GRASP

Responsabilidades y métodos

- Responsabilidad “Un contrato u obligación de un tipo o clase”[OMG]
- Obligaciones de un objeto respecto a su comportamiento
- Dos categorías:
  - Conocer
  - Hacer

Responsabilidades con “hacer”

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

Responsabilidades con “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

A menudo pueden inferirse del modelo conceptual por los atributos y asociaciones explicadas en él

Responsabilidades y métodos

- Diseño orientado a responsabilidades (RDD)
- Personas y sus responsabilidades que colaboran
- Los métodos se ponen en práctica para cumplir responsabilidades

Patrones

- “Descripción de un problema y su solución que recibe un nombre y que puede emplearse en otros contextos” [Alexander]
- Se codifican en formato estructurado que describe el problema y su solución
- “Describen maneras comunes de hacer las cosas, sobre temas que se repiten en los diseños” [Fowler]
- Codifican el conocimiento

Patrones de los principios generales para asignar responsabilidades (GRASP)

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones

Patrones GRASP

- Experto
- Creador
- Alta Cohesion
- Bajo Acoplamiento
- Controlador

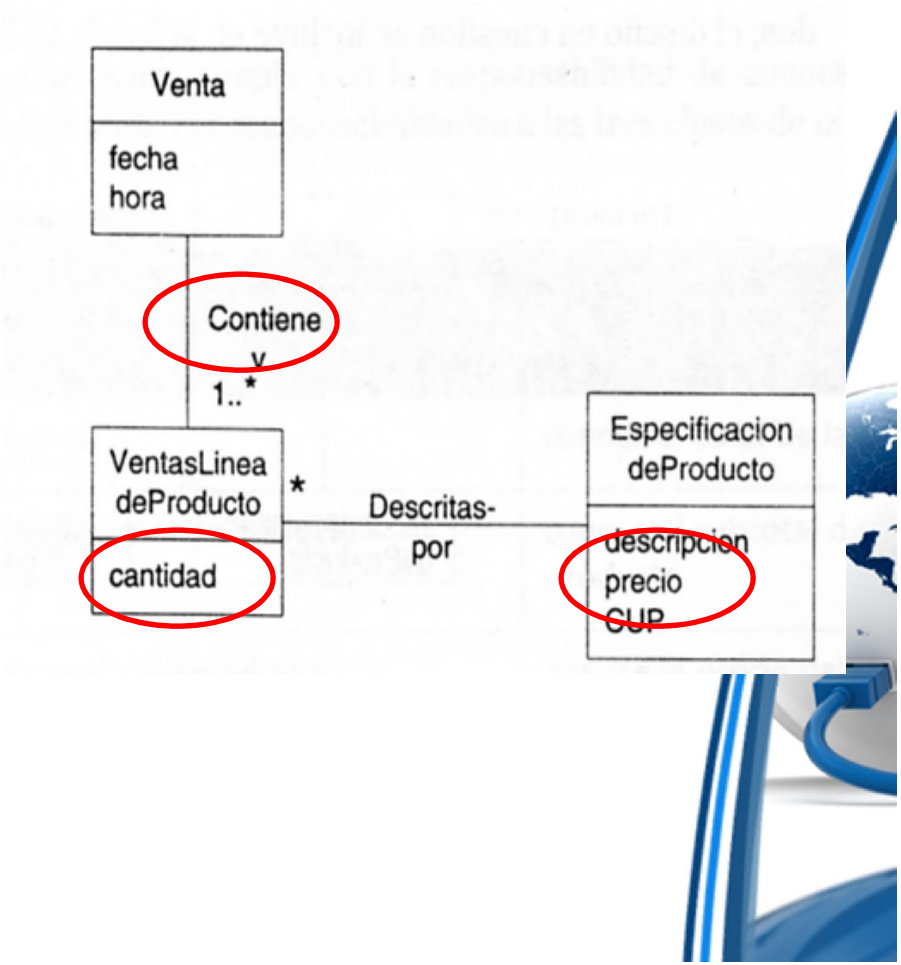
Patrón Experto

Problema	Cuál es un principio fundamental para asignar las responsabilidades en los objetos?
Solución	Asignar la responsabilidad a la clase que cuenta con la información necesaria para cumplir con la responsabilidad

- Utilizado para asignar responsabilidades
- “Intuición” de que los objetos hacen cosas relacionadas con la información que poseen (“Hacerlo Yo mismo”)

Ejemplo: total venta

- POS: quien es el responsable de conocer el gran total de la venta?
- Que información hace falta para calcular el gran total?



