Patrones de comportamiento con PHP, Parte I

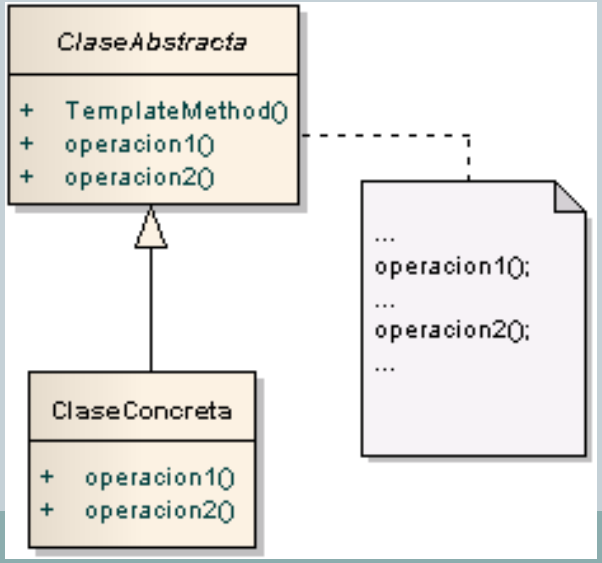
ABRAHAM SÁNCHEZ LÓPEZ
GRUPO MOVIS
FCC-BUAP

Introducción

- El diseñador de un sistema orientado a objetos se enfrenta a menudo al problema del descubrimiento de objetos.
- Esto puede realizarse a partir de los dos aspectos siguientes:
  - La estructuración de los datos.
  - La distribución de los procesamientos y de los algoritmos.
- Los patrones de estructuración aportan soluciones a los problemas de estructuración de datos y de objetos.
- El objetivo de los patrones de comportamiento consiste en proporcionar soluciones para distribuir el procesamiento y los algoritmos entre los objetos.
- Estos patrones organizan los objetos así como sus interacciones especificando los flujos de control y de procesamiento en el seno de un sistema de objetos.

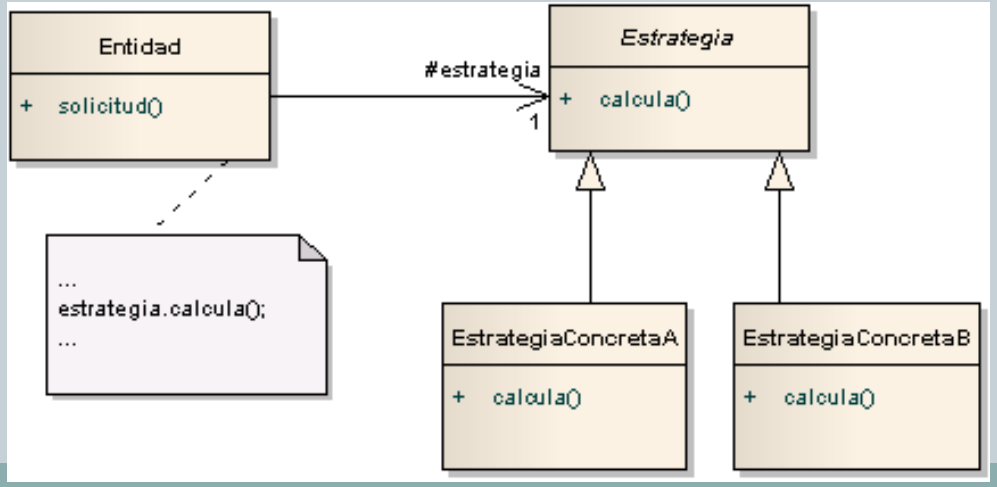
Distribución por herencia o delegación, I

- Un primer enfoque para distribuir un procesamiento consiste en repartirlo en subclases.
- Este reparto se realiza mediante el uso en la clase de métodos abstractos que se implementan en las subclases.
- Como una clase puede poseer varias subclases, este enfoque habilita la posibilidad de obtener variantes en las partes descritas en las subclases. Esto se lleva a cabo mediante el patrón Template Method tal y como se ilustra en la siguiente figura.



Distribución por herencia o delegación, II

- Una segunda posibilidad de reparto se lleva a cabo mediante la distribución de procesamiento en los objetos cuyas clases son independientes.
- En este enfoque, un conjunto de objetos que cooperan entre ellos concurren a la realización de un procesamiento o de un algoritmo.
- El patrón Strategy ilustra este mecanismo en la siguiente figura. El método *solicita* de la clase Entidad invoca para realizar su procesamiento al método *calcula* especificado mediante la interfaz Estrategia. Cabe observar que esta última puede tener varias implementaciones.



Distribución por herencia o delegación, III

- La siguiente tabla, indica para cada patrón de comportamiento, el tipo de reparto utilizado.

Chain of Responsibility	Delegación
Command	Delegación
Interpreter	Herencia
Iterator	Delegación
Mediator	Delegación
Memento	Delegación
Observer	Delegación
State	Delegación
Strategy	Delegación
Template Method	Herencia
Visitor	Delegación

El patrón Chain of Responsibility, I

Descripción

- El patrón Chain of Responsibility construye una cadena de objetos tal que si un objeto de la cadena no puede responder a la solicitud, puede transmitirla a su sucesor y así sucesivamente hasta que uno de los objetos de la cadena responde.

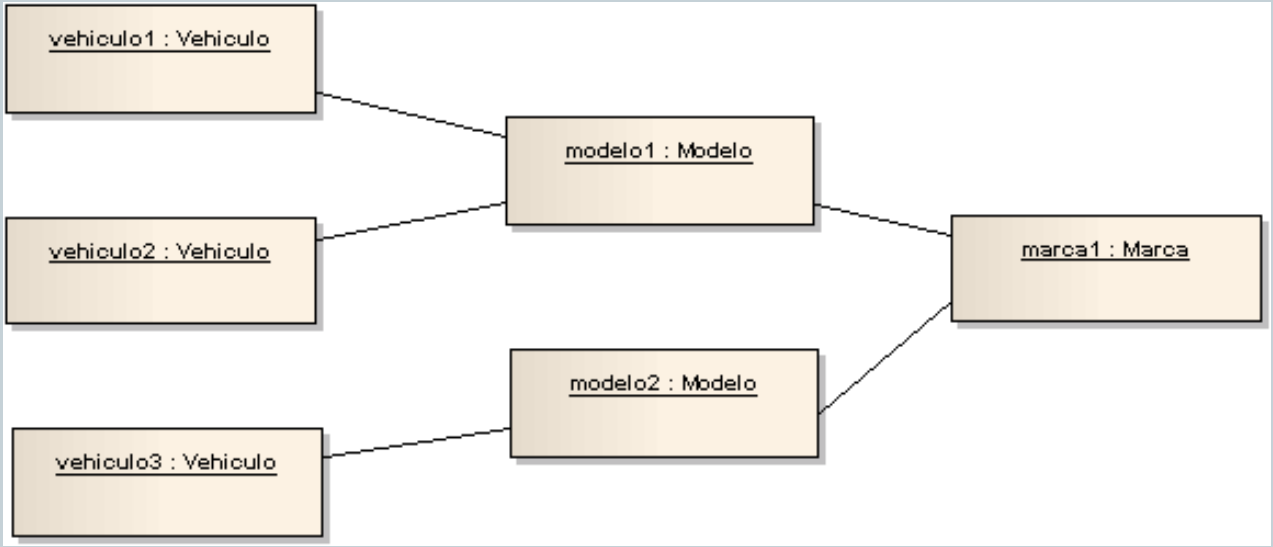
Ejemplo

- Nos situamos en el marco de la venta de vehículos de ocasión. Cuando se muestra el catálogo de vehículos, el usuario puede solicitar una descripción de uno de los vehículos a la venta.
- Si no se encuentra esta descripción, el sistema debe reenviar la descripción asociada al modelo de este vehículo.
- Si de nuevo esta descripción no se encuentra, se debe reenviar la descripción asociada a la marca del vehículo.
- Si tampoco existe una descripción asociada a la marca entonces se envía una descripción por defecto.

