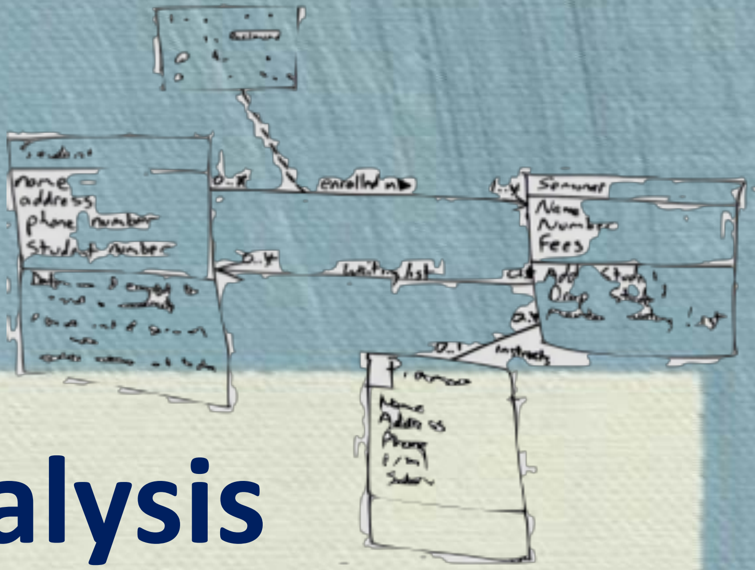


Software Modeling and Analysis
CS 284

Lab 4

Class Diagrams (Part 1)

Quick Review

- **Relations in use case diagrams**
- <<include>>
- generalization
- <<extend>>

In today’s session you’ll ..

- Understand the purpose and function of the class diagrams
- Learn how to model the class compartments
- Learn how to model the class attributes
- Learn how to model the class operations

Objects and Classes

- An object is any person, place, thing, concept, event, screen, or report applicable to your system.

- Objects have attributes and they have methods.

- A class is a representation of an object and, in many ways, it is simply a template from which objects are created.

Objects and Classes (Cont.)

- Classes form the main building blocks of an object-oriented application.
- Example:
  - Although thousands of students attend the university, you would only model one class called Student.
  - Students have student numbers, names, addresses, and phone numbers. Those are all examples of the attributes of a student. Students also enroll in courses, drop courses, and request transcripts which represent what the student can do (methods).

Class Diagrams

- A system's structure is made up of a collection of pieces often referred to as objects.

- Classes describe the different types of objects that your system can have.

- **A class diagram describes the types of objects (i.e., classes) in the system and the various kinds of static relationships that exist among them.**

- **Class diagrams also show the properties and operations of a class and the constraints that apply to the way objects are connected.**

Class Diagrams (Cont.)

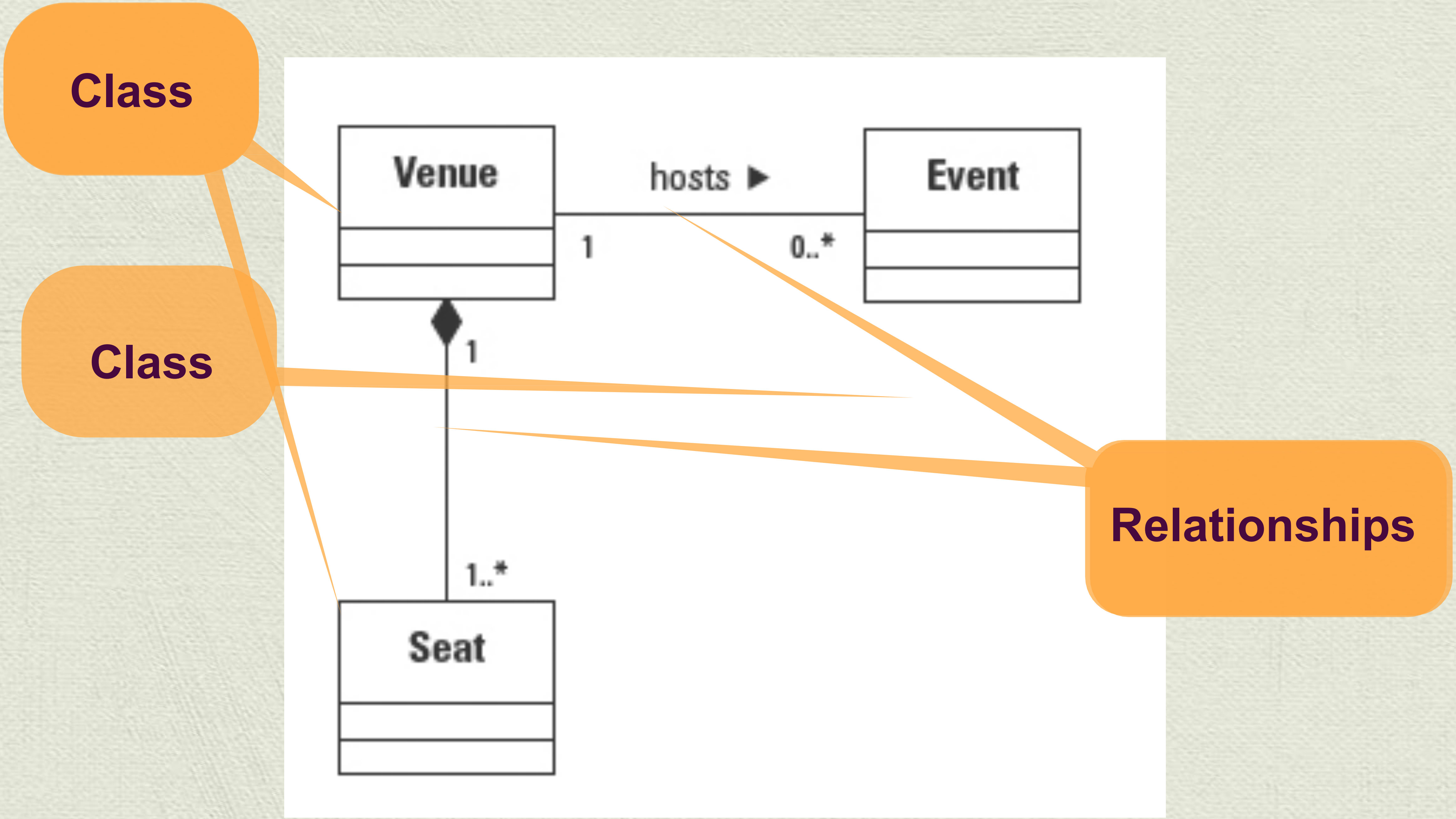
- Class diagrams are the primary source for forward engineering (**turning a model into code**), and the target for reverse engineering (**turning code into a model**).

