# Sesión 21
# **Diagrama de Clases (II)**

Unidad 4

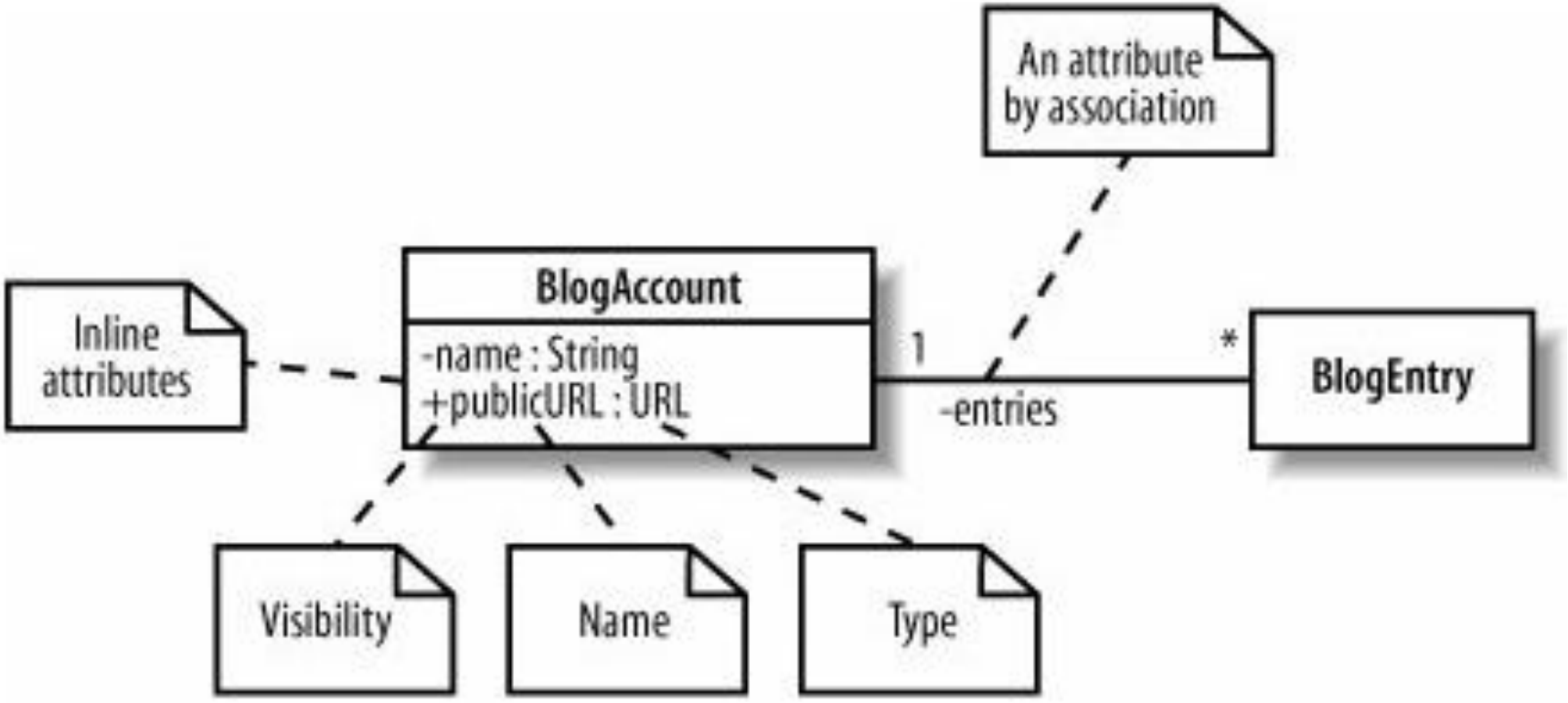
**Modelado del comportamiento estático del sistema**

Mg. Gustavo G. Delgado Ugarte

# Atributos (Propiedades)

- Los atributos de una clase son las piezas de información que representan el estado de un objeto
- Estos atributos pueden ser representados en un diagrama de clases ya sea
  - Colocándolos dentro de su sección del cuadro de la clase: atributos en línea
  - Por asociación con otra clase

# Atributos (Propiedades)



# Atributos (Propiedades)

- El atributo suelen tener una firma que contiene
  - Una propiedad de visibilidad
  - Un nombre
  - Un tipo
- El nombre del atributo es la única parte de la firma que es absolutamente necesario que esté presente para que la clase sea válida

# Atributos (Propiedades)

- El nombre de un atributo puede ser cualquiera, pero no hay dos atributos en la misma clase que tengan el mismo nombre
- El tipo de atributo puede variar dependiendo de cómo la clase se implementará en el sistema, pero suele ser una clase, un cadena de caracteres o un tipo primitivo (como un entero)

