

Diseño

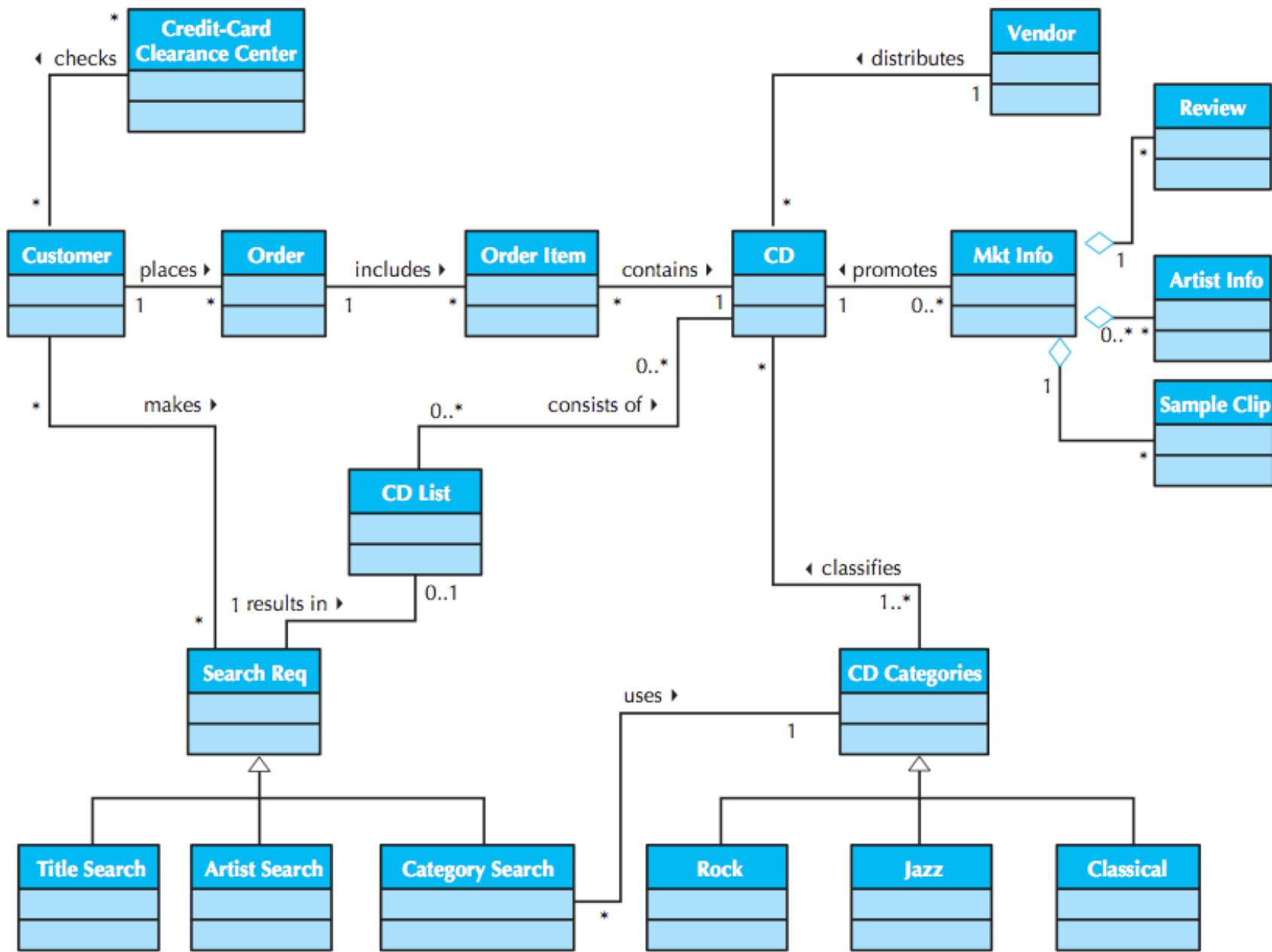
Análisis vs Diseño

- En modelo de Cascada tradicional luego de levantamiento de requisitos viene una etapa de análisis
- La etapa de diseño vendría después de concluir la etapa de análisis
- Bajo enfoque ágil análisis y diseño no tienen una separación tan nítida
- El paso de análisis a diseño es suave y casi imperceptible
- En ambos casos lo que se busca es construir un modelo
- modelo de análisis se parece mucho a un modelo de datos
- modelo de diseño incluye aspectos relacionados con el "como"

Modelo de Dominio (Análisis)

- Identificación de las clases de dominio
- Nivel de detalle en clases es mucho menor
- Por lo general se comienza con atributos (sin métodos)
- Modelo incluye clases de dominio y no objetos relacionados con implementación
- Se parece mucho a un modelo E-R pero pueden aparecer clases no persistentes e incluso algunos métodos

Ejemplo Modelo de Dominio



Descubrimiento de Objetos

- Buscar candidatos en
 - descripción de requisitos
 - casos de uso
 - historias de uso
- Buscar adicionalmente en
 - espacio general del problema
 - dispositivos físicos con los que interactúa
 - roles
 - eventos que deben ser registrados
 - ubicaciones geográficas
 - unidades organizacionales
- Algunos son objetos candidatos
- Algunos son solo atributos

Análisis del texto

- A common or improper noun implies a class of objects.
- A proper noun or direct reference implies an instance of a class.
- A collective noun implies a class of objects made up of groups of instances of another class.
- An adjective implies an attribute of an object.
- A doing verb implies an operation.
- A being verb implies a classification relationship between an object and its class.
- A having verb implies an aggregation or association relationship.
- A transitive verb implies an operation.
- An intransitive verb implies an exception.
- A predicate or descriptive verb phrase implies an operation.
- An adverb implies an attribute of a relationship or an operation.

Source: These guidelines are based on Russell J. Abbott, "Program Design by Informal English Descriptions," *Communications of the ACM* 26, no. 11 (1983): 882–894; Peter P-S Chen, "English Sentence Structure and Entity-Relationship Diagrams," *Information Sciences: An International Journal* 29, no. 2–3 (1983): 127–149; and Graham, *Migrating to Object Technology*.

Alice and the other student evaluated the instructor.

A stack **is a structure** where you can **push** or **pop elements** in a lifo manner.

This implementation of a stack **has an array** and an index

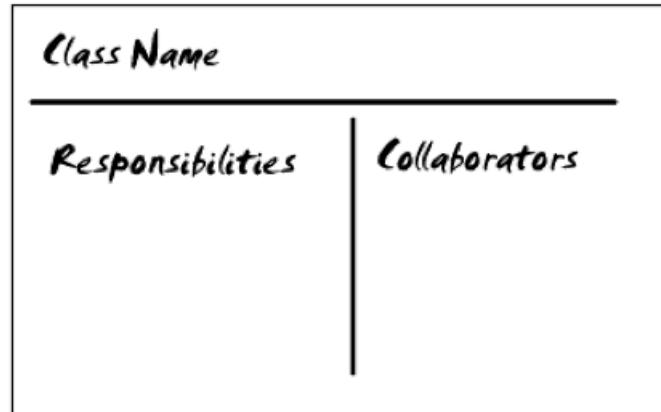
Evaluar Candidatos

- cada objeto debe guardar algún dato
- más de 1 atributo (o de lo contrario atributo)
- tiene métodos ?
- todas las instancias mismos atributos
- todas las instancias mismos métodos

Responsabilidades y Colaboradores

- Responsabilidades
 - qué es lo que el objeto sabe
 - qué es lo que el objeto es capaz de hacer
- Colaboradores
 - otras clases que participan

Uso de Tarjetas CRC



- permiten explorar las interacciones entre objetos
- CRC clase - responsabilidad - colaborador
- responsabilidades - que es lo que sabe y que hace con esa información
- colaboradores - clases que colaboran para lograr el propósito

