Patrones para el modelado de Software Orientado a Objetos

Manual de Referencia

Preparado por Ing. Judith Meles

Tabla de Contenidos

Patrones de Software

Uno de los pilares de la filosofía de objetos es la posibilidad de REUSAR, es decir hacer una sola vez las cosas y usarlas cada vez que haga falta.
La pregunta es: ¿Solo podemos reusar código? La respuesta es NO.
Podemos reusar las experiencias, soluciones de análisis y diseño que ya hemos aplicado anteriormente.
Los patrones *realizan una descripción de un problema que ocurre y de cómo solucionarlo*. Son estereotipos que utilizamos una y otra vez para solucionar problemas parecidos. Se pueden utilizar millones de veces y además perfeccionarlos.
El objetivo o meta que se persigue es tener: *Catálogos de Patrones Estandarizados*.

"**Un patrón no busca lo particular sino que destaca la esencia"**.

¿Cómo se expresan los patrones en Objetos?
Son soluciones expresadas en términos de objetos e interfaces se expresan en base a la estructura y/o comunicación de objetos y clases para resolver los problemas del contexto.

Un buen patrón:
- ⇒ Plantea una solución al problema.
- ⇒ Provee conceptos (captura soluciones)
- ⇒ Permite derivar soluciones desde primeros principios.
- ⇒ Describe relaciones.
- ⇒ Debe tener en cuenta al componente humano.

En la orientación a objetos el patrón enfoca un solo aspecto de un problema, y debe enfocarlo identificando:
- ⇒ Clases participantes.
- ⇒ Instancias
- ⇒ Roles
- ⇒ Colaboraciones
- ⇒ Distribución de Responsabilidades.

El patrón, también debe especificar:
- ⇒ Cuando aplicarlo
- ⇒ Si debe o no ser aplicado. Es decir debe especificar las restricciones de uso para dicho patrón.
- ⇒ Consecuencias de la aplicación.
- ⇒ Que debe tenerse en cuenta cuando se lo está aplicando.
- ⇒ Debe formularse estableciendo la relación entre un contexto, un sistema y una configuración que permita resolver el problema.

Componentes de un Patrón:

1. Nombre
2. Propósito
3. Sinónimo
4. Colaboraciones
5. Contexto
6. Explicación
7. Fuerzas
8. Motivación
9. Ejemplos
10. Patrones Relacionados
11. Aplicabilidad
12. Estructura
13. Participantes
14. Consecuencias

Patrones para la Construcción de Modelos de Objeto del Dominio

Un patrón de modelo de objetos es una agrupación de objetos con responsabilidades estereotipadas y escenarios de interacción.

Esta sección presenta Patrones organizados en varias familias de patrones. Primero se encontrará:

- Patrón Fundamental

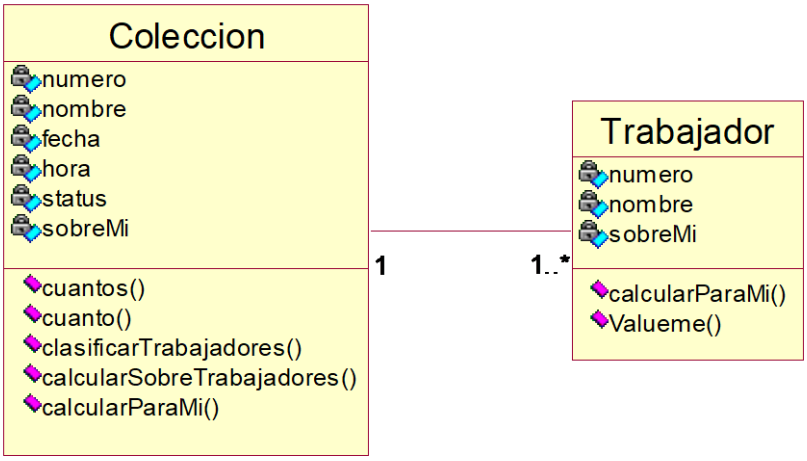
Y luego:

- Patrones Transaccionales.

El patrón fundamental

Este Patrón es el fundamental del modelado de objetos: es la plantilla que todos los patrones siguen:

#1. Patrón: "Colección-Trabajador"	el patrón fundamental



- Colección-trabajador es el patrón fundamental del modelo de objetos.

- Todos los otros patrones del modelo de objetos son variaciones de este tema.

- Interacciones típicas de objetos:

  cuántos ---> calcParaMí calcSobreTrabajadores ---> calcParaMí
  cuánto ---> calcParaMí clasificarTrabajadores ---> valúeMe

- Otras notas:

  "sobreMí" lo ayuda a uno a pensar qué otros atributos podrían ser necesarios.
  "calcularParaMí" lo ayuda a uno a pensar qué cálculos específicos podrían ser necesarios.
  "clasificarMe" lo ayuda a uno a pensar qué servicios de ordenamiento o comparación podrían existir.
  "valúeMe" lo ayuda a uno a pensar qué servicios de auto-evaluación podrían ser necesarios.

Patrones Transaccionales

Los patrones transaccionales son aquellos patrones que tienen un jugador de transacción_ o tienen jugadores que comúnmente juegan con un jugador de transacción.
Los patrones de transacción son:

- actor-participante.
- participante-transacción
- lugar-transacción
- ítem especifico-transacción
- transacción- detalle de transacción
- transacción- transacción subsiguiente
- detalle de transacción- detalle de transacción subsiguiente
- ítem- detalle de transacción
- ítem específico-detalle de transacción
- ítem- ítem específico
- asociación-otra asociación
- ítem específico-jerarquía de ítem

Se muestra un resumen de los patrones de transacciones, ilustrando como pueden interconectarse unos con otros:

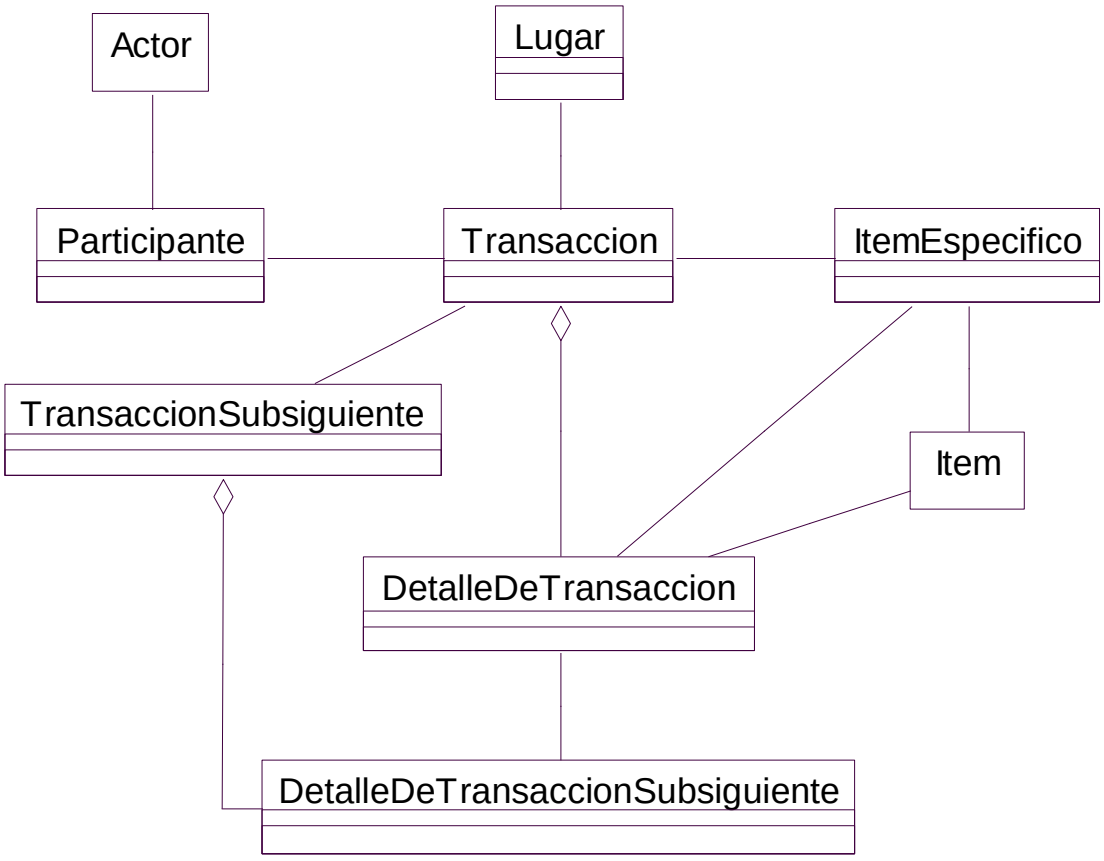
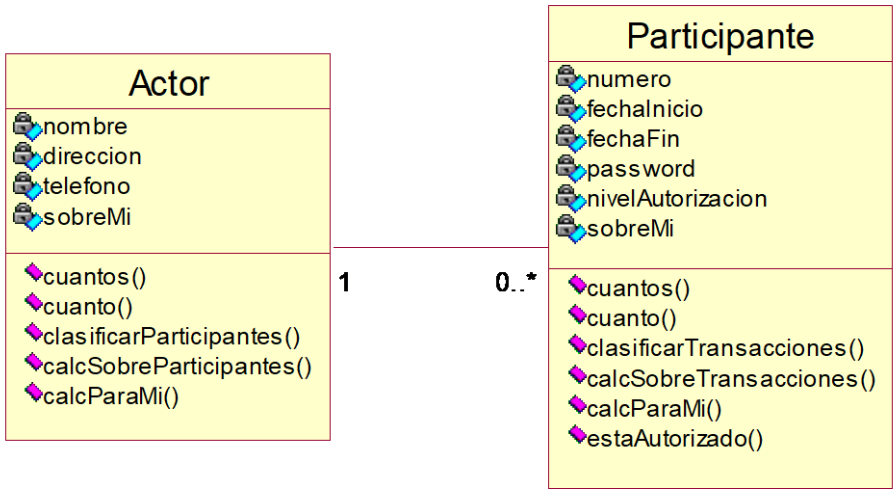


Figura 7.1: Una vista de los patrones de transacciones.

#2. Patrón: "Actor-Participante" patrones de transacción

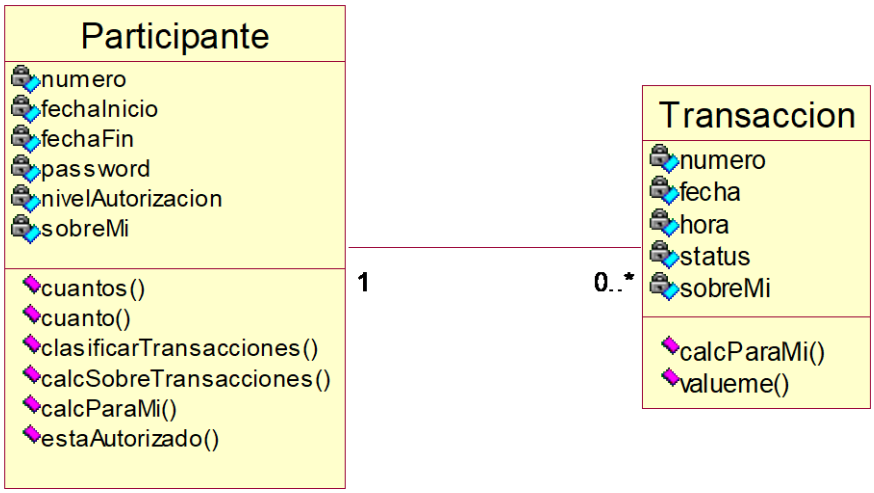


- Interacciones típicas de objetos:
  cuántos ---> calcParaMí clasificarParticipantes ---> valúeMe
  cuánto ---> calcParaMí tomarNombre <--- tomarNombre
  calcSobreParticipantes---->calcParaMí tomarDirección <--- tomarDirección

- Ejemplos:
  Actor: persona, organización (agencia, compañía, corporación, fundación)
  Participante: agente, solicitante, comprador, cajero, oficinista, cliente, distribuidor, delegado,
  donante, miembro, oficial, profesional, remitente, estudiante, subscriptor, supervisor, profesor, trabajador.
  Ejemplos adicionales: cualquier cosa usada para diferentes misiones, como avión-misión civil,
  avión-misión militar.

- Combinaciones:
  participante-transacción, actor-participante, otra vez (por ejemplo, cliente-cliente de oro,
  para un cliente que quizás participe como un cliente de oro, al menos mientras él califica).

#3. Patrón: "Participante-Transacción" patrones de transacción

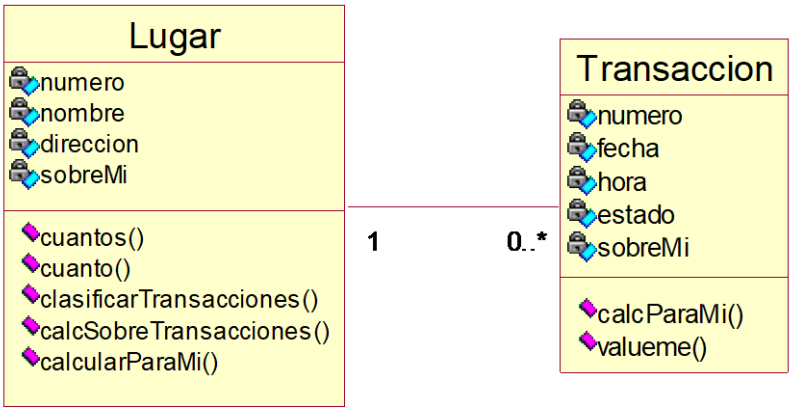


- Interacciones típicas de objetos:
  cuántos ---> calcParaMí
  cuánto ---> calcParaMí
  calcSobreTransacciones ---> calcParaMí
  clasificarTransacciones ---> valúeMe

- Ejemplos:
  Participante: agente, solicitante, comprador, cajero, oficinista, cliente, distribuidor, delegado,
  donante, miembro, oficial, profesional, remitente, estudiante, subscriptor, supervisor, profesor, trabajador.
  Transacción: contrato, asignación, acuerdo, autorización, entrega, depósito, incidente, investigación, orden, pago, reporte de problema, compra, reembolso, registración,
  renta, reservación, venta, traslado, envío, subscripción, carga horaria, título, retiro.

- Combinaciones:
  actor-participante, participante-transacción, lugar-transacción, transacción-transacción subsiguiente, transacción-detalle de transacción, ítem específico-transacción.

#4. Patrón: "Lugar-Transacción"	patrones de transacción



- Interacciones típicas de objetos:
  cuántos ---> calcParaMí
  cuánto ---> calcParaMí
  calcSobreTransacciones ---> calcParaMí
  clasificarTransacciones ---> valúeMe

- Ejemplos:
  Lugar: aeropuerto, línea de ensamblaje, banco, clínica, garage, entidad geográfica, hangar,
  hospital, planta, región, mostrador, estación, almacén, zona.
  Transacción: contrato, asignación, acuerdo, autorización, entrega, depósito, incidente, investigación, orden, pago, reporte de problema, compra, reembolso, registración,
  renta, reservación, venta, traslado, envío, subscripción, carga horaria, título, retiro.

- Combinaciones:
  participante-transacción, ítem específico - transacción, transacción - detalle de transacción, transacción-transacción subsiguiente. También, usando "lugar" como un contenedor: contenedor-contenido, contenedor-detalle de contenedor.

#5. Patrón: "Item Específico-Transacción"	patrones de transacción

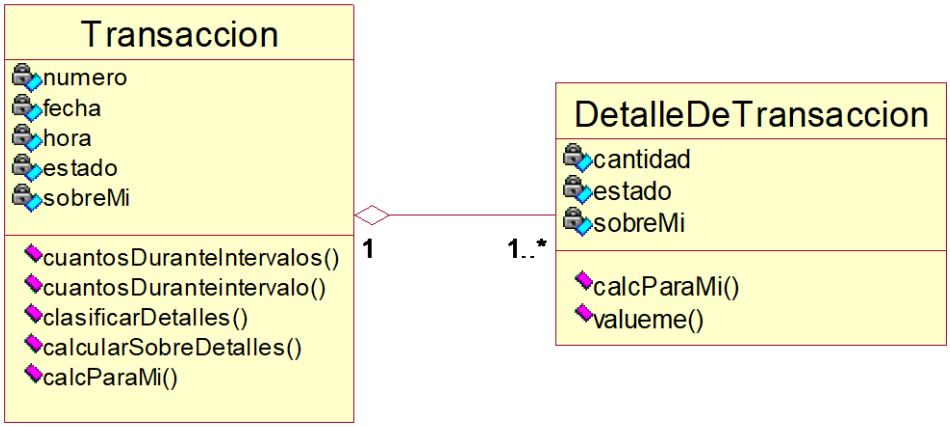
- Interacciones típicas de objetos:
  cuántos ---> calcParaMí
  cuánto ---> calcParaMí
  calcSobreTransacciones ---> calcParaMí
  clasificarTransacciones ---> valúeMe

- Ejemplos:
  Item Específico: contenedor específico, barco específico, registro específico, vehículo específico.
  Transacción: contrato, asignación, acuerdo, autorización, entrega, depósito, incidente, investigación, orden, pago, reporte de problema, compra, reembolso, registración, renta, reservación, venta, traslado, envío, subscripción, carga horaria, título, retiro.

