4.2 UML: diagramas de diseño

Ingeniería del Software Avanzada
Técnicas de análisis y diseño

Objetivos

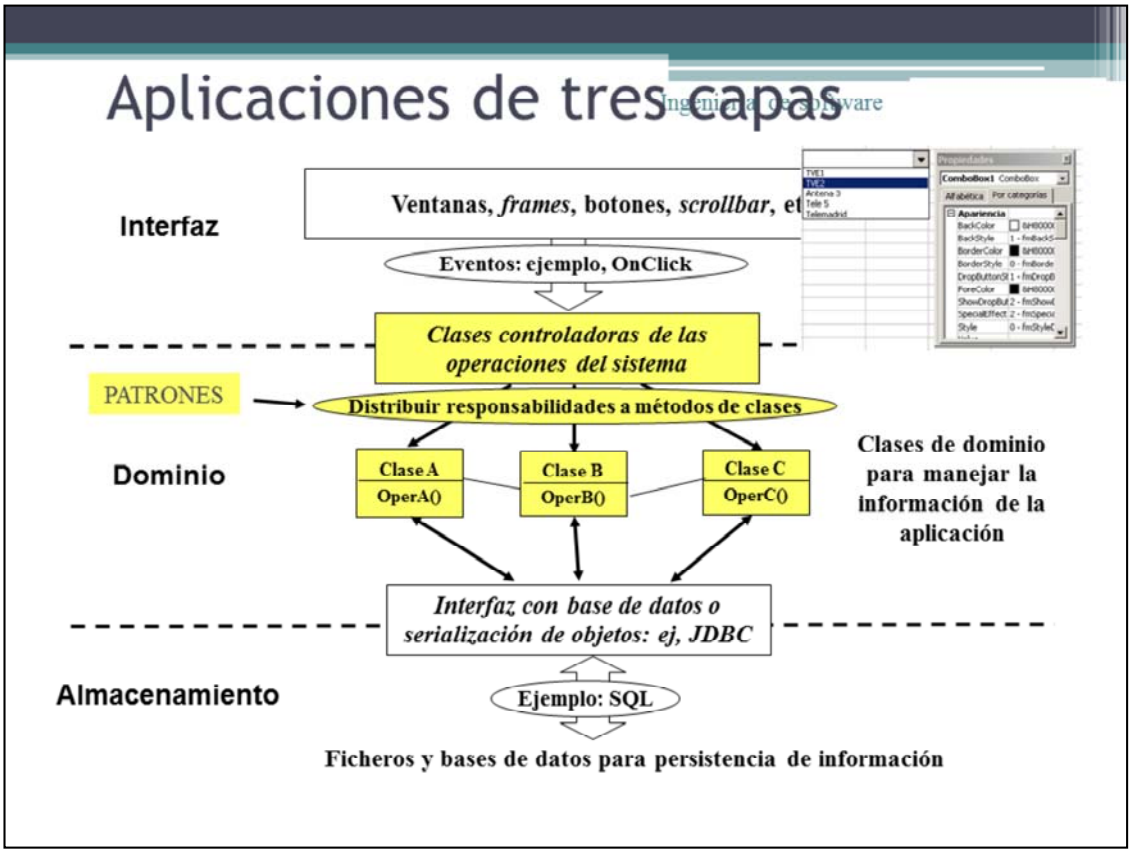
- Comprender la notación de los diagramas de colaboración
- Ser capaces de generar un diagrama de colaboración para cada caso de uso a partir de la información de la fase de análisis
- Conocer los patrones y cómo aplicarlos a los diagramas de colaboración
- Conocer las nociones de visibilidad, dependencias, etc. y aplicar las consecuencias de los diagramas de colaboración a la creación de un diagrama de clases de diseño

Fase de diseño

- Tres actividades básicas:
  - Crear diagramas de colaboración o secuencia para casos de uso u operaciones
    - A partir del contrato y de diagrama de secuencia del sistema
    - Aplicar patrones GRASP y otros
    - Principalmente para la capa de dominio
  - Crear el modelo de clases de diseño:
    - Actualizando el modelo conceptual con métodos, nuevas clases, especificando visibilidad, etc.
  - Documentar la arquitectura del sistema
    - Diagramas de despliegue y componentes

Arquitectura

- Asumimos arquitectura de 3 capas:
  - Presentación: ventanas, formularios, etc.
  - Lógica de aplicación o dominio: reglas de negocio
  - Almacenamiento: persistente
- Aislar la lógica en n-capas
  - Objetos de dominio: conceptos del problema
  - Servicios: funciones de interacción de BD, comunicación,...
- Distintas configuraciones Cliente/Servidor
- Ventajas:
  - Aislar la lógica, distribución en varios nodos, asignar tareas de diseño por capas
  - Apoyo en patrón separación modelo-vista



Java Database Connectivity, más conocida por sus siglas **JDBC**[1], es una API que permite la ejecución de operaciones sobre BD desde JAVA, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice