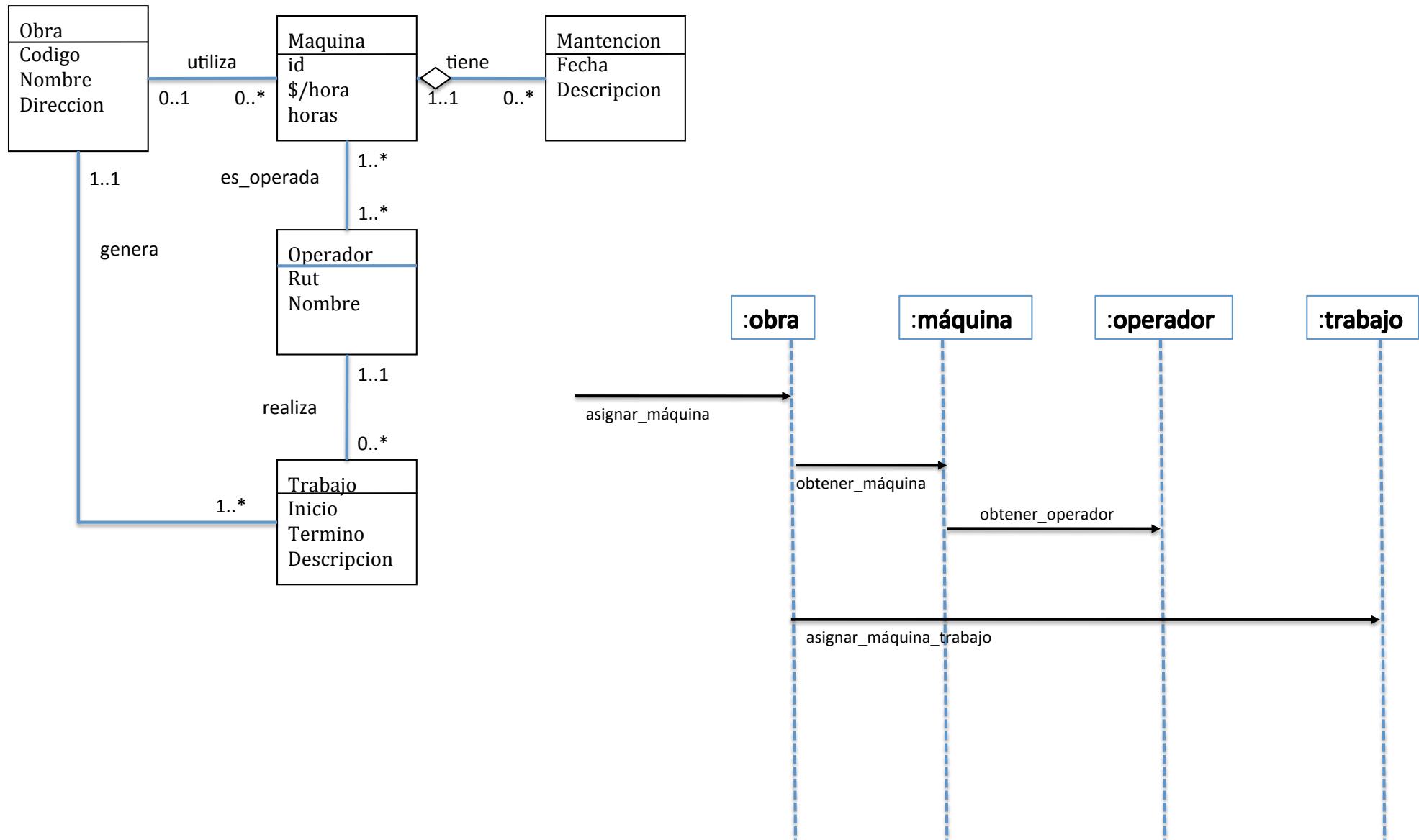
Una sesión de CRC

- no más de 4 a 5 participantes
- informal
- foco en un escenario
- se suele descubrir la necesidad de otras clases, agregar responsabilidades o ajustar algunas

Ejemplo: Empresa Constructora

Una empresa constructora que mantiene normalmente entre 4 y 8 obras en ejecución simultáneamente maneja un pool de **máquinas** comunes que son usadas en las distintas **obras**. Cada una de las máquinas tiene un nombre y un identificador además de otra información asociada: características, costo de operación por hora, horas de uso, horas desde última mantención, etc. Cada una de estas máquinas puede ser operada por un cierto número de **operadores** que la empresa emplea en una modalidad por hora. Cada fin de semana se debe pagar a cada operador de acuerdo al trabajo realizado para lo cual se emite un detalle de cada **trabajo** realizado en este lapso (obra, fecha, hora inicio, hora término, máquina asociada). Para cada máquina se mantiene una historia con el detalle de las **mantenciones** periódicas que se le han realizado.

Modelo de Dominio



Ejemplo: Reserva Aviones

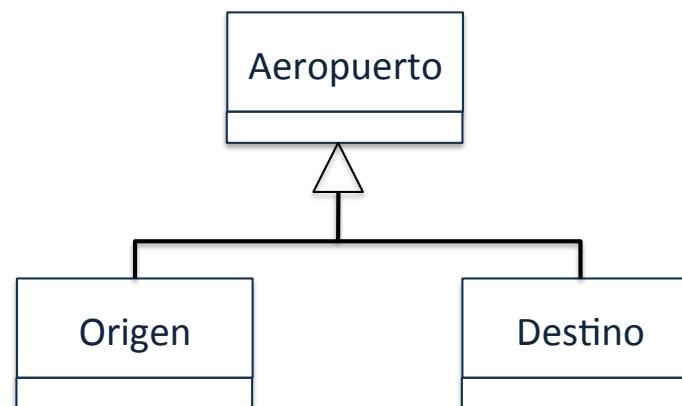
Considera un sistema de reserva de vuelos para una agencia de viajes, con las siguientes características. Las líneas aéreas ofrecen varios vuelos. Un vuelo es abierto para hacer reservas y nuevamente cerrado por orden de la compañía. Un cliente puede reservar uno o más vuelos y para diferentes pasajeros. Una reserva se refiere a un único vuelo y a un único pasajero, y puede ser cancelada o confirmada. Los vuelos tienen aeropuerto, fecha y hora de salida y de llegada, y pueden incluir escalas en aeropuertos; cada escala tiene una hora de llegada y una hora de salida. Cada aeropuerto atiende a una o más ciudades.

Clases

- aerolíneas
- vuelos
- reservas
- aeropuertos
- pasajeros
- clientes
- ciudades

Opciones

- Aeropuertos de origen y destino como subclases



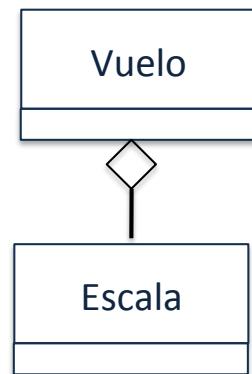
Opciones

- Escala como subclase de aeropuerto



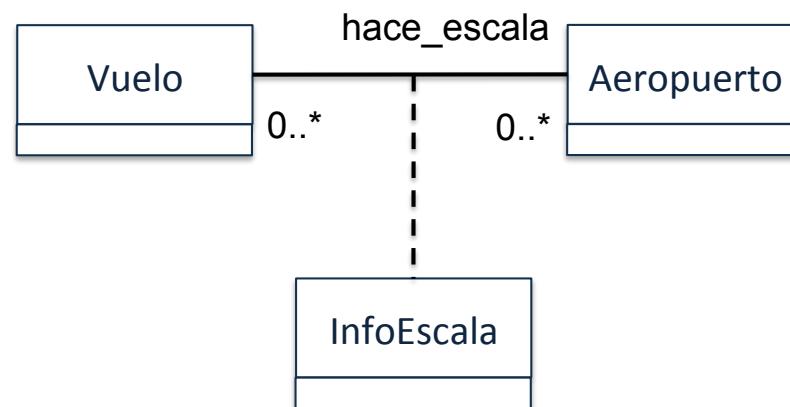
Opciones

- Vuelo contiene número de escalas
- escalas pueden compartirse

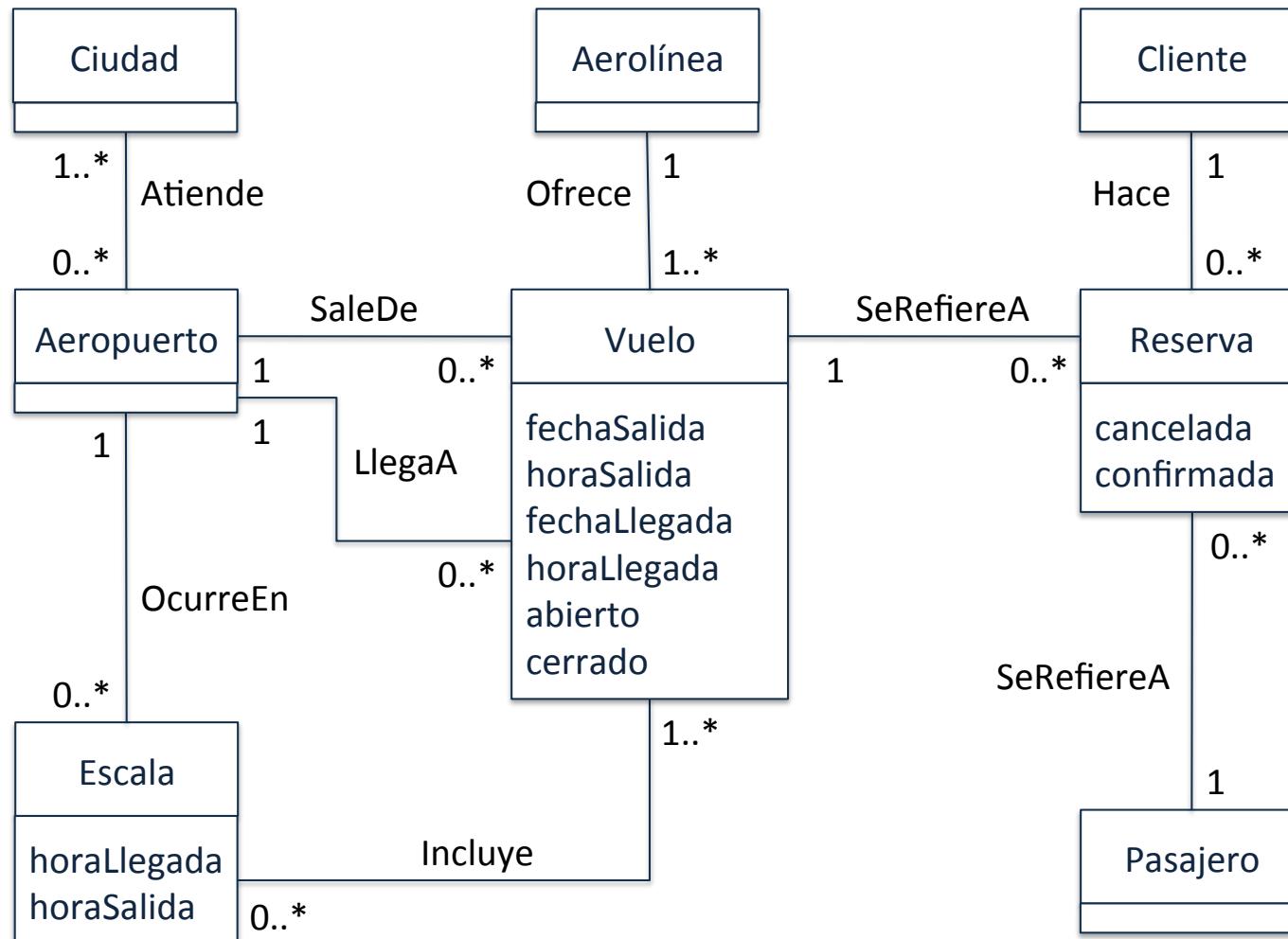


Opciones

- Escala como clase asociativa



Modelo Elegido



Del Análisis al Diseño

- La pregunta fundamental en etapa de análisis es ¿qué?
- La pregunta fundamental en etapa de diseño es ¿cómo?