El patrón Chain of Responsibility, II

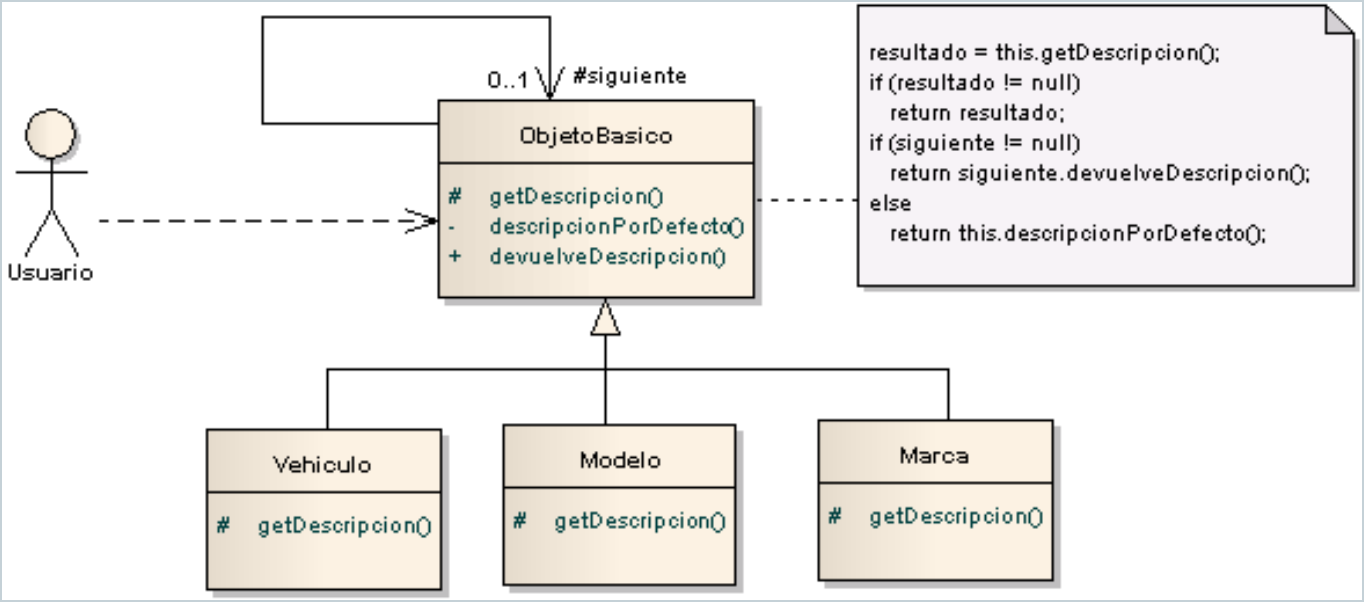
- De este modo, el usuario recibe la descripción más precisa disponible en el sistema.
- El patrón Chain of Responsibility proporciona una solución para llevar a cabo este mecanismo.
- Consiste en enlazar los objetos entre ellos desde el más específico (el vehículo) al más general (la marca) para formar la cadena de responsabilidad.
- La solicitud de la descripción se transmite a lo largo de la cadena hasta que un objeto pueda procesarla y enviar la descripción.
- El diagrama de objetos UML de la figura del acetato 8 ilustra esta situación y muestra las distintas cadenas de responsabilidad (de izquierda a derecha).
- La figura del acetato 9 representa el diagrama de clases del patrón Chain of Responsibility aplicado al ejemplo.
- Los vehículos, modelos y marcas se describen mediante subclases concretas de la clase ObjetoBasico. Esta clase abstracta incluye la asociación siguiente que implementa la cadena de responsabilidad. Incluye a su vez tres métodos:

El patrón Chain of Responsibility, III



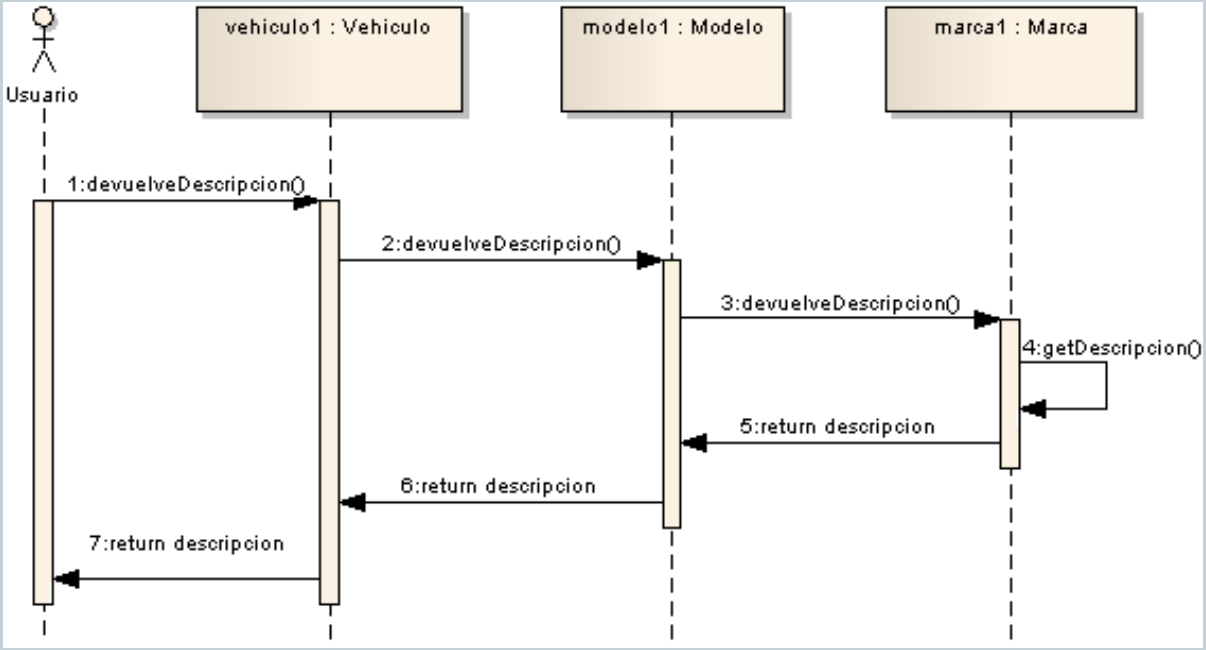
- getDescripcion es un método abstracto. Está implementado en las subclases concretas. Esta implementación debe devolver la descripción si existe o bien el valor null en caso contrario.
- descripcionPorDefecto devuelve un valor de descripción por defecto, válido para todos los vehículos del catálogo.
- devuelveDescripcion es el método público destinado al usuario. Invoca al método getDescripcion. Si el resultado es null, entonces si existe un objeto siguiente se invoca a su método devuelveDescripcion, en caso contrario se utiliza el método descripcionPorDefecto.

El patrón Chain of Responsibility, IV



- La figura del acetato 10 muestra un diagrama de secuencia que es un ejemplo de solicitud de una descripción basada en el diagrama de objetos de la figura del acetato 8.
- En este ejemplo, ni el vehículo ni el modelo1 poseen una descripción. Solo la marca1 posee una descripción, la cual se utiliza para el vehiculo1.

El patrón Chain of Responsibility, V

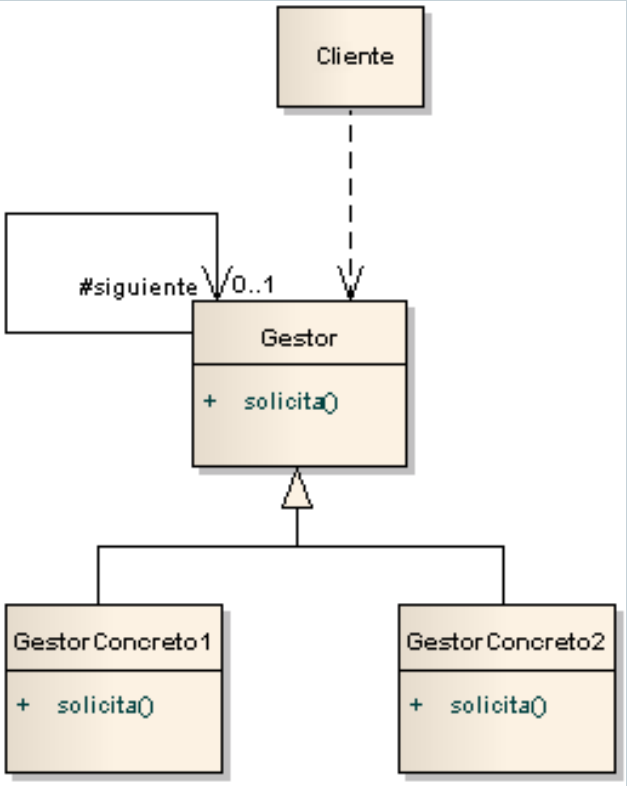


Estructura

1. Diagrama de clases

- La figura del acetato 11 describe la estructura genérica del patrón.

El patrón Chain of Responsibility, VI



2. Participantes

- Los participantes del patrón son los siguientes:

El patrón Chain of Responsibility, VII

- Gestor (ObjetoBasico) es una clase abstracta que implementa bajo la forma de una asociación la cadena de responsabilidad así como la interfaz de las solicitudes.
- GestorConcreto1 y GestorConcreto2 (Vehículo, Modelo y Marca) son las clases concretas que implementan el procesamiento de las solicitudes utilizando la cadena de responsabilidad si no pueden procesarlas.
- Cliente (Usuario) inicia la solicitud inicial en un objeto de una de las clases GestorConcreto1 o GestorConcreto2.

3. Colaboraciones

- El cliente realiza la solicitud inicial a un gestor. La solicitud se propaga a lo largo de la cadena de responsabilidad hasta el momento en el que uno de los gestores puede procesarla.

Dominios de aplicación

- El patrón se utiliza en los siguientes casos:
  - Una cadena de objetos gestiona una solicitud según un orden que se define dinámicamente.
  - La forma en que una cadena de objetos gestiona una solicitud no tiene por qué conocerse en sus clientes.

El patrón Chain of Responsibility, VIII

Ejemplo en PHP

- Presentamos a continuación un ejemplo escrito en PHP.
- La clase ObjetoBasico se describe a continuación. Implementa la cadena de responsabilidad mediante el atributo siguiente cuyo valor podemos fijar mediante el método setSiguiente.
- Los demás métodos corresponden con las especificaciones presentadas en la figura del acetato 9.

El patrón Command, I

Descripción

- El patrón Command tiene como objetivo transformar una solicitud en un objeto, facilitando operaciones tales como la anulación, el encolamiento de solicitudes y su seguimiento.

