Ejer 4 → Cálculo de Sueldos

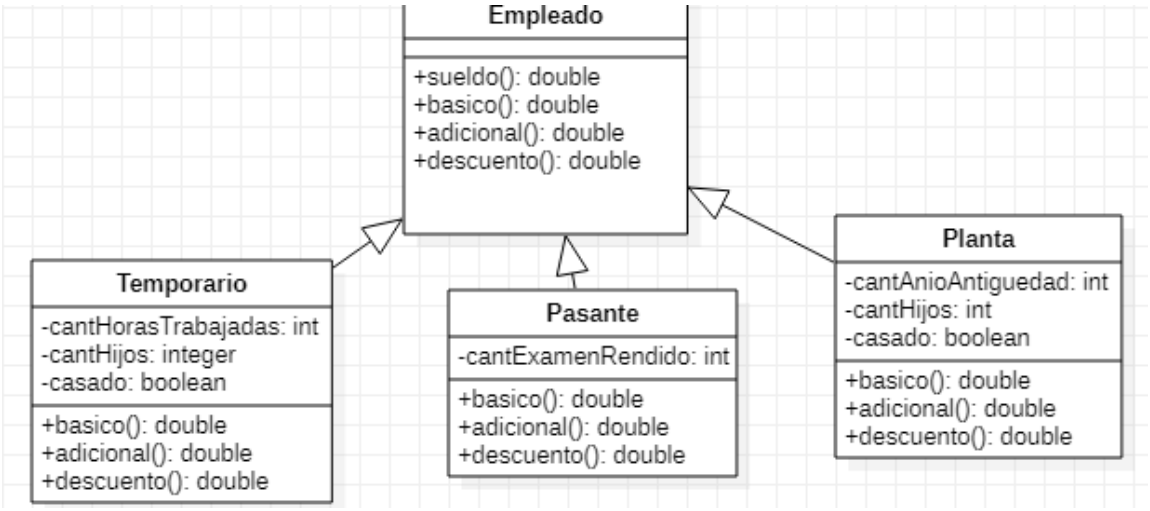
Consultar

Si no hay v.i es una interfaz?

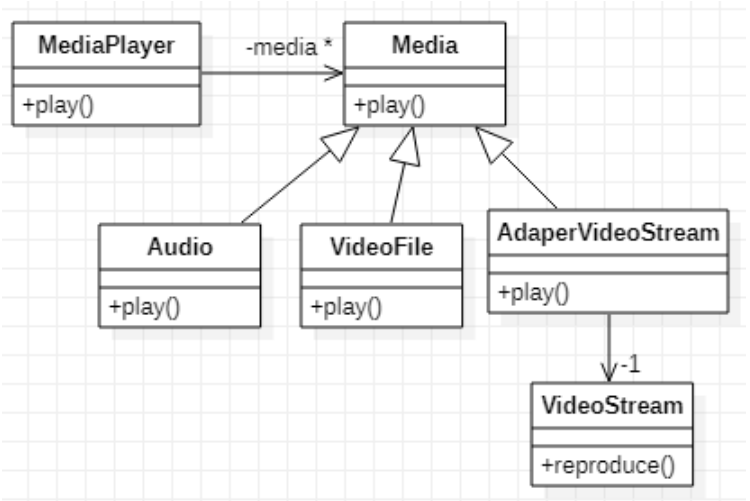
En este caso hice a empleado como clase abstracta para hacer el template method en sueldo(). Osea obligar a las subclases a implementar básico(), adicional() y descuento()

La interfaz no tiene código (no tiene v.i)

