Patrones de Diseño: Ejercicios.

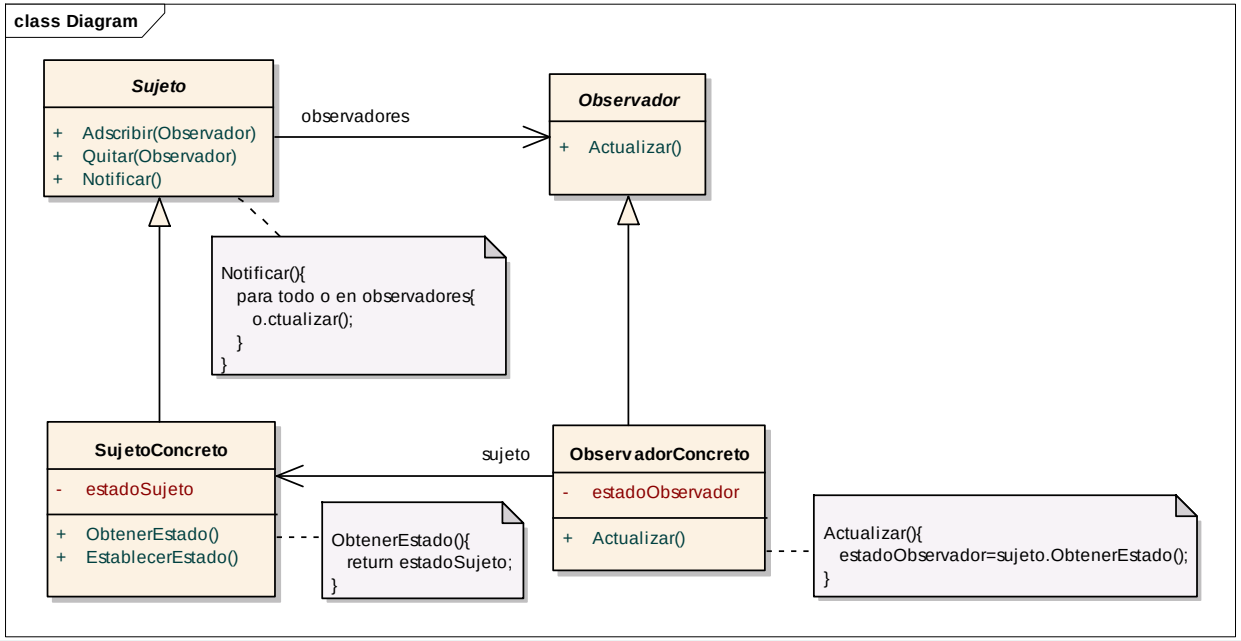
Patrones de Comportamiento

Ejercicio 1

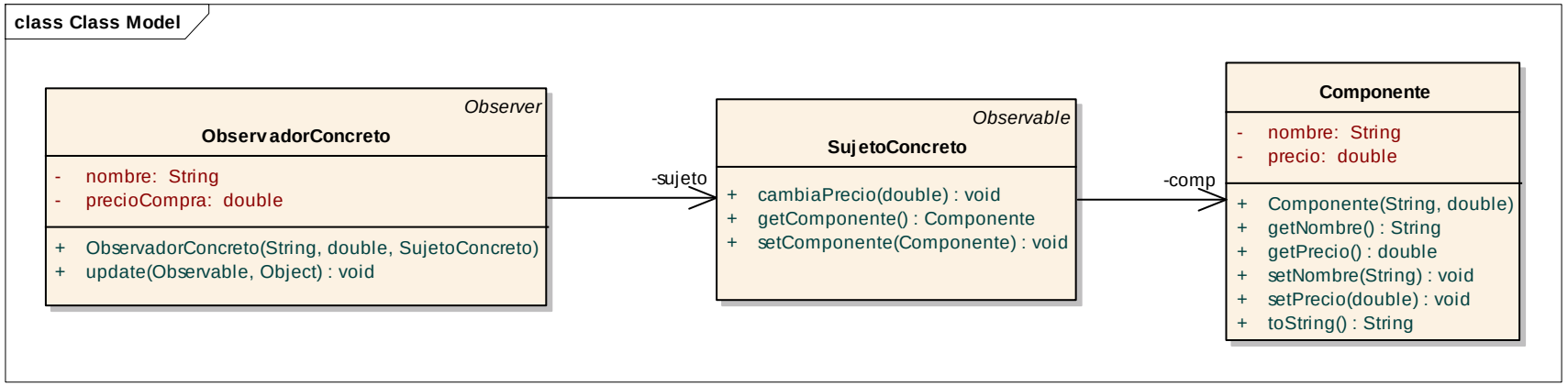
- Una tienda de informática vende componentes de ordenador, los precios de estos componentes varían de forma diaria. Los clientes pueden solicitar que se les avise en el momento en el que el precio de un componente ha alcanzado un determinado precio o es inferior al indicado por el cliente. Se quiere realizar una aplicación Java que notifique a los clientes estos cambios de precio. Encontrar el mejor patrón que se adapta a esta situación.

