Practica 3 - Patrones de diseños

Dudas

CUIDADO CON UML

si tengo una subclase de una clase abstracta / interfaz solo escribo los metodos que sobreescrito, los que no vuelvo a escribir no
si tengo variables de instancia correspondientes a otra clase tengo que hacer una flecha con el texto “arma 1” por ejemplo, si tengo una lista “arma *”

23 - Revisar Completo
Ya hay un test implementado para probar la funcionabilidad de el metodo getPosts().
¿A que se refiere con patron de test si tengo todos los datos y las clases disponibles para testar? → Aplique un Spy pero los test doubles no se aplicaban cuando no disponíamos de la implementacion de una clase especifica?
¿Porque aca si se implementaria?
En este caso que hay que sobreescribir un metodo que devuelve una lista, como debo implementar el retorno del spy? ¿Con un null o con valores precargados?

22a y b - Revisar completo
¿Estan bien aplicados los Test Doubles?
¿Esta bien pensado el Spy (MixingTankSpy)?

21 - ~~Revisar si esta bien aplicado el patron Null Object~~ - Si, en este caso es correcto, si tuviera otro caso deberia implementar la interfaz (uml del patron que nos dan)

~~El objeto nulo no se instancia mas de una vez → Singleton~~ → Esta bien

20 - ~~Revisar Completo~~ → God

~~¿Hay que hacer como el strategy que se establece uno por defecto?~~ → Si

~~¿Esta bien testear las diferentes armaduras, las armas y los personajes pero el juego individualmente no?~~ → Claro, el metodo construirPersonaje en la clase Juego funciona como un constructor, no se testea

~~¿Que visibilidad tiene el metodo abstracto del template method aplicado en el decorator? (EJEMPLO EJ18)~~ → Protected

~~¿Como puedo diferenciar cuando tengo que usar un Decorator y o un Strategy ya que los 2 agregar funcionalidad?~~ → Todo depende

22 - ~~¿Es correcto el tipo de test double elegido?~~ → Si pq tiene logica sino podes implementar un mock, falta agregar lógica

~~¿Podria aplicarse el tipo Stub? Ya que deberia proporcionar respuestas predefinidas~~ → Si

~~¿Que me determina que tipo de TestDouble tengo que aplicar?~~ → El contexto pero si nos piden

20 - ~~Revisar UML~~ → Esta bien falta agregar el dispacher del arma

~~¿Como represento el combate?~~ → Con 2 dispacher uno para la armadura y el arma

18 - ~~Es correcto que FileOO2 responda al mensaje PrettyPrint con su nombre? ya que en todos los ejemplos siempre empezaban por el nombre (en el caso que quiera cambiar de lugar y mostrarse al final o en otro lado se va a repetir ya que agregue un decorador nombre) Es correcto?~~ → Lo mejor es que se agregue como decorador y prettyPrint del archivo va vacio

~~¿Funciona como un pasamanos? Es correcto?~~ → Si perfecto

14 - ~~Revisar todo ¿poner los numeros de la liquidez hardcodeados esta mal en este caso?~~ → NO pq se esta cargando en el setter Liquidez

13 - ~~Revisar ¿que mas tengo que testear?~~ → Perfecto

12 - ~~Revisar ¿Qué mas tengo que testear?~~ → Estan bien asi

~~En el ArchivoTest ¿es necesario testear los metodos archivoMasNuevo y archivoMasGrande?~~ → Esta bien agregarlos

11 - ~~Revisar completo Test y UML~~ → Perfecto si quiero mejorar el diseño puedo implementar el equals para comparar las listas (REVISAR)

19 - ~~¿Es correcto mi UML?~~
~~Debo crear la clase TemperatureAdapter y no utilizar el metodo getTemperature del HomeWeatherStation ya que este siempre muestra la info siempre en Fahrenheit?~~
→ Sacar Adapter y Strategy

~~A la hora de crear los ConcreteDecorator, estos solo son Max y Min temperature y PromedioTemperaturaHistorica? ¿No es necesario agregar a los demas datos (Presion, RadiacionSolar etc,?¿Es decir, cuando nos dan el String que debe imprimir ya formateado (Temperatura, Presion, Radiacion) este NO PUEDE cambiar su orden?~~ → Si se tienen que agregar los demas datos
18 -
~~Revisar ¿Como puedo evitar el primer espacio en el String que se devuelve? ¿Como seria el metodo prettyPrint() de FileOO2~~
? → Deberia tener el nombre del archivo

~~¿Cómo puedo marcar que FileManager utiliza FileComponent en el UML? ya que no posee una instancia?~~ → Quedan como 2 clases separadas

17 - ~~Revisar si la forma de otorgar acceso esta bien planteada, consiste en un simple booleano el cual se puede setear en true o false facilitando las pruebas.~~
→ Perfecto

15- ~~¿Cómo debo implementar el consumo para cada componente si no es un dato, lo mismo con el precio?~~ → Son datos o sea variables de instnacia

~~¿Como modelo el tema de agregar un pad termino, un cooler y la fuente adaptar al consumo?~~ → Se agrega cuando agregas el procesador

~~Los datos de los Productos en el Patron Builder, es necesario descomponer sus atributos en otras clases? (Ejemplo: Pan, Precio pan o CPU, PrecioCPU? → Se generarían muchas clases de datos~~ → No hace falta se podria utilizar directamenteuna lista y listo

~~¿Es correcto que si una parte no se incluye en el en Builder (Sin GPU integrada) se deba realizar ese paso ingresando vacio?~~ → Si se ingresa vacio

11- ~~Revisar completo~~ → Rehacer

10 - ~~Revisar completo → Probar funcionalidad de los encriptadores en el setup y comparar~~ → Esta bien pero modificar los test en el setup y comprar mensajes cifrados

9 - ~~Reserva funciona un poco como un pasamanos, eso es correcto? ¿Que otras alternativas tengo?~~ → Esta bien

~~Por lo general cuando tengo diferentes formas de mostrar algo, del estilo de:~~

~~Ej 4: "Temperatura F: 86; Presión atmosf: 1008; Radiación solar: 200; Promedio: 86;"~~
~~Ej 5: "Temperatura C: 30; Presión atmosf: 1008; Radiación solar: 200; Promedio: 30; Mínimo: 27 Máximo: 32;"~~
~~Ej 6: "Temperatura C: 30; Presión atmosf: 1008; Radiación solar: 200; Mínimo: 27 Máximo: 32;"~~
~~Ej 7: "Temperatura C: 30; Presión atmosf: 1008; Radiación solar: 200; Mínimo: 27 Máximo: 32; Promedio: 30;"~~

~~nombre - extensión~~
~~nombre - fecha de creación - extensión~~
~~nombre - tamaño - permisos - extensión~~

~~¿voy a tener que utilizar un decorator ? ya que la forma de mostrar los datos pueden combinarse?~~ → Si siempre que tengamos una situaciom similiar hay que aplicar un decorator

8 - ¿~~Como deberia ser el contructor de Dispositivo?¿Debe tener asignado de una una Estrategia y un Tipo de Conexion por defecto? ¿O debe tener el constructor vacio?~~

~~Revisar TEST!!!~~ → Rehacer test pero para inicializar en un estado, no asumir ninguno y directamente pasarlo por parametro en el constructor

7 - ~~Revisar (incluso test) y consultas UML ¿es correcto que las concreteState llamen a metodos de la clase Calculadora?~~ → Si en este caso si

6- ~~Idem 7~~ → En este caso esta bien

5 - Revisar (incluso tests), ~~esta bien aplicado el patron? los metodos para procesar las listas DEBEN estar en la clase decodificador?~~ → Esta bien aplicado

4 - ~~Revisar~~ esta bien faltan test

~~1 Revisar~~ → Ta joya que no haya tenido que hacer varios test para probar la funcionabilidad de los test

2 ~~Revisar si esta bien la herencia de métodos innecesarios y que sucede con la v.i. "casado" y si esta bien planteado la suma de las horas extra para el empleado temporar~~io → Ta perfecto, con la vi casado, siempre se tiene que hacer un balance entre los malos olores, ¿que es peor queee?

