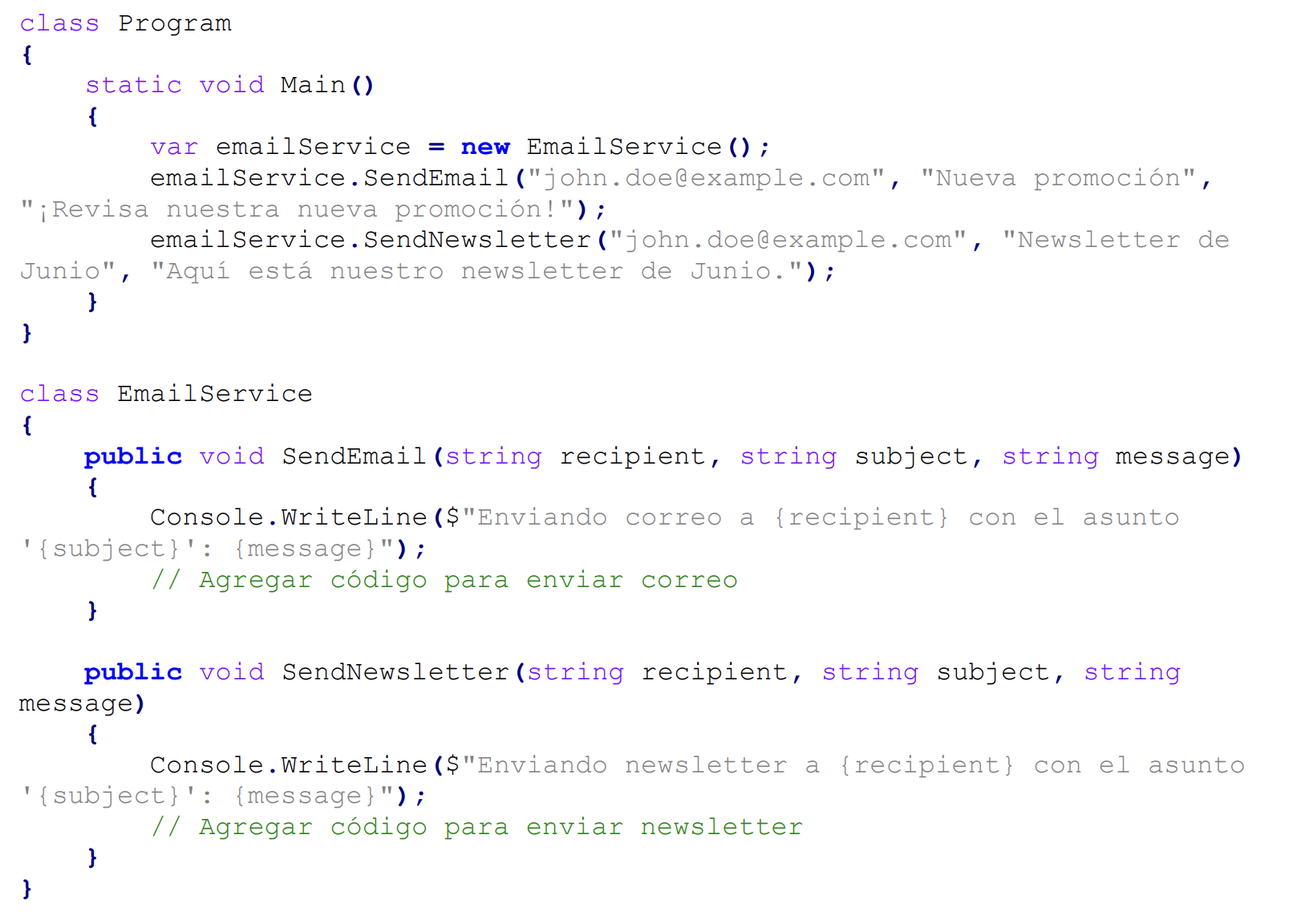
El patrón que soluciona sería State porque Es State porque en base a condiciones cambia el estado del objeto y permite agregar otros estados sin modificar la clase.