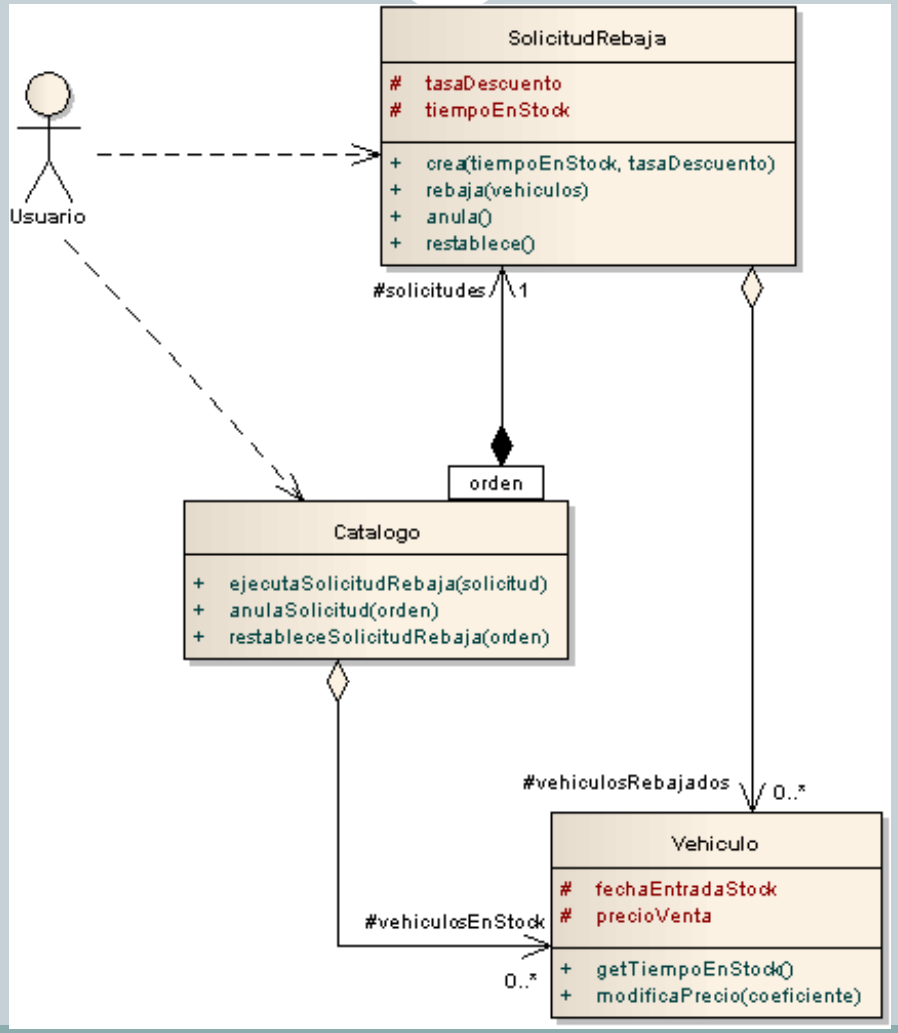
Ejemplo

- En ciertos casos, la gestión de una solicitud puede ser bastante compleja: puede ser anulable, encolada o trazada.
- En el marco del sistema de venta de vehículos, el gestor puede solicitar al catálogo rebajar el precio de los vehículos de ocasión que llevan en el stock cierto tiempo. Por motivos de simplicidad, esta solicitud debe poder ser anulada y, eventualmente, restablecida.
- Para gestionar esta anulación, una primera solución consiste en indicar a nivel de cada vehículo si está o no rebajado.
- Esta solución no es suficiente, pues un mismo vehículo puede estar rebajado varias veces con tasas diferentes.
- Otra solución sería conservar su precio antes de la última rebaja, aunque esta solución no es satisfactoria pues es la anulación puede realizarse sobre otra solicitud de rebaja que no sea la última.

El patrón Command, II

- El patrón Command resuelve este problema transformando la solicitud en un objeto cuyos atributos van a contener los parámetros así como el conjunto de objetos sobre los que la solicitud va a ser aplicada.
- En nuestro ejemplo, esto hace posible anular o restablecer una solicitud de rebaja.
- La figura del acetato 16 ilustra esta aplicación del patrón Command a nuestro ejemplo.
- La clase SolicitudRebaja almacena sus dos parámetros (tasaDescuento y tiempoEnStock) así como la lista de vehículos para los que se ha aplicado el descuento (asociación vehiculosRebajados).
- Conviene observar que el conjunto de vehículos referenciados por SolicitudRebaja es un subconjunto del conjunto de vehículos referenciados por Catálogo.
- Durante la llamada al método ejecutaSolicitudRebaja, la solicitud pasada como parámetro se ejecuta y, a continuación, se almacena en un orden tal que la última solicitud almacenada se encuentra en la primera posición.

El patrón Command, III



El patrón Command, IV

- El diagrama de la figura del acetato 18 muestra un ejemplo de secuencia de llamadas.
- Los dos parámetros proporcionados al constructor de la clase SolicitudRebaja son la tasa de descuento y la duración mínima de almacenamiento, expresada en meses.
- Por otro lado, el parámetro orden del método anulaSolicitudRebaja vale cero para la última solicitud ejecutada, uno para la penúltima, etc.
- Las interacciones entre las instancias de SolicitudRebaja y de Vehiculo no están representadas, con el fin de simplificar.
- Para comprender bien su funcionamiento, conviene revisar el código PHP presentado más adelante.

Estructura

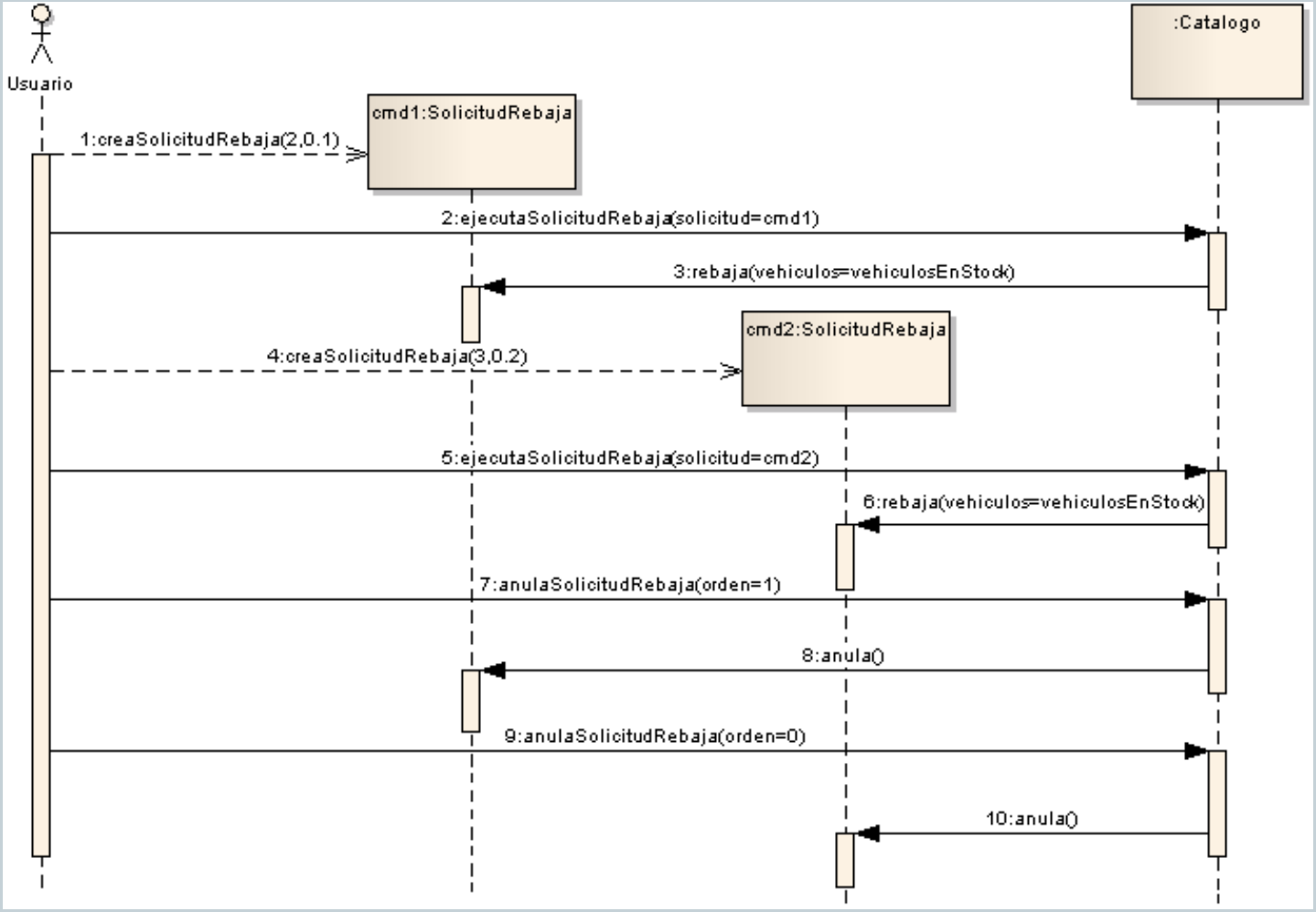
1. Diagrama de clases

- La figura del acetato 19 detalla la estructura genérica del patrón.