# Multiplicidad (Atributos)

- A veces un atributo representa más de un objeto
- La multiplicidad permite especificar que un atributo representa en realidad una colección de objetos, y puede ser aplicado tanto a atributos en línea, como a atributos por asociación

# Multiplicidad (Atributos)

# Multiplicidad (Atributos)

- * significa varios
- [n..m] indica que el atributo es al menos n y como máximo m, siendo n<m
  - M puede ser también un *
    - Ej. [1..5] , [1..*]
- {xxxx} representa una cadena de propiedad
  - {unique}
  - {ordered}

# Multiplicidad (Atributos)

- La cadena de propiedad indica un modificador que se aplica a los atributos
  - {readonly}  La propiedad se puede leer pero no cambiar
  - {union} La propiedad es una unión de subconjuntos
  - {subsets <property>} La propiedad es un subconjunto de <property>
  - {redefines <property>} La propiedad es una nueva definición de <property> (sobrescritos por herencia)

# Multiplicidad (Atributos)

- La cadena de propiedad indica un modificador que se aplica a los atributos
  - {ordered}, {unordered} Un conjunto ordenado o desordenado
  - {unique}, {nonunique} o {bag} Un conjunto puede o no puede contener varios elementos idénticos
  - {sequence} Una lista ordenada (array, los elementos idénticos están permitidos)
  - {composite} La propiedad es una parte dependiente de la existencia de otra