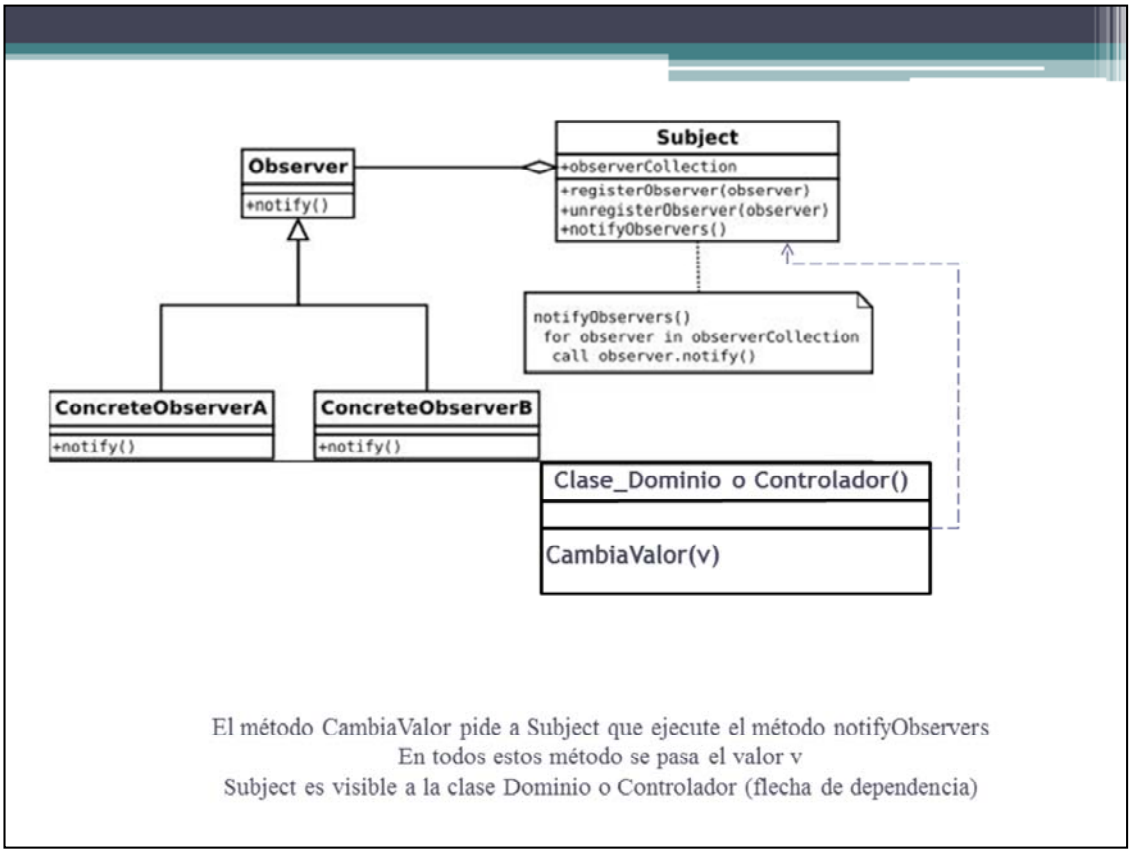
Patrón: separación modelo-vista

- Problema:
  - Desacoplar objetos del dominio y de vistas (interfaz GUI), para mayor reutilización y aislar los cambios en interfaz
- Solución:
  - Definir clase de dominio que no tengan acoplamiento ni visibilidad directa con la vista: conservar datos de dominio en clases de aplicación y no en ventanas
    - Clases de dominio: no conocerán existencia de ventanas
    - Clases de presentación (interfaz): tienen acceso a dominio
    - No se enviarán datos ni mensajes a ventanas: sólo dar respuestas
    - En ocasiones (control, simulación), habrá necesidad de comunicación indirecta

IGU = interfaz gráfica de usuario

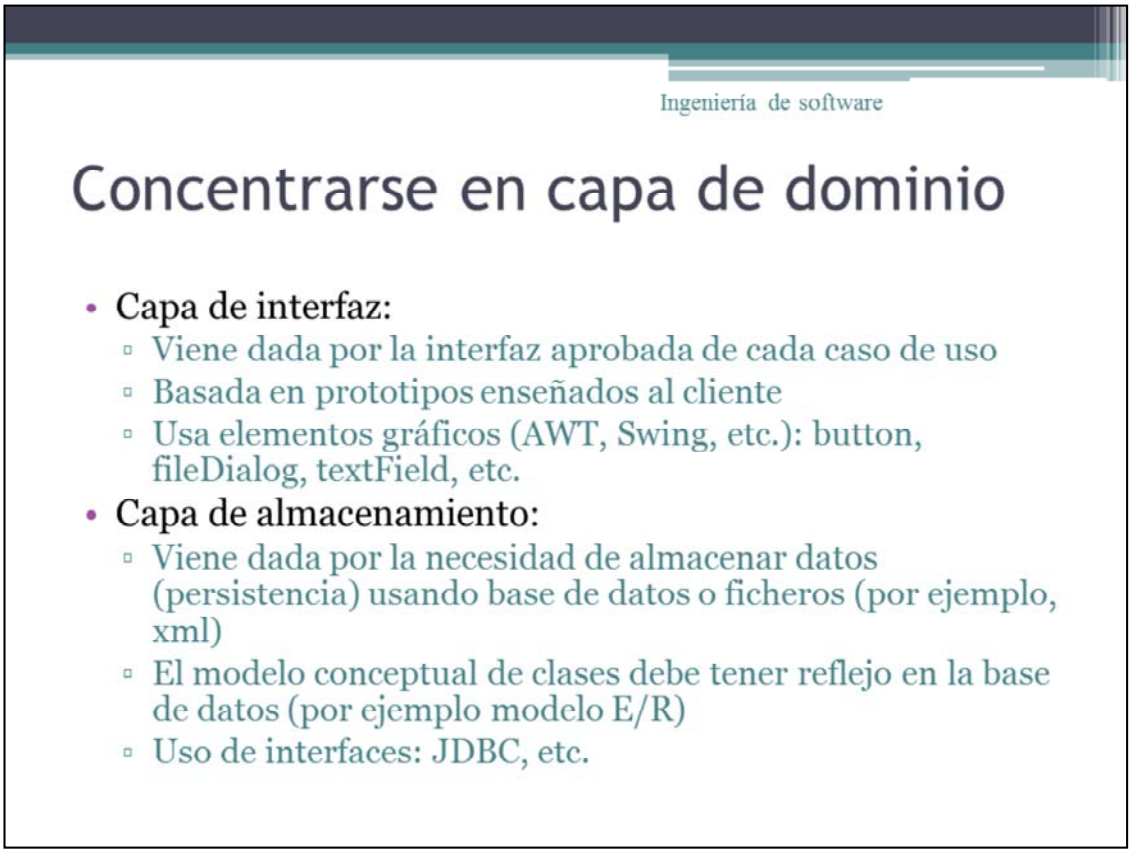
Comunicación indirecta

- Recurrir al patrón observador (publicar-suscribir)
- Problema
  - Un cambio de estado (evento) ocurre dentro de un editor de evento y otros objetos están interesados, pero el editor no debería tener conocimiento de suscriptores
- Solución
  - Definir un sistema de notificación indirecta de eventos
    - Las ventanas se suscriben
  - Crear una clase de gestión de eventos
    - No requiere acoplamiento directo
    - Puede transmitirse a cualquier número de suscriptores



http://naldog.blogspot.com.es/2013/02/patron-observer-en-c-para-chilean-people.html

Visibilidad global, ya que Subject es una clase estática (solo tiene una instancia) y se instancia globalmente, no en el método CambiaValor, pero en el método CambiaValor llamamos a Vsubject.notifyObservers(v)



La Abstract Window Toolkit es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java. AWT es ahora parte de las Java Foundation Classes (JFC) - la API estándar para suministrar una interfaz gráfica de usuario (GUI) para un programa Java.

Swing es una biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas.

la serialización consiste en un proceso de codificación de un objeto en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria) con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible como XML.

Diagramas de colaboración

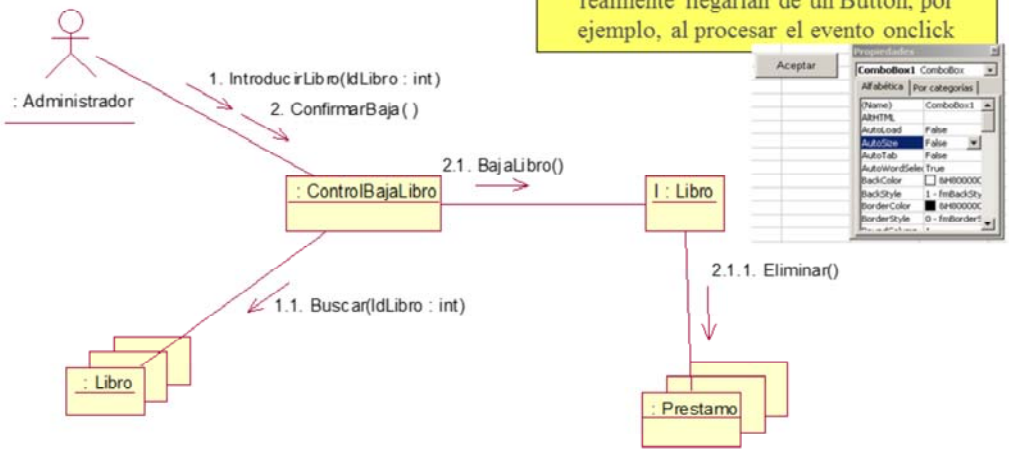
- Muestran la interacción entre objetos
  - Incluye las relaciones entre objetos
  - No indica tiempo: sólo números de secuencia
- Como en diagrama de secuencia:
  - Afecta a subconjunto del modelo de clases: colaboración
    - Conjunto de clases y sus relaciones implicados en una acción
    - Marca el contexto de una interacción
- Colaboración:
  - Diagramas de objetos (clases) con mensajes en las asociaciones