- Combinaciones:

participante-transacción, lugar-transacción, transacción - detalle de transacción, transacción-transacción subsiguiente. También, usando "ítem especifico" como un contenedor: contenedor-contenido, contenedor-detalle de contenedor.

#6. Patrón: "Transacción-Detalle de Transacción"	patrones de transacción

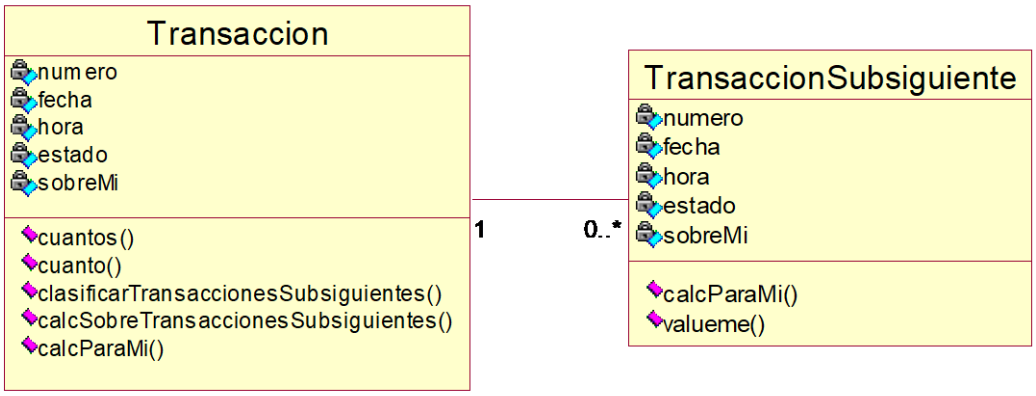


- Interacciones típicas de objetos:
  cuántosDuranteIntervalo ---> calcParaMí calcSobreDetalles ---> calcParaMí
  cuántoDuranteIntervalo ---> calcParaMí clasificarDetalles ---> valúeMe

- Ejemplos:
  Transacción: contrato, asignación, acuerdo, autorización, entrega, depósito, incidente, investigación, orden, pago, reporte de problema, compra, reembolso, registración, renta, reservación, venta, traslado, envío, subscripción, carga horaria, título, retiro.
  Transacción-detalle de transacción: depósito-detalle de depósito, orden-detalle de orden,
  pago-detalle de pago, venta-detalle de venta, retiro-detalle de retiro.

- Combinaciones:
  participante-transacción, lugar-transacción, ítem específico-transacción,
  transacción-transacción subsiguiente, ítem específico-detalleTransacción,
  ítem-ítem específico.

#7. Patrón: "Transacción-Transacción Subsiguiente"	patrones de transacción



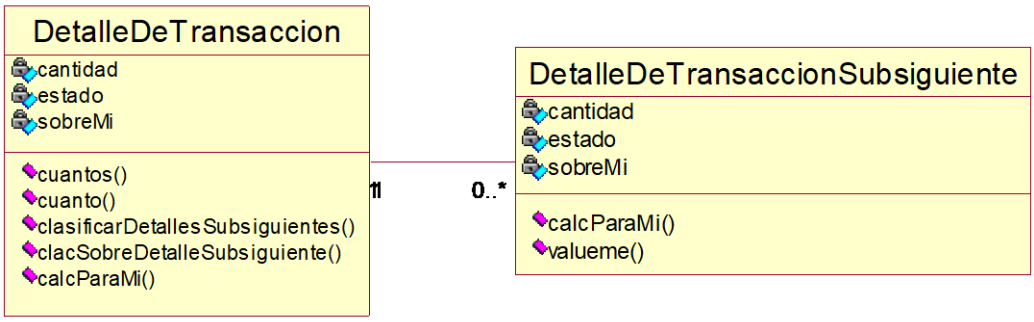
- Interacciones típicas de objetos:
  cuántos ---> calcParaMí calcSobreTransaccionesSubsiguientes ---> calcParaMí
  cuánto ---> calcParaMí clasificarTransaccionesSubsiguientes ---> valúeMe

- Ejemplos:
  Transacción: contrato, asignación, acuerdo, autorización, entrega, depósito, incidente, investigación, orden, pago, reporte de problema, compra, reembolso, registración, renta, reservación, venta, traslado, envío, subscripción, carga horaria, título, retiro.
  Transacción-transacción subsiguiente: resultado intermedio-resultado final, orden-envío, compra-pago, reservación-venta, multa-pago.

- Combinaciones:
  participante-transacción, lugar-transacción, ítem específico - transacción, transacción - detalle de transacción, transacción subsiguiente-detalle de transacción subsiguiente.

- Notas:
  Diseñe las transacciones en secuencia temporal (en el orden en que ocurren usualmente).
  Si la transacción subsiguiente y sus objetos detalle corresponden 1 a 1 con la transacción y sus objetos detalle, combínelas.

#8. Patrón: "Detalle de Transacción-Detalle de Transacción Subsiguiente"	patrones de transacción

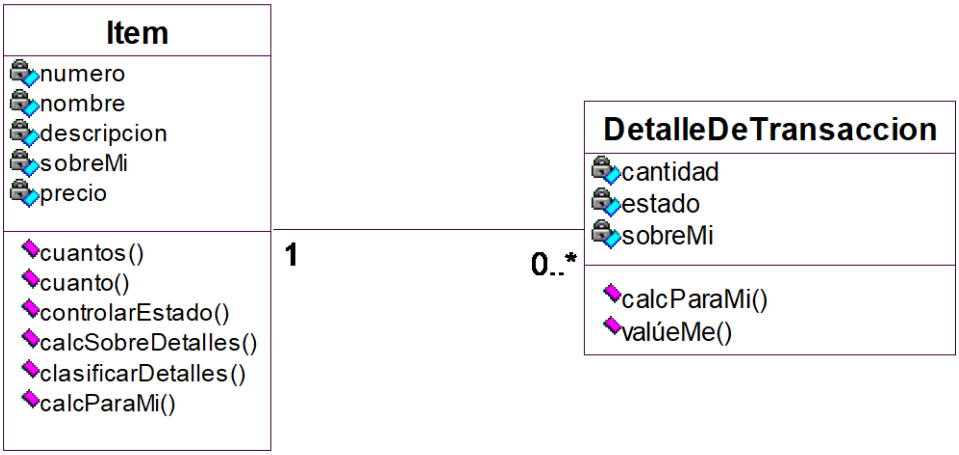


- Interacciones típicas de objetos:
  cuántos ---> calcParaMí calcSobreDetallesSubsiguientes ---> calcParaMí
  cuánto ---> calcParaMí clasificarTransacciones ---> valúeMe

- Ejemplos:
  Detalle de Transacciones: contrato, asignación, acuerdo, autorización, entrega, depósito, incidente, investigación, orden, pago, reporte de problema, compra, reembolso, registración, renta, reservación, venta, traslado, envío, subscripción, carga horaria, título, retiro.
  Detalle Transacción-DetalleTransacción Subsiguiente: detalle de orden, detalle de envío, detalle de reserva, detalle de renta, detalle de entrega.

- Combinaciones:
  Transacción-detalle de transacción, transacción subsiguiente- detalle de transacción subsiguiente, ítem- item específico - ítem específico- detalle de transacción.

#9. Patrón: "Item -Detalle de Transacción"	patrones de transacción

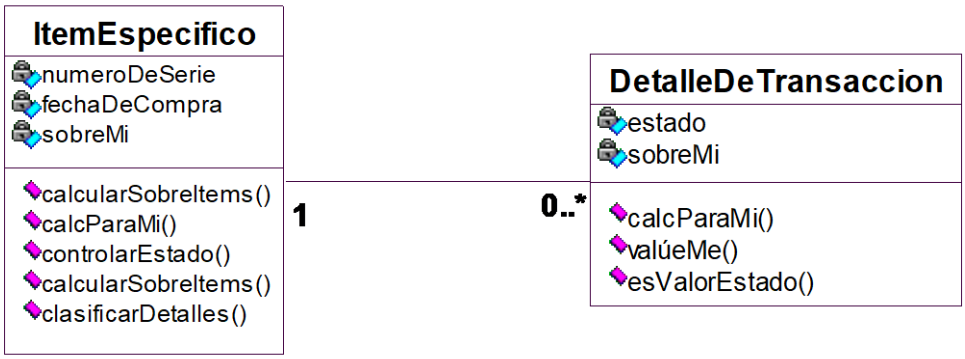


- Interacciones típicas de objetos:
  cuántos ---> obtenerCantidad
  calcularSobreDetallesArtículo ---> calcParaMí clasificarDetalles ---> valúeMe

- Ejemplos:
  (con detalles de transacción): ítem-detalle de orden, ítem-detalle de pago, ítem-detalle de venta.
  (con detalles de contenedor): ítem-detalle de repositorio, ítem-detalle de depósito.

- Combinaciones:
  transacción-detalle de transacción, transacción subsiguiente-detalle de transacción subsiguiente,
  detalle de transacción-detalle de transacción subsiguiente.

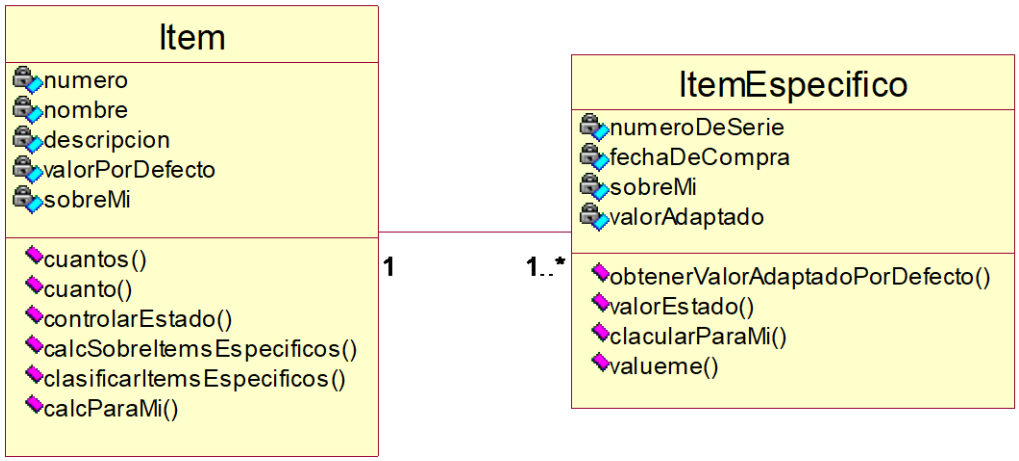
#10. Patrón: "Item Específico-Detalle de Transacción"	patrones de transacción



- Interacciones típicas de objetos:
  Cuántos ---> calcParaMí clasificarDetalle ---> valueMe
  CalcSobreItemDeLinea ---> calcParaMí controlarStatus ---> esValorEstado
- Ejemplos:
  aeroplano específico- detalle de transacción; vehículo específico – detalle de transacción, video – detalle de alquiler

- Combinaciones:
  transacción - detalle de transacción, transacción subsiguiente- detalle de transacción subsiguiente, detalle de transacción- detalle de transacción subsiguiente.

#11. Patrón: "Item -Item Específico”	patrones de transacción

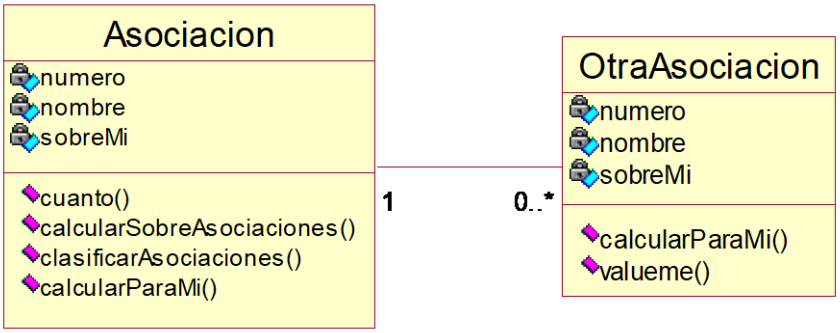


- Interacciones típicas de objetos:
  Cuántos ---> calcParaMí clasificarItemsEspecíficos ---> valueMe
  controlarStatus ---> esValorStatus obtenerValorPor Default -
  →obtenerClienteporDefault
  calcSobreItemsEspecíficos ---> calcParaMi

- Ejemplos:
  Aeroplano-aeroplano-específico, descripción de préstamos-préstamo específico; descripción de trabajo- trabajo específico; descripción de video – video casette; categoría de precio-item- ítem específico (Patrón aplicado dos veces); categoría de impuesto-ítem- ítem específico (Patrón aplicado dos veces).

- Combinaciones:
  Item específico-transacción; ítem- detalle de transacción.

#12. Patrón: "Asociación-Otra Asociación "	patrones de transacción



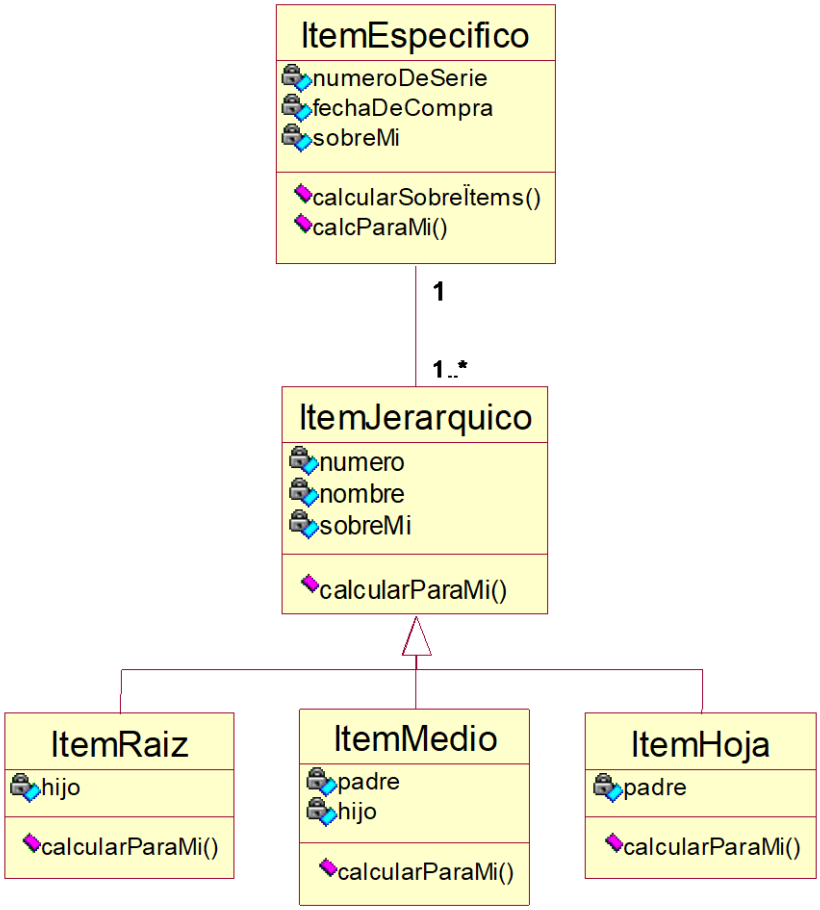
- Interacciones típicas de objetos:
  cuántos ---> calcParaMí calcSobreOtrasAsociaciones ---> calcParaMí
  clasificarOtrasAsociaciones ---> valúeMe

- Ejemplos:
  Edificio-sensor; conductor-vehículo; alerón-giro, aeroplano-pista de aterrizaje; muelle de carga-orden; orden-transporte; camión-muelle de carga

- Combinaciones:
  Cualquier otro Patrón

- Nota:
  Las asociaciones son objetos que se conocen entre sí, sin ser necesaria información a cerca de esa asociación o historia a cerca de ella.
  En algunas instancias de Patrón, un “otra asociación” puede necesitar conocer algún número de asociaciones.