- The rules in the Class diagram are used to generate code.

- The code generates objects, while the application is running,

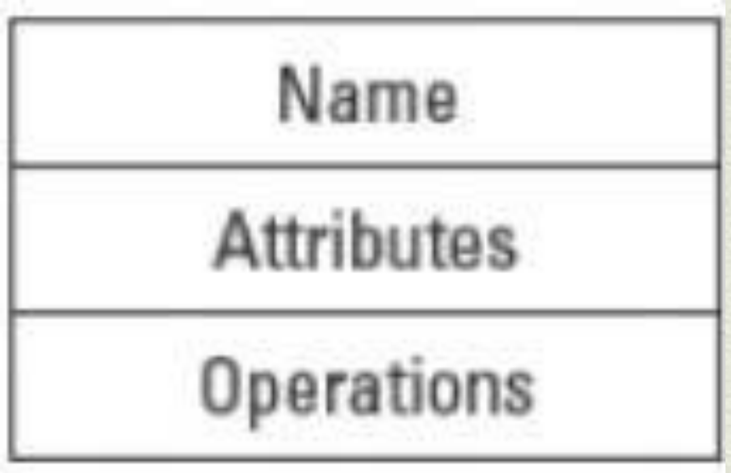
that behave according to the rules defined by the Class diagram.

An Overview of the Graphical Notations



Modeling a Class

- Classes form the foundation of the Class diagram.
- a class in UML is drawn as a rectangle split into up to three compartments (sections):
  - The top section contains the name of the class.
  - The middle section contains the attributes.
  - The final section contains the operations.



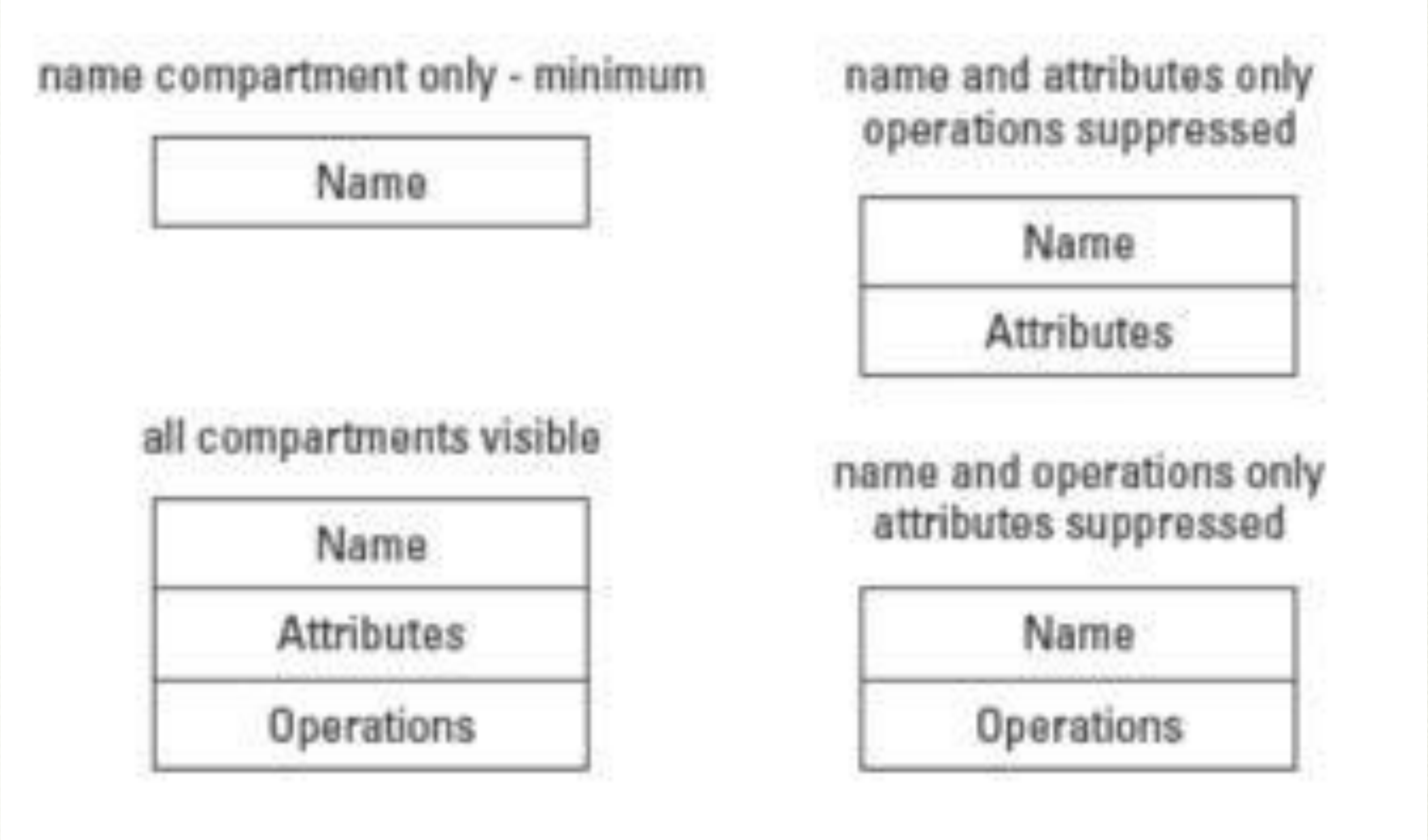
Modeling a Class (Cont.)