(Notación, pp. 89-90)

Notación (I)

Diagrama para un caso de uso

Obviamos la capa de interfaz: los mensajes del exterior aparecen como directamente enviados por el actor aunque realmente llegarían de un Button, por ejemplo, al procesar el evento onclick

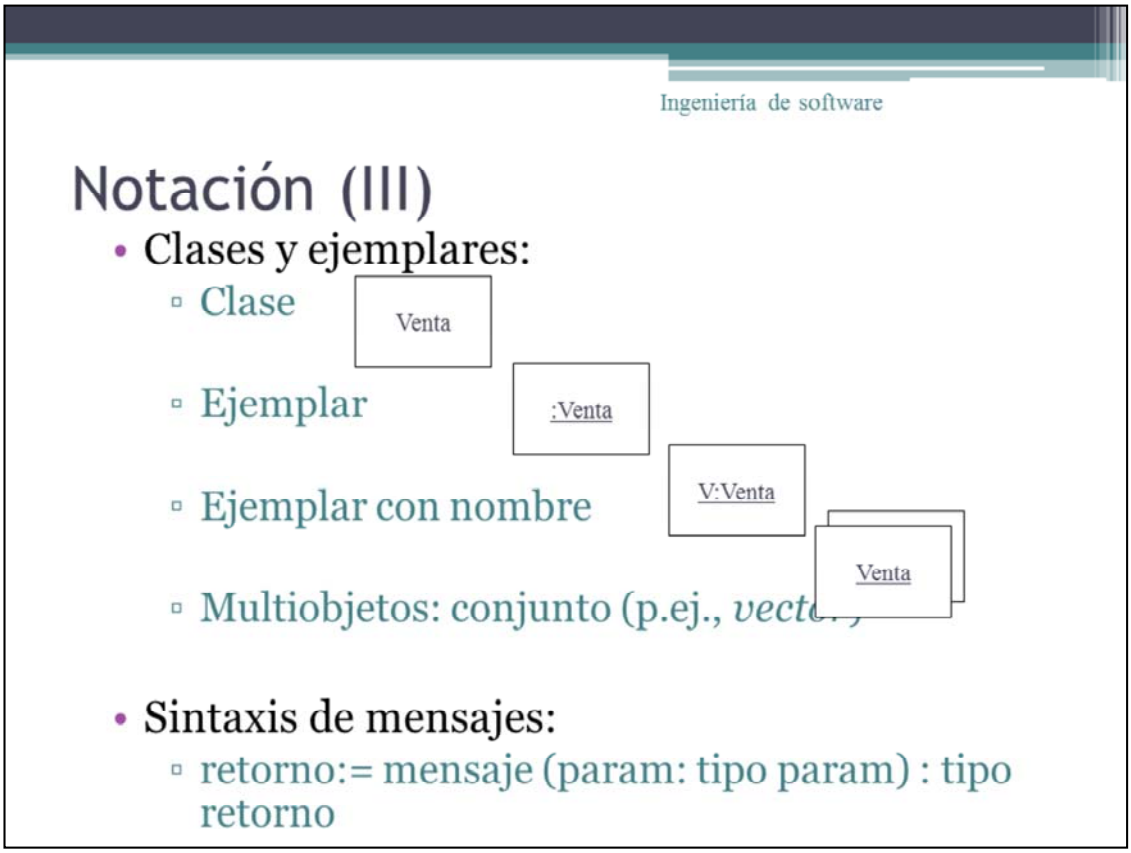


Notación (II)

- Diagrama representa vínculos entre objetos
  - Incluye vínculos temporales
    - Paso de parámetros, variables locales
  - Los vínculos pueden incluir flecha de navegabilidad
- Mensajes
  - En flecha junto a línea de unión con nombre y parámetros
  - Añaden número de secuencia comenzando en el 1
    - En flujo procedimental, los números se anidan
  - Iteración: * [i:=1..n]; condición: [x > 0] (mismo número de secuencia)
- Consecuencia de mensajes sobre objetos:
  - Creado {new}, destruido{destroyed}, creado y destruido {transient}
- Aparecen todos los objetos implicados en ejecución
  - Incluso los afectados indirectamente o sólo accedidos

(Notación, pp. 89-90)

(Booch et al., 1999, p. 216)



Una clase estática es básicamente igual que una clase no estática, pero existe una diferencia: no se pueden crear instancias de una clase estática. En otras palabras, no puede utilizar la palabra clave new para crear una variable del tipo clase (no tendrá ejemplares)

Una clase estática se puede utilizar como un contenedor adecuado para los conjuntos de métodos que solo funcionan en parámetros de entrada y no tienen que obtener ni establecer campos internos de instancia. Por ejemplo, en la biblioteca de clases .NET Framework, la clase estática System.Math contiene varios métodos que realizan operaciones matemáticas, sin ningún requisito para almacenar o recuperar datos único de una instancia determinada de la clase Math.