Componentes, Módulos, Clases

- Un diseño debe poder entenderse
- Complejidad hace necesario descomponer la solución en unidades menores
- Modularidad es clave en lograr ese objetivo
- A menudo se requiere descomposición jerárquica: 5 subsistemas, cada uno con 4 componentes, cada una con una cierta estructura de clases
- Muchas posibles descomposiciones, ¿cuál elegir?

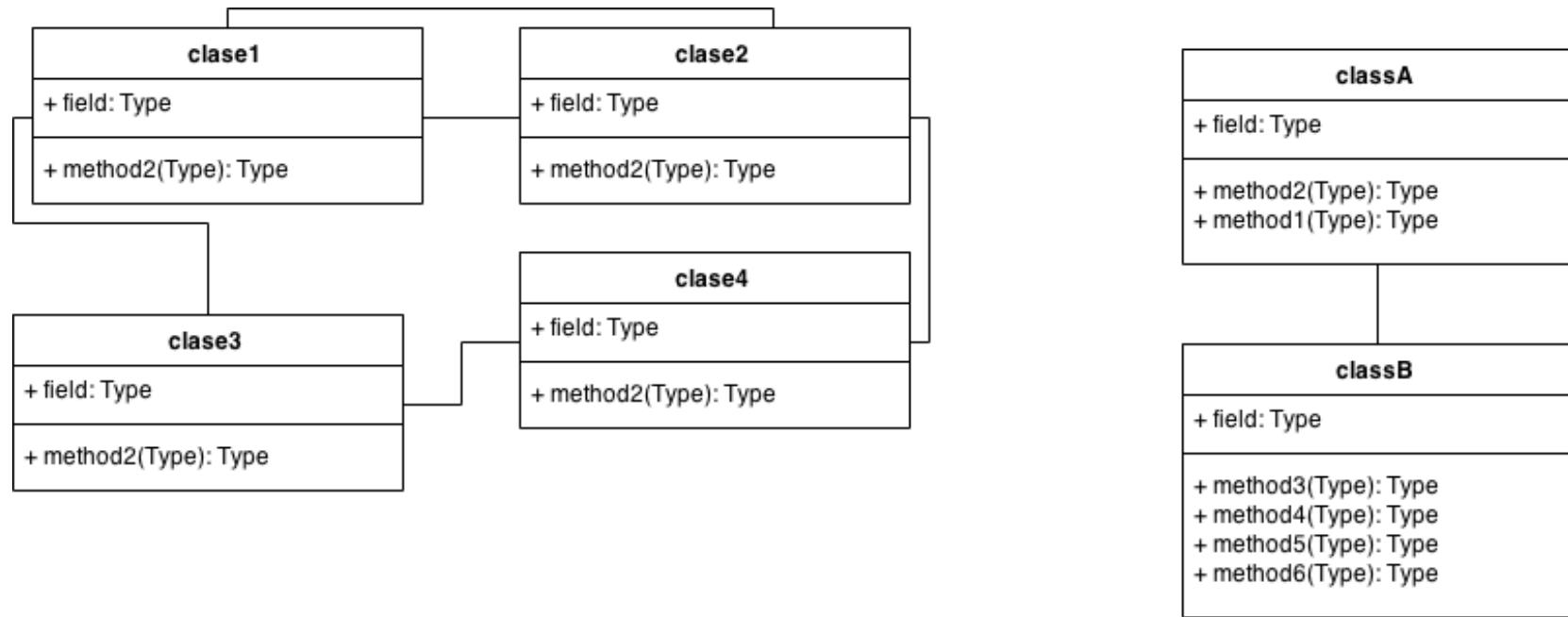
Atributos de un buen diseño

- fácil de entender
- fácil de modificar
- fácil de reutilizar
- fácil de testear
- fácil de integrar
- fácil de programar

Recordar Concepto de Deuda Técnica

- Tienes que agregar una pieza de funcionalidad
 - opción 1 - naive, rápido de implementar
 - opción 2 - diseño mas limpio pero tomará mas tiempo
- Al optar por 1 estamos contrayendo una "deuda técnica"
- Al igual que las deudas financieras se paga mas adelante pero con intereses
- El pago de la deuda va a afectar el desarrollo de nuevas features

¿Cual diseño es mejor ?



Criterio Fundamental:

Buscamos diseños con bajo acoplamiento y alta cohesión

Acoplamiento

- grado de interacción o interdependencia entre unidades
- bajo acoplamiento facilita entender, mantener y testear
- Categorías de mayor (peor) a menor (mejor)
 - acoplamiento por contenido (acceso a datos internos)
 - acoplamiento por variables globales
 - acoplamiento de control (influye en flujo de control del otro)
 - acoplamiento de datos (paso solo de los datos necesarios)

Tipos de Acoplamiento

Level	Type	Description
Good	No Direct Coupling	The methods do not relate to one another; that is, they do not call one another.
	Data	The calling method passes a variable to the called method. If the variable is composite (i.e., an object), the entire object is used by the called method to perform its function.
	Stamp	The calling method passes a composite variable (i.e., an object) to the called method, but the called method only uses a portion of the object to perform its function.
	Control	The calling method passes a control variable whose value will control the execution of the called method.
↓	Common or Global	The methods refer to a “global data area” that is outside the individual objects.
Bad	Content or Pathological	A method of one object refers to the inside (hidden parts) of another object. This violates the principles of encapsulation and information hiding. However, C++ allows this to take place through the use of “friends.”

Source: These types were adapted from Meilir Page-Jones, *The Practical Guide to Structured Systems Design*, 2nd ed. (Englewood Cliffs, NJ: Yordon Press, 1988); and Glenford Myers, *Composite/Structured Design* (New York: Van Nostrand Reinhold, 1978).

Cohesión

- Merriam-Webster define cohesion como "*the act or state of sticking together tightly*", es decir el grado en que los elementos de una unidad están relacionados entre sí
- Una clase enfocada en una sola cosa es más fácil de entender, mantener y testear
- Categorías de cohesión de menor (peor) a mayor (mejor)
 - coincidencial
 - lógica
 - temporal
 - procedural
 - comunicacional
 - secuencial
 - funcional

Tipos de Cohesión

Level	Type	Description
Good	Functional	A method performs a single problem-related task (e.g., calculate current GPA).
	Sequential	The method combines two functions in which the output from the first one is used as the input to the second one (e.g., format and validate current GPA).
	Communicational	The method combines two functions that use the same attributes to execute (e.g., calculate current and cumulative GPA).
	Procedural	The method supports multiple weakly related functions. For example, the method could calculate student GPA, print student record, calculate cumulative GPA, and print cumulative GPA.
	Temporal or Classical	The method supports multiple related functions in time (e.g., initialize all attributes).
	Logical	The method supports multiple related functions, but the choice of the specific function is chosen based on a control variable that is passed into the method. For example, the called method could open a checking account, open a savings account, or calculate a loan, depending on the message that is sent by its calling method.
Bad	Coincidental	The purpose of the method cannot be defined or it performs multiple functions that are unrelated to one another. For example, the method could update customer records, calculate loan payments, print exception reports, and analyze competitor pricing structure.

Source: These types were adapted from Page-Jones, *The Practical Guide to Structured Systems*, and Myers, *Composite/Structured Design*.

Bajo acoplamiento

+

Alta cohesión

Cohesión y acoplamiento en un tren



- Carros acoplados mediante interfaz simple y pequeña
- Separación en carros permite separar contenidos cohesionados

En la literatura también

- Se buscan párrafos de alta cohesión y poco acoplamiento entre ellos

I recently had a discussion with someone about the use of the term "perfect" when labeling the intervals of a fourth and a fifth. As mentioned in my last post, these intervals are perfect consonances. However, unlike octaves and unisons, fourths and fifths can be altered. The label "perfect" is used to distinguish the consonant form of these intervals from the diminished or augmented forms.

This discussion caused me to think about the nature of perfect fifths and fourths in today's music compared to music of the past. Technically, today's fifths and fourths are not exactly perfect. To understand why, we need to have a brief discussion on tuning systems.

A tuning system is a method or formula for obtaining the correct distances between musical intervals on an instrument. The Pythagorean tuning system (created by the mathematician Pythagoras) was used till the beginning of the 16th century. The system was based on a scale that was composed of actual perfect fifths which measure to be 702 cents in distance. Unfortunately, this system results in uneven interval distances for across the pitch spectrum. Unisons and octaves are perfect, but there is one fifth (the wolf fifth) within the sequence that is a different size. This causes the other intervals within the sequence to have two different sizes throughout the series.

People experimented with other tuning systems throughout the years in order to have more consistent intervals. Eventually the system of equal temperament was accepted as the dominant tuning system. In this system octaves are subdivided into halve steps of equal distance. This results in fifths that are slightly flat when compared to a pure perfect fifth. However, the mathematical inconsistencies that resulted from Pythagorean tuning (and other systems) are eliminated. The distances between intervals are equal across every key and register of the pitch spectrum.

This equal temperament system is the one that we are used to hearing now when we listen to music. Most people do not even realize that the fifth they hear on an equal tempered piano is not pure. Piano tuners, however, are very aware of this. They are trained to hear the proper beating sound of an equal tempered fifth.