# Multiplicidad (Atributos)

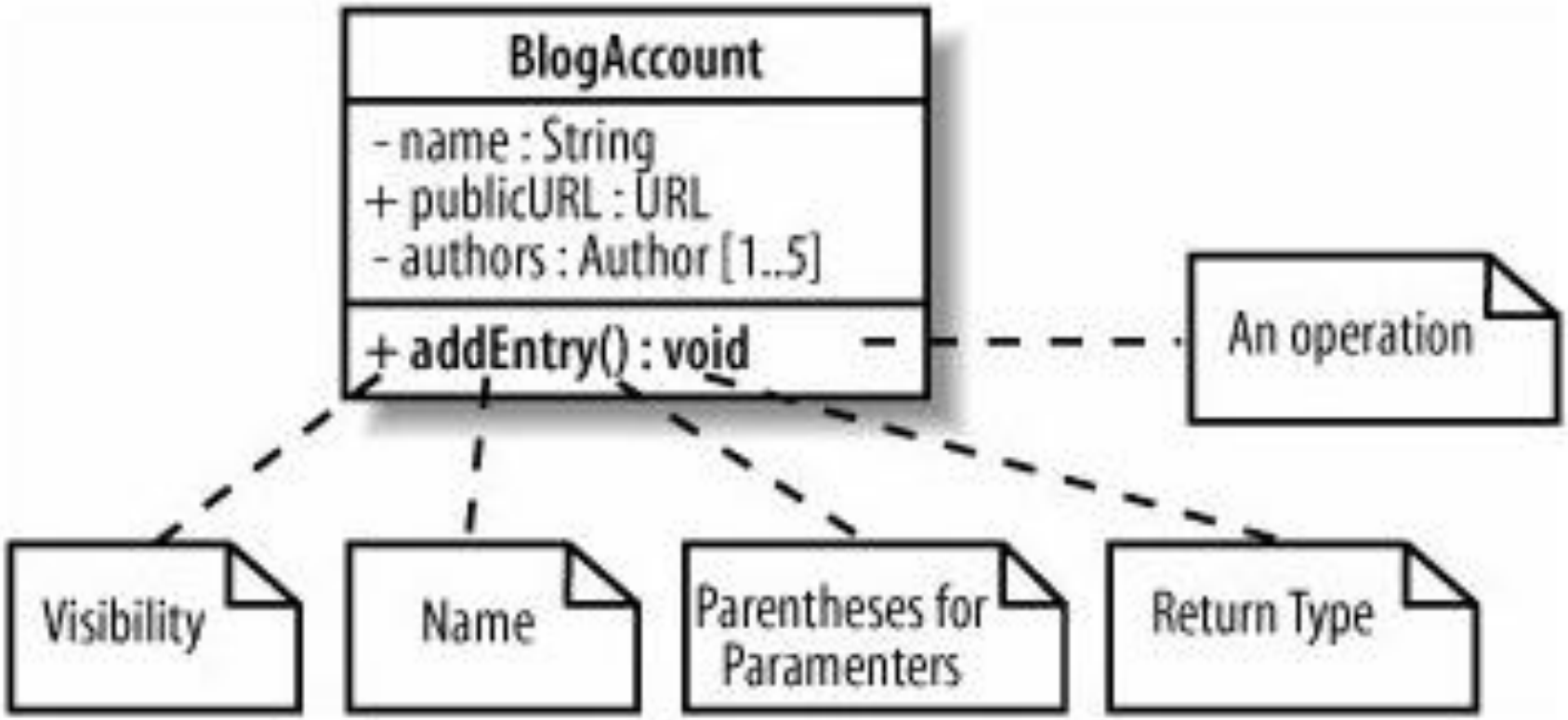
# Operaciones

- Las operaciones de una clase describen lo que una clase puede hacer, pero no necesariamente cómo se va a hacer
- Una operación es más como una promesa o un contrato mínimo que declara que la clase va a contener un comportamiento que hace lo que la operación dice que hará
- La colección de todas las operaciones que contiene una clase debe abarcar a todos los comportamientos que contiene la clase, incluyendo todo el trabajo que mantiene los atributos de la clase y, posiblemente, algún comportamiento adicional que está estrechamente relacionado con la clase

# Operaciones

- Las operaciones en UML se especifican en un diagrama de clases con una firma que está, como mínimo, compuesto por
  - Una propiedad de visibilidad
  - Un nombre
  - Un par de paréntesis en el que los parámetros que se necesitan para la operación haga su trabajo puedan ser suministrados
  - Un tipo de retorno

# Operaciones



# Parámetros

- Los parámetros se utilizan para especificar la información proporcionada a una operación para que pueda completar su trabajo
- Como mínimo, un parámetro tiene que tener su tipo especificado
- Más de un parámetro se puede pasar a una operación, separando los parámetros con una coma

# Parámetros


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

# Tipos de Retorno

- Un tipo de retorno se especifica después de dos puntos al final de la firma de una operación y se especifica el tipo de objeto que será devuelto por la operación

# Tipos de Retorno

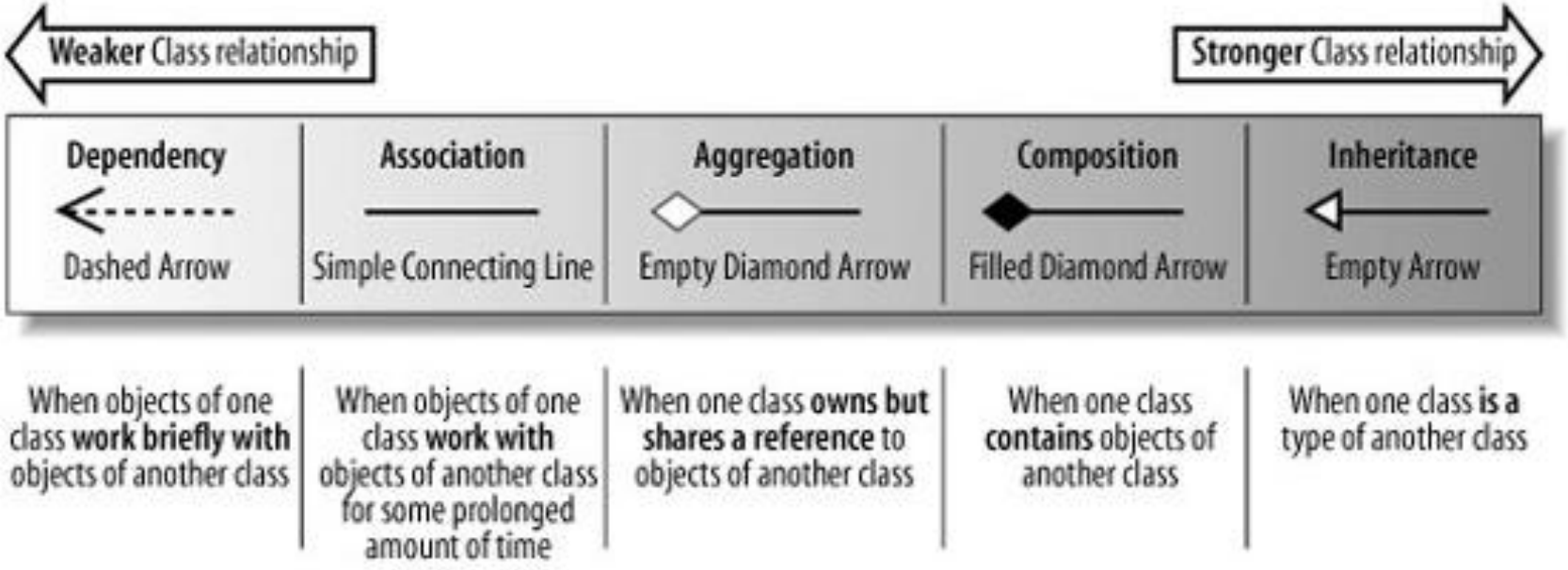
- Hay una excepción en el que no es necesario especificar un tipo de retorno: cuando se declara un constructor de un clase
- Un constructor crea y devuelve una nueva instancia de la clase que se especifica, por lo tanto, no es necesario declarar explícitamente el tipo de retorno