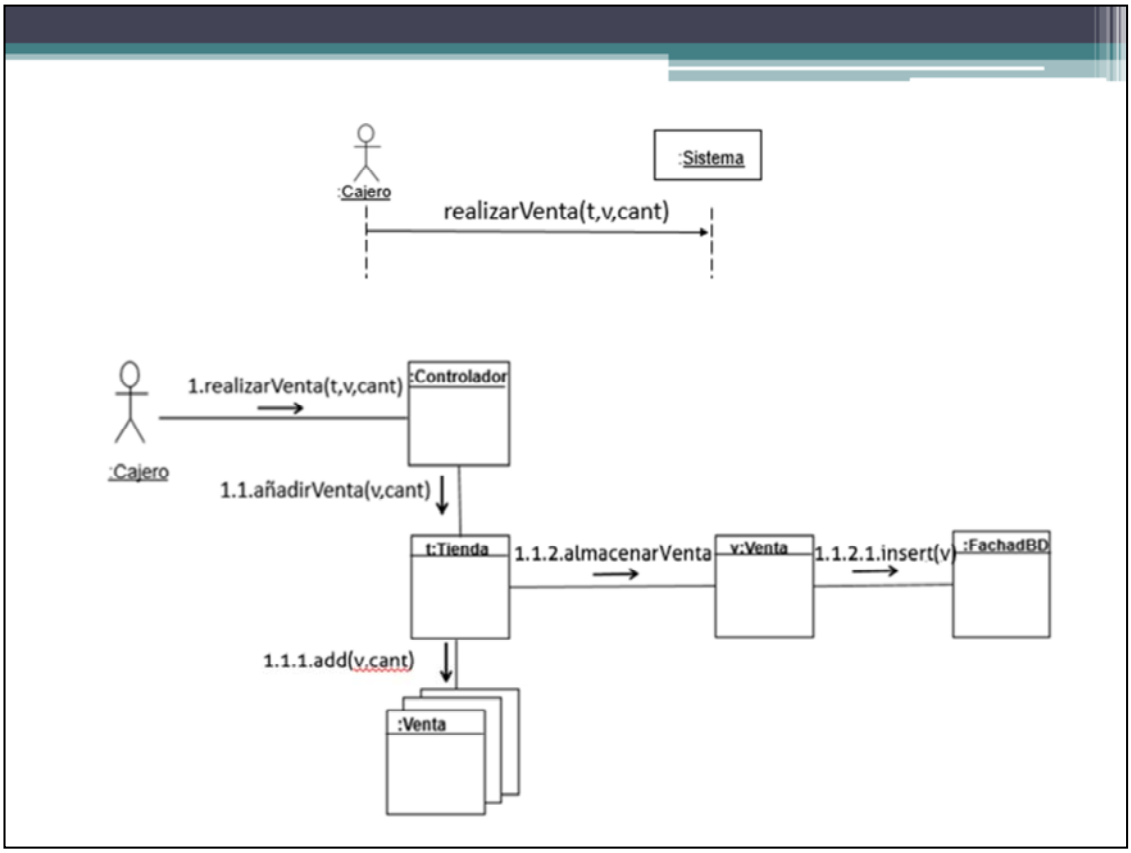
Crear diag. de colaboración

Información de partida:

•Diagrama de secuencia del sistema; Modelo conceptual de clases; Contrato de operación.

•También iniciar con descripciones de casos de uso: A partir de ellas, generar diagrama de secuencia y contrato

- Pasos:
  - Dibujar el actor (Rational: arrastrar)
  - Incorporar una clase de control específica para el caso
    - Por ejemplo, para "Alta de libro" poner "ControlAltaLibro"
  - Incorporar los mensajes del diagrama de secuencia desde el actor a la clase de control:
    - Los que deban ser tratados en la capa de dominio
  - Asignar respuesta a mensajes a las clases:
    - Analizando en orden de ejecución las acciones a realizar
    - Cuidado con búsquedas y tratamiento de asociaciones
  - Representar acciones como mensajes
  - Revisar diagrama para asegurar cumplimiento de patrones


:Cajero
:Sistema
realizarVenta(t,v,cant)
:Cajero
1.realizarVenta(t,v,cant)
:Controlador
1.1.añadirVenta(v,cant)
t:Tienda
1.1.2.almacenarVenta
v:Venta
1.1.2.1.insert(v)
:FachadBD
1.1.1.add(v,cant)
:Venta

Patrones

- Principios generales y guías de desarrollo
- Patrón:
  - Descripción de un problema y su solución
  - Nombre propio: facilita la comunicación
  - "Pareja problema/solución con un nombre y aplicable a otros contextos y que sugiere cómo usarlo en situaciones nuevas"
- Los patrones no suelen contener ideas nuevas
  - Solo formalizar la experiencia de desarrollo

(Larman, 1999, pp.189-190)

Patrones GRASP

- Muchos patrones
  - Continuamente se proponen más
- Patrones GRASP
  - *General Responsibility Asignment Software Patterns*
  - Propuestos por C.Larman
  - Aplicables a asignación de responsabilidades para clases, al dibujar diagramas de colaboración
- GRASP:
  - Experto, Creador, Gran Cohesión, Bajo Acoplamiento, Controlador
    - Gran cohesión: influye en experto y controlador
    - Bajo acoplamiento: influye en creador

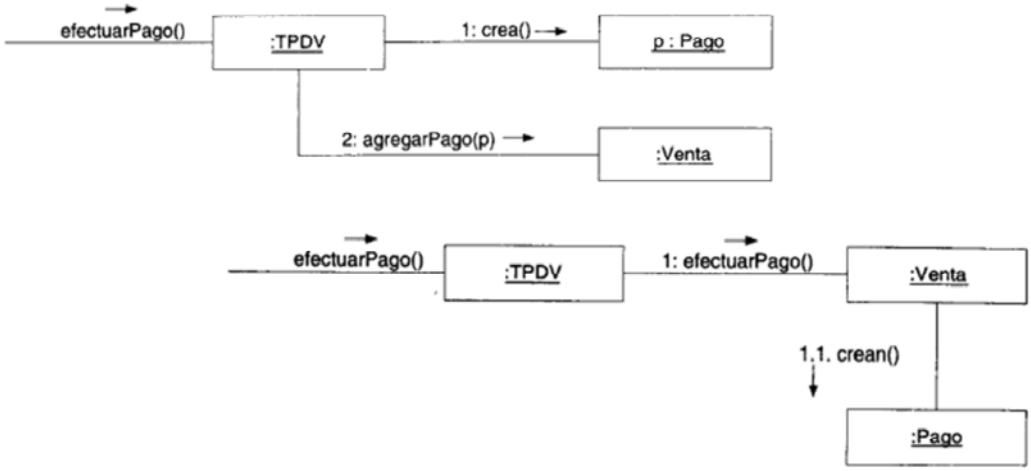
(Larman, 1999, pp.189-190)

Metapatrones

- Metapatrones: guías generales de diseño
- Bajo acomplamiento: relación entre clases
  - X tiene atributo de ejemplar de Y o clase Y; X tiene operación que referencia (parámetro) a Y o clase Y; X es subclase directa o indirecta de Y; Y es una interfaz y X lo implementa
- Problemas de acoplamiento:
  - sensible a cambios, menos entendibles, etc.
- Alta cohesión: coherencia de funciones de una clase
  - Ideal: una función por clase
  - Deseable: varias funciones similares (usan mismos datos o hacen cosas similares)

Bajo acoplamiento y Alta cohesión: clases con poca visibilidad y con pocas funciones y relacionadas entre sí

Tenemos que crear una instancia de Pago y asociarla a la Venta. Venta ya tiene visibilidad de Pago por otros casos de uso:
En el primer diagrama TPDV ve dos clases, en el segundo solo una (acoplamiento)
En el primero TPDV realiza dos funciones (crea Pago y lo agrega a Venta), en el segundo solo le pide a Venta que efectué un pago y Venta lo crea agregándoselo (cohesión)