#13. Patrón: "Item Específico – Item Jerárquico”	patrones de transacción



- Interacciones típicas de objetos:
  Cuántos ---> calcParaMí calcSobreItems ---> calcParaMi

- Ejemplos:
  Cuenta-jerarquía de descripción de cuentas; mercadería- jerarquía de descripción de mercadería, organización -

- Combinaciones:
  Item específico-ítem.

Patrones de Agregación

Los Patrones de agregación son:

- contenedor-contenido.
- contenedor-detalle de contenedor.
- grupo-miembro
- todo-parte
- parte de compuesto-parte
- paquete-componente de paquete

Estos Patrones se interconectan con otros Patrones, algunas veces con la ayuda de los Patrones transaccionales “asociación- otra asociación”.

Aquí está una descripción de los Patrones de agregación:

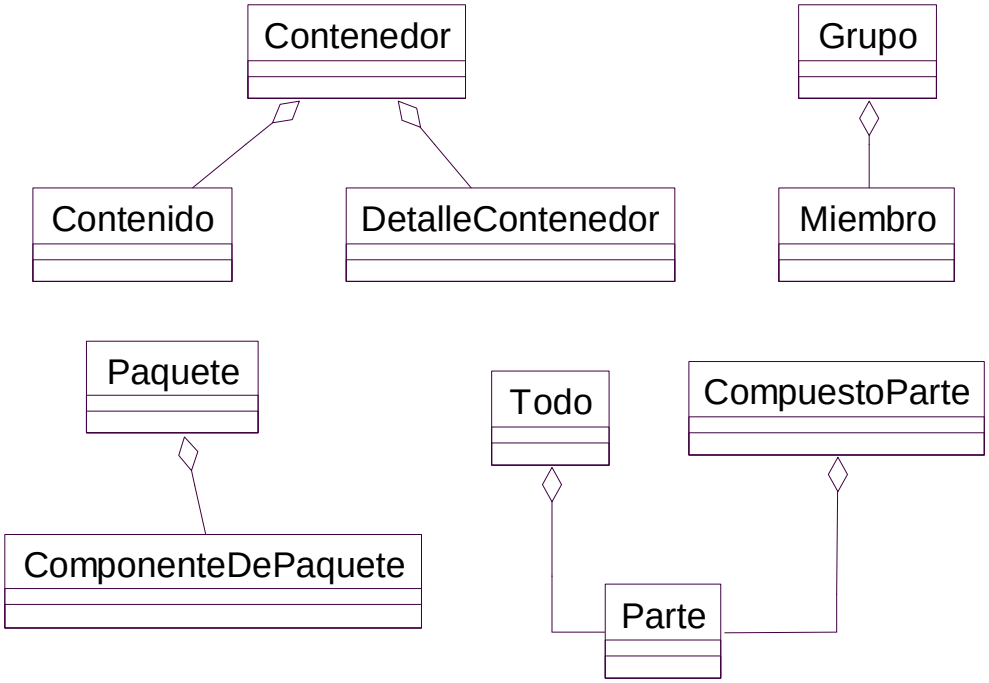
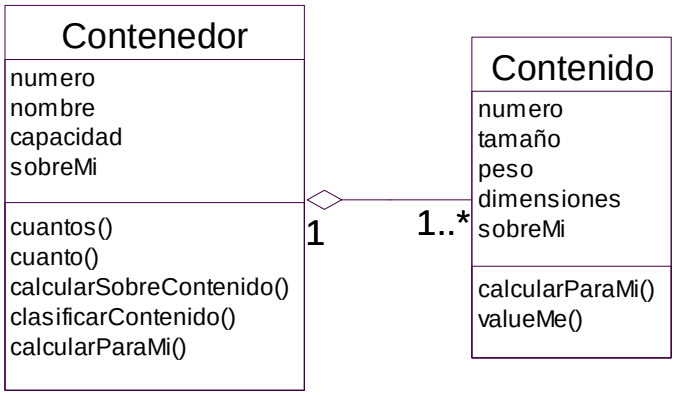


Figura 7-2: Un resumen de los Patrones de agregación

#14. Patrón: "Contenedor-Contenido"	patrones de agregación

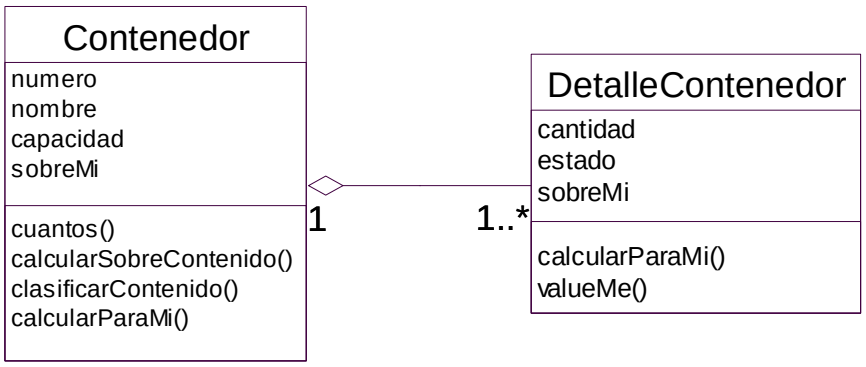


- Interacciones típicas de objetos:
  cuántos ---> calcParaMí calcSobreContenido ---> calcParaMí
  cuánto ---> calcParaMí clasificarContenido ---> valúeMe

- Ejemplos:
  Contenedores: aeropuerto, pasillo, banco, depósito, edificio, gabinete, carpeta, garaje, hangar, hospital, armario, sala, caja fuerte, almacén.
  Contenedor-contenido: avión-cargamento, avión-pasajero, edificio-habitación, almacén-artículo, catálogo-artículo de catálogo, depósito-dársena de carga.

- Combinaciones:
  contenedor-contenido (otra vez), contenedor-detalle de contenedor, grupo-miembro, ensamblado-parte. A la vez, cuando "contenedor" o "contenido" es un participante, lugar, o ítem específico: actor-participante, participante-transacción, lugar-transacción, ítem específico-transacción, detalle de ítem específico-detalle de transacción, ítem-ítem específico.

#15. Patrón: "Contenedor-Detalle de Contenedor"	patrones de agregación

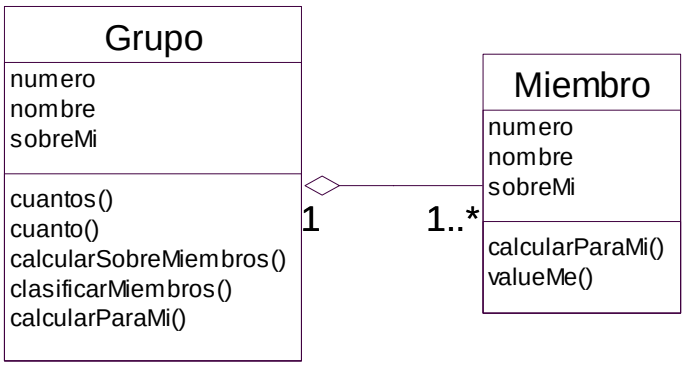


- Interacciones típicas de objetos:
  cuántos ---> calcParaMí calcSobreDetalle ---> calcParaMí
  clasificarDetalle ---> valúeMe

- Ejemplos:
  aeropuerto-detalle de aeropuerto; almacén-detalle de almacén; depósito-detalle de depósito
  - Combinaciones:
    Ítem-detalle de ítem; ítem específico- detalle de ítem. También, cuando "contenedor" es un participante, lugar o ítem específico: actor-participante; participante-transacción; lugar-transacción; ítem específico- transacción; detalle de ítem específico-detalle de ítem; ítem-ítem específico.

  - Nota:
    Cuando trabaje con contenedores, aplique este patrón al contenedor más pequeño en ese dominio, con sus responsabilidades del sistema.

#16. Patrón: "Grupo-Miembro"	patrones de agregación

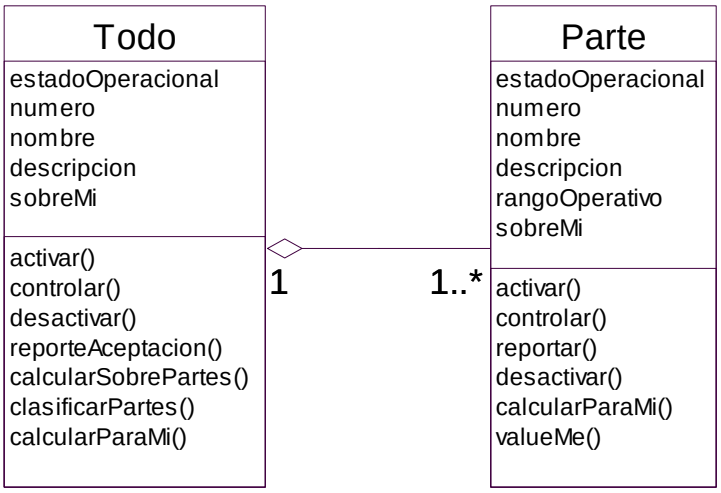


- Interacciones típicas de objetos:
  cuántos ---> calcParaMí calcSobreMiembros ---> calcParaMí
  clasificarMiembros ---> valúeMe

- Ejemplos:
  compañía-empleado; equipo-miembro del equipo

- Combinaciones:
  Grupo-miembro (otra vez), contenedor- contenido, todo-parte. También, cuando “grupo” o “miembro”es un participante, lugar, o ítem específico: actor-participante; participante-transacción; lugar-transacción; ítem específico-transacción; detalle de ítem específico-detalle de ítem; ítem-ítem específico.

#17. Patrón: "Todo-Parte"	patrones de agregación

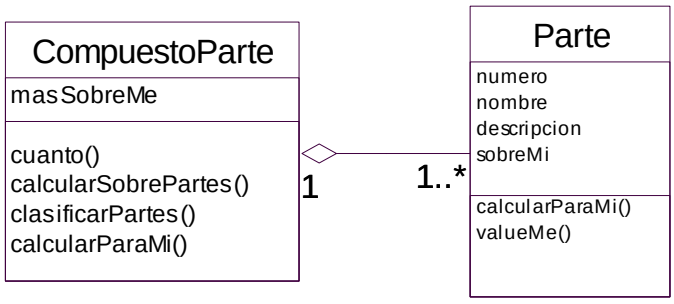


- Interacciones típicas de objetos:
  controlar ---> controlar
  reporteAceptación → reporte
  cuantos → calcParaMi
  calcSobrePartes ---> calcParaMí
  clasificarPartes ---> valúeMe

- Ejemplos:
  Aeroplano-motor; motor-parte del motor.

- Combinaciones:
  Todo-parte (otra vez), contenedor- contenido, grupo-miembro. También, cuando "todo" o "parte" es un ítem específico: ítem específico-transacción; ítem específico-detalle de ítem; ítem-ítem específico.

#18. Patrón: "Compuesto de parte-Parte"	patrones de agregación

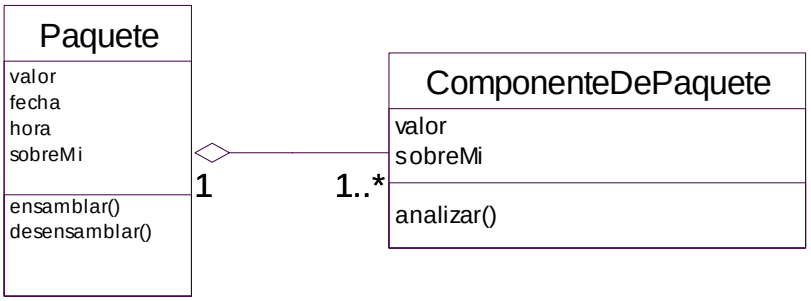


- Interacciones típicas de objetos:
  calcSobrePartes ---> calcParaMí clasificarPartes ---> valúeMe
  cuantos → calcParaMi

- Ejemplos:
  Compuesto de parte-parte; ensamble electrónico-parte electrónica; ensamble de hardware-parte de hardware.

- Combinaciones:
  Todo-parte. También, cuando "compuesto de parte" o "parte" es tratado como un ítem específico: ítem específico-transacción; ítem específico-detalle de ítem; ítem-ítem específico.

#19. Patrón: "Paquete-Componente de Paquete"	patrones de agregación



- Interacciones típicas de objetos:
  desensamblar → crear ensamblar → obtenerValor
- Ejemplos:
  registro-campo; mensaje- señal, paquete telemétrico- componente telemétrico.

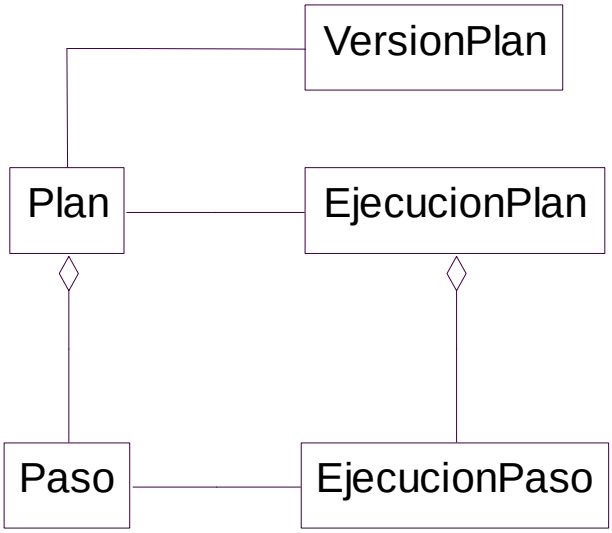
- Combinaciones:
  Cuando "componente de paquete" es tratado como un ítem específico: ítem específico-transacción; ítem específico-detalle de ítem; ítem-ítem específico.

Patterns de Plan

Los patrones de plan son:

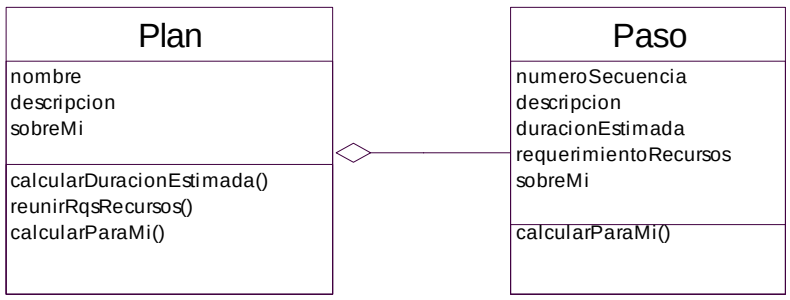
- plan-paso.
- plan-ejecución de plan
- plan-ejecución- paso de ejecución
- paso-paso de ejecución
- plan-versión de plan

Aquí está una descripción de los patrones de plan:



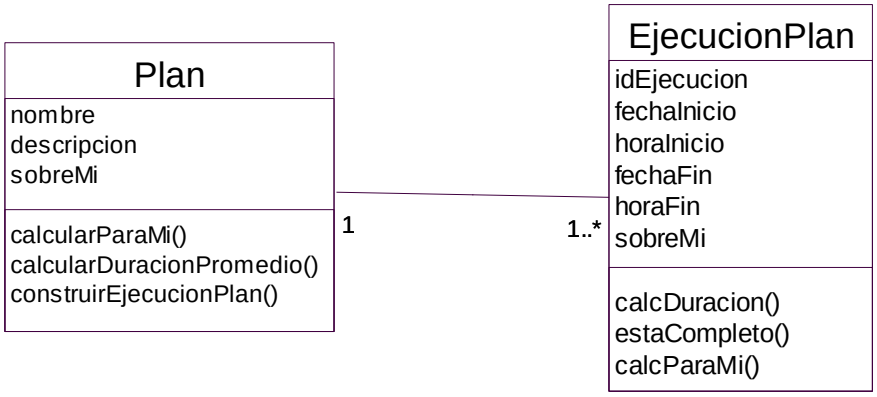
Una descripción de los patrones de plan.

# 20. Patrón: "Plan-Paso "	patrones de plan



- Interacciones típicas de objetos:
  calcularDuracionEstimada → obtenerDuracionestimada
  reunirRqsRecursos → obtenerRequerimRecursos
- Ejemplos:
  Planes: Plan de proyecto; plan de batalla; receta de cocina, procedimiento; trabajo en lote.
  Plan-Paso: Plan de proyecto-tarea; plan de batalla-pasos tácticos; receta-pasos de receta; procedimiento- paso de procedimiento; trabajo-paso de trabajo.
- Combinaciones:
  plan-ejecución de plan; paso- ejecución de paso, plan- versión de plan.