# Tipos de Retorno

# Relaciones

- Las clases no viven en el vacío, ellas trabajan juntas usando diferentes tipos de relaciones.
- Las relaciones entre las clases tienen diversas fuerzas
- La fuerza de una relación de clase se basa en la dependencia que las clases involucradas en la relación tienen, una de otra
  - Dos clases que son muy dependientes una de otra se dice que están bien acoplados
    - Los cambios a una clase, es más probable que afecte a la otra clase
  - El alto acoplamiento es usualmente, pero no siempre, una cosa mala, por lo tanto, mientras más fuerte es la relación, más cuidado se ha de tener

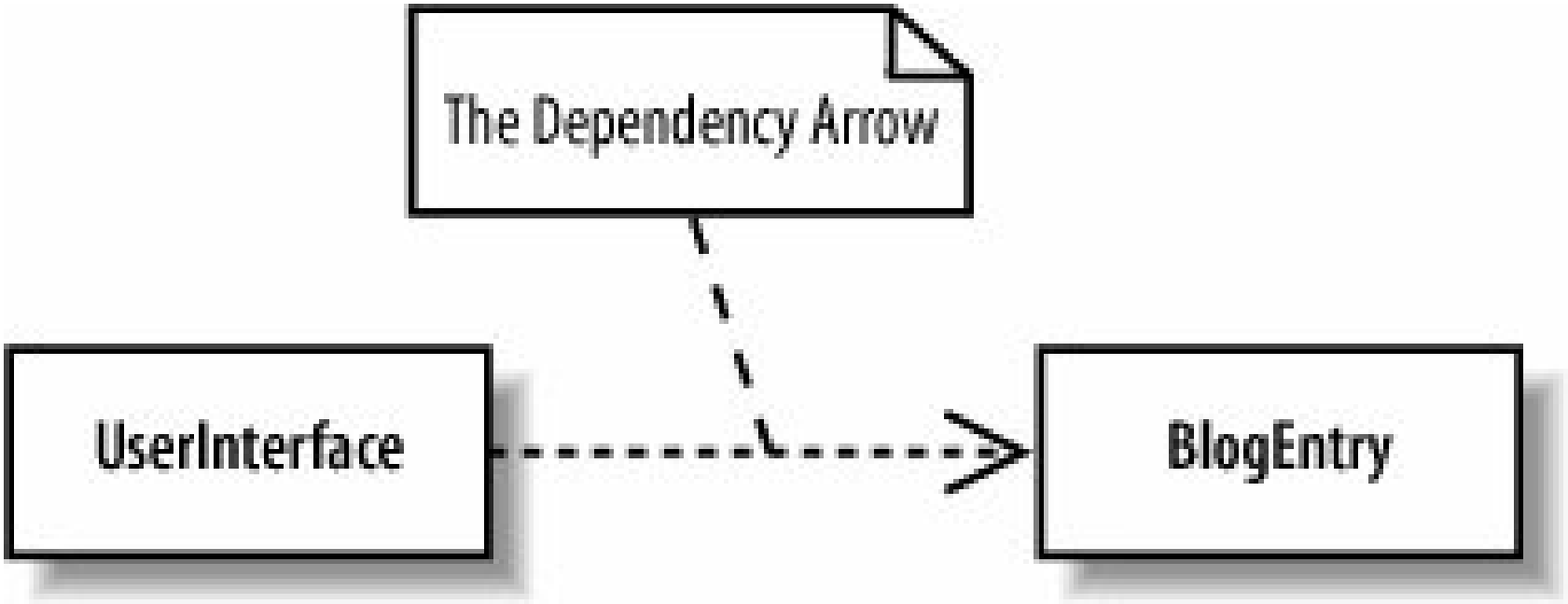
# Relaciones


Weaker Class relationship
Stronger Class relationship
Dependency
Dashed Arrow
Association
Simple Connecting Line
Aggregation
Empty Diamond Arrow
Composition
Filled Diamond Arrow
Inheritance
Empty Arrow
When objects of one class work briefly with objects of another class
When objects of one class work with objects of another class for some prolonged amount of time
When one class owns but shares a reference to objects of another class
When one class contains objects of another class
When one class is a type of another class

# Dependencia

- Una dependencia entre dos clases declara que una clase necesita conocer a otra clase para utilizar los objetos de esa clase
- Una dependencia sólo implica que los objetos de una clase pueden trabajar juntos, por lo que es considerado como la más débil de las relaciones directas que puede existir entre dos clases