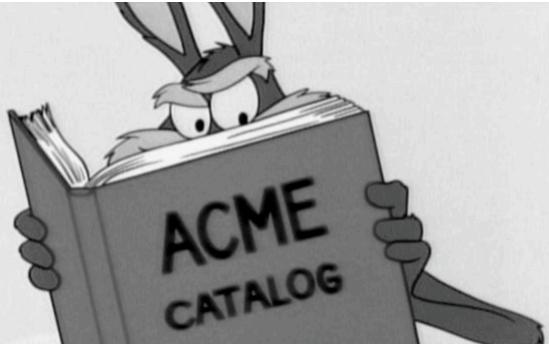
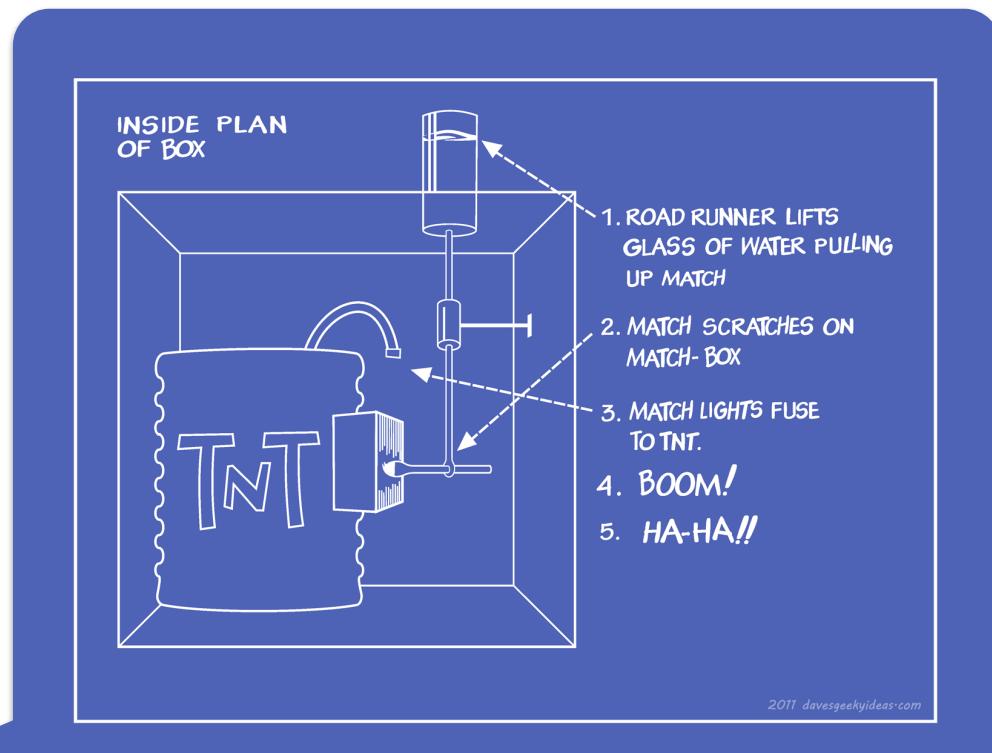
A classic example of the impact of tuning systems is Bach's Well-Tempered Clavier. This is a collection of preludes and fugues written in all 24 major and minor keys for solo keyboard. This collection was composed before equal temperament, and was originally played on instruments that were tuned with other systems. The mathematical inconsistencies of these systems caused the different keys to possess different sonic qualities and characters. This sonic variety is lost when the pieces are performed on keyboard instruments tuned with an equal temperament. So, we have gained symmetry in our intervals by sacrificing the individual character of the different key signatures and the pure fifth.

Por qué alta cohesión y bajo acoplamiento

- interfaces simples
- comunicación simple
- cambios afectan a sectores limitados del código
- aumenta la reusabilidad
- aumenta la extensibilidad

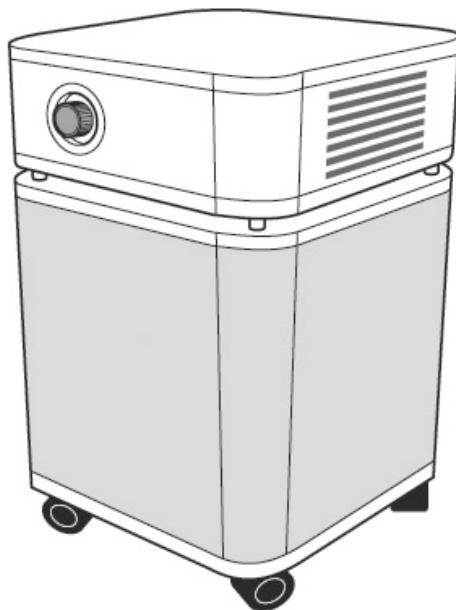
Lenguaje de Modelación Unificado (UML)

- Actividad de diseño se favorece al trabajar con un lenguaje gráfico de alto nivel

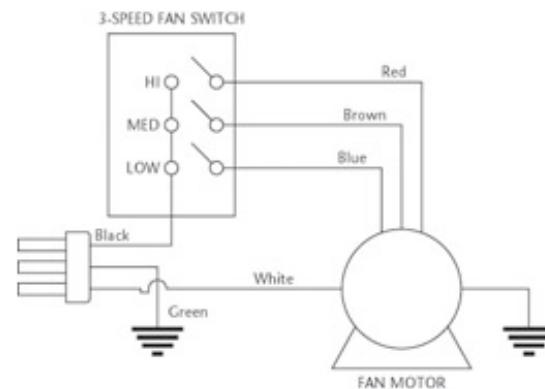


UML es un lenguaje visual para describir artefactos de software

artefacto



lenguaje visual



```

public class Payment {
    private float amount;
    public Payment (float cashTendered){this.amount = cashTendered;}
    public float getAmount() { return amount; }
}

public class ProductCatalog{
    private HashTable productSpecifications = new Hashtable();
    public ProductCatalog(){
        ProductSpecification ps = new ProductSpecification(100, 1, "product 1");
        productSpecifications.put (new Integer(100), ps);
        ps = new ProductSpecification(200, 1, "product 2");
        productSpecifications.put (new Integer(200), ps);
    }

    public ProductSpecification getSpecification (int upc){
        return (ProductSpecification)productSpecifications.get(new Integer(upc));
    }
}

class POST {
    private ProductCatalog productCatalog;
    private Sale sale;
    public POST(ProductCatalog catalog){productCatalog = catalog;}
    public void endSale() {sale.becomeComplete();}
    public void enterItem(int upc, int quantity){
        if (isNewSale() ) sale = new Sale();
        ProductSpecification spec = productCatalog.specification(upc);
        sale.makeLineItem(spec, quantity);
    }
    public void makePayment(float cashTendered){
        sale.makePayment(cashTendered);
    }
    private boolean isNewSale(){return (sale == null) ||(sale.isComplete());}
}

public class ProductSpecification{
    private int upc = 0;
    private float price = 0;
    private String description = "";
    public ProductSpecification(int upc, float price, String description){
        this.upc = upc;
        this.price = price;
        this.description = description;
    }
    public int getUPC {return upc; }
    public float getPrice() {return price; }
    public String getDescription() {return description; }
}

```

```

class Sale {
    private Vector lineItems = new Vector();
    private Date date = new Date();
    private boolean isComplete = false;
    private Payment payment;
    public float getBalance () {return payment.getAmount() - total(); }
    public void becomeComplete() {isComplete = true; }
    public boolean isComplete() {return isComplete; }
    public void makeLineItem(ProductSpecification spec, int quantity) {
        lineItems.addElement(new SaleLineItem(spec, quantity));
    }
    public float total(){
        float total = 0;
        Enumeration e = lineItems.elements();
        while(e.hasMoreElements() ) {
            total += ( (SaleLineItem) e.nextElement() ).subtotal();
        }
        return total;
    }
    public void makePayment (float cashTendered) {
        payment = new Payment (cashTendered);
    }
}

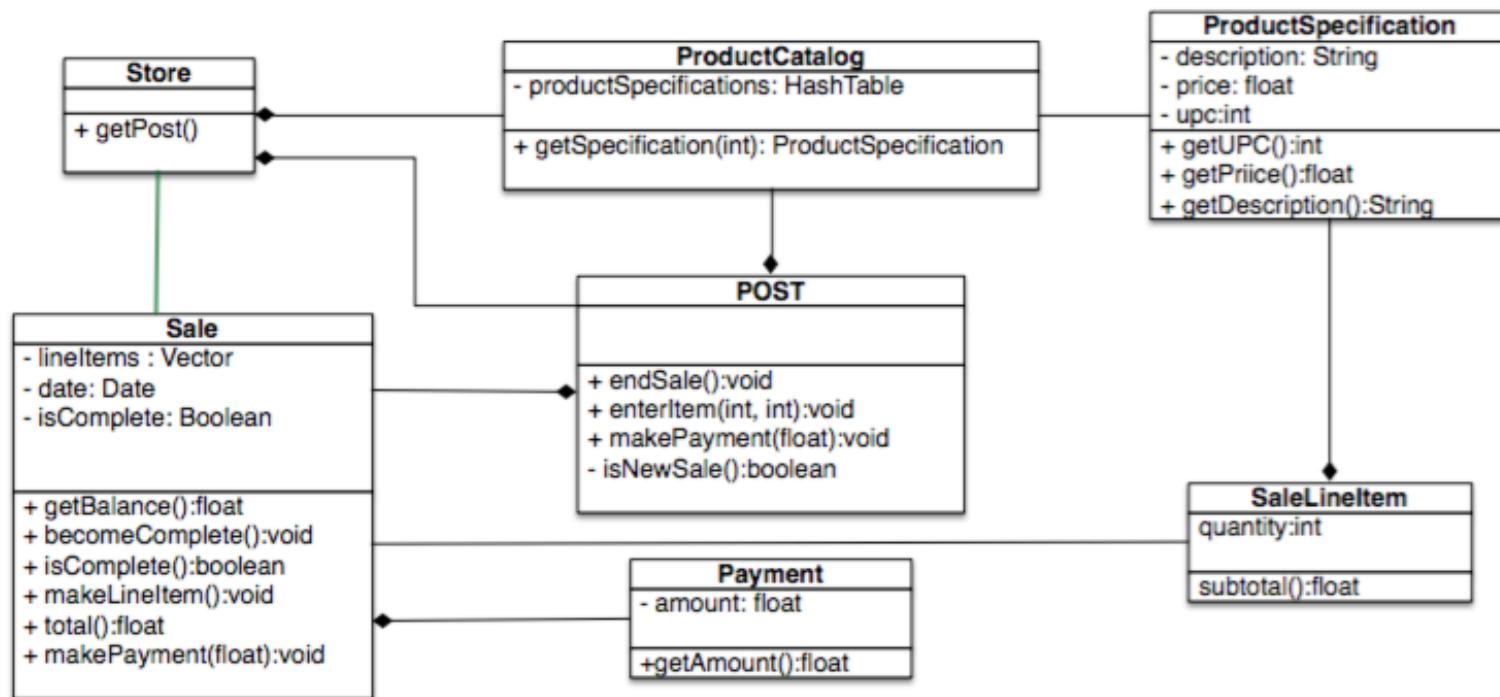
class SaleLineItem{
    private int quantity;
    private ProductSpecification productSpec;
    public SaleLineItem (ProductSpecification spec, int quantity) {
        this.productSpec = spec;
        this.quantity = quantity;
    }
    public float subtotal() { return quantity* productSpec.getPrice(); }
}

class Store {
    private ProductCatalog productCatalog = new ProductCatalog();
    private POST post = new POST (productCatalog);
    public POST getPOST() {return post; }
}