# 21. Patrón: "Plan-Ejecución de Plan"	patrones de plan

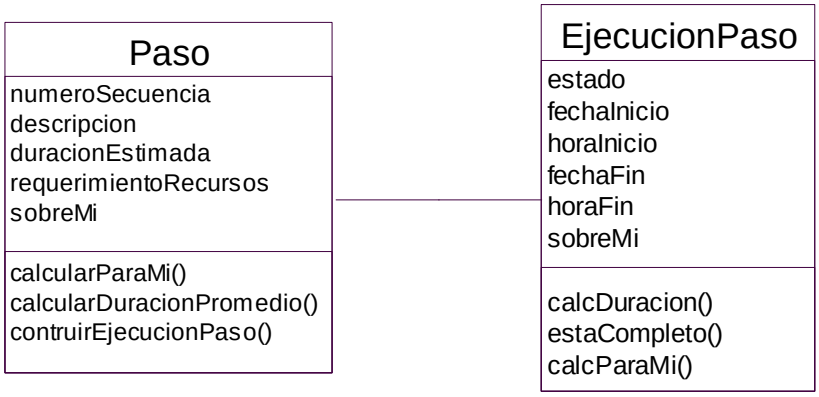


- Interacciones típicas de objetos:
  calcDuracionPromedio → calcDuracion construirEjecucionPlan → crear
- Ejemplos:
  plan de batalla- ejecución de batalla; receta-ejecución de receta; trabajo- ejecución de trabajo;
  procedimiento- ejecución de procedimiento.

- Combinaciones:
  plan- paso; ejecución de plan- ejecución de paso; plan- versión de plan.

- Nota:
  Esta en una ejecución real de un plan en una fecha y una hora. Use este pattern cuando un plan pueda ser ejecutado varias veces. (Si el plan es ejecutado una sola vez, las responsabilidades por la ejecución del plan podrían agregarse al plan.)

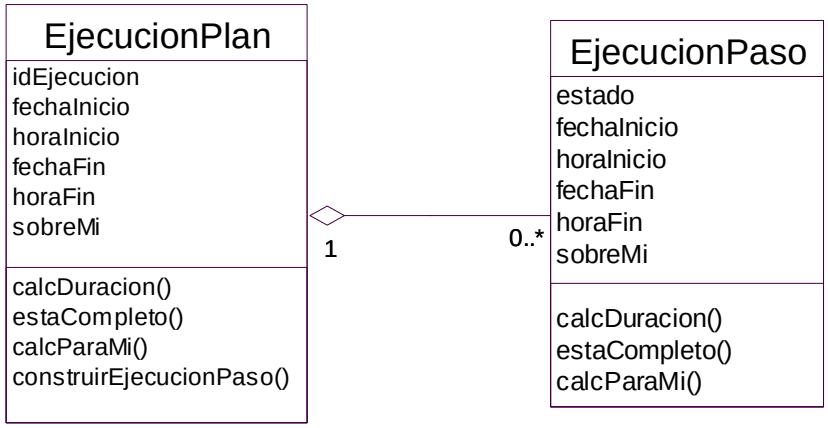
# 22. Patrón: "Paso-Ejecución de Paso"	patrones de plan



- Interacciones típicas de objetos:
  calcularDuracionPromedio → calcDuracion
  construirEjecucionPaso → crear
- Ejemplos:
  táctica-ejecución táctica; ejecución de receta – ejecución de paso de receta; paso de trabajo- ejecución de paso de trabajo; paso de procedimiento- ejecución de paso de procedimiento.

- Combinaciones:
  plan- paso; ejecución de plan- ejecución de paso.

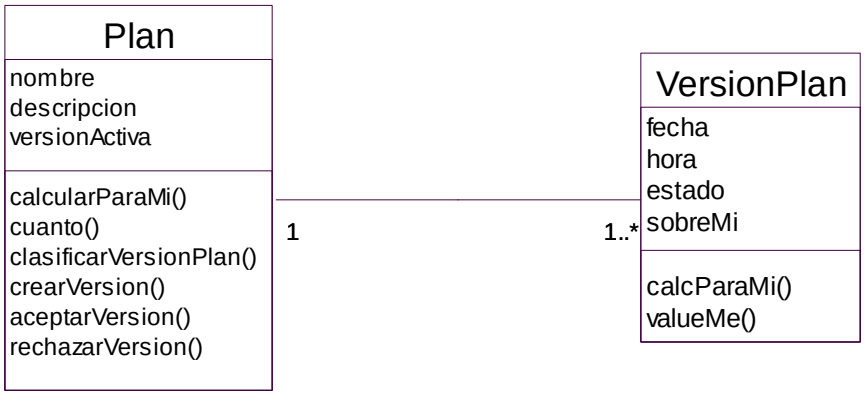
# 23. Patrón: "Ejecución de Plan –Ejecución de Paso "	patrones de plan



- Interacciones típicas de objetos:
  calcDuracion → calcDuracion
  estaCompleto → estaCompleto construirEjecucionPaso → crear
- Ejemplos:
  ejecución de batalla –ejecución de tácticas; ejecución de receta – ejecución de paso de receta; ejecución de trabajo- ejecución de paso de trabajo; ejecución de procedimiento- ejecución de paso de procedimiento.

- Combinaciones:
  plan- paso; plan- ejecución de plan.

# 24. Patrón: "Plan-Versión de Plan "	patrones de plan



- Interacciones típicas de objetos:
  cuantos → calcParaMi
  clasificar → valuaMe
  crearVersion → crear
  aceptarVersion → setStatus
  rechazarVersion → setStatus
- Ejemplos
  - Combinaciones:
    plan- paso; plan- ejecución de plan.

Patrones para Asignación de Responsabilidades: (GRASP)

Se aplican durante la construcción de diagramas de interacción, al asignar responsabilidades a los objetos y al diseñar la colaboración entre ellos.
Es importante acordar una definición de **responsabilidad**: contrato u obligación de un tipo o clase. Las responsabilidades se relacionan con las obligaciones de un objeto respecto a su comportamiento. Las responsabilidades pertenecen básicamente a dos categorías: **el conocer** y **el hacer**.

Las responsabilidades relacionadas con el **hacer**:

- Hacer algo en uno mismo.
- Iniciar una acción en otros objetos.
- Controlar y coordinar actividades en otros objetos.

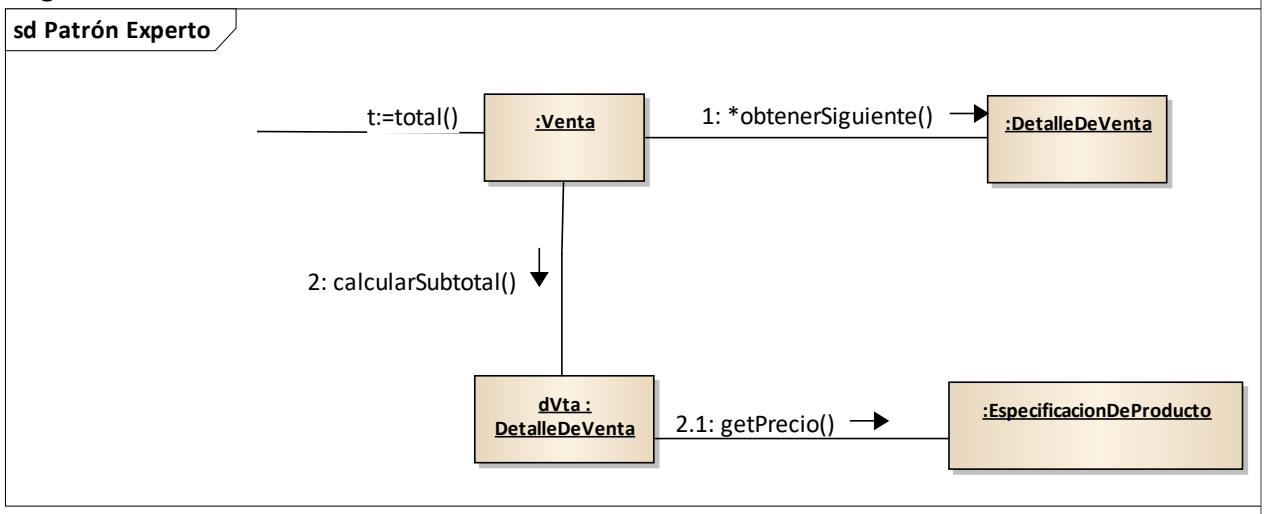
Las responsabilidades relacionadas con el **conocer**:

- Estar enterado de los datos privados encapsulados.
- Estar enterado de la existencia de objetos conexos.
- Estar enterado de cosas que se pueden derivar o calcular.

Solución: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir con la responsabilidad.

Problema: ¿Cuál es el principio fundamental en virtud del cual se asignan las responsabilidades en el diseño orientado a objetos?
Comience asignando las responsabilidades con una definición clara de ellas.

Ejemplo: Sistema de Punto de Venta, ¿quién debe conocer el total de una venta?
Diagrama de Comunicación:



Conclusión:

Venta: conoce el total de la venta
DetalleDeVenta: conoce el subtotal de la Línea de Producto
EspecificaciónDeProducto: conoce el precio del producto.

Explicación:

- Expresa la idea que los objetos hacen cosas relacionadas con la información que poseen.
- Puede haber expertos parciales (ej. DetalleDeVenta), que ayudan para cumplir con una responsabilidad.

Beneficios:

- Se conserva bajo el acoplamiento lo que favorece a tener sistemas robustos y de fácil mantenimiento.
- El comportamiento se distribuye con las clases que cuentan con la información requerida fomentando la creación de clases sencillas y cohesivas.

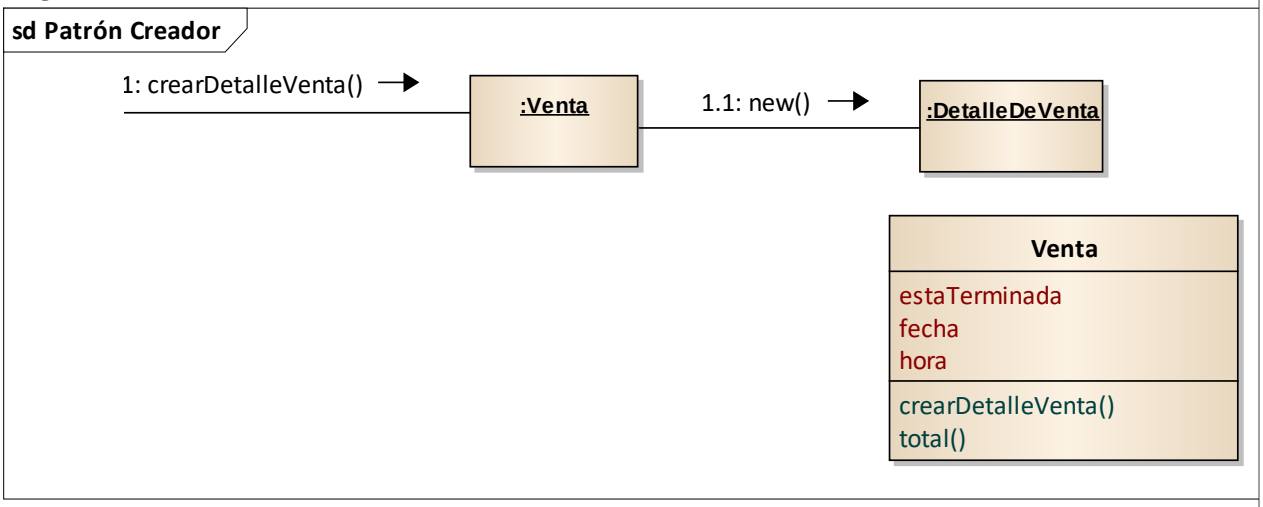
Otras formas de designar este patrón: "Juntar responsabilidades e información"; "Lo hace el que conoce"; "Animación"; "Lo hago yo mismo"; "Unir los servicios a los atributos sobre los que operan"

Solución: Asignar a la clase B la responsabilidad de crear una instancia de la clase A en uno de los siguientes casos:

- B agrega los objetos de A.
- B contiene los objetos de A.
- B registra las instancias de los objetos de A.
- B utiliza específicamente los objetos de A.
- B tiene los datos de inicialización que serán transmitidos a A cuando sea creado (así B es un experto respecto de la creación de A).

Problema: ¿Quién debería ser el responsable de crear una nueva instancia de alguna clase?

Ejemplo: Sistema de Punto de Venta, ¿quién debe crear el detalle de venta?
Diagrama de Comunicación:



Conclusión:

Venta: crea el DetalleDeVenta

Explicación:

- Guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito es encontrar un creador que debemos conectar con el objeto producido, esto soporta el bajo acoplamiento.
- El agregado “agrega” la parte, el contenedor “contiene” el contenido, el registro “registra”.

Beneficios:

- Da soporte al bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. como la clase creada tiende a ser visible al creador (razón por la cual se la eligió), el acoplamiento no aumenta.

Patrones Conexos:

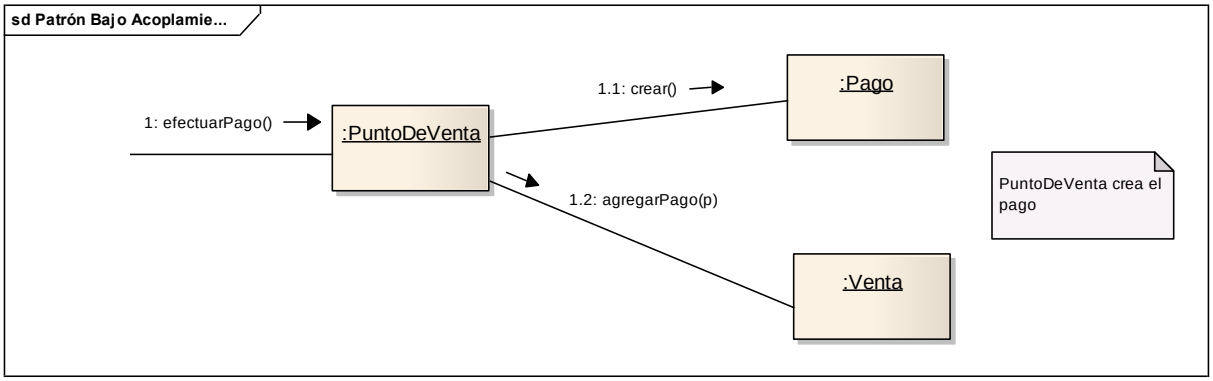
- Bajo Acoplamiento.
- Parte- Todo: define los objetos que soportan el encapsulamiento de sus componentes.

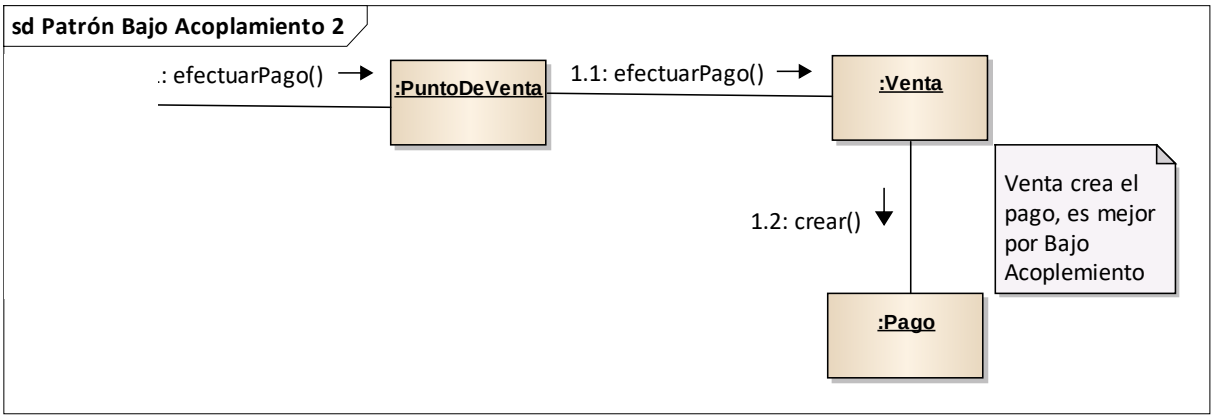
Solución: Asignar una responsabilidad para mantener bajo el acoplamiento.

Problema: ¿Cómo dar soporte a una dependencia escasa y a un aumento de reutilización? Las clases con acoplamiento alto o fuerte presentan los siguientes problemas:

- Los cambios de las clases afines ocasionan cambios locales.
- Son más fáciles de entender cuando están aisladas.
- Son más difíciles de reutilizar porque se requiere la presencia de otras clases de las que depende.