Ejercicio 1. Solución

- El patrón que mejor se adapta a este ejercicio es el Observer.
  - *Permite definir dependencias uno-a-muchos de forma que los cambios en un objeto se comuniquen a los objetos que dependen de él.*



Ejercicio 1. Solución



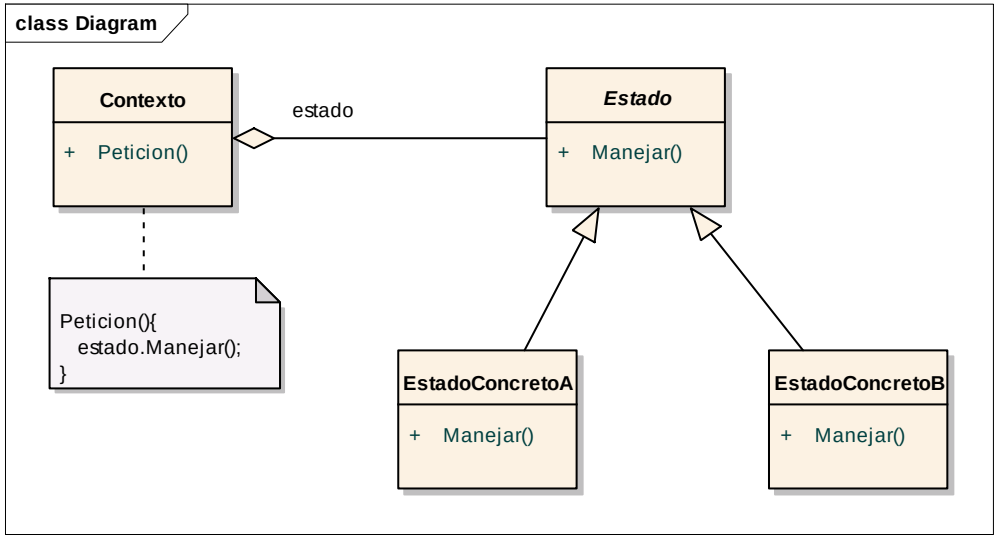
- Identificamos a continuación los elementos del patrón:
  - Sujeto: Observable.
  - SujetoConcreto: SujetoConcreto.
  - Observador: Observer.
  - ObservadorConcreto: ObservadorConcreto.

Ejercicio 2

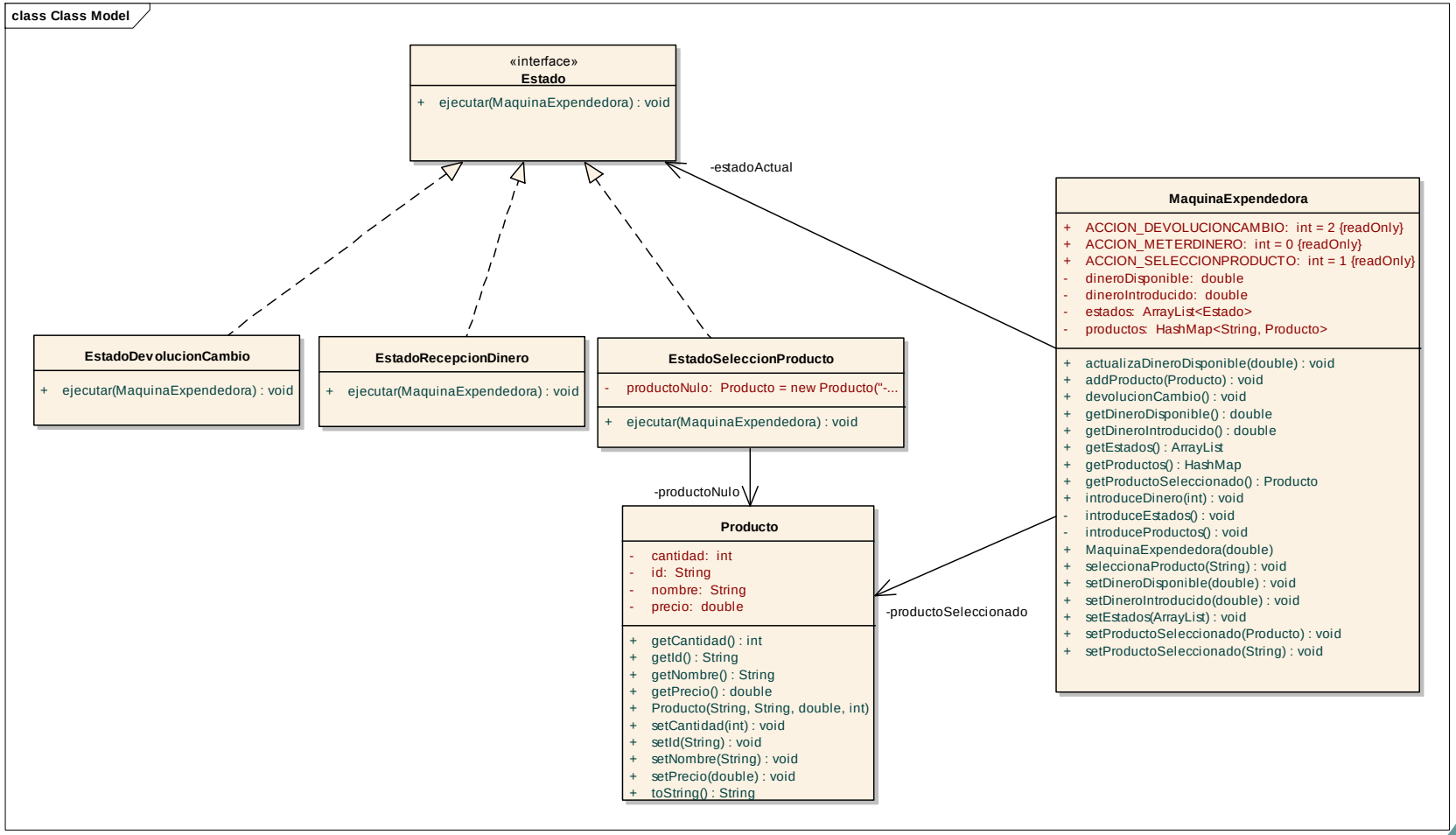
- Se quiere realizar una aplicación que simule el comportamiento de una máquina expendedora de productos alimenticios. La máquina expendedora posee tres estados: recepción del dinero, selección del producto y devolución del cambio en caso de que sea posible. Encontrar el mejor patrón que represente el funcionamiento de la máquina expendedora.