3 ~~¿Tengo que implementarlo realmente?~~ → No

Ejercicio 1 b ✅

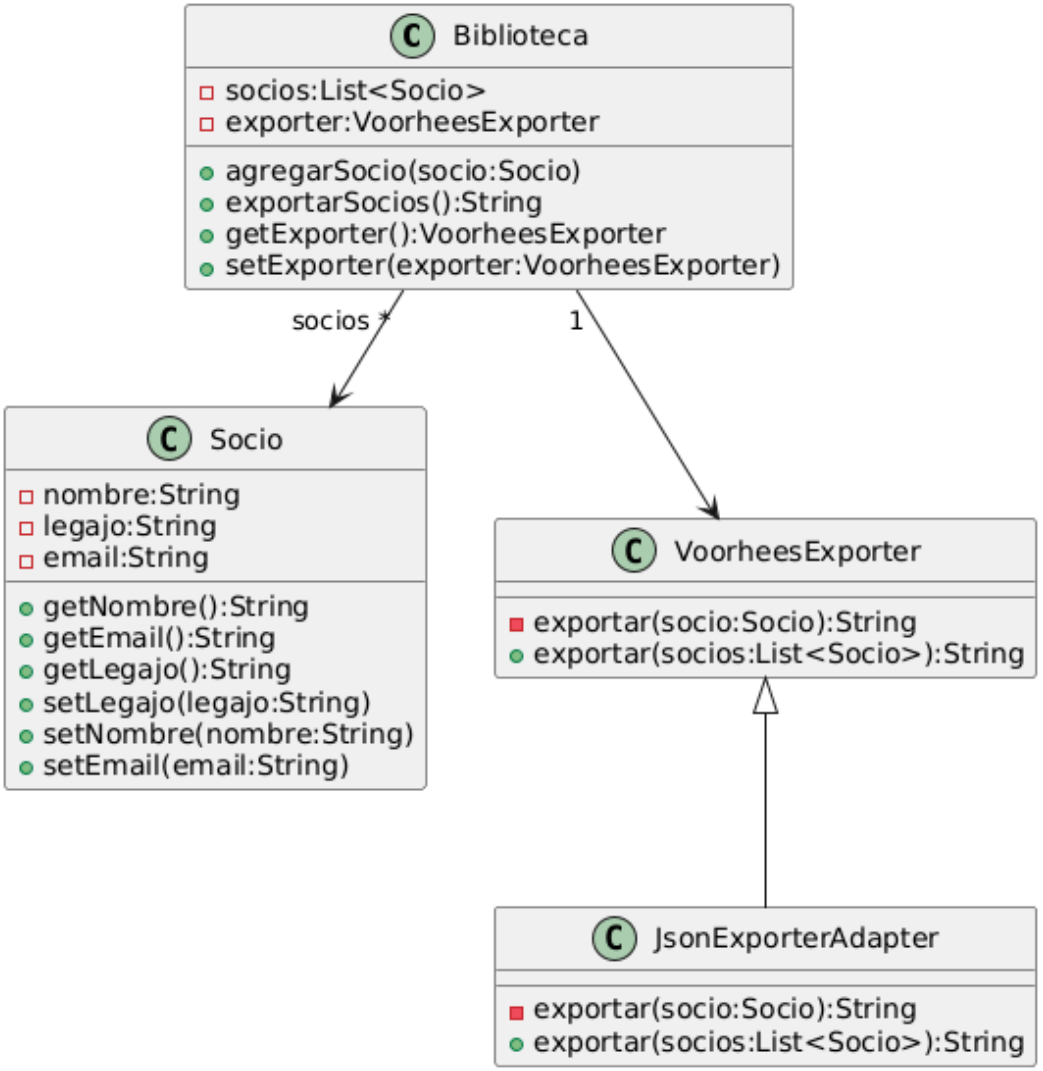
Se aplico el Patron Adapter

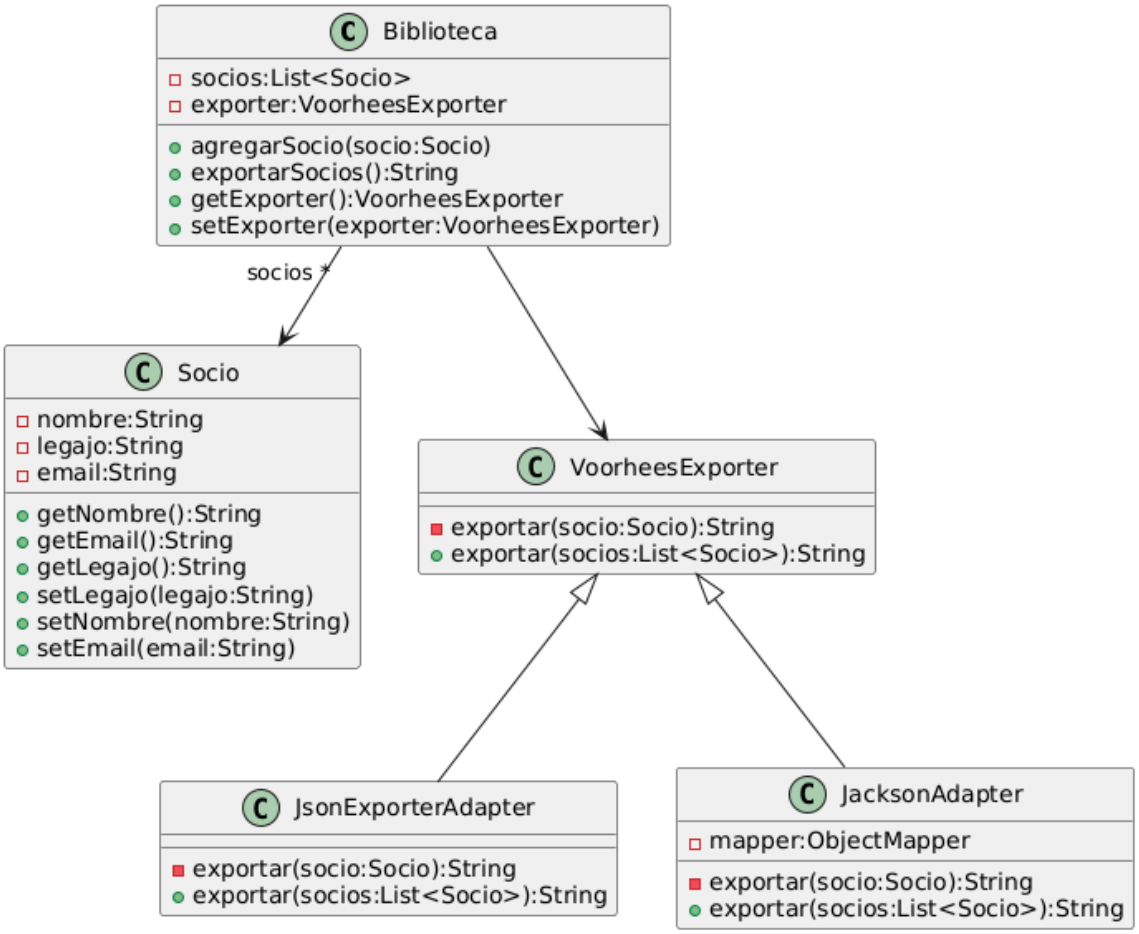
Adaptable → Libreria JSON

Adaptador → JSONExporterAdapter y JacksonAdapter

Objetivo→ VoorheesExporter

Cliente → Biblioteca



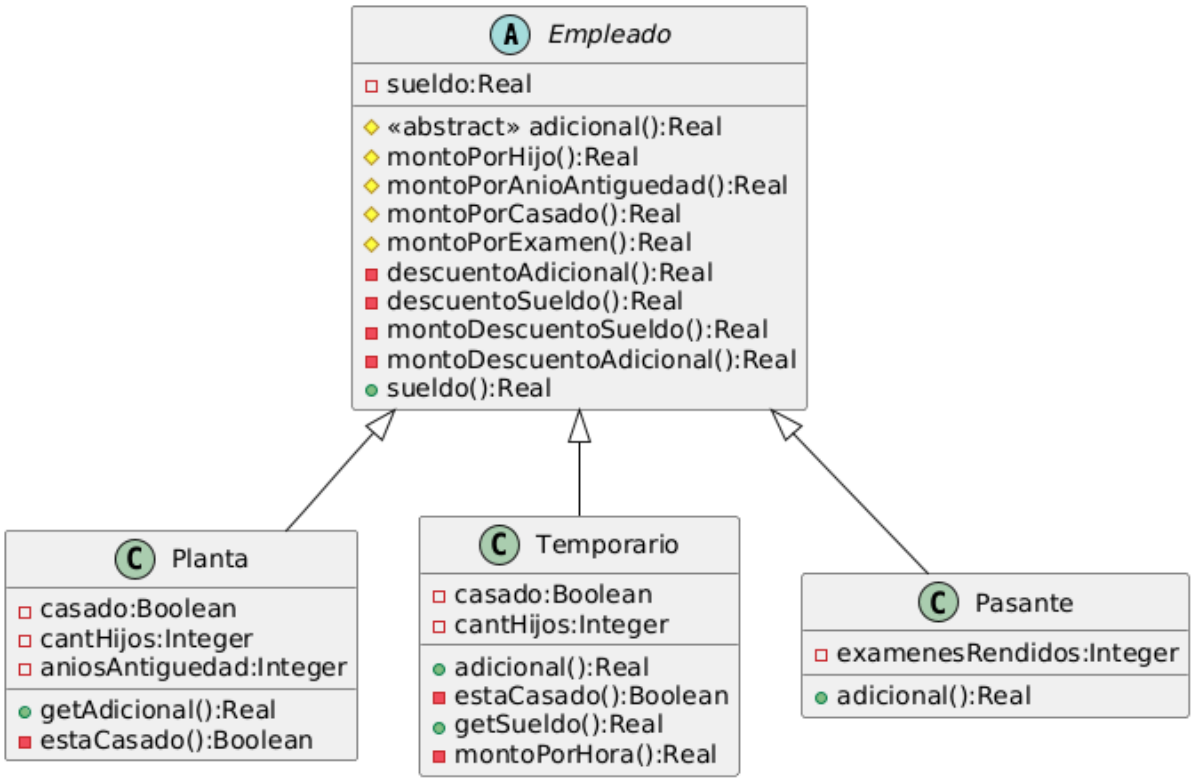


Ejercicio 2 ✅

Se aplico el Patron Template Method

Abstract Class → Empleado

Concrete Class → Planta, Temporario y Pasante



Ejercicio 3 ✅

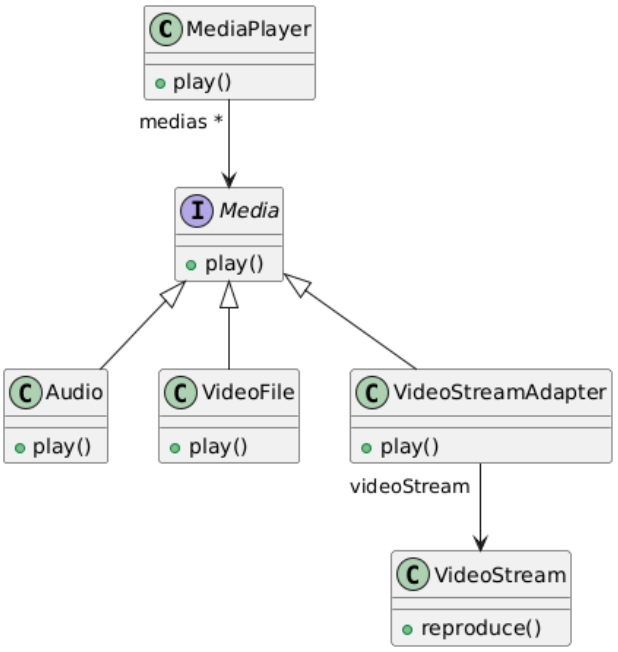
Se aplico el Patron Adapter

Adaptable → VideoStream

Adaptador → VideoStreamAdapter

Objetivo→ Media

Cliente → MediaPlayer



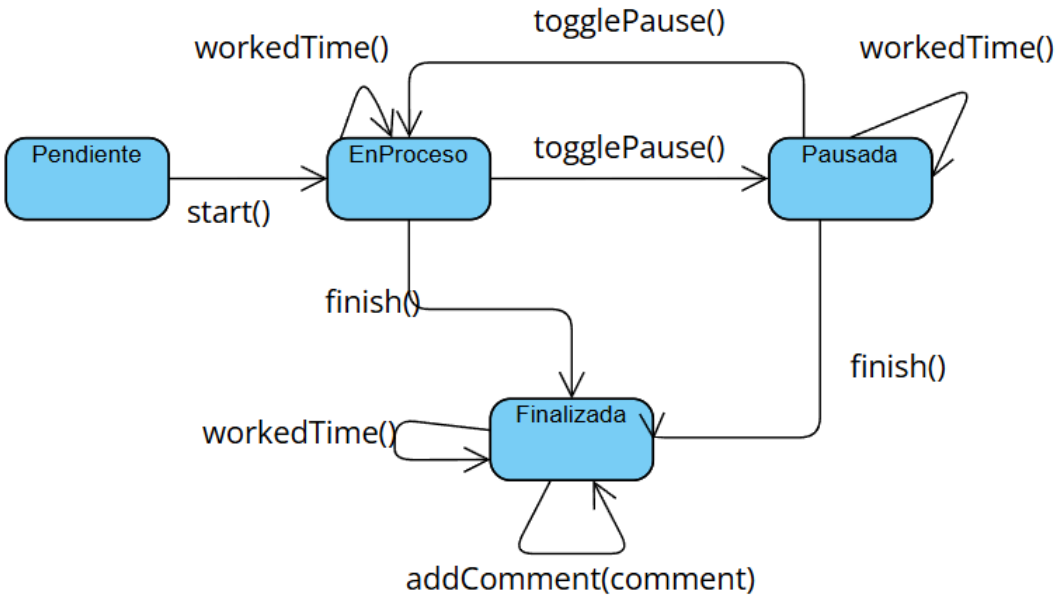
Ejercicio 4 ✅

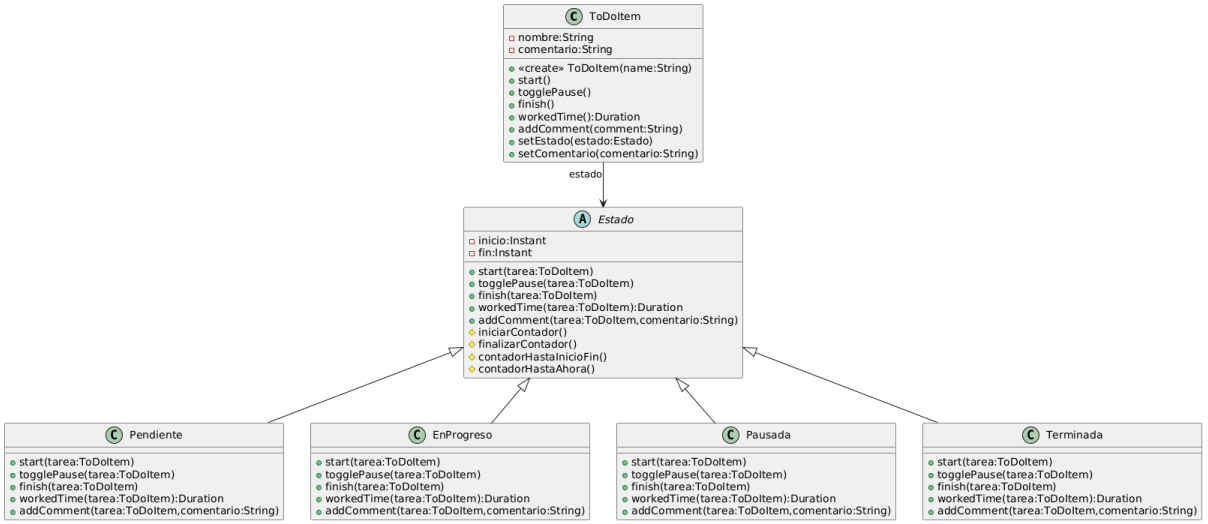