Ejemplo:





Conclusión:

Venta: crea el Pago.

Explicación:

Es la meta principal a tener en cuenta durante el diseño. Es el patrón "evaluativo" que aplica un diseñador al evaluar sus decisiones de diseño. Las formas comunes de acoplamiento de Tipo X a Tipo Y son:

Tipo X **posee un atributo** que se refiere a una instancia Tipo Y o al propio Tipo Y.

Tipo X **tiene un método** que referencia una instancia Tipo Y o al propio Tipo Y. Suele incluirse un parámetro o variable local Tipo Y o el objeto devuelto de un mensaje es una instancia de Tipo Y.

Tipo Y es una interfaz y Tipo X la implementa.

Tipo X es una subclase directa o indirecta de Tipo Y.

El bajo acoplamiento estimula asignar una responsabilidad de modo tal que no incremente el acoplamiento.

El patrón de Bajo Acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que aumenta la oportunidad de mayor productividad.

Un grado moderado de acoplamiento es normal y necesario si quiere crearse un sistema orientado a objetos, donde los objetos colaboran entre sí.

Beneficios:
No se afectan componentes por cambios de otros componentes.
Fáciles de entender por separado.
Fáciles de Reutilizar.

Patrón Alta Cohesión

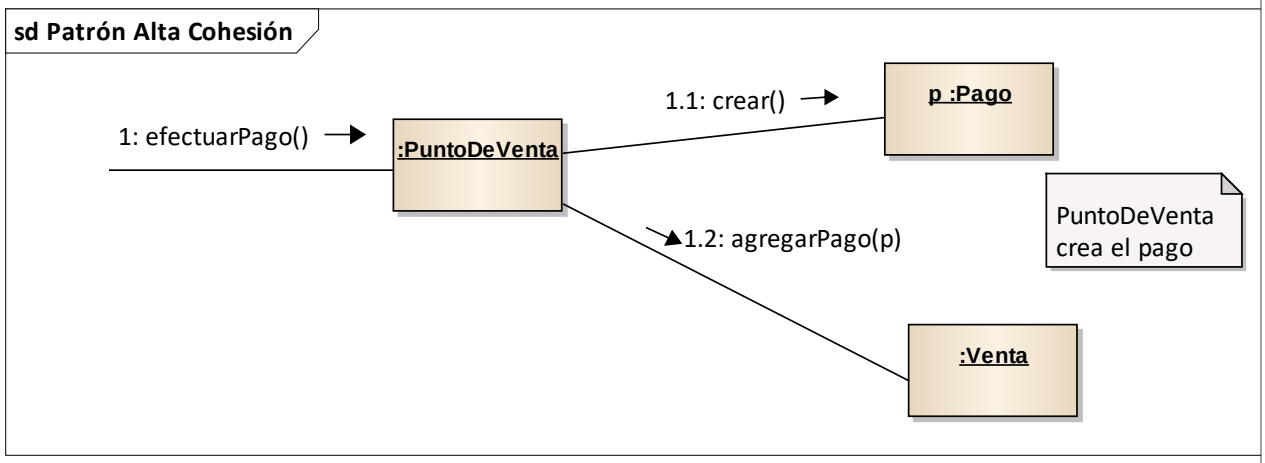
Solución: Asignar una responsabilidad de modo que la cohesión siga siendo alta.

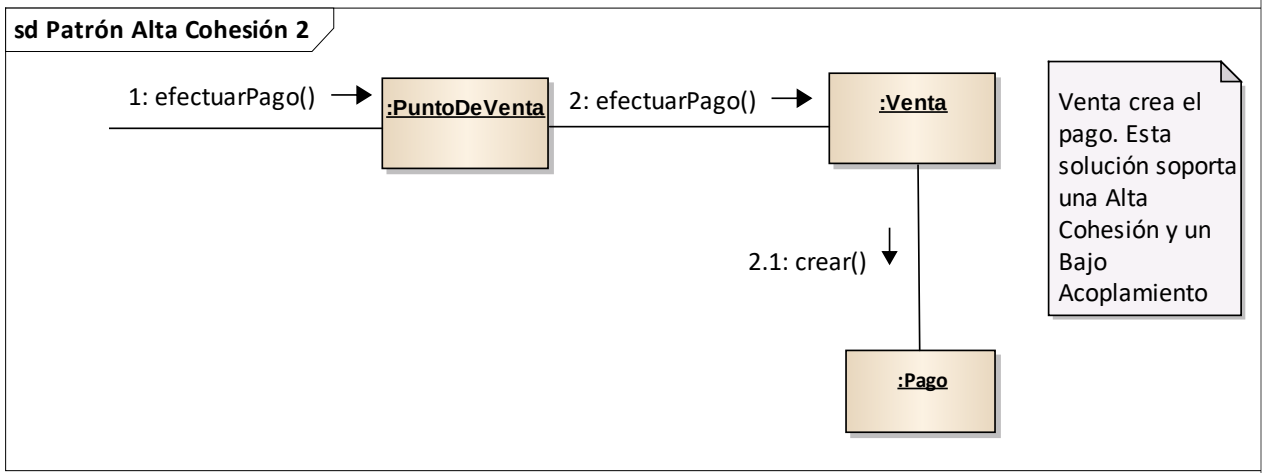
Problema: ¿Cómo mantener la complejidad dentro de límites manejables?
La cohesión funcional es la medida de cuan enfocadas o relacionadas están las responsabilidades de una clase.
Una alta cohesión caracteriza a las clases con responsabilidades muy relacionadas que no realicen un trabajo enorme. Las clases con baja cohesión presentan los siguientes problemas:

- Son difíciles de entender.
- Son difíciles de reutilizar.
- Son difíciles de conservar.
- Son delicadas: las afectan constantemente los cambios.

Las clases con baja cohesión a menudo representan un grado de abstracción alto o han asumido responsabilidades que deberían haber delegado en otros objetos.

Ejemplo:





Conclusión:

Venta: crea el Pago.

Explicación:

Al igual que el Patrón de Bajo Acoplamiento es un patrón que debe tenerse en cuenta al tomar las decisiones de diseño. Es un patrón evaluativo que el diseñador aplica al valorar sus decisiones.

Una clase de alta cohesión posee un número relativamente pequeño de operaciones con una importante funcionalidad relacionada y poco trabajo por hacer. Colabora con otros objetos para compartir esfuerzo si la tarea es grande.

Se asemeja al mundo real, si alguien asume demasiada responsabilidad sobre todo la que se debe delegar, no será eficiente.

Beneficios:

Mejoran la claridad y la facilidad con que se entiende el diseño.

Se simplifican el mantenimiento y las mejoras en funcionalidad.

A menudo se genera el bajo acoplamiento.

La ventaja de una gran funcionalidad es que soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.

Solución: Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema, a una clase que represente una de las siguientes opciones:

- El sistema "global" (controlador de fachada).
- La empresa u organización global (controlador de fachada).
- Algo en el mundo real que es activo (por ejemplo el papel de una persona) ya que pueda participar en la tarea (controlador de tareas).
- Un manejador artificial de todos los eventos del sistema de un caso de uso, generalmente denominado "Manejador <Nombre de Caso de Uso> (controlador de use case).

Utilice la misma clase de controlador con todos los eventos del sistema en el mismo caso de uso.
No utilice clases como: ventana, vista, documento, generalmente las reciben y delegan al controlador.

Problema: ¿Quién debería encargarse de atender un evento del sistema?
Un evento del sistema es un **evento de alto nivel generado por el actor externo; se asocia a operaciones del sistema: que emite en respuesta a los eventos**.
Un controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Define además el método de su operación.

Ejemplo: ¿Quién debería manejar los eventos del sistema: terminarVenta(); introducirProducto(); efectuarPago()?
De acuerdo al patrón controlador tenemos estas opciones:

- Representa al sistema global → TPDV
- Representa la empresa u organización → Tienda
- Representa algo activo → Cajero
- Representa al manejador del caso de Uso → ManejadorVenderProductos

Explicación:
La mayor parte de los sistemas reciben eventos de entrada externa, que generalmente incluyen una IGU (Interfaz Gráfica de Usuario) operada por una persona, o señales de sensores.
En todos los casos en un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada.
La misma clase debería usarse con todos los eventos de un caso de uso, de modo que podamos conservar la información referente al estado del caso de uso, lo que servirá para identificar eventos fuera de secuencia (ejemplo: efectuar Pago entes de Terminar venta).
Pueden usarse varios controladores en un caso de uso).
Un defecto frecuente es asignar demasiada responsabilidad al controlador, quién debería delegar la responsabilidad a otros objetos mientras coordina la actividad.

La primera categoría de controlador es un **Controlador de Fachada**, representan al sistema global, son adecuados cuando *el sistema solo tiene unos cuantos eventos o cuando es imposible redirigir los mensajes a otros controladores*.
La categoría de **manejador artificial de caso de uso**, plantea un controlador para cada caso de uso, que es un concepto artificial no un objeto del dominio, cuyo objetivo es dar soporte al sistema. ¿Cuándo usarlo? **Si cualquiera de las otras opciones genera diseños de alto acoplamiento o baja cohesión**. Esto ocurre cuando un controlador empieza a saturarse con demasiadas responsabilidades.
Un controlador de caso de uso es una buena alternativa cuando **hay muchos eventos del sistema entre varios procesos:** asigna su manejo a clases individuales controlables, además que ofrece una base para reflexionar sobre el estado del proceso actual.

Beneficios:
Mayor potencial de los componentes reusables: garantiza que la empresa o los procesos del dominio sean manejados por la capa del dominio y no por la de interfaz.

Reflexionar sobre el estado de un caso de uso: a veces no es necesario asegurar que las operaciones ocurran en una cierta secuencia legal o poder saber el estado actual de la actividad y las operaciones del caso de uso.

Problemas y Soluciones:

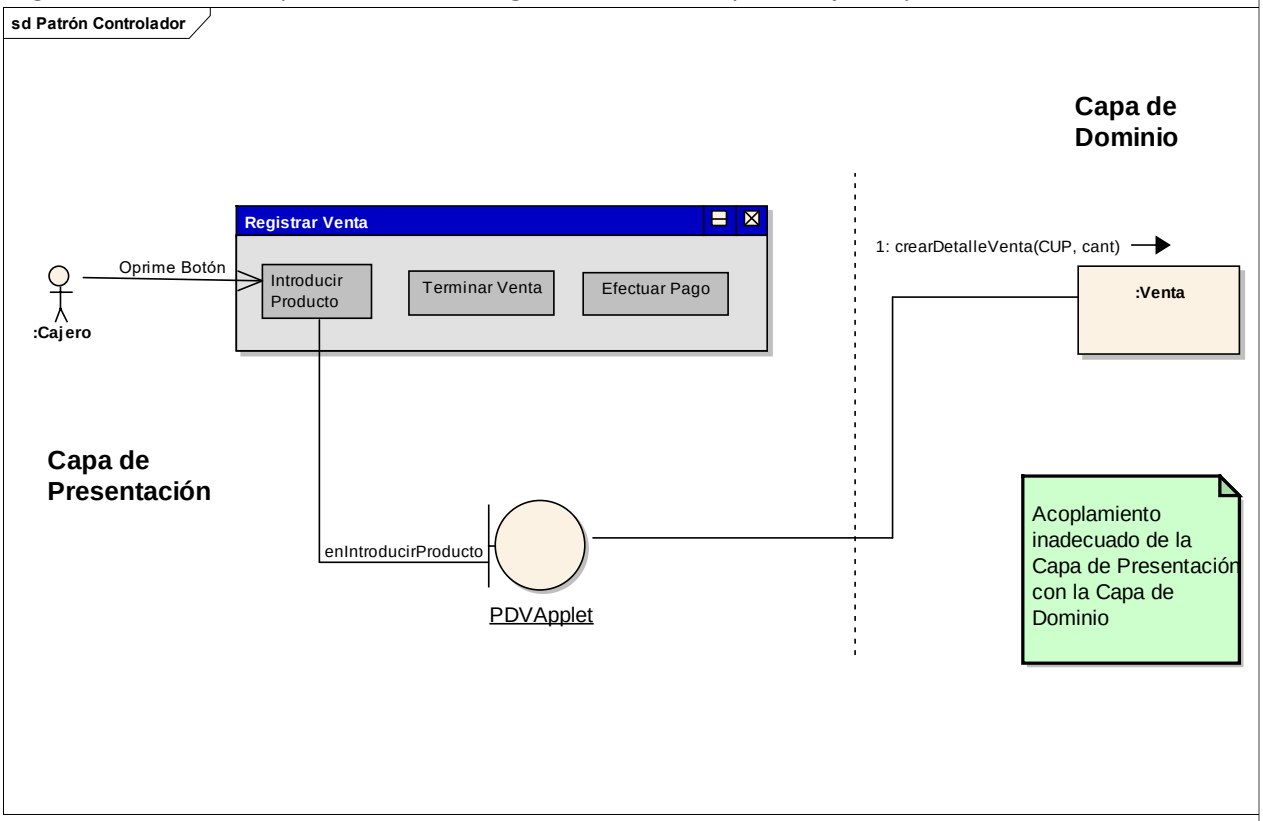
Controladores Saturados: un controlador mal diseñado presentará baja cohesión, dispersa y con demasiada responsabilidad, se lo denomina controlador saturado. Entre los signos de saturación podemos mencionar los siguientes:

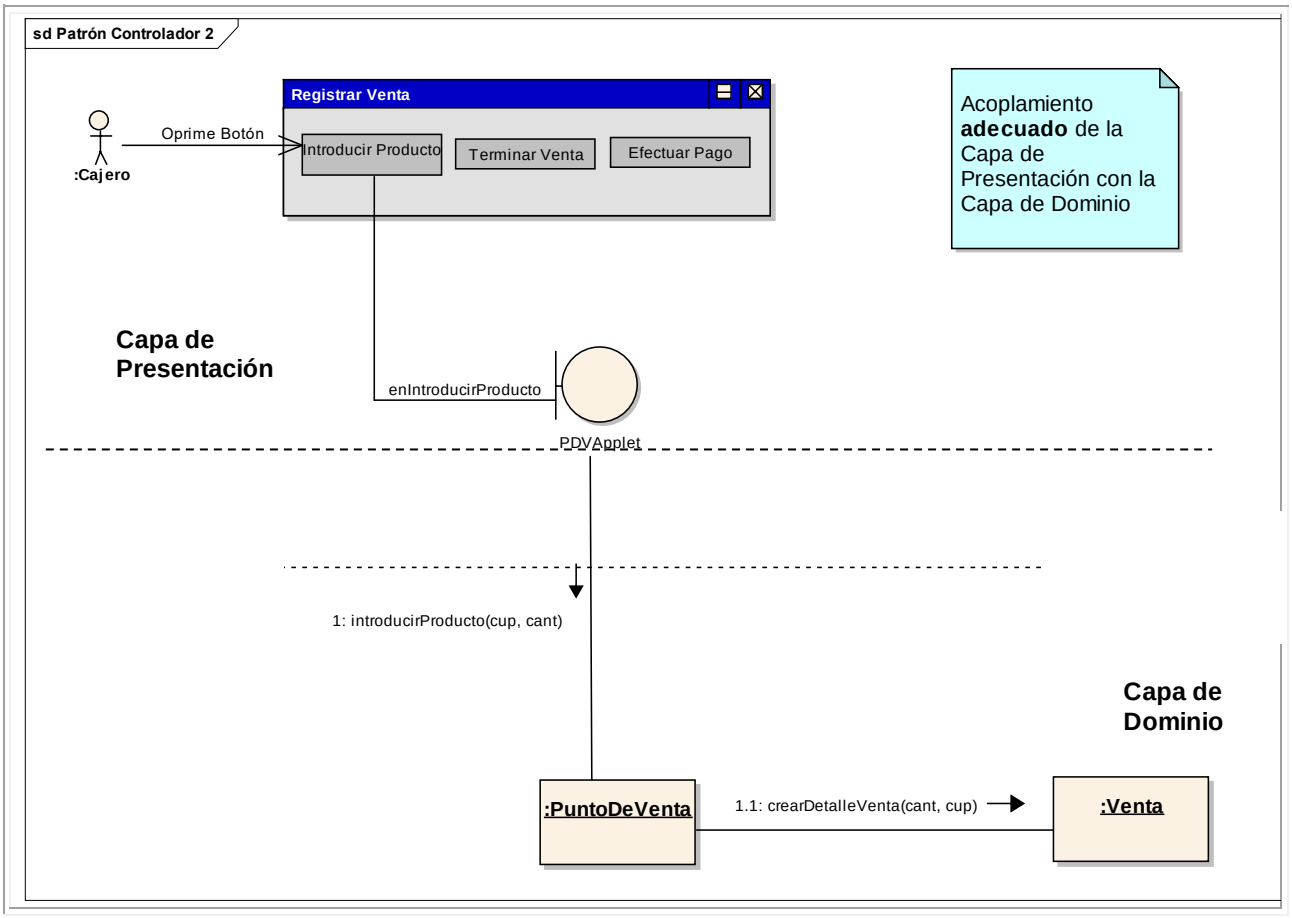
- Hay una sola clase controlador que recibe todos los eventos del sistema y éstos son excesivos. Tal situación suele ocurrir con un **controlador de papeles o de fachada.**
- El controlador realiza él mismo muchas tareas necesarias para cumplir el evento del sistema, **sin delegar el trabajo,** lo que suele ser una violación de los patrones Experto y Alta Cohesión.
- Un controlador posee muchos atributos y conserva información importante sobre el dominio, información que debería haber sido distribuida entre otros objetos y también duplica información en otra parte.