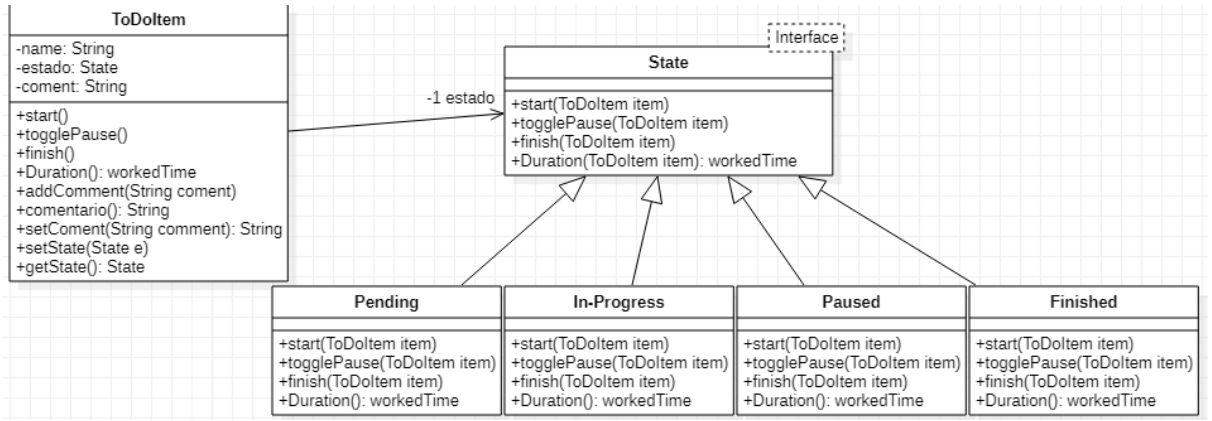
La clase abstracta puede o no tener v.i



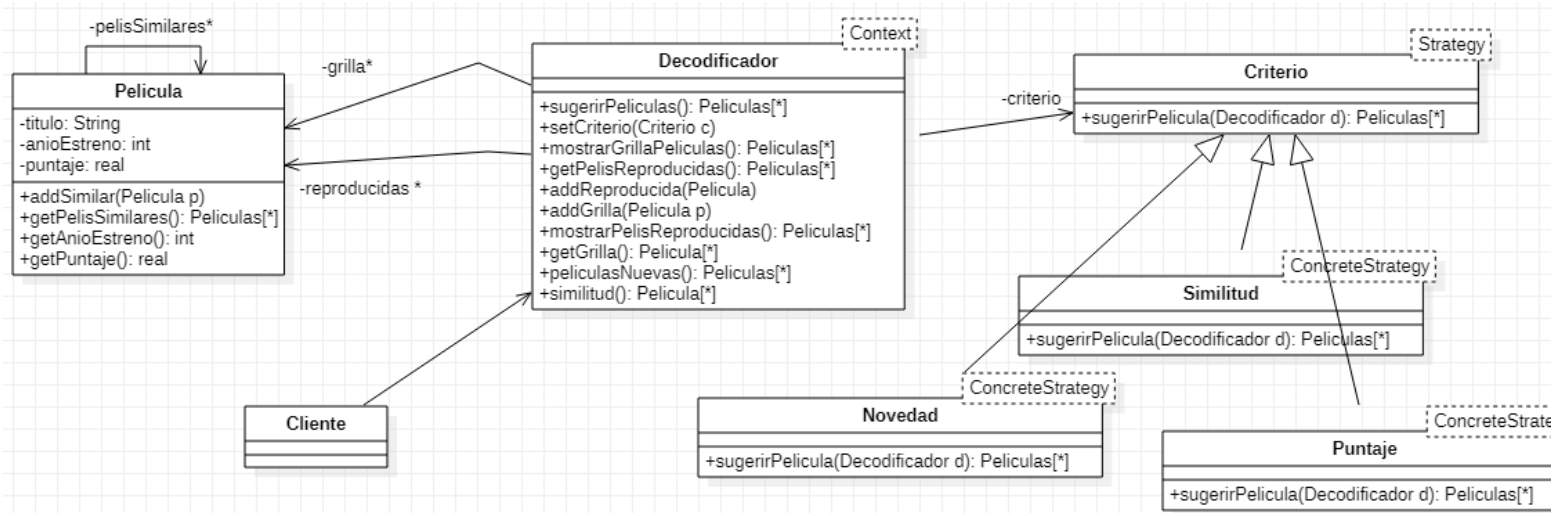
Ejer 5



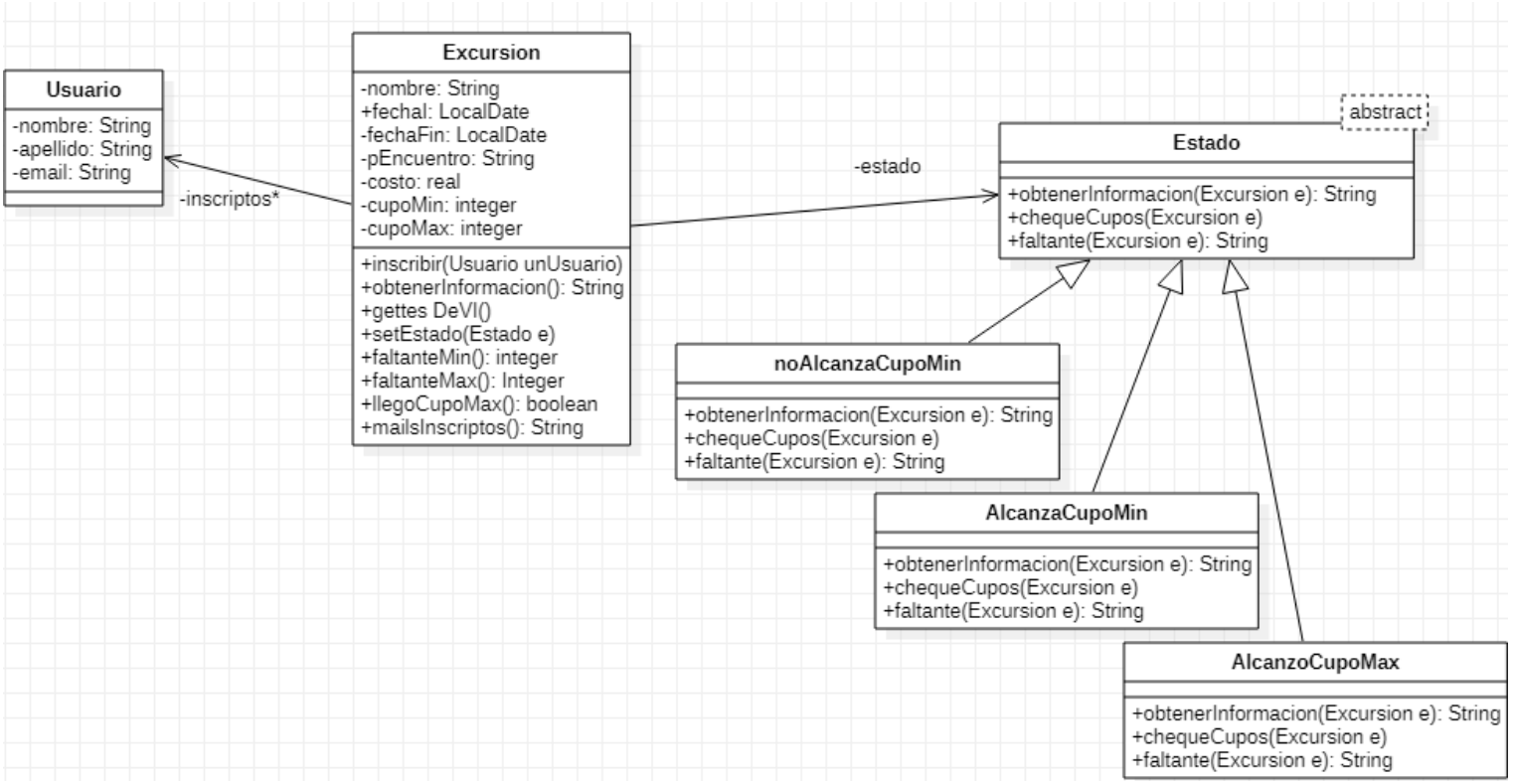
Ejer 6 todos



Ejer 7: Decodificador de películas

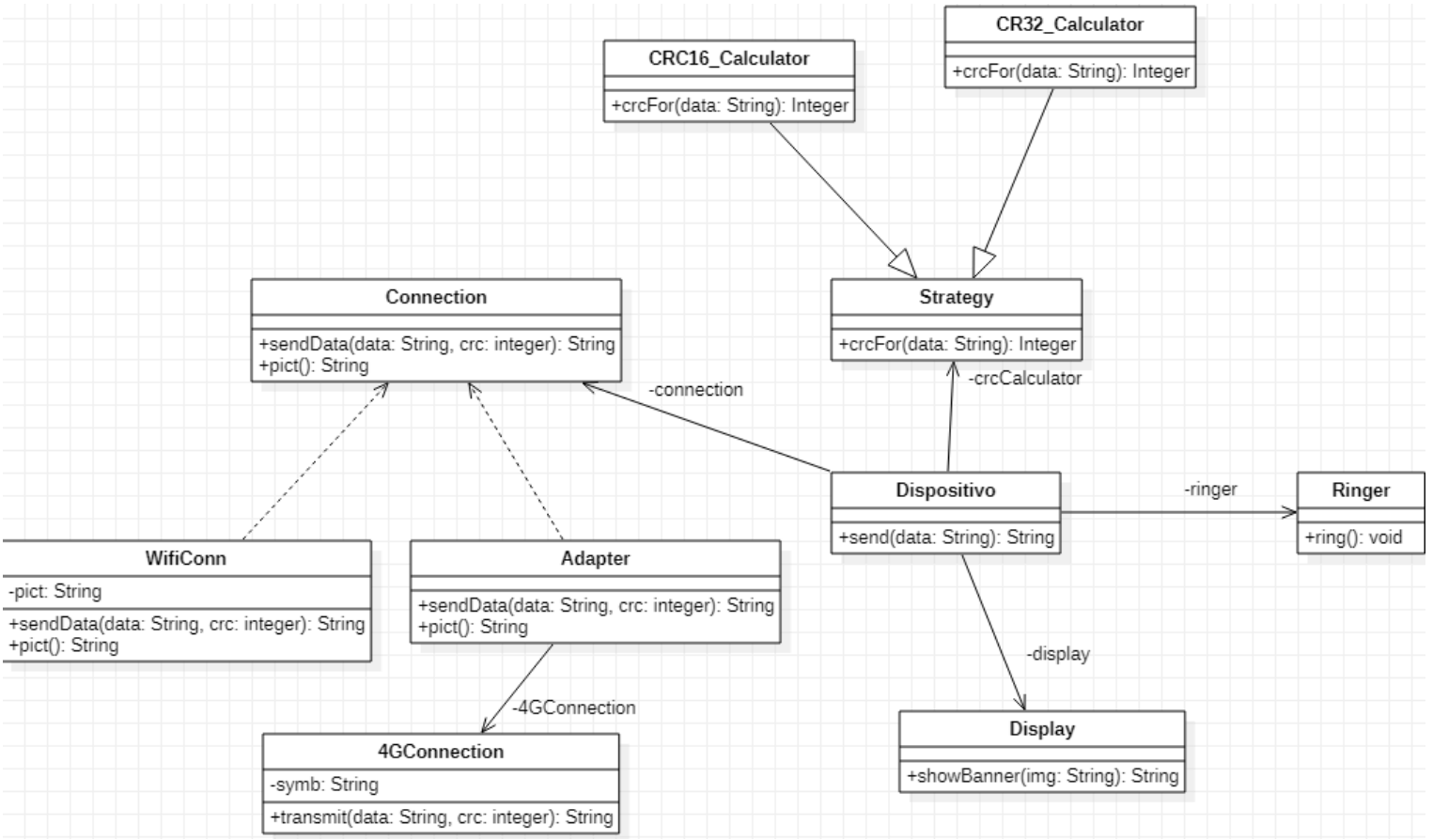


Ejer 8 Excursiones

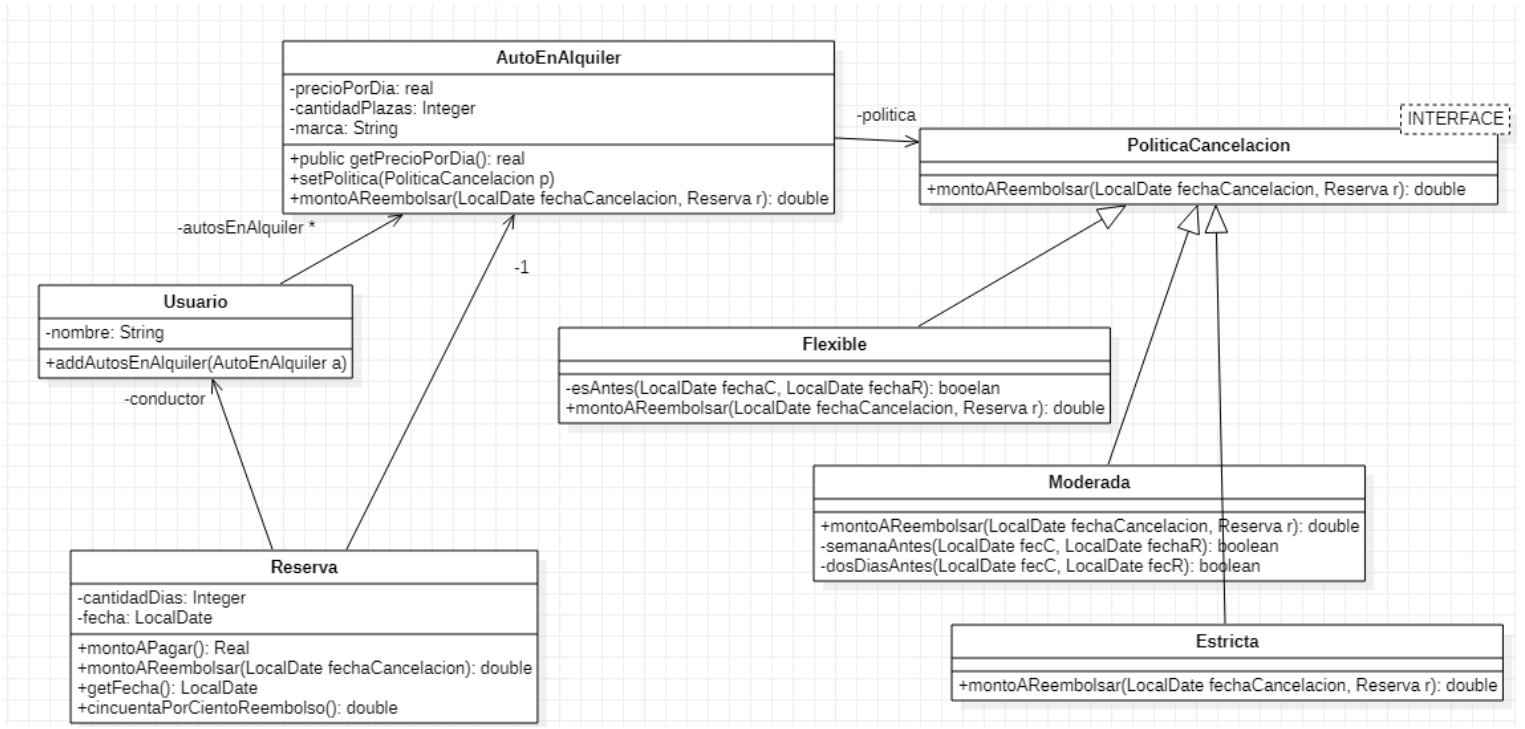


Ejer 10: Dispositivo móvil y conexiones. Consultar implementación y UML, paso test