Se aplico el Patron State

Context→ ToDoItem

Estado → State

ConcreateState → EnProgreso, Terminado, Pausada y Pendiente





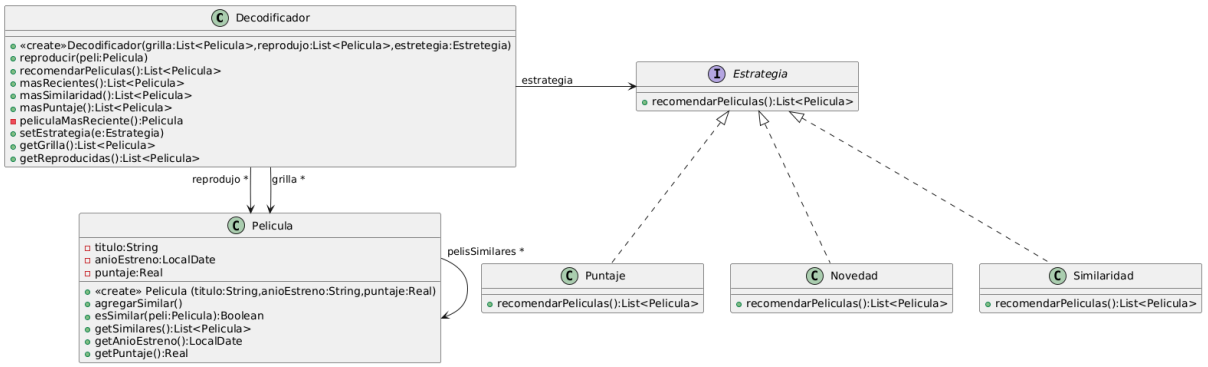
Ejercicio 5 ✅

Se aplico el patron Strategy

Context → Decodificador

Strategy → Estrategia

ConcreteStrategy → Puntaje, Similaridad y Novedad



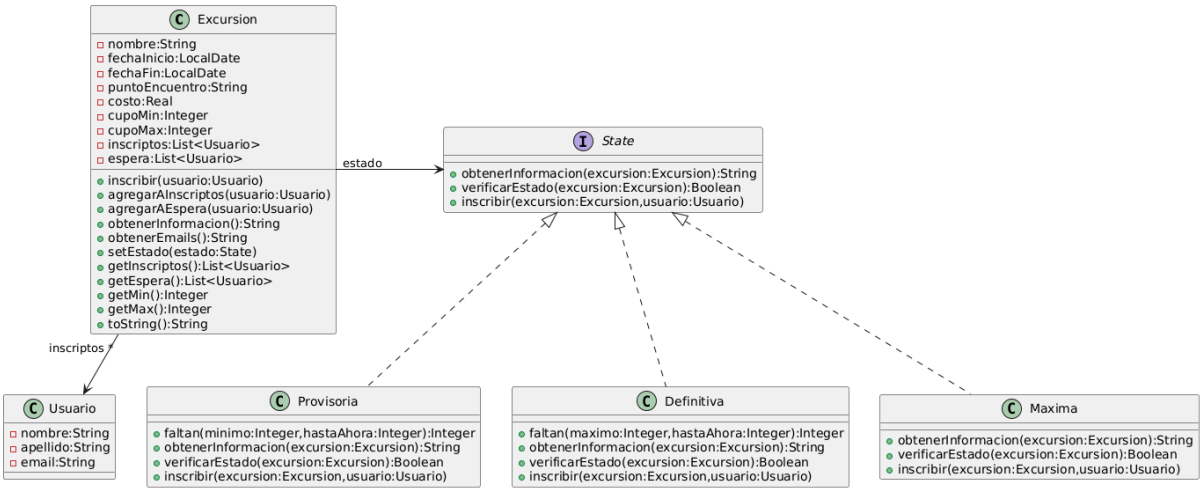
Ejercicio 6 ✅

Se aplico el patron State

Context → Excursion

Estado → State

ConcreteState → Definitiva, Provisoria y Máxima



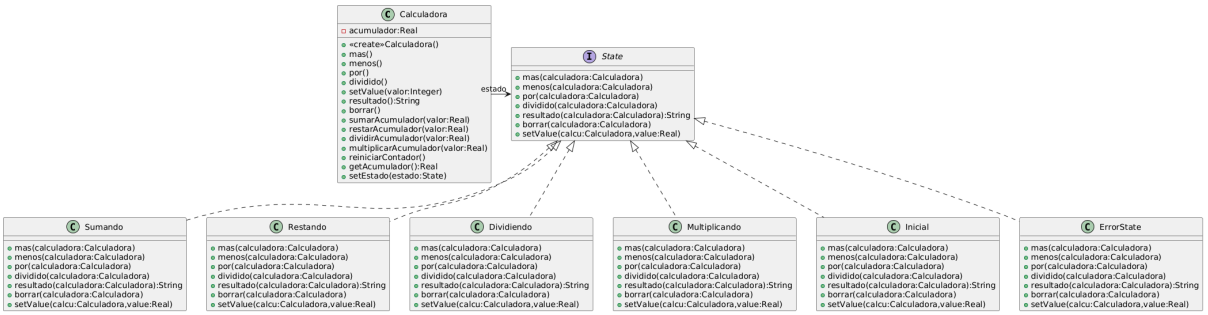
Ejercicio 7 ✅

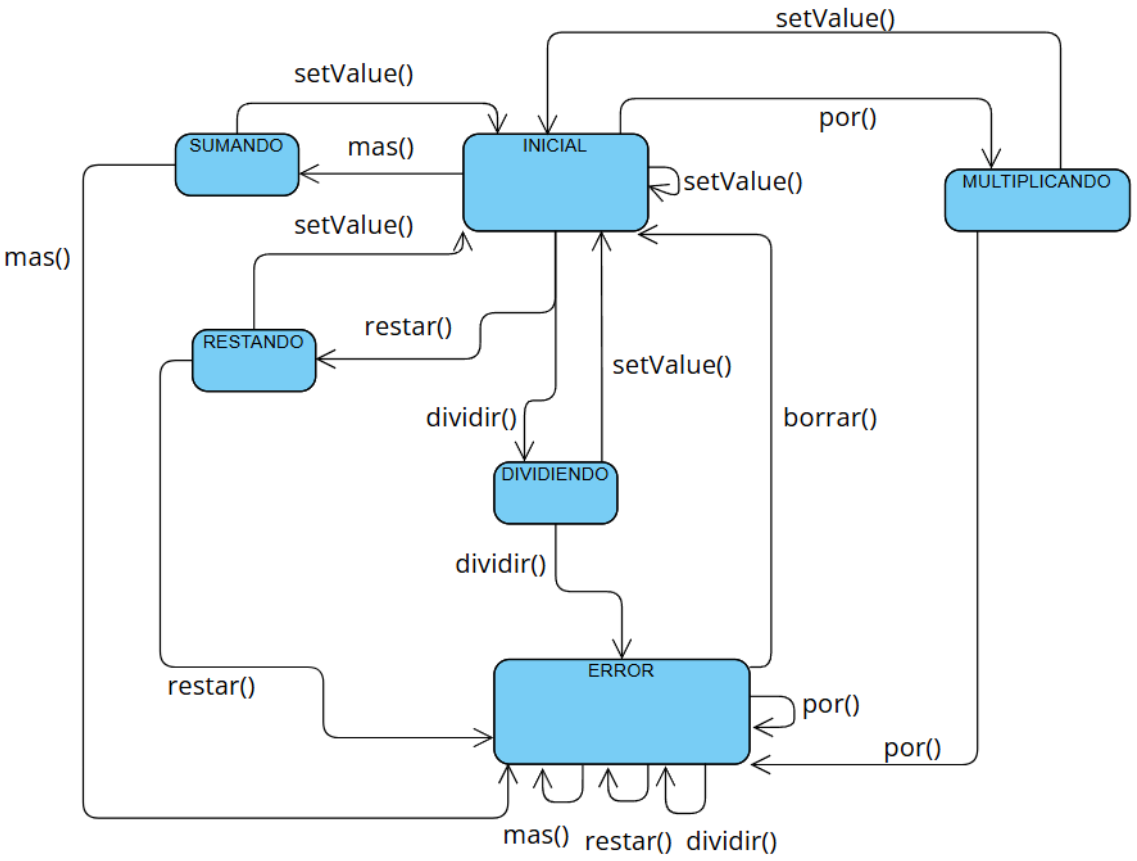
Se aplico el patron State

Conext → Calculadora

Estado → State

ConcreteState → Inicial, Sumando, Restando, Dividiendo, Multiplicando y ErrorState