Ejercicio 2. Solución

- El patrón que mejor se adapta a este ejercicio es el State.
  - *Permitir que un objeto se comporte de distinta forma dependiendo de su estado interno, como si cambiase la clase a la que pertenece. Permite cambiar fácilmente el comportamiento de un objeto en tiempo de ejecución.*



Ejercicio 2. Solución



Ejercicio 2. Solución

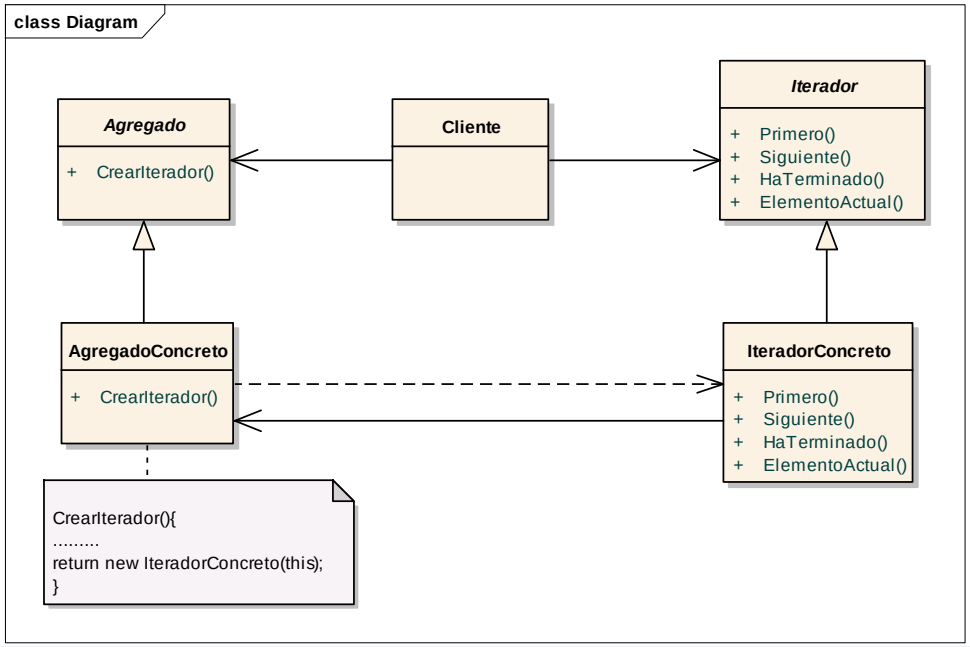
- Identificamos a continuación los elementos del patrón:
  - Contexto: MaquinaExpendedora.
  - Estado: Estado.
  - EstadoConcreto: EstadoDevolucionCambio, EstadoRecepcionDinero, EstadoSeleccionProducto.

Ejercicio 3

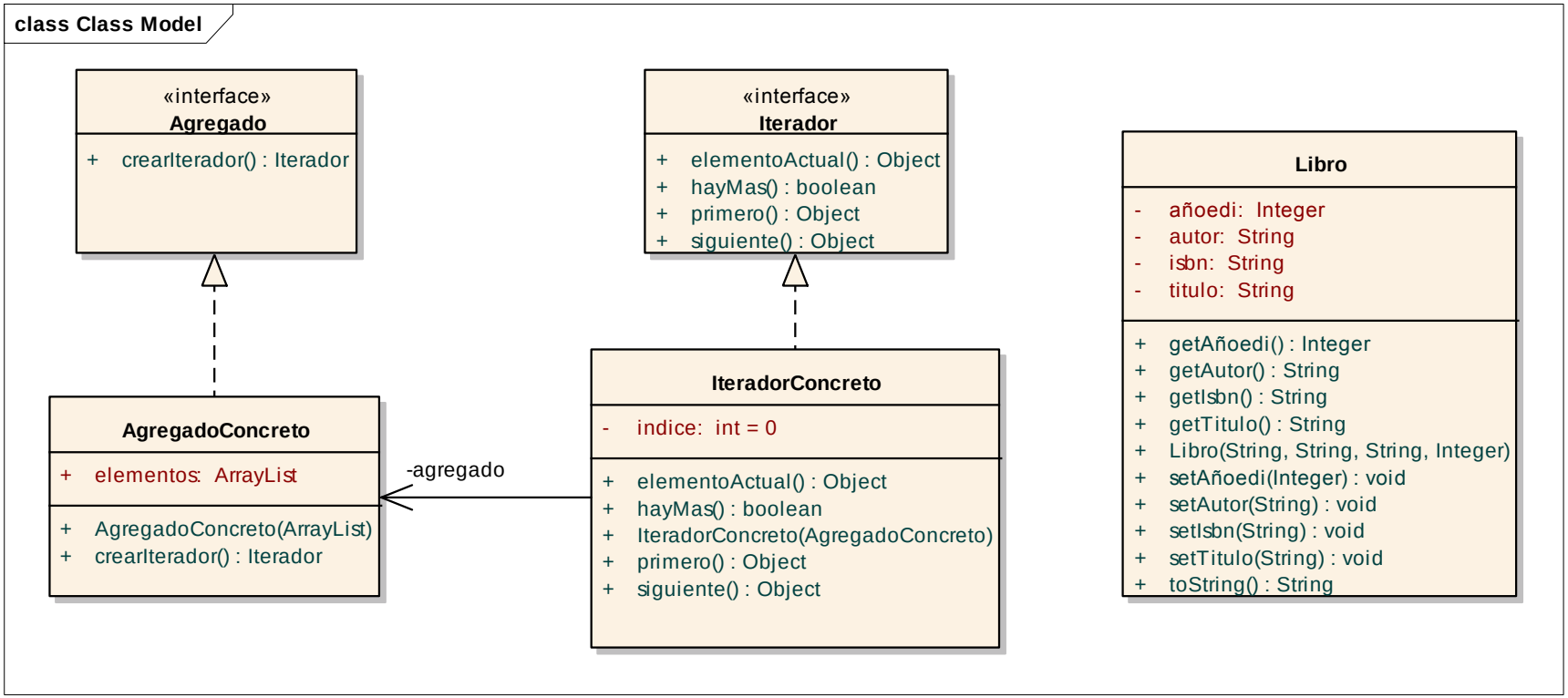
- Se quiere realizar una aplicación para consultar las publicaciones de una biblioteca. Actualmente solo se poseen libros y cada libro tendrá los atributos: año de edición, autor, isbn y título. La aplicación permitirá tener distintos usuarios consultando de forma simultánea la colección de libros de la biblioteca. La aplicación debe ser adaptable a la consulta de nuevos tipos de publicaciones. ¿Qué patrón utilizarías?