- The attributes and operations sections are optional.

- Hiding them does not change the fact that they exist.

- It merely enables you to keep the people who are reviewing your Class diagram focused on the elements about which you need their insights.

Modeling a Class (Cont.)



name compartment only - minimum

Name

name and attributes only
operations suppressed

Name

Attributes

all compartments visible

Name

Attributes

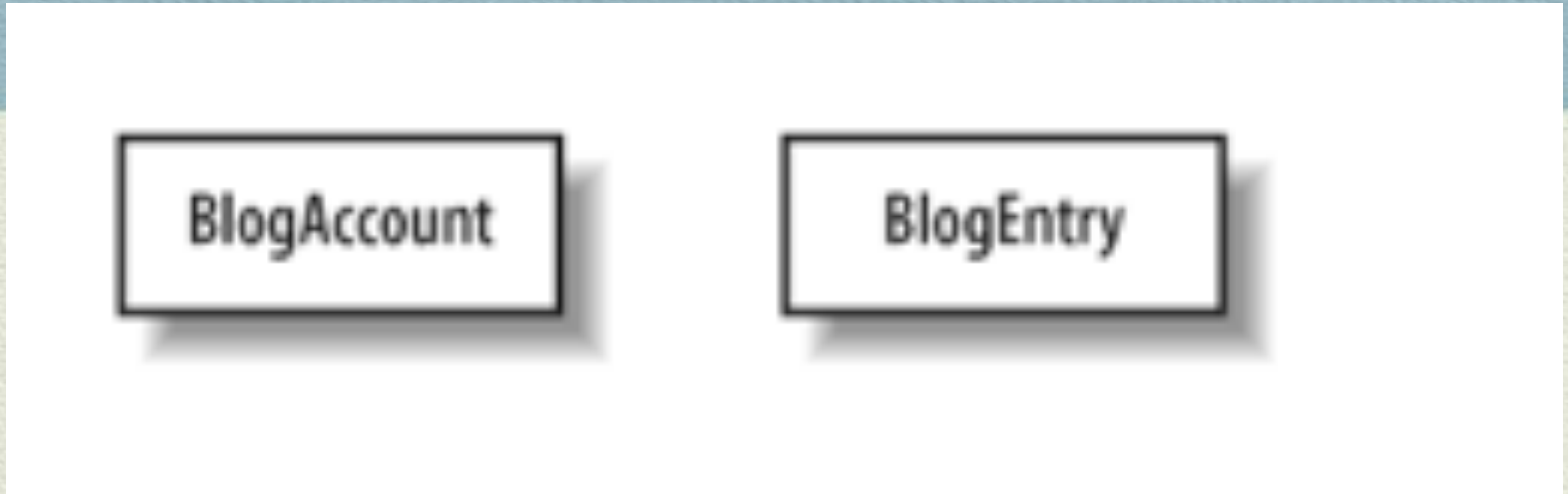
Operations

name and operations only
attributes suppressed

Name

Operations

Modeling a Class–Example



- The BlogAccount class defines the information that the system will hold relating to each of the user's accounts.
- The BlogEntry class defines the information contained within an entry made by a user into her blog.
-

Modeling the Name Compartment–Class Name

- The name always resides in the topmost compartment.
- The name is nearly always a singular noun or noun phrase such as User and BlogAccount.
- The capitalization rules for a class name typically correspond to the language that will be used to code the application.
- Since the code generated from class names usually does not support spaces in the name, it is a good idea to use underscores or hyphens or simply no spaces between the words.