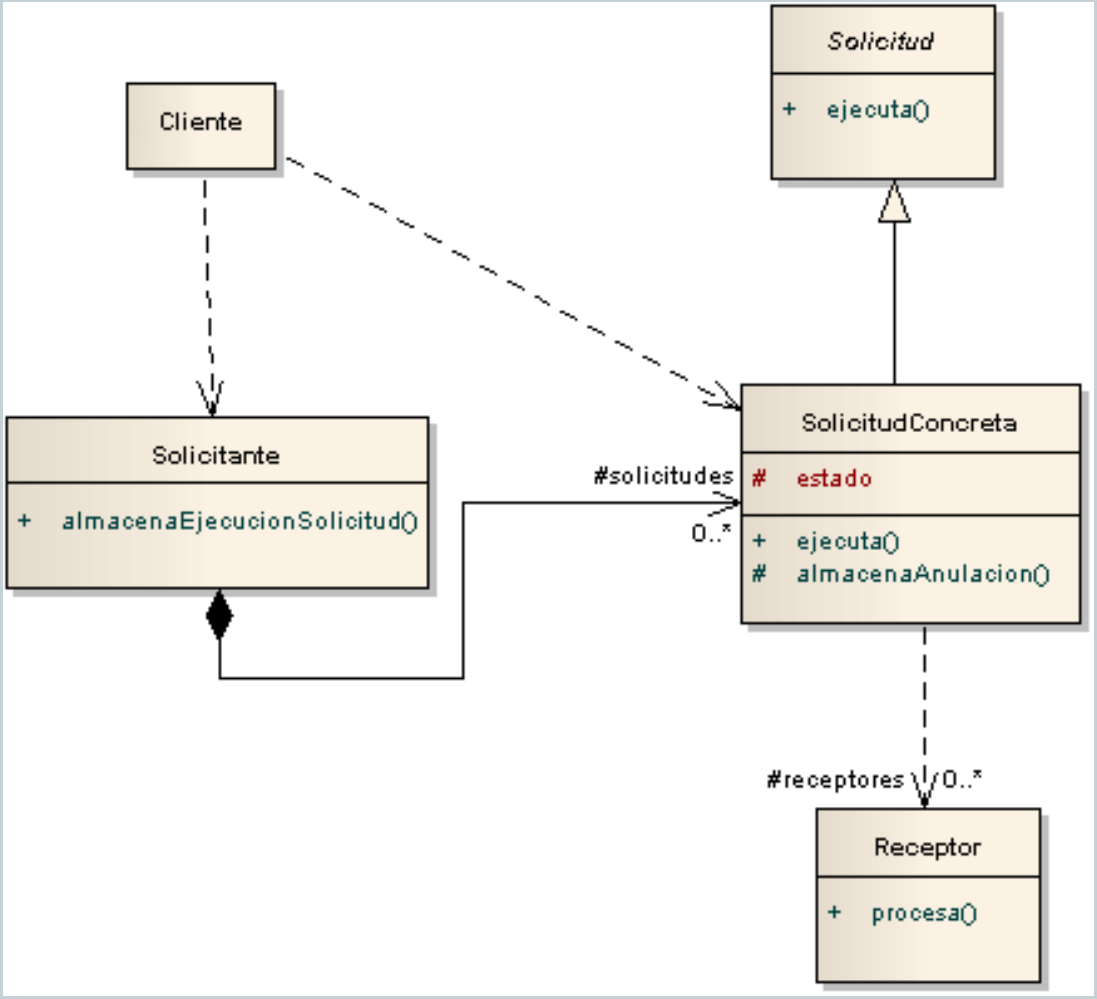
2. Participantes

- Los participantes del patrón son los siguientes:

El patrón Command, V



El patrón Command, VI



El patrón Command, VII

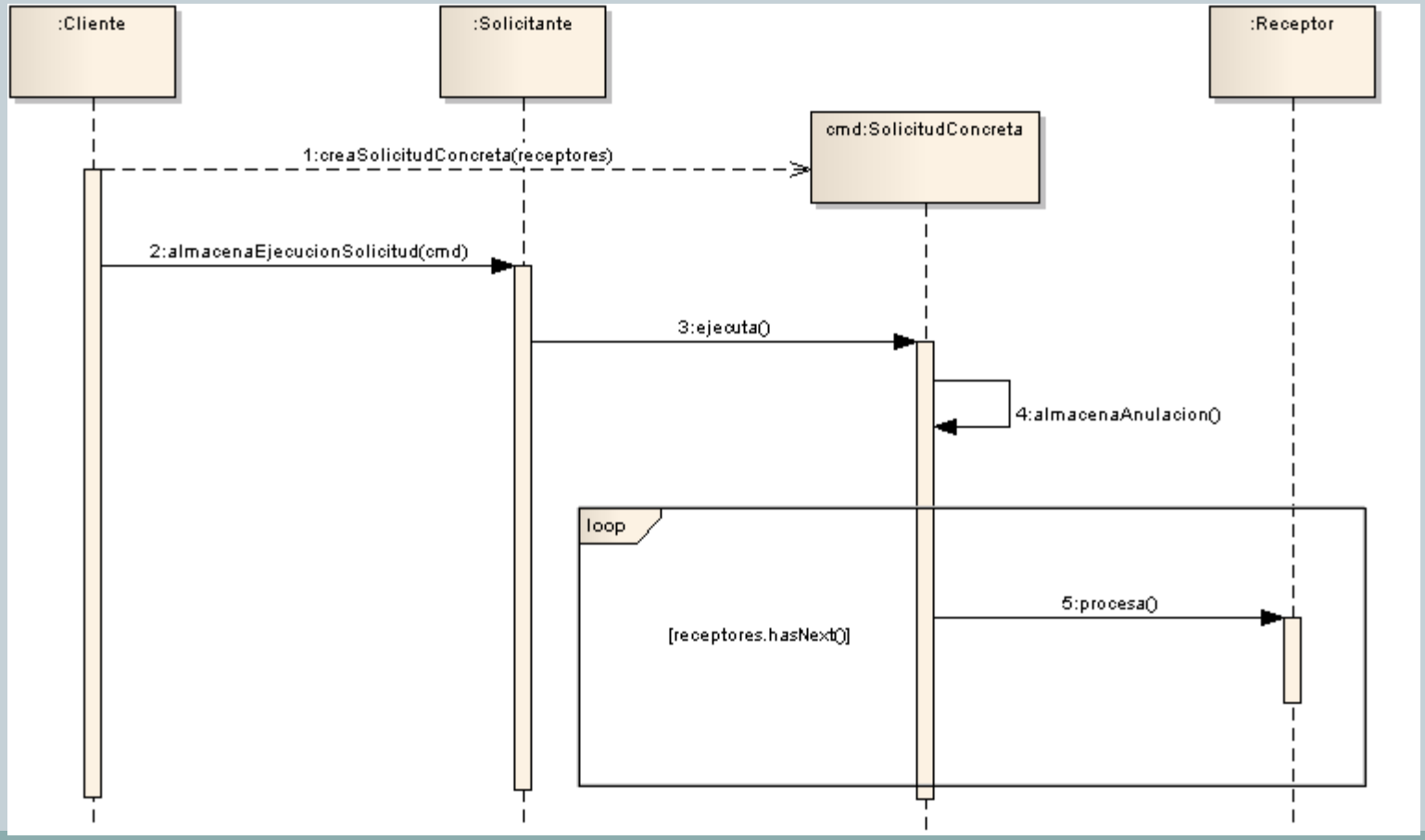
- Solicitud es la interfaz que presenta la firma del método *ejecuta* que ejecuta la solicitud.
- SolicitudConcreta (SolicitudRebaja) implementa el método *ejecuta*, gestiona la asociación con el o los receptores e implementa el método almacenaAnulacion que almacena el estado (o los valores necesarios) para poder anularla a continuación.
- Cliente (Usuario) crea e inicializa la solicitud y la transmite al solicitante.
- Solicitante (Catálogo) almacena y ejecuta la solicitud (método almacenaEjecucionSolicitud) así como eventualmente las solicitudes de anulación.
- Receptor (Vehículo) ejecuta las acciones necesarias para realizar la solicitud o para anularla.

3. Colaboraciones

- La figura del acetato 21 ilustra las colaboraciones del patrón Command:
  - El cliente crea una solicitud concreta especificando el o los receptores.
  - El cliente transmite esta solicitud al método almacenaEjecucionSolicitud del solicitante para almacenar la solicitud.
  - El solicitante ejecuta a continuación la solicitud llamando al método ejecuta.
  - El estado o los datos necesarios para realizar la anulación se almacenan (método almacenaAnulacion).

El patrón Command, VIII

- La solicitud pide al o a los receptores que realicen las acciones correspondientes.



El patrón Command, IX

Dominios de aplicación

- El patrón se utiliza en los siguientes casos:
- Un objeto debe configurarse para realizar un procesamiento concreto. En el caso del patrón Command, es el solicitante el que se configura mediante una solicitud que contiene la descripción de un procesamiento a realizar sobre uno o varios receptores.
- Las solicitudes deben encolarse y poder ejecutarse en un momento cualquiera, eventualmente varias veces.
- Las solicitudes pueden ser anuladas.
- Las solicitudes deben quedar registradas en un archivo de log.
- Las solicitudes deben estar reagrupadas bajo la forma de una transacción. Una transacción es un conjunto ordenado de solicitudes que actúan sobre el estado de un sistema y pueden ser anuladas.