```

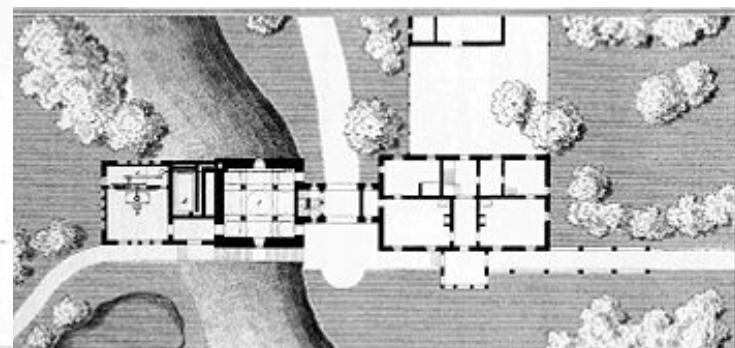
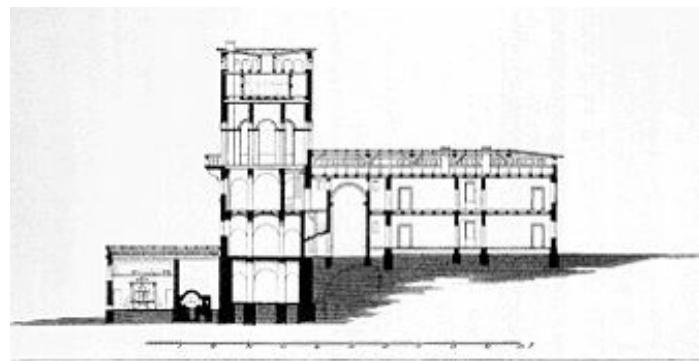
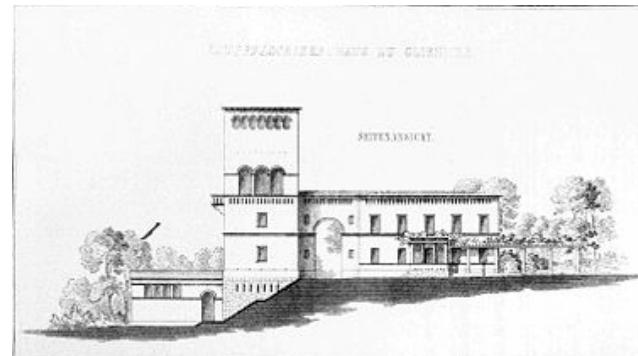
Artefacto

Representación Visual del Artefacto



Hay mas de 10 tipos de diagramas

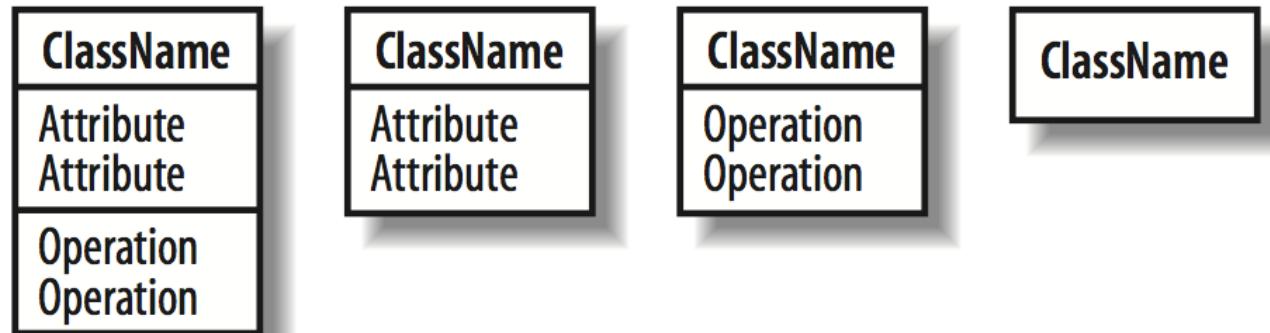
- Análogamente que en arquitectura se requieren distintas vistas



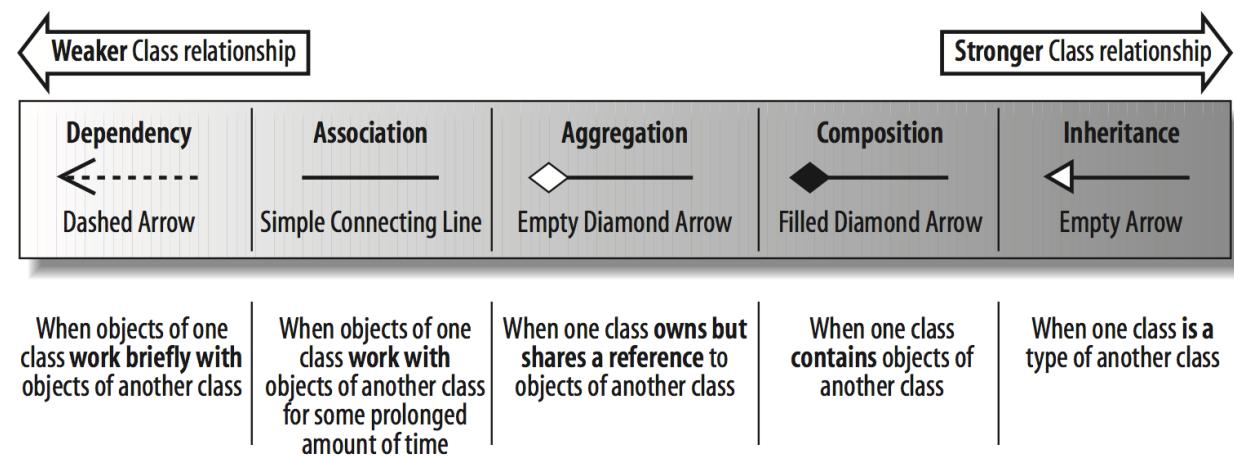
Estructura : El diagrama de clases

- Rectángulos para las clases

- nombre
- atributos
- métodos



- Líneas representan asociaciones entre clases
(adornos para indicar asociaciones especiales)

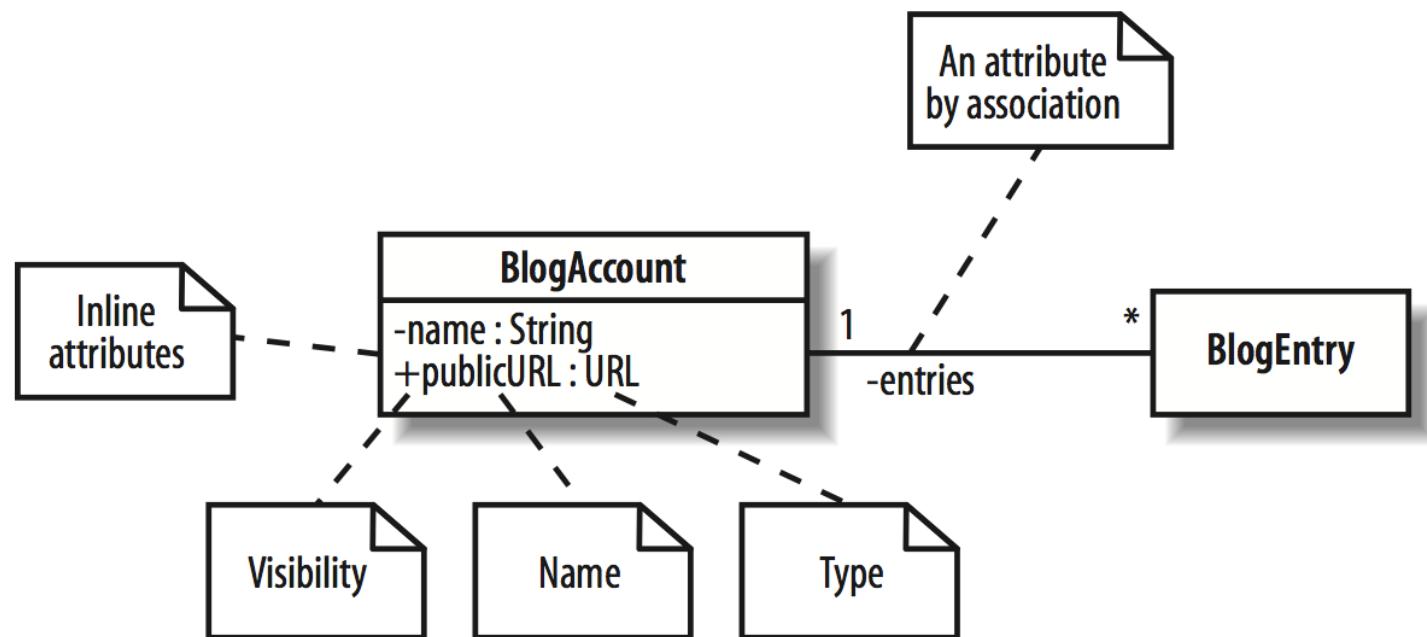


Visibilidad

- Qué partes de una clase son visibles para instancias de otras clases (encapsulación)
- Hay cuatro tipos de visibilidad
 - público (+) accesible a todos
 - protegido (#) accesible solamente a descendientes
 - privado (-) solo para uso interno
 - package (~) accesible a clases del mismo package