Ejercicio 8 ✅

Para poder configurar el calculador de CRC, que puede ser el CRC16_Calculator, el CRC32_Calculator, o pueden ser nuevos a futuro. → Debemos aplicar el patron Strategy ya que con esto permitiremos cambiar entre las distintas calculadoras e incluir mas a futuro

Strategy → Strategy

Context → Dispositivo

ConcreteStrategy → CRC32_Calculator y CRC16_Calculator

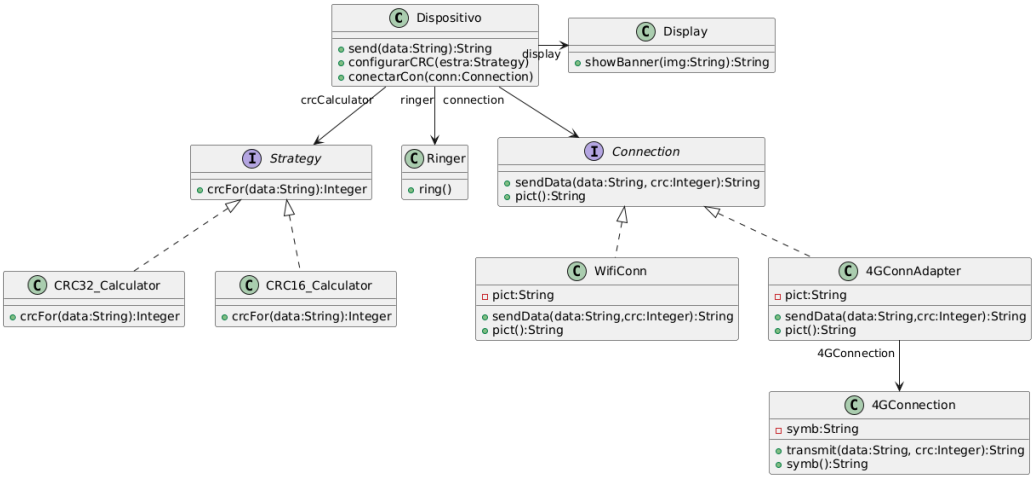
Para poder cambiar la conexión, ya sea la 4GConnection o la WifiConn. En este método se espera que se pase a utilizar la conexión recibida, muestre en el display su símbolo y genere el sonido. → Debemos aplicar el patrón Adapter y crear una clase la cual adapte el comportamiento de 4G Connection a lo requerido por la interfaz