Ejemplo en PHP

- Presentamos a continuación un ejemplo escrito en PHP. La clase Vehiculo se describe en PHP como aparece en el código fuente. Cada vehículo posee un nombre, una fecha de entrada en el almacén y un precio de venta. El método modificaPrecio permite ajustar el precio mediante un coeficiente.

El patrón Interpreter, I

Descripción

- El patrón Interpreter proporciona un marco para representar mediante objetos la gramática de un lenguaje con el fin de evaluar, interpretándolas, expresiones escritas en este lenguaje.

Ejemplo

- Queremos crear un pequeño motor de búsqueda de vehículos con ayuda de expresiones booleanas según una gramática muy sencilla que se muestra a continuación:

  expresión ::= termino || palabra-clave || (expresión)

  termino ::= factor ‘o’ factor

  factor ::= expresión ‘y’ expresión

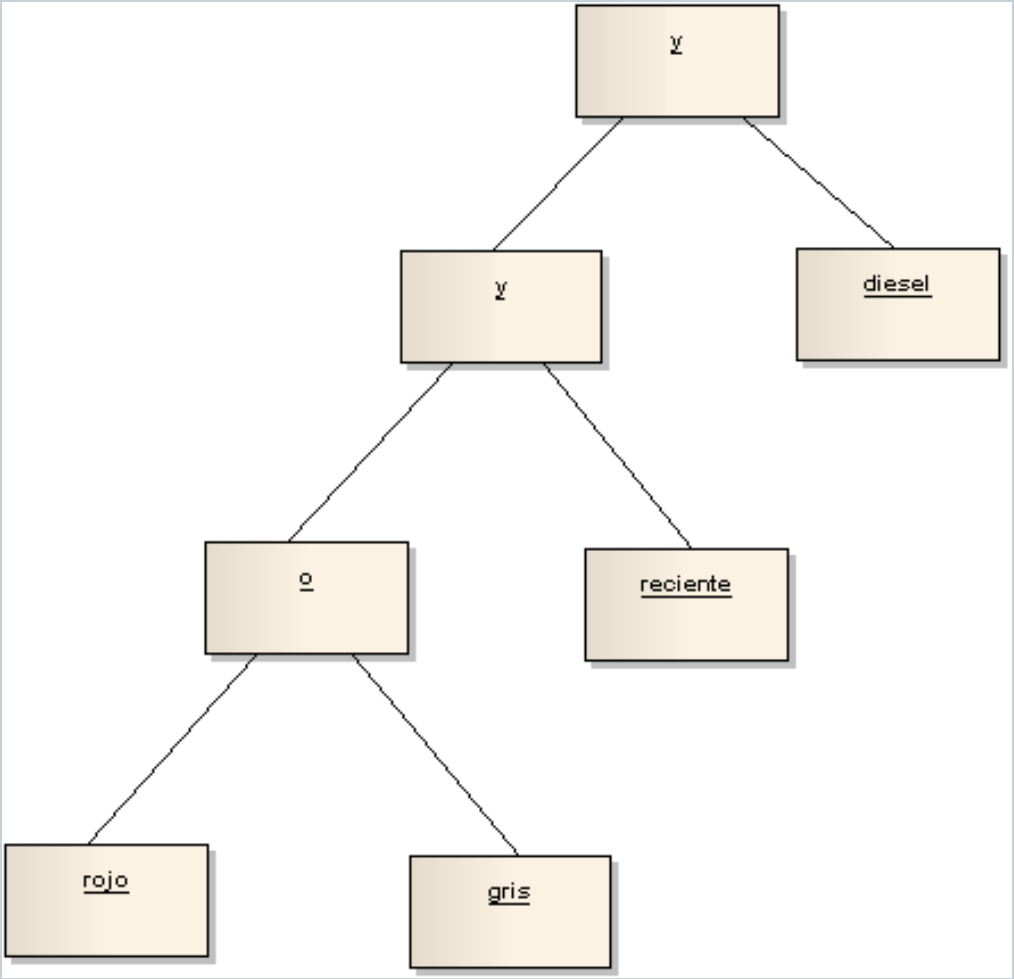
  palabra-clave ::== ‘a’.. ‘z’, ‘A’..’Z’ {‘a’.. ‘z’, ‘A’..’Z’ }*

- Los símbolos entre comillas son símbolos terminales. Los símbolos no terminales son expresión, término, factor y palabra-clave. El símbolo de partida es expresión.

El patrón Interpreter, II

- Vamos a implementar el patrón Interpreter para poder expresar cualquier expresión que responda a esta gramática según un árbol sintáctico constituido por objetos con el objetivo de poder evaluarla e interpretarla.
- Tal árbol está constituido únicamente por símbolos terminales. Para simplificar, consideramos que una palabra-clave constituye un símbolo terminal en tanto que es una cadena de caracteres.
- La expresión (rojo o gris) y reciente y diesel se van a traducir por el árbol sintáctico de la figura del acetato 25
- La evaluación de tal árbol para la descripción del vehículo se realiza comenzando por la cima.
- Cuando un nodo es un operador, la evaluación se realiza calculando de forma recursiva el valor de cada subárbol (primero el de la izquierda y después el de la derecha) y aplicando el operador. Cuando un nodo es una palabra-clave, la evaluación se realiza buscando la cadena correspondiente en la descripción del vehículo.

El patrón Interpreter, III



El patrón Interpreter, IV

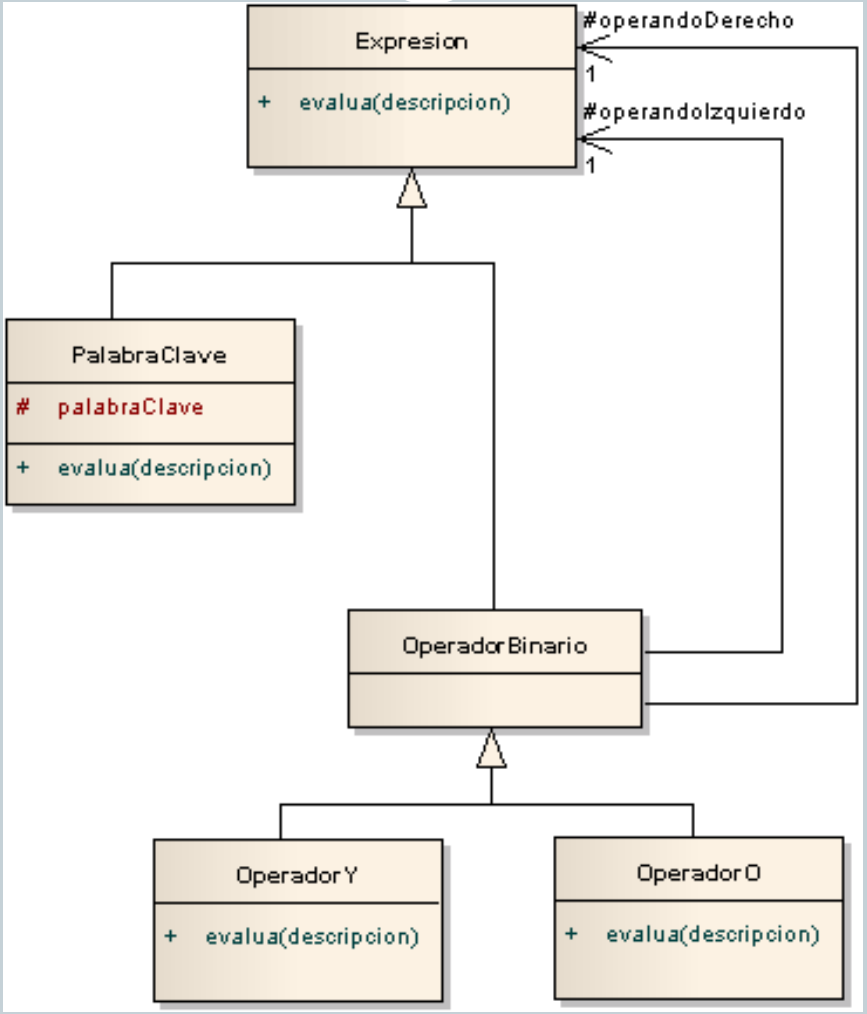
- El motor de búsqueda consiste por lo tanto en evaluar la expresión para cada descripción y en reenviar la lista de vehículos para los que la evaluación es verdadera.
- Esta técnica de búsqueda no está optimizada, y por lo tanto sólo es válida para una cantidad pequeña de vehículos.
- El diagrama de clases que permite describir los árboles sintácticos como el de la figura del acetato 25 está representado en la figura del acetato 27.
- El método *evalua* permite evaluar la expresión para una descripción de un vehículo que se pasa como parámetro.

