

Construcción de Software


Capítulo 2

Un Proceso basado en UML

Contenidos

- Introducción
- Modelado del Negocio
- Modelado de Requisitos
- Modelado del Análisis
- Patrones GRASP
- Modelado del Diseño
- Casos Prácticos

Contenidos

- Introducción 
- Modelado del Negocio
- Modelado de Requisitos
- Modelado del Análisis
- Patrones GRASP
- Modelado del Diseño
- Casos prácticos

Un proceso simple

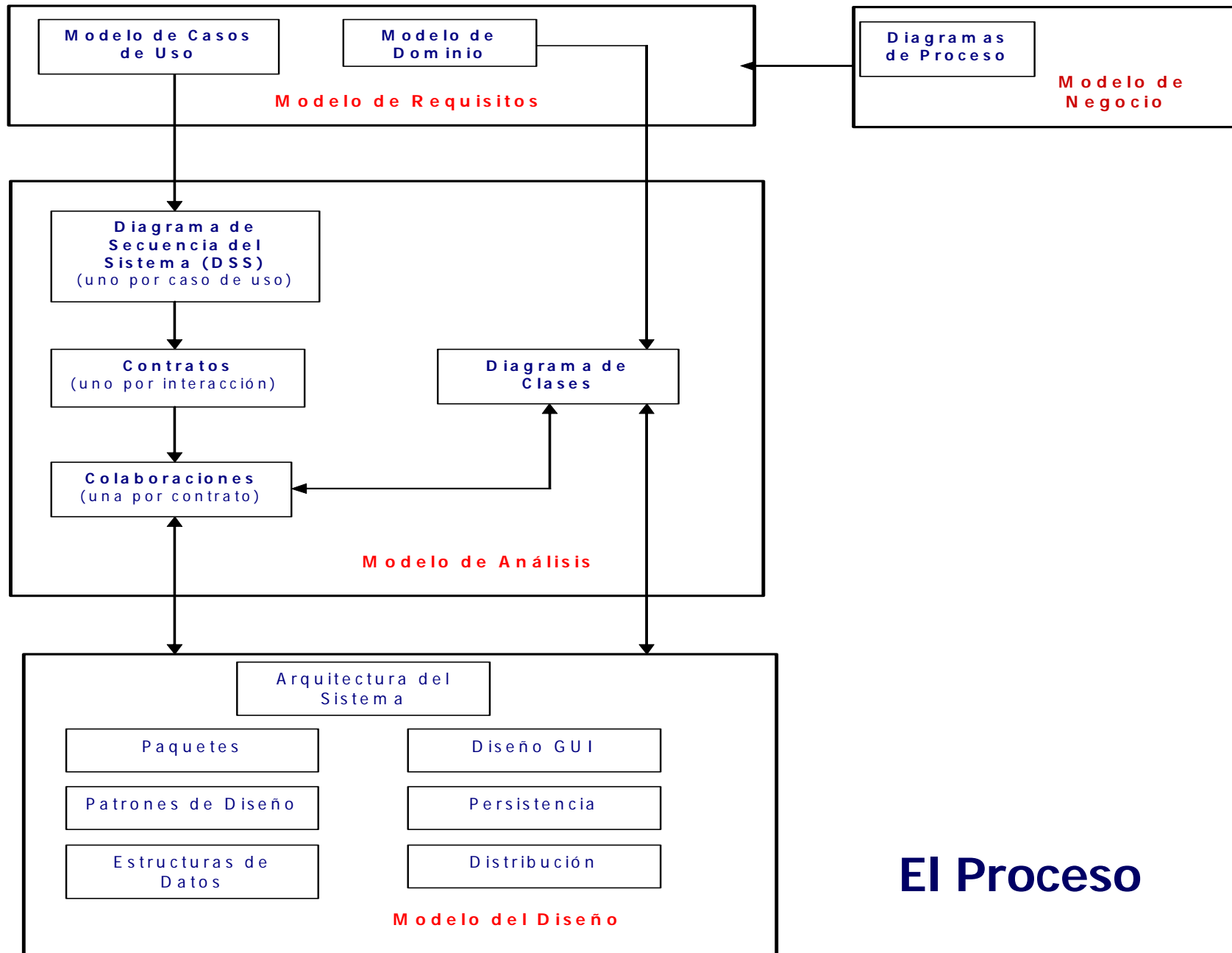
- Descrito en *"UML y Patrones"*, C. Larman, Prentice-Hall, 2002.
- Fácil de aprender y usar.
- No incluye modelado del negocio.
- Conformar con el Proceso Unificado, UP
 - Dirigido por casos de usos.
 - Desarrollo iterativo e incremental
- Conformar con modelado ágil

Etapas del Proceso

- **Inicio**
- Comprender **procesos del negocio** (opcional)
- Obtener y especificar **requisitos** del sistema
- Identificar y especificar clases y colaboraciones para objetos del dominio (**análisis**)
- Resolver problemas de **diseño** (arquitectura, base de datos, redes, patrones, nuevas clases y colaboraciones...)
- **Implementación y pruebas**
- **Validación**

Etapas del Proceso

- Modelado del **Negocio** (opcional)
- Modelado de **Requisitos**
- Modelado del **Análisis**
- Modelado del **Diseño**
- **Implementación**
- **Pruebas**



El Proceso

Inicio


- Estudio de necesidades de la empresa, ver si es viable, alternativas, análisis de riesgos, oportunidad.
- Definición de objetivos del proyecto
- Estimación aproximada del coste
- Duración una o dos semanas

¿Debemos abordar el proyecto?

Inicio

- Primeros talleres de requisitos
- Plan para la primera iteración
- Casos de uso escritos en formato breve, excepto unos pocos que se consideran claves.
- Se identifican riesgos
- Escribir borrador del documento *Visión y Especificación Complementaria*
- Prototipado

Contenidos

- Introducción
- Modelado del Negocio 
- Modelado de Requisitos
- Modelado del Análisis
- Patrones GRASP
- Modelado del Diseño
- Casos prácticos

Modelado del Negocio

Objetivo:

Comprender el conjunto de **procesos de negocio** que tienen lugar dentro de una empresa, como paso previo a establecer los requisitos del sistema a desarrollar.

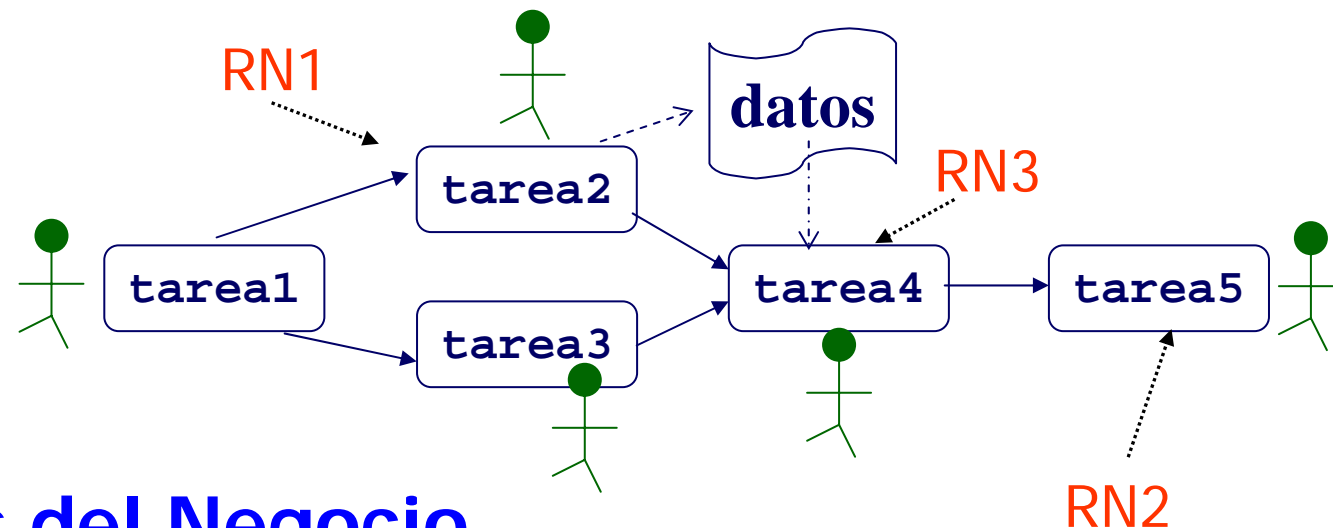
¿**Cómo** consigue la empresa sus objetivos?

Procesos de Negocio

- Una organización tiene una serie de objetivos que satisface a través de ***Procesos de Negocio***
- Elementos de un proceso de negocio:
 - Flujo de Tareas, Agentes, Información y Reglas Negocio
- ***Reglas de Negocio*** regulan el funcionamiento de la empresa
 - Describen restricciones y comportamientos
 - **NO** son requisitos, pero influyen en ellos

Procesos y Reglas del Negocio

Procesos del Negocio

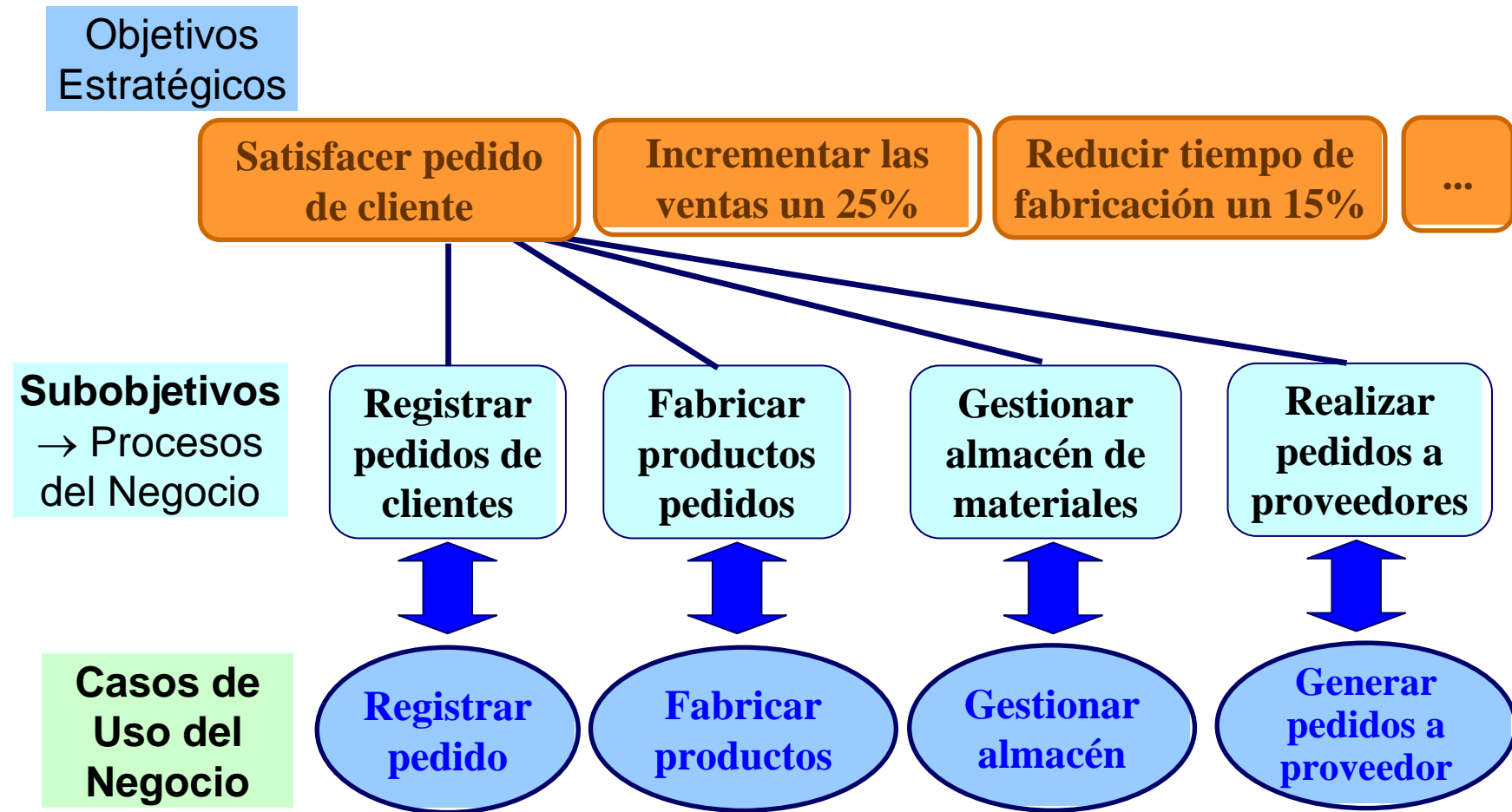


Reglas del Negocio

Determinan **políticas** y **estructura de la información**

Ejemplo

Empresa que fabrica productos bajo demanda



Etapas del modelado del negocio

- Identificar y definir los ***procesos de negocio*** según los ***objetivos*** de la organización.
- Definir un ***caso de uso del negocio*** para cada proceso del negocio (*diagrama de casos de uso del negocio* muestra el contexto y los límites de la organización).
- Identificar los ***roles*** implicados en los diferentes procesos del negocio (*diagrama de roles*).

Etapas del modelado del negocio

- Modelar el flujo de tareas asociado a cada proceso de negocio mediante ***escenarios*** (diagramas de secuencia) y ***diagramas de procesos*** (diagramas de actividades) que muestran la interacción entre roles para conseguir el objetivo.
- Especificar las *informaciones* y *actividades* incluidas en cada diagrama de actividades.

Diagrama *Casos de Uso del Negocio*

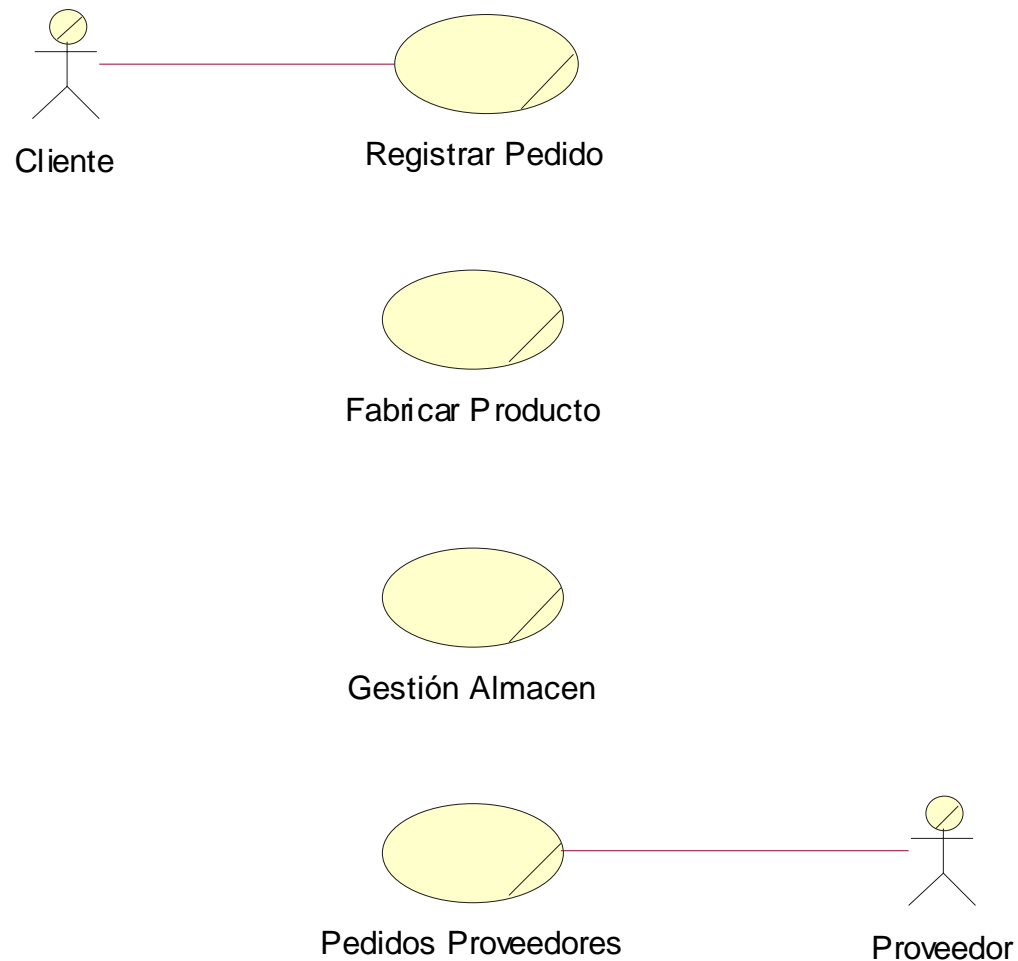
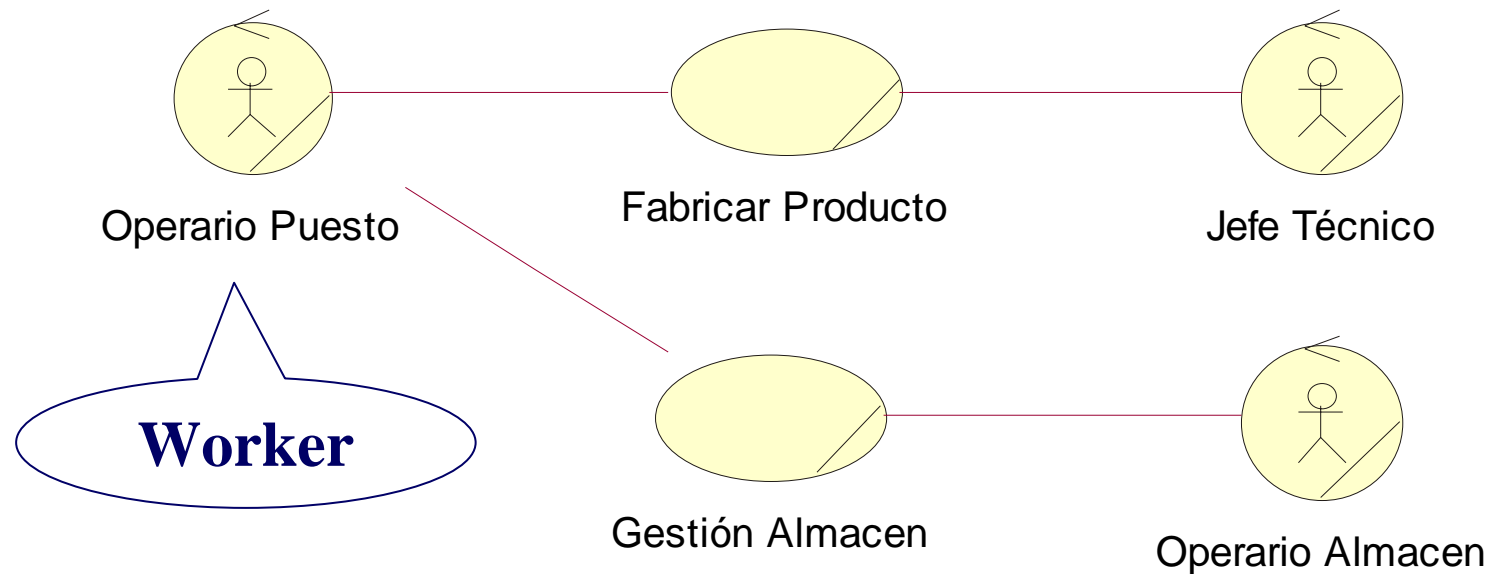


Diagrama *Casos de Uso del Negocio*



Casos de Uso del Negocio

- **Descripción Textual**
 - Plantillas
- **Diagramas**
 - ***Diagrama de roles:***
aspecto estructural de colaboración entre roles
 - ***Escenario:***
aspecto de comportamiento de la colaboración
 - ***Diagrama de Proceso:***
workflow que realiza el caso de uso del negocio

Ejemplo de Caso de Uso del Negocio

“Registrar Pedido”

1. El cliente realiza un pedido que incluirá la fecha del pedido, los datos del cliente y los productos solicitados.
2. El comercial revisa el pedido (completándolo si es necesario) y le da curso, enviándolo al jefe técnico para que realice el análisis del mismo.
3. El jefe técnico analiza la viabilidad de la fabricación de cada producto del pedido por separado.
 - si el producto pedido está en el catálogo, se acepta la fabricación del mismo,
 - en caso contrario, el producto es especial, y el jefe técnico estudia su fabricación
 - si ésta es viable, la fabricación del producto especial es aceptada,
 - si no es viable, el producto no será fabricado.
4. Una vez estudiado el pedido completo, el jefe técnico
 - informa al departamento comercial de la aceptación/rechazo de cada producto integrante del pedido.
 - si todos los productos de un pedido han sido aceptados, genera una orden de trabajo para cada producto, a partir de una plantilla de fabricación (la estándar, si el producto estaba catalogado, o bien una nueva generada para el producto, si éste estaba fuera del catálogo). Cada orden de trabajo es enviada al jefe de producción, y queda pendiente de su lanzamiento.
5. El comercial comunica al cliente el resultado del análisis de su pedido.

Plantilla Caso de Uso del Negocio

Proceso de Negocio	Registrar Pedido
Objetivo	Registrar pedido de un cliente
Descripción	<p>1. El cliente envía una orden de pedido, que debe incluir la cliente y productos solicitados. Es posible que sea un empleado quien introduzca el pedido, a petición de un cliente que realizó su pedido por teléfono o lo envió por fax o correo ordinario al dpto. comercial de la empresa.</p> <p>2. El empleado revisa el pedido (completándolo, si es necesario), y comienza su procesamiento enviándolo al jefe técnico, encargado de su análisis.</p> <p>3. El jefe técnico analiza la viabilidad de cada producto pedido por separado:</p> <ul style="list-style-type: none"> • Si el producto pedido está en el catálogo, su fabricación • En caso contrario es considerado un <i>producto especial</i> <ul style="list-style-type: none"> - Si es viable, la fabricación del producto especial es aceptada; - Si no es viable, el producto especial no será fabricado. <p>4. Una vez estudiado el pedido completo, el jefe técnico...</p> <ul style="list-style-type: none"> • Informa al depto comercial de la aceptación o rechazo de cada producto pedido; • Si todos los productos de un pedido han sido aceptados, se crea una orden de trabajo para cada producto, a partir de una plantilla de fabricación (la estándar si el producto estaba catalogado, o una nueva, específicamente diseñada para el producto, si éste no estaba en el catálogo). Cada orden de trabajo es enviada al jefe de producción y queda pendiente de su lanzamiento. <p>5. El comercial comunica al cliente el resultado final del análisis de su pedido.</p>
Prioridad	Básico
Riesgos	...
Posibilidades	...
Tiempo Ejec.	...
Coste Ejec.	...