Hay varias formas de resolver el problema de un controlador saturado:

1- Agregar más controladores: un sistema no necesariamente debe tener uno solamente. Además del controlador de fachada, se recomienda los de papeles o de casos de uso.
2- Diseñe el controlador de modo que delegue fundamentalmente o otros objetos el desempeño de las responsabilidades de la operación del sistema.

Advertencia: los controladores de papeles pueden conducir a la obtención de diseños deficientes si se les asignan demasiadas responsabilidades, en general no son muy aconsejados para usarlos.





Patrón Polimorfismo

Solución: cuando por el tipo varían las alternativas o comportamientos afines, las responsabilidades del comportamiento se asignarán, mediante operaciones polimórficas, a los tipos en que el comportamiento presenta variantes.

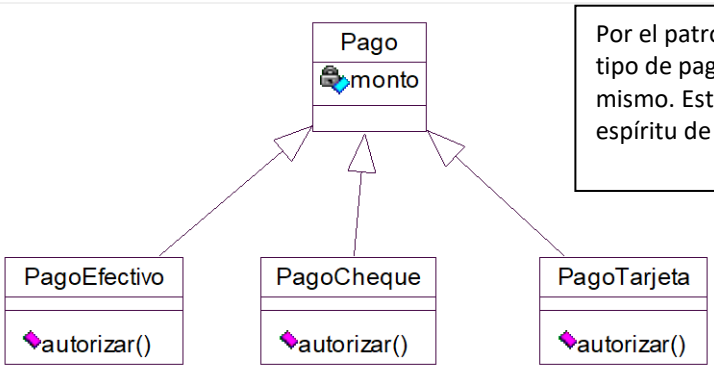
Corolario: no utilizar pruebas con el tipo de un objeto, ni utilizar lógica condicional para plantear diversas alternativas
basadas en el tipo.

Problema: ¿Cómo manejar alternativas basadas en el tipo? ¿De qué manera crear componentes de software conectables?

Alternativas basadas en el tipo: la variación condicional es un tema esencial en la programación, el usar estructuras IF-THEN-ELSE o CASE, dificulta la modificación (extensión)

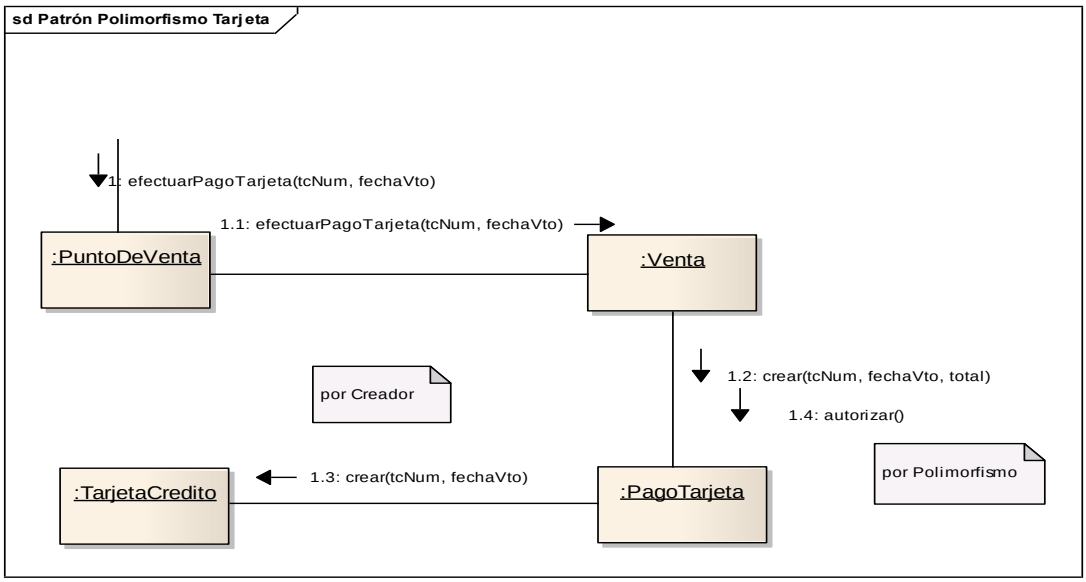
Componentes de Software conectables: viendo los componentes en las relaciones cliente- servidor ¿cómo podemos reemplazar un componente servidor sin afectar al cliente?

Ejemplo: Cada pago debe saber como autorizarse

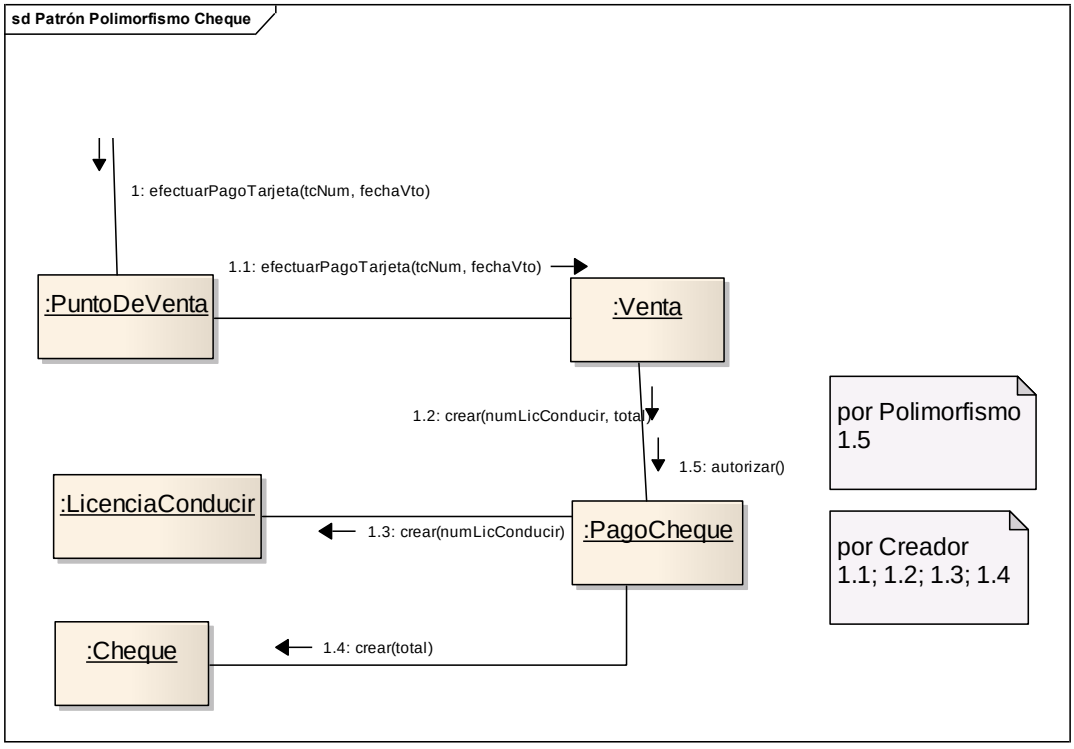


Por el patrón Polimorfismo cada tipo de pago ha de autorizarse a si mismo. Esto corresponde al espíritu de "Lo hago yo mismo".

Creación de un Pago con Tarjeta



Creación de un Pago con Cheque



Explicación:

- Este patrón está acorde con el patrón Experto y el espíritu de "Lo hago yo mismo".
- Un diseño basado en asignación de responsabilidades mediante el polimorfismo puede ser extendido fácilmente para que realice nuevas variantes.
- Cuando vemos los objetos en las relaciones cliente/servidor, los objetos cliente requieren poca o nula modificación el introducir un nuevo servidor a condición de que este soporte las operaciones polimórficas que espera el cliente.

Beneficios:

- Es fácil agregar futuras extensiones que requieren las variaciones imprevistas.

Conocido como o semejante a:

"Lo hago yo mismo"
"Elección del mensaje"
"No preguntes ¿qué clase es?"

Patrón Fabricación Pura

Solución: asignar un conjunto altamente cohesivo de responsabilidades a una clase artificial que no representa nada en el dominio del problema: una clase inventada para dar soporte a una alta cohesión, bajo acoplamiento y reutilización.

Problema: ¿A quién asignar la responsabilidad cuando uno está desesperado y no quiere violar los patrones de Alta cohesión y Bajo Acoplamiento?
Hay situaciones en las cuales la asignación de responsabilidades a las clases del dominio origina problemas de mala cohesión o acoplamiento o escaso potencial de reuso.

Ejemplo:
Supongamos que una clase del dominio VENTA necesita ser persistente y guardarse en un RDBMS (Administrador de Base de Datos Relacional). En virtud del Patrón Experto justifica que la responsabilidad sea de VENTA. Pero analicemos las siguientes implicancias:

- La tarea requiere un número considerable de operaciones de soporte orientadas a la base de datos, ninguna relacionada al concepto de vender, lo que reduce la COHESIÓN.
- La clase venta estaría acoplada a la interfaz de la BDR (que proporciona el proveedor) esta aumenta el acoplamiento y ni siquiera es con otro objeto/clase del dominio.
- Guardar objetos en la base de datos relacional es una tarea muy general en que debemos brindar soporte a muchas clases. Si se asignan a la clase venta habrá poca REUTILIZACIÓN o mucha DUPLICACIÓN en otras clases que cumplen la misma función.

Una solución razonable consiste en crear una clase nueva que se encargue de guardar objetos en algún tipo de almacenamiento persistente que puede llamarse AgenteAlmacenamientoPersistente, que sirve para resolver los siguientes problemas de diseño:

- La Venta conserva su buen diseño, con Alta cohesión y Bajo Acoplamiento.
- La clase AgenteAlmacenamientoPersistente es relativamente cohesiva pues su único propósito es guardar los objetos en un medio persistente que además es extremadamente genérico y reusable.

Explicación:
Para diseñar una fabricación pura debe buscarse un potencial de reutilización, asegurándose que sus responsabilidades sean pequeñas y cohesivas. En general son responsabilidades de Granularidad fina.
Se considera que la fabricación pura es parte de la capa de servicios orientada a objetos, de alto nivel en una arquitectura.

Beneficios:

- Alta cohesión
- Aumenta el potencial de reuso.

Problemas Posibles:

Puede perderse el espíritu de los buenos diseños orientados a objeto que se centran en objetos y no en funciones pues las clases de fabricación pura se dividen atendiendo a su funcionalidad. Si se abusa de esto, terminará en un diseño centrado en procesos que se implementa en un lenguaje orientado a objetos.

Patrones Relacionados:

- Bajo Acoplamiento
- Alta cohesión
- Una fabricación pura normalmente asume las responsabilidades de la clase del dominio a la cual se asignarían basándose en el patrón Experto.
- Muchos patrones son ejemplo del patrón de Fabricación Pura.

Solución: Asignar responsabilidad a un objeto intermedio para que medie entre otros componentes o servicios y éstos no terminen directamente acoplados.

Problema: ¿A quién se asignarán responsabilidades a fin de evitar el acoplamiento directo? ¿De qué manera desacoplar objetos de modo que se obtenga un bajo acoplamiento y se conserve un alto potencial de reutilización?

Ejemplo:

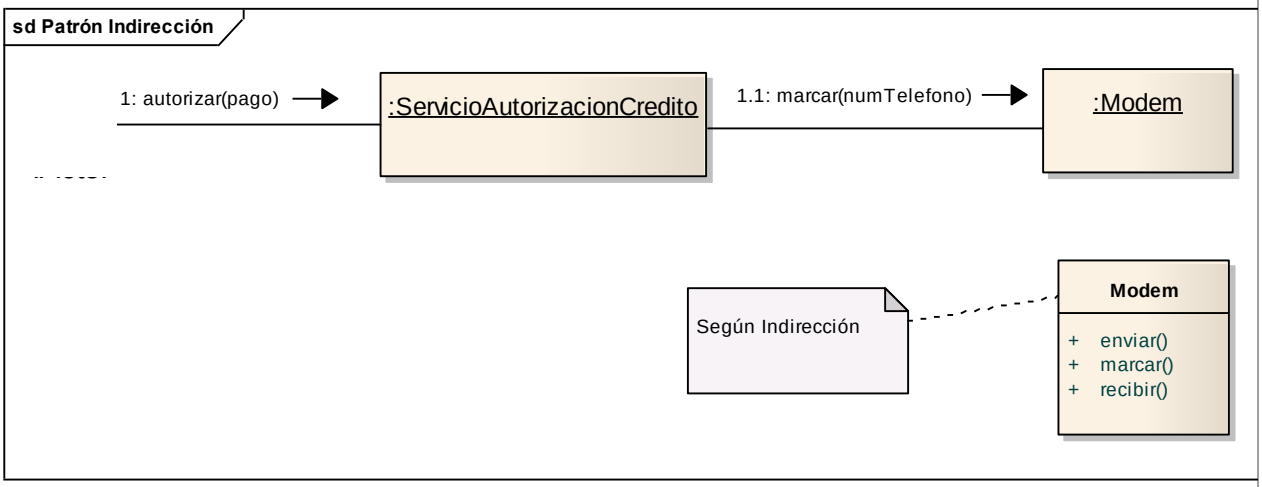
El ejemplo de fabricación pura para desacoplar la Venta y los servicios de la base de datos relacional introduciendo la clase AgenteAlmacenamientoPersistente, es un ejemplo para apoyar la indirección.

MODEM

Suponga que:

- Una aplicación de terminal situada en el Punto de Venta necesita usar un MODEM para transmitir solicitudes de pago con tarjeta de crédito.
- El sistema operativo tiene una llamada de función API de bajo nivel para hacer.
- Una clase ServicioDeAutorizacionDeCrédito asume la responsabilidad de hablar con el MODEM.

Si el ServicioDeAutorizacionDeCrédito invoca a la función API directamente estará acoplado con la API del sistema operativo en cuestión. se agrega una clase intermedia MODEM responsable de traducir los requerimientos de la clase a la API y viceversa.



Solución: Los objetos se suscriben e eventos ante un AdministradorDeEventos; otros publican eventos para el AdministradorDeEventos que los notifica a los suscriptores. A través de la indirección del AdministradorDeEventos se desacoplan los editores y los suscriptores.

Explicación:

- Muchos patrones actuales de diseño son especializaciones de la Fabricación pura, también muchos son especializaciones de la Indirección, ejemplo: ADAPTADOR, FACHADA y OBSERVADOR (GAMMA). Además muchas clases de Fabricación pura se generan como consecuencia del patrón indirección.
- El motivo de la Indirección casi siempre es el Bajo Acoplamiento; se agrega un intermediario con el fin de desacoplar otros componentes o servicios.

Beneficios:

- Bajo Acoplamiento.

Patrones Relacionados:

- Bajo Acoplamiento
- Mediador (GAMMA)
- Muchos intermediarios de Indirección son situaciones de Fabricación Pura.