Atributos

- Nombre y tipo
- Si es estático se subraya
- Generalmente en el rectángulo
- Puede también ponerse como asociado



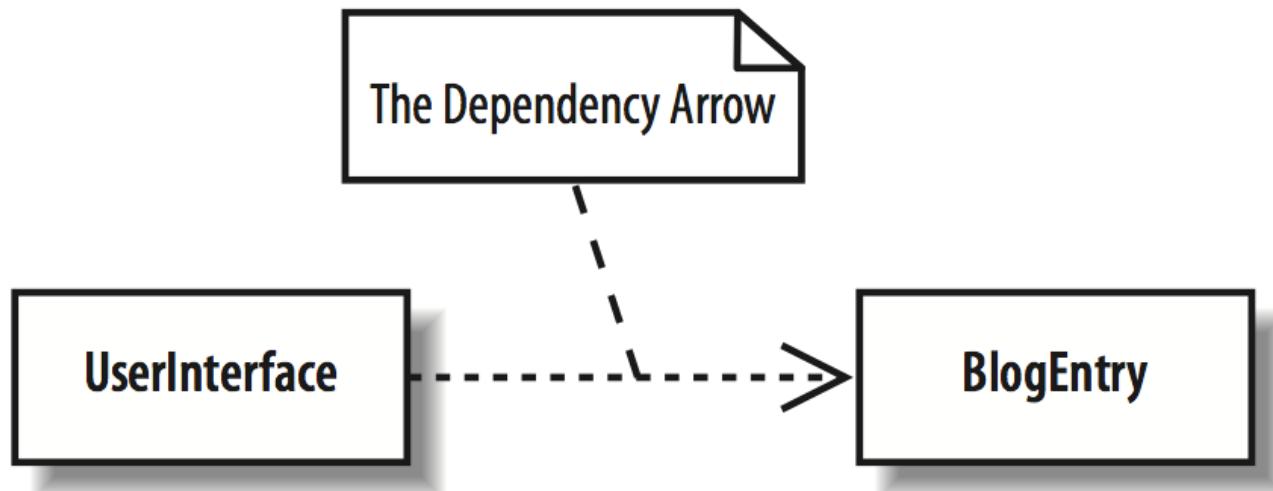
Operaciones

- Nombre
- Parámetros
- Retorno

BlogAccount
- name : String + publicURL : URL - authors : Author [1..5]
+ addEntry(newEntry : BlogEntry, author : Author) : boolean + BlogAccount(name : String, publicURL : URL)

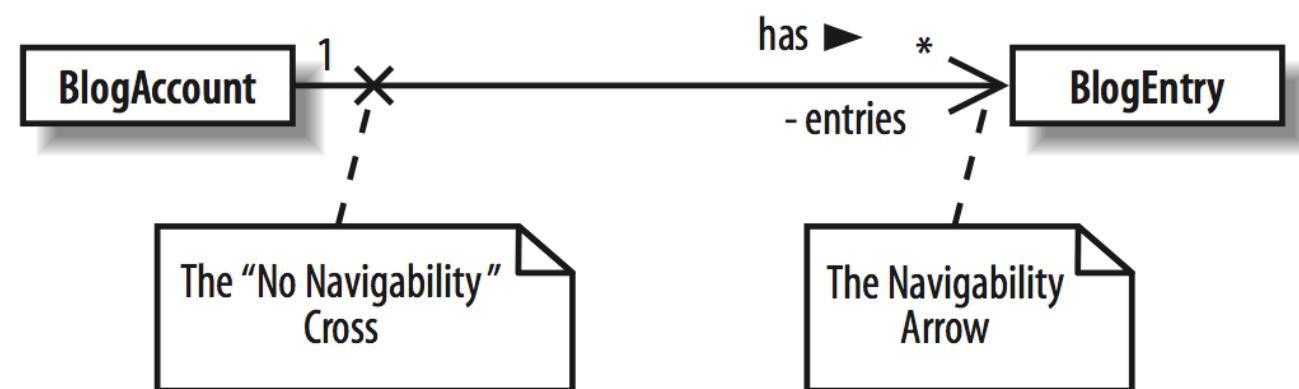
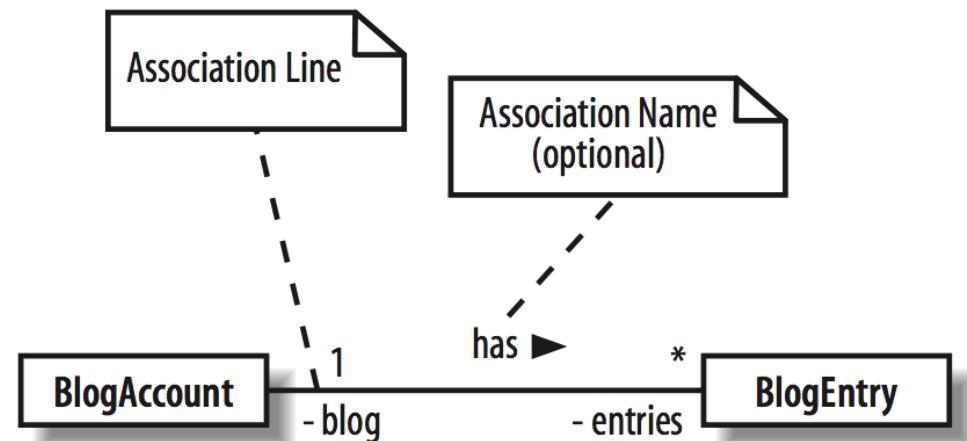
Asociaciones : Dependencia

- Una clase necesita usar objetos de otra clase



Asociaciones: Asociación

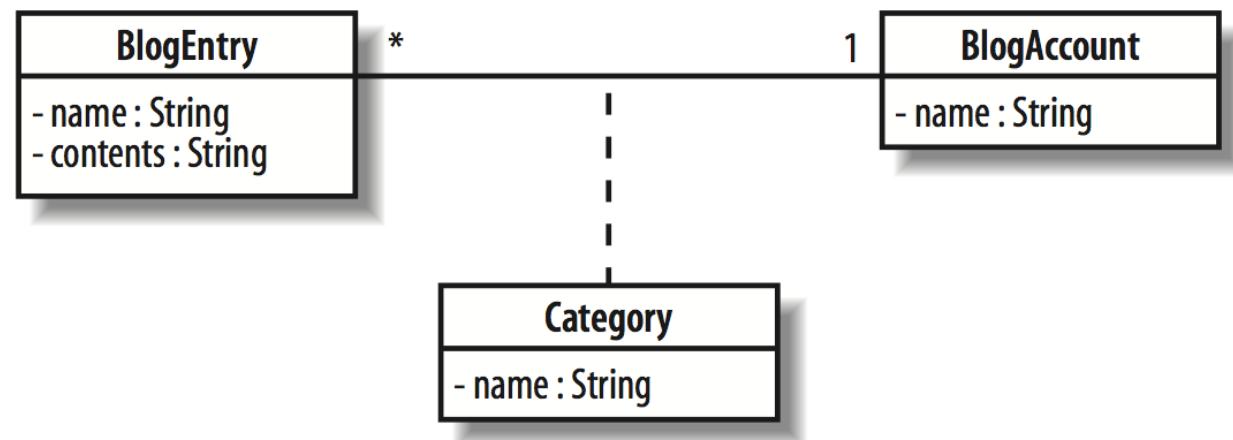
- Una clase contiene una referencia a objeto de otra clase en un atributo
- Puede agregarse flecha para indicar dirección de navegabilidad



Clase de Asociación

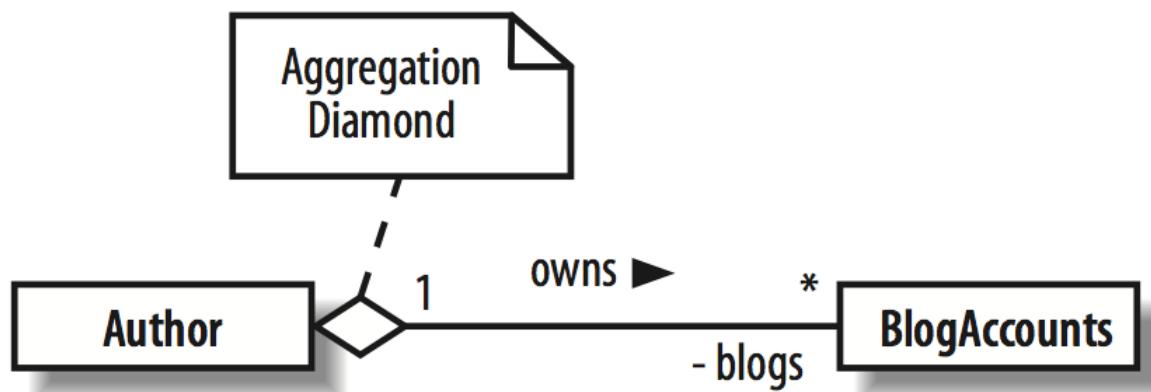
- A veces la asociación misma es compleja y puede tener atributos

```
public class BlogAccount {  
    private String name;  
    private Category[] categories;  
    private BlogEntry[] entries;  
}  
  
public class Category {  
    private String name;  
}  
  
public class BlogEntry {  
    private String name;  
    private Category[] categories  
}
```