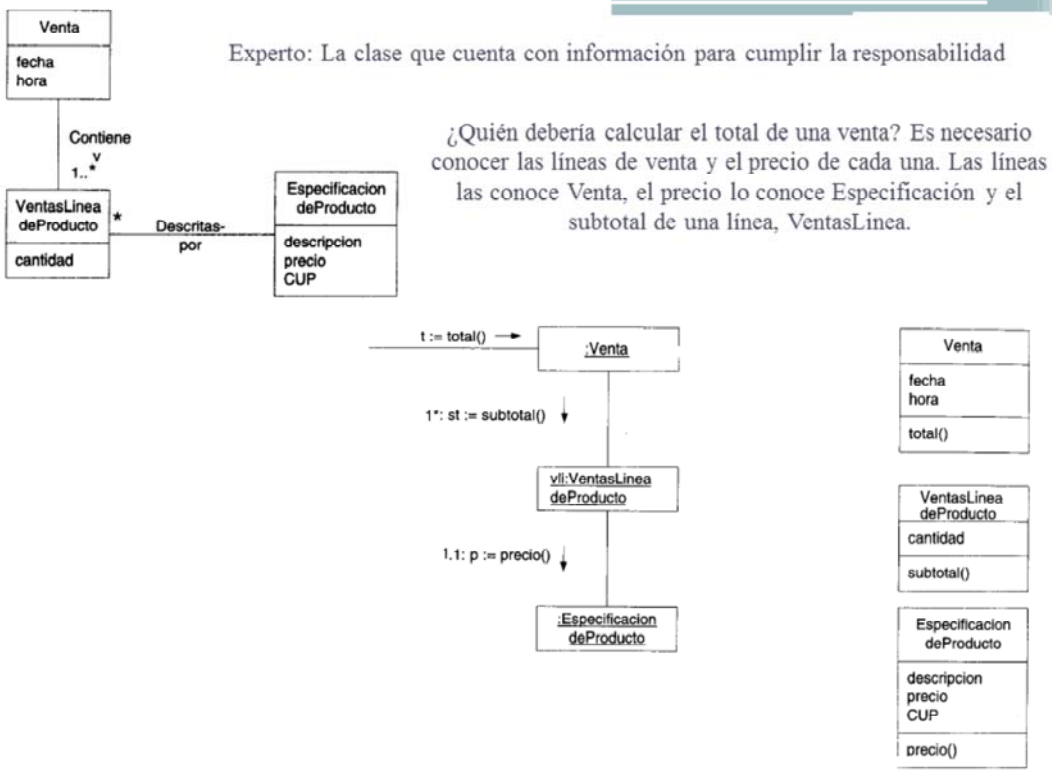


Experto

- Problema:
  - ¿Responsabilidades de manejo de información?
- Solución:
  - Asignar responsabilidad al experto en información
    - La clase que cuenta con información para cumplir responsabilidad
- Explicación:
  - Pueden existir muchos expertos parciales para colaborar
- Beneficios:
  - Bajo acoplamiento y gran cohesión

Experto: La clase que cuenta con información para cumplir la responsabilidad

¿Quién debería calcular el total de una venta? Es necesario conocer las líneas de venta y el precio de cada una. Las líneas las conoce Venta, el precio lo conoce Especificación y el subtotal de una línea, VentasLinea.

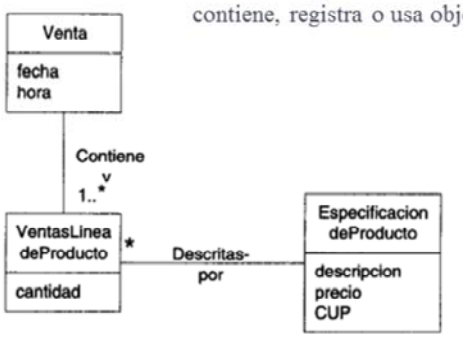


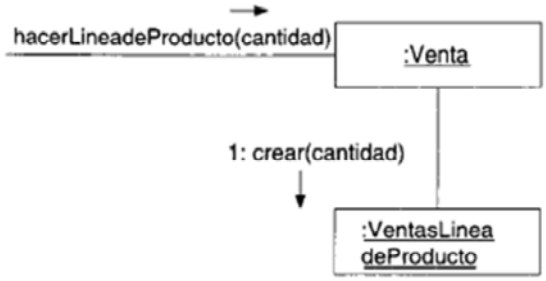
Creador

- Problema:
  - ¿Responsable de crear/borrar un ejemplar de alguna clase?
- Solución:
  - Asignar a B responsabilidad de crear ejemplares de A si:
    - B agrega objetos de A, B contiene objetos de A, B registra ejemplares de A, B usa objetos de A, B tiene datos de inicio de A
- Explicación:
  - Contenedor, Registrador, Agregador…crean contenido, registro, agregado
  - Se aplica también al borrado de objetos
- Beneficios:
  - Bajo acoplamiento: creado es visible ya al creador

Creador: Asignar a B responsabilidad de crear ejemplares de A si B agrega, contiene, registra o usa objetos de A

¿Quién debería crear Líneas de venta? Venta, ya que agrega objetos VentasLínea





Controlador

- Atención de eventos del sistema:
  - Capa de presentación no maneja eventos
  - Operaciones en capa de dominio
- Opciones para asignar una clase de control que:
  - Trate todos los eventos del sistema global o de la empresa
  - Trate todos los eventos generados por un actor
  - Trate eventos para cada caso (controlador de caso)
- Sugerencias:
  - Controladores que manejen un caso, para separar procesos
  - Controlador global: si hay pocos eventos
  - Controlador de actor: muchos eventos de varios procesos
- Problema de controladores saturados
  - Demasiados eventos, responsabilidades y demasiada complejidad

Controlador: la clase interfaz no ve a las clases de dominio, solo a la clase controlador

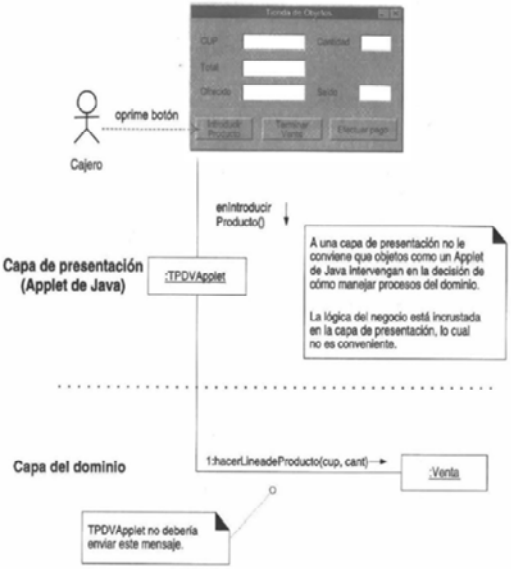


Figura 18.18 Acoplamiento inadecuado de la capa de presentación a la capa del dominio.