Patrón No hables con Extraños

Solución:
Se asigna responsabilidad a un objeto directo del cliente para que colabore con uno indirecto de modo que el cliente no necesite saber nada del objeto indirecto. También conocido como Ley de Dementer, impone restricciones, a los objetos a los cuales deberíamos enviar mensajes dentro de un método. El patrón establece que en un método los mensajes deben ser enviados a los siguientes objetos:

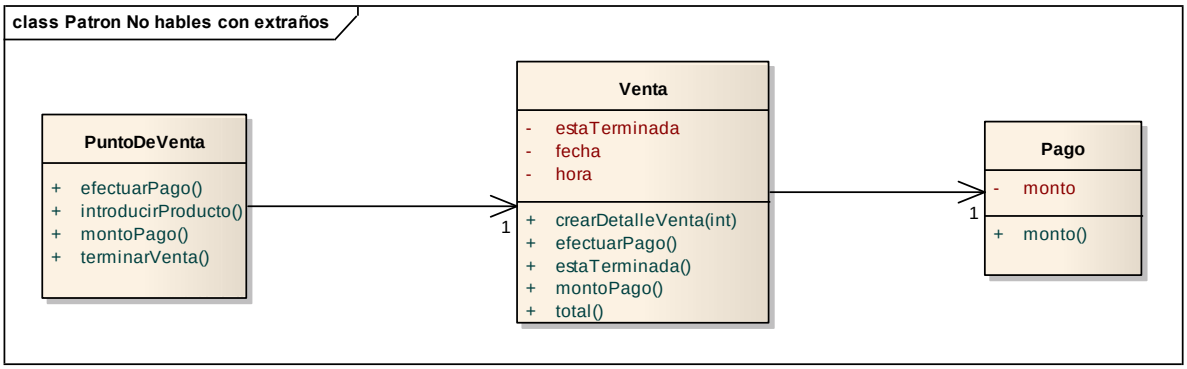
1- El objeto (self).
2- Un parámetro del método.
3- Un atributo de self.
4- Un elemento de una colección que sea atributo de self.
5- Un objeto creado en el interior del método.

Con esto se busca no acoplar al cliente al conocimiento de objetos indirectos, ni sus representaciones internas. Los objetos directos son "conocidos", los indirectos son "extraños", y un cliente debería tener solo conocidos.

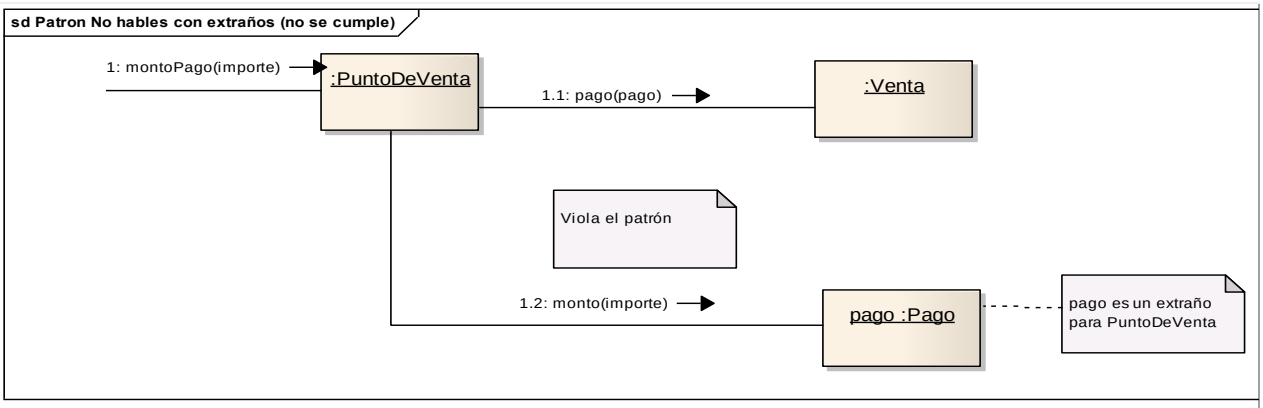
Problema: ¿A quién asignar las responsabilidades para evitar tener que conocer la estructura de los objetos indirectos? Si un objeto conoce las conexiones internas y estructuras de otros tendrá alto acoplamiento.
¿Si un objeto cliente necesita servicios o información de un objeto indirecto, como podrá hacerlo sin acoplarse?

Ejemplo:
Supongamos en una aplicación Punto de Venta que una instancia TPDV posee un atributo referente a una Venta, la cual cuenta con un atributo referente a un pago.
Y además suponga que:

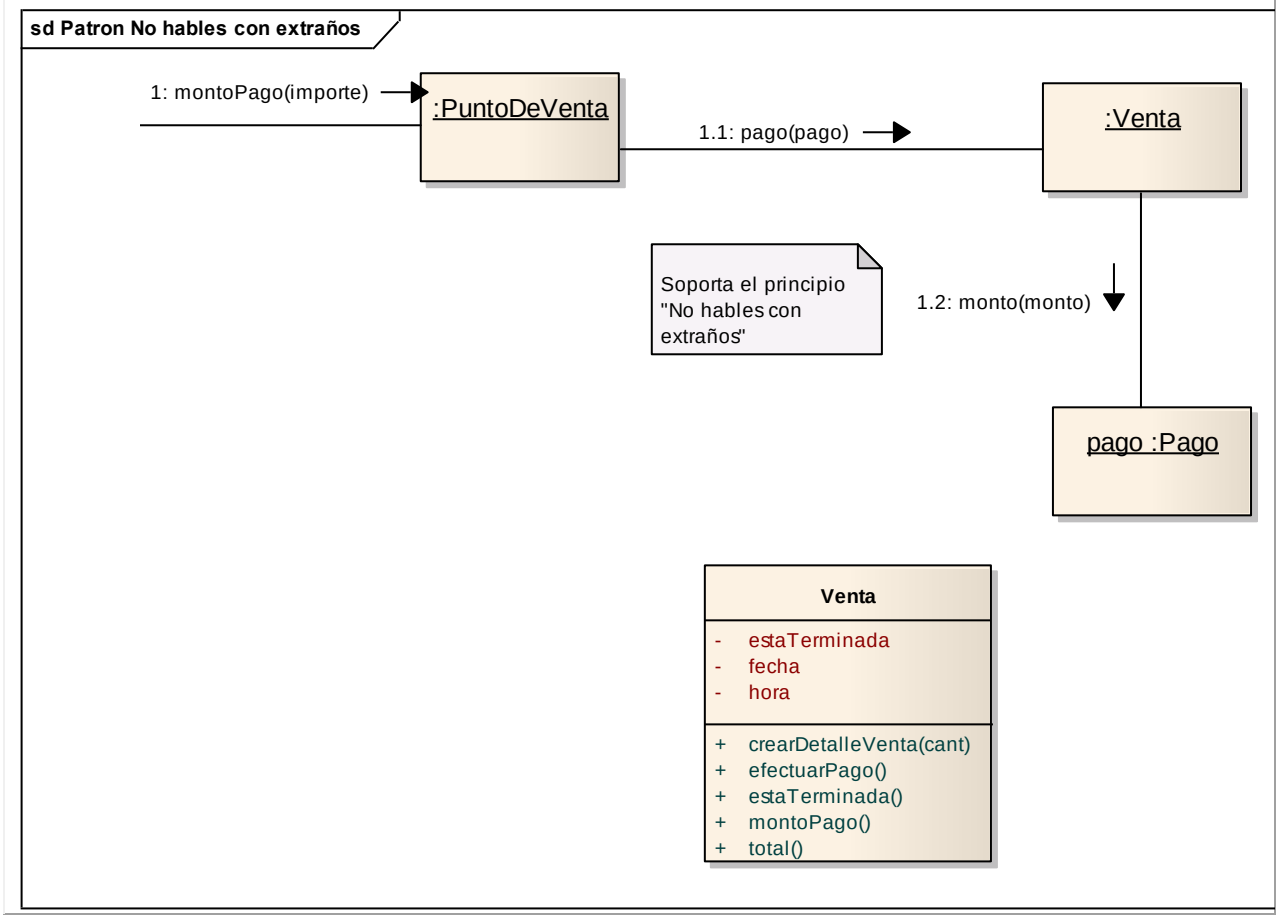
- Las instancias TPDV soportan la operación montoDePago, que devuelve el actual monto ofrecido como pago.
- Las instancias de Venta soportan la operación Pago, la cual devuelve la instancia Pago asociada a la Venta.



Una forma de ofrecer el monto de pago (violando el patrón de No hables con extraños)



La solución consiste en agregar una responsabilidad al objeto directo Venta para que devuelva a TPDV el monto de pago, de modo que la instancia no tenga que hablar con extraños.



Explicación:

- El patrón se refiere a no obtener una visibilidad temporal frente a objetos indirectos que son de conocimiento de otros objetos pero no del cliente.
- La desventaja: la solución se acopla entonces a la estructura interna de otros objetos. Ello origina alto acoplamiento, diseño menos robusto y más propenso a requerir cambios, si se alteran las relaciones estructurales indirectas.

Violación de la Ley:

Hay situaciones donde conviene prescindir de la ley, una situación común es cuando algún tipo de agente o “servidor de objetos” (generalmente Fabricación Pura) encargado de devolver otros objetos, basándose en una consulta mediante el valor de una clave. Se juzga aceptable obtener visibilidad ante un objeto X a través de un agente y luego enviarle directamente un mensaje a X aunque eso viole la Ley de Demeter.

Beneficios:

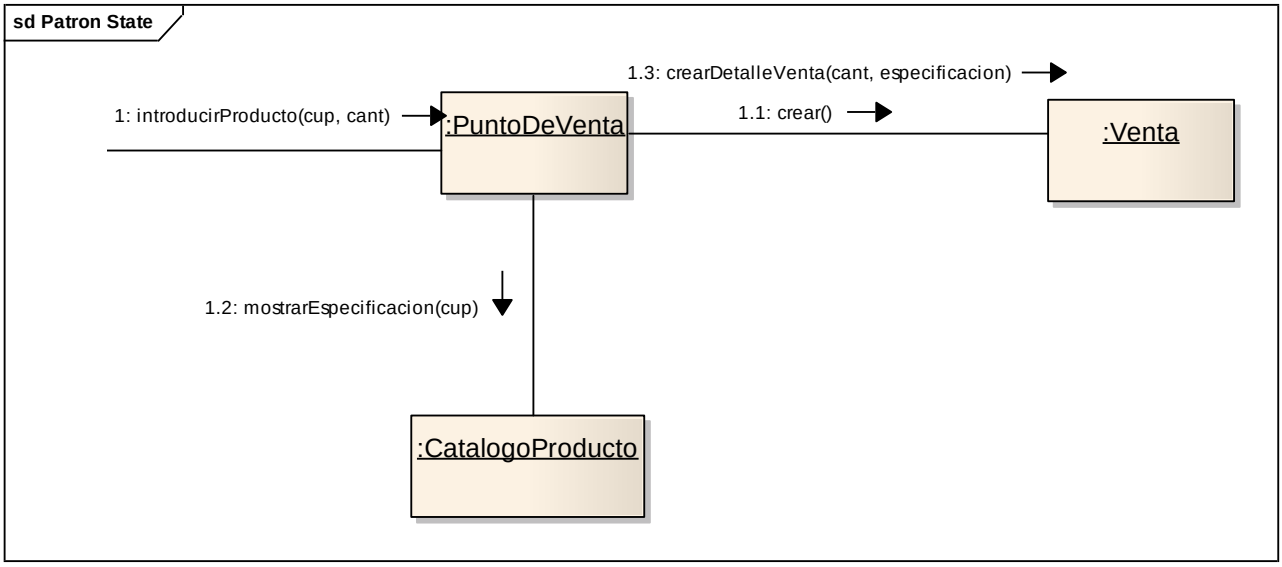
- Bajo Acoplamiento.

Patrones Relacionados:

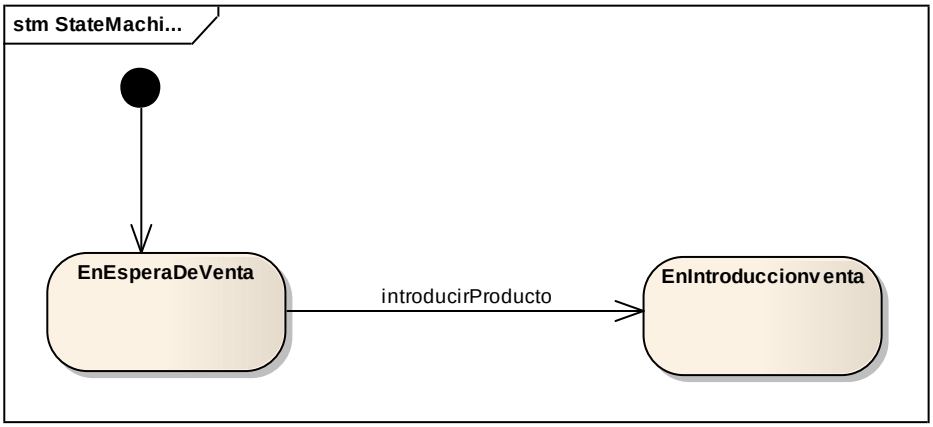
- Bajo Acoplamiento
- Indirección
- Cadena de responsabilidades (GAMMA)

Patrón Estado (GAMMA)

Cuando el comportamiento de una clase depende de su estado, en vez de servirse de una prueba condicional, se puede utilizar el patrón de Estado.



TPDV



Problema: El comportamiento de un objeto depende de su estado. La lógica condicional no es buena debido a su complejidad, escalabilidad o duplicación.

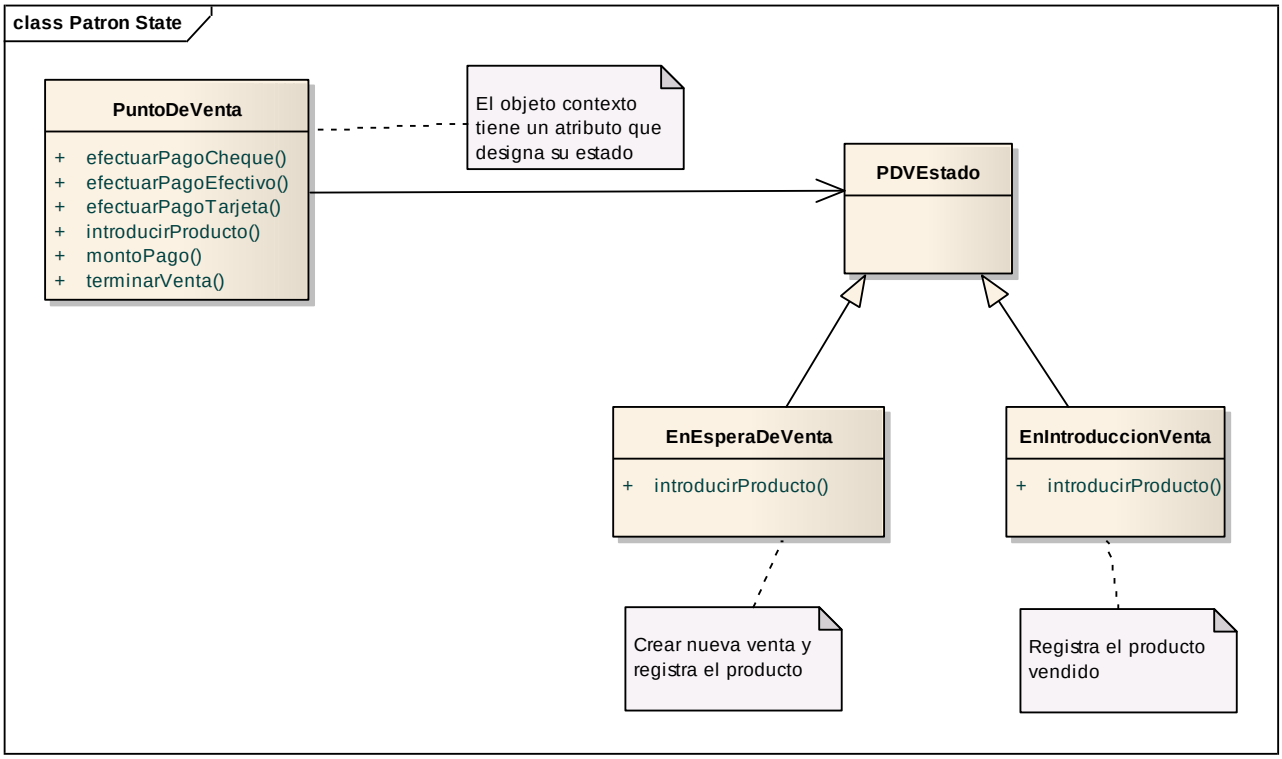
Solución:

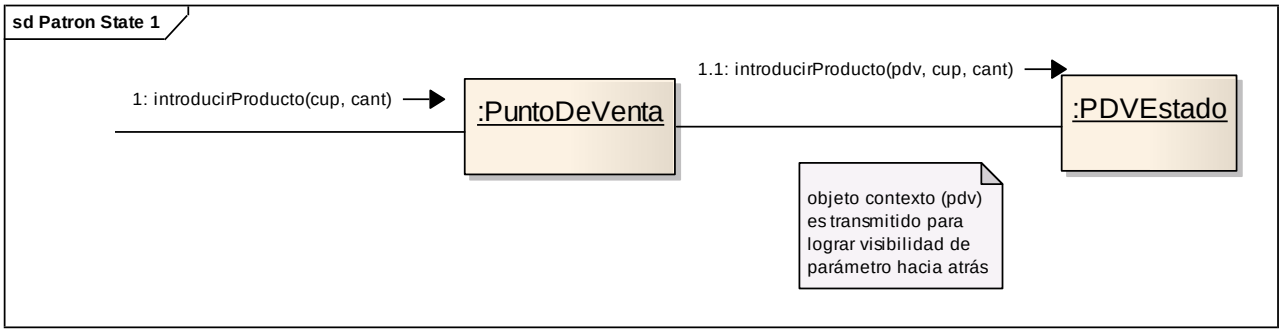
1- Crear una clase para cada estado que influya en el comportamiento del objeto dependiente del estado (el objeto "contexto").