Ejercicio 3. Solución

- El patrón que mejor se adapta a este ejercicio es el Iterador.
  - *Proporcionar una forma coherente de acceder secuencialmente a los datos de una colección, independientemente del tipo de colección.*



Ejercicio 3. Solución

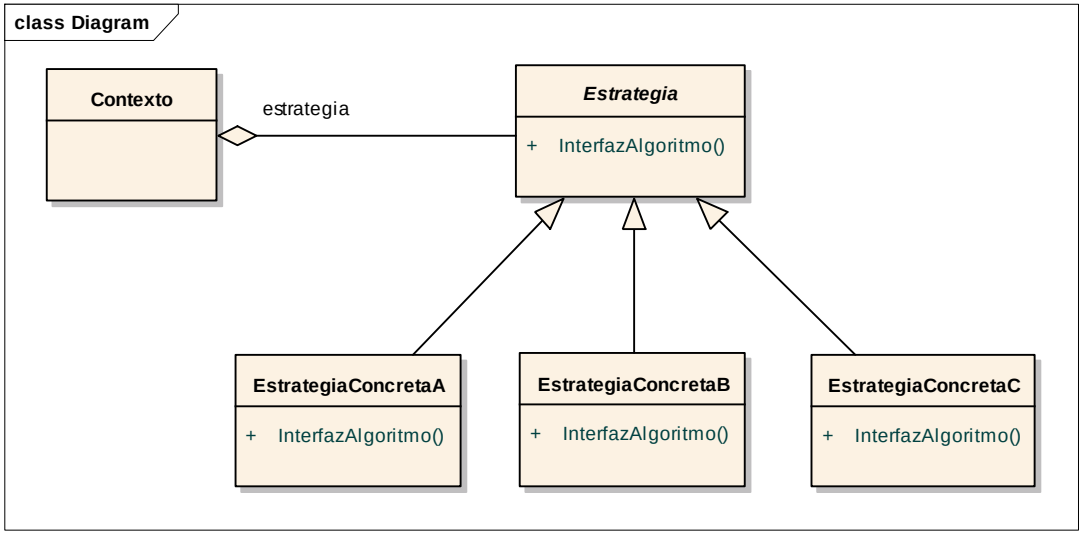


Ejercicio 4.

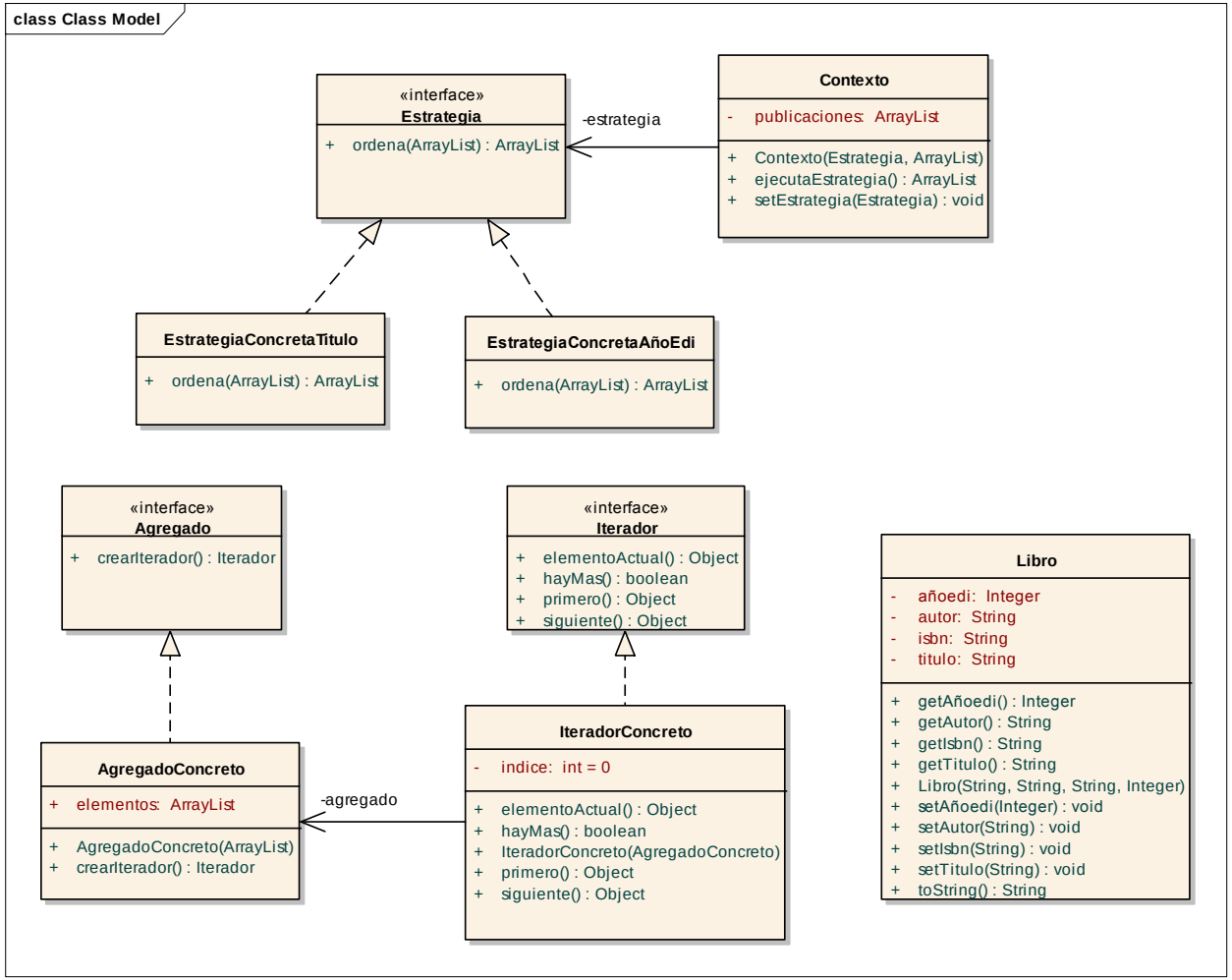
- Se quiere realizar una ampliación del ejercicio 3 de forma que la aplicación pueda presentar los libros ordenados por título y por año de edición y además soportar nuevas formas de ordenación. ¿Qué patrón utilizarías?.