EJERCICIO 7

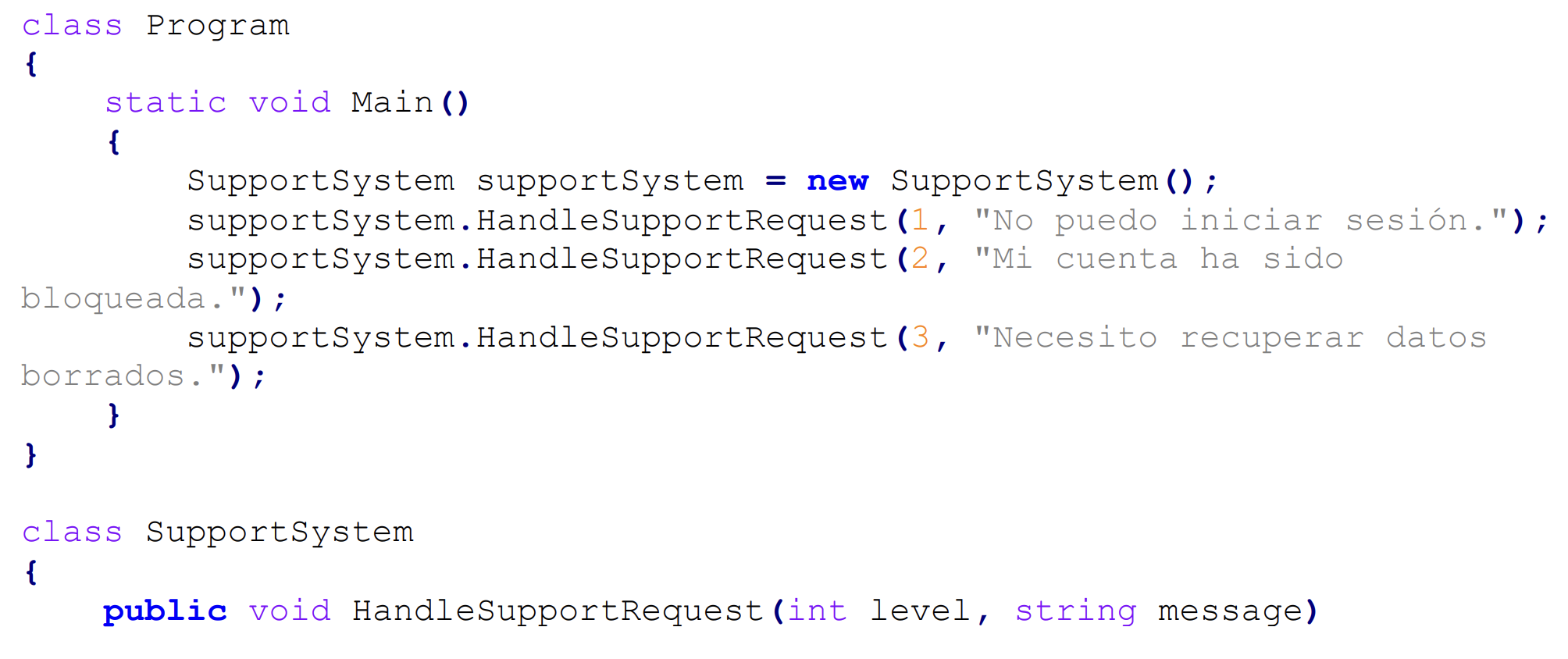


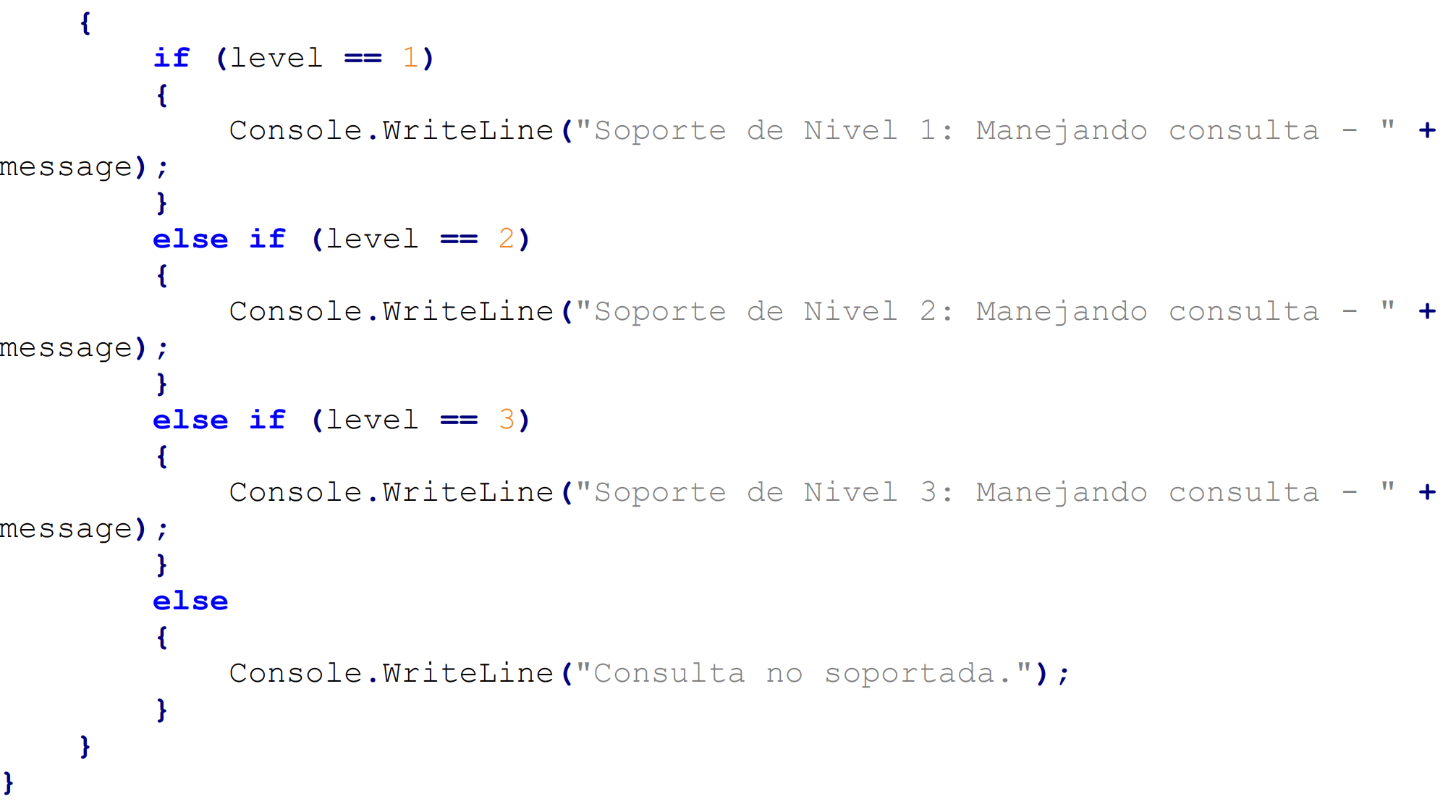
El patrón que soluciona es Strategy porque permite definir una familia de algoritmos, colocar cada uno de ellos en una clase separada y hacer sus objetos intercambiables.