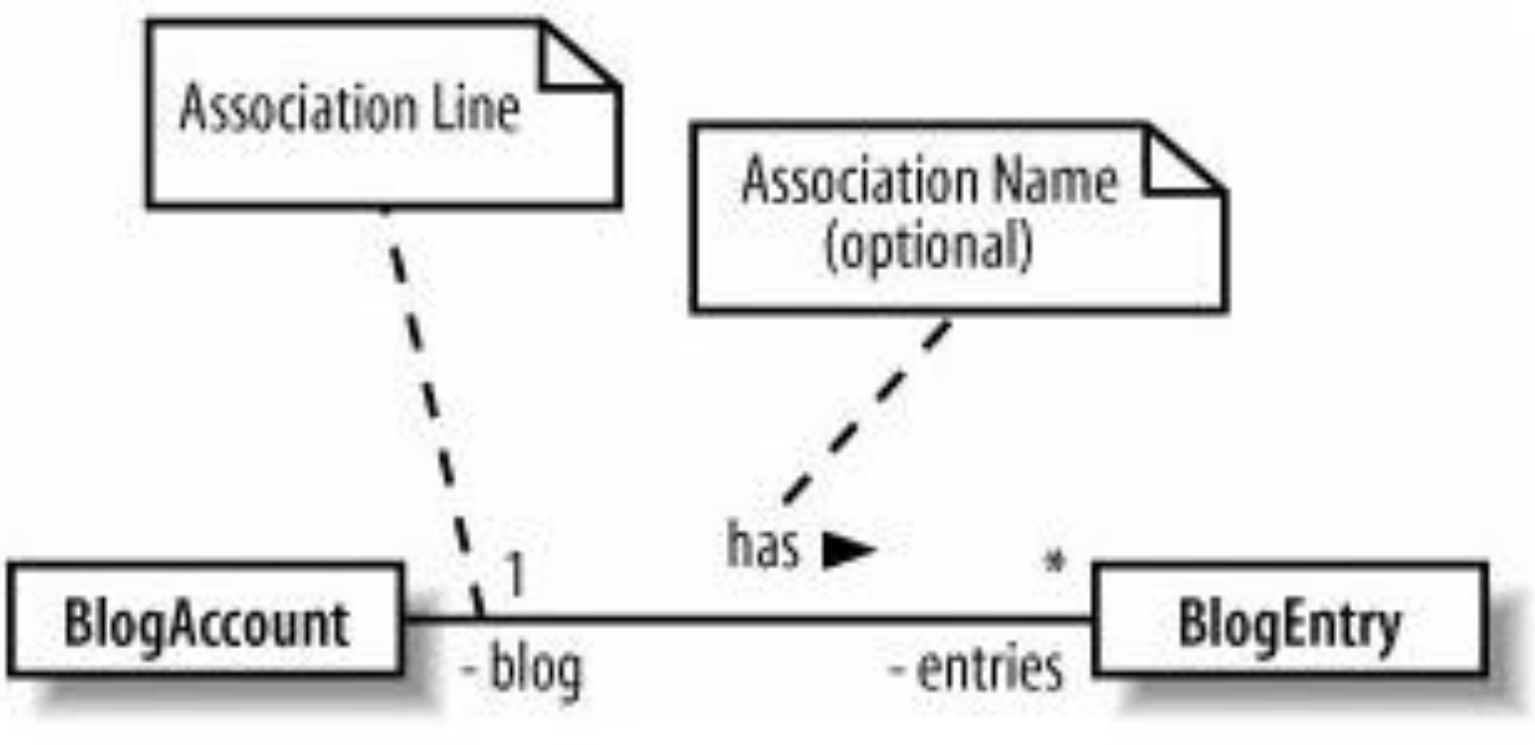
# Dependencia



# Asociación

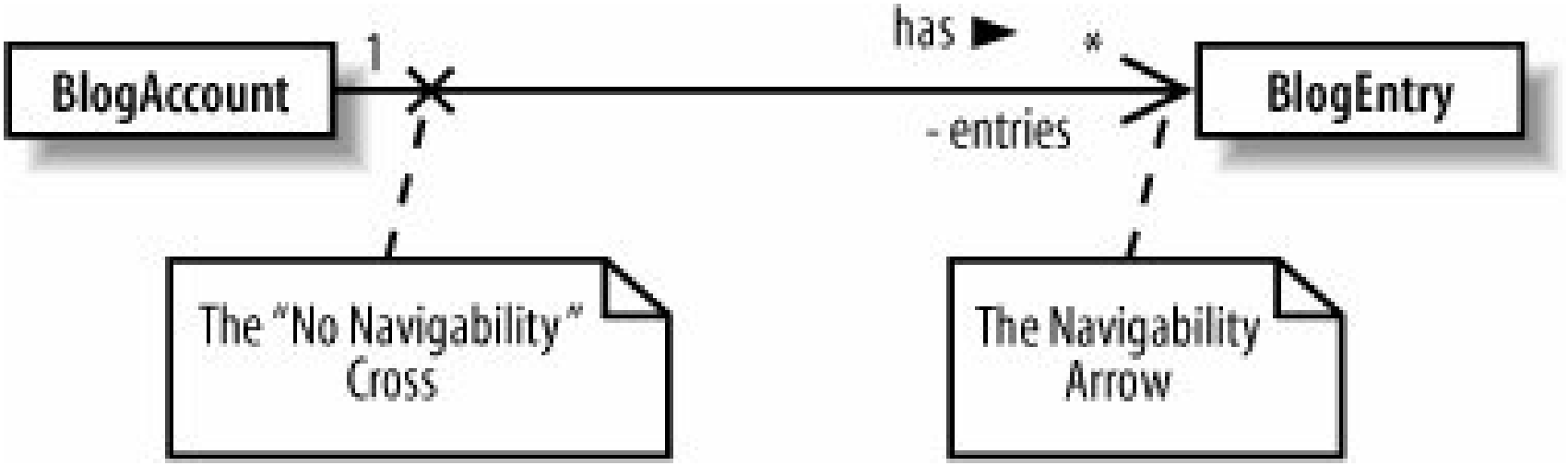
- La dependencia sólo permite a una clase utilizar objetos de otra clase
- La asociación significa que una clase en realidad contienen una referencia a un objeto u objetos, de otra clase en la forma de un atributo
- Si encuentras que una clase trabaja con un objeto de otra clase, entonces la relación entre las clases es un gran candidato para la asociación en lugar de una dependencia