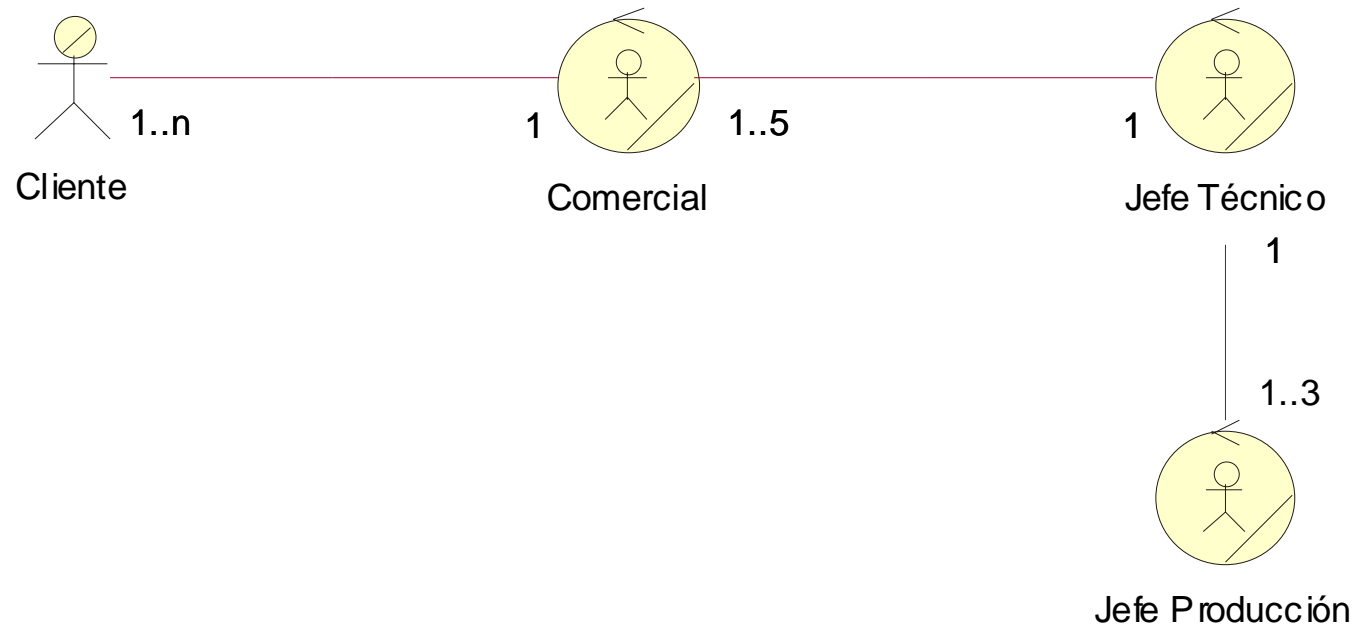
Rol Externo

Roles Internos

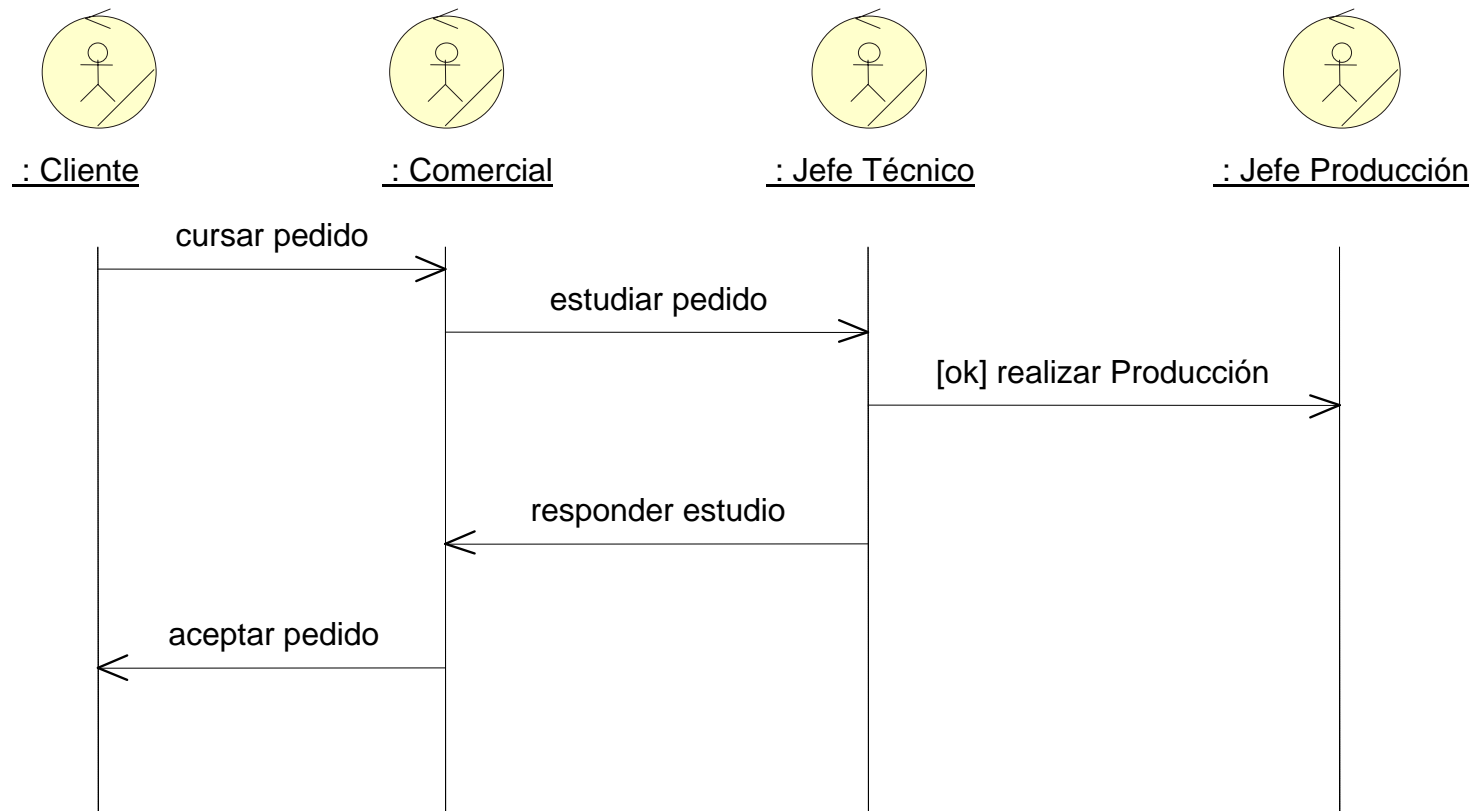
Modelado del negocio

- Identificamos los **agentes o roles** participantes (En el ejemplo: *Cliente, Comercial, Jefe Técnico y Jefe Producción*)
- Creamos **escenarios** para mostrar la colaboración entre los agentes, distinguimos entre flujos básicos y alternativos:
 - *Escenarios*: diagramas de secuencia (objetos son roles)

Workers en "Registrar Pedido"



Escenario "Registrar Pedido"



Flujos de actividades

- Mostrar flujo del proceso mediante *diagramas de proceso*
 - diagramas de actividades con calles que corresponden a roles
 - una actividad puede ser compleja para ser descrita en otro diagrama.
 - Incluir sólo informaciones relevantes

Diagrama de Proceso

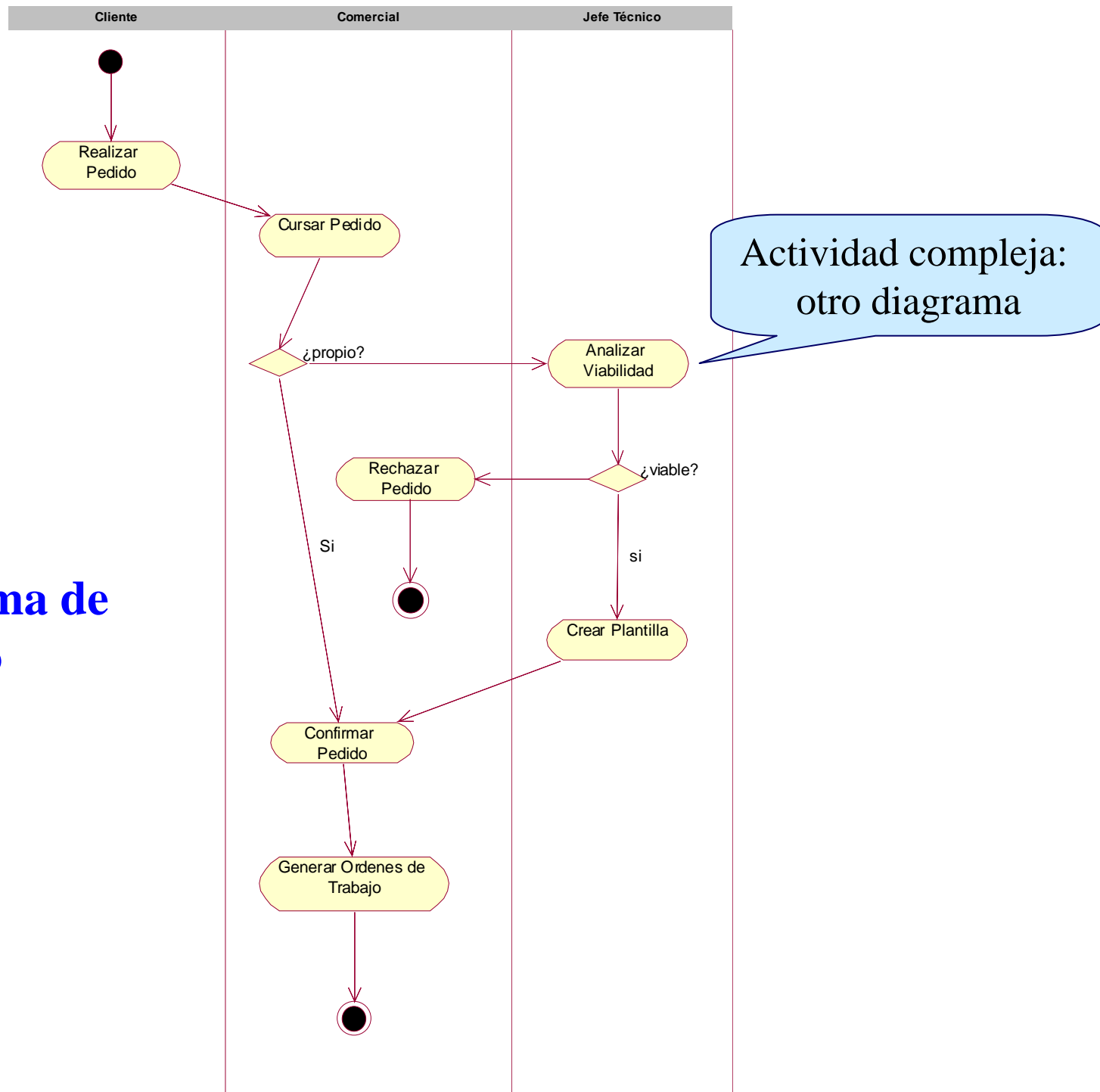
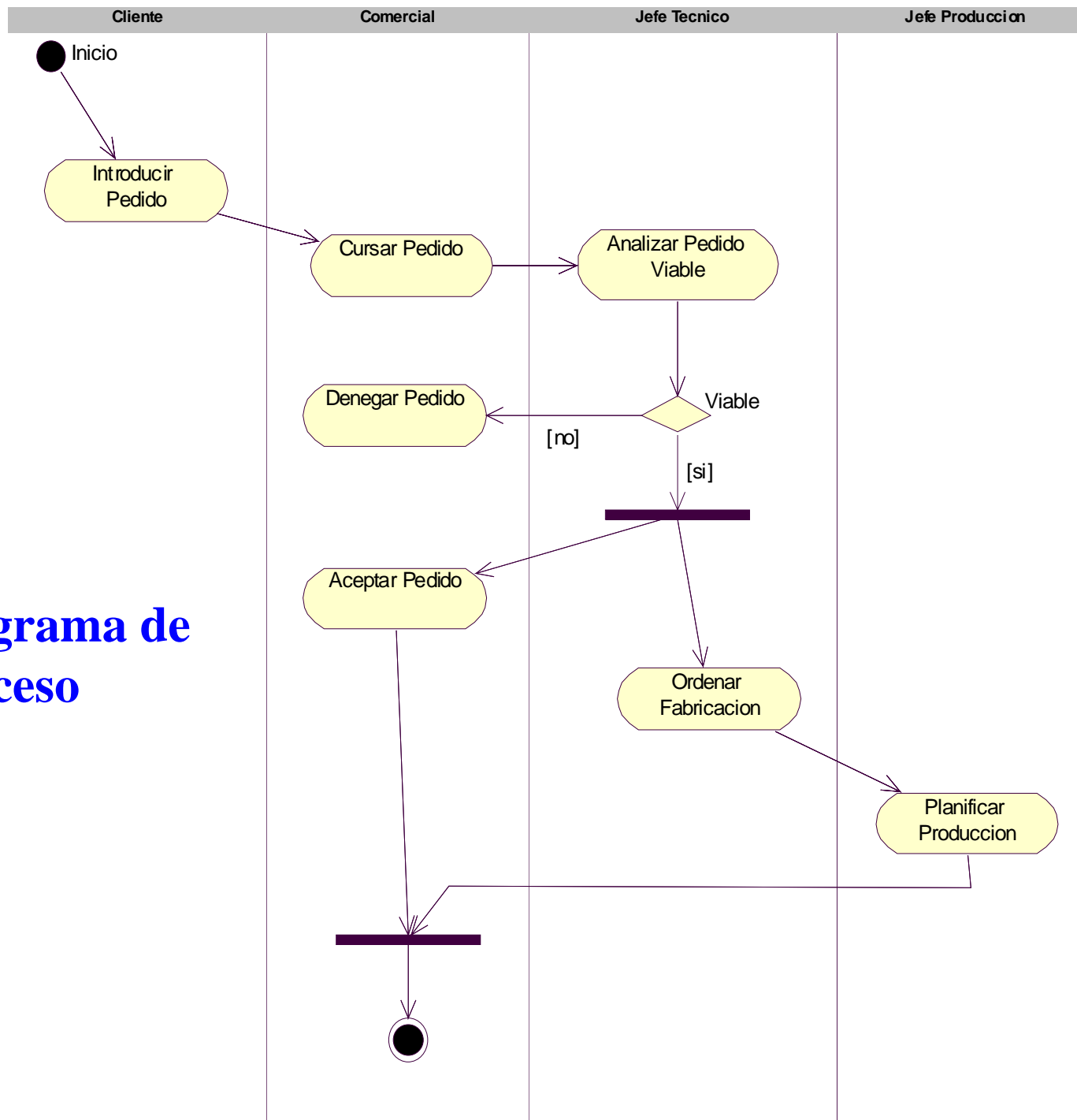


Diagrama de Proceso



Reglas de Negocio

- Reglas de **restricción**
 - Especifican políticas o condiciones que restringen la estructura y comportamiento de las informaciones
 - *Estímulo-Respuesta*
 - *Restricción de operación*
 - *Restricción de estructura*
- Reglas de **derivación**
 - Especifican políticas o condiciones para inferir nuevos hechos a partir de otros.

Glosario

...

Objeto de Información: Pedido

Atributos

Código de pedido
Fecha de solicitud
Fecha límite de entrega
Conjunto de {Producto}
Cliente
Importe total
Estado Actual

Restricciones

- El código de pedido identificará unívocamente el pedido, y será asignado automáticamente por el sistema
- La fecha de solicitud será anterior a la fecha límite de entrega.
- Un pedido contendrá al menos un producto; no existe límite máximo de productos.
- Un pedido siempre será solicitado por uno y solamente un cliente.
- El importe total será calculado a partir del precio de cada producto pedido.

Clase del Dominio : - por especificar -

...

...

Actividad: Ordenar fabricacion

Origen: Analizar viabilidad

Agente: Jefe Tecnico

Precondiciones: La fabricacion de todos los productos pedidos es viable. Existe una plantilla de fabricación para cada uno de dichos productos.

Postcondiciones: Ha sido creada una orden de trabajo para cada producto, con estado *pendiente*, y ha sido enviada al jefe de producción para su planificación.

Caso de Uso : - por especificar-

Actividad: Notificar aceptacion de pedido

Origen: Analizar viabilidad

Agente: Comercial

Precondiciones: La fabricación de todos los productos pedidos es viable.

Postcondiciones: Se ha comunicad al cliente la aceptación de su pedido. El estado del pedido es *aceptado*.

Caso de Uso : - por especificar-

Trazabilidad

Especificación de las *actividades*

Contrato: nombre de la actividad realizada por los actores

Origen: Actividad/es precedente/s

Agente: Actor que realiza la actividad

Precondición: Estado previo a la realización de la actividad

Postcondición: Estado posterior a la realización de la actividad

Caso de uso: Nombre del caso de uso que se corresponde con la actividad. Este campo no se rellenará hasta que no se identifiquen los casos de uso.

Especificación de las *informaciones*


Nombre de la información

Atributos: Listado de los atributos de la información

Restricciones: Restricciones sobre los atributos de la información, referidas tanto al significado como al valor de los mismos.

Clase: Nombre de la clase que modelará esta información. En principio no se indica nada, y sólo se rellena este campo cuando la clase es identificada en el modelado conceptual.

Contenidos

- Introducción
- Modelado del Negocio
- Modelado de Requisitos 
- Modelado del Análisis
- Patrones GRASP
- Modelado del Diseño
- Casos prácticos

Modelado de Requisitos

Objetivo:

Se establecen los **requisitos funcionales** y **no funcionales** del sistema.

A partir del modelo del negocio (si se hace) se construye el ***modelo de casos de uso*** y el ***modelo conceptual***.

Tipos de Requisitos

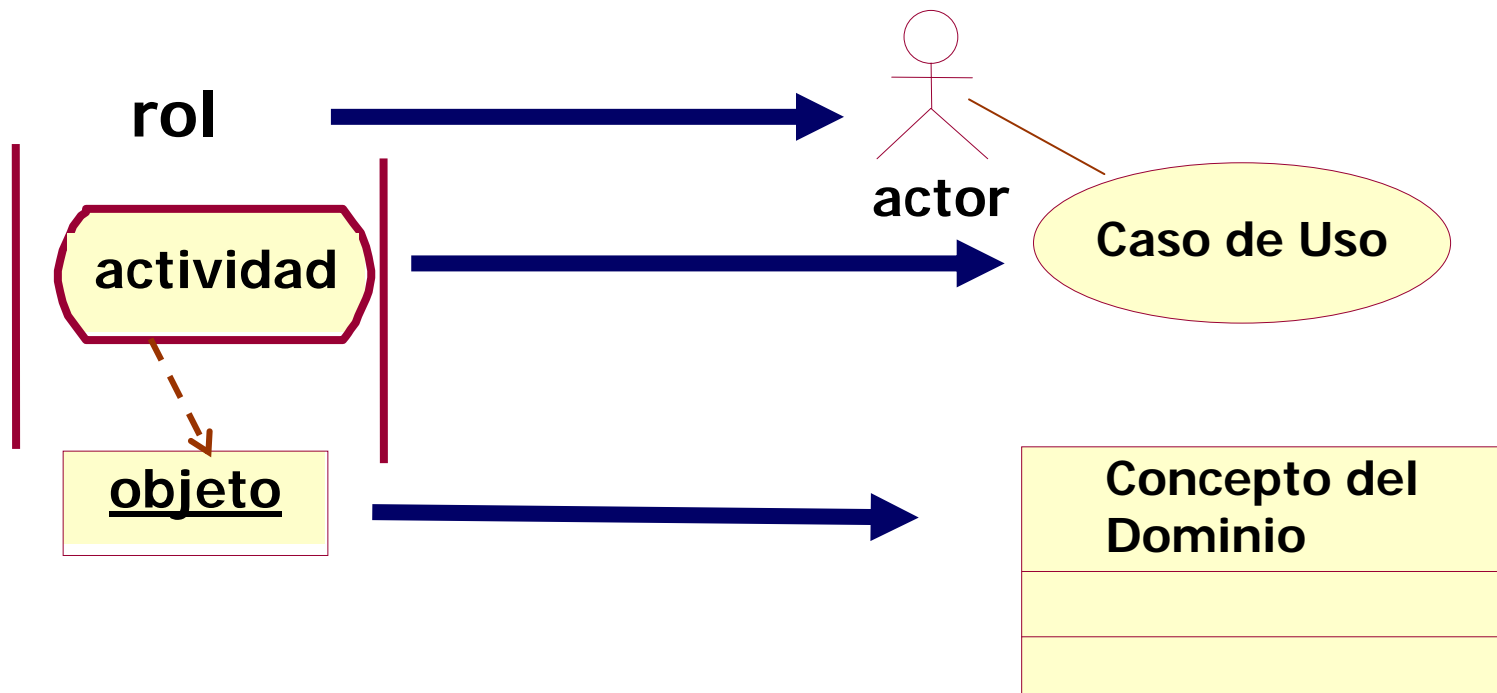
- **Funcionales**
- **No Funcionales**
 - Usabilidad
 - Fiabilidad
 - Rendimiento
 - Adaptabilidad, Mantenimiento, Configurable
 - Implementación: lenguajes, herramientas,...
 - GUI
 - Legales

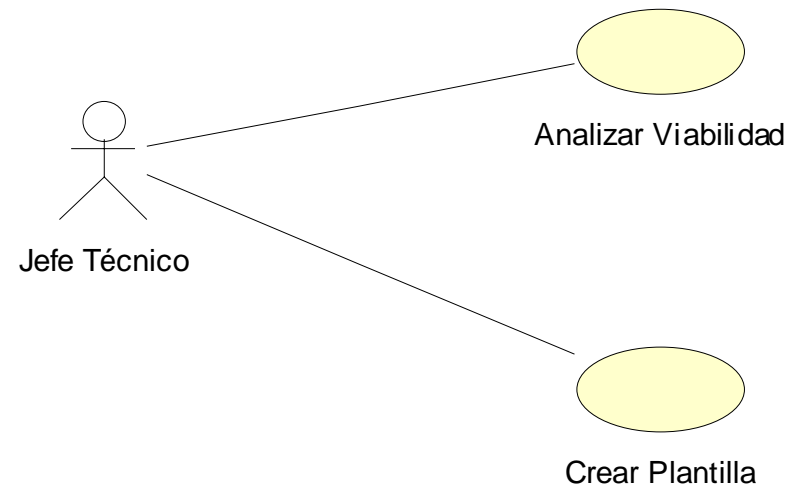
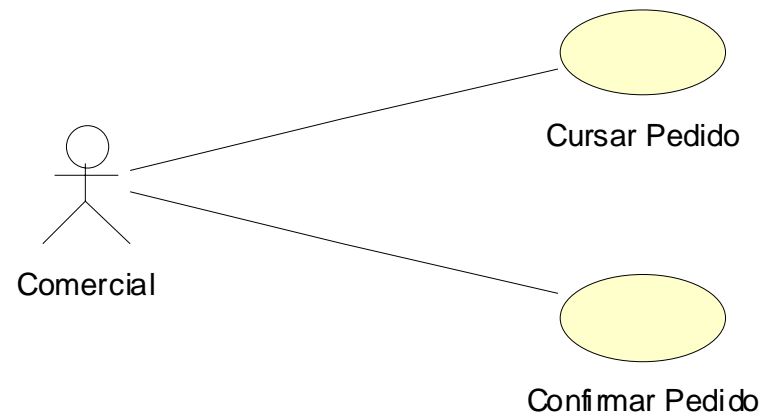
Del modelo de negocio al modelo de requisitos

- Extraer los ***casos de uso del sistema*** a partir de las actividades que aparecen en los diagramas de actividades.
- Establecer el ***modelo conceptual*** a partir de las informaciones incluidas en los diagramas de actividades.

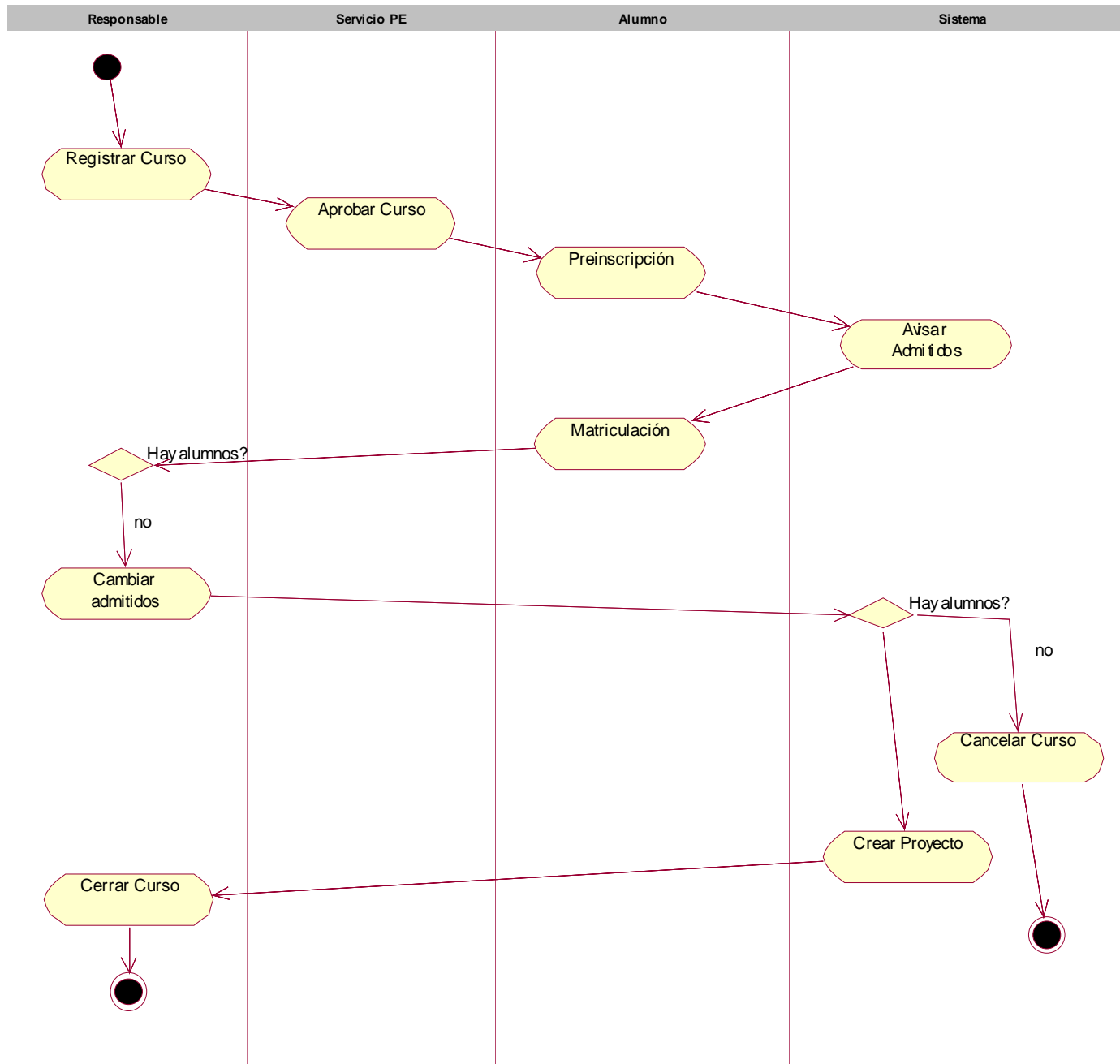
Del modelo de negocio al modelo de requisitos

- De los **diagramas de proceso...**



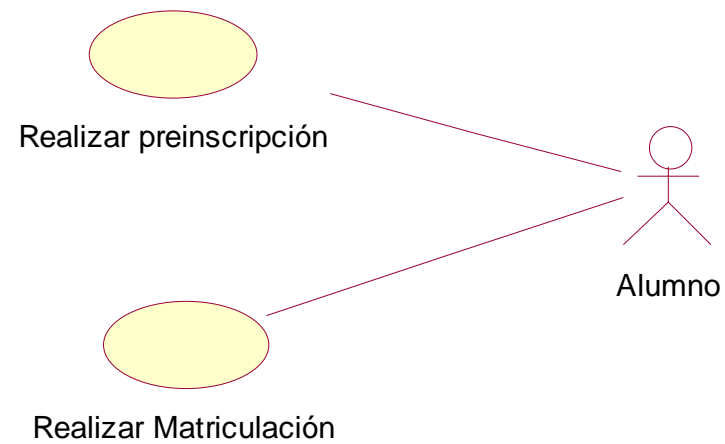


Casos de Uso para "Registrar Pedido"



Gestión Cursos PE

```
graph LR;
  Responsable((Responsable)) --- Registrar[Registrar curso];
  Responsable --- Rebajar[Rebajar Cupo];
  Responsable --- CerrarCurso[Cerrar curso];
  Sistema((Sistema)) --- CerrarPre[Cerrar Preinscripción];
  Sistema --- CerrarMat[Cerrar Matriculación];
  Sistema --- Cancelar[Cancelar curso];
```



Identificación de casos de uso

- *Objetivos Estratégicos* → casos de uso del negocio
 - Ejemplo: Evitar pérdidas
- *Objetivos de Usuario* → casos de uso
 - Ejemplo: Realizar Venta
- *Subfunciones* → ¿casos de uso?
 - Ejemplo: Iniciar sesión (*Login*)

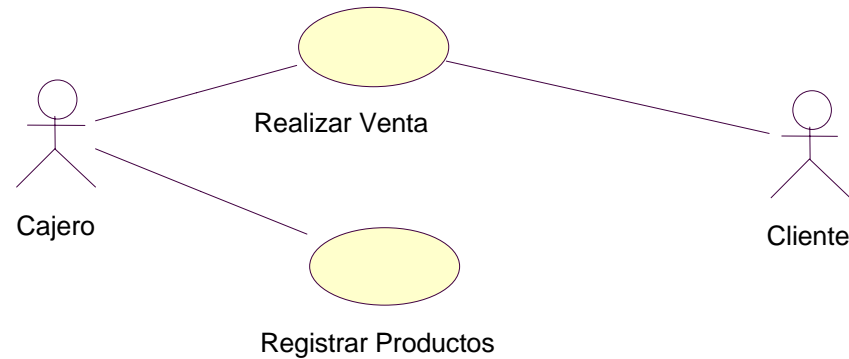
Identificación de casos de uso

- Establecer los límites del sistema
- Identificar los **actores** principales
 - ¿Es el *cliente* un actor en el sistema TPV?
- Identificar **sus objetivos** de usuario
 - Posible uso de los eventos externos
- **Definir un caso de uso por objetivo de usuario**
 - Excepción: casos de uso para manejar información (crear, eliminar, modificar, consultar)
- Formato expandido y esencial

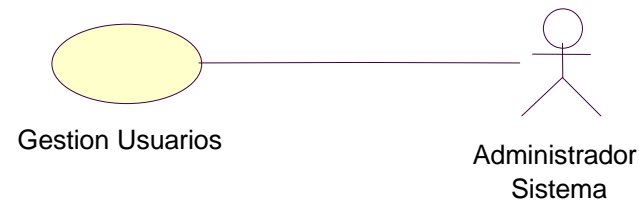
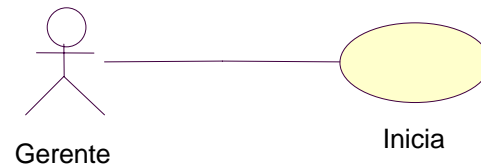
Plantilla *usecases.org*

- Actor Principal
- Personas involucradas e Intereses
- Precondiciones
- Postcondiciones
- Escenario Principal (Flujo Básico)
- Extensiones (Flujos Alternativos)
- Requisitos especiales
- Tecnología y Lista Variaciones de datos
- Frecuencia
- Cuestiones abiertas

Ejemplo: Terminal Punto de Venta



Casos de Uso



Caso de uso *“Realizar Venta”*

Resumen: Un cliente llega al TPV con un conjunto de artículos. El Cajero registra los artículos y se genera un ticket. El cliente paga en efectivo y recoge los artículos.