Cliente → Dispositivo

Objetivo → Connection

Adaptador → 4GConnAdapter

Adaptable → 4GConnection



Ejercicio 9 ✅

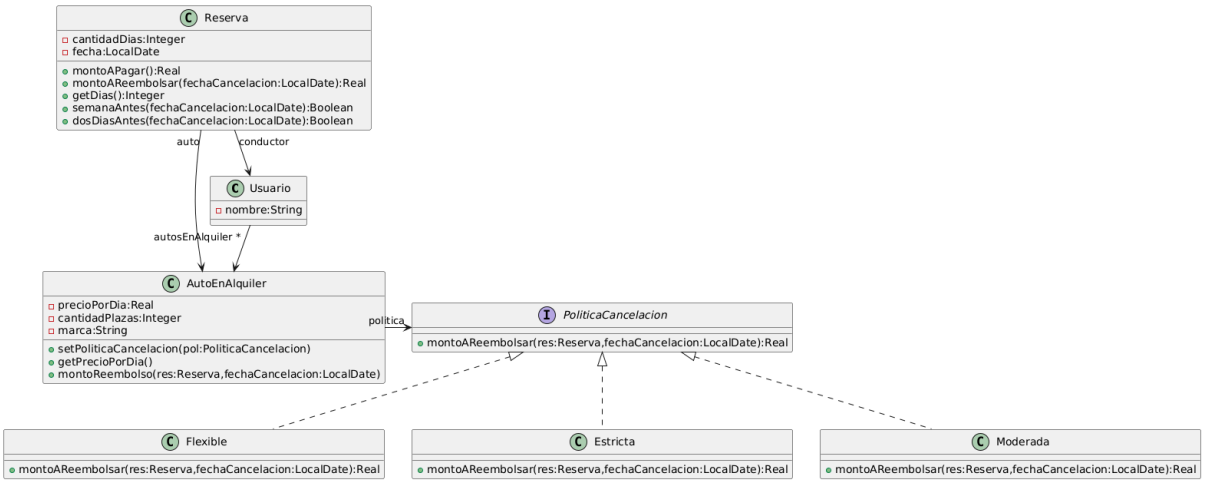
Se debe aplicar el patrón Strategy ya que las políticas pueden cambiar en tiempo de ejecucion

Strategy → PoliticaCancelacion

Context → AutoEnAlquiler

ConcreteStrategy → Flexible, Moderada y Estricta

Las ventajas son que se puede cambiar la politica cuando se necesite y ademas permite la incorporacion de nuevas politicas



Ejercicio 10 ✅

Debemos aplicar 2 patrones, el strategy y luego el adapter

Strategy → Cifrador

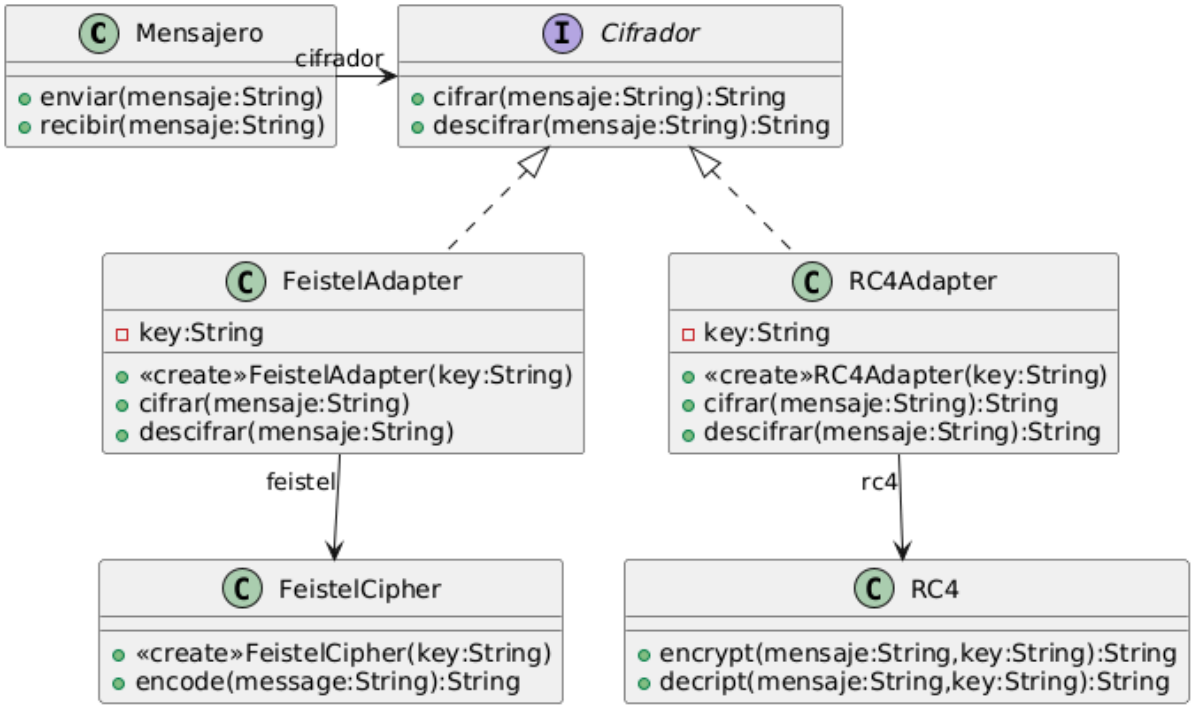
ConcreteStrategy → FeistelAdapter y RC4Adapter

Context → Mensajero

Adapador → RC4Adapter y FeistelAdapter

Adaptable → RC4 y FeistelCipher

Target → Cifrador



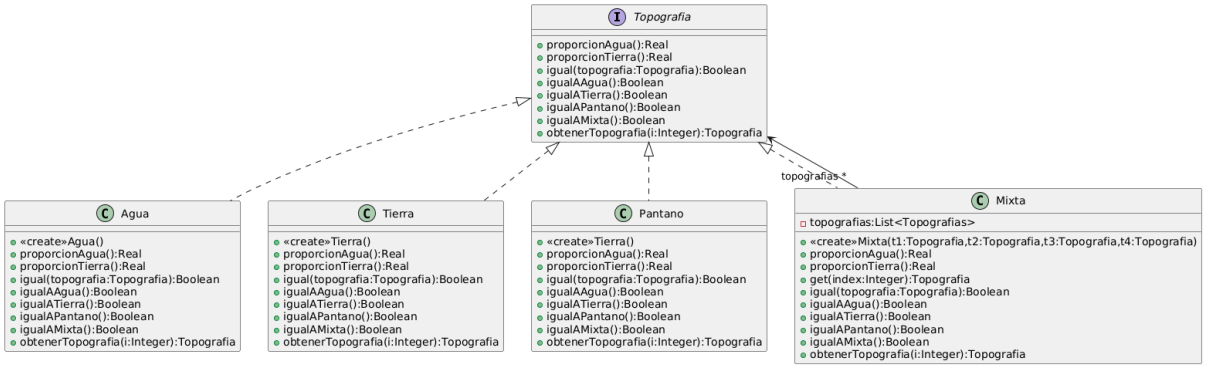
Ejercicio 11✅ Se puede mejorar el procesamiento del igual en Mixta

Se debe aplicar el patron Composer ya que debemos componer a la entidad Mixta que esta conformada por un conjunto de entidades

Component → Topografia

Leaf → Agua, Tierra, Pantano

Composite → Mixta



Ejercicio 12 ✅

Se debe aplicar el patron Composer ya que debemos componer a la entidad Directorio en funcion de Archivos

Component → Contenido

Cliente → FileSystem

Leaf → Archivo

Composite → Directorio

Ejercicio 13 ✅

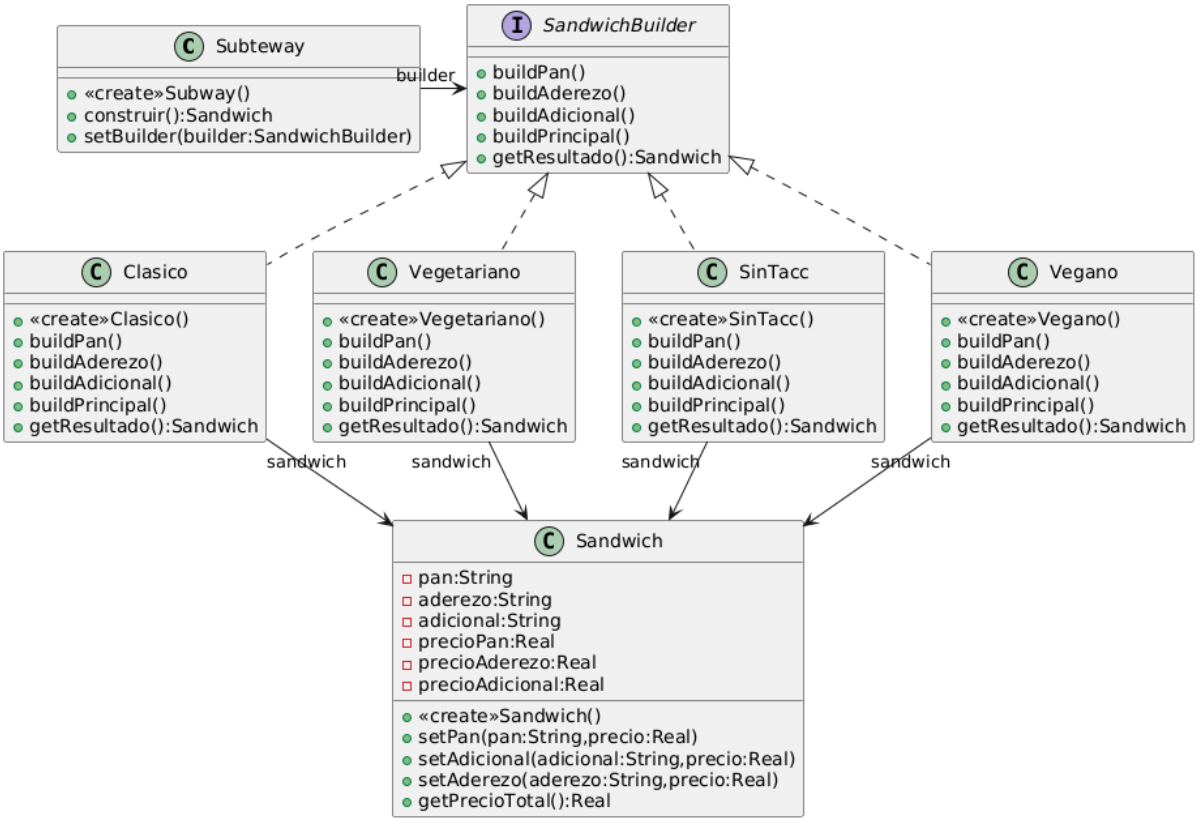
Se debe aplicar el patron Builder ya que el modo de construir los distintos sandwichs es igual pero cambian los componentes