- Asociación se muestra con una línea simple conexión de dos clases

# Asociación


Association Line
Association Name
(optional)
has
1
*
BlogAccount
BlogEntry
- blog
- entries

# Asociación

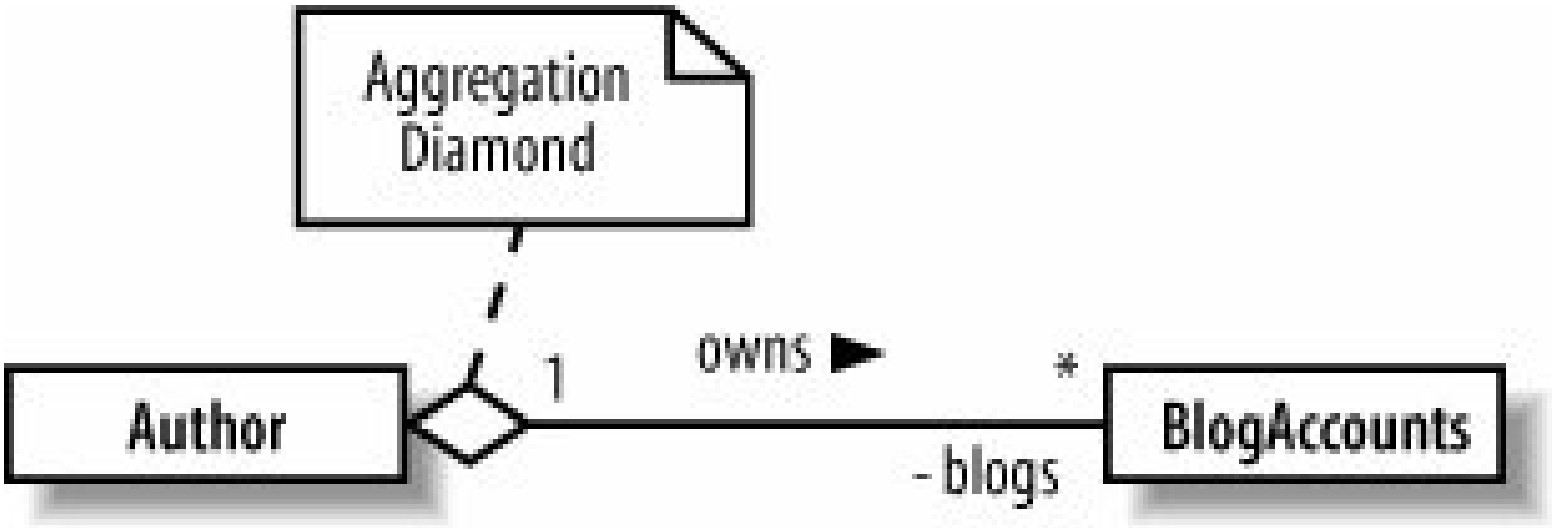
- La navegabilidad se aplica a menudo a una relación de asociación para describir qué clase contiene el atributo que soporta la relación