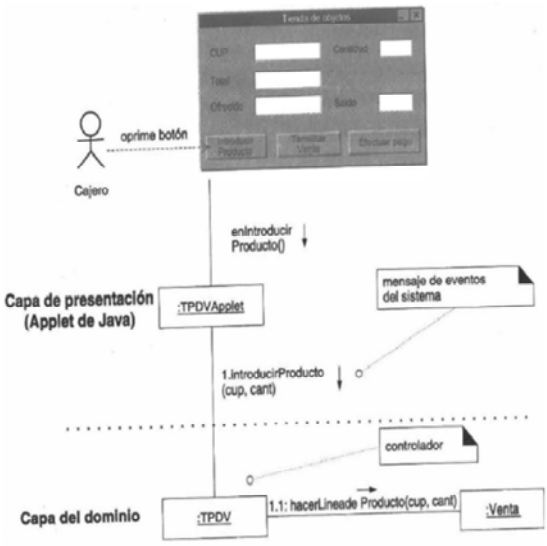


Figura 18.17 Acoplamiento adecuado de la capa de presentación a la capa del dominio.

Comando

- Patrón Comando: sistemas de manejo de mensajes
  - Aplicaciones que no cuentan con interfaz y reciben mensajes de un sistema externo
  - Comando: definir un controlador para todos los eventos
  - Definir una clase para cada mensaje, con método Ejecutar, y una superclase abstracta, también con método Ejecutar
  - Controlador instancia una clase Comando para cada mensaje y activa Ejecutar (método polimórfico)

En el ejemplo Interruptor es el Controlador

Comando: cada clase ejecuta un procedimiento diferente, pero el controlador instancia la clase concreta y ejecuta (por polimorfismo se ejecuta el método de la clase instanciada)

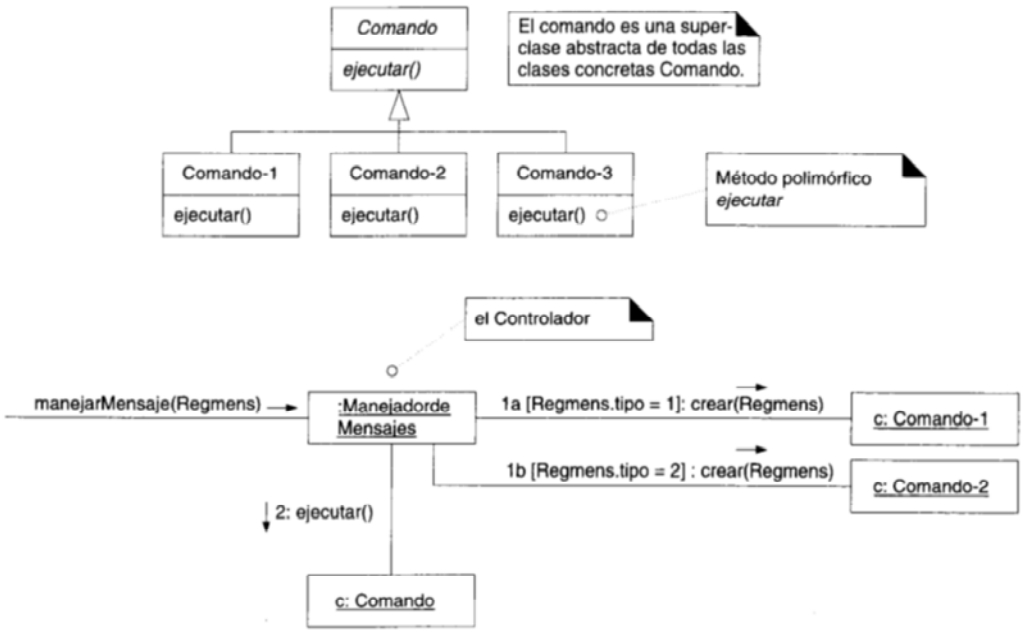


Diagrama de clases de diseño

- Exige crear antes:
  - Diagramas de interacción: identifican métodos y nuevas clases (por ejemplo: de control, de lista de objetos, etc.)
  - Modelo conceptual: a partir de él, se agregan detalles de definición de clases o incluso nuevas clases
  - Suelen crearse en paralelo a los diagramas de interacción
    - Herramientas como Rational lo hacen
- Deben incluir los detalles para pasar al código:
  - Clases, asociaciones y atributos
  - Interfaces, con operaciones y constantes
  - Métodos
  - Información sobre tipos de atributos
  - Navegabilidad
  - Dependencias

Cómo crear diagrama de diseño

- Identificar nuevas clases a partir de diagramas de interacción
- Dibujar en un diagrama
- Incluir atributos siguiendo el modelo conceptual
- Agregar nombres de métodos de diag. interacción
- Incorporar tipos a atributos y métodos
- Agregar asociaciones para visibilidad de atributos
- Agregar flechas de navegación (visibilidad)
- Agregar líneas de dependencias (visibilidad no de atributos: global o declarada localmente)

Algunos detalles

- Agregar nombres de métodos:
  - Incorporar métodos a las clases según mensajes de diagramas de interacción
  - Crear: ejemplar e inicialización
    - Java: *new* + constructor
  - Conviene incluir por defecto métodos de acceso:
    - Consultan valor de atributo (get) o establecen valor (set)
  - Sintaxis UML, independiente de lenguaje
    - Pero incorporar información de tipos

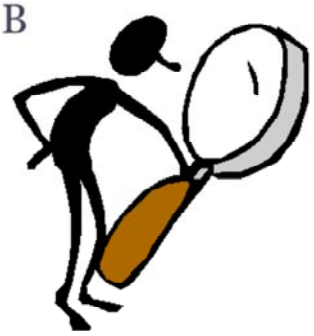
Analizar la visibilidad

- Capacidad de "ver" de un objeto a otro
  - Alcance o ámbito de objeto
- Cuatro tipos de visibilidad
  - Atributos
  - Parámetros
  - Declarada localmente
  - Global
- Para enviar mensajes a un objeto, debe ser visible
  - Lo visible o accesible para cada clase se debe determinar con los diagramas de colaboración

Visibilidad de atributos

- Visibilidad de atributos de A a B:
  - B es atributo de A
  - Relativamente permanente porque persiste mientras existan A y B

```
Public class A
{
...
Private B ObjB;
...
}
```



Visibilidad de parámetros

- Visibilidad de parámetros de A a B:
  - B se transmite como parámetro de método de A
  - Relativamente temporal porque persiste sólo dentro del ámbito del método

```
HacerMetodo (B b, int c)
{
...
}
```

  - Frecuentemente se transforma en visibilidad de atributos
    - El parámetro transmitido se utiliza en un momento de creación de ejemplares para ser asignado a un parámetro

Visibilidad declarada localmente

- Visibilidad declarada localmente de A a B
  - Se declara que B es objeto local dentro de un método de A
  - Es relativamente temporal porque sólo persiste en el ámbito del método
  - Dos medios habituales para la visibilidad:
    - Crear nuevo ejemplar local y asignarlo a variable local
    - Asignar variable local al objeto devuelto de la llamada a un método

```
HacerMetodo (int d, int c)
{
B b = obtener (d, c)
}
```

Visibilidad global

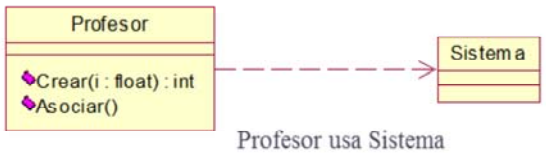
- Visibilidad global de A a B
  - Cuando B es global para A
  - Es relativamente permanente porque persiste mientras existan A y B
  - Medio más obvio:
    - Asignar ejemplar a variable global