Estructura

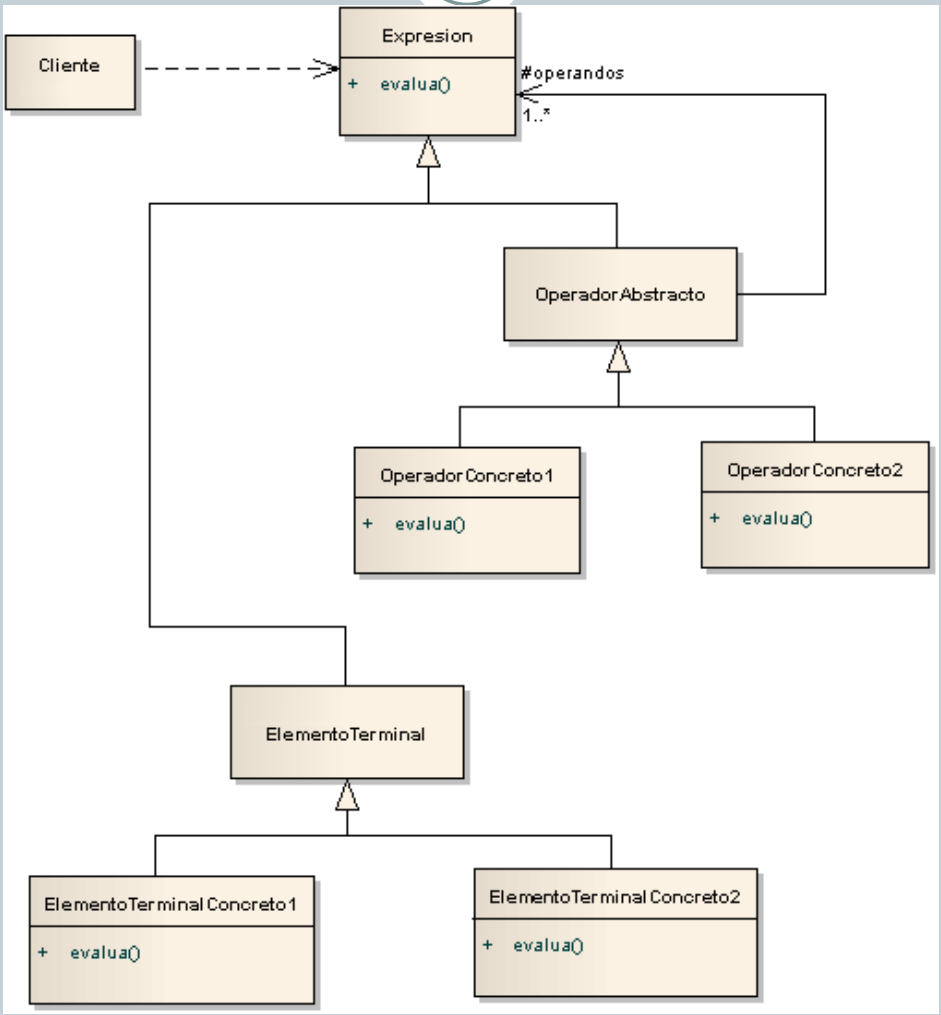
1. Diagrama de clases

- La figura del acetato 28 describe la estructura genérica del patrón.
- Este diagrama de clases muestra que existen dos tipos de sub-expresiones, es decir:

El patrón Interpreter, V



El patrón Interpreter, VI



El patrón Interpreter, VII

- Los elementos terminales que pueden ser nombres de variables, enteros, números reales.
- Los operadores que pueden ser binarios como en el ejemplo, unarios (operador -) o que tomen más argumentos como las funciones.

2. Participantes

- Los participantes del patrón son los siguientes:
  - Expresion es una clase abstracta que representa cualquier tipo de expresión, es decir cualquier nodo del árbol sintáctico.
  - OperadorAbstracto (OperadorBinario) es también una clase abstracta. Describe cualquier nodo del tipo operador, es decir que posea operandos que son sub-árboles del árbol sintáctico.
  - OperadorConcreto1 y OperadorConcreto2 (OperadorY, OperadorO) son implementaciones del OperadorAbstracto que describen completamente la semántica del operador y por lo tanto son capaces de evaluarlo.
  - ElementoTerminal es una clase abstracta que describe cualquier nodo correspondiente a un elemento terminal.

El patrón Interpreter, VIII

- ElementoTerminalConcreto1 y ElementoTerminalConcreto2 (PalabraClave) son clases concretas que se corresponden con un elemento terminal, capaces de evaluar este elemento.

3. Colaboraciones

- El cliente construye una expresión bajo la forma de un árbol sintáctico cuyos nodos son instancias de las subclases de Expresion. A continuación, solicita a la instancia situada en la cima del árbol que proceda a realizar la evaluación:
- Si esta instancia es un elemento terminal, la evaluación es directa.
- Si esta instancia es un operador, tiene que proceder con la evaluación de los operandos en primer lugar. Esta evaluación se realiza de forma recursiva, considerando a cada operando como la cima de una expresión.

Dominios de aplicación

- El patrón se utiliza para interpretar expresiones representadas bajo la forma de árboles sintácticos. Se aplica principalmente en los siguientes casos:
  - La gramática de las expresiones es simple.
  - La evaluación no necesita ser rápida.
- Si la gramática es compleja, es preferible utilizar analizadores sintácticos especializados. Si la evaluación debe realizarse rápidamente, puede resultar necesario el uso de un compilador.

El patrón Interpreter, IX

Ejemplo en PHP

- A continuación se muestra el código completo de un ejemplo en PHP que no sólo permite evaluar un árbol sintáctico sino que también lo construye.
- La construcción del árbol sintáctico, llamados análisis sintáctico, también está repartida en las clases, es decir las de la figura del acetato 27 bajo la forma de métodos de clase (métodos precedidos por la palabra reservada *static* en PHP).
- El código fuente de la clase Expresion aparece a continuación. La parte relativa a la evaluación se limita a la declaración de la firma del método *evalua*.
- Los métodos siguientePieza, analiza y parsea están dedicados al análisis sintáctico. El método analiza se utiliza para parsear una expresión entera mientras que parsea está dedicado al análisis ya sea de una palabra-clave o bien de una expresión escrita entre paréntesis.

El patrón Iterator, I

Descripción

- El patrón Iterator proporciona un acceso secuencial a una colección de objetos a los clientes sin que estos tengan que preocuparse de la implementación de esta colección.

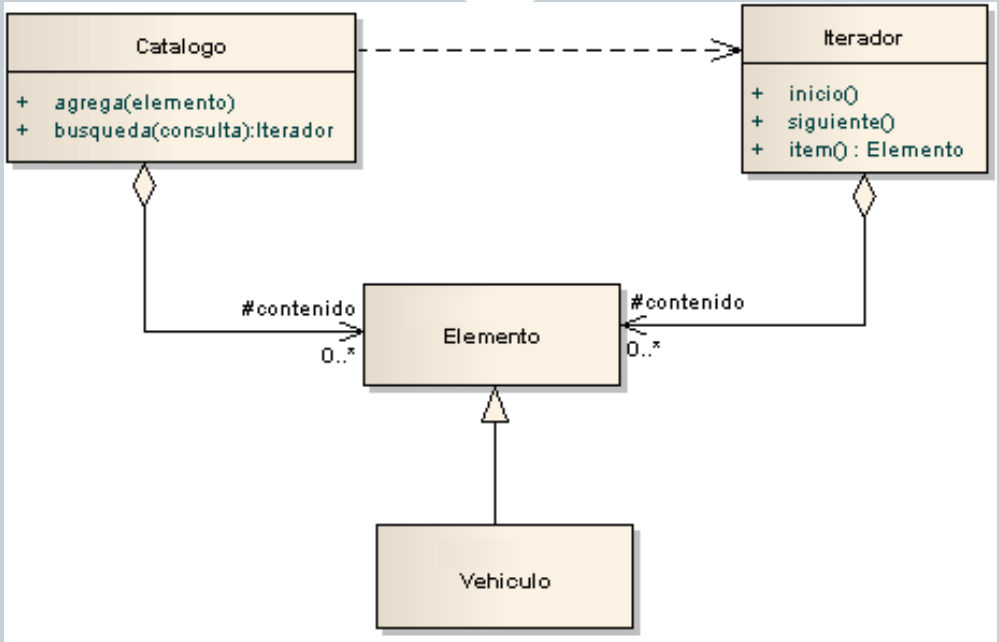
Ejemplo

- Queremos proporcionar un acceso secuencial a los vehículos que componen el catálogo. Para ello, podemos implementar en la clase del catálogo los siguientes métodos:
  - inicio: inicializa el recorrido por el catálogo.
  - item: reenvía el vehículo en curso.
  - siguiente: pasa al vehículo siguiente.
- Esta técnica presenta dos inconvenientes:
  - Hace aumentar de manera inútil la clase catálogo.
  - Sólo permite recorrer el catálogo una vez, lo cual puede ser insuficiente (en especial en el caso de aplicaciones multitarea).

El patrón Iterator, II

- El patrón Iterator proporciona una solución a este problema.
- La idea consiste en crear una clase Iterator donde cada instancia pueda gestionar un recorrido en una colección. Las instancias de esta clase Iterator las crea la clase colección, que se encarga de inicializarlas.
- El objetivo del patrón Iterator es proporcionar una solución que pueda ser configurada según el tipo de elementos que componen la colección. Presentamos por lo tanto dos clases abstractas genéricas:
  - Iterador es una clase abstracta genérica que incluye los métodos inicio, ítem y siguiente.
  - Catalogo es a su vez una clase abstracta genérica que incluye los métodos que crean, inicializan y devuelven una instancia del Iterador.
- A continuación es posible crear las subclases concretas de estas dos clases abstractas genéricas, subclases que relacionan en particular los parámetros de genericidad con los tipos utilizados en la aplicación.
- La figura del acetato 34 muestra el uso del patrón Iterator para recorrer los vehículos del catálogo que responden a una consulta.