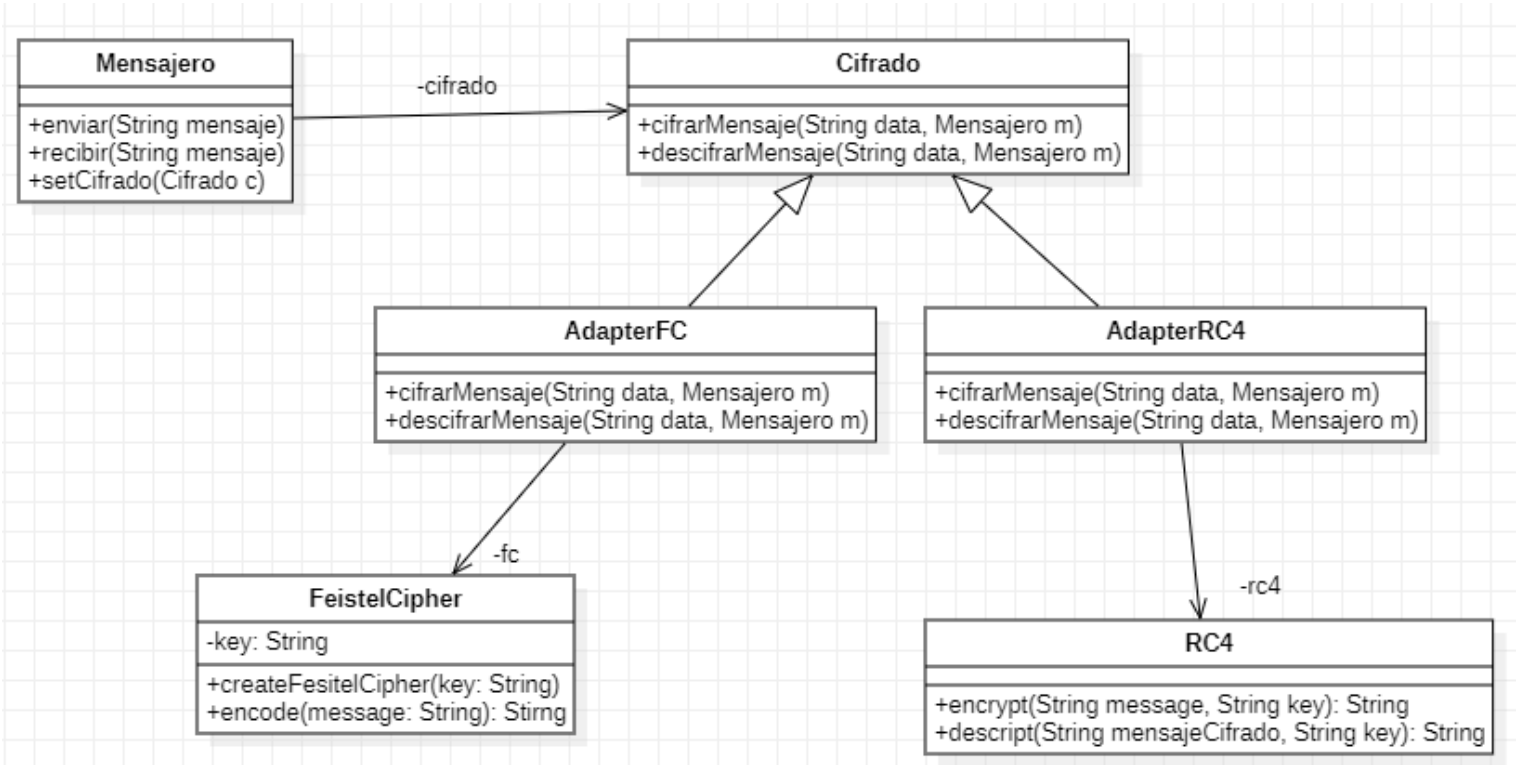
Y



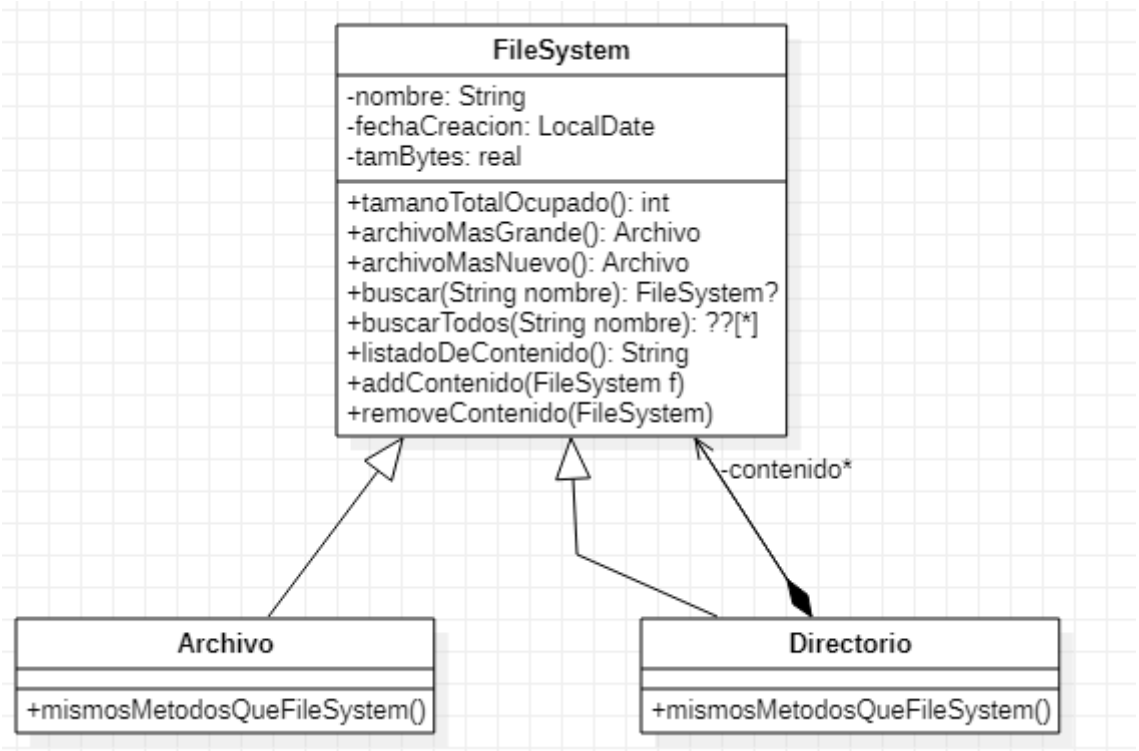
Ejer 11 Alquiler de automóviles



Ejer 12 Mensajero - No está terminado, falta test. El código si está listo, revisar.
Se usó Adapter & Strategy