Navegabilidad y dependencias

- Asociación con visibilidad de fuente a destino
  - Marca cómo implementar el atributo de asociación
  - Situaciones para incorporar flecha:
    - A envía mensaje a B
    - A crea ejemplar de B
  - Visibilidad de atributos: flecha normal de asociación y navegabilidad
- Dependencias
  - Dependencia general : conocer existencia de objeto
    - Por ejemplo la clase interfaz conoce a la clase controlador
  - Tipo: visibilidad de parámetro, global o declarada local

Dependencia

- Dependencia: relación de uso
  - Se identifican en el diseño (diagramas de colaboración)
  - Indica que un cambio en un elemento puede afectar a otro
  - Indica que un elemento usa otro:
    - Una clase que usa otra como parámetro de operación o variable declarada en operación
    - Va de la clase con operación a la que sirve de parámetro/variable



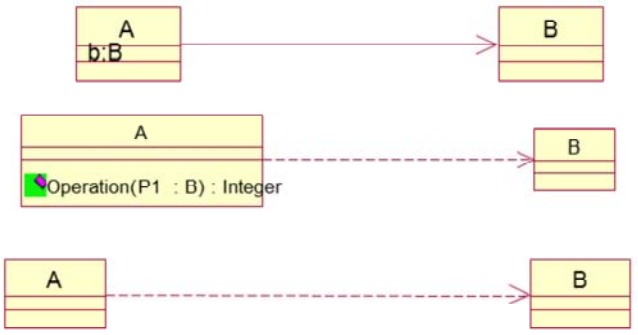
Profesor usa Sistema

(Eriksson y Penker, 1998, p. 56): cómo describir caso

(Booch et al., 1999, p. 195) Cómo describir flujo de eventos y ejemplo de cajero

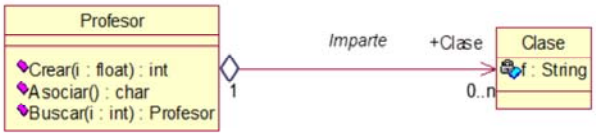
Navegabilidad

- Notación
- ¿Cuándo ocurre?:
  - Atributos
  - Parámetros
  - Local:
    - Metodo 1 (p1: p)
      - b: B



Efectos de implementación

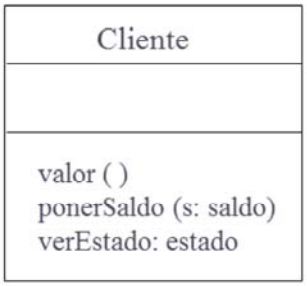
- **Implementar asociación con navegación**




```
public class Profesor
    public Clase Clase[];
    public int Crear(float i)
    public char Asociar()
    public Profesor Buscar(int i)
```

Documentar operaciones

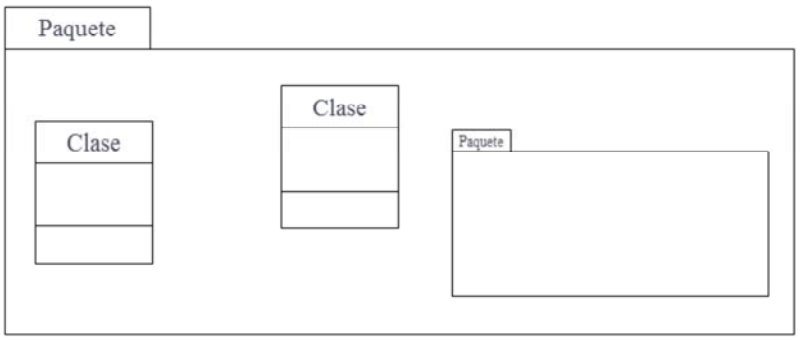
- Operaciones:
  - Implementación de un servicio que puede ser requerido a cualquier objeto
  - Una clase puede tener 0, 1 o varias operaciones
  - Especificación:
    - Nombre: si hay varias palabras, iniciales en mayúscula
    - Signatura: nombre, tipo y valores por defecto de todos los parámetros y, en el caso de funciones, un tipo de retorno



(Booch et al., 1999, pp. 44-45)

Organizar modelos: paquetes

- Permite grupos de elementos o subsistemas en UML
  - Define en un ámbito de nombres
- Notación

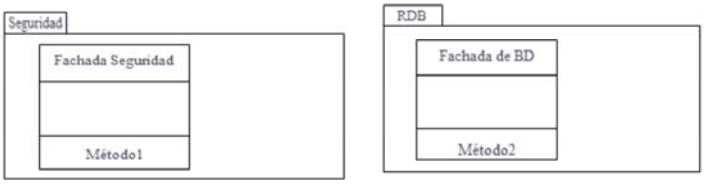


Paquetes y capas

- Una forma de utilizar paquetes
  - Aislar cada capa en un paquete
  - También agrupar elementos para un servicio común
  - Buscar gran cohesión y bajo acoplamiento
- Visibilidad entre paquetes:
  - Acceso a dominio: clases de interfaz tienen visibilidad de controladores
  - Acceso a servicios: limitar acceso a una o pocas clases (fachada)
- Otros patrones relacionados:
  - Fachada y Separación modelo-vista

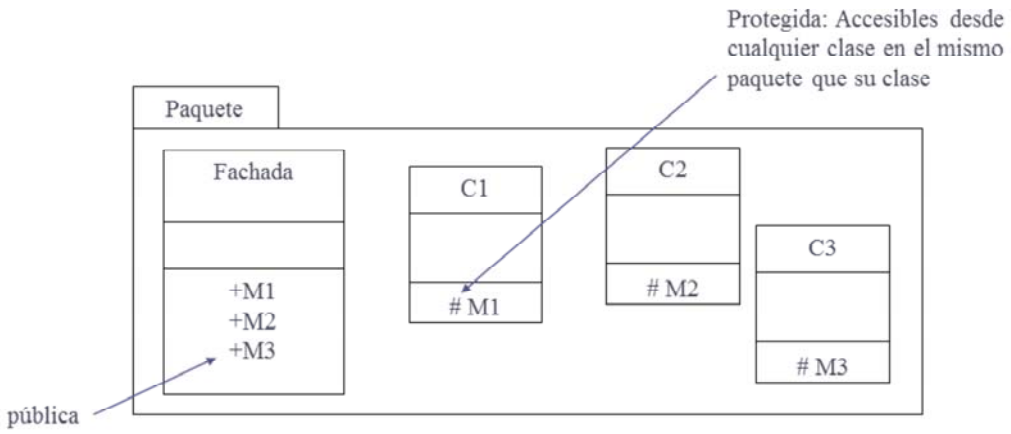
Patrón adicional: fachada

- Problema:
  - Se requiere una interfaz unificada y común para un conjunto heterogéneo de interfaces
- Solución:
  - Definir una clase individual que unifique la interfaz y que se responsabilice de colaborar con clases de otros paquetes y con clases privadas del mismo paquete