Resultado: Calcular Total

Clase	Responsabilidad
Venta	Conoce el total de la venta
VentasLineaProducto	Conoce el subtotal de la línea de producto
DescripcionProducto	Conoce el precio del producto

Beneficios

- Se conserva el encapsulamiento: los objetos se valen de su propia información para hacer lo que se les pide

- Clases “sencillas”: el comportamiento se distribuye

Patrón Creador

Problema	Quién debería ser responsable de crear una nueva instancia de alguna clase?
Solución	Asignarle a la clase B la responsabilidad de crear una instancia de la clase A en uno de los siguientes casos: o B agrega los objetos A o B contiene los objetos A o B registra las instancias de los objetos A o B utiliza estrechamente los objetos de A o B tiene los datos de inicializacion que serán transmitidos a A cuando se crea (Experto)

- Guía la asignación de responsabilidades relacionadas con la creación de objetos

Patrón Creador

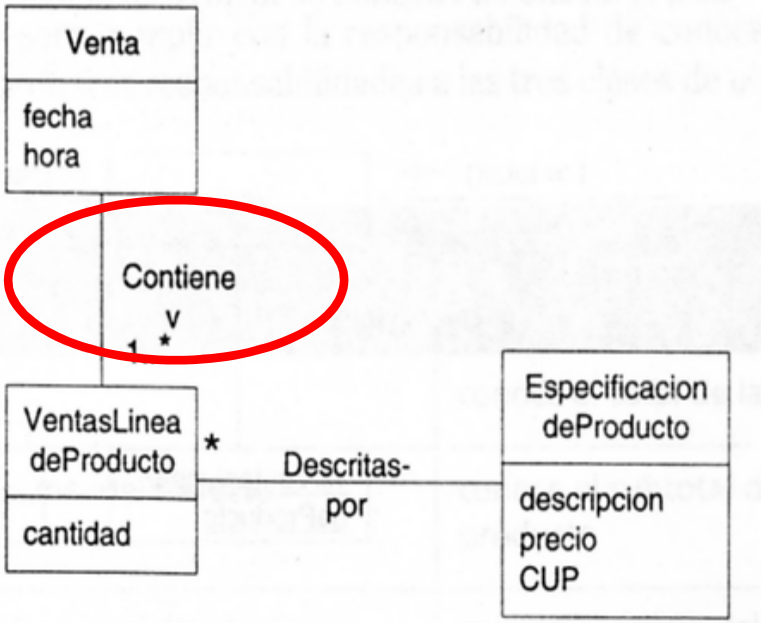
- Tiene como propósito encontrar un creador que debemos conectar con el objeto producido en cualquier evento

- Indica que la clase incluyente del contenedor o registro, es idónea para asumir la responsabilidad de crear la cosa contenida o registrada

Ejemplo: Instancia VentaLineaProducto

- POS: quien debería encargarse de crear una instancia de VentaLineaProducto

- Buscar una clase que agregue, contenga y realice otras operaciones sobre este tipo de instancias



Beneficios

- Se brinda soporte a un bajo acoplamiento

- Menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización

Patrón: Bajo Acoplamiento

- Acoplamiento: Medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas

Problema	Cómo dar soporte a bajas dependencias, bajo impacto en cambio y a un aumento de la reutilización?
Solución	Asignar una responsabilidad para mantener bajo acoplamiento

- Una clase con bajo acoplamiento no depende de muchas otras

Patrón Bajo Acoplamiento

- Alto acoplamiento:
  - Los cambios de las clases afines ocasionan cambios locales
  - Son más difíciles de entender cuando están aisladas
  - Son más difíciles de reutilizar (requieren presencia de otras clases de las que dependen)