El patrón Iterator, III



- Este diagrama de clases utiliza parámetros genéricos que suponen ciertas restricciones (TElemento es un subtipo de Elemento y TIterador es un subtipo de Iteador<TElemento>). Las dos clases Catalogo e Iterador poseen una asociación con un conjunto de elementos, siendo el conjunto de elementos referenciados por Iterador un subconjunto de los referenciados por Catalogo.

El patrón Iterator, IV

- Las subclases CatalogoVehiculo e IteradorVehiculo heredan mediante una relación que fija los tipos de parámetros de genericidad de su superclases respectivas.

Estructura

1. Diagrama de clases

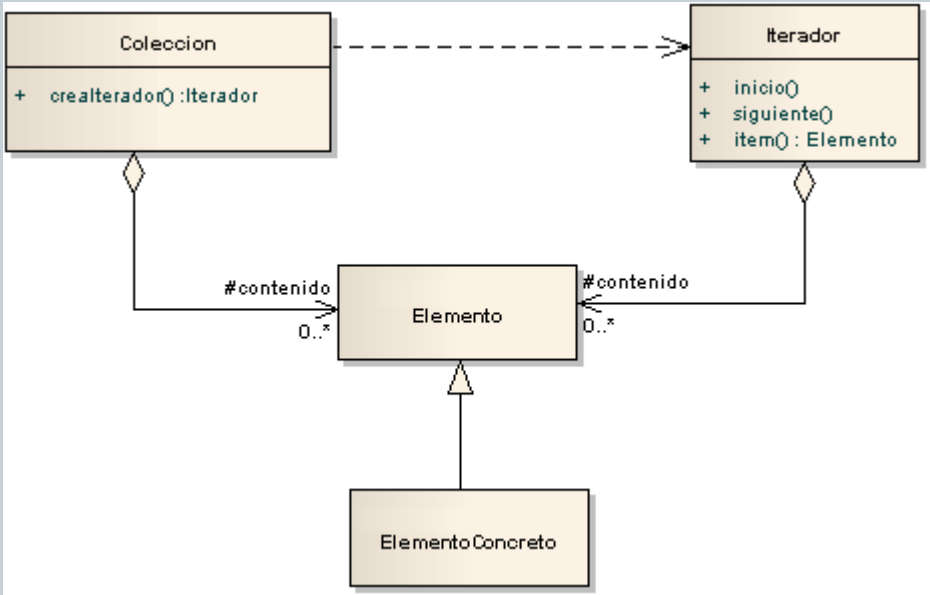
- La figura del acetato 36 detalla la estructura genérica del patrón, que es muy parecida al diagrama de clases de la figura del acetato 34.

2. Participantes

- Los participantes del patrón son los siguientes:
  - Iterador es la clase abstracta que implementa la asociación del iterador con los elementos de la colección así como los métodos. Es genérica y está parametrizada mediante el tipo TElemento.
  - IteradorConcreto (IteradorVehiculo) es una subclase concreta de Iterador que relaciona TElemento con ElementoConcreto.
  - Coleccion (Catalogo) es la clase abstracta que implementa la asociación de la colección con los elementos y el método creaIterador.

El patrón Iterator, V

- Elemento es la clase abstracta de los elementos de la colección.
- ElementoConcreto (Vehiculo) es una subclase concreta de Elemento utilizada por IteradorConcreto y ColeccionConcreta.



3. Colaboraciones

- El iterador guarda en memoria el objeto en curso en la colección. Es capaz de calcular el objeto siguiente del recorrido.

El patrón Iterator, VI

Dominios de aplicación

- El patrón se utiliza en los siguientes casos:
  - Es necesario realizar un recorrido de acceso al contenido de una colección sin acceder a la representación interna de esta colección.
  - Debe ser posible gestionar varios recorridos de forma simultánea.

Ejemplo en PHP

- Presentamos a continuación un ejemplo en PHP del recorrido del catálogo de vehículos con ayuda de un iterador.
- El código fuente de la clase abstracta Elemento se muestra a continuación. Los elementos poseen una descripción.
- El método palabraClaveValida verifica si aparece cierta palabra clave en la descripción.

El patrón Mediator, I

Descripción

- El patrón Mediator tiene como objetivo construir un objeto cuya vocación es la gestión y el control de las interacciones en un conjunto de objetos sin que sus elementos deban conocerse mutuamente.

Ejemplo

- El diseño orientado a objetos favorece la distribución del comportamiento entre los objetos del sistema. No obstante, llevada al extremo, esta distribución puede llegar a tener un gran número de enlaces que obligan casi a cada objeto a conocer a todos los demás objetos del sistema.
- Un diseño con tal cantidad de enlaces puede volverse de mala calidad. En efecto, la modularidad y las posibilidades de reutilización de los objetos se reducen. Cada objeto no puede trabajar sin los demás y el sistema se vuelve monolítico, perdiendo toda su modularidad.
- Además para adaptar y modificar el comportamiento de una pequeña parte del sistema, resulta necesario definir numerosas subclases.

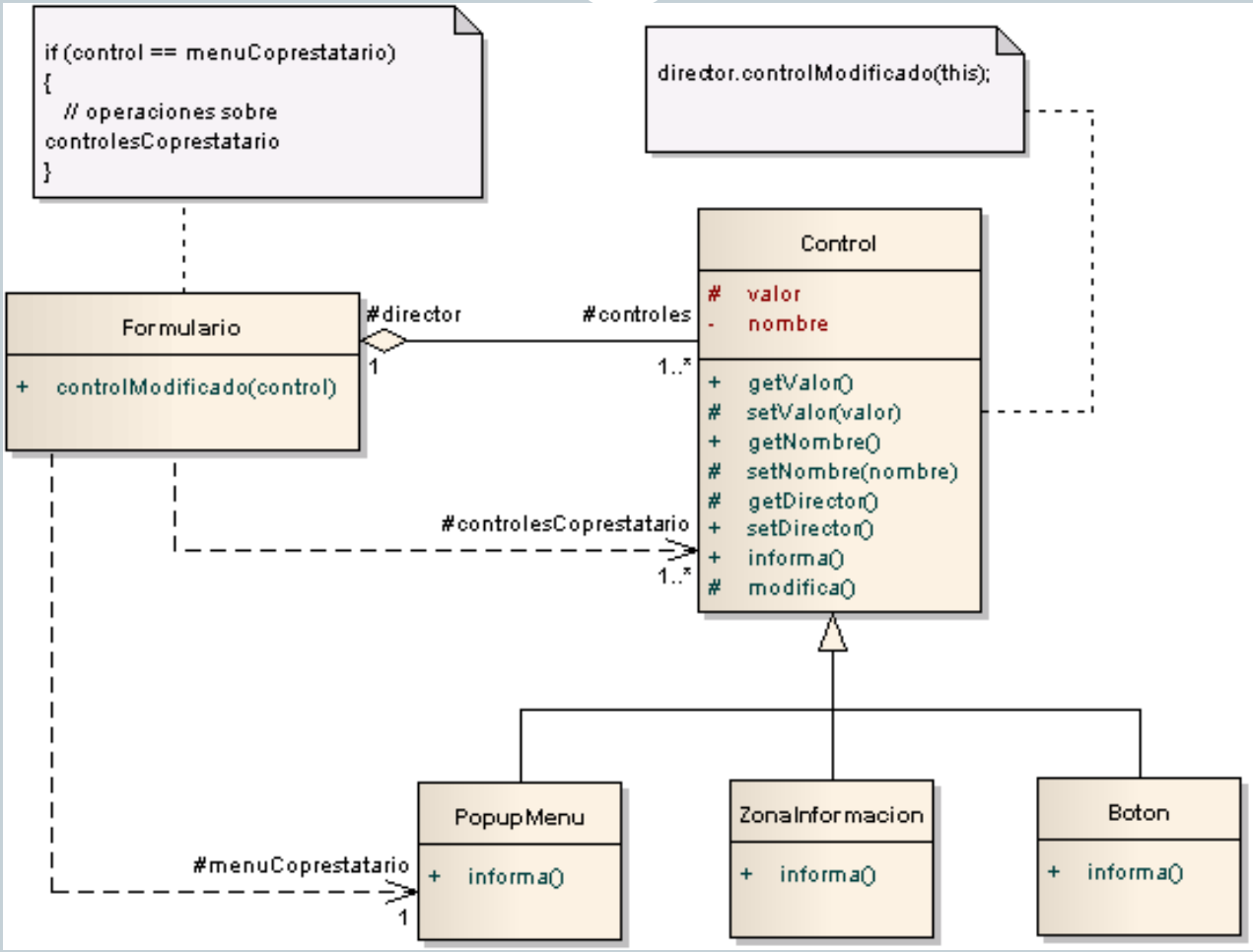
El patrón Mediator, II

- Las interfaces de usuario dinámicas son un buen ejemplo de tal sistema. Una modificación en el valor de un control gráfico puede conducir a modificar el aspecto de otros controles gráficos como, por ejemplo:
  - Volverse visible u oculto.
  - Modificar el número de posibles valores (para un menú).
  - Cambiar el formato de los valores que es necesario informar.
- La primera posibilidad consiste en enlazar cada control cuyo aspecto cambia en función de su valor. Esta posibilidad presenta los inconvenientes citados anteriormente.
- La otra posibilidad consiste en implementa el patrón Mediator.
- Este consiste en construir un objeto central encargado de la coordinación de los controles gráficos.
- Cuando se modifica el valor de un control, previene al objeto mediador que se encarga de invocar a los métodos correspondientes de los demás controles gráficos para que puedan realizar las modificaciones necesarias.