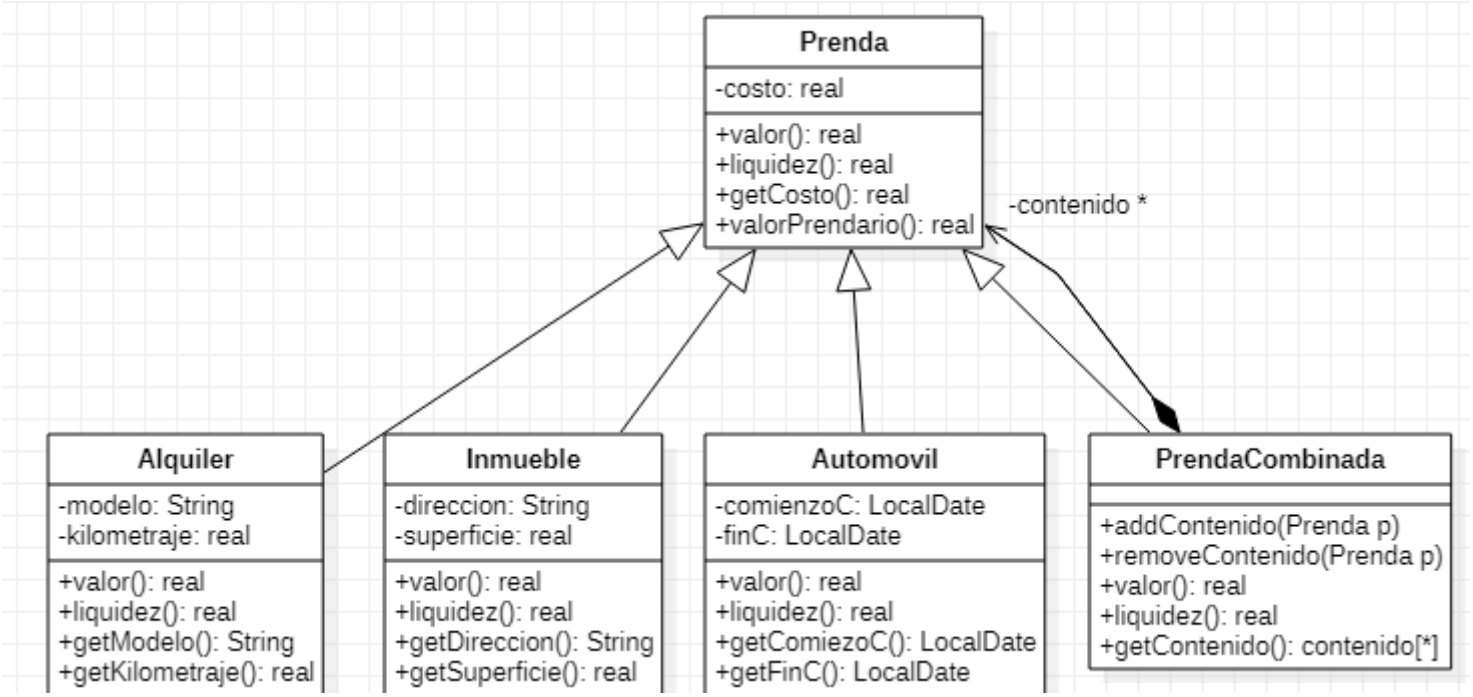


Ejer 14: FileSystem

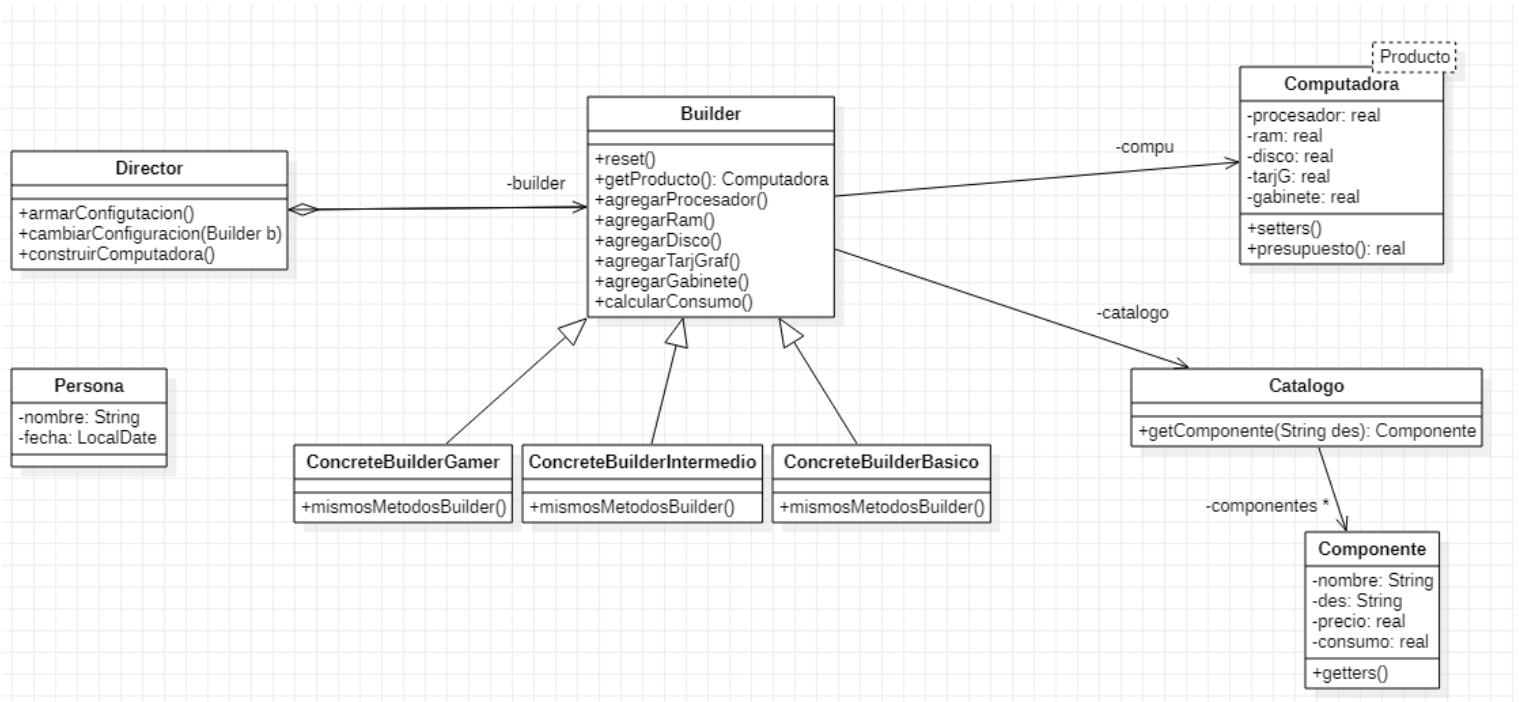


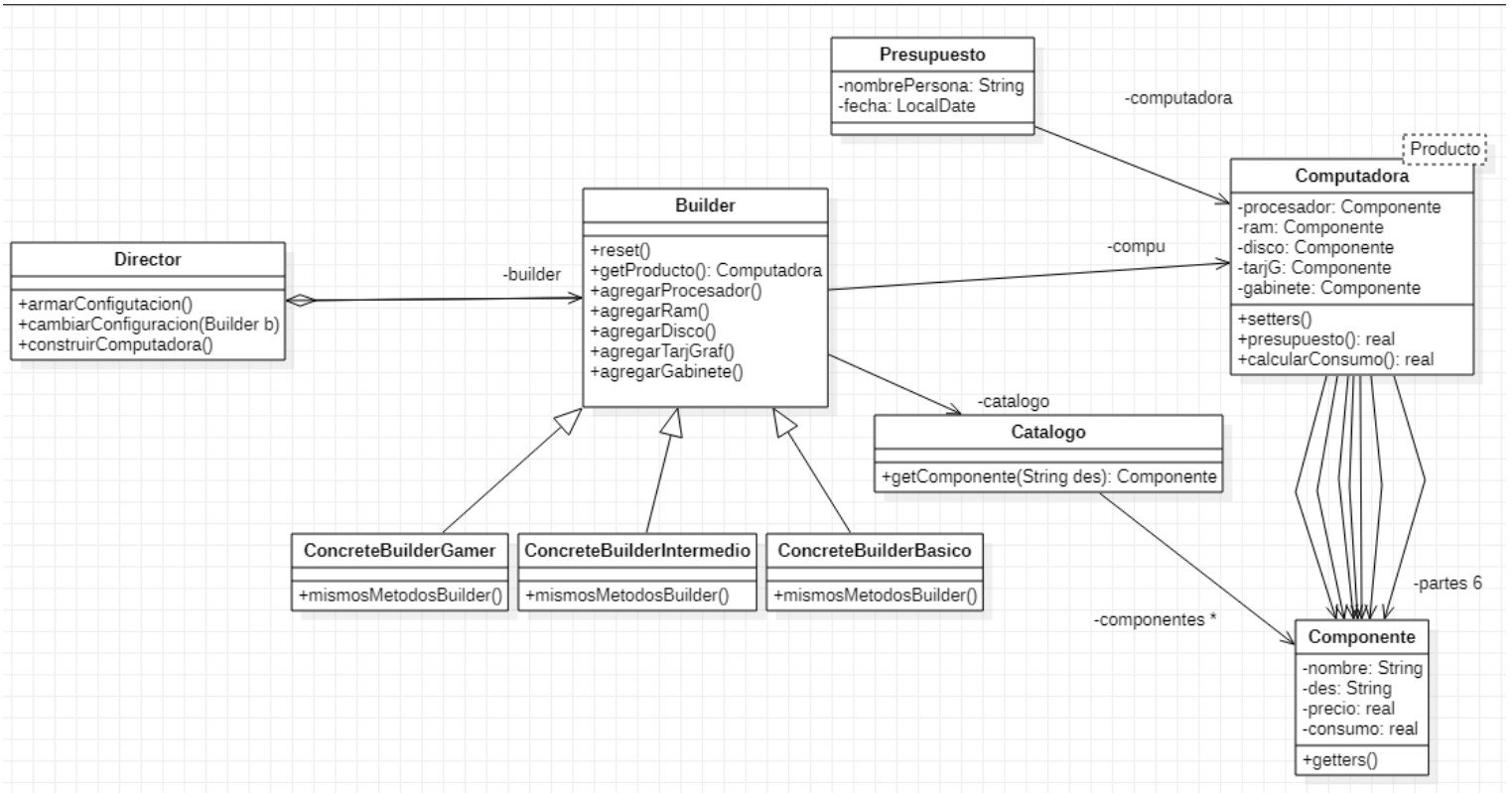
Ejercicio 16: Préstamos Prendarios. En este caso elegí seguridad así que los métodos de add y remove los pongo en el compuesto. La implementación consultar que está bien pensada.

Hay un template en prenda



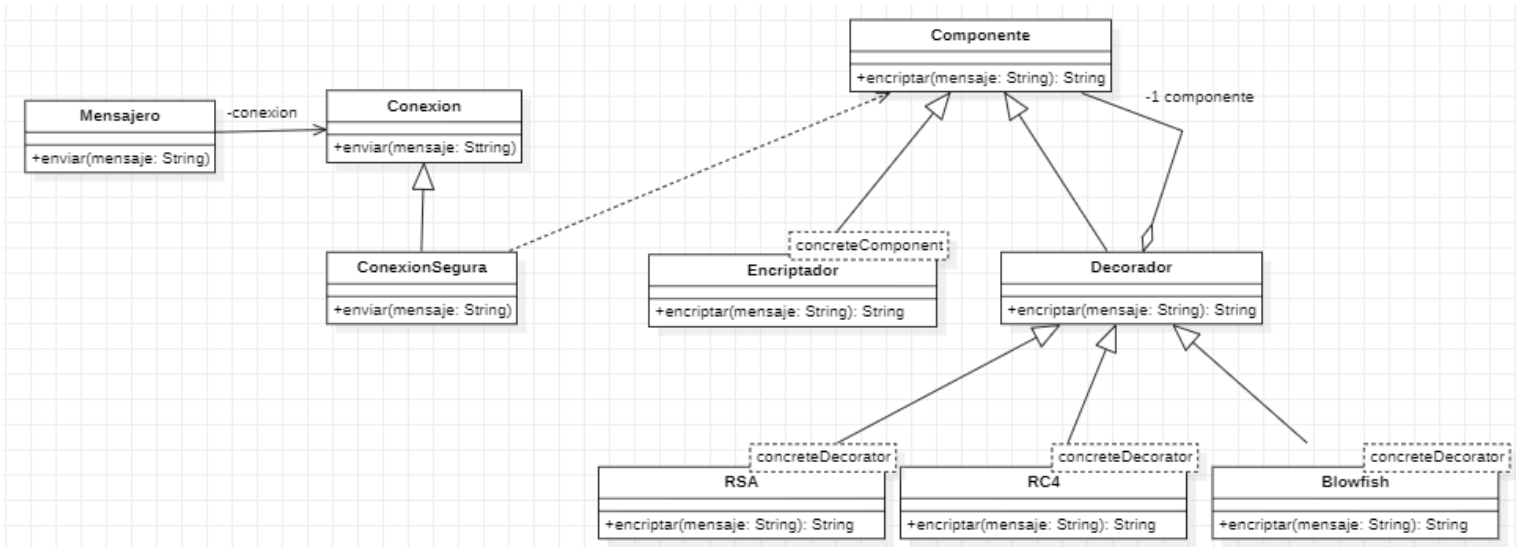
17: Builder
Builder: especifica una interfaz abstracta para crear las partes de un producto.
Producto: objeto completo a ser construido
Director: Conoce los pasos para construir el producto. Utiliza el Builder para construir las partes que va ensamblando.Sigue una receta, template method?
Concrete Builder: implementa la interfaz Builder para construir y ensamblar las partes del producto. Define la representación a crear. Proporciona una interfaz para devolver el producto.



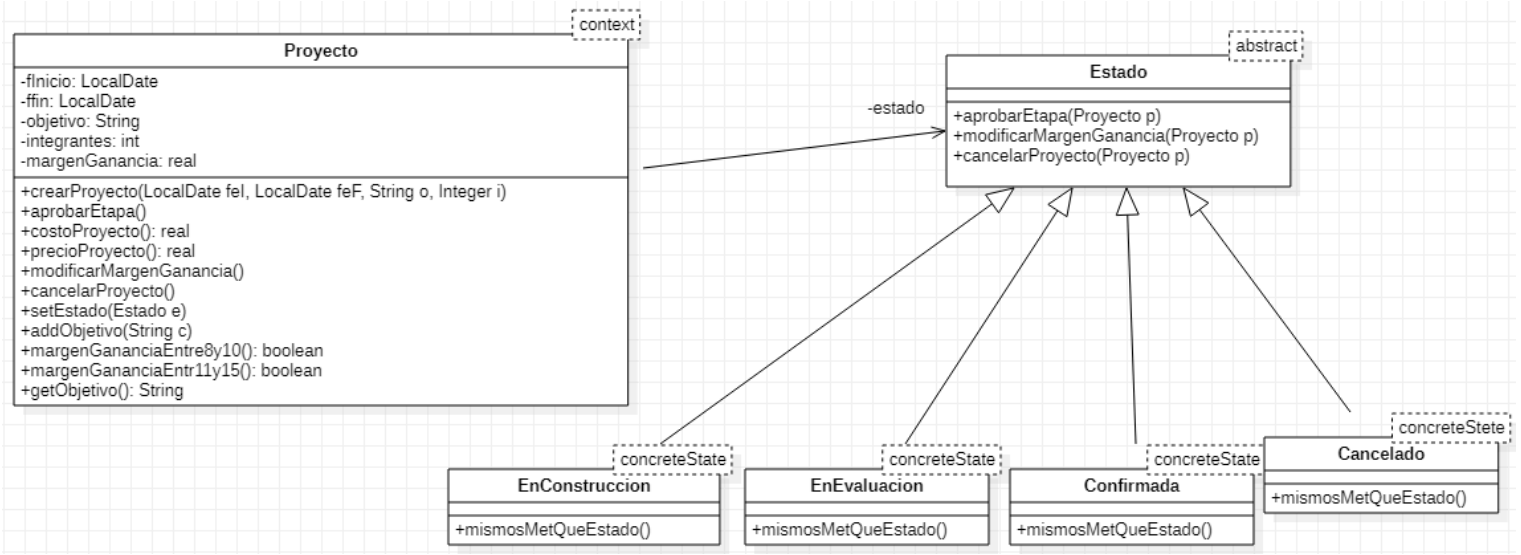


De computadora a componente puede ser una lista pero la tenes que limitar a 6 componentes y que cada componente sea uno distinto al otro.
La representación de las v.i estan mal, no deberían estar, dado que puse 6 flechas de computadora a componente, pero para no poner al final de la flecha el nombre de cada v.i lo deje asi.