Actor Principal: Cajero

Personal Involucrado e Intereses:

- Cajero: quiere entradas precisas, rápida y sin errores de pago
- Compañía: quiere registrar transacciones y satisfacer clientes.
- ...

- **Precondición:** El cajero se identifica y autentica
- **Postcondiciones:** Se registra la venta. Se calcula el impuesto. Se actualiza contabilidad e inventario...

Caso de uso *“Realizar Venta”*

Flujo Básico:

1. A: El cliente llega al TPV con los artículos.
2. A: El cajero inicia una nueva venta
3. A: El cajero introduce el identificador de cada artículo.
4. S: El sistema registra la línea de venta y presenta descripción del artículo, precio y suma parcial.

El Cajero repite los pasos 3 y 4 hasta que se indique.

5. S: El Sistema presenta el total
6. A: El Cajero le dice al Cliente el total a pagar
7. S: El Cliente paga y el sistema gestiona el pago.
8. S: El Sistema registra la venta completa y actualiza Inventario.
9. S: El Sistema presenta recibo

Caso de uso *“Realizar Venta”*

Extensiones (Flujos Alternativos):

3a. Identificador no válido

1. El Sistema señala el error y rechaza la entrada

3-6a. El Cliente pide eliminar un artículo de la compra

1. El Cajero introduce identificador a eliminar
2. El sistema actualiza la suma

...

7a. Pago en efectivo

1. El Cajero introduce cantidad entregada por el cliente
2. El Sistema muestra cantidad a devolver

...

....

Caso de uso *“Realizar Venta”*

Requisitos especiales:

- Interfaz de usuario con pantalla táctil en un monitor de pantalla plana. El texto debe ser visible a un metro de distancia.
- Tiempo de respuesta para autorización de crédito de 30 sg. el 90% de las veces

...

Lista de Tecnología y Variaciones de Datos:

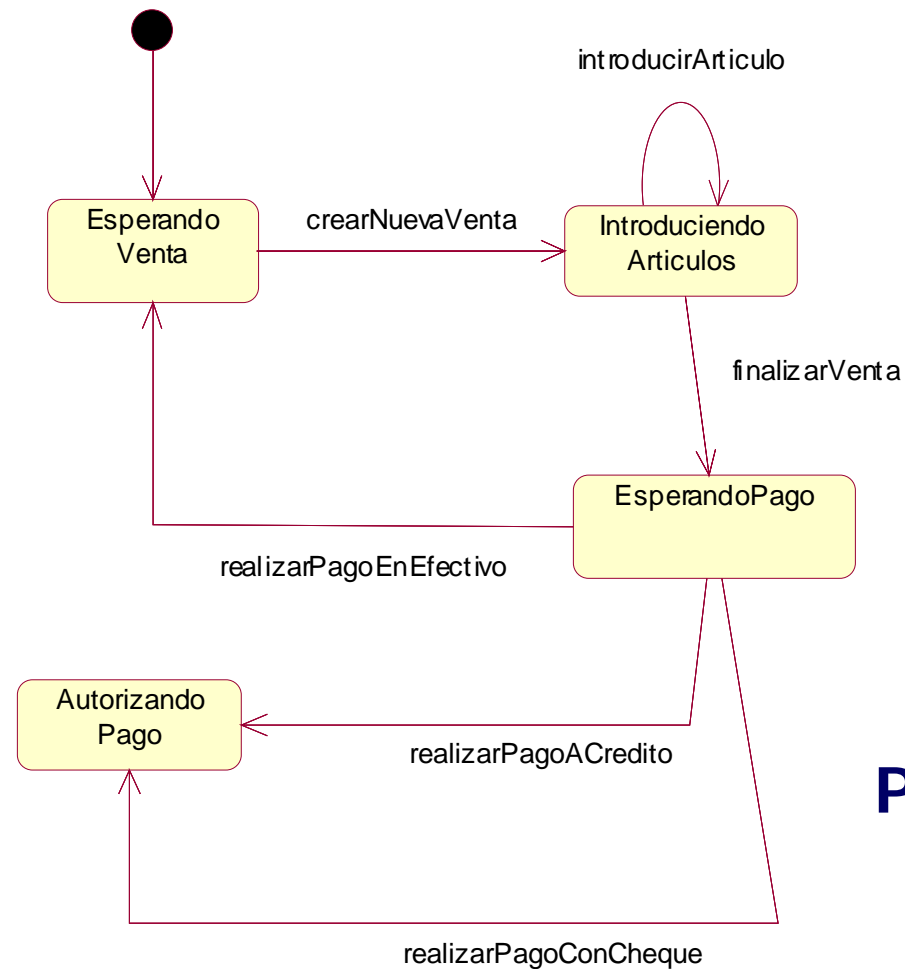
- El identificador podría ser cualquier esquema de código UPC, EAN,...
- La entrada de información de la tarjeta se realiza mediante un lector de tarjetas.

...

Cuestiones Pendientes:

- Explorar cuestiones de recuperación de accesos a servicios remotos
- ¿Qué adaptaciones son necesarias para diferentes negocios?

Diagramas de estado de casos de uso



Procesar Venta

Elaboración de casos de uso

- ¿Cuándo?
 - Al principio de la iteración en formato extendido y esencial, se refinan a lo largo de las iteraciones
- ¿Dónde?
 - En talleres de captura de requisitos
- ¿Quién?
 - Usuarios finales, clientes, desarrolladores
- ¿Cómo? (Herramientas)
 - Herramientas de requisitos *web-enabled* integradas con un procesador de texto (texto cdu) y herramientas CASE UML (diagramas de cdu)

Recomendaciones sobre casos de uso

- Define bien los límites del sistema.
- Los actores interaccionan con el sistema.
- Pregunta qué quieren los actores del sistema:
Objetivos.
- Distingue el flujo normal de los flujos alternativos.
- Lo importante es escribir la descripción del caso de uso, no dibujar diagramas de casos de uso.
- Uso limitado de las relaciones *include* y *extend*

Recomendaciones sobre casos de uso

- Usa *include* para factorizar parte de un flujo que aparece en varios casos de uso.
- Las extensiones pueden incluirse dentro del caso de uso base como flujos alternativos en vez de incluir casos de uso aparte.
- El propio sistema puede disparar casos de uso.
- No incluir casos de uso CRUD; casos de uso *Crear* y *Consulta* si son relevantes.
- No especificar casos de uso que incluyan detalles de diseño de interfaces de usuario.

Modelo Conceptual (o Dominio)

- Representa el **vocabulario** del dominio: ideas, conceptos, objetos
- No son modelos de elementos software
- *Clases conceptuales*
- Clases no incluyen operaciones, sólo atributos
- Atributos: números o texto
- Asociaciones entre clases

Identificar Clases Conceptuales

- Si se hace modelado del negocio:
 - “Los objetos información, entrada y salida de las actividades de los *diagrama de proceso*, representan entidades y conceptos del dominio”.
- Una *clase conceptual* para cada información relevante.
- De la especificación del diccionario se extraen:
 - atributos, asociaciones, restricciones
- Se refina a lo largo de las iteraciones
- “Más vale que sobren clases que falten”

Del Modelo del Negocio al Modelo Conceptual

Objeto información ***"Pedido"** (modelo del negocio)*

***Atributos:** codigo, importe, estado, fechaTopeEntrega,..*

***Asociaciones:** Pedido-Cliente y Pedido-Producto,..*

***Restricciones:** fechaTopeEntrega > fechaRecepcion, ..*

- También es posible identificar objetos que pasan por varios estados (*diagrama de estados preliminar*)

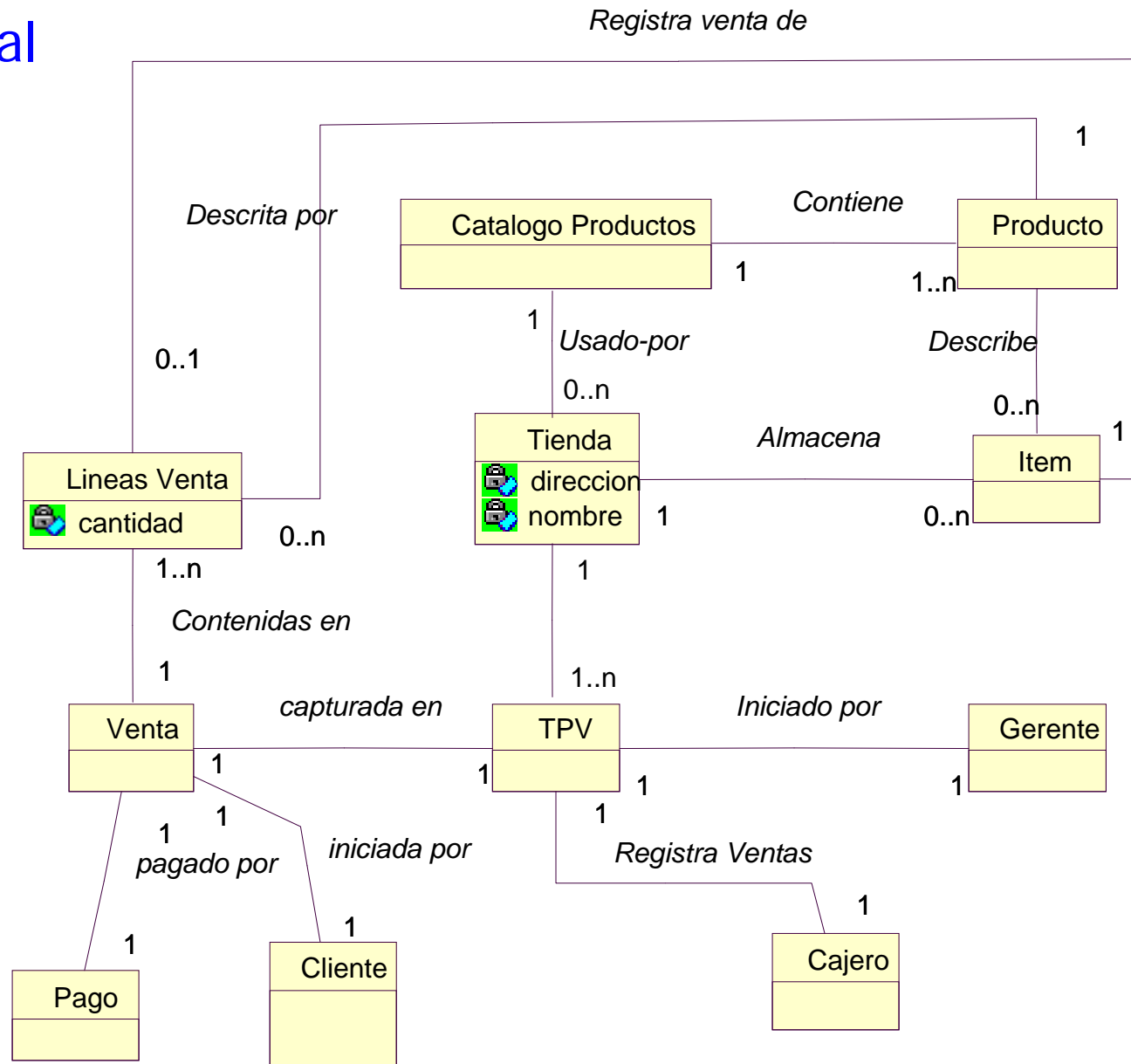
Identificar Clases Conceptuales

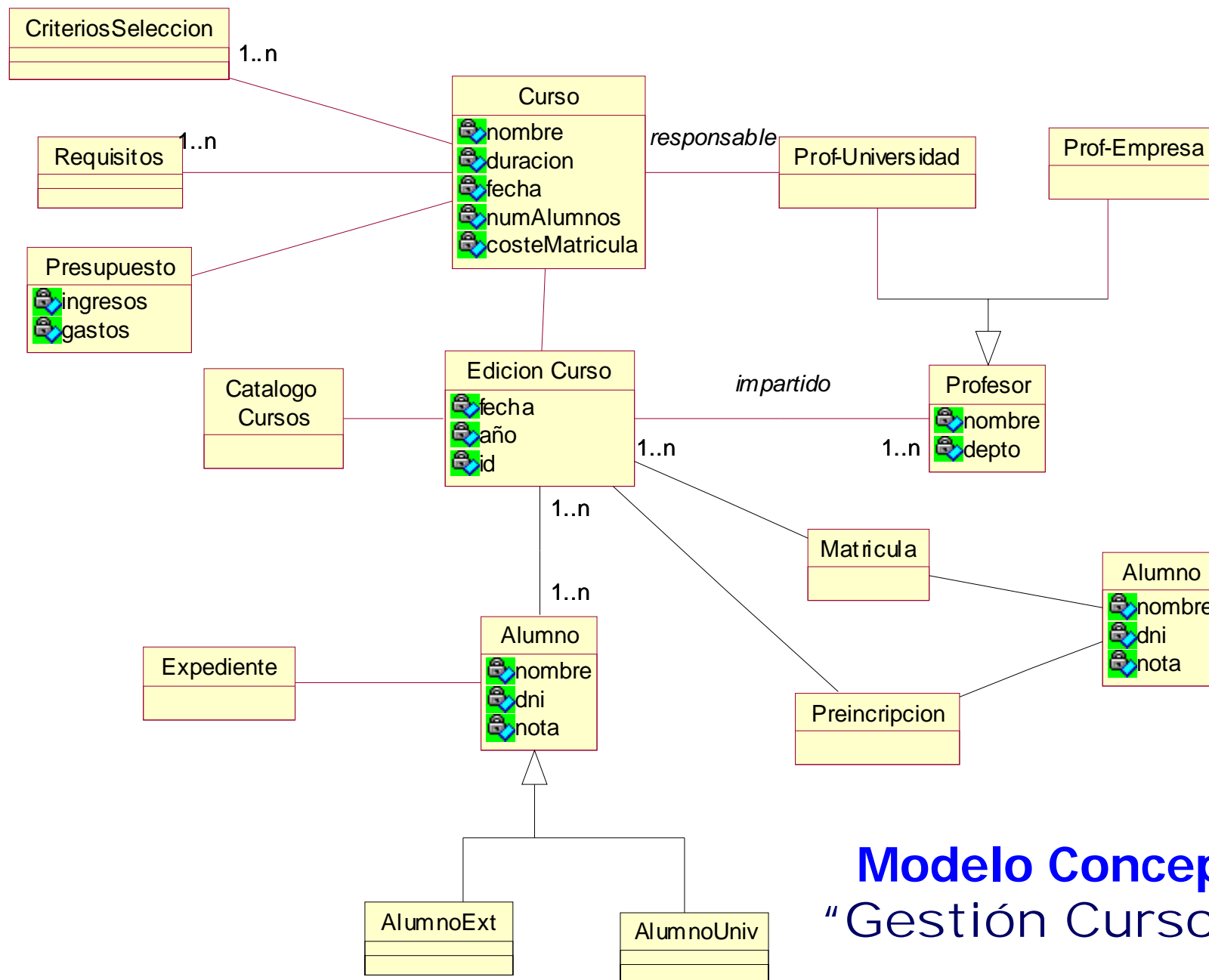
- Si no se hace modelado del negocio:
 - Usar una lista de categorías de clases
 - Identificar nombres de las frases
- Categorías de clases
 - Objetos físicos
 - Especificaciones o descripciones de cosas
 - Lugares
 - Transacciones
 - Líneas de una transacción

Identificar Clases Conceptuales

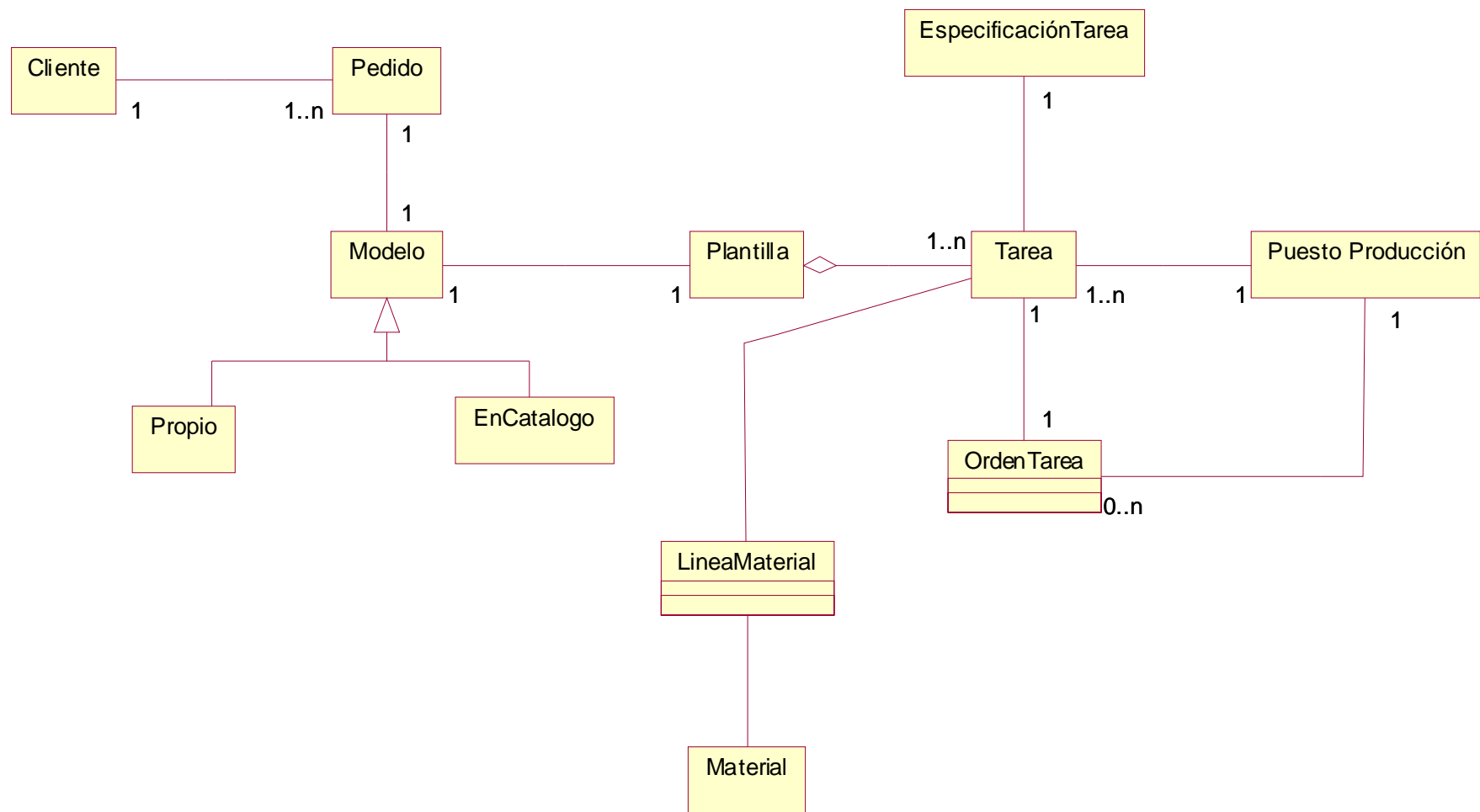
- Categorías de clases (continua)
 - Contenedores de cosas
 - Cosas en un contenedor
 - Dispositivos externos al sistema
 - Conceptos abstractos
 - Eventos
 - Procesos
 - Reglas y Políticas
 - Catálogos
 - Contratos, documentos legales
 - Instrumentos y servicios financieros
 - Manuales, documentos, trabajos, libros

Modelo Conceptual TPV





Modelo Conceptual "Gestión Cursos PE"



Modelo conceptual "Fabricación bajo demanda"

Identificar Asociaciones

- Se incluyen aquellas:
 - para la que el **conocimiento** de la relación deba mantenerse algún tiempo
 - derivadas de la *Lista de Asociaciones*
- *Lista de Asociaciones*
 - A es parte física de B
 - A es parte lógica de B
 - A está físicamente contenida en B
 - A está lógicamente contenida en B
 - A es una descripción de B

Identificar Asociaciones

- Lista de Asociaciones (continua)
 - A es una línea de una transacción o informe B
 - A es conocido/registrado/informado/capturado en B
 - A es miembro de B
 - A es una subunidad organizacional de B
 - A usa o maneja B
 - A comunica con B
 - A está relacionado a una transacción B
 - A es el siguiente a B
 - A es propiedad de B
 - A es un evento relacionado con B

Identificar Asociaciones

“Encontrar clases conceptuales es más importante que encontrar asociaciones. Se debe dedicar más tiempo a encontrar clases conceptuales que asociaciones”

- Centrarse en las asociaciones “necesita-conocer”
- Muchas asociaciones dificultan la comprensión de los diagramas
- Evitar asociaciones redundantes
- En la implementación se notará que falta o sobra alguna asociación

Asociaciones en TPV

- A es parte física de B
 - TPV-Caja
- A es parte lógica de B
 - LineaVenta-Venta
- A está físicamente contenida en B
 - Item-Tienda, TPV-Tienda
- A está lógicamente contenida en B
 - EspecificacionProducto-CatalogoProductos
- A es una descripción de B
 - EspecificacionProducto-Item

Asociaciones en TPV

- A es una línea de una transacción o informe B
 - LineaVenta-Venta
- A es conocido/registrado/informado/capturado en B
 - Venta-TPV
- A es miembro de B
 - Cajero-Tienda
- A usa o maneja B
 - Cajero-TPV, Gerente-TPV
- A comunica con B
 - Cliente-Cajero
- A está relacionado con una transacción B
 - Cliente-Pago, Cajero-Pago
- A es el siguiente a B
 - LineaVenta-LineaVenta
- A es propiedad de B
 - TPV-Tienda

Atributos

- Son valores de tipos de datos: Identidad no tiene sentido
- Tipos Primitivos: *Boolean, Date, Numero, Time, String*
- Tipos no primitivos: *Dinero, Direcciones, Colores, Número Tlfno, Número Seguridad Social, DNI, Código de Producto Universal, Código Postal,..*
- Tipos no primitivos se deben modelar como clases si:
 - Están formados por varias partes
 - Son aplicables operaciones
 - Tiene otros atributos
 - Es una cantidad con una unidad
- No utilizar atributos como claves ajenas

Documentos de Análisis Requisitos

- Visión
 - Define visión del producto de las personas involucradas, en términos de sus necesidades y características.
- Casos de Uso
- Especificación Complementaria
 - Captura otros requisitos
- Glosario
 - Captura términos y definiciones

Especificación Complementaria

- ***Funcionalidad***
 - Abarca varios casos de uso
 - Ej. “Almacenar información errores”, “Cualquier uso requiere autenticación de usuario”
- ***Requisitos No Funcionales (Atributos de calidad)***
 - Usabilidad, Mantenimiento, Adaptación, Fiabilidad, Rendimiento
 - Restricciones Implementación (hardware, software, desarrollo)
 - Componentes comprados y *free open source*
 - Interfaces
 - Reglas de Negocio (Ej. Reglas de descuento, Cálculo impuestos)
 - Cuestiones Legales
 - Cuestiones sobre el entorno físico y operacionales
 - Información en dominios relacionados

Visión

- Oportunidad
- Definición del problema
- Alternativas
- Descripción personal involucrado (*stakeholder*)
- Objetivos usuario
- Perspectiva del producto
- Beneficios del producto
- Lista de **características** del producto
- Coste y precio
- Otros requisitos y restricciones

Lista de Características del Sistema

- Alguna funcionalidad no puede ser asignada a un caso de uso:
“El sistema debe hacer transacciones con sistemas externos de inventario, contabilidad y cálculo de impuestos”
- Para algunas aplicaciones conviene más una lista de características que casos de uso
 - Ej. Servidor de aplicación

¿Qué hacemos primero?