Ejercicio 4. Solución

- El patrón que mejor se adapta a este ejercicio es el Strategy. *Se puede combinar con el Iterador para recorrer la colección ordenada.*
  - *Definir un grupo de clases que representan un conjunto de posibles comportamientos. Estos comportamientos pueden ser fácilmente intercambiados en una aplicación, modificando la funcionalidad en cualquier instante.*



Ejercicio 4. Solución



Ejercicio 4. Solución

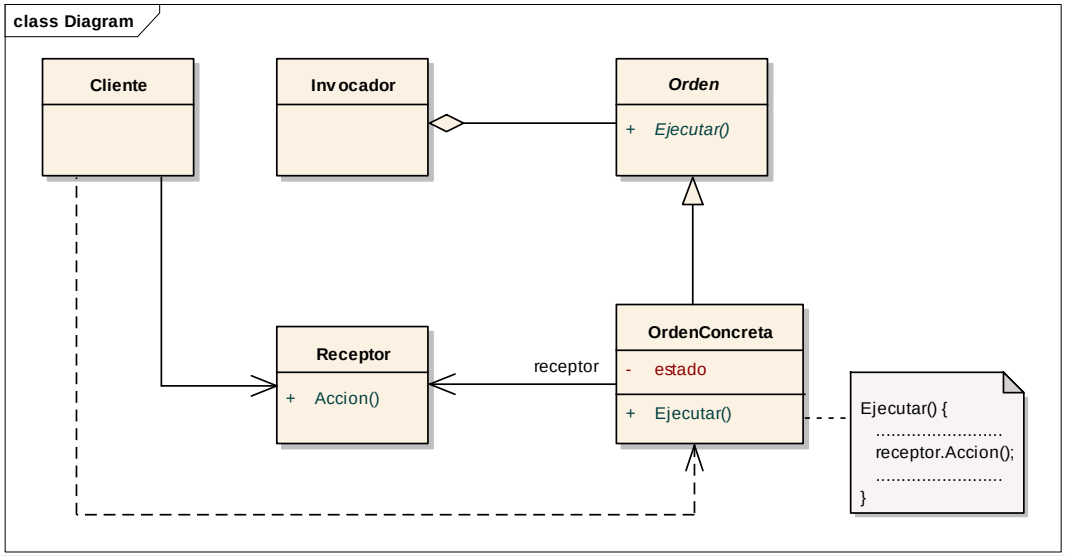
- Identificamos a continuación los elementos del patrón:
  - Contexto: Contexto.
  - Estrategia: Estrategia.
  - EstrategiaConcreta: EstrategiaConcretaTitulo, EstrategiaConcretaAñoEdi.

Ejercicio 5

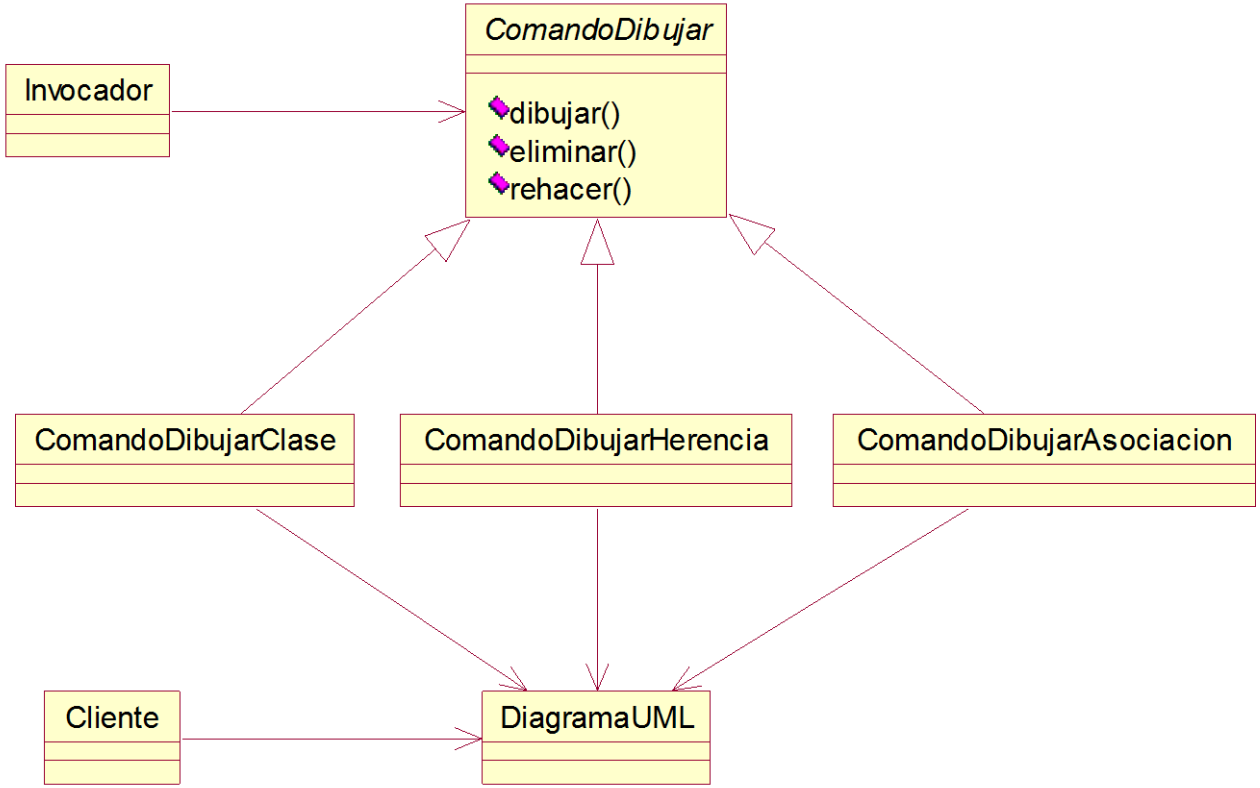
- Se desea implementar un editor gráfico que permite crear diagramas de clase UML. El editor dispone de operaciones tales como “dibujar/eliminar clase", "dibujar/eliminar relación de asociación", “dibujar/eliminar relación de herencia", etc. Describe una solución basada en algún patrón para conseguir que el editor ofrezca una facilidad de deshacer/rehacer órdenes.