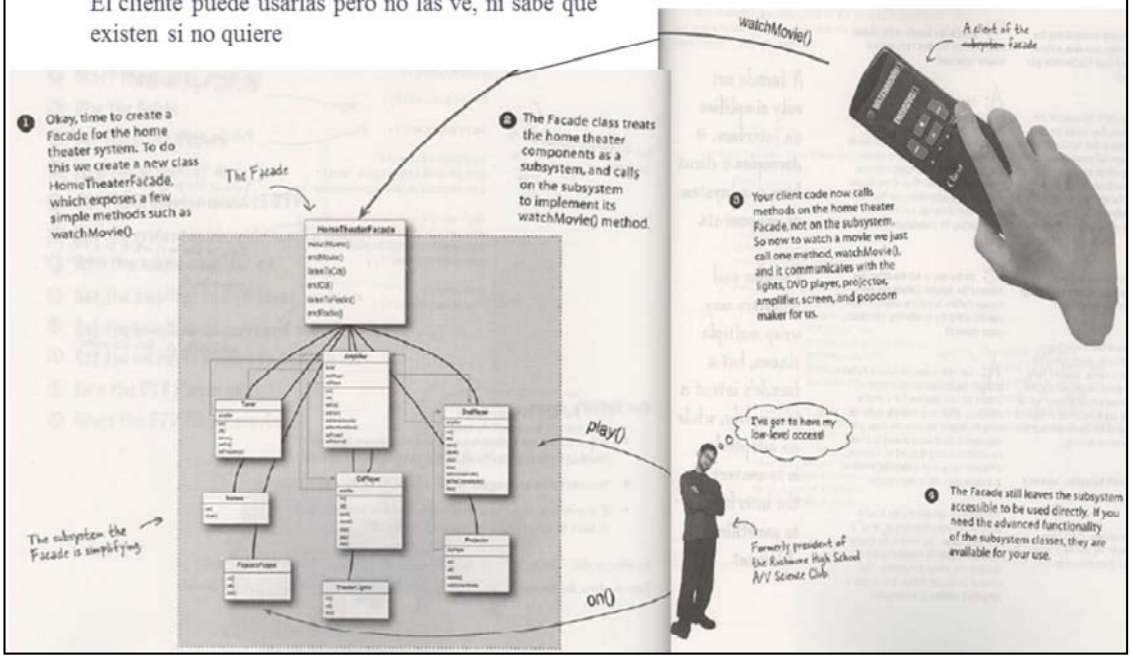
Fachada

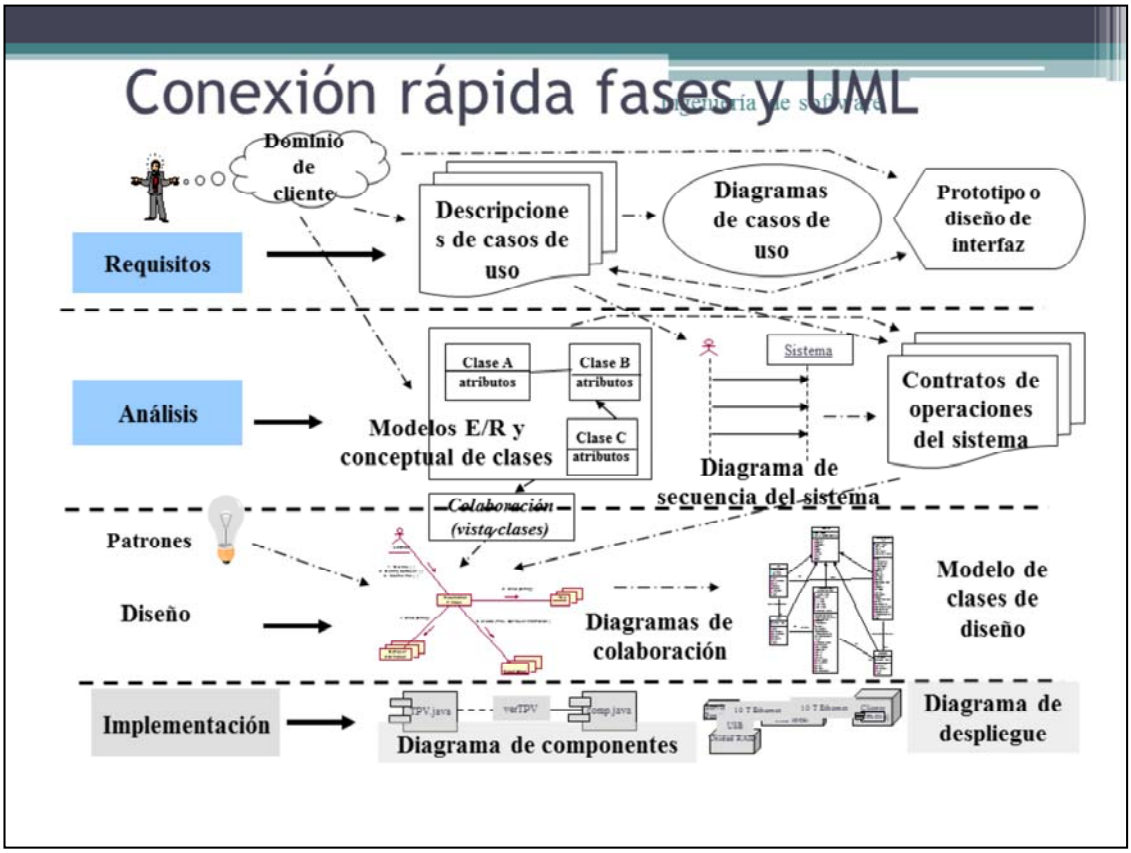
- Coordinado con la visibilidad de paquete



Fachada – Una clase que contiene métodos que combinan o llaman a métodos de otras clases

La clase fachada tiene visibilidad de las otras clases
El cliente puede usarlas pero no las ve, ni sabe que existen si no quiere




Conexión rápida fases y UML
Ingeniería de software
Dominio de cliente
Requisitos
Descripciones de casos de uso
Diagramas de casos de uso
Prototipo o diseño de interfaz
Análisis
Clase A
atributos
Clase B
atributos
Clase C
atributos
Modelos E/R y conceptual de clases
Sistema
Contratos de operaciones del sistema
Diagrama de secuencia del sistema
Colaboración (vista clases)
Patrones
Diseño
Diagramas de colaboración
Modelo de clases de diseño
Implementación
Diagrama de componentes
Diagrama de despliegue

Referencias

- Libros:
  - E. Freeman et al., “Head First Design Patterns”. O'Reilly Media, 2004

Tests

1) Señalar las frases correctas:

a) El patrón controlador sugiere una clase de control para cada acción de creación de objetos

→ b) El patrón creador permite lograr un menor acoplamiento

c) El patrón experto evita asignar la responsabilidad a clases que ya contienen la información que se necesita

d) Ninguna de las anteriores

2) Indicar los tipos de visibilidad entre dos clases:

- De atributos
- De parámetros
- Visibilidad local
- Visibilidad global