El patrón Mediator, III

- En nuestro sistema de venta online de vehículos, es posible solicitar un préstamo para adquirir un vehículo rellenando un formulario online.
- Es posible solicitar el préstamo solo o con otra persona.
- Esta elección se realiza con la ayuda de un menú. Si la elección resulta solicitar el préstamo con otro prestatario, existe toda una serie de controles gráficos relativos a los datos del coprestatario que deben mostrarse y rellenarse.
- La figura del acetato 41 ilustra el diagrama de clases correspondiente.
- Este diagrama incluye las siguientes clases:
  - Control es una clase abstracta que incluye los elementos comunes a todos los controles gráficos.
  - PopupMenu, ZonaInformacion y Boton son las subclases concretas de Control que implementan el método *informa*.
  - Formulario es la clase que realiza la función de mediador. Recibe las notificaciones de cambio de los controles invocando al método controlModificado.

El patrón Mediator, IV



El patrón Mediator, V

- Cada vez que el valor de un control gráfico se modifica, se invoca el método modificado del control.
- Este método heredado de la clase abstracta Control invoca a su vez al método controlModificado de Formulario (el mediador).
- Este invoca, a su vez, a los métodos de los controles del formulario para realizar las acciones necesarias.
- La figura del acetato 43 ilustra este comportamiento de forma parcial sobre el ejemplo. Cuando el valor del control menuCoprestatario cambia, se informan los datos relativos al nombre y apellidos del coprestatario.

Estructura

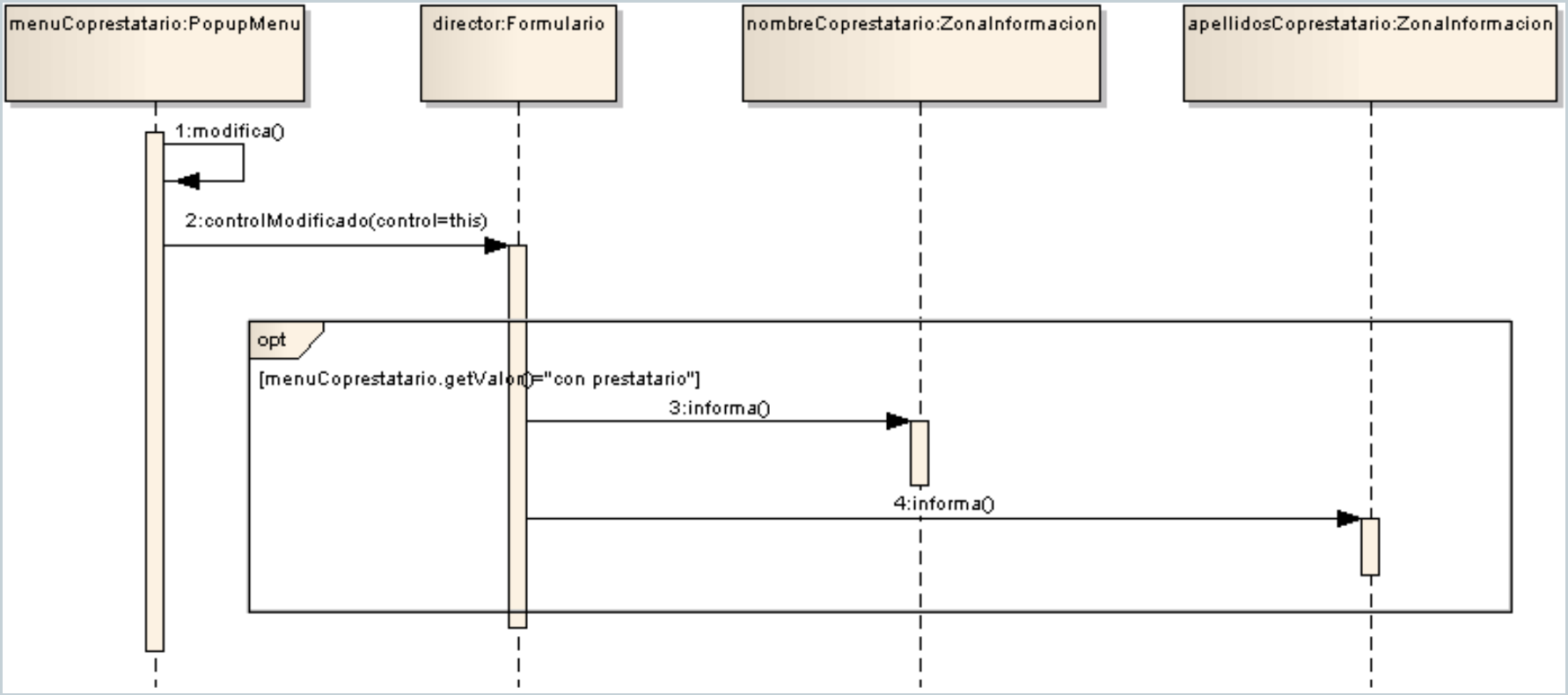
1. Diagrama de clases

- La figura del acetato 44 detalla la estructura genérica del patrón.

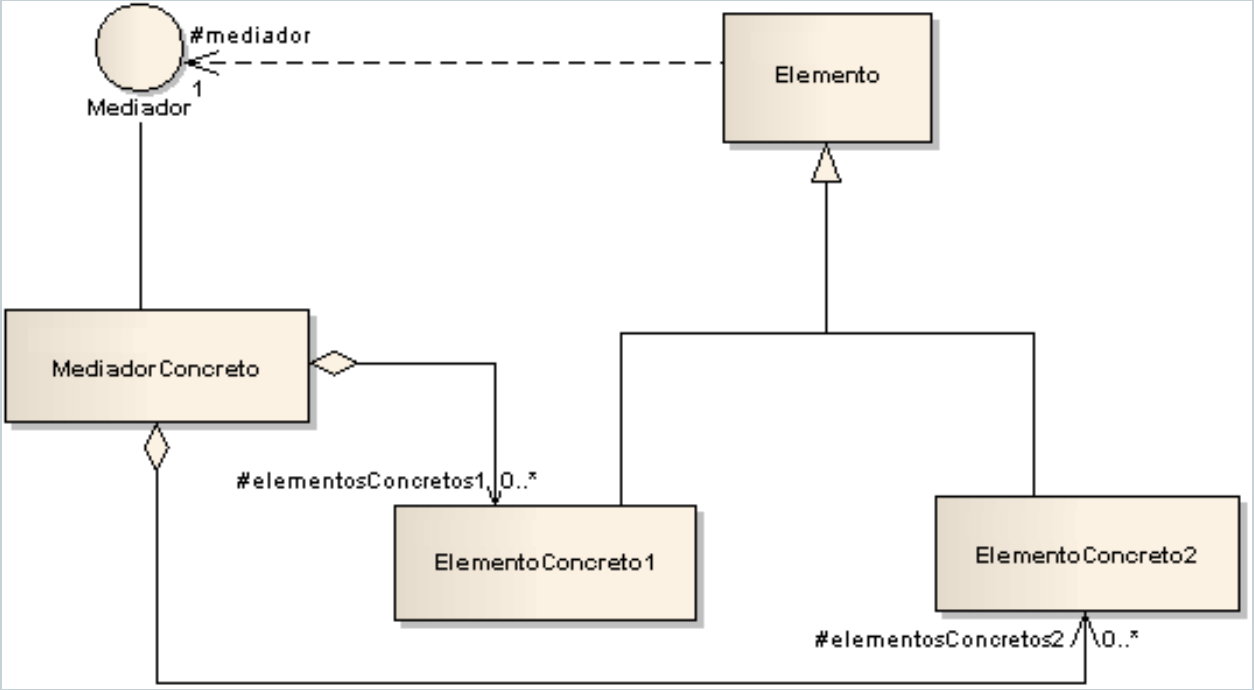
2. Participantes

- Los participantes del patrón son los siguientes:

El patrón Mediator, VI



El patrón Mediator, VII



- Mediador define la interfaz del mediador para los objetos Elemento.
- MediadorConcreto (Formulario) implementa la coordinación entre los elementos y gestiona las asociaciones con los elementos.
- Elemento (Control) es la clase abstracta de los elementos que incluyen sus atributos,

El patrón Mediator, VIII

- asociaciones y métodos comunes.
- ElementoConcreto1 y ElementoConcreto2 (PopupMenu, ZonaInformacion y Boton) son las clases concretas de los elementos que se comunican con el mediador en lugar de con los demás elementos.

3. Colaboraciones

- Los elementos envían y reciben mensajes del mediador. El mediador implementa la colaboración y la coordinación entre los elementos.

Dominios de aplicación

- El patrón se utiliza en los siguientes casos:
  - Un sistema está formado por un conjunto de objetos basado en una comunicación compleja que conduce a asociar numerosos objetos entre ellos.
  - Los objetos de un sistema son difíciles de reutilizar puesto que poseen numerosas asociaciones con otros objetos.
  - La modularidad de un sistema es mediocre, obligando en los casos en los que se debe adaptar una parte del sistema a escribir numerosas subclases.

El patrón Mediator, IX

Ejemplo en PHP

- A continuación proponemos simular la información de un formulario con la ayuda de entradas/salidas clásicas basada en una introducción secuencial de datos en cada control hasta que se valida el botón “OK” (mediante teclado).
- Un menú permite elegir si el préstamo se realiza o no con un coprestatario.