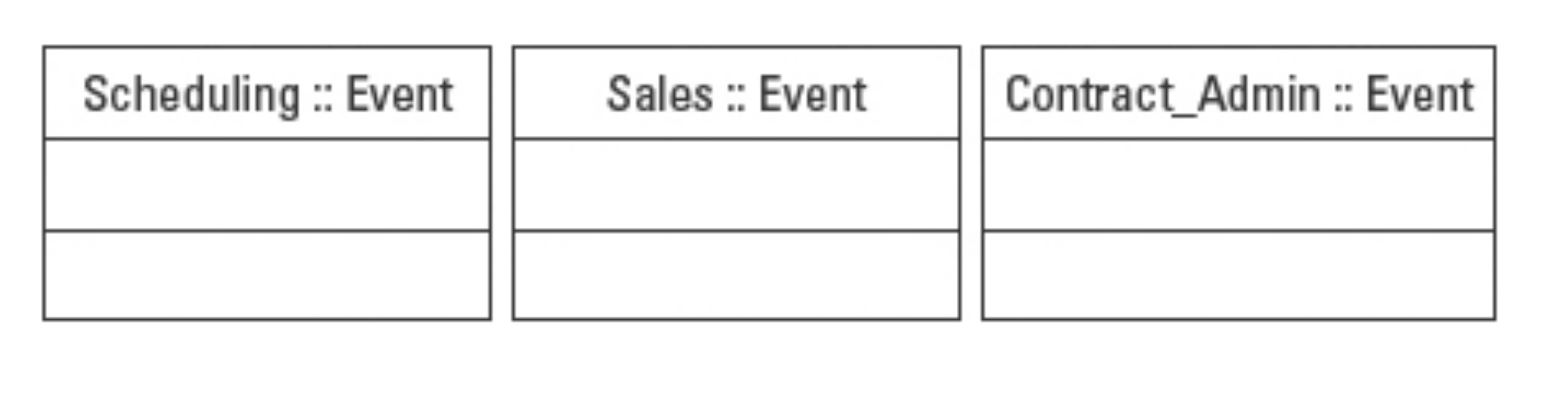
Modeling the Name Compartment– Class Name (Cont.)

- A class name must be unique within a package.
- The same class name may occur in multiple packages.
- This redundancy often happens when systems are worked on by different teams or developed as parts of different projects.
- To clarify which class you mean to reference you must qualify the class name with the name of the package that owns it.

Modeling the Name Compartment– Class Name (Cont.)

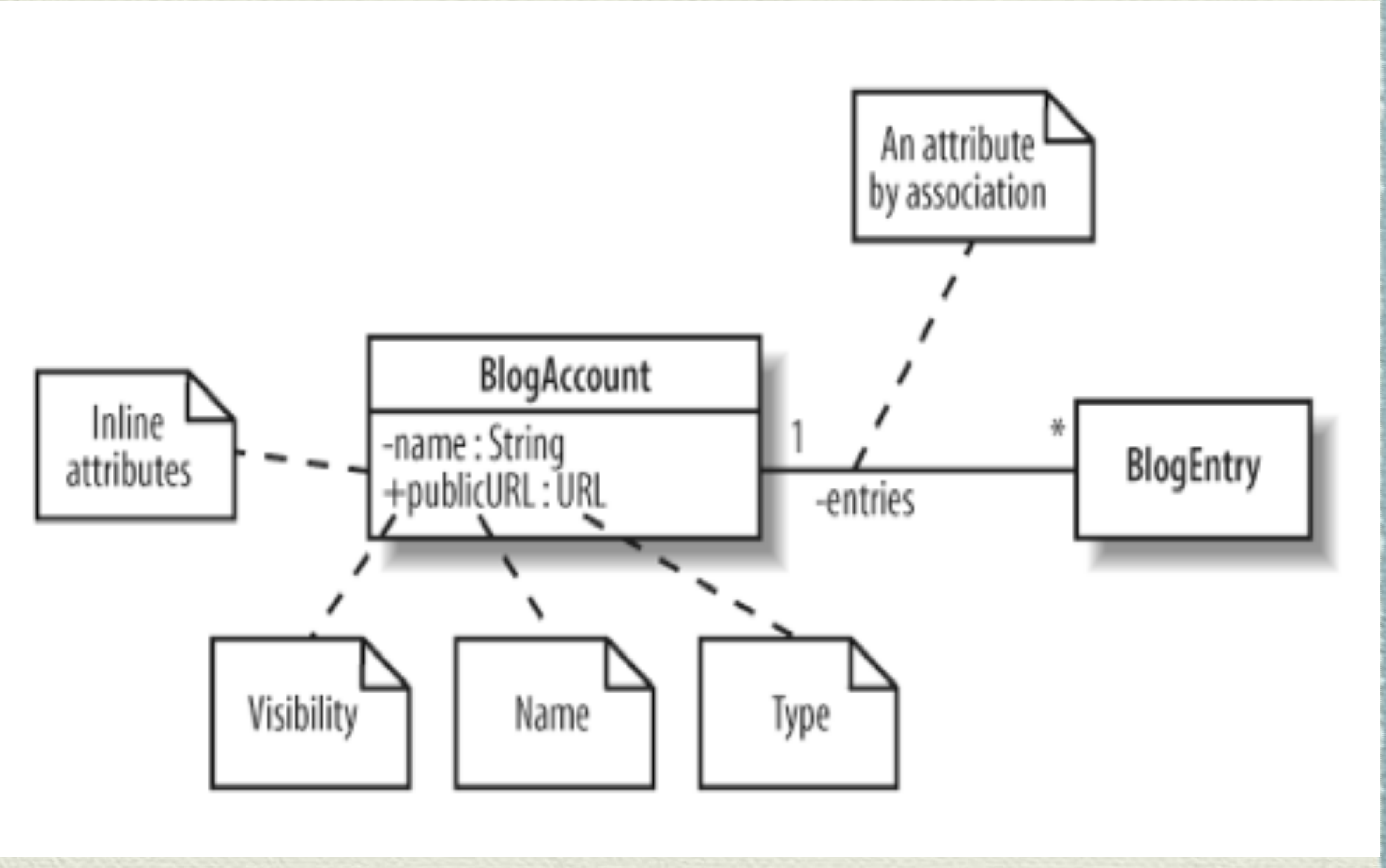
- The format for a fully qualified class name is:

Package_Name :: Class_Name



Modeling the Attributes

- Attributes can be represented on a class diagram either by placing them inside their section of the class box—known as inline attributes —or by association with another class.



Inline or by Association?!

- Use inline attributes for small things, such as dates or Booleans—in general, value types.

- Use attribute by associations for more significant classes, such as BlogAccount, Customer, Student..etc.

Modeling the Attributes (cont.)

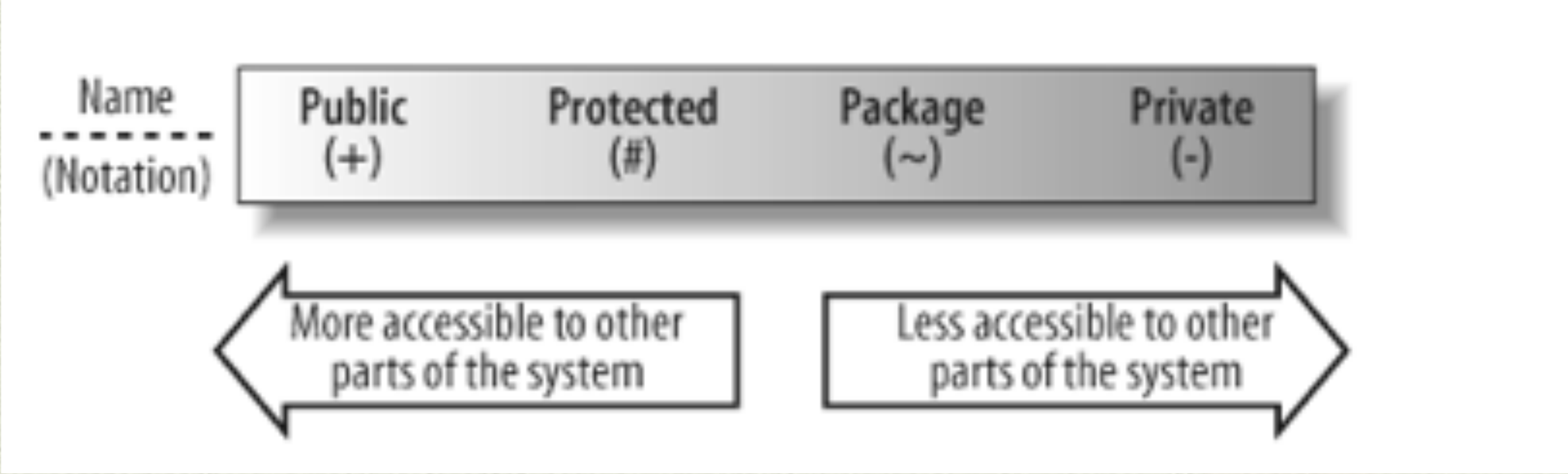
- The **attribute** notation describes a property as a line of text (signature) within the class box itself. The full form of an attribute is:

visibility name: type multiplicity = default {property strings}

- Your attribute will usually have a signature that contains a visibility property, a name, and a type, although the attribute's name is the only part of its signature that absolutely must be present.