Patrón Bajo Acoplamiento

- Patrón evaluativo: Se aplica al juzgar las decisiones de diseño
- Estimula asignar responsabilidades de modo que su colocación no incremente el acoplamiento
- Soporta diseño de clases más independientes
- Se debe buscar un grado moderado de acoplamiento

Ejemplo: Pago

- POS: crear instancia de Pago y asociarla a la Venta que instancia. Qué clase se encargará?
- No incrementar el acoplamiento

Beneficios

- No se afectan por cambios de otros componentes
- Fáciles de entender por separado
- Fáciles de reutilizar

Patrón: Alta Cohesión

- Cohesión: Medida de cuán relacionadas y enfocadas están las responsabilidades de una clase

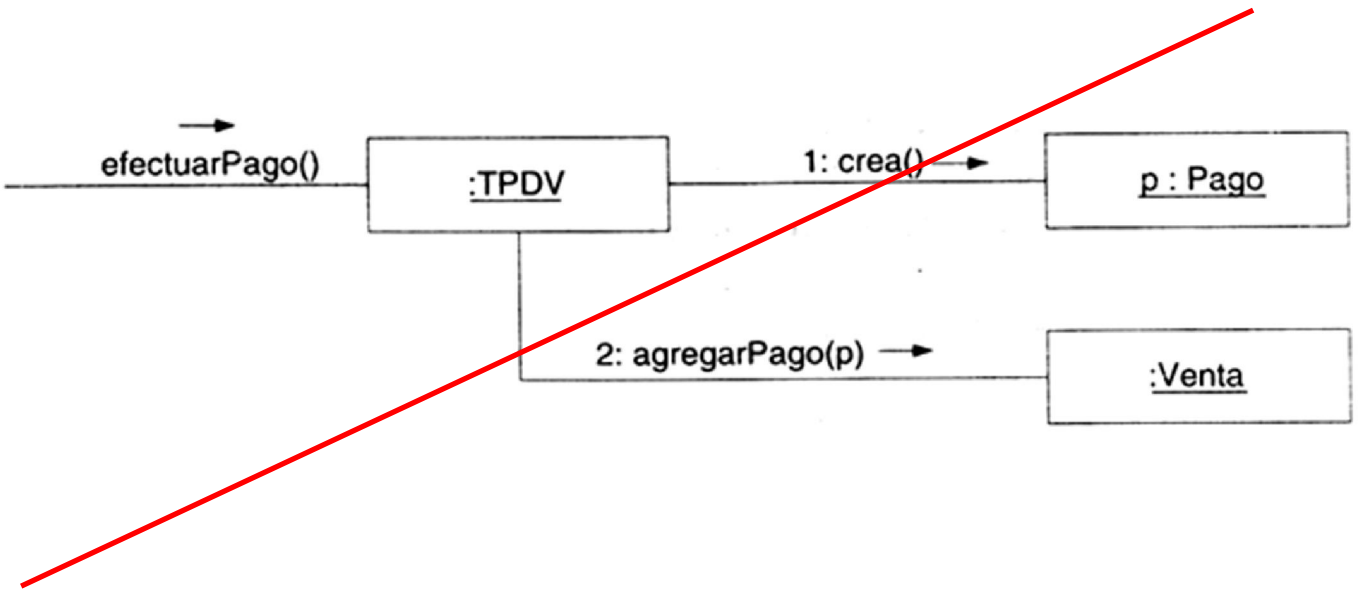
Problema	Cómo mantener la complejidad dentro de límites manejables?
Solución	Asignar una responsabilidad de modo que la cohesión siga siendo alta

Patrón: Alta Cohesión

- Baja cohesión: Hace muchas cosas no afines o un trabajo excesivo. Problemas:
  - Son difíciles de comprender
  - Difíciles de reutilizar
  - Difíciles de mantener
  - Delicadas, las afectan constantemente los cambios
- A menudo son clases que han asumido responsabilidades que deberían haber delegado
- Patrón evaluativo: Se aplica al juzgar las decisiones de diseño
- “Elementos colaboran para producir un comportamiento bien definido” [Booch]