1. Escribir un borrador poco elaborado de la *Visión* en la *Etapas Inicial*
2. Identificar *objetivos del usuario* y *casos de uso* para ellos
3. Escribir algunos casos de uso y comenzar *Especificación Complementaria*
4. Refinar Visión

Elaboración de Requisitos y Visión

- ¿Cuándo?
 - Etapa Inicial, un poco
 - Modelado de requisitos en cada iteración
- ¿Dónde?
 - Inicio en talleres de requisitos, se completa después
- ¿Quién?
 - Analista de sistemas, Arquitecto Software, Usuarios
- ¿Cómo?
 - Herramientas de requisitos *web-enabled* integradas con un procesador de texto

Casos de uso e iteraciones

- Asignar prioridad a casos de uso
- Escribir casos de uso en su forma expandida
- Asignar casos de uso a iteraciones.
- Varias versiones de un caso de uso complejo, para añadir complejidad de modo incremental.
- Necesidad de comunicación con el usuario
- Al final un caso de uso *esencial* se transforma en su forma *concreta*.


Iteraciones

- Dirigidas por el riesgo
 - Asociar a cada caso de uso un nivel de riesgo e importancia para el negocio
- Comenzar pronto con la programación
- Realizar pruebas
- Rápida retroalimentación
- Se obtiene la arquitectura en las primeras iteraciones

Prototipo Inicial

- Prototipo desechable
- Fácil de construir con herramientas visuales.
- Sirve para validar los requisitos: analista y usuarios llegan a un acuerdo sobre la funcionalidad y vocabulario.
- Utilizar los casos de uso.
- Incluye las interfaces de usuario

Contenidos

- Introducción
- Modelado del Negocio
- Modelado de Requisitos
- Modelado del Análisis 
- Patrones GRASP
- Modelado del Diseño
- Casos prácticos

Modelo del análisis

Objetivo:

- A partir de los casos de uso obtener el **diseño preliminar** del sistema que deberá ser refinado en el modelo del diseño.
- Nivel de abstracción más alto que en el diseño. **Visión ideal del sistema.**
- Se define una arquitectura del sistema.

Modelo del análisis

- Para cada caso de uso se define un ***diagrama de secuencia del sistema*** que muestra los eventos que un actor genera durante la interacción con el sistema (caja negra)
- Cada evento da origen a una **operación** del sistema
- El efecto de las operaciones se describen mediante ***contratos*** especificados mediante una plantilla (opcional) .

Modelo del análisis

- Para cada ***operación del sistema*** se define una colaboración (***diagrama de interacción***) que muestra cómo deben colaborar los objetos para satisfacer la postcondición expresada en el ***contrato*** de dicha operación.
- Diseñar las colaboraciones, asignando responsabilidades a las clases. Utilizaremos ***patrones GRASP*** (luego *patrones de diseño*).
- Crear el **modelo de clases** a partir del modelo conceptual, conforme se definen las colaboraciones

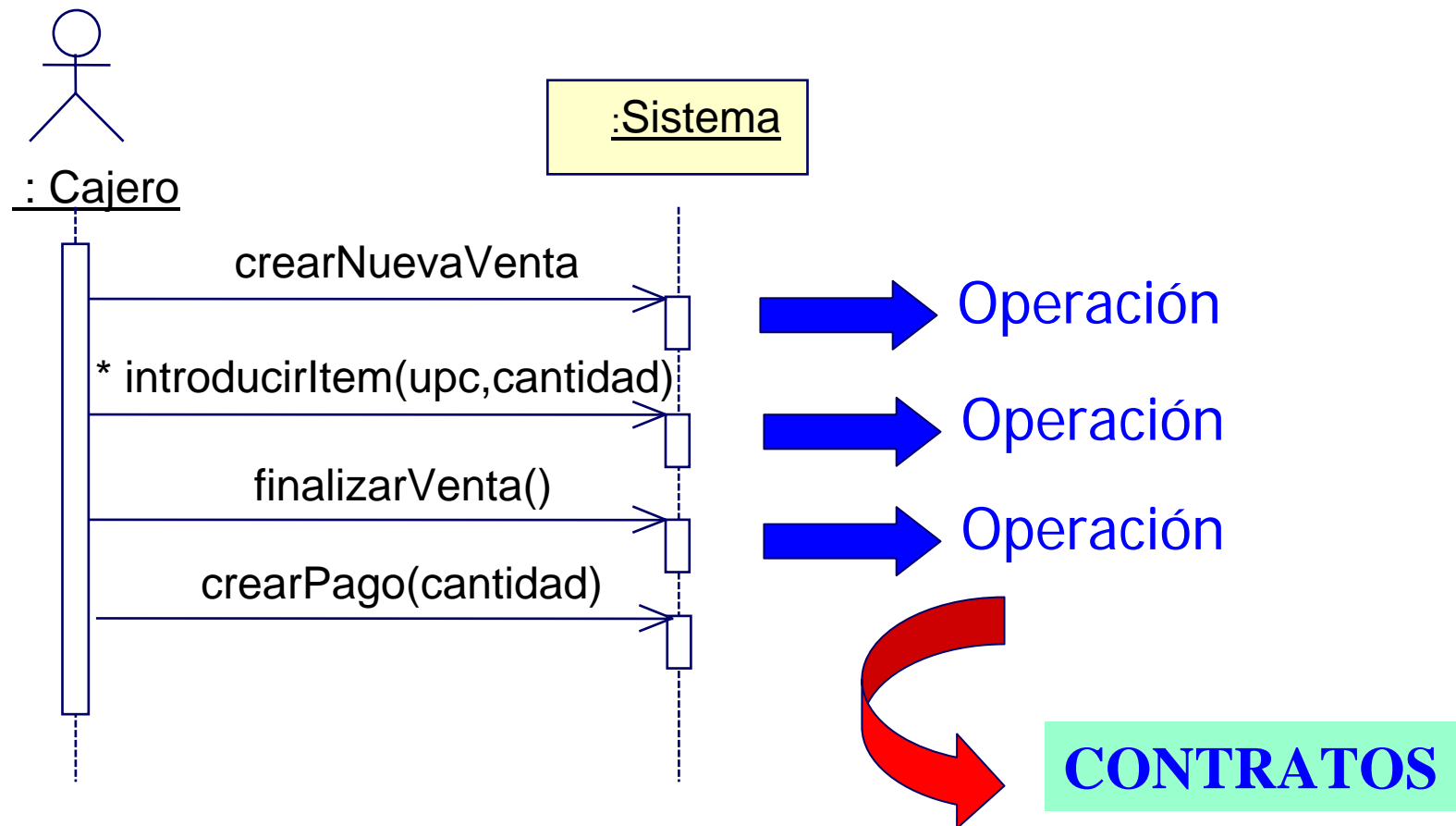
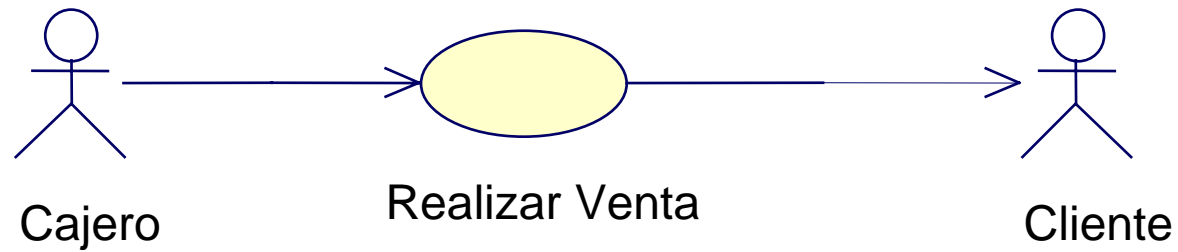
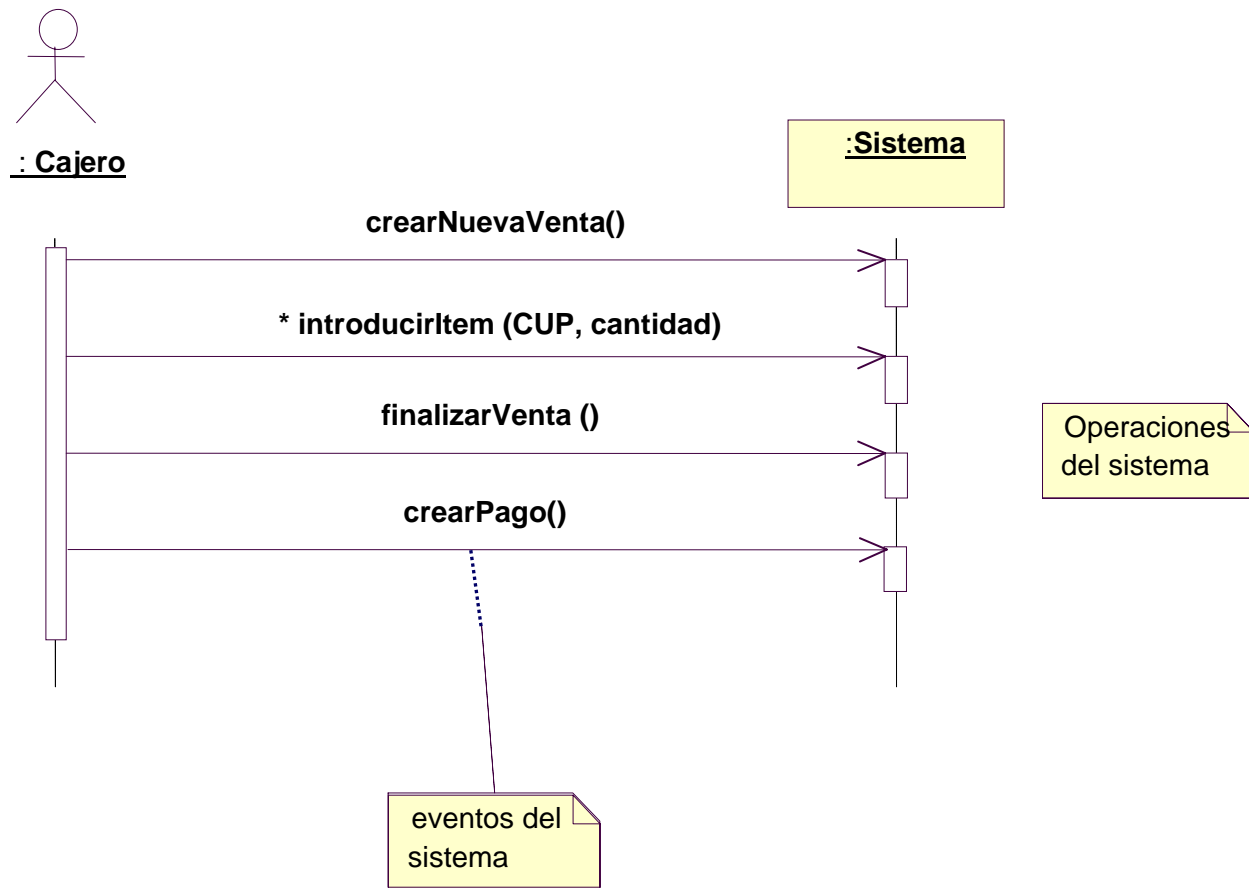


Diagrama de Secuencia del Sistema

(Caso de Uso "Realizar Venta")

1. Cliente llega al TPV con ítem
 2. Cajero inicia venta
 3. Cajero introduce id ítem
 4. Sistema registra línea de venta y presenta parcial
- Cajero repite pasos 3 y 4 hasta que acaba
5. Sistema presenta total
 6. Cajero requiere pago
 7. ...



Contratos

- Descripción detallada del comportamiento del sistema en términos de cambios de estado a los objetos en el *Modelo Conceptual*, tras la ejecución de una *operación del sistema*.
- Definición basada en pre y postcondiciones
- Las **postcondiciones** indican
 - Creación y Eliminación de objetos
 - Creación y Eliminación de asociaciones
 - Modificación de atributos

Contratos

- Las postcondiciones serán **incompletas** y se descubrirán detalles en el diseño.
- Escribimos las postcondiciones a partir del modelo conceptual.

¿Son redundantes los contratos con los casos de uso?

Contratos

- “Útiles cuando hay mucha complejidad y la precisión es un valor añadido”
- “**Normalmente no son necesarios**, si un equipo escribe un contrato para cada caso de uso es una señal de unos casos de uso muy pobres o falta de colaboración con los expertos o que les gusta escribir documentación innecesaria”
- “En la práctica la mayoría de detalles se pueden inferir de los casos de uso de modo obvio, aunque obvio es un concepto muy escurridizo”.

Contratos

1. Identificar operaciones del sistema en DSS
2. Construir un contrato para operaciones complejas y quizá sutiles en sus resultados, o que no están claras en el caso de uso.
3. Describir postcondiciones
 - Creación/Eliminación de instancias
 - Creación/Eliminación de asociaciones
 - Modificación de atributos
 - ¡No olvidar crear asociaciones!
 - Se puede usar OCL

Plantilla de un Contrato

Nombre operación	Signatura de la operación
Referencias cruzadas (opcional)	Casos de uso en los que puede tener lugar esta operación
Precondiciones	Suposiciones relevantes sobre el estado del sistema o de objetos del modelo conceptual, antes de ejecutar la operación. Suposiciones no triviales
Postcondiciones	Estado de objetos del dominio después de que se complete la operación

Contrato “IntroducirItem”

Nombre: introducirItem (itemID: itemID, cantidad: integer)

Referencias Cruzadas: Registrar Venta

Precondiciones: Hay una venta en curso

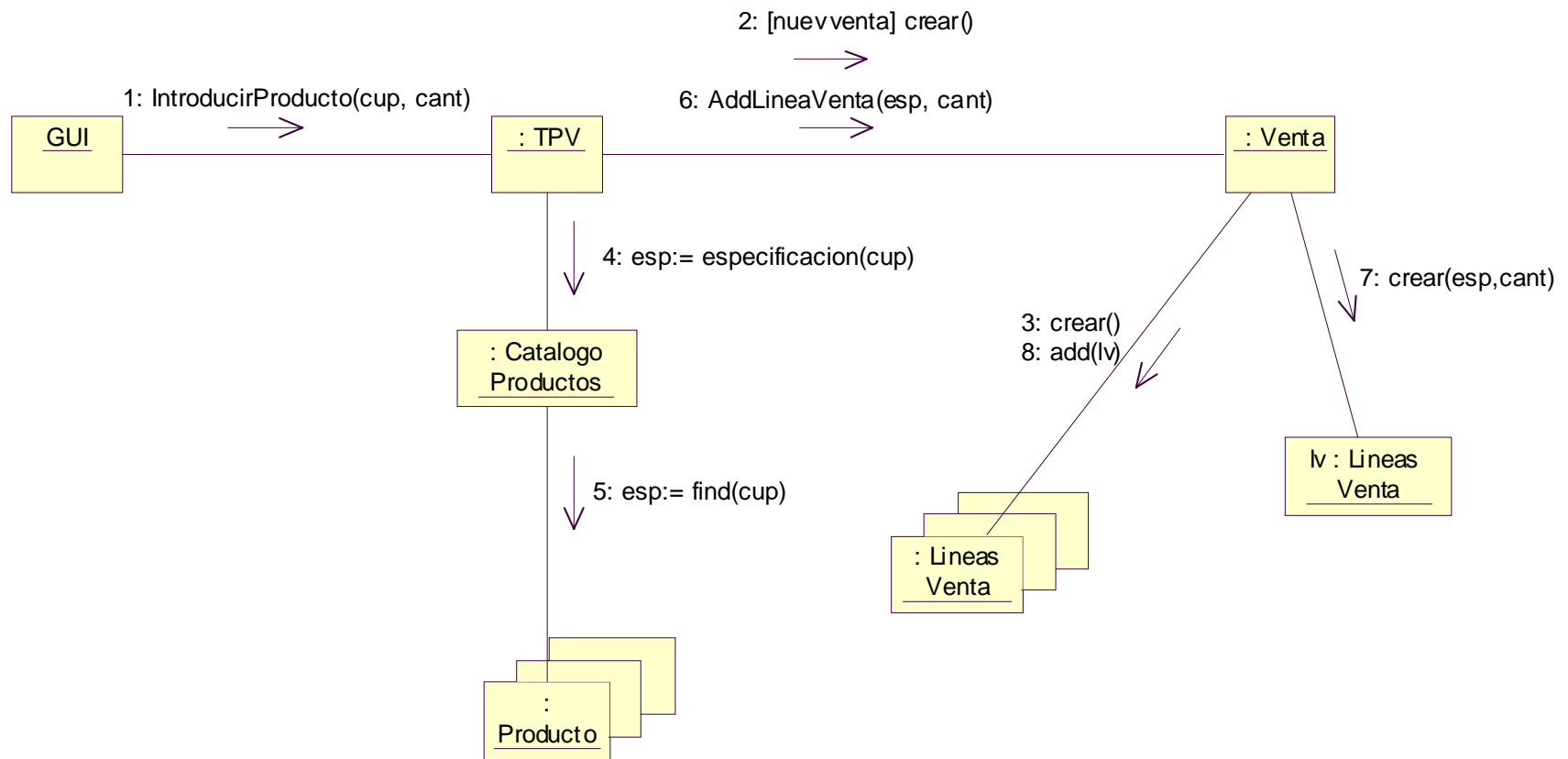
Postcondiciones:

- Se creó una instancia *lv* de *LineaVenta*
- Se asoció *lv* a la venta en curso *v*
- Se asignó *cantidad* a *lv.cantidad*
- *lv* se asoció a una *EspecificaciónProducto* según *itemID*

Colaboración

- Diagramas de secuencia o colaboración
- Crear estos diagramas es una actividad de AOO/DOO muy creativa.
- Diseño de colaboraciones es la parte más difícil.
- Punto de partida para la programación.
- Crear diagramas en parejas.
- Crear modelo de clases de análisis en paralelo.

Diagrama de Colaboración




Modelo de clases del análisis

- El *modelo de clases del análisis* se crea a partir del *modelo conceptual*.
 - Se añade una clase por cada tipo de objeto del dominio que participa en la colaboración.
 - Se añaden atributos según los contratos.
 - Se añaden métodos según las colaboraciones.
- Para aquellas clases con objetos con comportamiento dependiente del estado, se construye una máquina de estados:
 - Dispositivos, Controladores, Ventanas, etc.

Modelo del análisis

- En la primera iteración, en esta fase se definen los subsistemas.
- En el modelo del diseño se refinarán las colaboraciones del análisis para añadir aspectos relacionados con la plataforma, patrones de diseño, etc.

Contenidos

- Introducción
- Modelado del Negocio
- Modelado de Requisitos
- Modelado del Análisis
- Patrones GRASP 
- Modelado del Diseño
- Casos prácticos

Patrones GRASP

- “Describen los principios básicos de asignación de responsabilidades a clases”.
- “Distribuir responsabilidades en la parte más difícil del diseño OO. Consume la mayor parte del tiempo”.
- Patrones GRASP:

Experto

Bajo Acoplamiento

Controlador

Indirección

Creador

Alta Cohesión

Polimorfismo

Variaciones Protegidas

Responsabilidades y Métodos

- “Contrato u obligación de una clase”.
- Dos categorías:
 - **Conocer**
 - datos encapsulados privados
 - existencia de objetos conectados
 - datos derivados o calculados
 - **Hacer**
 - Algo él mismo, como crear un objeto o realizar un cálculo
 - Iniciar una acción en otros objetos
 - Controlar y coordinar actividades en otros objetos

Responsabilidades y Métodos

- Responsabilidades “conocer” se pueden inferir de las asociaciones y atributos del modelo conceptual.
- Puede asignarse a varias clases y métodos dependiendo de su granularidad.
- Una responsabilidad se implementa mediante uno o más métodos.
- Diagramas de interacción muestran elecciones en la asignación de responsabilidades.

Experto

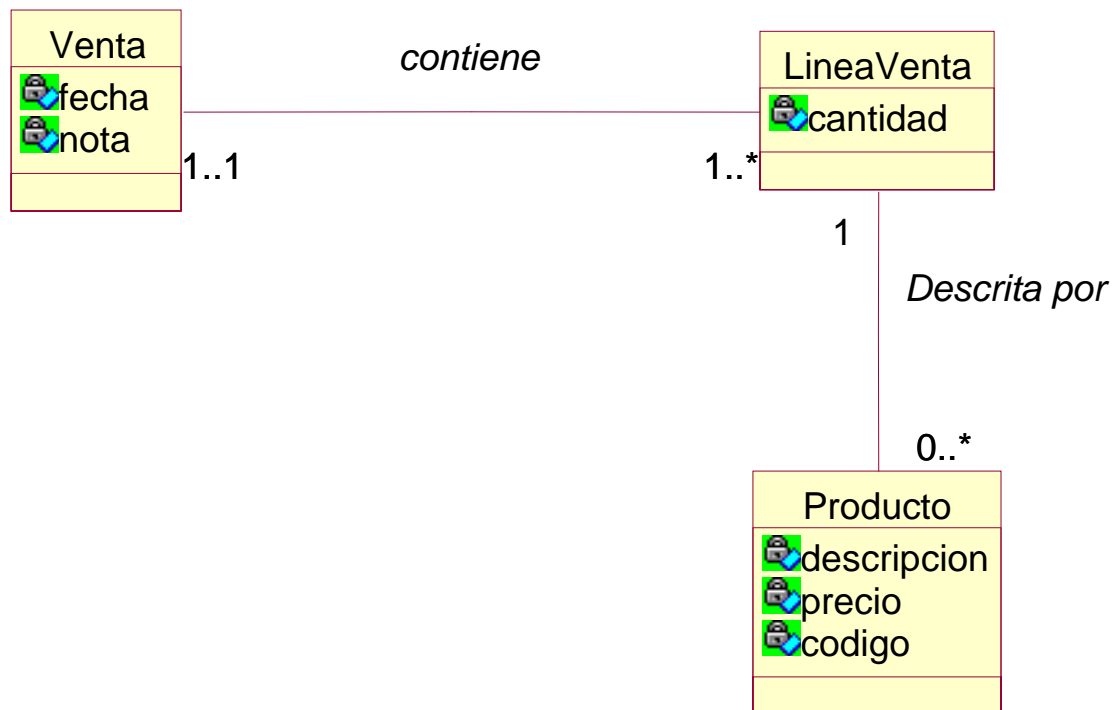
¿Cómo se asignan responsabilidades?

“Asignar una responsabilidad a la clase que tiene la información necesaria para cumplimentarla”

- Heurísticas relacionadas
 - Distribuir responsabilidades de forma homogénea
 - No crear clases “dios”
- Beneficios
 - Se conserva encapsulación: Bajo acoplamiento
 - Alta Cohesión: clases más ligeras

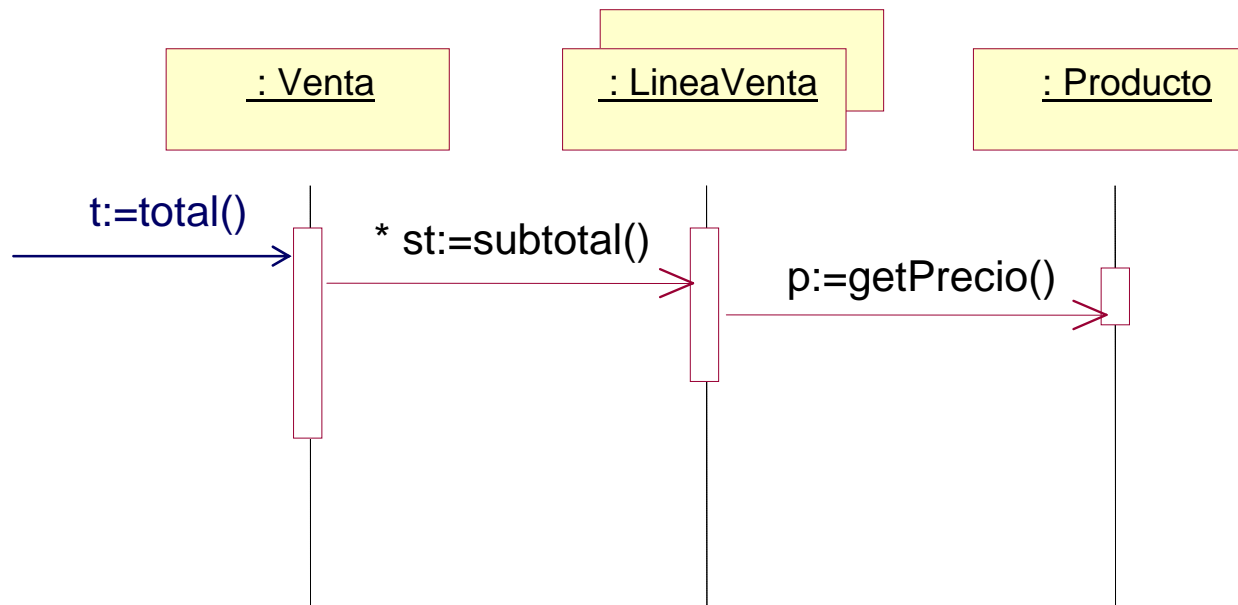
Ejemplo 1

- ¿Quién es el responsable de conocer el total de una venta?