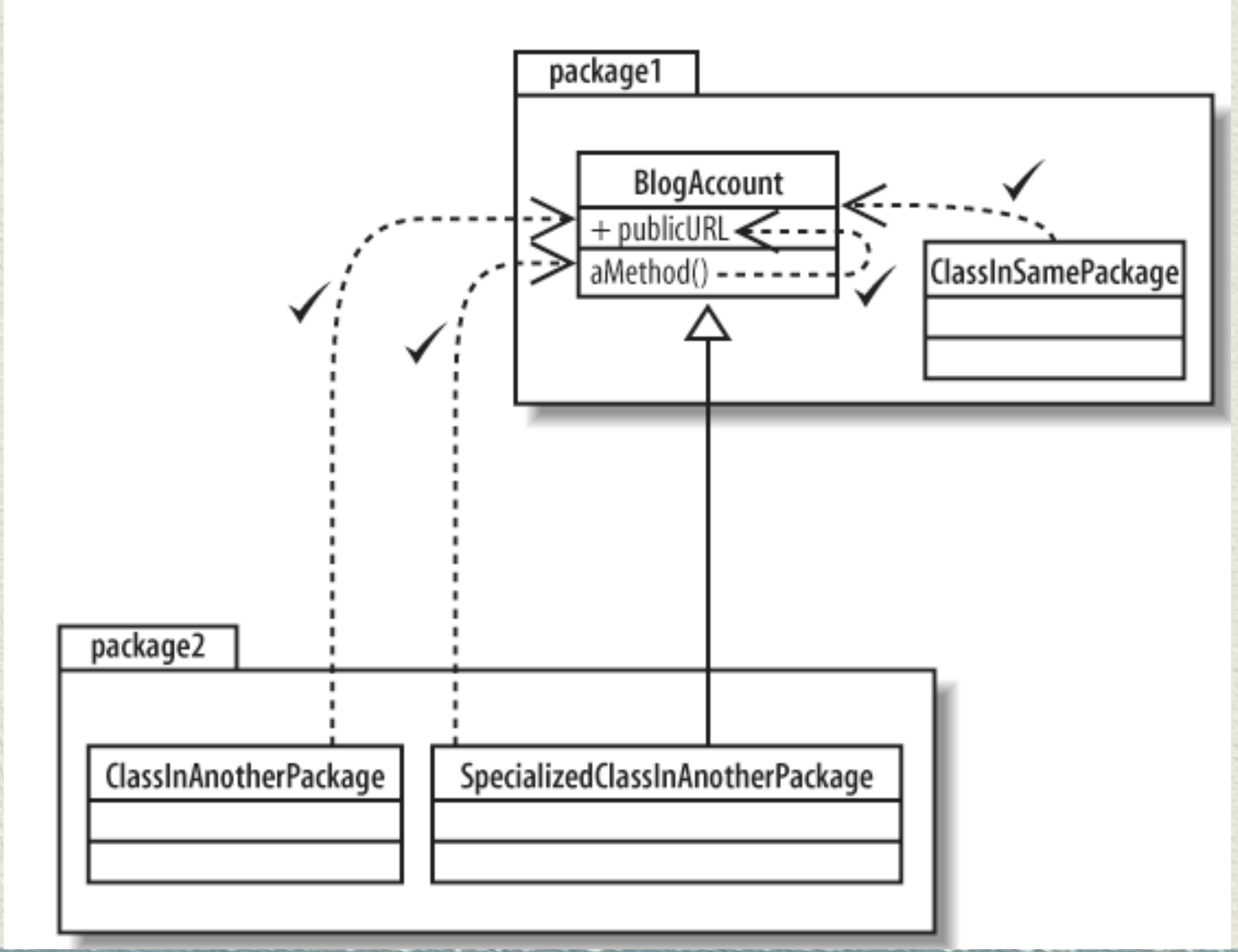
Visibility

- **How does a class selectively reveal its operations and data to other classes? By using visibility.**
- There are four different types of visibility that can be applied to the elements of a UML model.
- These visibility characteristics will be used to control access to both attributes, operations, and sometimes even classes


Name
(Notation)
Public
(+)
Protected
(#)
Package
(~)
Private
(-)
More accessible to other parts of the system
Less accessible to other parts of the system