EJERCICIO 8



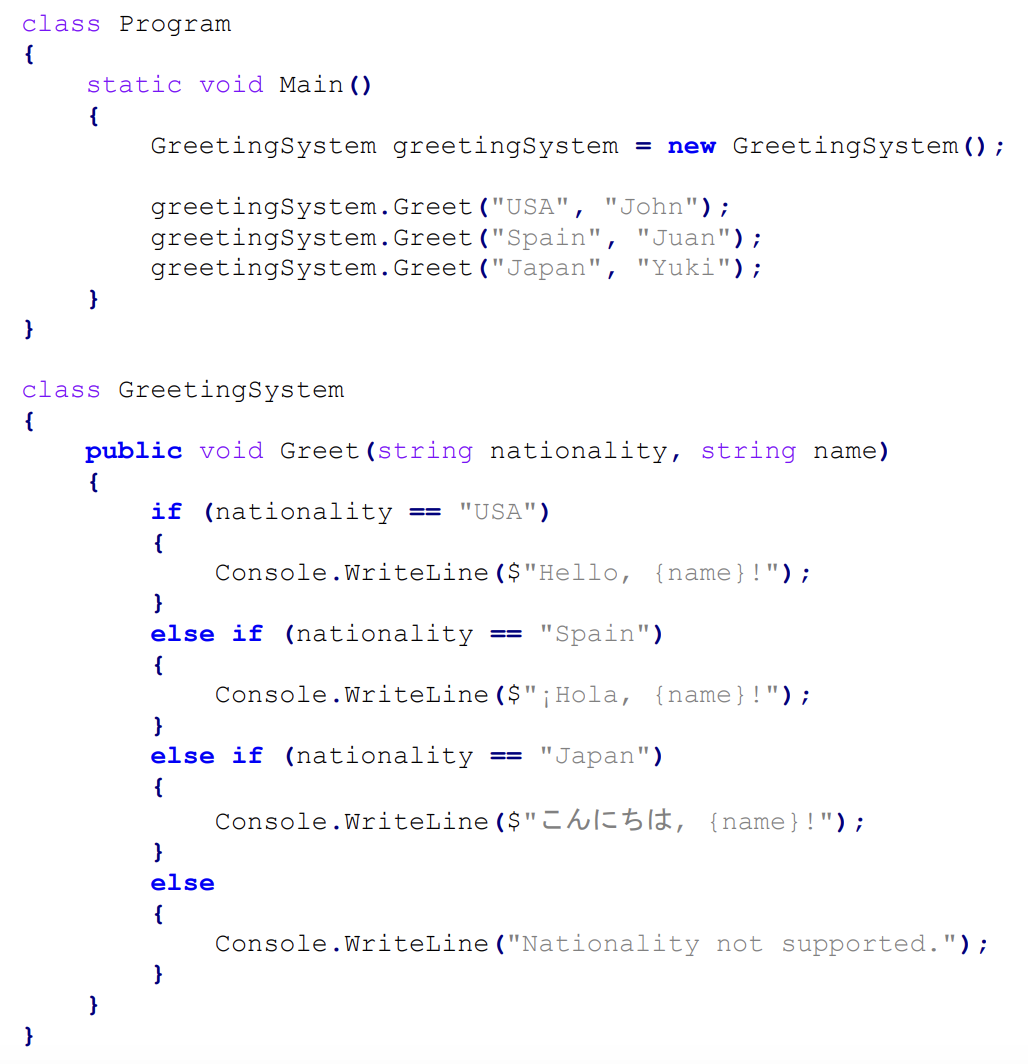
EJERCICIO 9



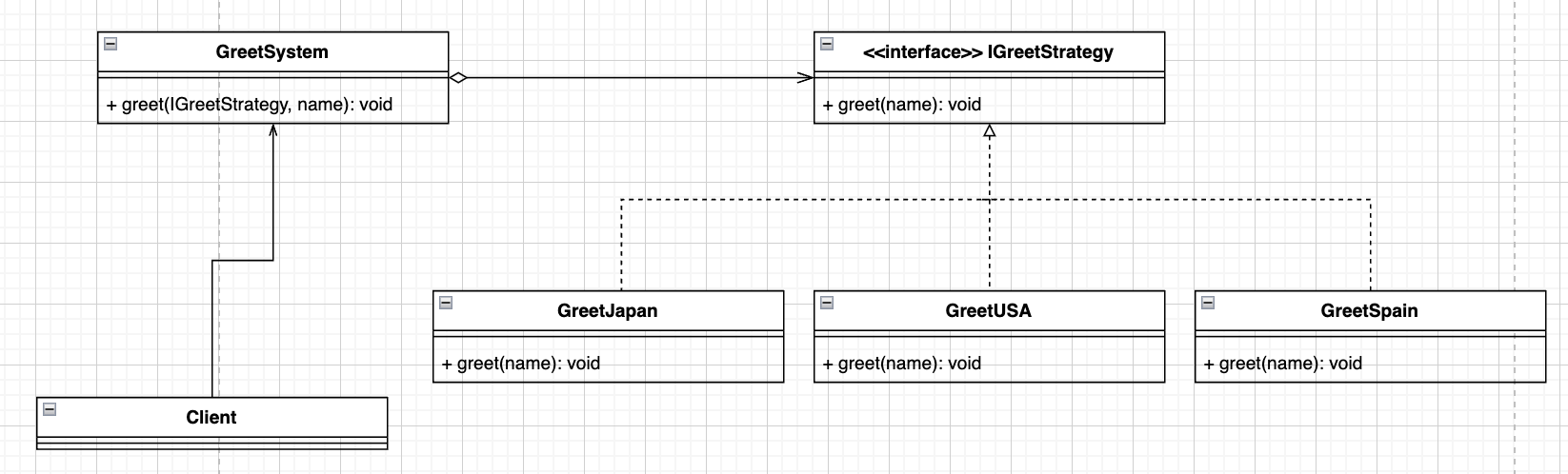


Dado que la letra plantea que Inicialmente, el sistema podría estar usando condicionales para determinar qué nivel de soporte debe manejar una consulta, hay solo un nivel de soporte a la vez. El patron sera **chain of responsibility.** Al recibir una solicitud, cada manejador decide si la procesa o si la pasa al siguiente manejador de la cadena.

EJERCICIO 10



El patrón que soluciona es Strategy porque permite definir una familia de algoritmos, colocar cada uno de ellos en una clase separada y hacer sus objetos intercambiables.



| using System;  public interface IGreetStrategy {  void Greet(string name);  }  public class GreetJapan : IGreetStrategy {  public void Greet(string name) {  Console.WriteLine("こんにちは、" + name);  }  }  public class GreetUsa : IGreetStrategy {  public void Greet(string name) {  Console.WriteLine("Hello, " + name);  }  }  public class GreetChina : IGreetStrategy {  public void Greet(string name) {  Console.WriteLine("你好，" + name);  }  }  public class Greet {  private IGreetStrategy greetStrategy;  public Greet(IGreetStrategy greetStrategy) {  this.greetStrategy = greetStrategy;  }  public void Greet(string name) {  greetStrategy.Greet(name);  }  }  class Program {  static void Main(string[] args) {  Greet greet = new Greet(new GreetJapan());  greet.Greet("小明");  greet = new Greet(new GreetUsa());  greet.Greet("Tom");  greet = new Greet(new GreetChina());  greet.Greet("小明");  }  } |
| --- |