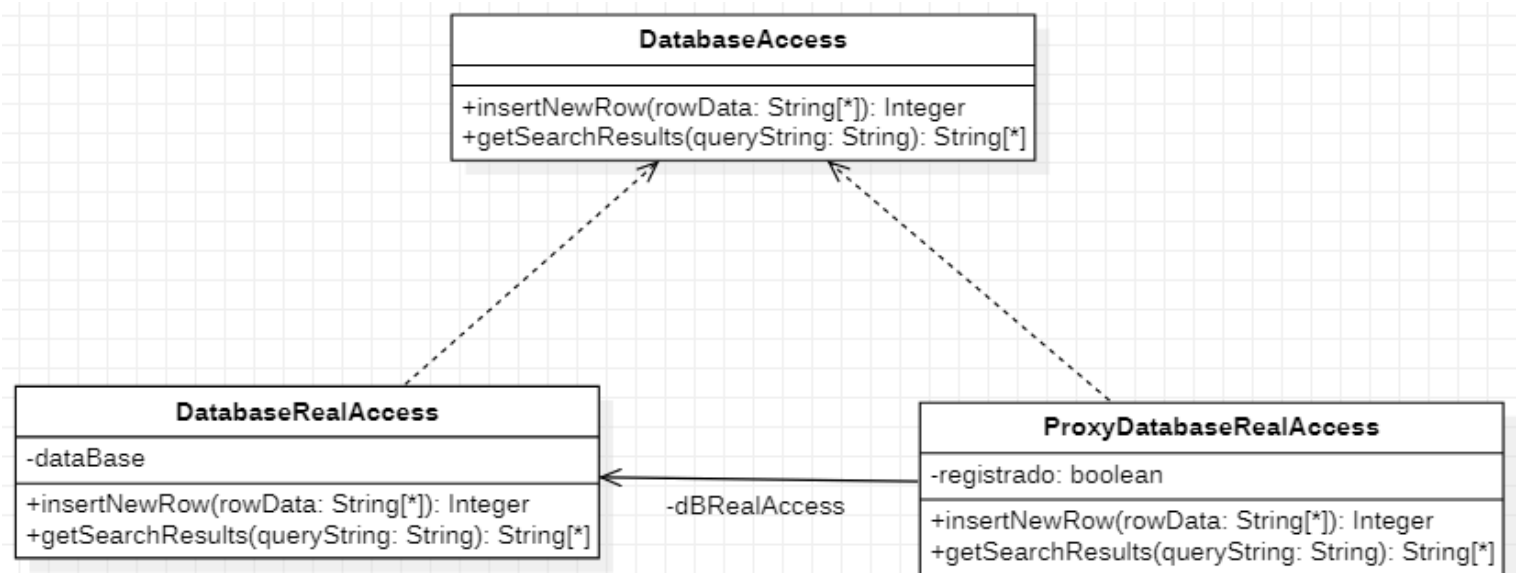
Ejer19 Encriptador.
Observaciones: puede ser un Strategy.
Elegí decorator porque en el enunciado dice" Modifique el diseño para que el objeto Encriptador pueda encriptar mensajes usando los algoritmos Blowfish **y** RC4, además del ya soportado RSA." Supuse que ese "**y"** representaba mas de uno a la vez. Esta consultado y esta bien esta forma.
Otra cosa para ver es que el encriptador se puede transformar en un nullEncritpador ya que es una clase que no hace nada. La tarea de encriptar la hacen los concreteDecorator.



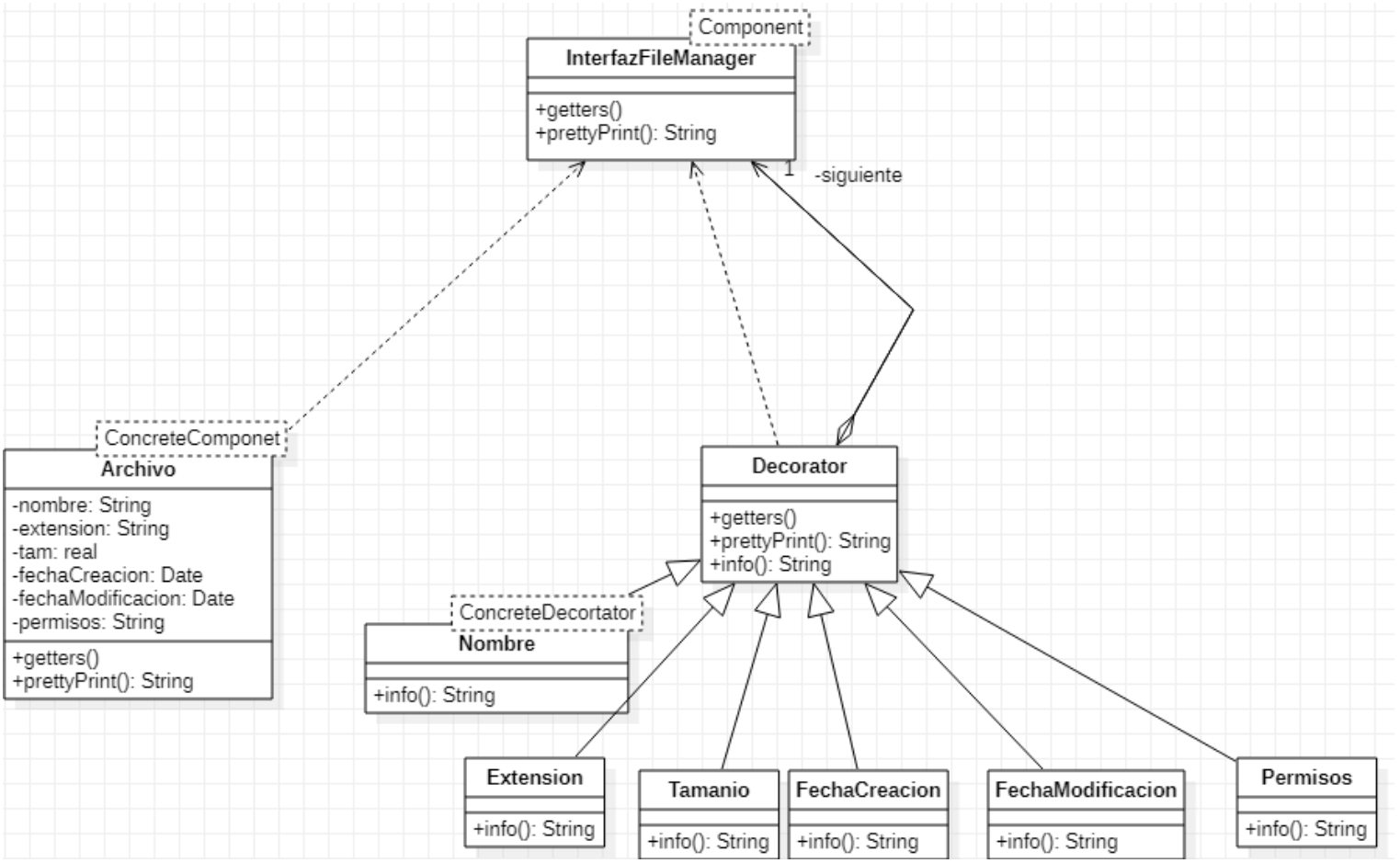
Ejer 20- Administrador de proyectos.
Observación: EL ESTADO CANCELAR es importante, no es obvio y sin ese estado el ejercicio esta desaprobado.
Consultar código, varios estados no hacen nada.



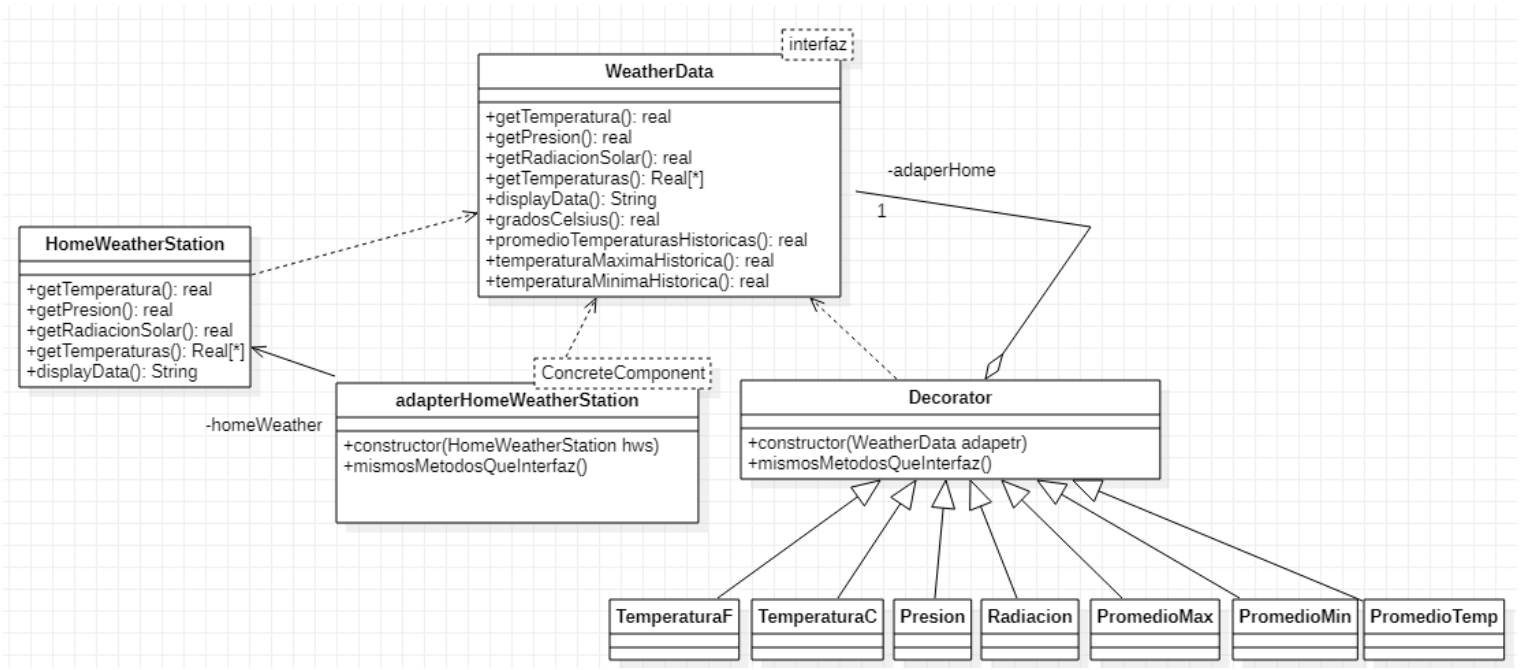
Ej 21: Acceso a la base de datos. Corregir codigo ProxyDatabaseRealAccess, use proxy de seguridad. El test no pasa, no lo toque, es el que venia en el ejercicio