# Agregación

- La agregación es en realidad una versión más fuerte de la asociación y se utiliza para indicar que una clase realmente posee, pero puede compartir objetos de otra clase
- Agregación se muestra mediante el uso de una punta de flecha en forma de diamante vacío al lado de la clase poseedora
- La agregación denota que la clase poseedora utiliza los objetos para su funcionamiento

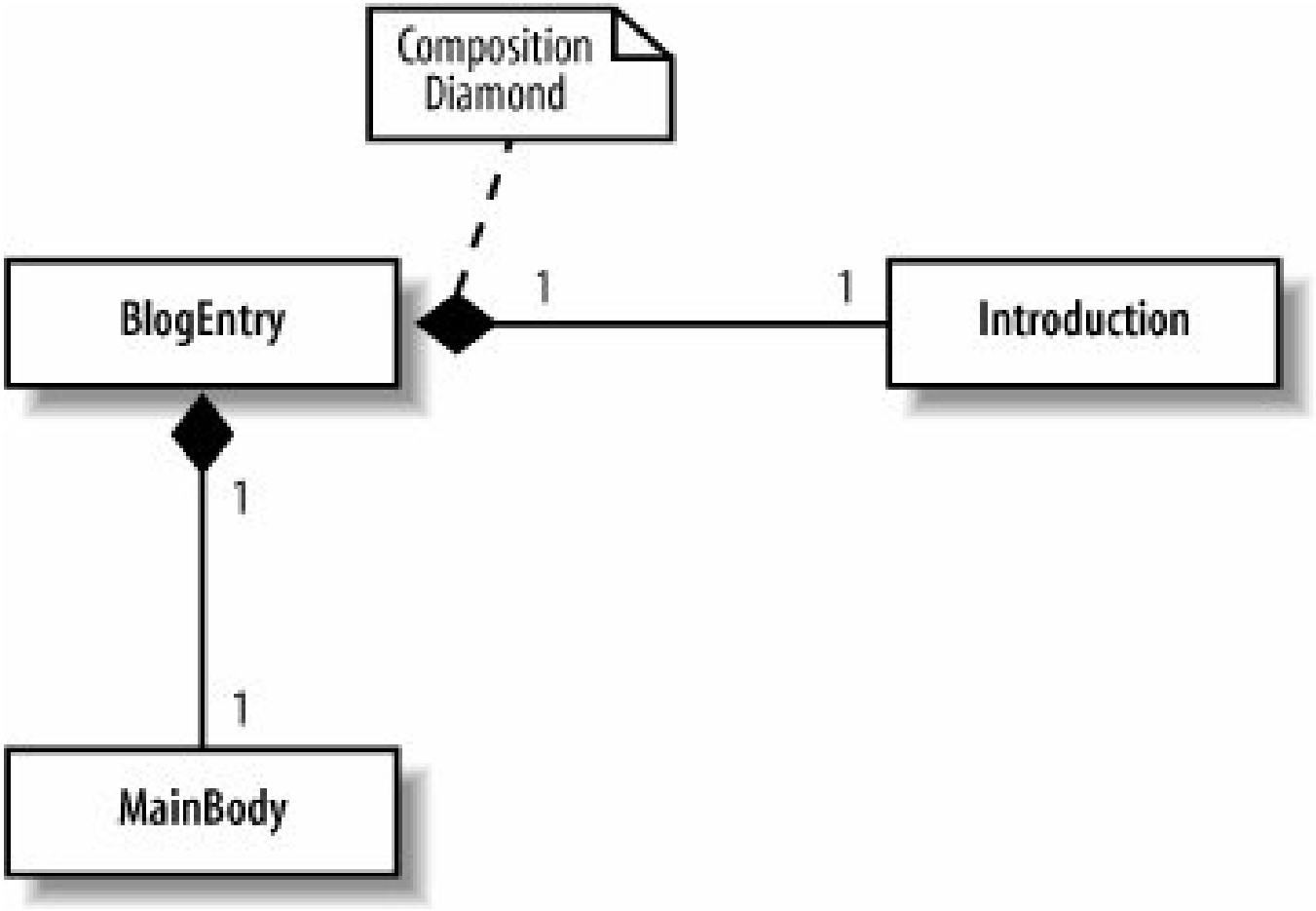
# Agregación



# Composición

- La composición es una relación aún más fuerte que la agregación, aunque trabajan de manera muy similar
- La composición denota que la clase poseedora contiene los objetos que compone, es decir es una relación Parte-Todo
- La Composición se muestra usando una punta de flecha en forma de diamante cerrado, o lleno