Ejemplo: Pago

- POS: Crear una instancia de Pago y asociarla a la venta. Que clase será la responsable de hacerlo



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 un bajo acoplamiento
- Soporta una mayor capacidad de reutilización, porque puede destinarse a un propósito muy específico

Patrón: Controlador

- Evento de sistema: evento de alto nivel generado por un actor externo (operaciones de sistema)

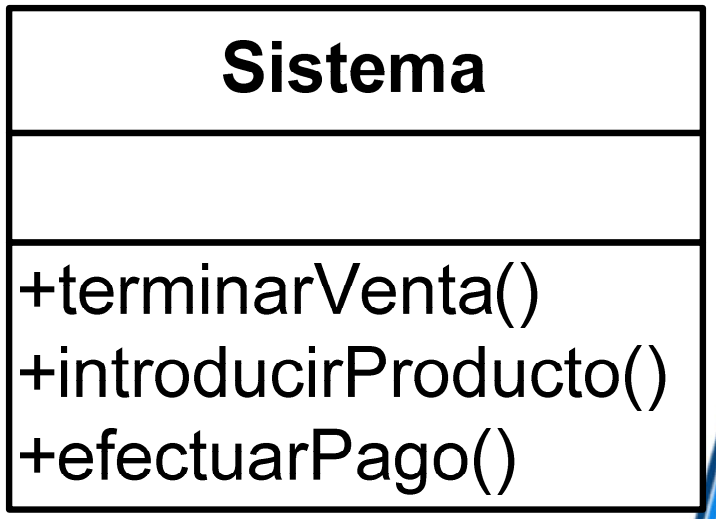
Problema	Quién debería de atender un evento de entrada del sistema?
Solución	Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase: o Representa el sistema global (controlador de fachada) o Un manejador artificial de todos los eventos del sistema de un CU o Algo en el mundo real que es activo (papel de una persona)

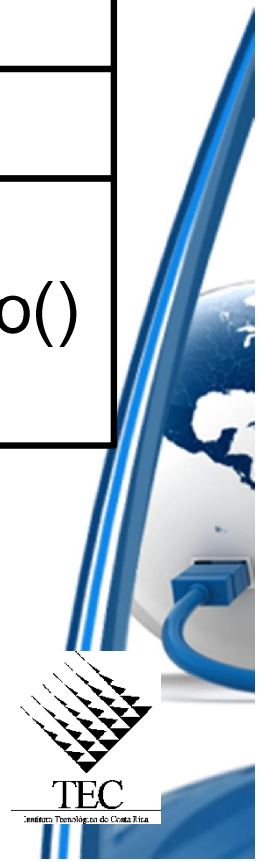
Patrón: Controlador

- Elegir los controladores que manejen los eventos de entrada
- Ofrece una guía para tomar decisiones apropiadas que generalmente se aceptan

Ejemplo: Operaciones POS

- Operaciones del punto de venta
- Quién debería ser el controlador de eventos sistémicos introducirProducto y terminarVenta