Ejer 22: File Manager

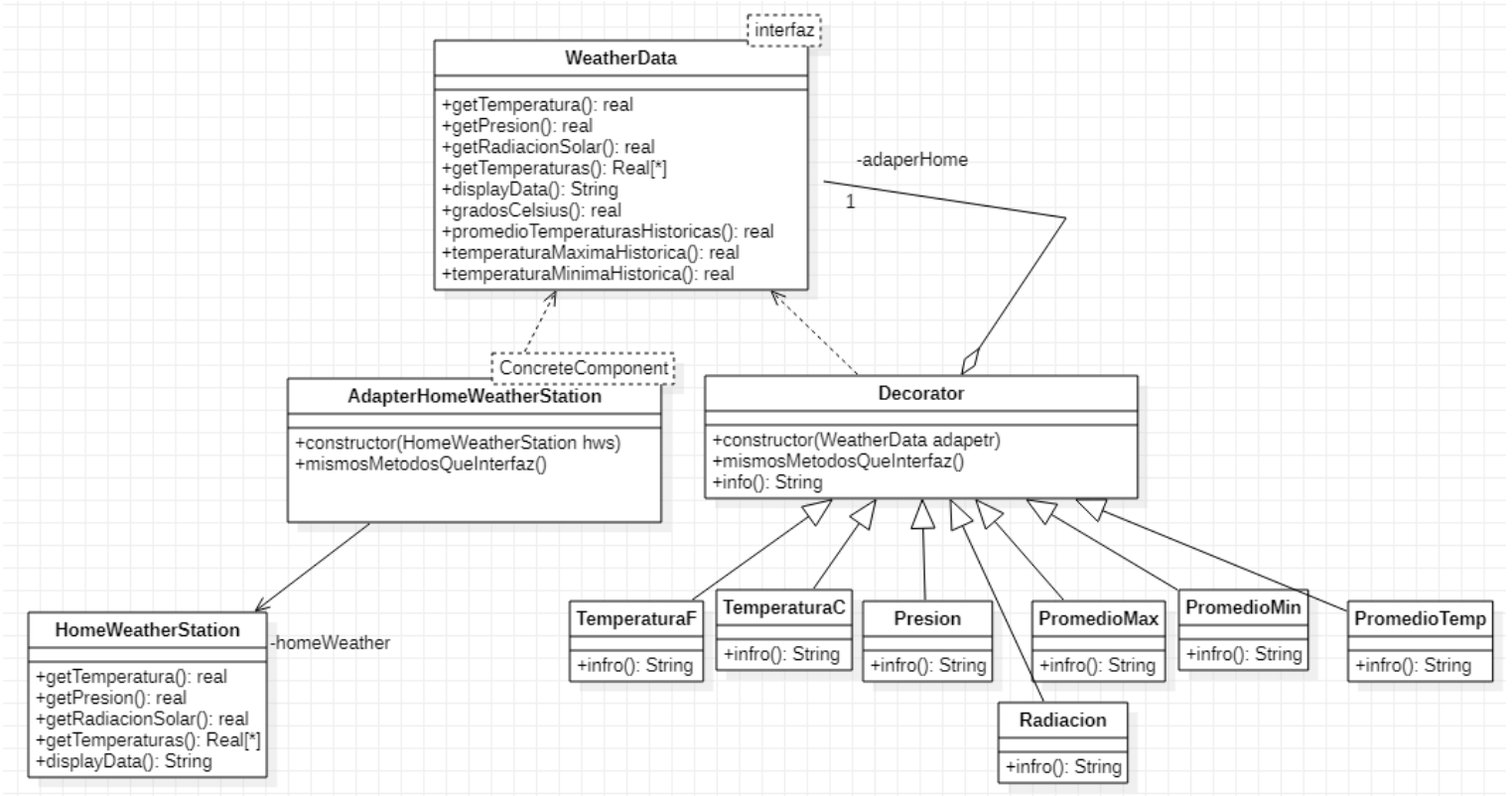


Ejer 23



No se como hacerlo porque los métodos nuevos que meti en la interfaz no los debería implementar la clase HomeWeatherStation porque no puedo modificarla, entonces no se donde meterlos para que adapter pueda funcionar con la interfaz WeatherData MAS los nuevos métodos .

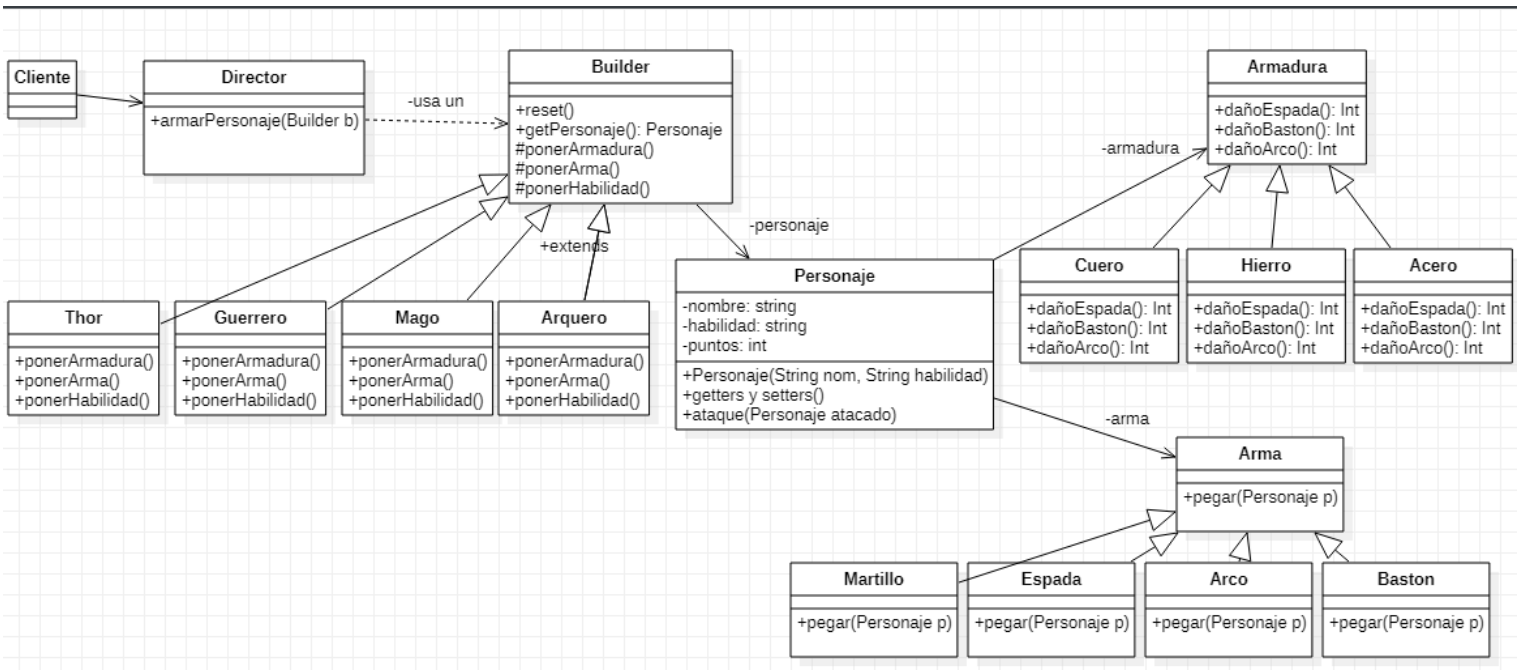
Opte por hacer esto: HomeWeather no usa mas esa interfaz.