Ejercicio 5. Solución

- El patrón que mejor se adapta a este ejercicio es el Command.
  - *Encapsular un comando en un objeto. Este objeto contiene el comportamiento y los datos necesarios para una acción específica. Permite parametrizar a los clientes con diferentes peticiones, hacer cola o llevar un registro de las peticiones. Además permite deshacer operaciones.*



Ejercicio 5. Solución



Ejercicio 5. Solución

- Identificamos a continuación los elementos del patrón:
  - Orden: ComandoDibujar.
  - OrdenConcreta: ComandoDibujarClase, ComandoDibujarHerencia, ComandoDibujarAsiciacion.
  - Invocador: Invocador.
  - Receptor: DiagramUML.
  - Cliente: Cliente.

Ejercicio 6

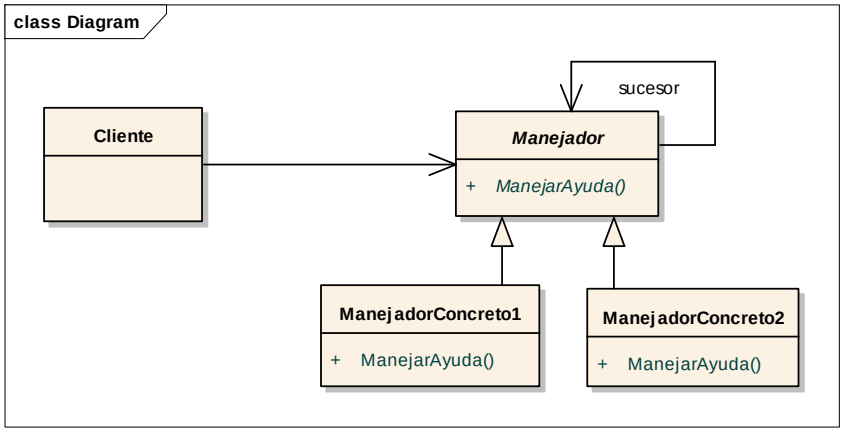
- Se desea implementar un software para un banco y uno de los puntos más importantes es saber quién puede aprobar un crédito. Por lo tanto el banco define las siguientes reglas de negocio:

Cantidad del crédito	**Quien puede aprobar el crédito**
Cantidad <=10.000€	Ejecutivo
10.000€ < Cantidad <= 50.000€	Líder inmediato del ejecutivo
50.000€ < Cantidad <=100.000€	Gerente
Cantidad >100.000€	Director

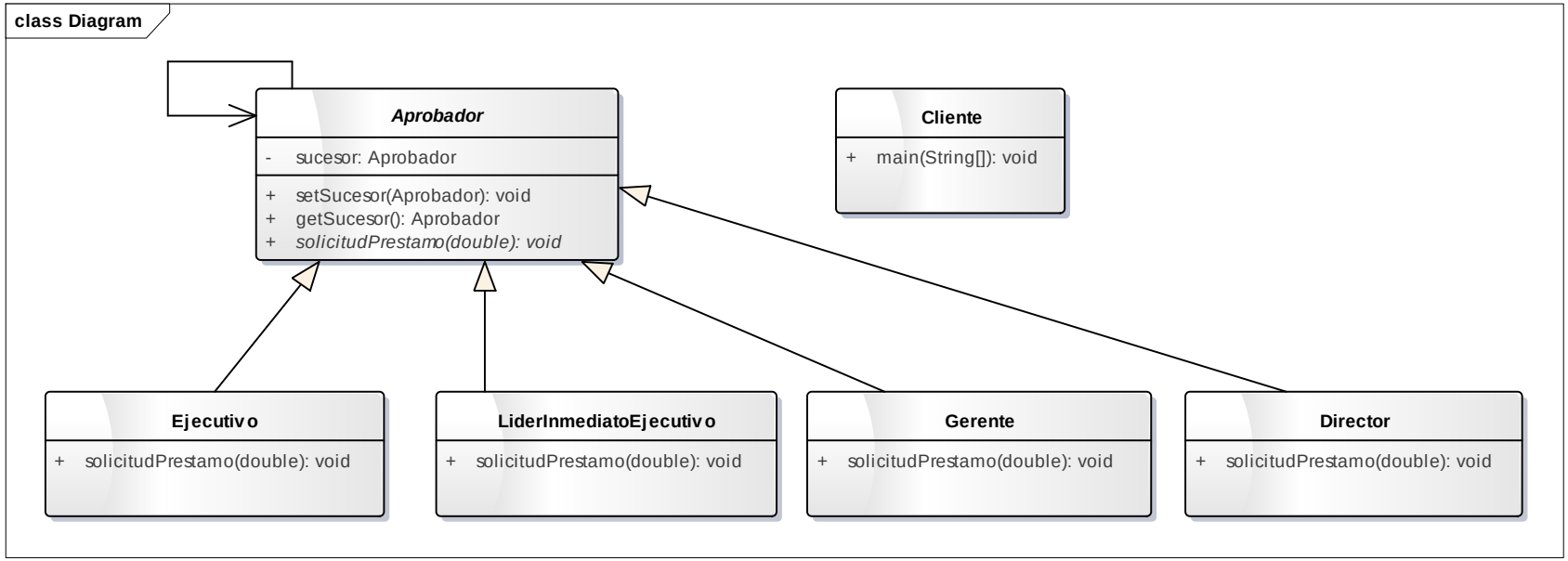
- ¿Qué patrón utilizarías para gestionar la aprobación del crédito?