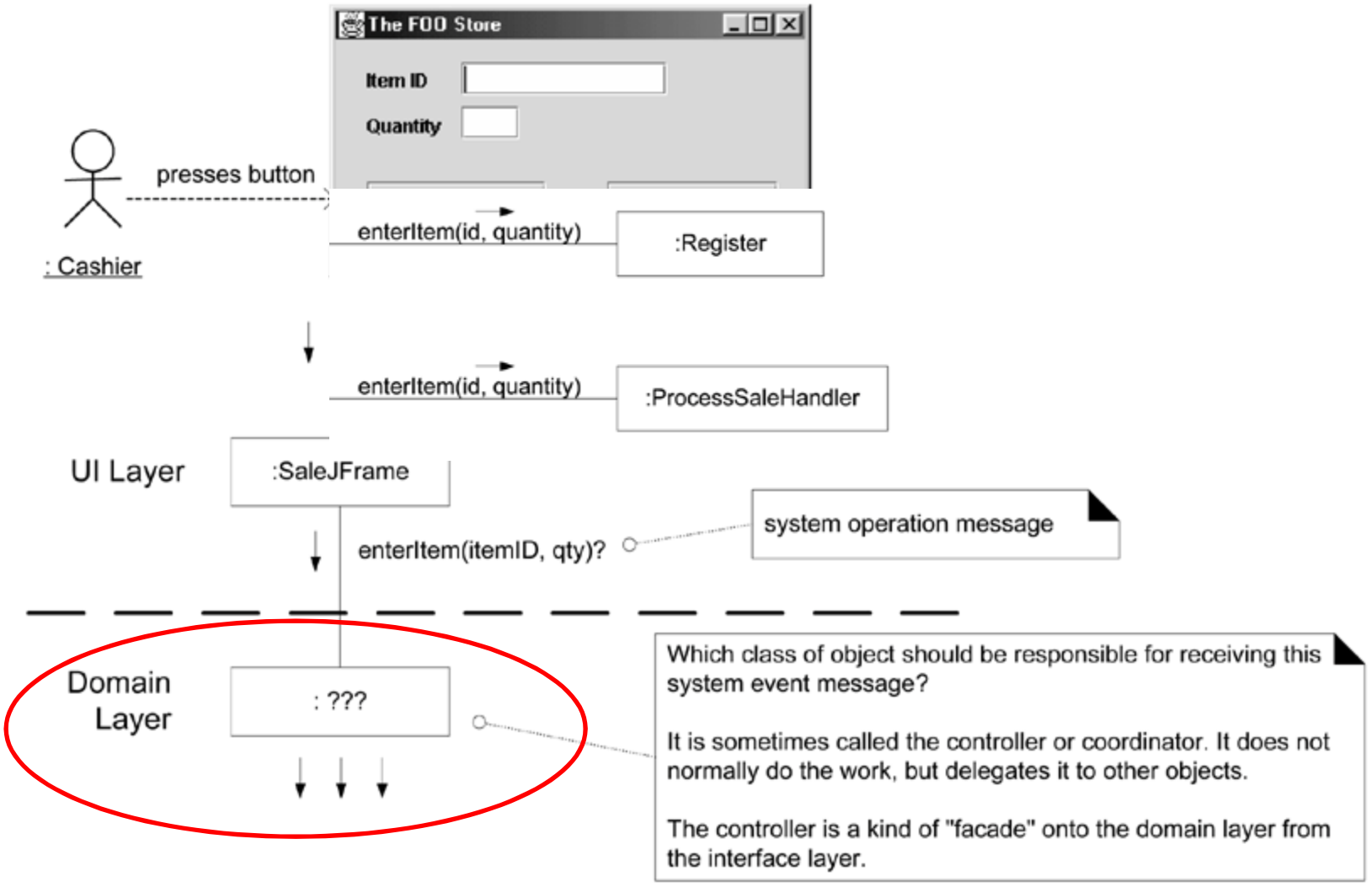




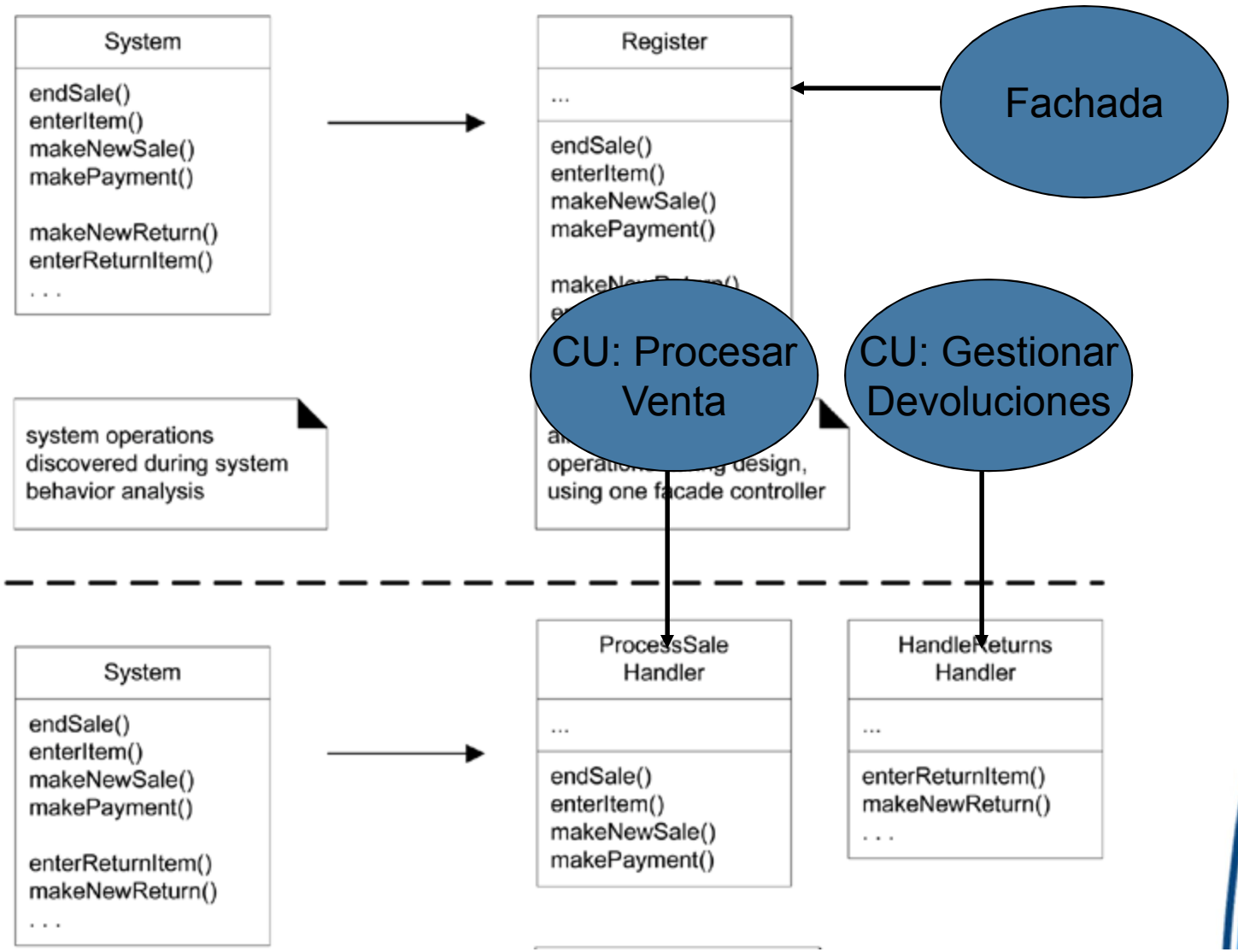
Ejemplo: Operaciones POS

Representa el “sistema” global	TPDV, Registro
Representa un manejador artificial de todas las operaciones del sistema de un CU	ManejadorProcesarVenta
Representa algo en el mundo real que está activo y que puede intervenir en la tarea	Cajero

Ejemplo: Operaciones POS



Ejemplo: Operaciones POS



Beneficios

- Mayor potencial de los componentes reutilizables: Garantiza que los procesos de dominio sean manejados por la capa de los objetos y no por la interfaz
- Reflexionar sobre el estado del caso de uso: Asegurarse que las operaciones del sistema sigan una secuencia y las operaciones en el CU

Controlador saturados

- Controlador dispersa y tiene demasiadas áreas de responsabilidad
- Signos:
  - Hay una sola clase controlador (control de fachada) que recibe todos los eventos del sistema y estos son excesivos.
  - El controlador realiza muchas tareas necesarias para cumplir el evento del sistema, sin delegar trabajo (violación al patrón Experto y alta cohesión)
  - Un controlador posee muchos atributos y conserva información importante sobre el sistema o dominio (debería redistribuirse entre otros objetos)

Posibles Soluciones

- Agregar más controladores: Un sistema no necesariamente debe tener uno solamente
- Diseñar el controlador de modo que delegue fundamentalmente a otros objetos el desempeño de las responsabilidades de la operación del sistema