# Composición



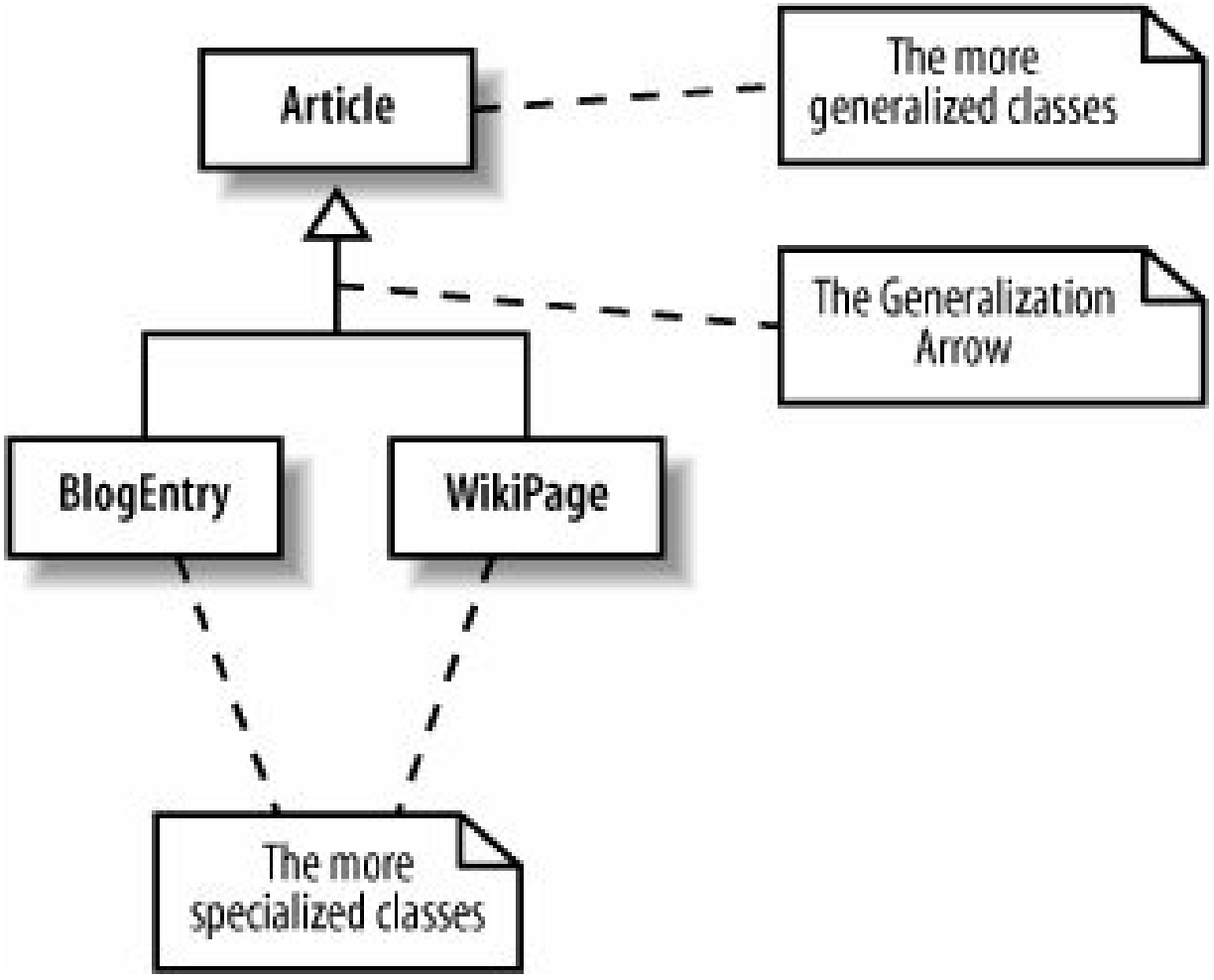
# Generalización

- La generalización y la herencia se utilizan para describir que una clase es un tipo de otra clase
  - Si encuentra que una clase tiene una parte que es objeto de otra clase, entonces la relación más probable será una asociación, agregación o composición
  - Si encuentra que la clase es un tipo de otra clase, entonces es posible que desee considerar el uso de la generalización
- En UML, la flecha de generalización se usa para indicar que una clase es un tipo de otra clase

# Generalización

- En UML, la flecha de generalización se usa para indicar que una clase es un tipo de otra clase
- Las Clases padre describen un tipo más general, que luego se hizo más especializado en las clases hijo

# Generalización



# Herencia Múltiple

- La herencia múltiple o la generalización múltiples en la terminología oficial de UML se produce cuando una clase hereda de dos o más clases padre
- Aunque la herencia múltiple es compatible con UML, aún no es considerada como buenas prácticas en la mayoría de los casos
  - Esto se debe principalmente al hecho de que la herencia múltiple presenta un problema complicado, cuando las dos clases padre superponen características o comportamiento.

# Herencia Múltiple