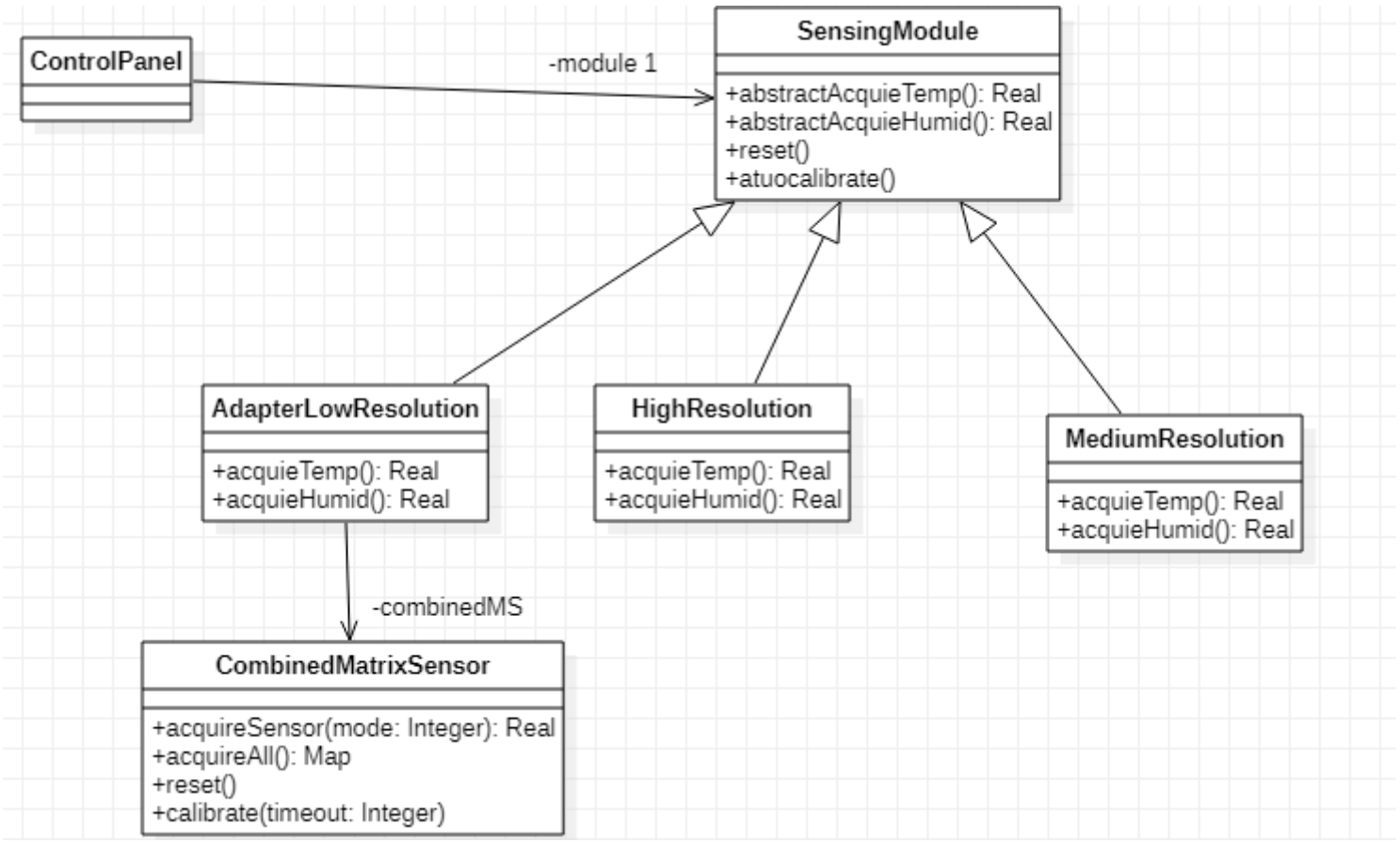
Ejer 24 Productos Financieros

1. crearía un director para crear al producto nuevo y el código iria ahí. Que llama a la membresiaBuilder (Crear un nuevo director, nueva receta) (hay 4 directores, uno por producto)
2. Nueva receta nuevo director
3. haciendo que la membresía sepa que builder instancias. (builder va a crear el composite)
4. desventaja: muchas clases
5. ventaja:

Ejercicio 25: Personajes de juegos de rol



25. CONSULTADO corregido x alejandra



1.Veo que la libreria del nuevo sensor tiene problemas con la interfaz de SensingModule que es la que el sistema usa. Hay que adaptarlo para poder implementar el nuevo sensor Low Resolution

2. Aplicaria un adapter. Consecuencias: puede agregar complejidad en codigo, se pierde transparencia dado que el cliente no sabe que esta interactuando con una clase adapter.

3. Codigo resultante:

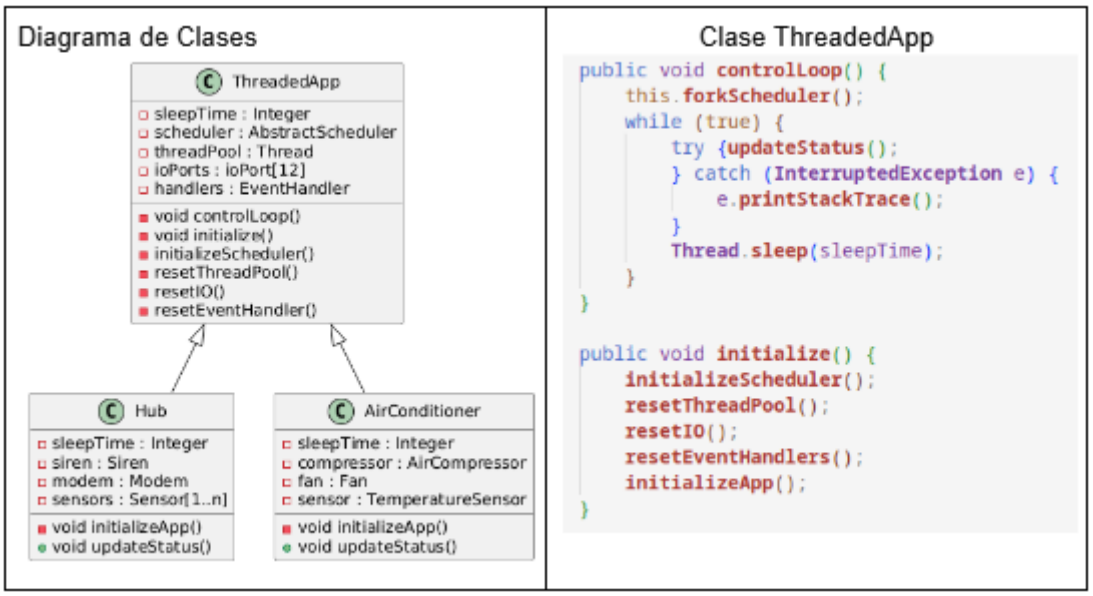
```
public class AdapterLowResolution extends SensingModule{

        private CombinedMatrixSensor combinedMS;

public AdapterLowResolution(CombinedMatrixSensor combinedMS)
     this.combinedMS = combinedMS;
}

public double acquieTemp(){
  return this.combinedMS.acquireSensor(1);
}
public double acquieTemp(){
  return this.combinedMS.acquireSensor(2);
}

public double reset(){
  return this.combinedMS.reset();
}
```

Ejer 30:Simulacion: CONSULTADO

a) 1. hay un esqueleto en la super clase 2. Los métodos pueden redefinirse en las subclases 3. **Y NO SE REDEFINE el metodo addEntity(..)**

b) Razones por la que pienso que no se logro aplicar un template method 1. No es una “receta” que todas las subclases pueden seguir, el metodo stop no lo puede ejecutar RTSimulation porque no lo tiene definido y la superclase tampoco. 2. pasa lo mismo con el metodo start() 3. El unico metodo que sobreescribe RTSimularion es register(..) 4. no estan definidos como abstractos los metodos stop y start que son lo que se deben llevar a cabo y no estan el la superclase implementandos, deberian estar en simulation como abstractos obligando a las subclases a redefinirllos

Ejer 31: ThreadedApp



1. controlLoop(): llama a la funcion forkScheduler() que es la planificadora de hilos. Entra en un bucle infinito while(true), intenta ejecutar el metodo upStatus() que es definido por las subclases Hub y AirConditioner. En casio de que salte la excepcion InterruptedException(..) imprime un mensaje y luego duerme al hilo por un tiempo determinado (sleepTime) y vuelve al try.. La unica forma que no se ejecute Thread.sleep(sleepTime) es si updateStatus() lanza una exception que no sea InterruptedException() ej: un nullPointerException.
2. initialize(): inicializa los panificadores, resetea los hilos, resetea las e/s, reinicia e inicializa el sistema. Prepara todo para el uso.
3. HOTSPOTS: partes variantes o puntos de extension del framework. En este caso son: initializeApp() y updateStatus()
4. Inversion de control: es cuando el framework (la superclase) llama a los metodos de la aplicación (sublcases). Si hay inversion de control en el metodo controlLoop() con la llamada a updateStatus() y en el meotdo initializeApp() en la llamada initializeApp(). Ambos metodos se encuentran en la clase ThreadedApp
5. consutlado **¿Cómo se instancia el framework?**

   El FW se esta extendiendo meditante la herencia. Lo que se instancia es una de las clases concretas de la app que usan el fw, como Hub o AirConditioner. Ejemplo de una instancia de aplicacion:

   AirConditioner ac = new AirConditioner(100, new Compressor(), new Fan(), new TemperatureSensor(c)); //ESTO NO, esta mal .

   Rta:
   Se instancia creando la clase AirConditioner haciendo que extienda de ThreadedApp (Felipe)

   La Instanciación del Framework: En teoría de frameworks, "instanciar el framework" significa **crear una aplicación concreta** que utilice ese esqueleto.
   Como el esqueleto está incompleto (tiene métodos abstractos o *hooks* que deben llenarse), no puedes usarlo "desde afuera" simplemente creando objetos. Debes meterte "adentro" (herencia) para completar la lógica. Por eso se dice que se instancia **creando una clase que extienda la base**.
   Si solo haces `new AirConditioner(...)`: Estás creando un objeto en tiempo de ejecución.
   Si haces `class AirConditioner extends ThreadedApp`: Estás **definiendo la lógica** que llena los huecos del framework. Sin este paso, el framework no sirve para nada, es solo un esqueleto muerto.

Resumen: La diferencia conceptual

Concepto	`new AirConditioner(...)`	`class AirConditioner extends ThreadedApp`
Qué es	Instanciación de un **objeto** (Runtime).	Especialización / Instanciación del **Framework** (Design time).
Rol	Es el uso final del objeto.	Es la construcción de la aplicación concreta sobre el framework.
Por qué es la Rta	Hacer `new` es inútil si antes no definiste *qué* hace esa clase dentro del framework.	**Esta es la respuesta correcta** porque es el mecanismo mediante el cual el framework genérico se convierte en una aplicación específica.

Y SI FUERA POR COMPOSICION EN LUGAR DE HERENCIA COMO SERIA?

Característica	Framework Caja Blanca (Herencia)	Framework Caja Negra (Composición)
Patrón Principal	**Template Method**: El padre define el esqueleto, el hijo completa los pasos.	**Strategy**: La clase principal delega el comportamiento a una interfaz intercambiable.
Relación	`AirConditioner extends ThreadedApp`	`ThreadedApp` tiene un `AppLogic`
Instanciación	Se crea una instancia de la **Hija.**	Se crea una instancia del **Framework** y se le pasa la lógica por constructor.
Flexibilidad	Menor (se define al compilar).	Mayor (puedes cambiar la lógica en tiempo de ejecución).

La instanciación se hace pasando tu lógica al motor del framework:

Java

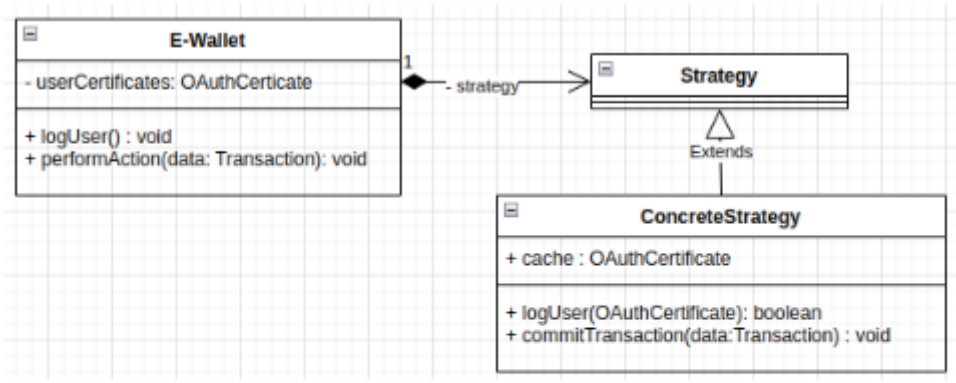
```java
// 1. Creo mi pieza de lógica
AppLogic miLogicaAire = new AirConditionerLogic();

// 2. Instancio el framework "inyectándole" mi lógica
ThreadedApp app = new ThreadedApp(miLogicaAire);

// 3. Arranco
app.run();
```

6. consutladoEs un framework de caja blanca porque requiere que el desarrollador use la herencia y redefina/ sobreescriba los metodos hook para agregar funcionalidad específica.
7. consutlado Test de unidad : si se pueden implementar los test de unidad requeridos para corroborar que los metodos son enviados en la secuencia esperada. Esto se haria con test double mock para la clase AirCompressor
   AirConditioner ac = new AirConditioner(100, new **MockAC**(), new Fan(), new TemperatureSendor(c));
   El objetivo del test es verificar el comportamiento de la clase bajo prueba (AirConditioner) al reemplazar la dependencia real (AirCompressor) con un MOCK. El Mock es el unico test double que se enfoca en verificar interacciones, protocolos y expectativas de secuencia.