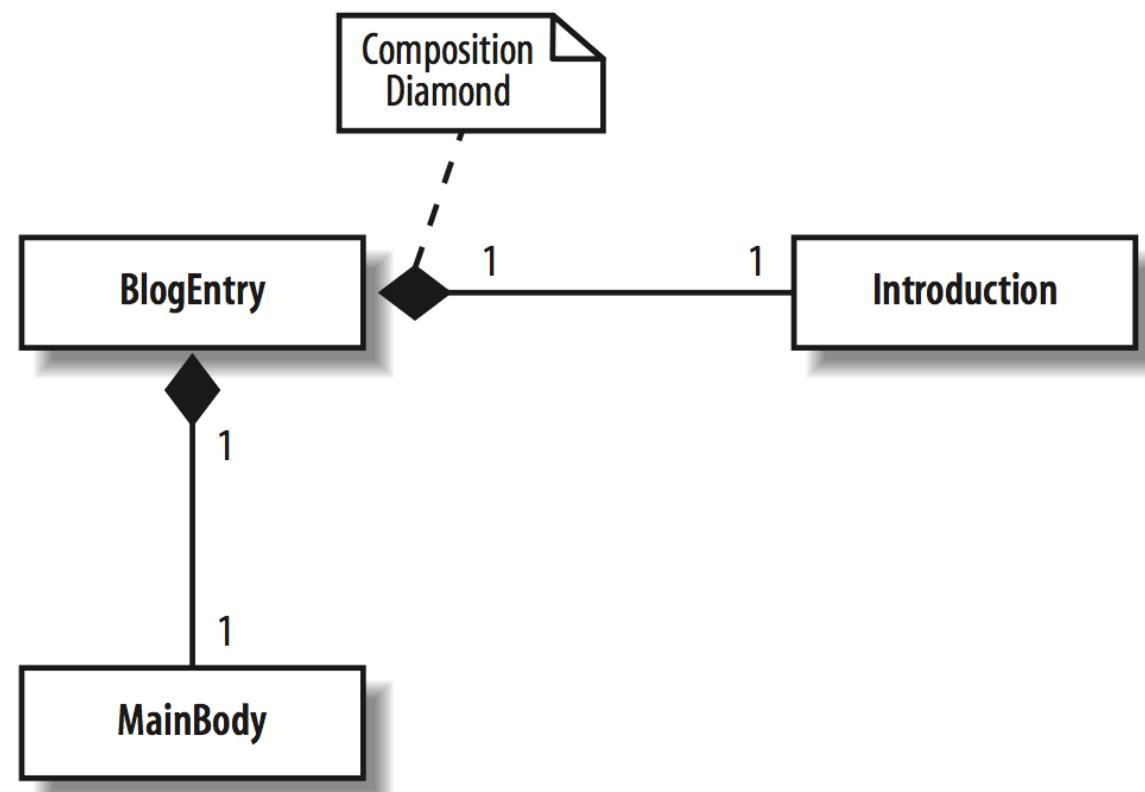
Asociaciones: Agregación

- Un poco mas fuerte que simple asociación
- Una clase es dueña de objetos de otras clase (aunque puede compartirlos)



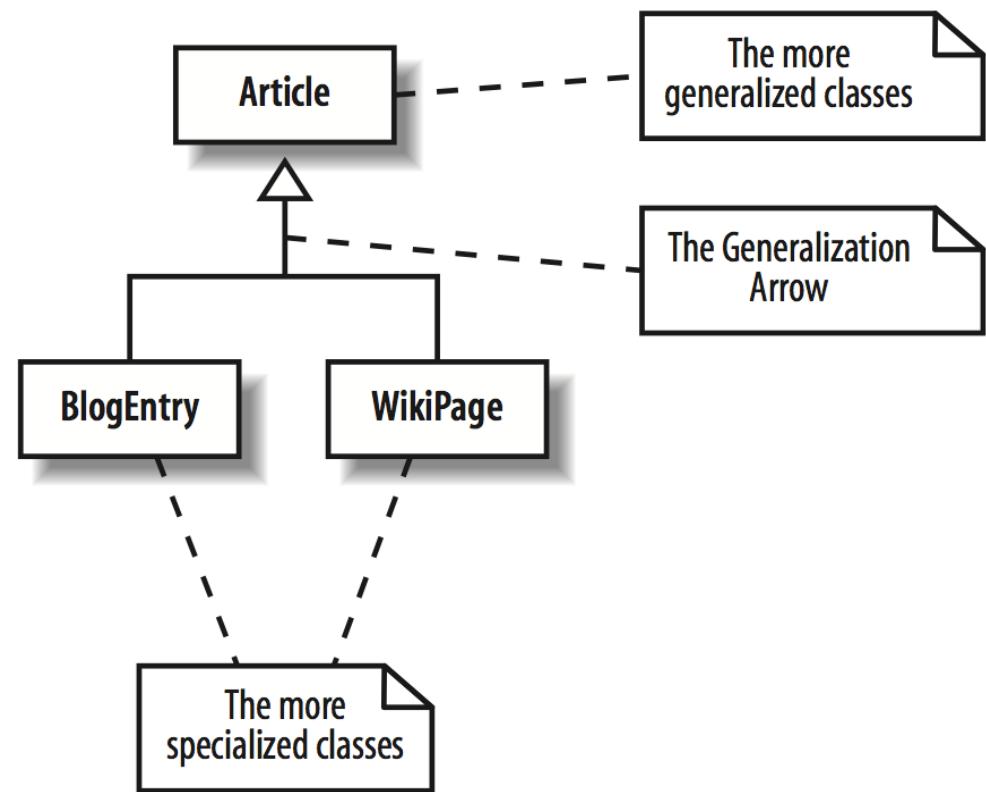
Asociaciones: Composición

- Asociación aún más fuerte
- Una clase es dueña en forma exclusiva o contiene objetos de otra clase



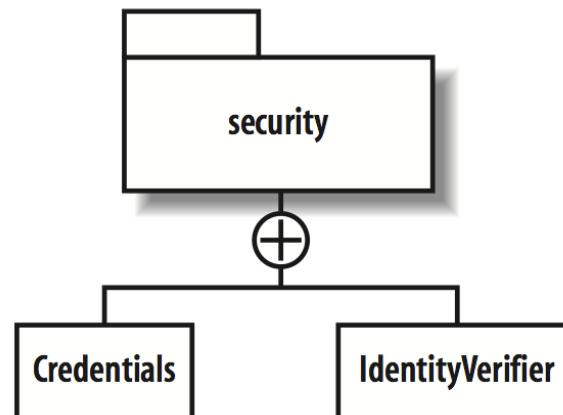
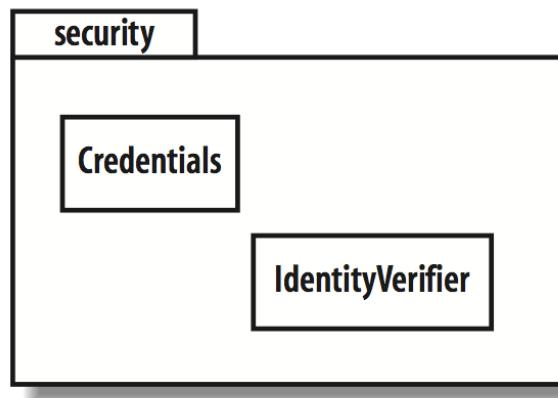
Asociaciones: Generalización

- tambien se le suele llamar (incorrectamente) “herencia”
- agregación y composición obedece a relación “has a”
- generalización es mas bien “is a” o “is a type of a”



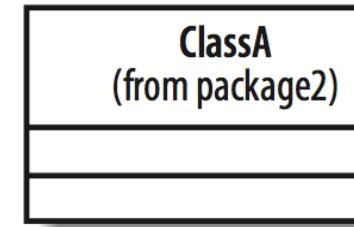
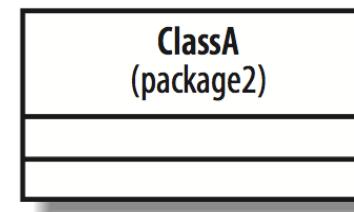
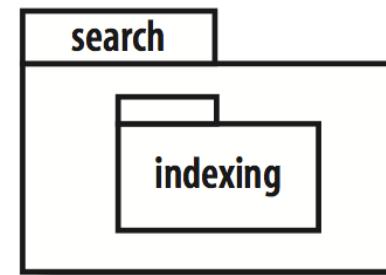
Packages

- Una forma de agrupación de elementos
- Se usa principalmente para clases pero puede usarse para agrupar otros elementos también
- Ejemplo
package security;
public class Credentials {
...
}
- Clases pueden dibujarse dentro o fuera del package ...