Ejercicio 6. Solución

- El patrón que mejor se adapta a este ejercicio es el Chain of Responsability .
  - *Desacoplar el emisor de un mensaje del objeto que presta el servicio. Para conseguir esto se establece una cadena en el sistema de forma que el emisor envía un mensaje al primer elemento de la cadena, éste lo procesa si puede, y sino, lo redirige a otro objeto de la cadena hasta que uno pueda procesarlo.*



Ejercicio 6. Solución



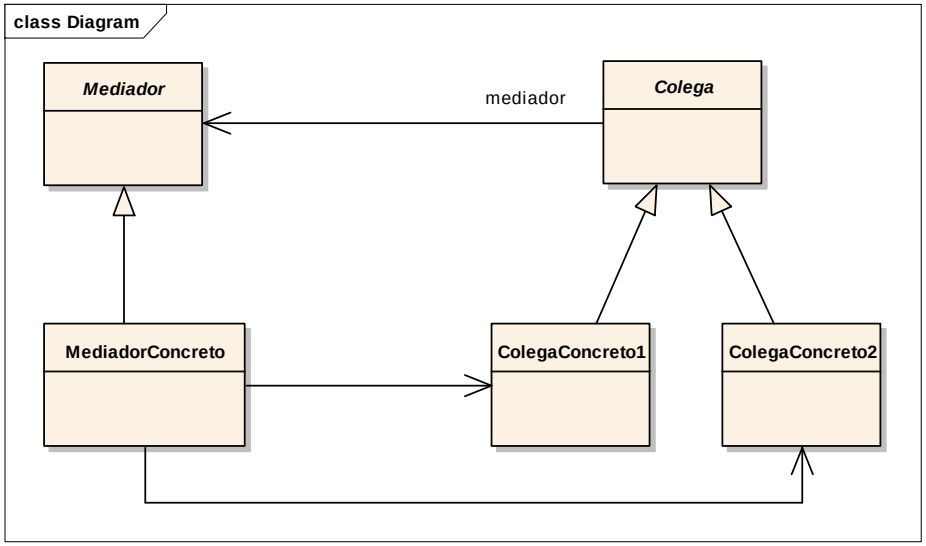
- Identificamos a continuación los elementos del patrón:
  - Manejador: Aprobador.
  - ManejadorConcreto: Ejecutivo, LiderInmediatoEjecutivo, Gerente, Director.
  - Cliente: Cliente.

Ejercicio 7

- Una ventana de diálogo es una colección de controles gráficos y no gráficos. La clase Dialogo provee un mecanismo para facilitar la interacción entre controles. Por ejemplo, cundo se selecciona un nuevo valor en un objeto ComboBox un objeto Label tiene que mostrar el nuevo valor seleccionado. Ambos, ComboBox y Label desconocen la estructura y existencia del otro y todas las interacciones son manejadas por el objeto Dialogo. ¿De qué patrón se trata?

Ejercicio 7. Solución

- Se trata del patrón Mediator.
  - *Proporcionar la posibilidad de que un único objeto se encargue de gestionar la comunicación entre distintos objetos, sin que éstos necesiten conocerse entre sí. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente.*



Ejercicio 8

- Uno de los patrones relacionados con Memento es Command: las órdenes (Command) pueden usar recuerdos para guardar el estado de las operaciones que pueden deshacerse.
- Partiendo del ejemplo Command que aparece en el proyecto PatronesComportamiento, implementar deshacer usando Memento.

Ejercicio 8. Solución

Clase_ComandoCambiarFecha

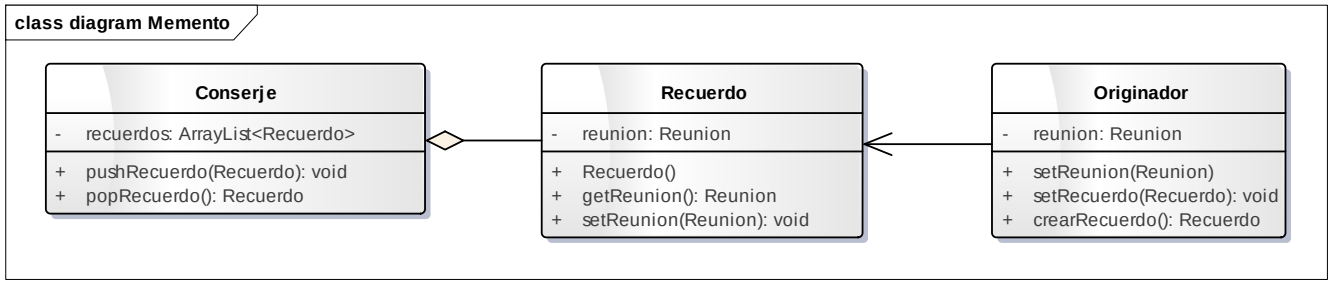
```
private Reunion reunion;
private Conserje conserje = new Conserje();

@Override
  public void ejecutar(String param) {
     Originador originador = new Originador();
     originador.setReunion(reunion);
     conserje.pushRecuerdo(originador.crearRecuerdo());
     reunion.setFecha(new FechaHora(param));
  }
  @Override
  public void deshacer() {
     Recuerdo recuerdo = conserje.popRecuerdo();
     if (recuerdo != null){
        Originador originador = new Originador();
        originador.setReunion(reunion);
        conserje.pushRecuerdo(originador.crearRecuerdo());
        reunion.setFecha(recuerdo.getReunion().getFecha());
     }
  }
```