| public interface IArticulo {  String getNombre();  double calculatePrecio();  }  public class Libro implements IArticulo {  private String nombre;  private double precio;  public Libro(String nombre, double precio) {  this.nombre = nombre;  this.precio = precio;  }  public String getNombre() {  return nombre;  }  public double calculatePrecio() {  return precio;  }  }  public class Electronico implements IArticulo {  private String nombre;  private double precio;  public Electronico(String nombre, double precio) {  this.nombre = nombre;  this.precio = precio;  }  public String getNombre() {  return nombre;  }  public double calculatePrecio() {  return precio;  }  }  public class Ropa implements IArticulo {  private String nombre;  private double precio;  public Ropa(String nombre, double precio) {  this.nombre = nombre;  this.precio = precio;  }  public String getNombre() {  return nombre;  }  public double calculatePrecio() {  return precio;  }  }  public abstract class Seccion {  public IArticulo Mostrar(String nombre, double precio) {  return createArticulo(nombre, precio);  }  public abstract IArticulo createArticulo(String nombre, double precio);  }  public class Libros extends Seccion {  public IArticulo createArticulo(String nombre, double precio) {  return new Libro(nombre, precio);  }  }  public class Electronicos extends Seccion {  public IArticulo createArticulo(String nombre, double precio) {  return new Electronico(nombre, precio);  }  }  public class Ropas extends Seccion {  public IArticulo createArticulo(String nombre, double precio) {  return new Ropa(nombre, precio);  }  }  public class Program {  public static void main(String[] args) {  Seccion seccion = new Electronicos();  IArticulo a = seccion.Mostrar("Teléfono", 500.0);  System.out.println(a.getNombre());  System.out.println(a.calculatePrecio());  }  } |
| --- |