Public Visibility

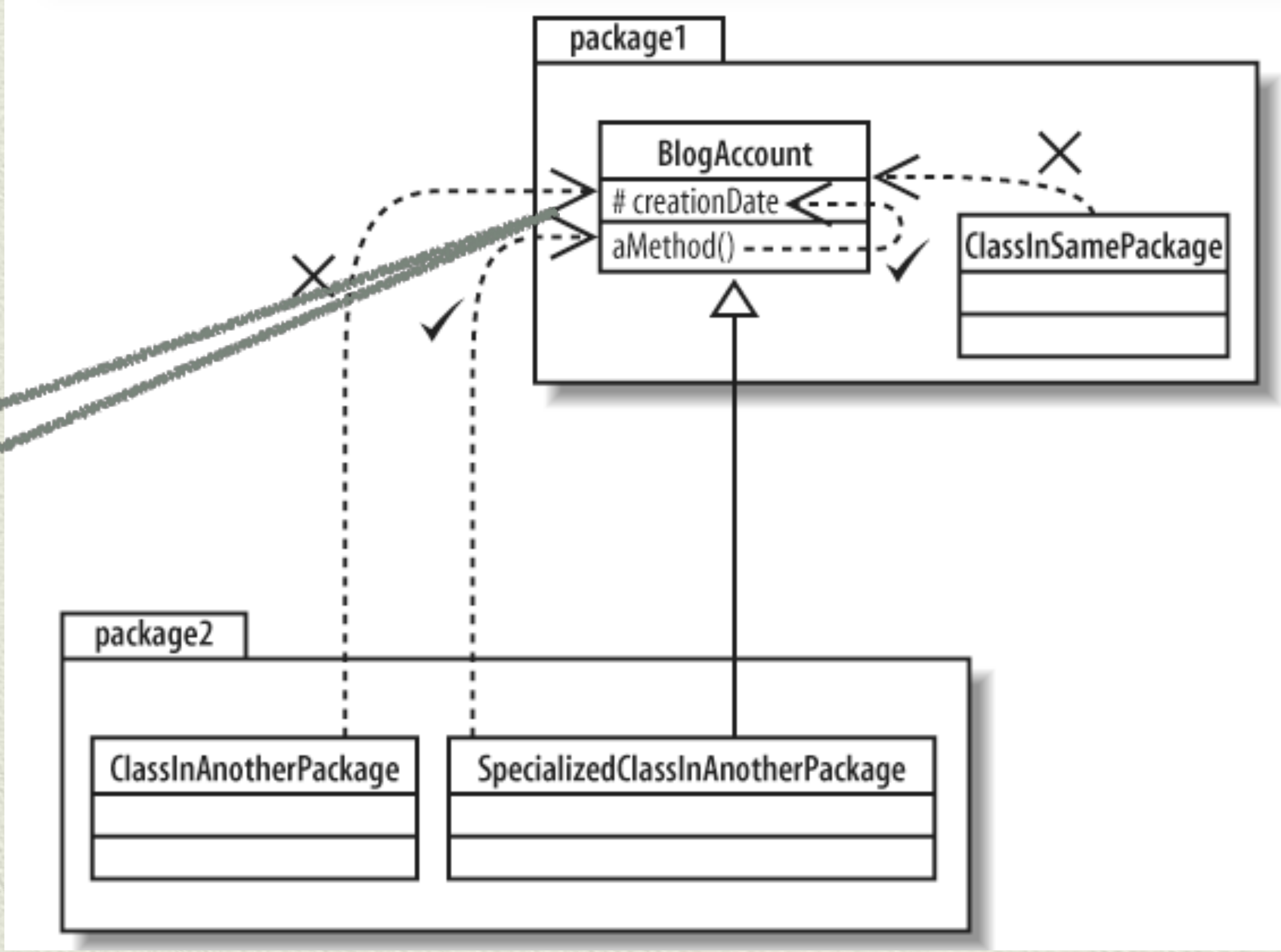
- Public visibility is the most accessible of visibility characteristics.
- Specified using the plus (+) symbol before the associated attribute or operation.



Protected Visibility

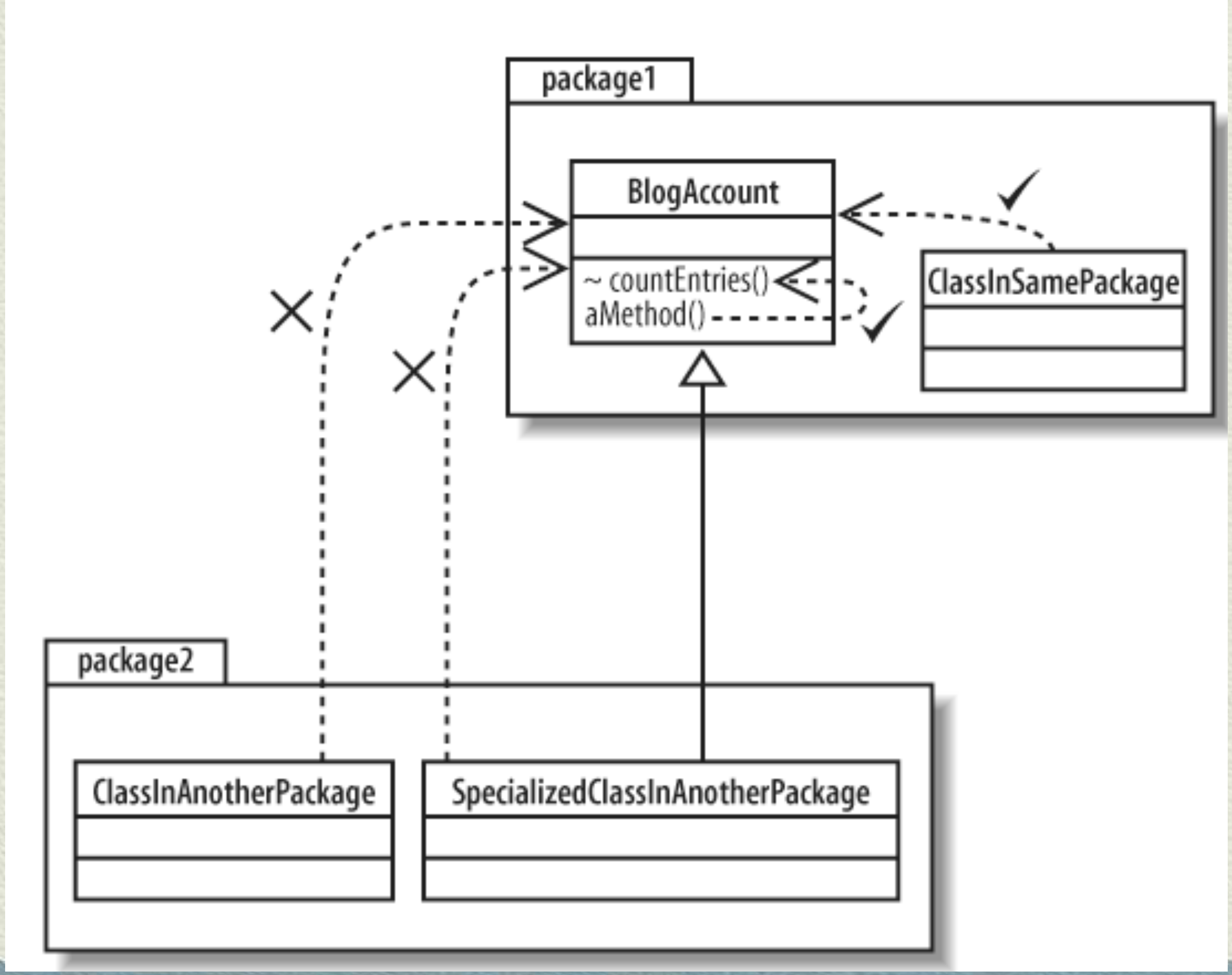
- Protected elements on classes can be accessed by methods that are part of your class and also by methods that are declared on any class that inherits from your class.
  - Protected attributes and operations are specified using the hash (#).