Clase ComandoCambiarLocalizacion

```
private Reunion reunion;
private Conserje conserje = new Conserje();

@Override
   public void ejecutar(String param) {
      Originador originador = new Originador();
      originador.setReunion(reunion);
      conserje.pushRecuerdo(originador.crearRecuerdo());
      reunion.setLocalizacion(param);
   }
   @Override
   public void deshacer() {
      Recuerdo recuerdo = conserje.popRecuerdo();
      if (recuerdo != null){
         Originador originador = new Originador();
         originador.setReunion(reunion);
         conserje.pushRecuerdo(originador.crearRecuerdo());
reunion.setLocalizacion(recuerdo.getReunion().getLocalizacion());
      }
   }
```

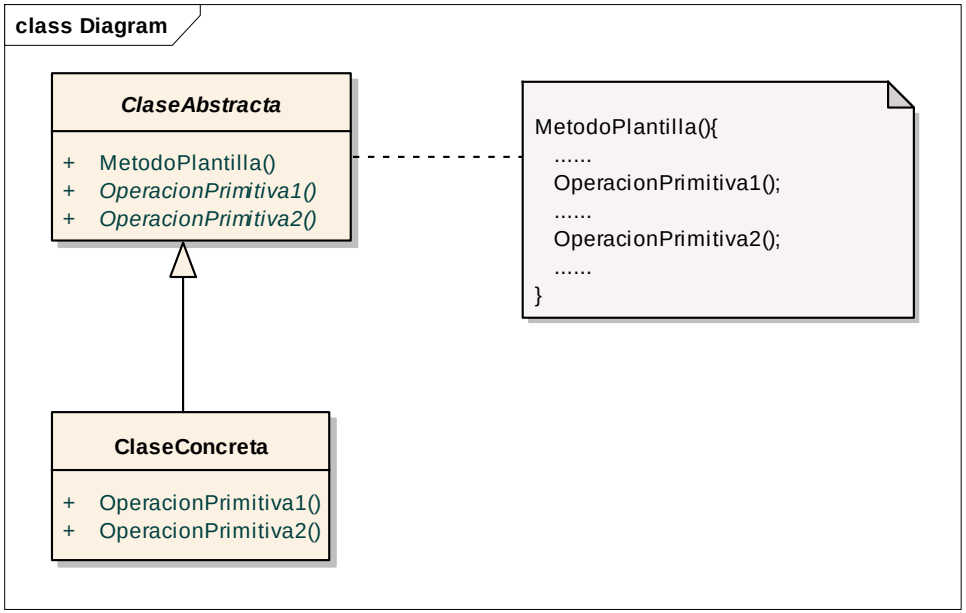


Ejercicio 9

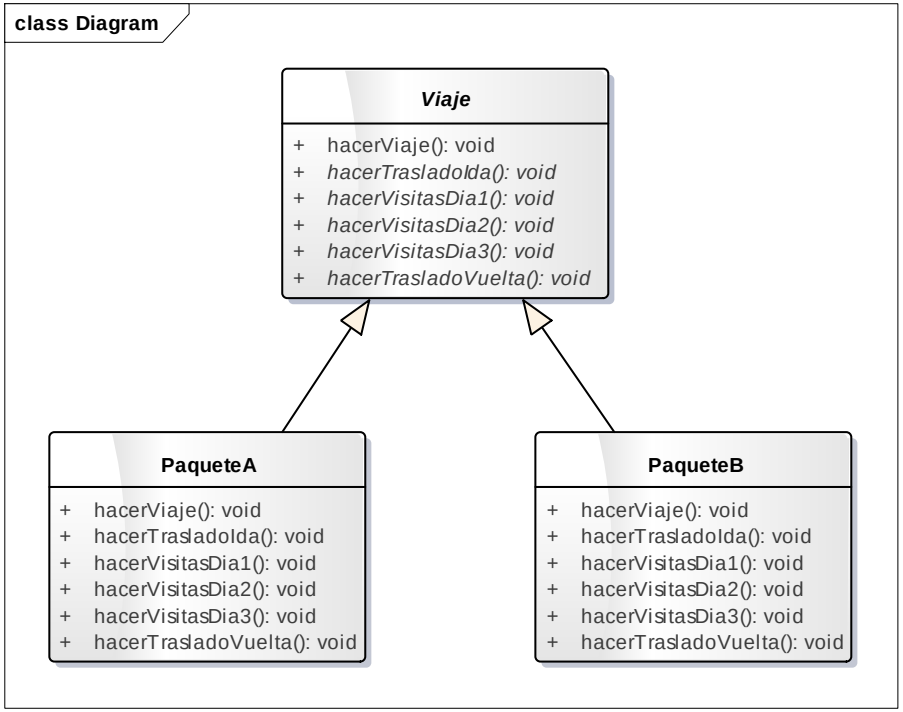
- Queremos desarrollar una aplicación para una agencia de viajes. Todos los viajes tienen un comportamiento común, pero hay varios paquetes diferentes. Por ejemplo, cada viaje tiene los siguientes pasos en común:
  - El turista es transportado al lugar de vacaciones por avión, tren, barco…
  - Cada día se visita algo.
  - El turista es transportado de vuelta.
- ¿Qué patrón utilizarías para la gestión de los diferentes paquetes?

Ejercicio 9. Solución

- El patrón que mejor se adapta a este ejercicio es Template Method.
  - *Proporcionar un método que permita que las subclases redefinan partes del método sin reescribirlo.*



Ejercicio 9. Solución



- Identificamos a continuación los elementos del patrón:
  - ClaseAbstracta: Viaje.
  - ClaseConcreta: PaqueteA, PaqueteB.