Patrón: Fabricación Pura

Problema	Quién debería tener la responsabilidad, cuando no se quiere violar los patrones de Alta Cohesión y Bajo Acoplamiento u otras metas, pero la solución ofrecida por el patrón Experto no es apropiada?
Solución	Asignar un conjunto altamente cohesivo de responsabilidades a una clase **artificial** que no representa nada en el dominio del problema: Una cosa inventada para dar soporte a una alta cohesión, bajo acoplamiento y reutilización

- Para utilizar una Fabricación Pura debe buscarse ante todo un gran potencial de reutilización, asegurándose que sus responsabilidades sean pequeñas y cohesivas

Ejemplo: Almacenar Venta en BD

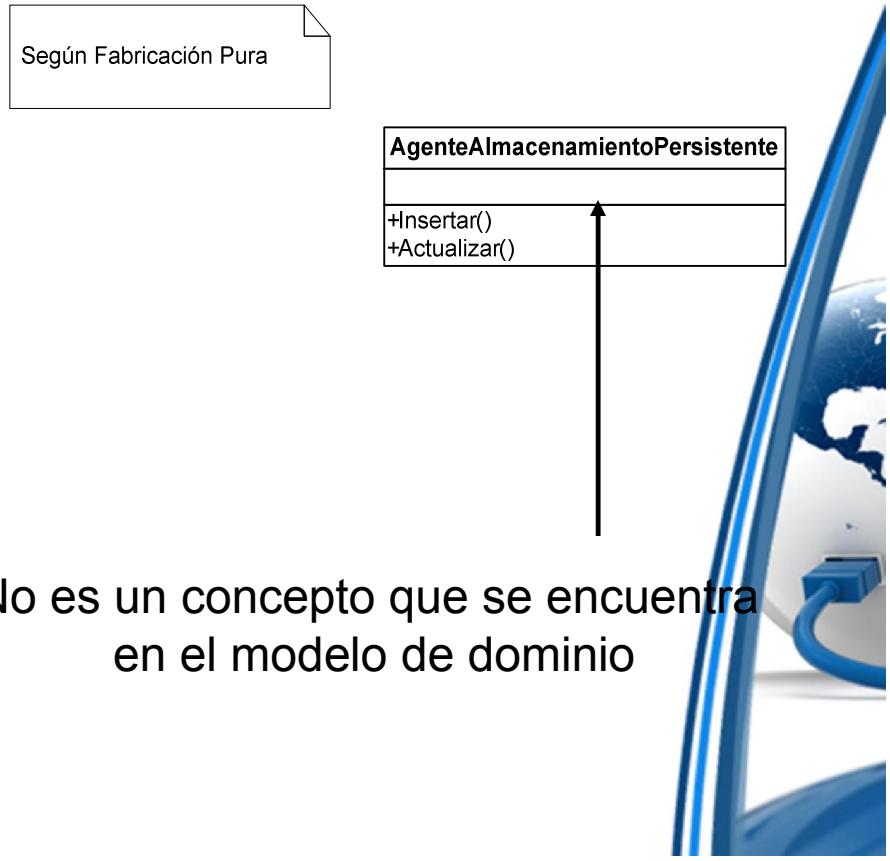
- Que implicaría usar el patrón experto?
  - POS: necesita guardar las instancias Venta en una Base de Datos (concepto no relacionado con vender)
  - Guardar Objetos en una Base de Datos es una tarea muy general
  - Implica duplicación y poca reutilización
  - Se debe acoplar la clase a una interfaz de base de datos

Ejemplo: Almacenar Venta en BD

- Crear una clase nueva (artificial) que se encargue sólo de guardar los objetos en la Base de Datos

Ejemplo: Venta

- Venta conserva su buen diseño, con alta cohesion y bajo acoplamiento
- AgenteAlmacenamientoPersistente es relativamente cohesiva (tiene un único propósito) y es altamente reutilizable



Beneficios

- Se brinda soporte a una Alta cohesion porque las responsabilidades se dividen en una clase de granularidad fina que se centra exclusivamente en un conjunto especifico de tareas
- Puede aunmentar el potencial de reutilizacion

Patrones GoF

- Pandilla de los Cuatro (Gang of Four)
- Tres Categorias:
  - De creación
  - Estructurales
  - Comportamiento