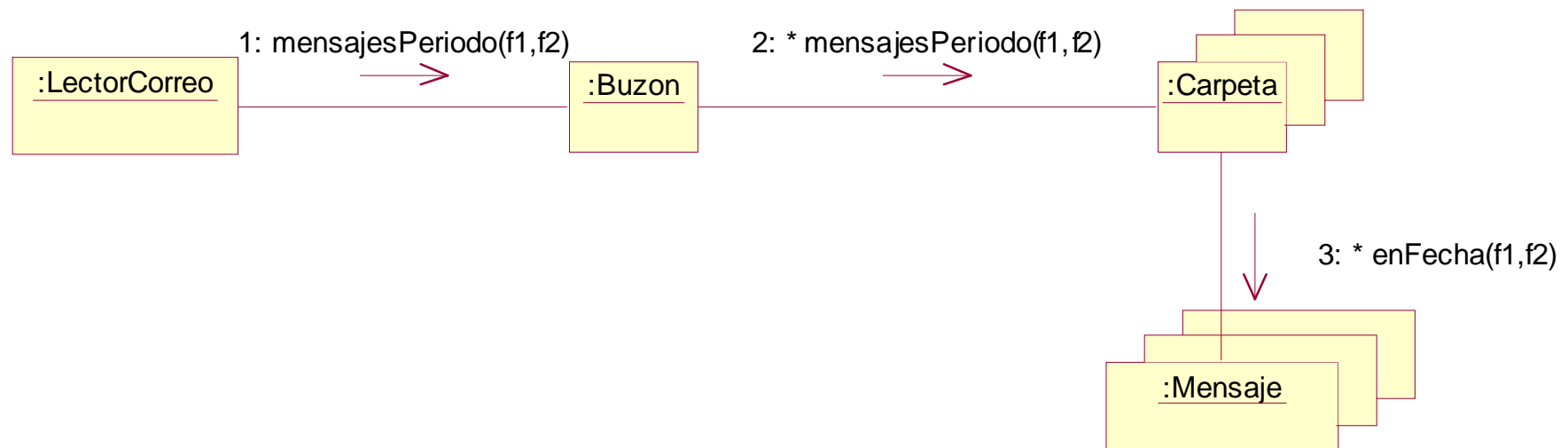
Ejemplo 1

¿Quién es el responsable de conocer el total de una venta?



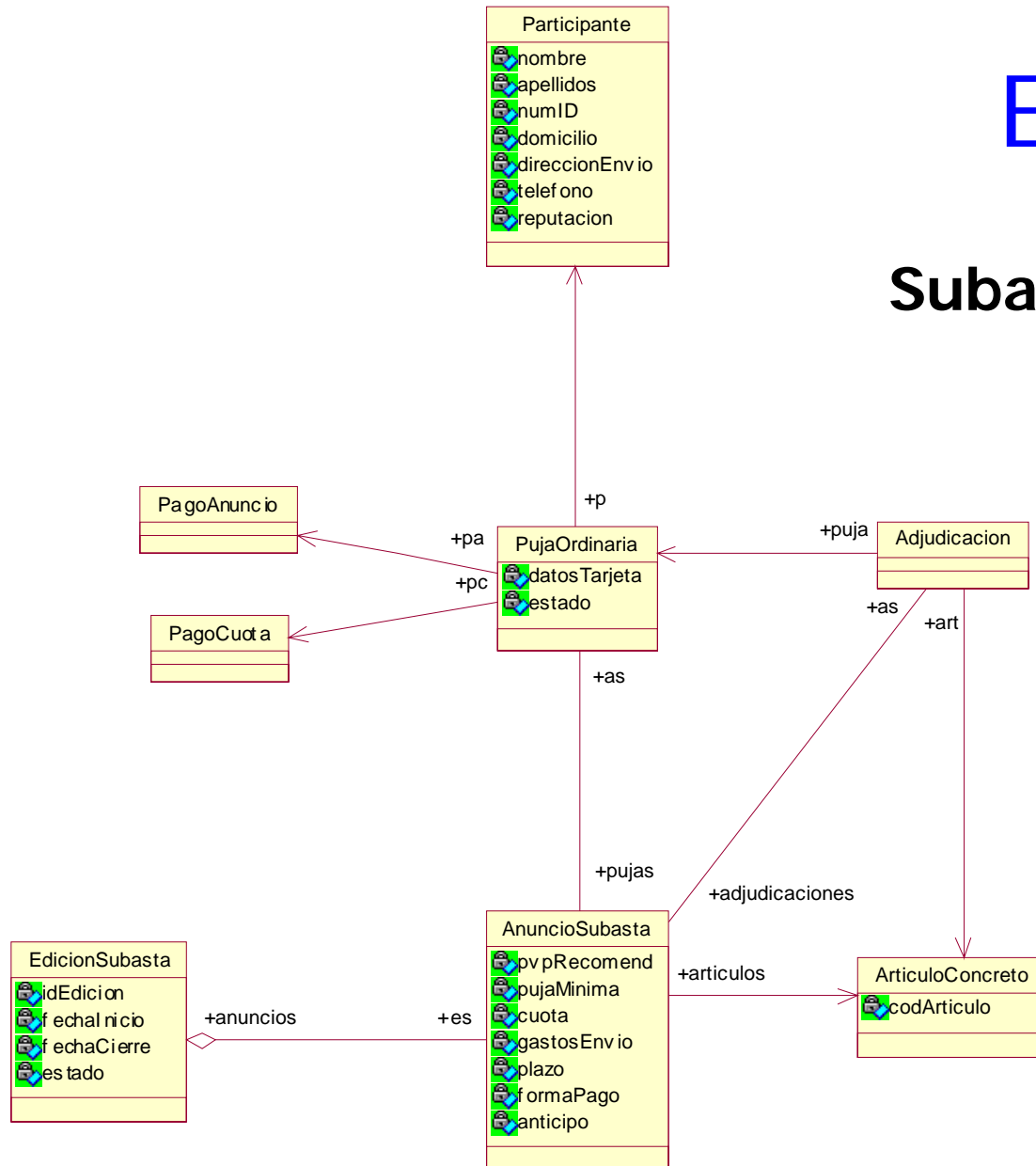
Ejemplo 2

¿Quién es el responsable de conocer todos los mensajes recibidos entre dos fechas?



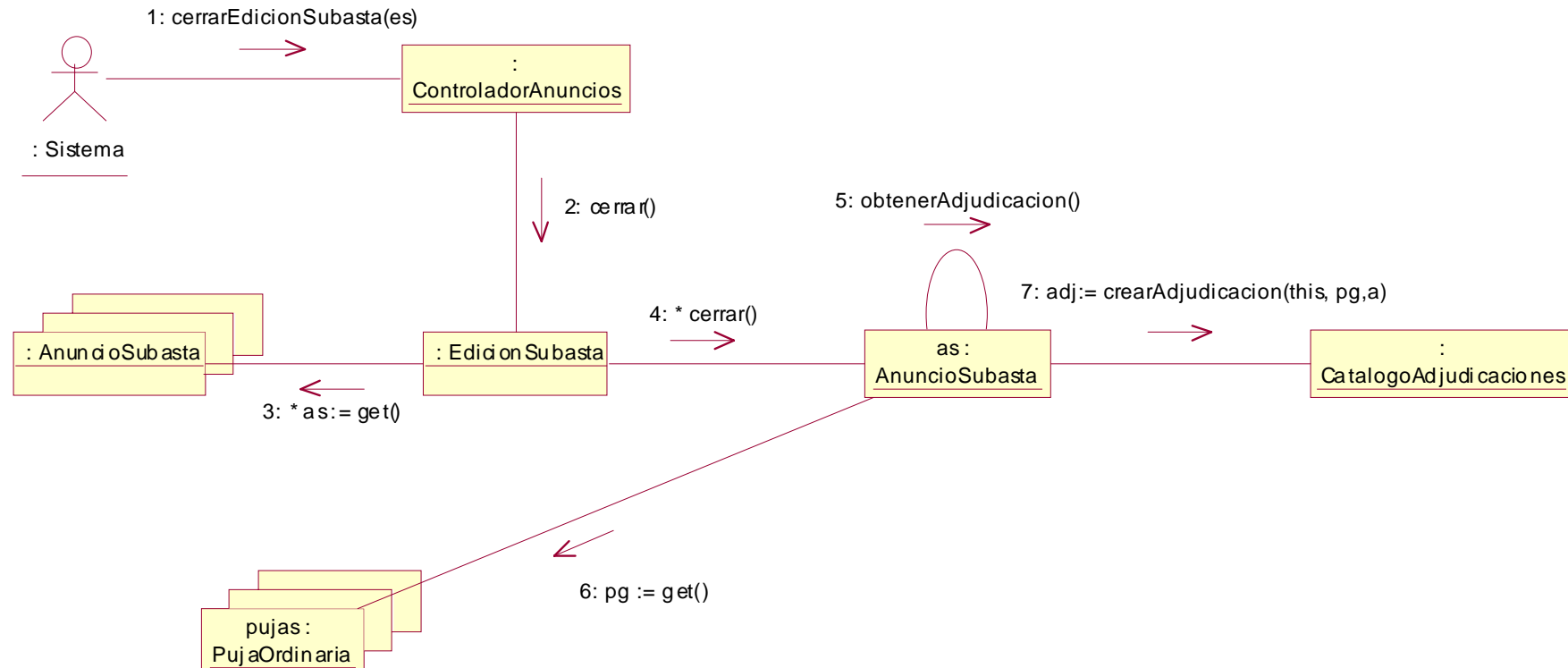
Ejemplo 3

Subastas por internet



Ejemplo 3

¿Quién es el responsable de obtener la puja ganadora?



Creador

¿Quién es responsable de crear una nueva instancia de una cierta clase?

“Asignar a la clase B la responsabilidad de crear instancias de una clase A si:

- B es una agregación de objetos de A
- B contiene objetos de A
- B registra instancias de A
- B hace un uso específico de los objetos de A
- B proporciona los datos de inicialización necesarios para crear un objeto de A”

Creador

- ¿Quién debería crear una instancia de la clase *LineaVenta*?

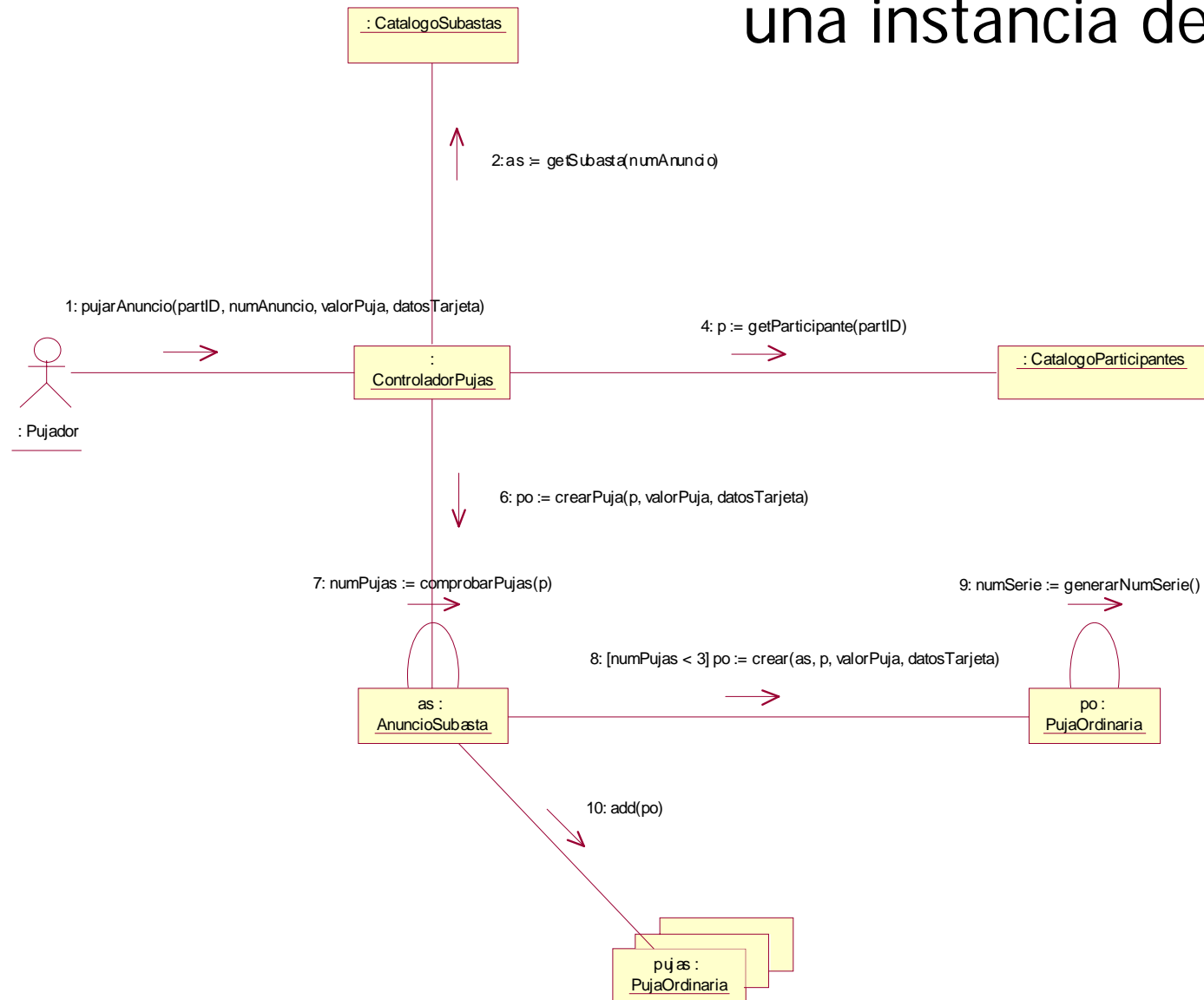
Una instancia de la clase *Venta* es una agregación de instancias de la clase *LineaVenta*

Venta incluirá *crearLineaVenta(int cantidad)*

- Beneficios:
 - Bajo acoplamiento

Ejemplo

¿Quién debería crear una instancia de puja?



Bajo Acoplamiento

¿Cómo reducir las dependencias entre clases?

“Asignar responsabilidad de modo que se consiga un bajo acoplamiento”

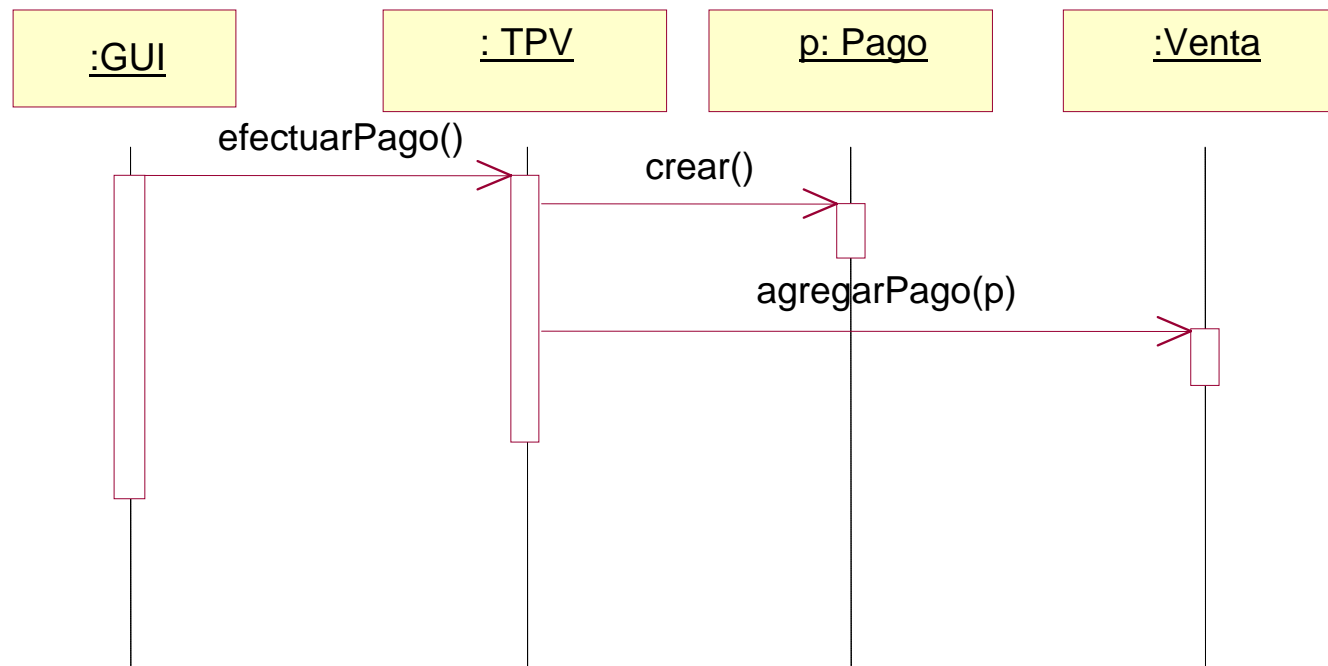
Es un patrón evaluativo

- No considerarlo de forma independiente, sino junto a los patrones Experto y Alta Cohesión.
- **Beneficios:** Facilita i) reutilización, ii) comprensión de las clases y iii) mantenimiento

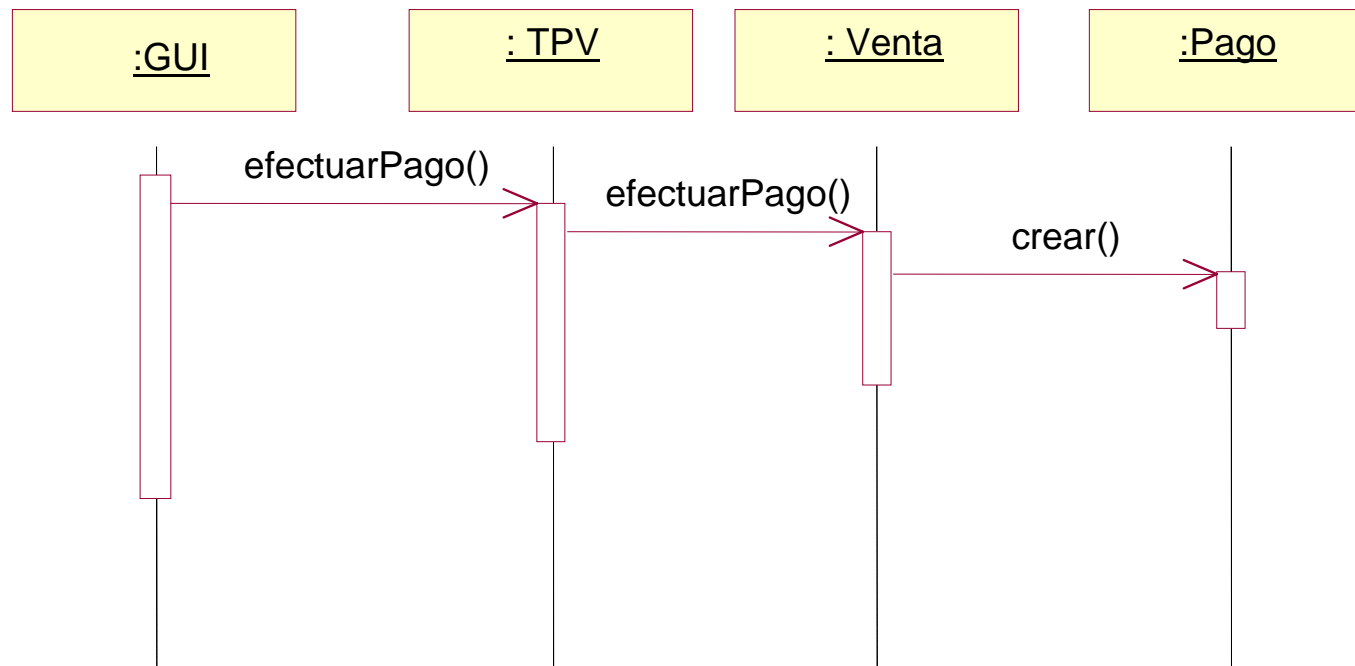
Tipos de dependencias

- Existe acoplamiento entre una clase A y otra B si:
 - i) A posee un atributo de tipo B
 - ii) A tiene un método con un parámetro, una variable local o devuelve un valor de tipo B
 - iii) A es subclase directa o indirecta de B
 - iv) A implementa la interfaz B (Java)

Ejemplo



Ejemplo



Alta Cohesión

¿Cuánto están de relacionadas las responsabilidades de una clase?

“ Asignar una responsabilidad de modo que la cohesión siga siendo alta”

- Baja Cohesión: Clases con responsabilidades que deberían haber delegado. Son “clases dios”. Difíciles de comprender, reutilizar, mantener.
- Ejemplo anterior: En el primer caso TPV tendría más operaciones, mejor delegar.

Alta Cohesión

- **Muy baja cohesión:**
 - Clase Acceso*BD-RPC*
 - Mezcla de funcionalidad
- **Baja cohesión:**
 - Clase Acceso*BD*
 - Muchos métodos
- **Alta cohesión:**
 - Clase Acceso*BD* + Familia de clases relacionada
- **Cohesión moderada:**
 - Clase *Empresa*: conoce empleados e información financiera

Alta Cohesión

- Una clase con alta cohesión no tiene muchos métodos, que están muy relacionados funcionalmente, y no realiza mucho trabajo. Colabora con otras clases.
- **Beneficios**
 - Mejora reutilización
 - Clases más claras, se comprenden mejor
 - Mejora mantenimiento

Controlador

“Quién se encarga de atender los eventos del sistema”

“Asignar responsabilidad de manejar eventos externos a un controlador:

- i) el sistema global, dispositivo o subsistema
 - ii) un manejador de los eventos de un caso de uso
- El mismo controlador para todos los eventos de un mismo caso de uso.

Controlador

- Cualquier arquitectura distingue entre *capa presentación* y *capa del dominio*.
- Las clases de la interfaz (capa presentación) no deben encargarse de realizar las tareas asociadas a un evento. Deben delegarlas a un controlador.
- Separar interfaz de la lógica de aplicación.
- Las operaciones del sistema en la capa del dominio.

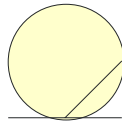
Controlador

- Un controlador es un objeto que no pertenece a la capa de presentación, se encarga de recibir y manejar un evento del sistema procedente normalmente de una interfaz gráfica.
- Una clase controlador incluye un método para cada operación del sistema que maneja.
- *Controlador fachada vs. Controlador caso de uso*

Controlador

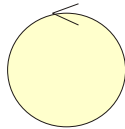
- ¿Cuándo debemos escoger un *controlador de caso de uso* ?
 - Cuando con las otras alternativas obtenemos controladores “saturados”
 - Es posible llevar el estado de la sesión
- En el Proceso Unificado se distingue entre:
 - *objetos entidad* (dominio)
 - *objetos control* (controladores)
 - *objetos frontera* (interfaces GUI)

Controlador



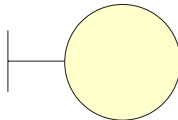
Alumno

Objeto Entidad



Matriculacion

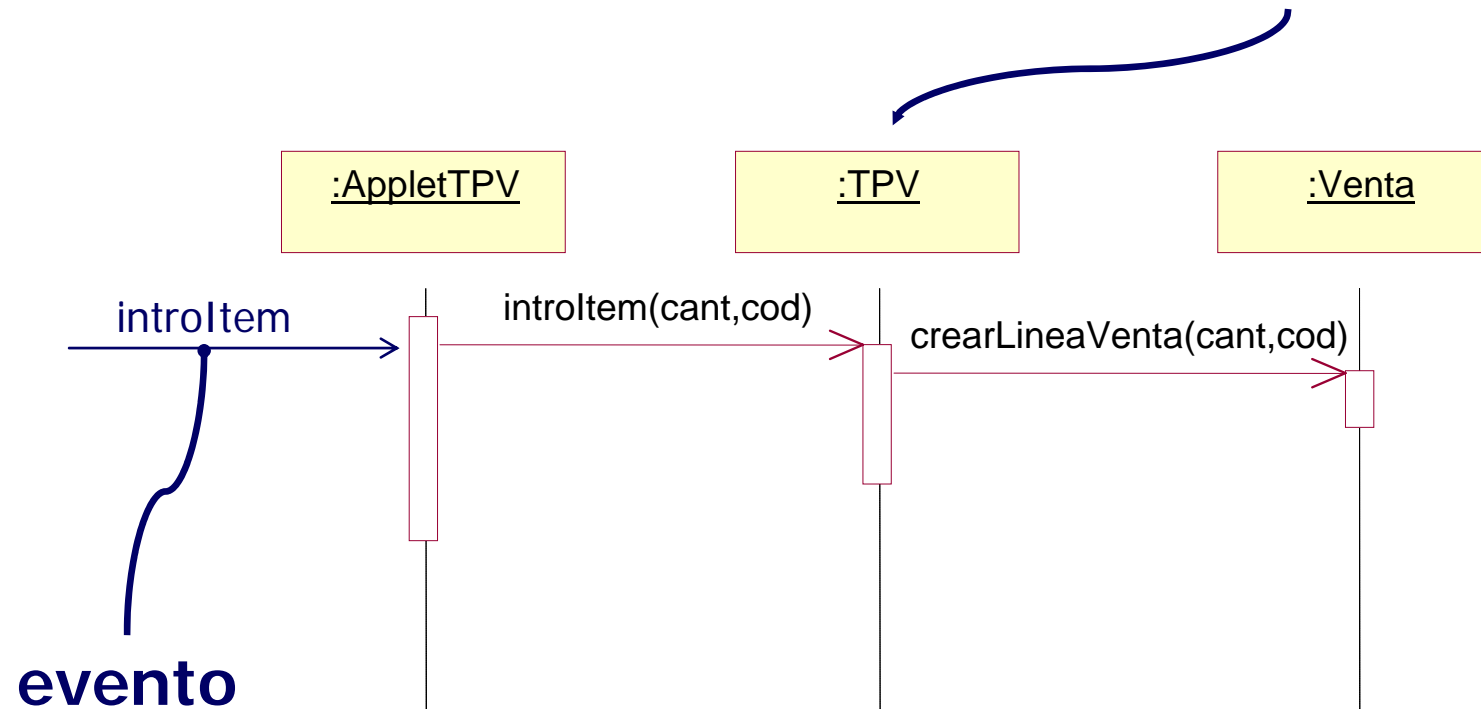
Objeto Control



GUIMatricula

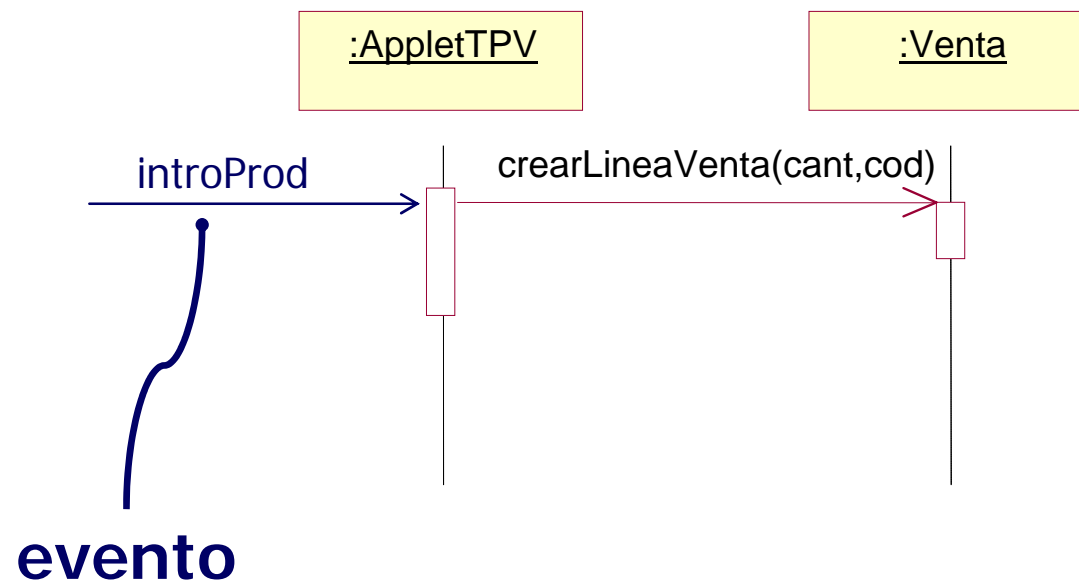
Objeto Frontera

Ejemplo controlador



Acoplamiento adecuado de la capa presentación con la capa del dominio

Ejemplo



Acoplamiento inadecuado de la capa presentación
con la capa del dominio

Controlador

Dado el evento *“Introducir ítem”*, tendríamos varios posibles controladores:

i) TPV, ii) Tienda, iii) Manejador Compra

- Beneficios de un controlador:
 - Favorece la reutilización
 - Posibilidad de capturar información sobre el estado de una sesión

Polimorfismo

¿Cómo manejar las alternativas basadas en el tipo? ¿Cómo crear componentes software conectables (*pluggable*)?