Director → Subteway

Builder → SandwichBuilder

ConcreteBuilder → Vegetariano, SinTacc, Vegano y Clasico

Producto → Sandwich



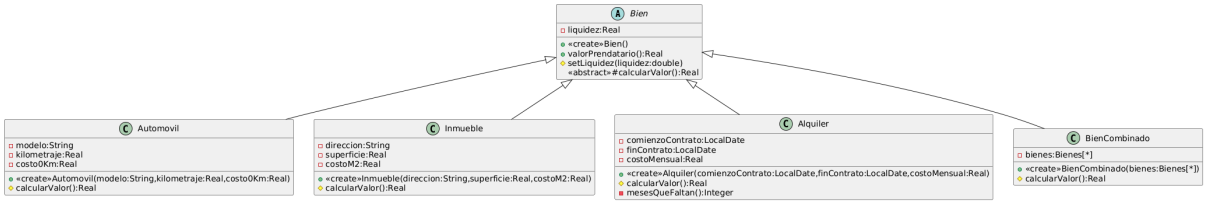
Ejercicio 14 ✅

Se debe aplicar el patrón Composite ya que se desea aceptar bienes combinados

Componente → Bien

Leaf → Automóvil, Alquiler e Inmueble

Composite → Combinadas



Ejercicio 15 Modificar Codigo

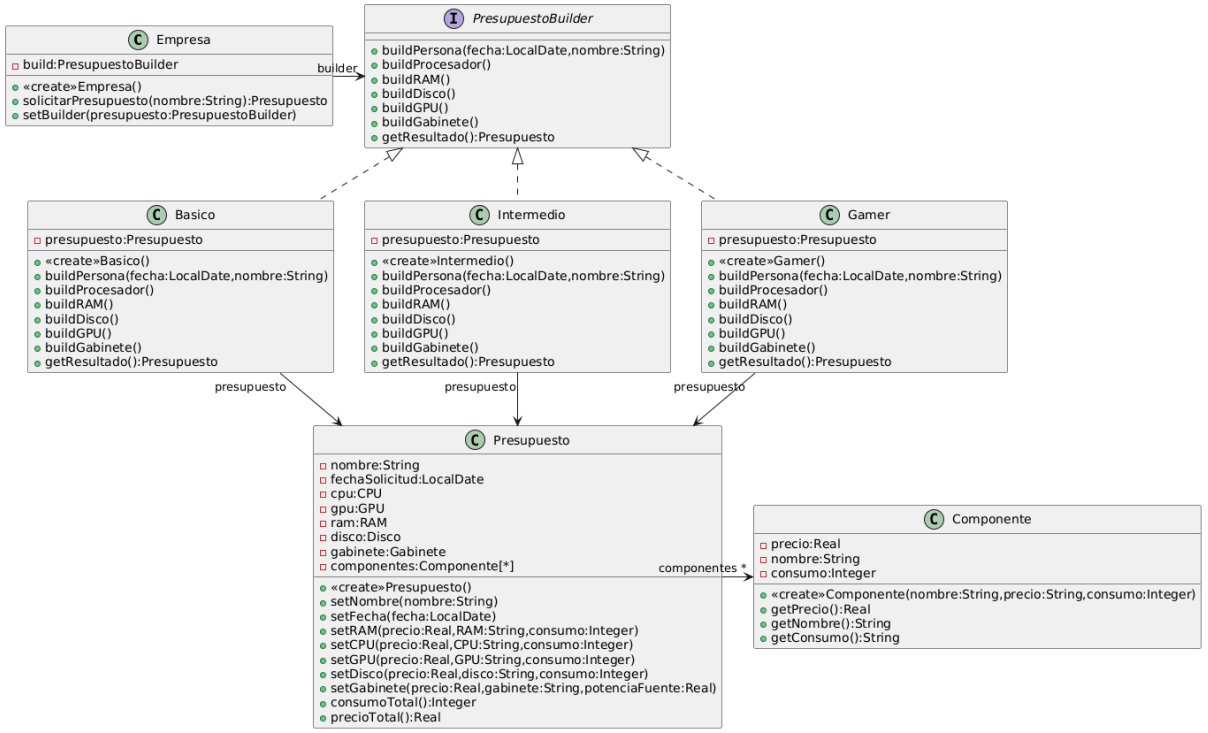
Se debe aplicar el patron Builder para la configuración de presupuestos.

Director → Empresa

Builder → PresupuestoBuilder

ConcreteBuilder → Basico, Intermedio y Gamer

Producto → Presupuesto



Ejercicio 16 ✅

El patrón que mejor describe el funcionamiento de los Filtros es el Decorator.

a. Si, el objetivo del patrón es poder utilizar varias capas e ir aplicando diferentes filtros sobre un objeto de manera de “envolverlo”, esto es posible ya que el programa permite aplicar no solo 1 filtro al mismo tiempo.

b. Component → Filtro

   Client → PNGFilterLauncher

ConcreteComponent → Artifacter, Dull, Rainbow, Repeater, RGBShifter y RGBShifterRepeater

Decorator → Filter

c. El cliente puede ser algo que se aleja del modelo presentado en el libro

Ejercicio 16b ✅

el patron que mejor describe el diseño de los pipes es el Decorator, ya que IM `ImageFilterPipe` decora a otro `ImageFilterPipe`, agregándole un nuevo filtro al procesamiento. La estructura es recursiva (como en Decorator), y se construyen cadenas dinámicamente.

a. Se pueden apilar filtros dinamicamente, agregando funcionalidad progresiva

b. Sí. Cada `ImageFilterPipe` contiene otro pipe (`nextPipe`) y aplica su filtro antes de delegar

c. Podría considerarse que el uso del método `addPipeFrom` es innecesariamente complejo para la simple función de encadenar filtros. Una estructura más clara o un Builder podría ser más intuitivo.

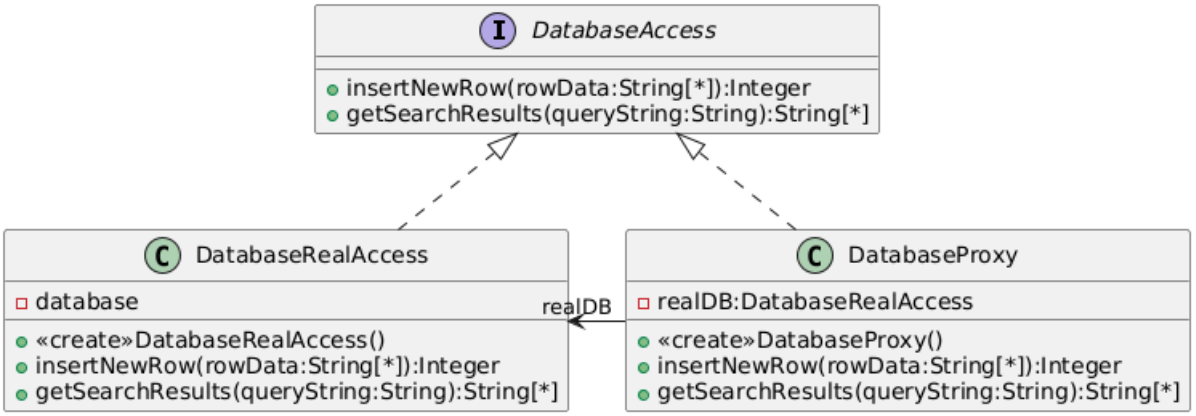
Ejercicio 17 ✅

Se debe aplicar el patron proxy del tipo proxy de proteccion ya que es necesario para verificar si se posee permisos

Subject → DatabaseAccess

RealSubject → DatabaseRealAccess

Proxy → DatabaseProxy



Ejercicio 18 ✅

Se debe aplicar el patron decorator para poder mostrar los diferentes datos de manera individual o en conjunto