   - Test stub → solo devuelve datos ej: getEnergy {return 200;} El stub solo sirve para que el SUT envie los mensajes, no para controlar ni verificar el orden de llamadas
   - Test spy → requiere verificación manual ; solo registra, uno mismo debe verificar manualmente el registro (buscar ejemplo)
   - Mock → se configura con expectativas de las llamadas y falla si el orden o los argumentos son incorrectos
   - Fake object → simula funcionalidad completa, no verifica protocolo de llamadas. (buscar ejemplo)

Ejercicio 32 E-WALLET- RESPUESTAS consultado

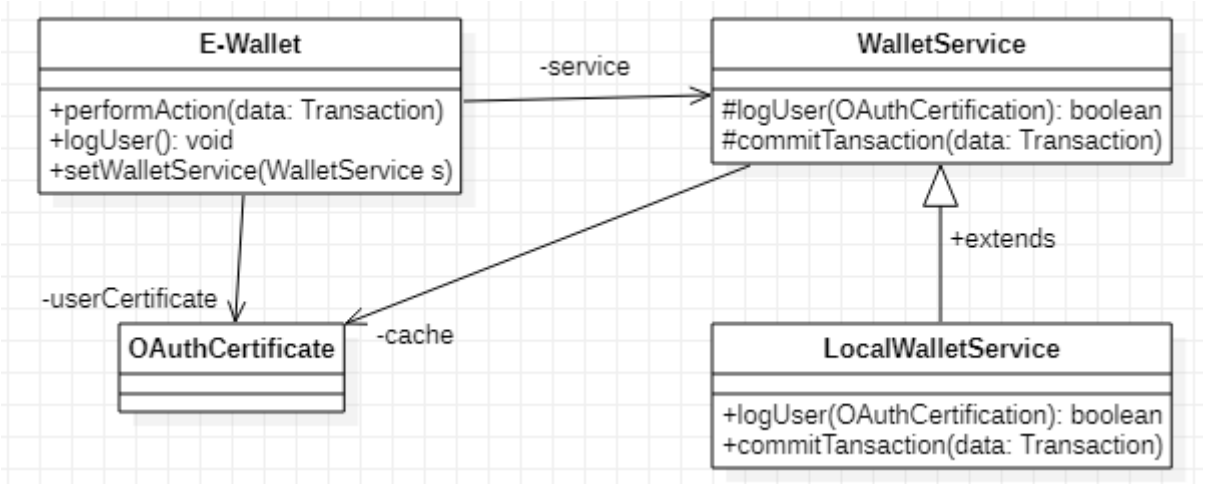
a. No, tiene nombres muy genericos. E-Wallet esta bien pero Strategy y concreteStrategy deberían llamarse diferentes, algo que sea parte del dominio del problema. Strategy fijarme otro nombre y ConcreteStrategy LocalWalletService



E-Wallet::logUser()
If (strategy.logUser(userCertificates))
then this.openScreen();
else this.reportLogingError();
	E-Wallet::performAction(d:Transaction)
	If (userCertificate not null)
	then strategy.commitTransaction(d);

b. No, la clase Strategy esta vacia. Para que E-Wallet pueda ejecutar strategy.logUser(), la clase Strategy debe declarar esos metodos como abstractos (o ser una interfaz). Actualmente el codigo del contexto no compilaria. Tambien falta el metodo setStategy en el contexto para inyectar la estrategia (o pasarlo por el constructor)
c. No. hay violacion de cohesion: la estrategia esta haciendo 2 cosas que no tienen nada que ver (Autenticar (logUser) y Transaccionar (commit..). Un patron Strategy debería encapsular un algortiom o familia de comportamientos por ejemplo solo formas de pago, no mezclar lógica de seguridad con logica de negocio.
d. Si, estado de la estrategia:
   i. la ConcreteStrategy tiene un atributo cache. Las estrategias suelen preferirse sin estado para poder ser compartidas o intercambiadas sin efectos secundarios.
   ii. Logica del contexto: E-Wallet hace if(userCertificate not null) antes de llamar a la estrategia. El contexto debería delegar mas y decidir menos.

e.



NO DEBERÍA HACER UN PROXY?

Ejer 33: Cifradores de texto:

A. Estan tratando de implementar el patron Composite. Elementos que lo confirman: la forma de estructurar del uml, es decir estan las leaf, el composite y el componente que es quien brinda una interfaz uniforme para los objetos simples y el compuesto y la relación de agregación entre MultiCypher y Cypher.
B. Las operaciones Cipher.add(Cypher):Cipher , Cipher.cipher(String) y Cipher.decipher(String) estan marcadas como abstractas porque cada clase lo implementa diferente, y al ser abstracto obliga a que esten implementados en todas las subclases, pudiendo tratar a los objetos de manera uniforme. En cuanto si esta bien o no si, considero que si, dado que se esta priorizando en el caso del add() la transparencia en lugar de la seguridad.
   rta feli: No, add no debería ser abstracto. Porque todas las hojas lo implementan de la misma forma entonces se repetiria codigo.

C.

En la clase Cipher	Class MultiCypher extends Cypher {//chequear

public Cypher add (Cypher c){
MultiCypher mc = new MultiCypher(this)
mc.add(c);
return mc;
	//Sobrescribe el método del padre
	
	@Override
	public Cypher add(Cypher nuevoCifrador) {
	
	//En este caso NO creamos uno nuevo, solo acumulamos
	
	this.hijos.add(nuevoCifrador)
	return this
	
	}

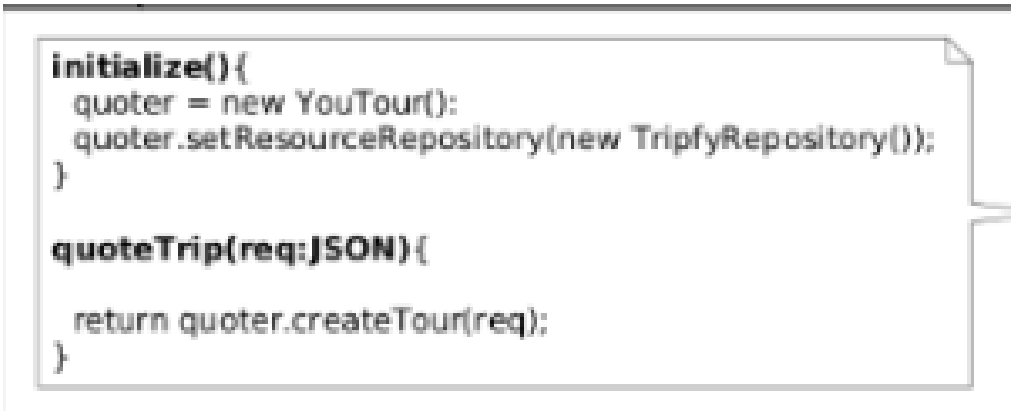
D.

Puntos correctos	Puntos incorrectos
- Se cumple perfecto “la posibilidad de utilizar cifradores combinados, los cuales se basan en otros cifradores, simples o complejo” - Es transparente para el cliente los objetos simples y los compuestos, el no sabe con cual trata.	- MultiCipher no tiene la v.i lista en la relacion de agregacion - Puede ser add abstract y remove no?

E.

Ejercicio 34: YouTour:

1. YouTourFramework es un framework de caja blanca porque hay herencia. **CONSULTAR, que deberia pasar para que no sea un framework? Para que deje de ser un framework y pase a ser una simple Librería o API, debe perder la Inversión de Control (IoC).**



2. La jerarquia TourCreator usa HotSpots para entender el framework ya que son todos metodos abstractos que van a redefinirse en las subclases LowCostoCreator y HighEndCreator
3. Si se quiere agregar la posiblidad de crear Tours sin transporte se deberia crear una nueva clase que funcione como director ya que si cambio los pasos cambio al director.

RTA CORRECTA //Corregido
Crear un nuevo director que ignore inyectTransport nada mas.
El director decide que usar y que no.
No importa que el builder tenga muchas cosas, el director usa lo que necesita. ←(rta felipe)

Estan bien todas las opciones menos “no se puede implementar ese requerimineto” depende de como lo justifiques pero se pueden todas. Cuando dice agregar se refiere a low costo sin transporte y precio alto sin transporte, ahi podes hacer concreteBuilder. Pero si queres low costo con transporte, low costo sin transporte, alto precio sin transporte y alto precio con transporte el director es buena porque si no son muchas clases con codigo repetido Te queda un directorConTrasnposte y otro directorSinTransporte y cuando queres sin transporte le pases directorSinTransporte y cuando queres con le pasas el otro. SI solo agregas una clase sin transporte si te conviene concreteBuilider.
Construir cosas nuevas → nuevo director
Construir nuevas versiones con lo mismo → concrete builder

Esta es la regla de oro:

- Si cambia el **Algoritmo** (el orden de los pasos, qué pasos se hacen y cuáles no) → Nuevo **Director**.
- Si cambia la **Representación** (si el transporte es bus o avión, si el hotel es 3 estrellas o 5 estrellas) → Nuevo **Builder**.

Resumen: La respuesta correcta es **Crear un nuevo Director**, porque "Un tour sin transporte" implica cambiar el **proceso de construcción** (la receta), eliminando el paso de inyectar transporte, y esa es responsabilidad del Director.

> *"Te queda un directorConTrasnposte y otro directorSinTransporte y cuando queres sin transporte le pases directorSinTransporte..."*

Traducción: Esta es la justificación de arquitectura. Es mejor tener:

- 2 Directores (Con y Sin transporte).
- 2 Builders (Barato y Caro).
- Total de combinaciones posibles: 4 (Barato con, Barato sin, Caro con, Caro sin).

Si hubieras elegido la opción de modificar los Builders (subclases), tendrías que haber creado 4 clases de Builders distintas para lograr lo mismo. **El Director ahorra código repetido.**