Cuando las alternativas o comportamiento relacionado varían según el tipo asigne la responsabilidad para el comportamiento a los tipos para los que varía el comportamiento.

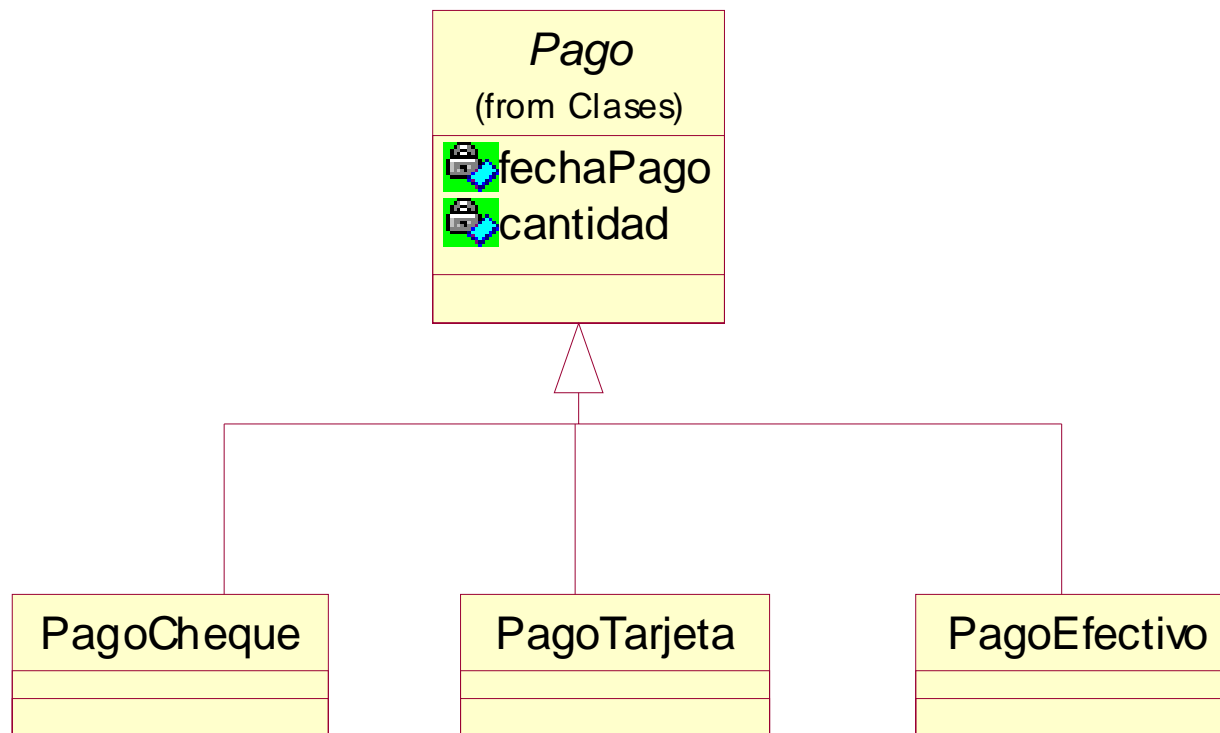
Polimorfismo

- No realizar análisis de casos de uso basado en el tipo de los objetos.

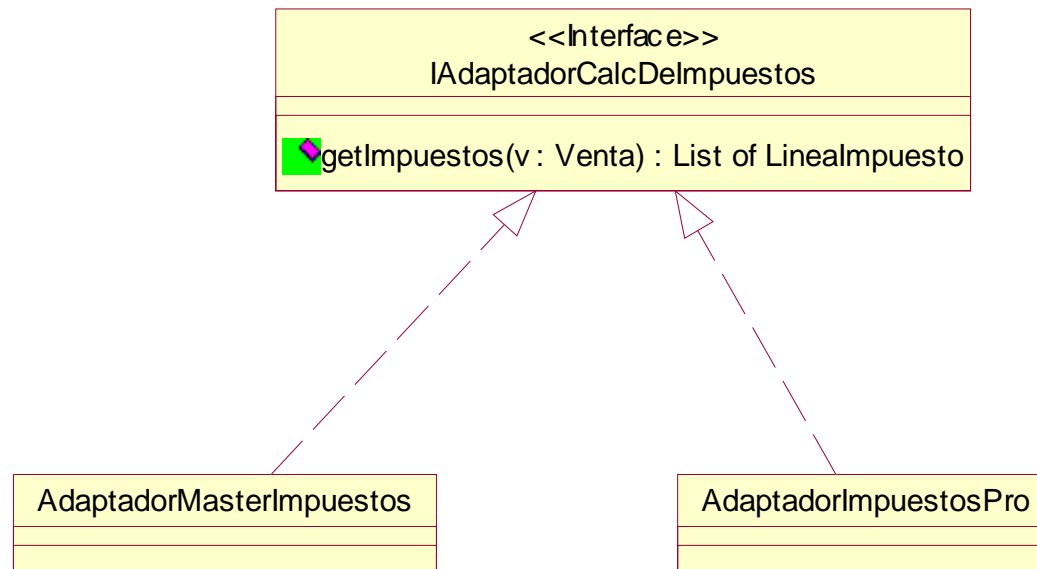
```
if tipoPago = "Cheque" then autorizarPagoCheque  
else if tipoPago = "Efectivo" then autorizarPagoEfectivo  
else if tipoPago = "Tarjeta" then autorizarPagoTarjeta  
else if ...
```

- Sustituir análisis de casos de uso por jerarquía de herencia.

Polimorfismo



Polimorfismo



- Se añaden fácilmente las extensiones necesarias por nuevas variaciones.
- Los clientes no son afectados por nuevas implementaciones.
- Usar si realmente el punto de variación está motivado

Indirección

¿Cómo evitar un acoplamiento directo entre dos clases?

Asignar la responsabilidad a un intermediario que crea una indirección

Ejemplo: Los adaptadores de cálculo de impuestos

Variaciones Protegidas (VP)

¿Cómo diseñar objetos, subsistemas y sistemas de manera que las variaciones o inestabilidades en estos elementos no tengan un impacto indeseable sobre otros elementos?

Identificar los puntos de variación previstos y asignar responsabilidades para crear una interface alrededor de ellos

Variaciones Protegidas (VP)

- Ejemplo: Adaptadores de cálculo de impuestos, se combina indirección con polimorfismo
- VP es un principio fundamental que motiva la mayoría de mecanismos y patrones en programación y diseño destinados a proporcionar flexibilidad y protección frente al cambio.

Variaciones Protegidas (VP)

- Diseños dirigidos por datos
 - Lectura de códigos, valores, nombres de ficheros,..., de una fuente externa
- Búsqueda de servicios
 - Servicios de nombres (JNDI de Java) o *traders* para obtener un servicio (UDDI para servicios web)
- Diseños dirigidos por un intérprete
- Diseños reflexivos o de nivel meta
 - Usar la introspección
- Acceso Uniforme
- Principio de Sustitución de Liskov

Variaciones Protegidas (VP)

- Diseños de ocultación de la estructura
 - Principio “no hables con extraños”
- VP se aplica a puntos de variación y puntos de evolución.
- VP es esencialmente lo mismo que los principios “Ocultación de Información” y “Abierto-Cerrado”

No hables con extraños

- No enviar mensajes sobre objetos indirectos, sino sobre la instancia actual (self), parámetros de métodos, atributos de la instancia actual y elementos de colecciones de la instancia actual.

```
class A {  
    B at1;  
    void met1() {  
        C obj = at1.getAt2();    }  
        obj.met2();  
    }
```

```
class B {  
    C at2;  
    C getAt2() {return at2;}  
}
```


No hables con extraños

```
class A {  
    B at1;  
    void met1(..) {  
        at1.met3;  
    }  
}
```


```
class B {  
    C at2;  
    C getAt2() {return at2;}  
    void met3() {at2.met2;}  
}
```

```
class C {  
    void met2() {..}  
}
```

Evitar recorrer largos caminos en la estructura de objetos y entonces enviar un mensaje: “diseño frágil”


No hables con extraños

```
class Registro {  
    private Venta venta;  
    public void metodoFragil () {  
        Dinero cantidad =  
            venta.getPago().getCantidadEntregada()  
        ...  
    }  
}
```



**Dinero cantidad =
venta.getCantidadEntregadaEnPago()**

Contenidos

- Introducción
- Modelado del Negocio
- Modelado de Requisitos
- Modelado del Análisis
- Patrones GRASP
- Modelado del Diseño 
- Casos prácticos

Modelo del diseño

Objetivo:

Refinar el diseño del sistema del modelo del análisis considerando los requisitos no funcionales y restricciones del entorno de implementación.

De manera iterativa se refina el modelo de clases y las colaboraciones del análisis hasta obtener un diseño del sistema adecuado para pasar a la implementación.

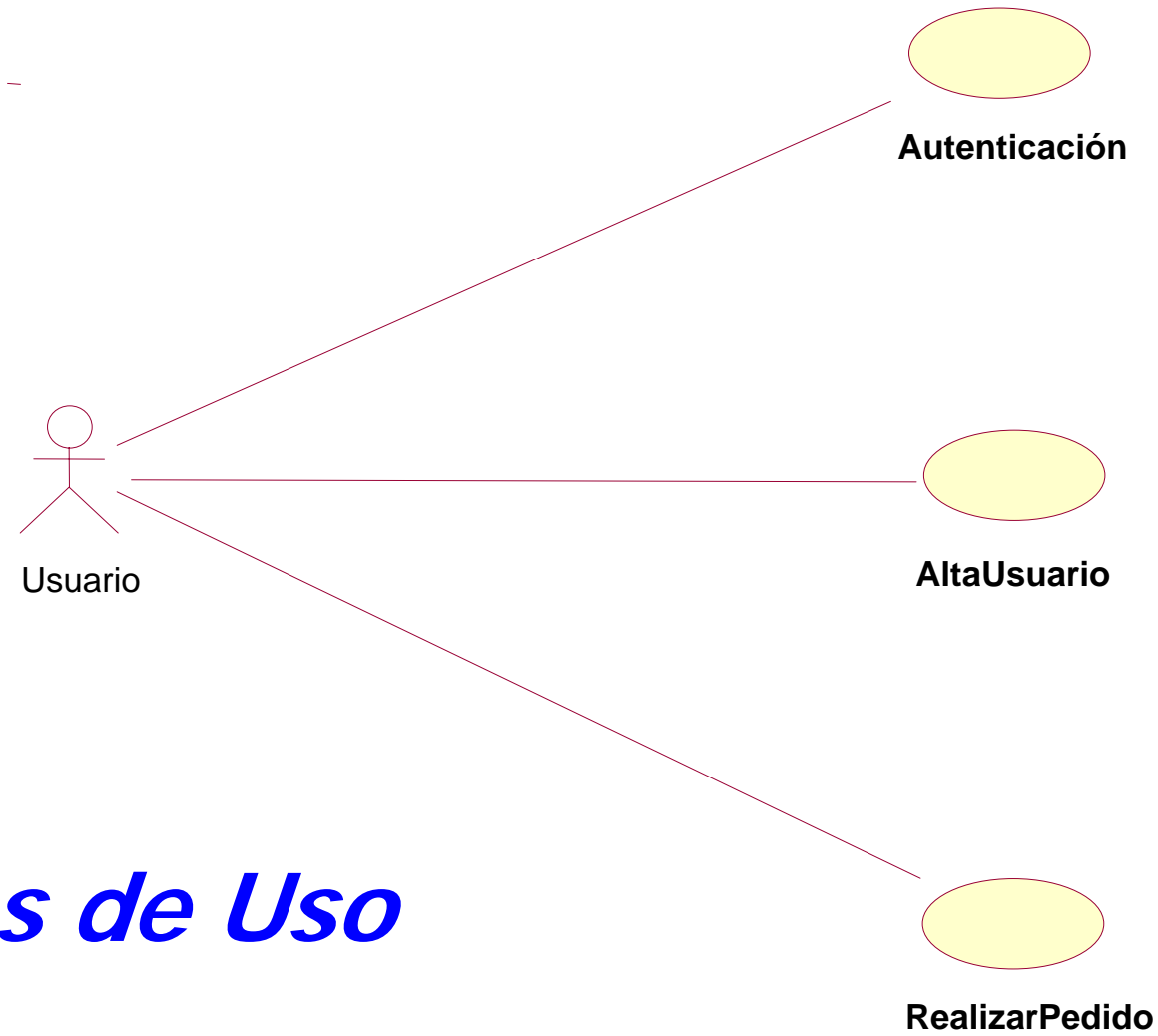
Cuestiones del diseño

- Identificar clases (atributos y métodos) e interfaces en el modelo de clases del diseño
- Establecer asociaciones entre clases.
- Establecer **navegabilidad** para todas las asociaciones.
- Determinar **visibilidad** entre clases.
- Incluir relaciones de **dependencia** entre clases.

Cuestiones del diseño

- Definición de la **arquitectura** del sistema
- **Subsistemas**: Paquetes
- **Patrones de diseño**
- **Estructuras de datos**
- Diseño del **interfaz de usuario**
- Manejo de la **persistencia**
- **Distribución**

Ejemplo 1: *PetStore*



Casos de Uso

Nombre: RealizarPedido

Objetivo: Realizar una compra seleccionando y añadiendo productos al carro de compras, pudiendo cambiar el número de unidades de un producto del carro y generando un pedido en el momento que el usuario decida finalizar la compra.

Actores: Usuario

Precondiciones: El usuario debe estar autenticado.

Flujo Principal:

1. A: Inicia compra
2. S: Crea un carro de compras nuevo asociado a la sesión del usuario.
3. A: El Usuario selecciona y añade un producto al carro de compras.
4. S: El sistema genera un nuevo ítem en el carro de compras correspondiente al nuevo producto.
5. S: Actualiza el importe total del carro de compras.
6. A: El Usuario selecciona un producto del carro de compras e introduce el número de unidades del producto que desea.
7. S: Actualiza el carro de compras reflejando las nuevas unidades. Si el número de unidades era cero, elimina el producto del carro de compras.
8. A: El Usuario finaliza la compra.
9. A: El Usuario introduce los datos personales del pedido: dirección, localidad, provincia, código postal.
10. A: El Usuario introduce los datos de la tarjeta de crédito: tipo de tarjeta, numero de tarjeta, mes y año de caducidad.
11. S: Genera pedido de compras con sus correspondientes líneas de pedido.

Alternativas:

- 3.1) El producto se encuentra en el carro de compras.
 - 3.1.1) S: El sistema incrementa en uno el número de unidades del producto del carro de compras.
- 9.1) La forma de pago es contra reembolso. En este caso, el usuario no tiene que introducir los datos de la tarjeta de crédito.
- 9.2) La forma de pago es mediante transferencia bancaria. En este caso, el usuario no tiene que introducir los datos de la tarjeta de crédito.

Comentarios:

Modelo Conceptual

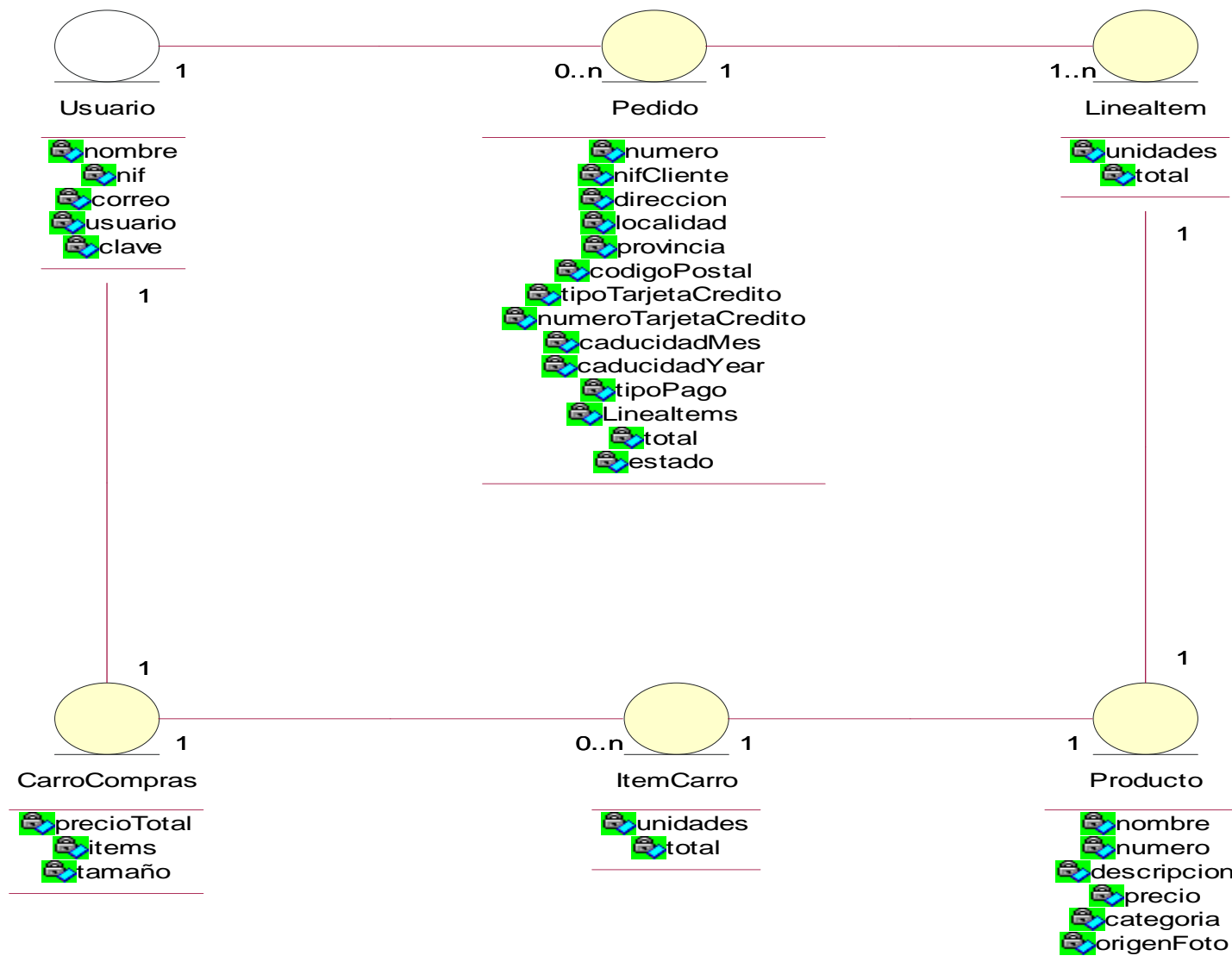
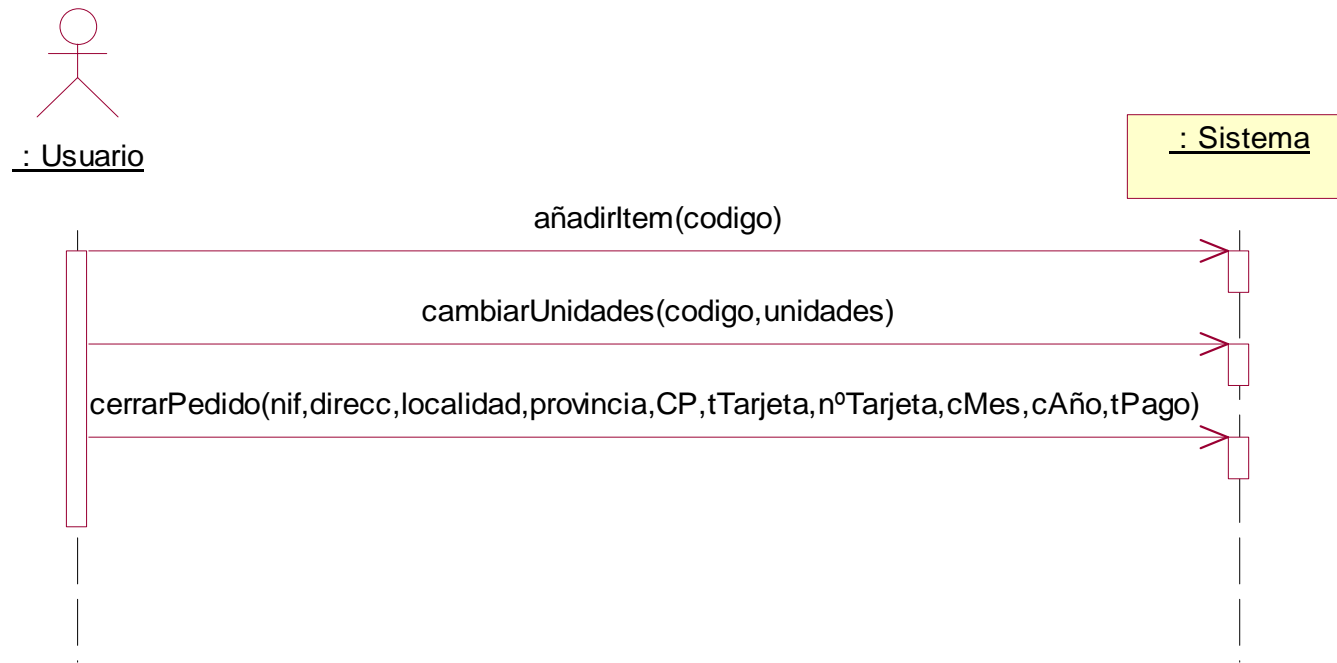


Diagrama de Secuencia del Sistema para "Realizar Pedido"



Colaboración "AñadirItem"