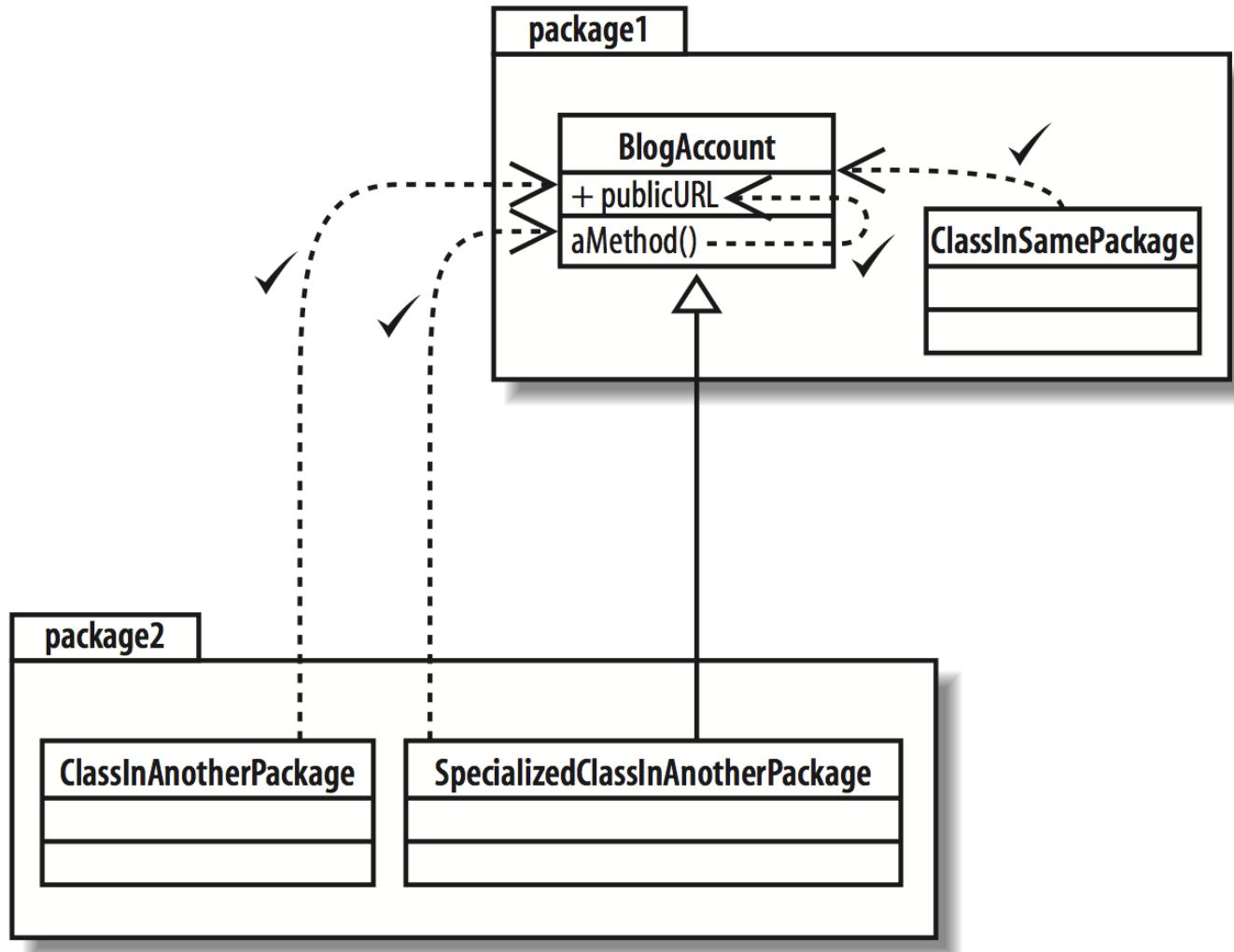


más sobre packages

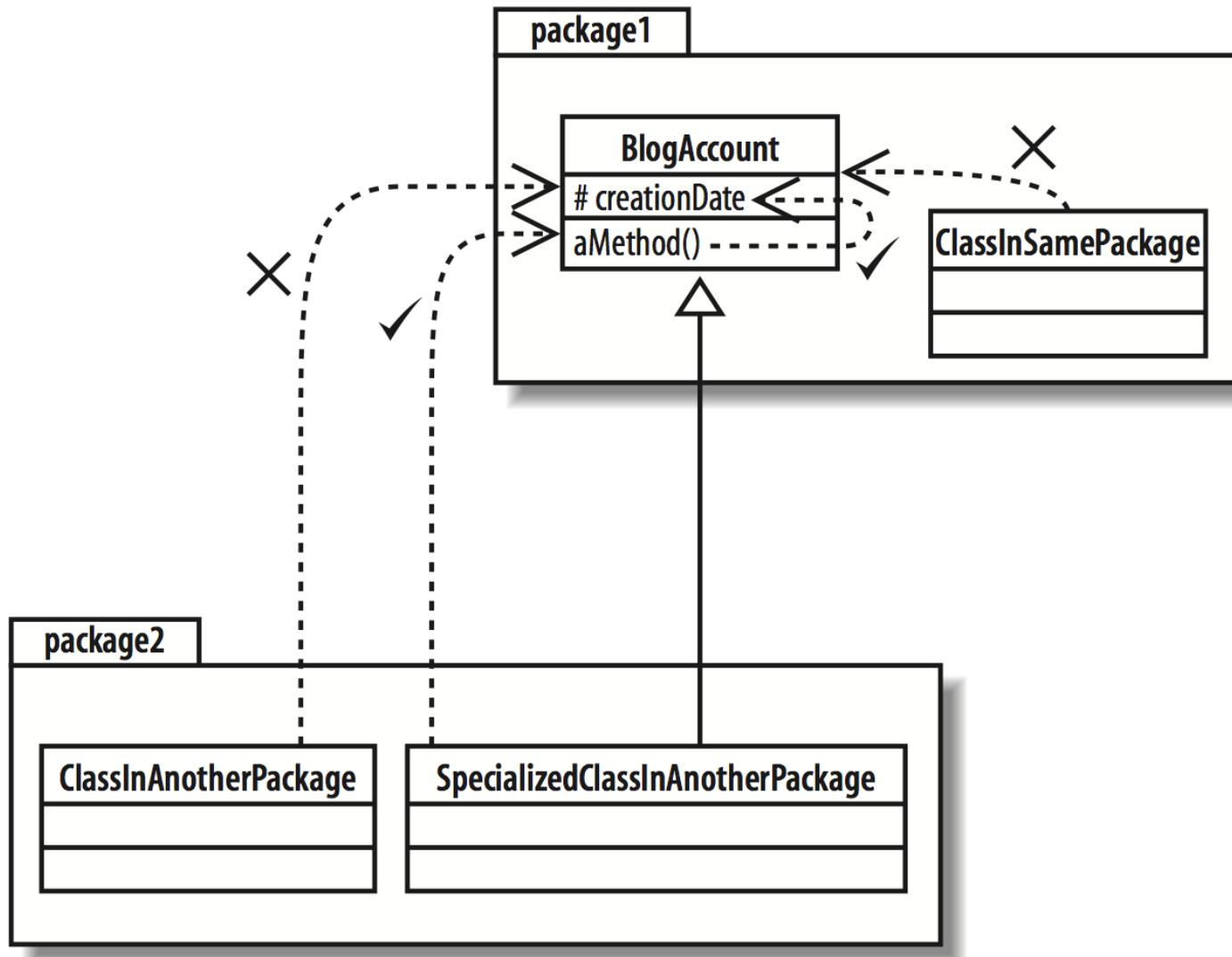
- pueden anidarse
- puede haber indicación del package en el nombre
- package proporciona espacio de nombres



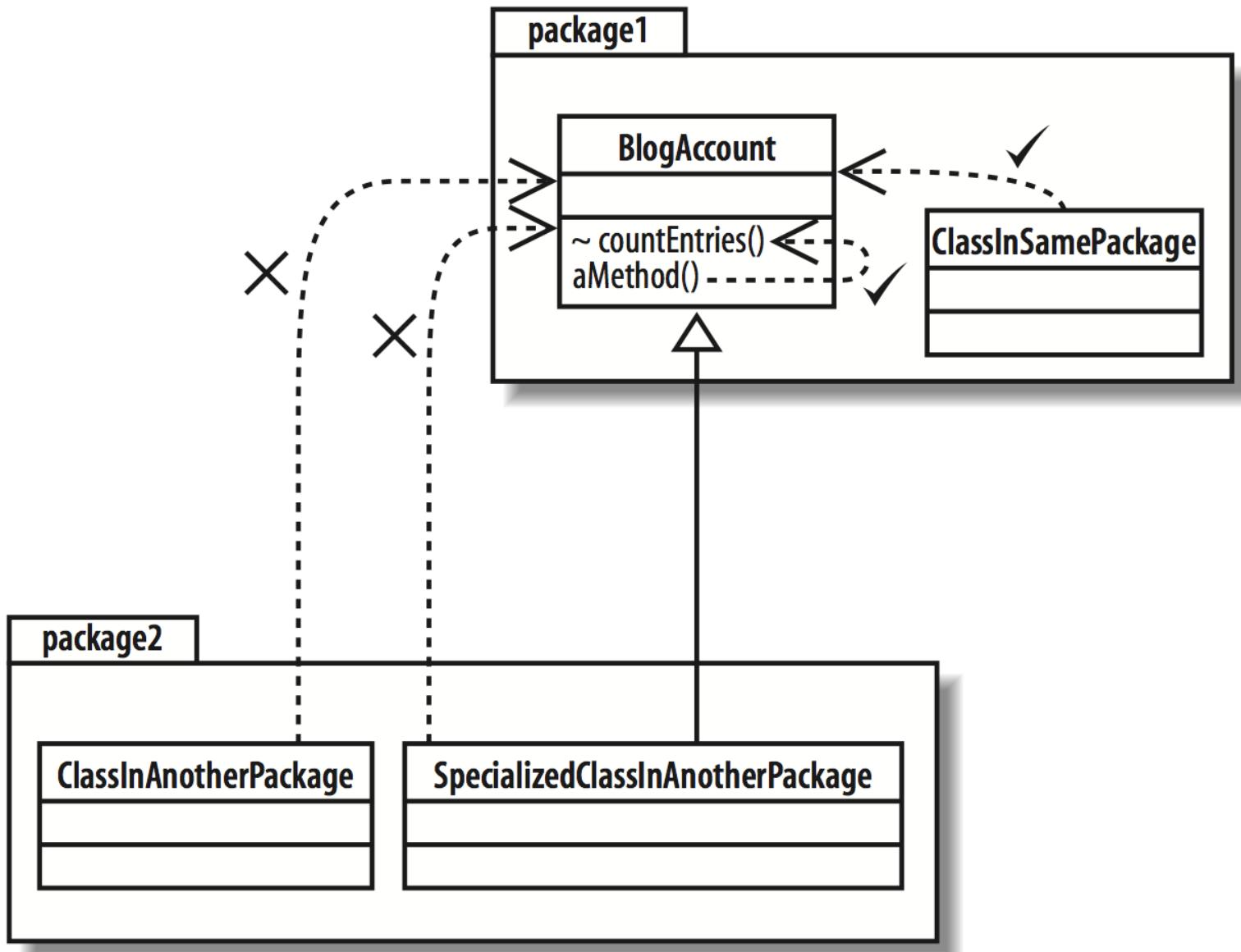
Visibilidad Pública



Visibilidad Protegida



Visibilidad de Package



Visibilidad Privada