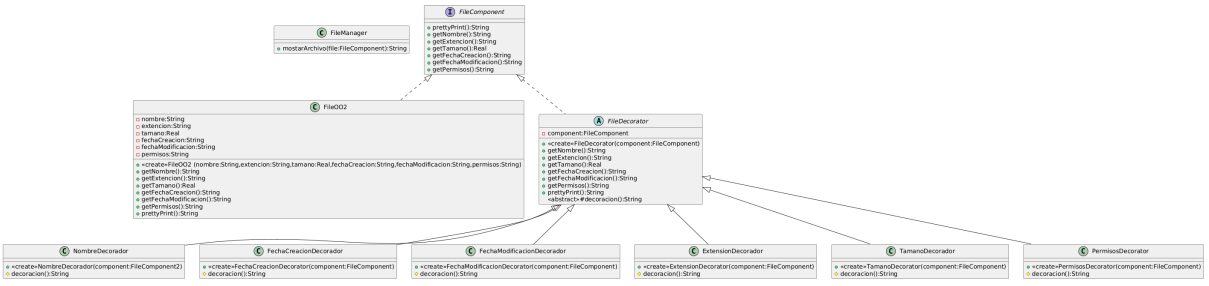
Component → FileComponent

Client → FileManager

ConcreteComponent → FileOO2

Decorator → FileDecorator

ConcreteDecorator → ExtensionDecorator, NombreDecorator, FechaCreacionDecorator, FechaModificacionDecorator, PermisosDecorator y TamanoDecorator



Ejercicio 19 Codear

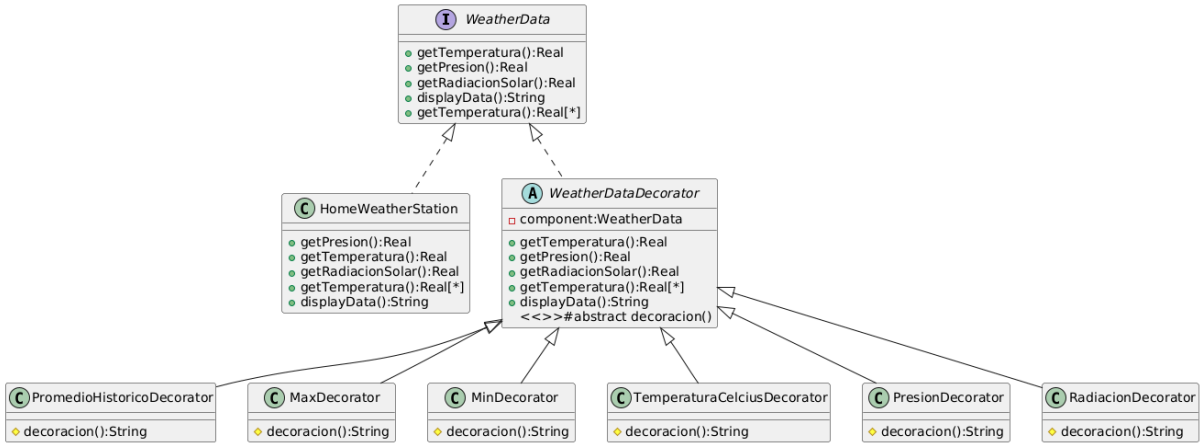
Se debe aplicar el patron Decorator

Componente → WeatherData

ConcreteComponente → HomeWeatherStation

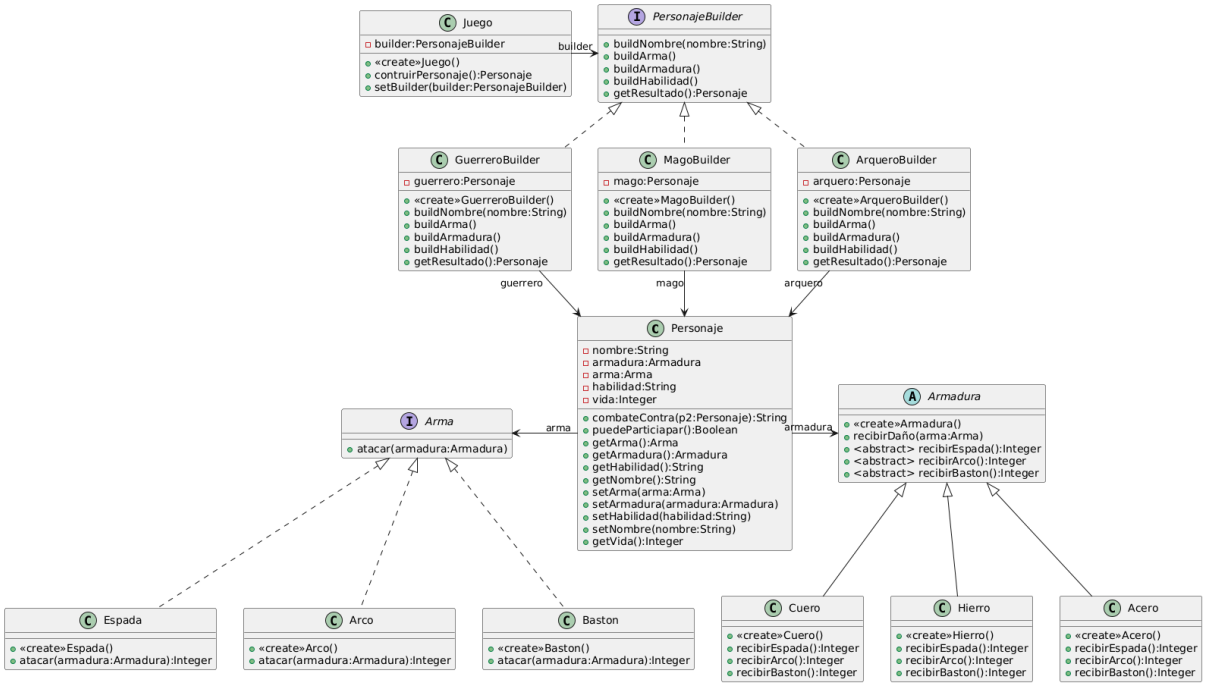
Decorator → WeatherDataDecorator

ConcreteDecorator → MinDecorator, MaxDecorator y PromedioHistoricoDecorator

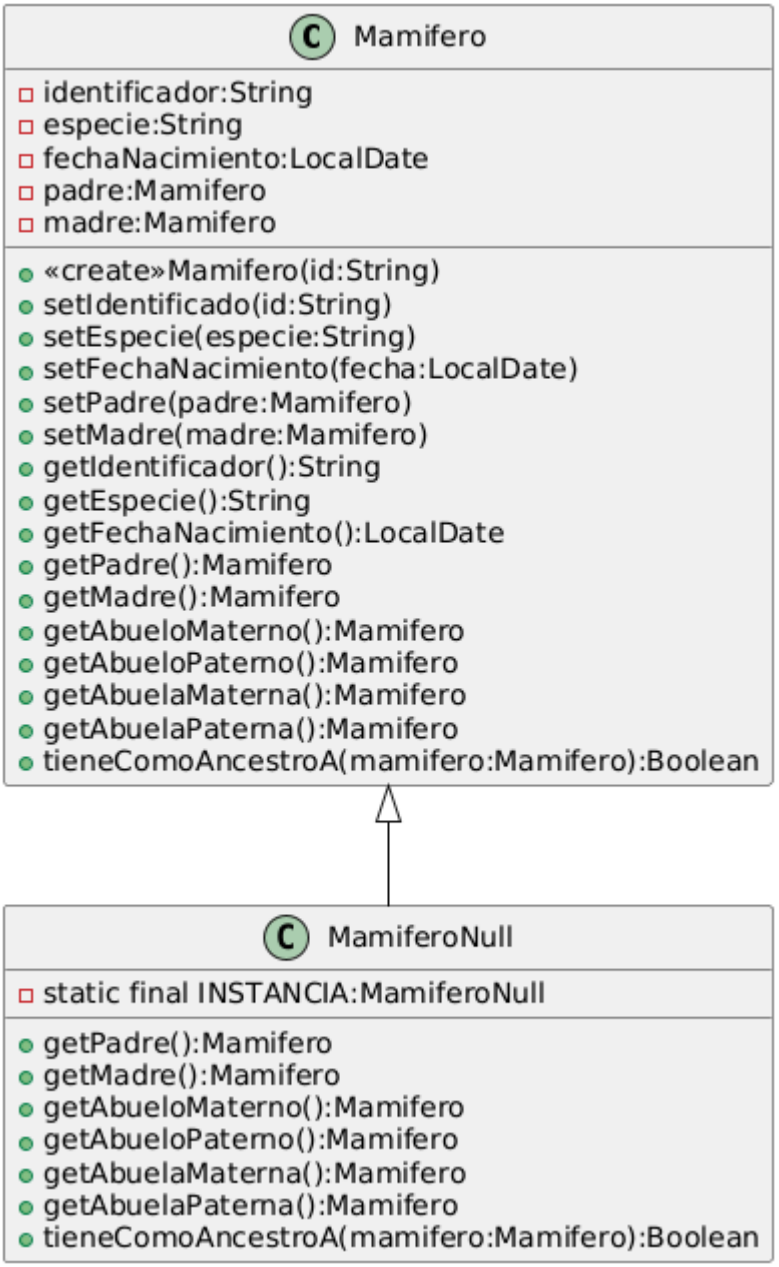


Ejercicio 20 ✅

Se debe aplicar el patron builder
Director → Juego
Builder → PersonajeBuilder
ConcreteBuilder → GuerreroBuilder, MagoBuilder y ArqueroBuilder
Producto → Personaje