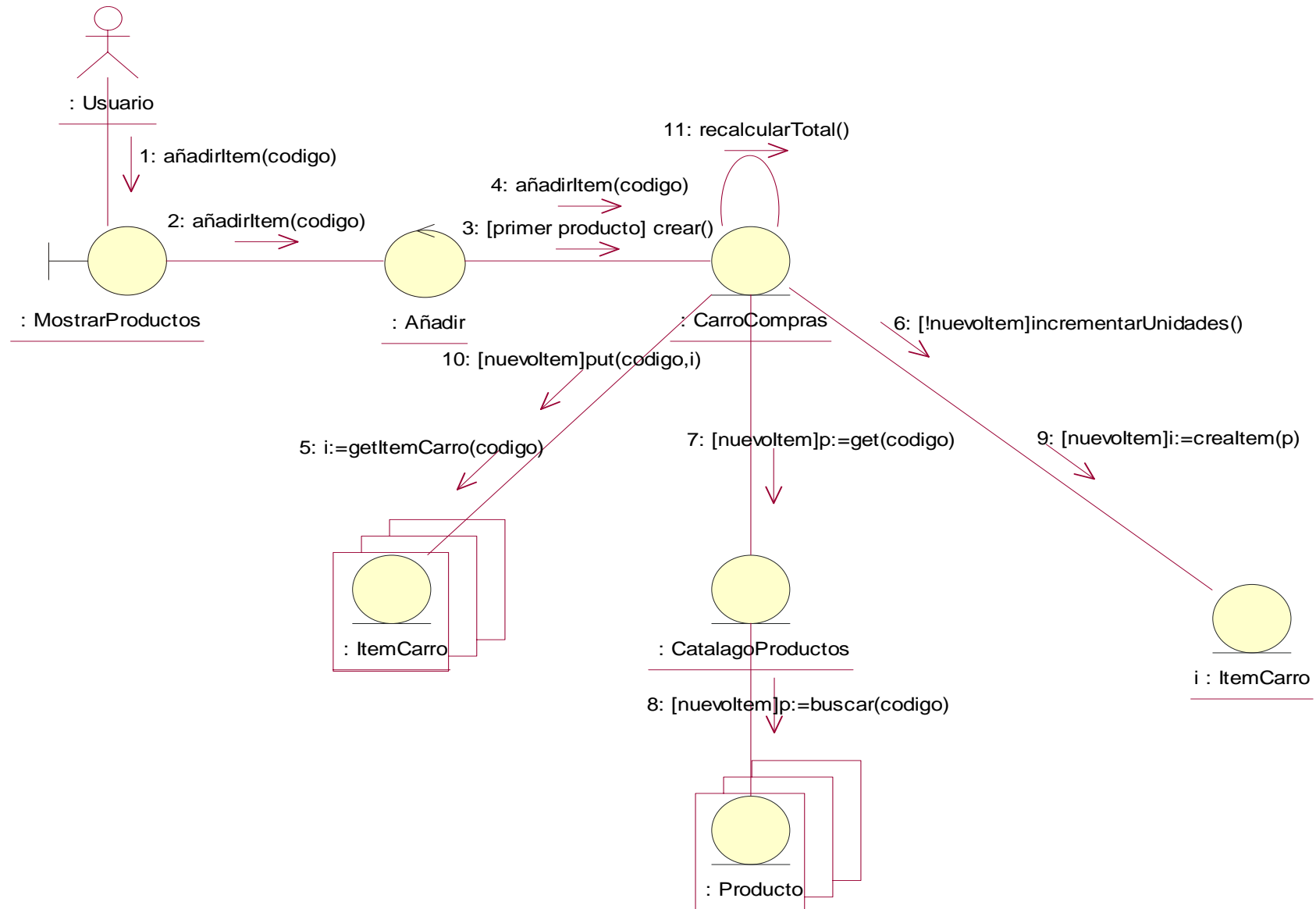
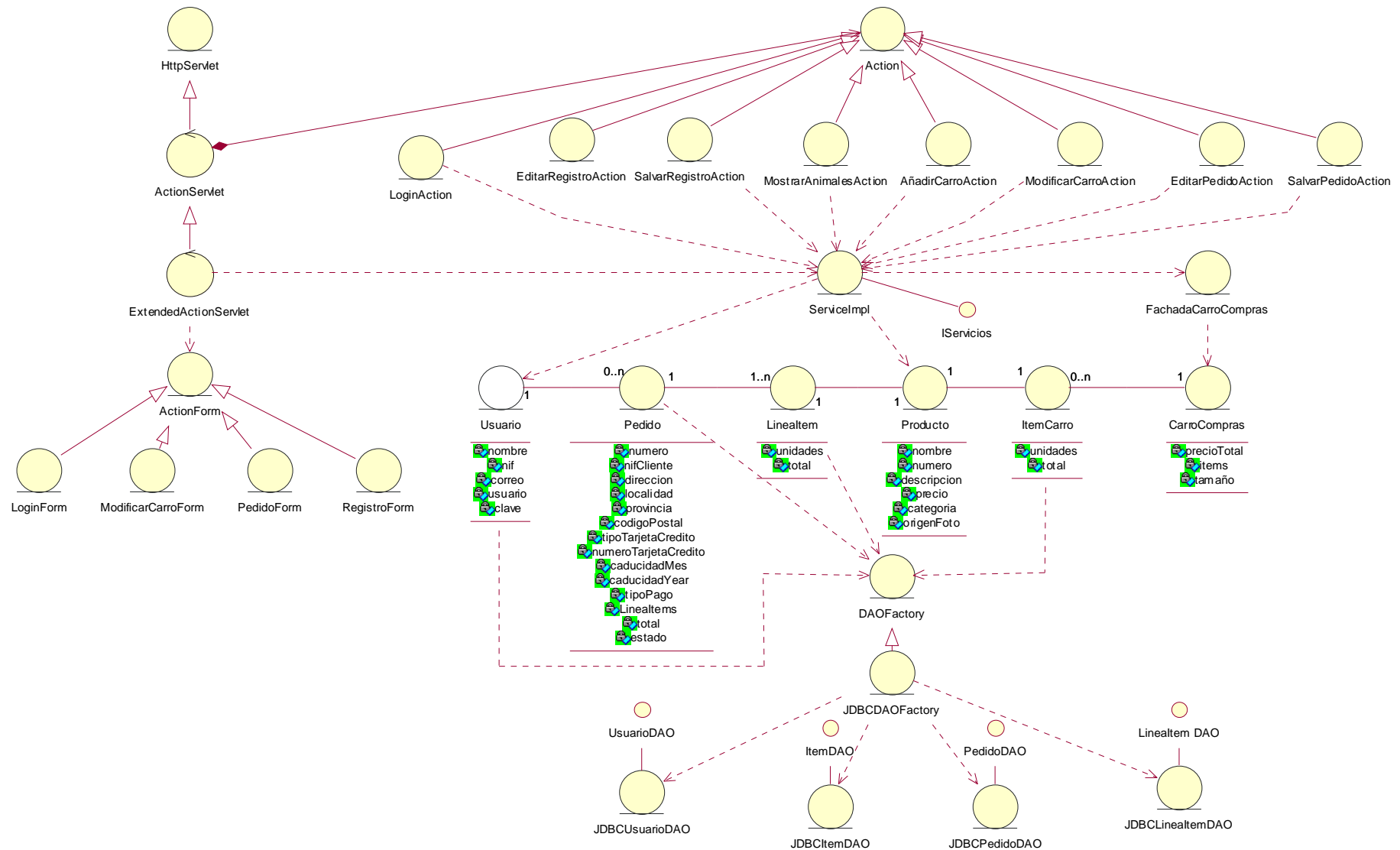
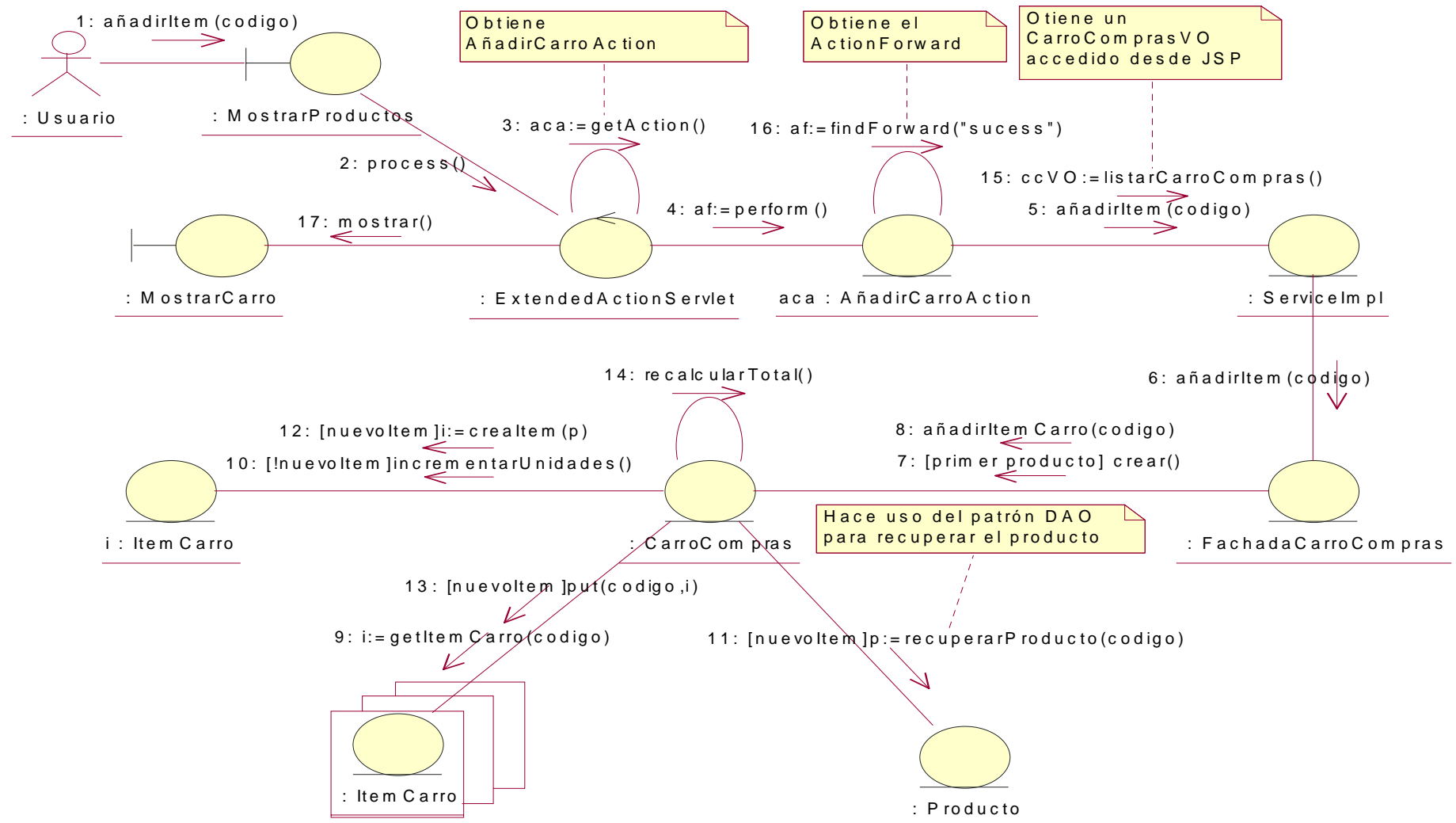


Diagrama de Clases (Struts y JDBC)



Colaboración “Añadir Item” (Struts y JDBC)



Ejemplo 2: Caso de uso *“Realizar Venta”* en sistema *TPV*

Resumen: Un cliente llega al TPV con un conjunto de artículos. El Cajero registra los artículos y se genera un ticket. El cliente paga en efectivo y recoge los artículos.

Actor Principal: Cajero

Personal Involucrado e Intereses:

- Cajero: quiere entradas precisas, rápida y sin errores de pago
 - Compañía: quiere registrar transacciones y satisfacer clientes.
 - ...
- **Precondición:** El cajero se ha identificado y autenticado
 - **Postcondiciones:** Se ha registrado la venta. Se ha calculado el impuesto. Se ha actualizado contabilidad e inventario...

Caso de uso *“Realizar Venta”*

Flujo Básico:

1. A: El cliente llega al TPV con los artículos.
2. A: El cajero inicia una nueva venta
3. A: El cajero introduce el identificador de cada artículo.
4. S: El sistema registra la línea de venta y presenta descripción del artículo, precio y suma parcial.

El Cajero repite los pasos 3 y 4 hasta que se indique.

5. S: El Sistema presenta el total
6. A: El Cajero le dice al Cliente el total a pagar
7. S: El Cliente paga y el sistema gestiona el pago.
8. S: El Sistema registra la venta completa y actualiza Inventario.
9. S: El Sistema presenta recibo

Caso de uso *“Realizar Venta”*

Extensiones (Flujos Alternativos):

3a. Identificador no válido

1. El Sistema señala el error y rechaza la entrada

3-6a. El Cliente pide eliminar un artículo de la compra

1. El Cajero introduce identificador a eliminar
2. El sistema actualiza la suma

...

7a. Pago en efectivo

1. El Cajero introduce cantidad entregada por el cliente
2. El Sistema muestra cantidad a devolver

...

....

Caso de uso *“Realizar Venta”*

Requisitos especiales:

- Interfaz de usuario con pantalla táctil en un monitor de pantalla plana. El texto debe ser visible a un metro de distancia.
- Tiempo de respuesta para autorización de crédito de 30 sg. el 90% de las veces

...

Lista de Tecnología y Variaciones de Datos:

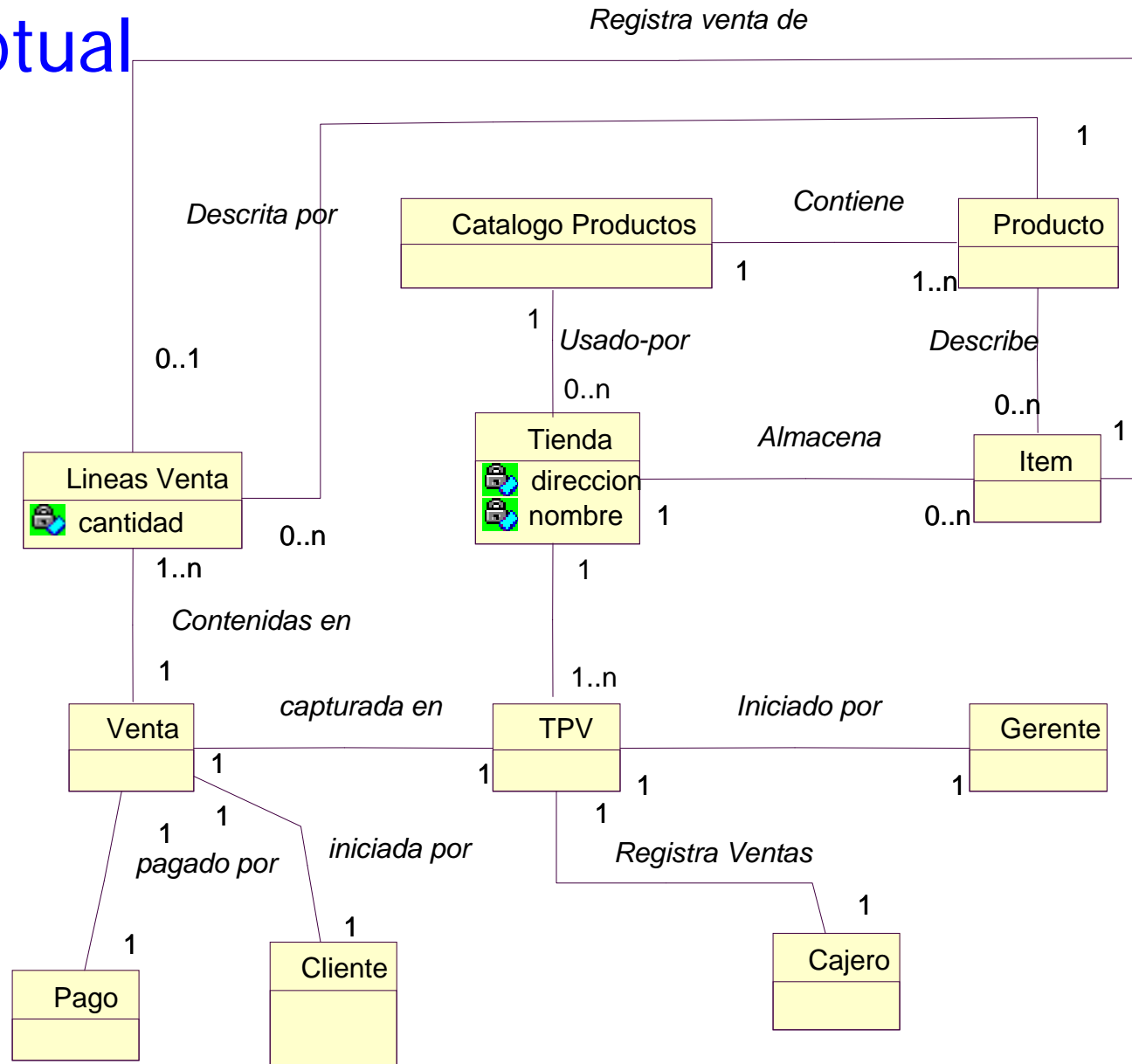
- El identificador podría ser cualquier esquema de código UPC, EAN,...
- La entrada de información de la tarjeta se realiza mediante un lector de tarjetas.

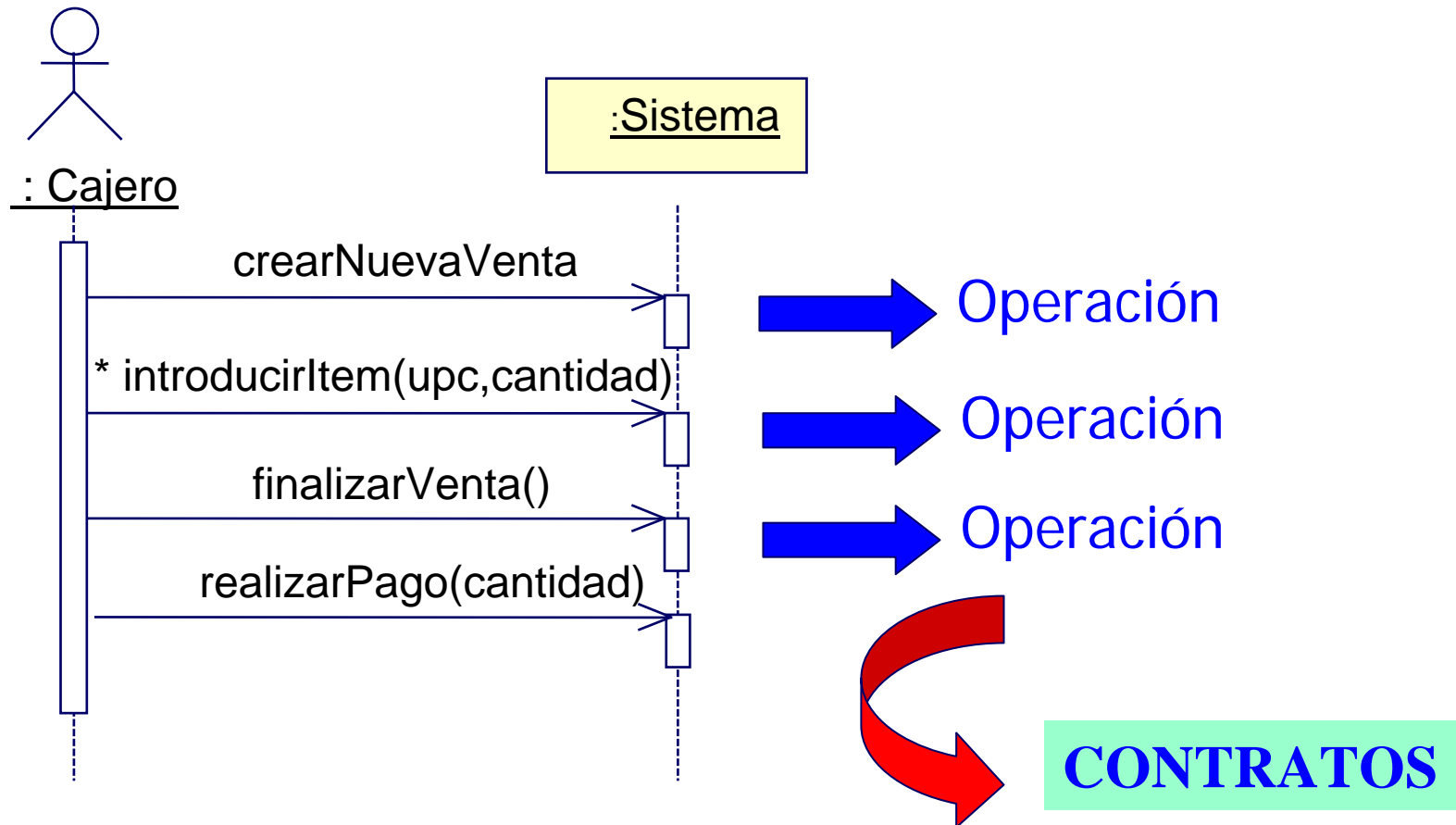
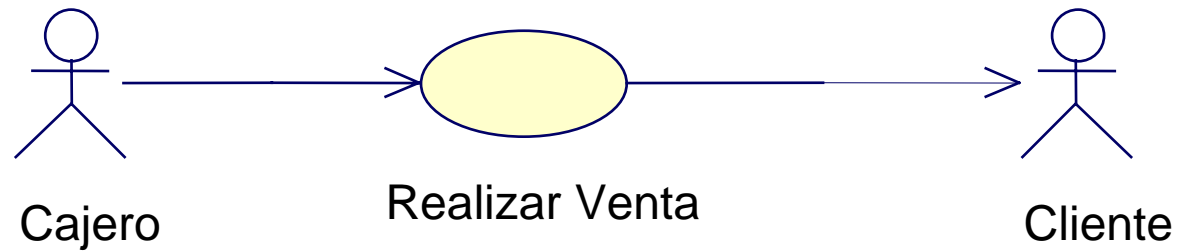
...

Cuestiones Pendientes:

- Explorar cuestiones de recuperación de accesos a servicios remotos
- ¿Qué adaptaciones son necesarias para diferentes negocios?

Modelo Conceptual





Operación: *crearVenta()*

Caso de Uso: *Realizar venta*

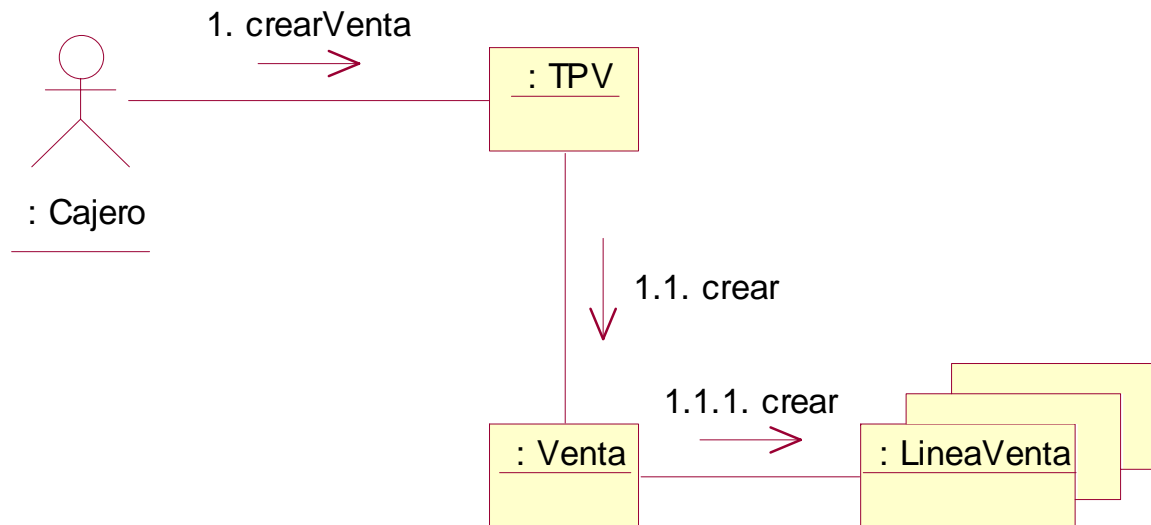
Precondición:

Postcondición:

una instancia de *Venta* *v* fue creada

v se asoció con *TPV*

atributos de *v* fueron inicializados



Operación: *introducirltem(itemID: itemID, cantidad: Dinero)*

Caso de Uso: *Realizar venta*

Precondición: se está procesando una venta

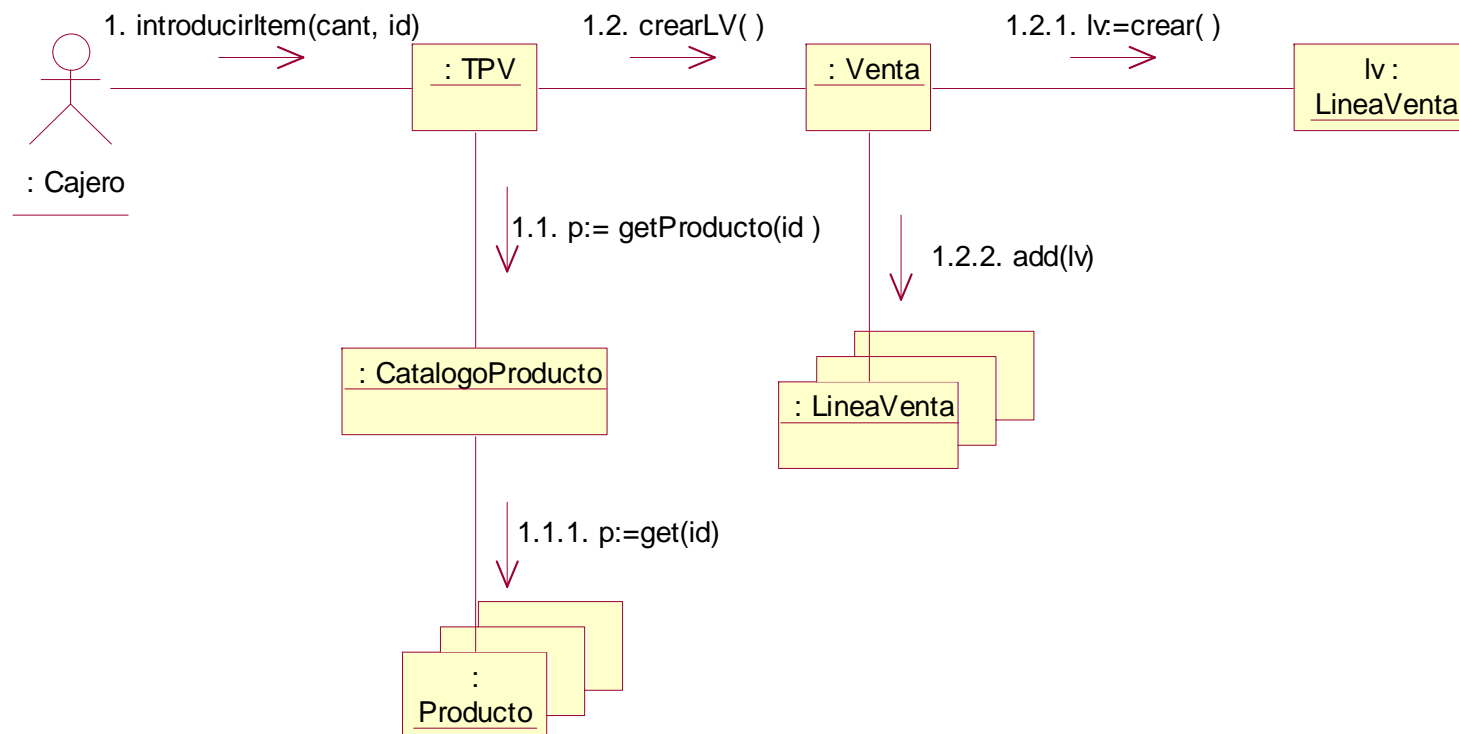
Postcondición:

una instancia de LineaVenta lv fue creada

lv se asoció con la Venta actual

lv.cantidad = cantidad

lv se asoció con el Producto cuyo código es itemID



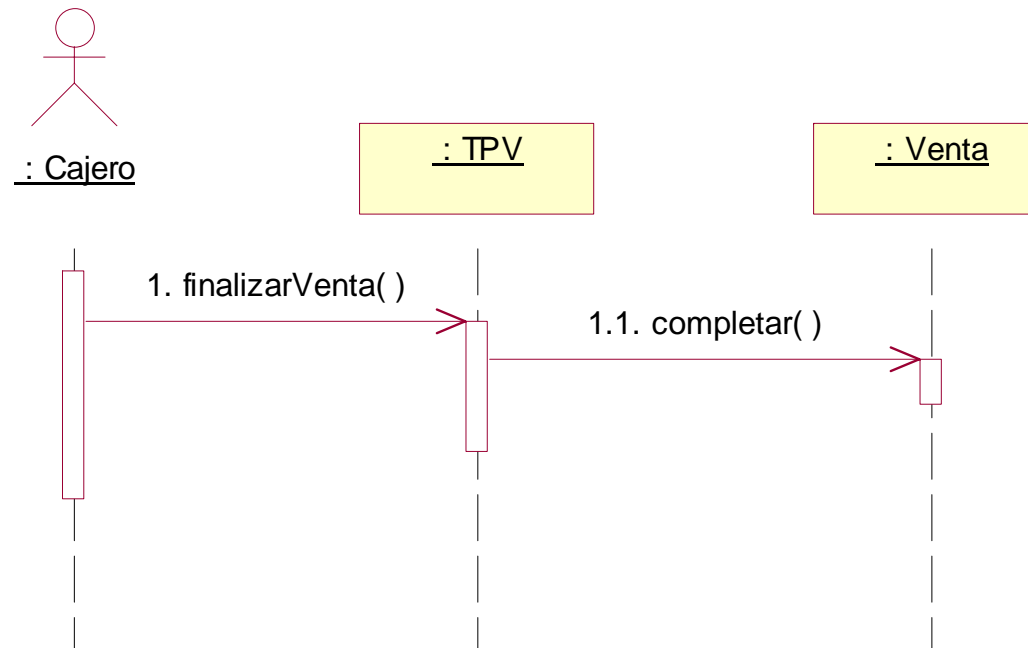
Operación: *finalizarVenta()*

Caso de Uso: *Realizar venta*

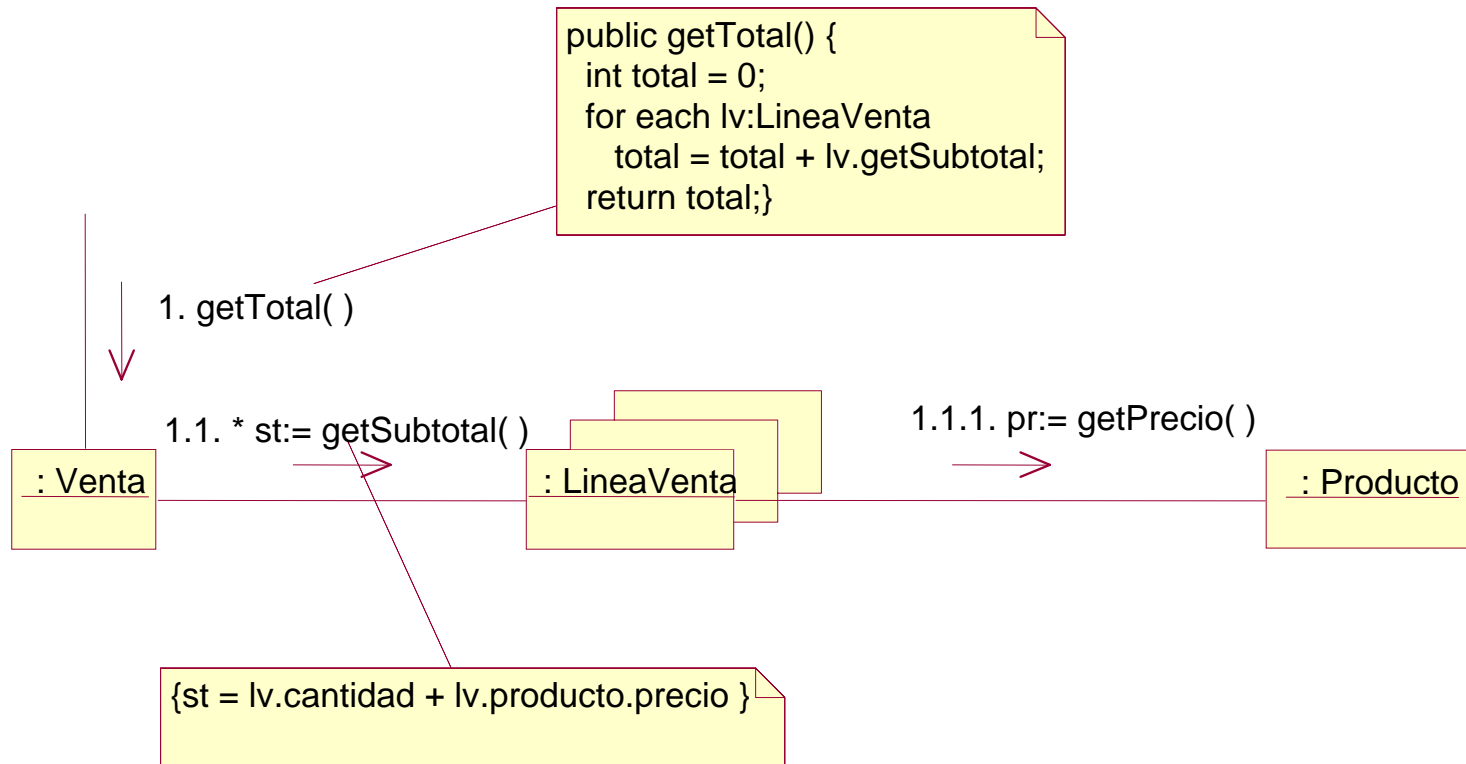
Precondición: se está procesando una venta

Postcondición:

v.esCompleta = true, siendo *v* la venta actual



“El sistema presenta el total de la venta más impuestos”



Operación: *crearPago(cantidad:Dinero)*

Caso de Uso: *Realizar venta*

Precondición: se está procesando una venta

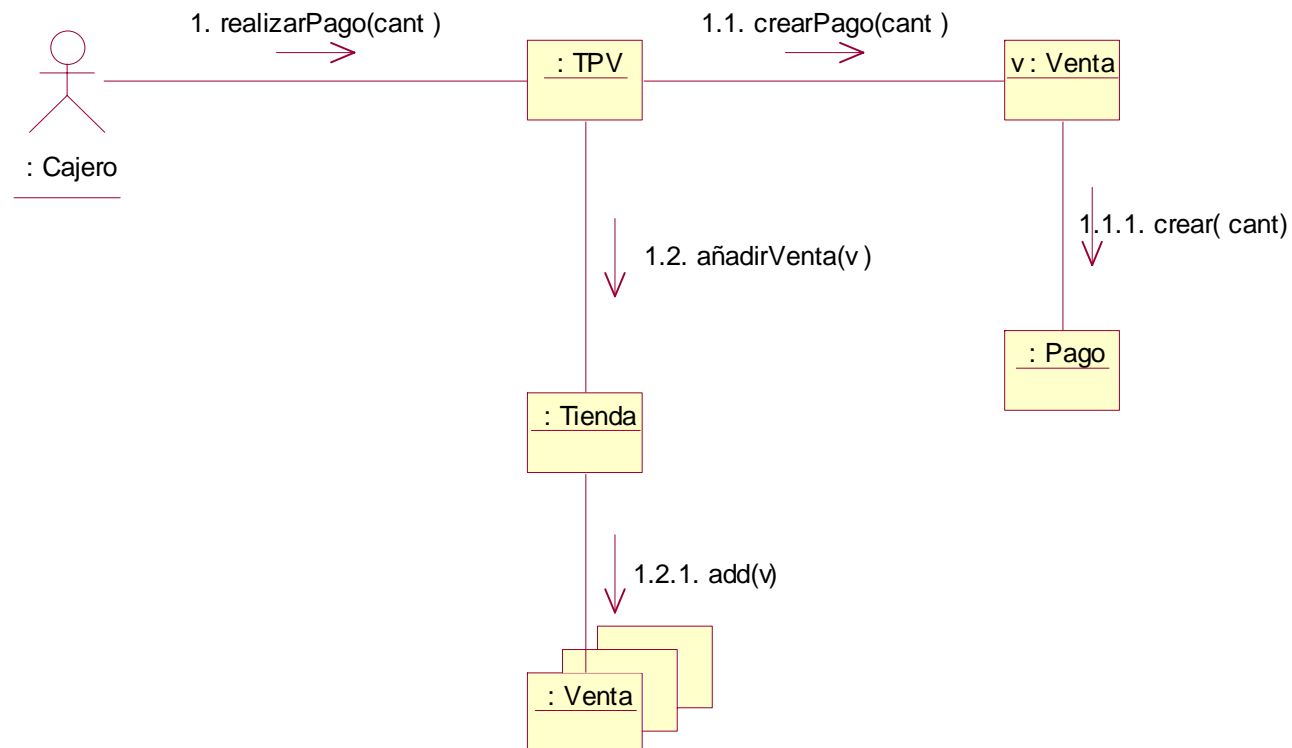
Postcondición:

una instancia de *Pago p* fue creada

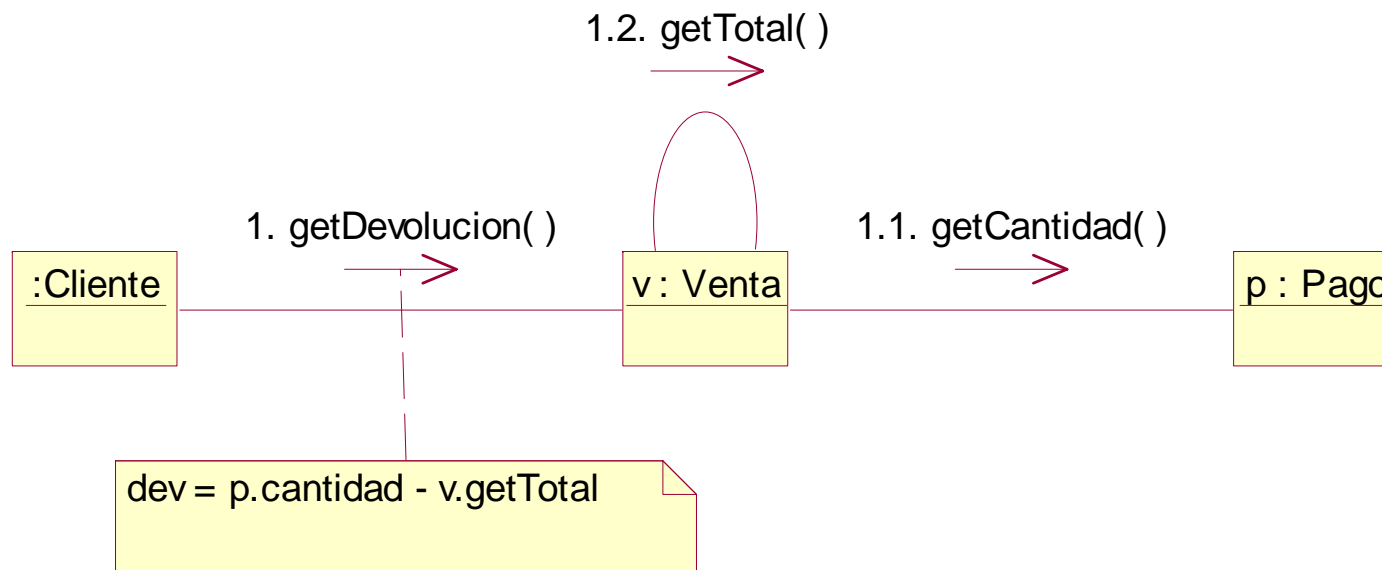
p.cantidadEntregada = cantidad

p se asoció con la venta actual

se asoció la venta actual con *Tienda* (para registrarla)



“el sistema imprime el recibo con la cantidad a devolver”



Operación: *Inicio*

Postcondición:

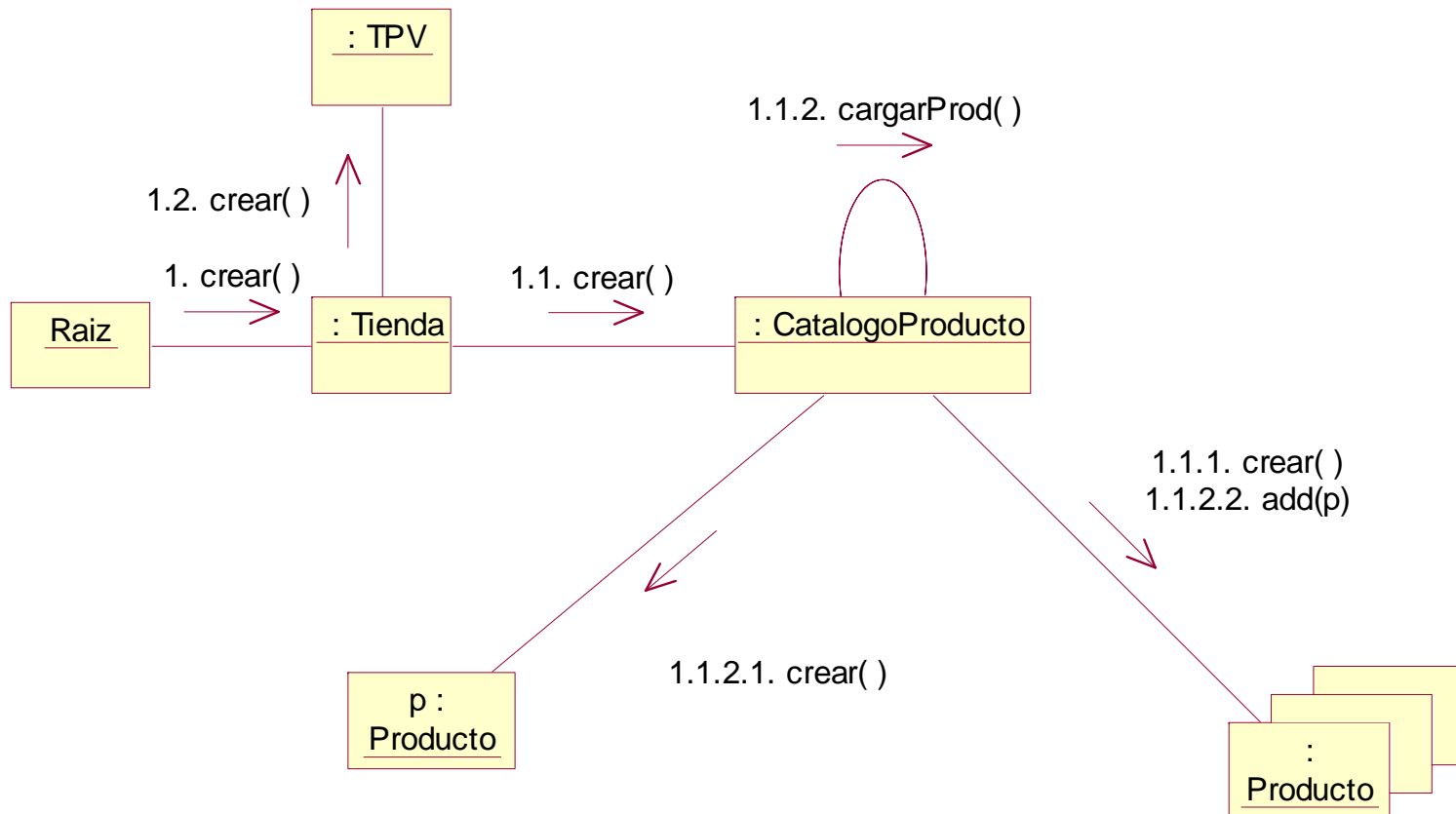
Una instancia de *Tienda*, *TPV* y *CatalogoProductos* fueron creadas.

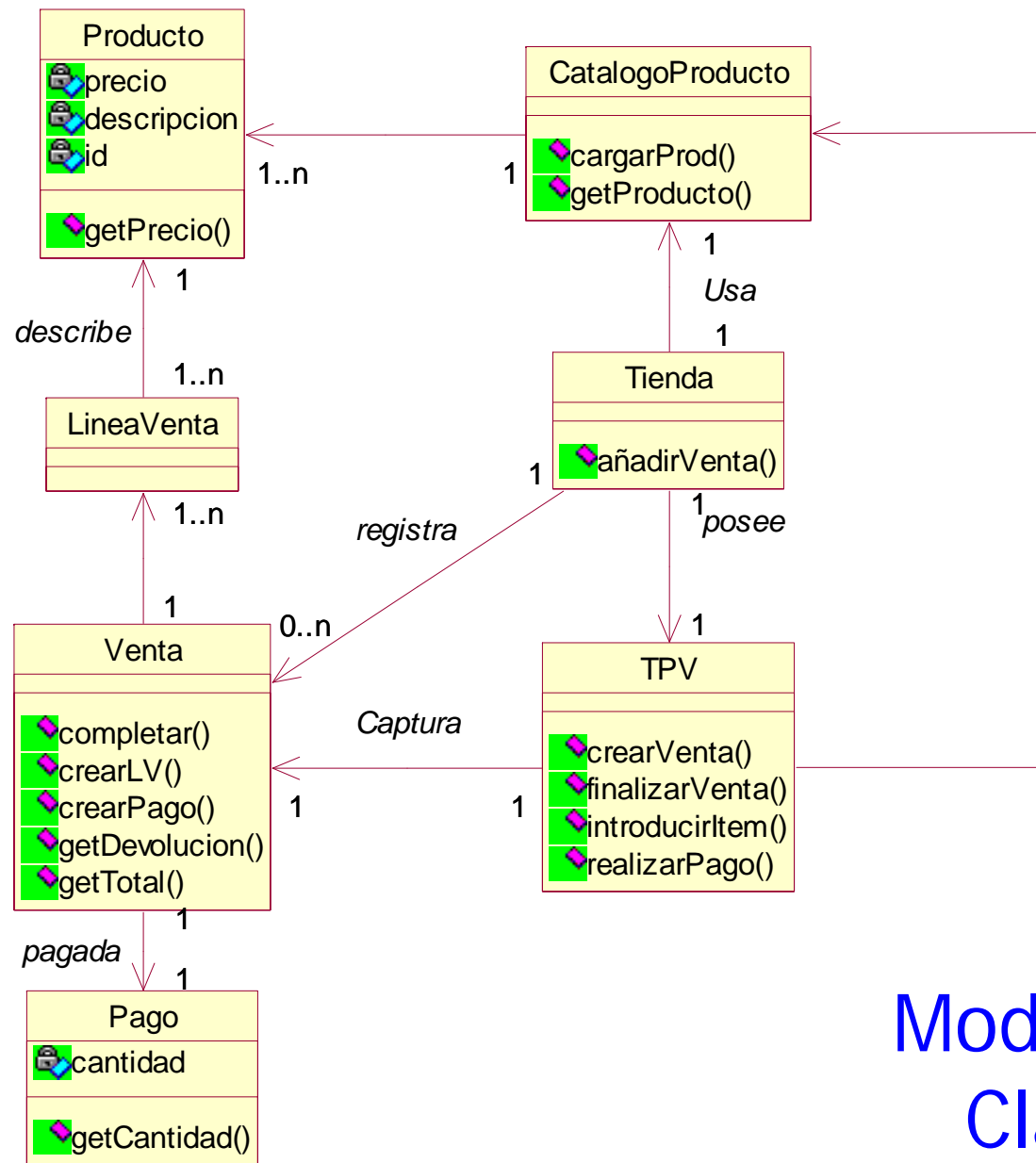
Instancias de *Producto* son creadas y asociadas a *CatalogoProductos*

Tienda se asocia a *CatalogoProductos*

Tienda se asocia a *TPV*

TPV se asocia a *CatalogoProductos*





Modelo de Clases

```

public class Producto {
    private ItemID id;
    private Dinero precio;
    private String descripcion;

    public Producto(ItemID id, Dinero precio, String desc) {
        this.id = id;
        this.precio = precio;
        this.descripcion = desc;}
    public ItemId getId() { return id; }
    public Dinero getPrecio() { return precio; }
    public String getDescripcion() { return descripcion; }
}

public class CatalogoProducto {
    private Map productos = new HashMap ();

    public CatalogoProductos () {
        ItemId id1 = new ItemID(100);
        ItemId id2 = new ItemID(200);
        Dinero precio1 = new Dinero (3);
        Dinero precio2 = new Dinero (5);
        Producto p;
        p:= new Producto (id1, precio1, "producto 1");
        productos.put(id1, p);}
        p = new Producto (id2, precio2, "producto 2");
        productos.put(id2, p);}

    public Producto getProducto (ItemId id) { return (Producto) productos.get(id); }
}

```

```
public class TPV {  
    private CatalogoProducto catalogo;  
    private Venta venta;  
  
    public TPV(CatalogoProducto cp) { catalogo = cp; }  
    public void crearNuevaVenta () { venta = new Venta(); }  
    public void finalizarVenta () { venta.completar(); }  
    public void introducirItem (ItemId id, int cant) {  
        Producto p = catalogo.getProducto (id);  
        Venta.crearLineaVenta(p, cant); }  
    public void realizarPago() { venta.crearPago(cant) }  
}
```

```
public class Pago {  
    private Dinero cantEntregada;  
  
    public Pago (Dinero cantidad) { cantEntregada = cantidad; }  
    public Dinero getCantEntregada () { return cantEntregada; }  
}
```

```
public class Venta {  
    private List lineaVentas = new ArrayList();  
    private Date fecha = new Date();  
    private boolean esCompleta;  
    private Pago pago;  
  
    public Dinero getDevolucion() { return pago.getCantEntregada(). minus(getTotal() ); }  
    public void completar() { esCompleta = true; }  
    public void crearLineaVenta(Producto p, int cant) {  
        lineaVentas.add(new LineaVenta(p,cant)); }  
    public Dinero getTotal() {  
        Dinero total = new Dinero();  
        Iterator i = lineaVentas.iterator();  
        while (i.hasNext()) {  
            LineaVenta lv = (LineaVenta) i.next();  
            total.add(lv.getSubtotal()); }  
        return total;  
    }  
    public void crearPago (Dinero cantEntregada) { pago = new Pago(cantEntregada); }  
}
```

```
public class LineaVenta {  
    private int cantidad;  
    private Producto producto;  
  
    public LineaVenta(Producto p, int cant) {  
        this.producto = p;  
        this.cantidad = cant; }  
    public Dinero getSubtotal () { return producto.getPrecio().times(cantidad); }  
}
```

```
public class Tienda {  
    private CatalogoProducto catalogo;  
    private TPV tpv;  
  
    public TPV getTPV { return TPV; }  
}
```

Validación del sistema

- Utilizar los casos de uso.
- Para cada caso de uso comprobar que el sistema muestra el comportamiento esperado, considerando el escenario principal y los excepcionales o alternativos.
- Considerar requisitos no funcionales.

Programación “Prueba primero”

- Práctica promovida por XP
- Ciclo “*escribo código de prueba, escribo código de producción, pruebo*”
- Ventajas
 - ¡Se escriben las pruebas!
 - Satisfacción del programador: ¡He superado la prueba!
 - Ayudan a comprender mejor las interfaces y comportamiento
 - Verificación de la corrección
 - No hay miedo a los cambios: ¡existen cientos de pruebas de unidad!

Programación “Prueba primero”

- Antes de escribir el método `getTotal()` en la clase `Venta`, escribimos un método de prueba de unidad en una clase `PruebaVenta` que
 - Crea una venta
 - Le añade varias líneas de venta
 - Obtiene el total y comprueba si tiene el valor esperado
- Luego escribimos el método `getTotal()` y realizamos la prueba.
- Junit es un *framework open source* para pruebas de unidad para código Java (www.junit.org).

Programación “Prueba primero”

```
class PruebaVenta extends TestCase {  
    public void pruebaTotal() {  
        Dinero total = new Dinero (7.5);  
        Dinero precio = new Dinero (2.5);  
        ItemID id = new ItemId (1);  
        Producto p = new Producto (id, precio, “producto 1”);  
        Venta venta = new Venta ();  
        venta. crearLineaVenta(p,1);  
        venta. crearLineaVenta(p,2);  
  
        assertEquals(venta.getTotal(), total);  
    }  
}
```

Arquitectura de tres capas

- Presentación
- Lógica de la Aplicación
- Almacenamiento
- Principio de **Separación Modelo-Vista**
 - Los objetos del modelo o dominio no conocen directamente a los objetos de la vista o presentación

Separación Modelo-Vista

- Los objetos del modelo (dominio) no deben conocer directamente a los objetos de la vista (presentación).
- Las clases del dominio encapsulan la información y el comportamiento relacionado con la lógica de la aplicación.
- Las clases de la interfaz (ventanas) son responsables de la entrada y salida, capturando los eventos, pero no encapsulan funcionalidad de la aplicación.

Separación Modelo-Vista

- Justificación
 - Clases cohesivas
 - Permitir separar el desarrollo de las clases de la vista y del dominio
 - Minimizar el impacto de los cambios en la interfaz sobre las clases del modelo.
 - Facilitar conectar otras vistas a una capa del dominio existente.
 - Permitir varias vistas simultáneas sobre un mismo modelo.
 - Permitir que la capa del modelo se ejecute de manera independiente a la capa de presentación.