2- Con base en el polimorfismo, asignar métodos a cada clase de estado para manejar el comportamiento del objeto contexto.

3- Cuando el objeto contexto reciba un mensaje dependiente del estado, el mensaje será enviado al objeto estado.

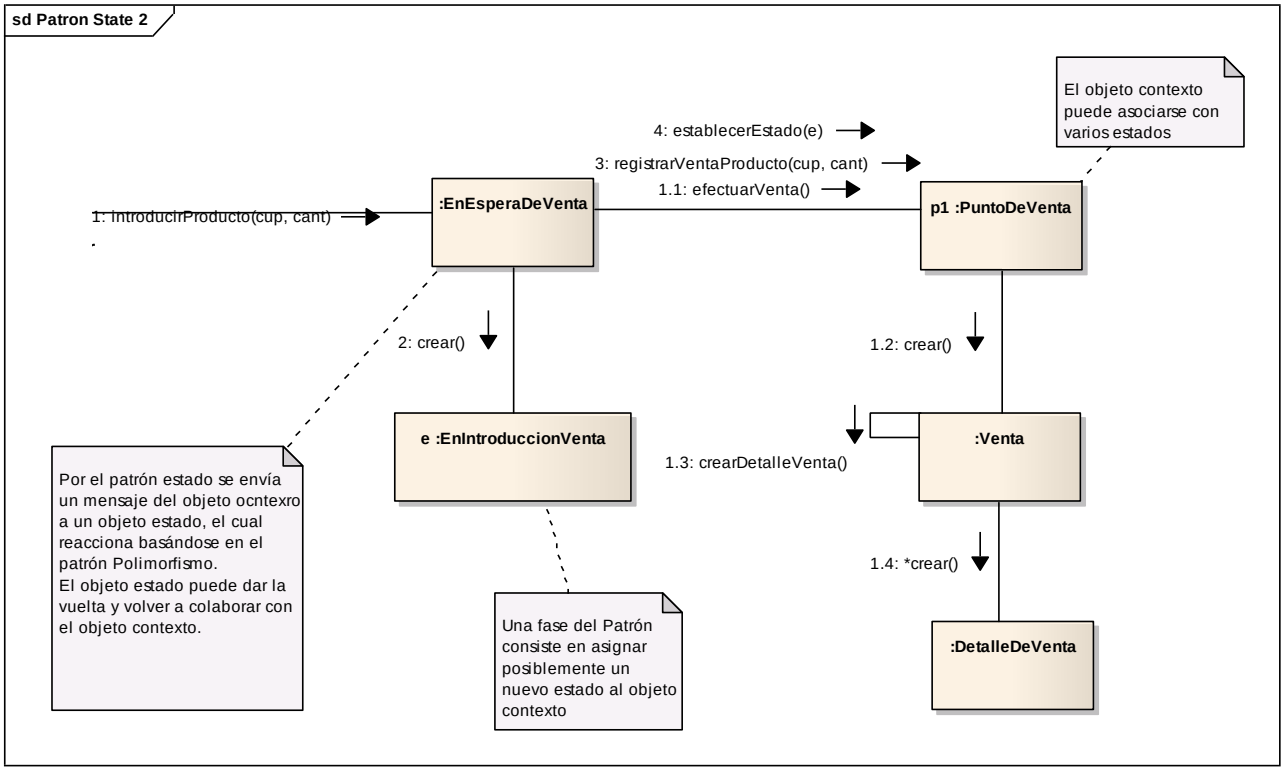
Ejemplo:
Diagrama de Clase

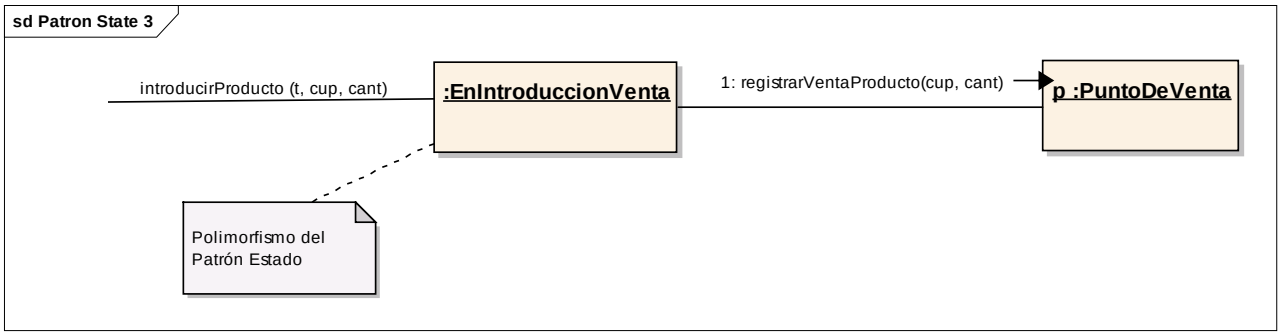




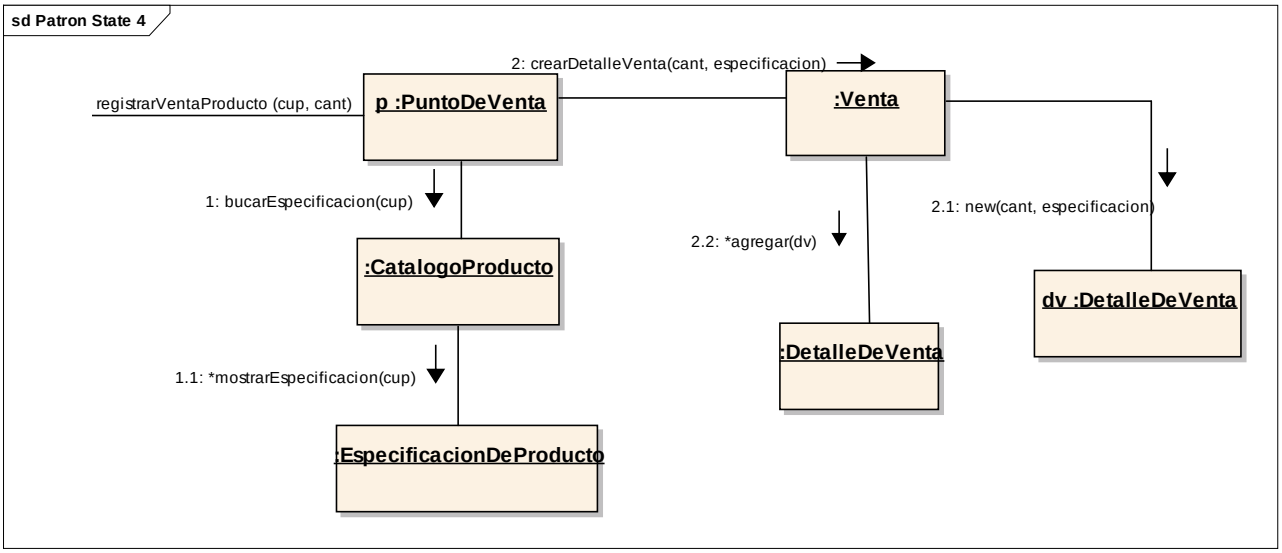
Para cada caso polimórfico de las concretas (subclases), se debe dibujar un diagrama de comunicación diferente, comenzando con el mensaje polimórfico.
En el diagrama de arriba se dibuja la clase abstracta PDVEstado, únicamente a los fines de acortar el modelado.

El patrón estado comienza con el objeto contexto (TPDV en este caso), enviando un mensaje al objeto estado asociado.





Finalmente, el Diagrama de Comunicación registrar Venta de Producto se muestra en la siguiente figura:



Conclusión:

- Este patrón es muy útil cuando el comportamiento de un objeto depende de su estado.
- Con él se elimina la lógica condicional en los métodos del objeto contexto y se obtiene un mecanismo elegante para extender el comportamiento de dicho objeto sin modificarlo.
- Si un sistema presenta muchos estados, este patrón puede ser idóneo por la proliferación de clases. Otra alternativa consiste en definir un intérprete de la máquina de estado que funcione contra un grupo de reglas de transición de estados.

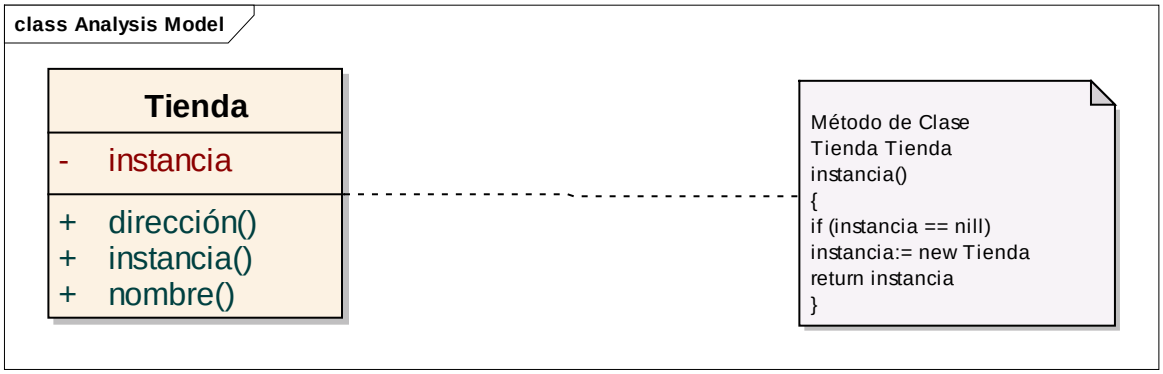
Patrón Singleton

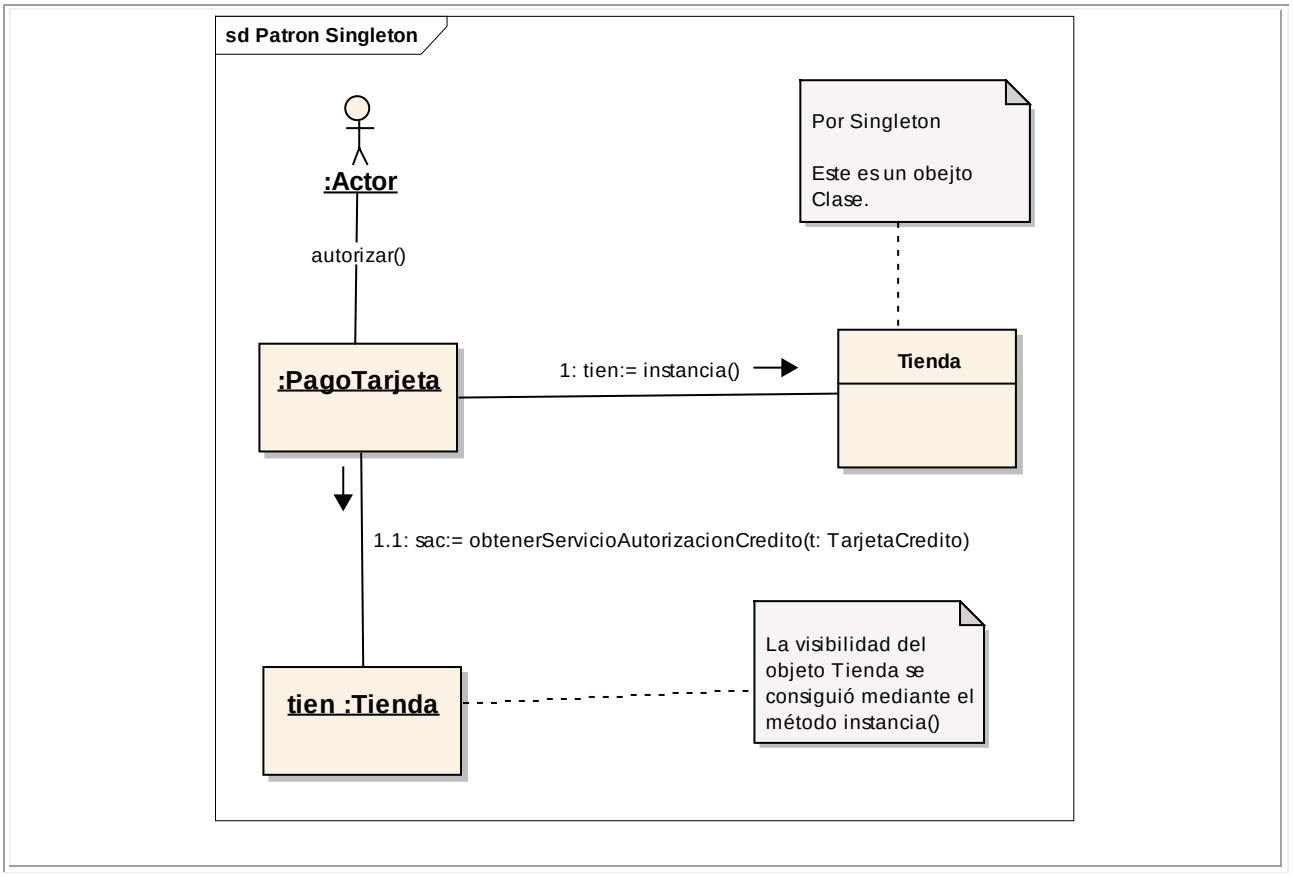
Problema: se permite exactamente una instancia de una clase: se trata de un singleton (solitario). Los objetos necesitan un solo punto de acceso.

Solución: Definir un método de clase (smalltalk) o una función que no sea miembro (en C++) y que devuelve el singleton.

Ejemplo:

- Cuando PagoTarjeta recibe el mensaje autorizar(), necesita enviar un mensaje al objeto TIENDA, que conoce cual servicio de autorización llamar, en función de la marca de tarjeta, puesto que TIENDA es el experto natural.
- Pero surge un problema de visibilidad: la instancia recién creada de PagoTarjeta no tiene acceso a la instancia TIENDA.
- Una solución consiste en transmitirla hacia abajo (como parámetro) desde TPDV (que posee visibilidad de atributo ante ella) y asignarla a un atributo de PagoTarjeta, para que también tenga visibilidad de atributo frente a ella. Lo anterior es aceptable, pero la opción es el SINGLETON.
- A veces conviene soportar la visibilidad global o un solo punto de acceso a una instancia individual de una clase y no a alguna forma de visibilidad. Y esto ocurre con la instancia TIENDA




sd Patron Singleton
:Actor
autorizar()
Por Singleton
Este es un obejto Clase.
:PagoTarjeta
1: tien:= instancia()
Tienda
1.1: sac:= obtenerServicioAutorizacionCredito(t: TarjetaCredito)
tien :Tienda
La visibilidad del objeto Tienda se consiguió mediante el método instancia()

Bibliografía de Referencia

- Gamma, Erich: DESIGN PATTERNS**. (Editorial Adison Wesley Año 1995).**
- Coad, Peter: OBJECT MODELS, STRATEGIES, PATTERNS & APLICATIONS **(Editorial Yourdon Press - Año 1995).**
- UML Y PATRONES – **Autor: Craig Larman (Editorial Prentice May - Año 1999).**

Historia de Cambios

Fecha	Versión	Descripción	Autor
	1.0	Versión Inicial	Judith Meles
05/08/2004	1.1	Se cambian algunos gráficos de patrones GRASP por que tenía el actor y no corresponde	Judith Meles
01/02/2010	1.2	Se cambian los gráficos a UML 2.0	Judith Meles
02/02/2010	1.3	Se cambia el nombre del diagrama de colaboración por el de Diagrama de Comunicación	Judith Meles
16/04/2013	1.4	Se corrigen los diagramas del Patrón de Alta Cohesión y del patrón creador	Judith Meles
24/04/2013	1.5	Se corrige el diagrama del Patrón de Bajo Acoplamiento	Judith Meles