Ejercicio 21 ✅



Clase Mamifero

```
public class Mamifero {

    private String identificador;
    private String especie;
    private LocalDate fechaNacimiento;
    private Mamifero madre = MamiferoNull.getInstancia();
    private Mamifero padre = MamiferoNull.getInstancia();

    public Mamifero (String identificador) {
```

```
        this.identificador = identificador;
    }

    public Mamifero () {

    }

    public String getIdentificador() {
        return identificador;
    }

    public void setIdentificador(String id) {
        this.identificador = id;
    }

    public String getEspecie() {
        return especie;
    }

    public void setEspecie(String especie) {
        this.especie = especie;
    }

    public java.time.LocalDate getFechaNacimiento() {
        return fechaNacimiento;
    }

    public void setFechaNacimiento(java.time.LocalDate fecha) {
        this.fechaNacimiento = fecha;
    }

    public Mamifero getMadre() {
        return this.madre;
    }

     public void setMadre(Mamifero madre) {
         if (madre == null) {
             this.madre = MamiferoNull.getInstancia();
```

```
    } else {
        this.madre = madre;
    }
}

public Mamifero getPadre() {
    return this.padre;
}

public void setPadre(Mamifero padre) {
    if (padre == null) {
        this.padre = MamiferoNull.getInstancia();
    }
    else {
        this.padre = padre;
    }
}

public Mamifero getAbuelaMaterna() {
    return this.getMadre().getMadre();
}

public Mamifero getAbueloMaterno() {
    return this.getMadre().getPadre();
}

public Mamifero getAbuelaPaterna() {
    return this.getPadre().getMadre();
}

public Mamifero getAbueloPaterno() {
    return this.getPadre().getPadre();
}

public boolean tieneComoAncestroA(Mamifero unMamifero) {
    return (tieneAncestro(this.getMadre(),unMamifero) || tieneAncestro(this.g
}
```

```
    private boolean tieneAncestro (Mamifero parent, Mamifero unMamifero) {
        return (unMamifero.equals(parent) || parent.tieneComoAncestroA(unMam
    }
}
```

Clase MamiferoNulo

```
public class MamiferoNull extends Mamifero {

    private static final MamiferoNull INSTANCIA = new MamiferoNull();

    private MamiferoNull() {
        super("NULO");
    }

    public static MamiferoNull getInstancia() {
        return INSTANCIA;
    }

    @Override
    public Mamifero getPadre() {
        return this;
    }

    @Override
    public Mamifero getMadre() {
        return this;
    }

    @Override
    public Mamifero getAbuelaMaterna() {
        return this;
    }

    @Override
    public Mamifero getAbueloMaterno() {
        return this;
    }
```

```
    @Override
    public Mamifero getAbuelaPaterna() {
        return this;
    }

    @Override
    public Mamifero getAbueloPaterno() {
        return this;
    }

    @Override
    public boolean tieneComoAncestroA(Mamifero unMamifero) {
        return false;
    }
}
```

Clase MamiferoTest

```
public class MamiferoTest {
    private Mamifero nala;
    private Mamifero melquiades;
    private Mamifero mufasa;
    private Mamifero alexa;
    private Mamifero elsa;
    private Mamifero scar;
    private Mamifero sarabi;
    private Mamifero anonimo;

    @BeforeEach
    void setUp() {
        nala = new Mamifero("Nala");
        melquiades = new Mamifero("Melquiades");
        mufasa = new Mamifero("Mufasa");
        alexa = new Mamifero("Alexa");
        elsa = new Mamifero("Elsa");
        scar = new Mamifero("Scar");
        sarabi = new Mamifero("Sarabi");
```

```
        anonimo = new Mamifero();

        alexa.setPadre(mufasa);
        alexa.setMadre(sarabi);
        mufasa.setPadre(melquiades);
        mufasa.setMadre(nala);
        sarabi.setPadre(scar);
        sarabi.setMadre(elsa);

    }

    @Test
    void testAbuelaMaterna() {
        assertEquals(elsa, alexa.getAbuelaMaterna());
        assertNull(sarabi.getAbuelaMaterna());
        assertNull(elsa.getAbuelaMaterna());
    }

    @Test
    void testAbuelaPaterna() {
        assertEquals(nala, alexa.getAbuelaPaterna());
        assertNull(mufasa.getAbuelaPaterna());
        assertNull(nala.getAbuelaPaterna());

    }

    @Test
    void testAbueloMaterno() {
        assertEquals(scar, alexa.getAbueloMaterno());
        assertNull(sarabi.getAbueloMaterno());
        assertNull(scar.getAbueloMaterno());

    }

    @Test
    void testAbueloPaterno() {
        assertEquals(melquiades, alexa.getAbueloPaterno());
        assertNull(mufasa.getAbueloPaterno());
```

```
    assertNull(melquiades.getAbueloPaterno());

}

@Test
void testEspecie() {
    anonimo.setEspecie("Panthera leo");
    assertEquals("Panthera leo", anonimo.getEspecie());
}

@Test
void testIdentificador() {
    anonimo.setIdentificador("Nala");
    assertEquals("Nala", anonimo.getIdentificador());
}

@Test
void testMadre() {
    anonimo.setMadre(alexa);
    assertEquals(alexa, anonimo.getMadre());
    assertNull(nala.getMadre());
}

@Test
void testPadre() {
    anonimo.setPadre(mufasa);
    assertEquals(mufasa, anonimo.getPadre());
    assertNull(nala.getPadre());
}

@Test
void testTieneComoAncestroA() {
    assertFalse(nala.tieneComoAncestroA(anonimo));
    assertFalse(mufasa.tieneComoAncestroA(anonimo));
    assertFalse(alexa.tieneComoAncestroA(anonimo));
    assertFalse(alexa.tieneComoAncestroA(alexa));
    assertTrue(alexa.tieneComoAncestroA(mufasa));
    assertTrue(alexa.tieneComoAncestroA(sarabi));
```

```
        assertTrue(alexa.tieneComoAncestroA(nala));
        assertTrue(alexa.tieneComoAncestroA(melquiades));
        assertTrue(alexa.tieneComoAncestroA(elsa));
        assertTrue(alexa.tieneComoAncestroA(scar));
    }
}
```

Ejercicio 22 Revisar

//PRIMERA VEZ QUE LO PENSE

En este caso yo implementaría el Fake Object ya que es una versión simplificada del objeto real, este tipo de objetos tiene una lógica interna pero no es adecuado para la producción

//SEGUNDA VEZ QUE LO PENSE
Se puede aplicar un Stub Object, en este caso tambien implementamos una jerarquia de MixingTank

MixingTank

⏫

MixingTankStub

Ejercicio 22b Revisar

En este caso yo aplicaría el tipo de Test Double de Spy Object, el cual nos permite registrar información sobre las interacciones con el objeto

Ahora, no podemos realizar llamadas sobre la clase MixingTank (implementacion real) ya que:
Los cambios de temperatura dependen de llamadas reales a Thread.sleep
No se puede verificar directamente si los metodos fueron llamados ni con que valores (no tenemos visibilidad de las interacciones)
Su comportamiento no es controlable desde un test unitario.

Por todo esto necesitaremos realizar test sobre una abstracción de la clase → Creamos una jerarquía

<<abstract>>

MixingTank

⏫

MixingTankSpy MixingTankReal

Ejercicio 23 Revisar

Tareas:

1. Analice la implementación entregada en el material adicional. Tenga en cuenta que la lista de posts de un usuario puede llegar a ser muy extensa, por lo cual se necesita demorar la generación de esta lista hasta que sea requerida (a través de la invocación al método getPosts()). ¿Se pudo completar este requerimiento en la implementación actual?

   No, no se pudo completar este requerimiento ya que el metodo findUserByUsername devuelve un objeto user con solamente el nombre y su correo asociado

2. Realizar las modificaciones necesarias en el método findUserByUsername() de la clase UserRepository teniendo en cuenta los requerimientos mencionados en el punto anterior. Implemente todo lo que considere necesario para satisfacerlo. Si utiliza patrones de diseño indique los roles en las clases utilizando estereotipos.

   Primero lo ideal es crear una clase concreta LazyUserProxy que implementa a PersistableUser, ya que, la clase User corresponde al modelo de negocio por lo tanto no debe tener conocimiento del mecanismo de persistencia.

   En este caso se aplica el Patron Proxy Virtual donde creamos un proxy que demora la creación de un objeto hasta que sea necesario
   Tema → PersistableUser
   TemaReal → User
   Proxy → LazyUserProxy

3. Realice los test necesarios relacionados a la funcionalidad de recuperar un usuario demorando la generación de la lista de Posts. ¿Qué patrón de test usaría?
   Los test necesarios relacionados a la funcionalidad de recuperar un usuario demorando la generación de la lista de Posts YA estan implementados
   De todas formas para probar el funcionamiento del metodo findPostsByUsername aplicamos el Test Double SPY para monitorear la llamada, creamos la clase PostRepositorySpy que herede de la clase PostRepository y sobreescriba el metodo, ademas debemos cambiar la clase UserRepository

```
//new PostRepository()
new PostRepositorySpy()
```