"This attribute or operation is useful inside my class and classes extending my class, but no one else should be using it."



Package Visibility

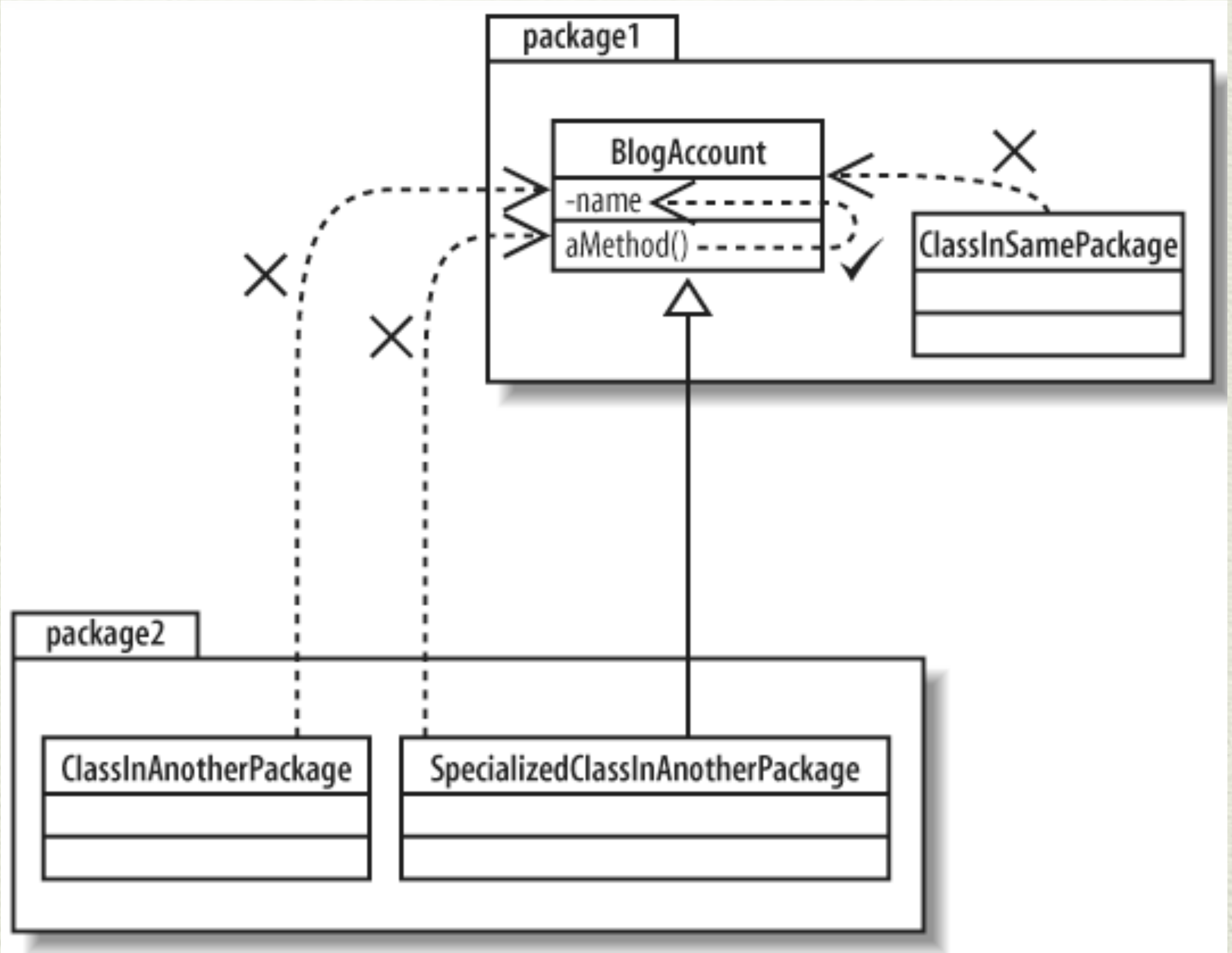
- If you add an attribute or operation that is declared with package visibility to your class, then any class in the same package can directly access that attribute or operation.
- Specified with a tilde (~)



Private Visibility

- Only the class that contains the private element can see or work with the data stored in a private attribute or make a call to a private operation.

- Specified with a minus (-)



Private Visibility

- It's a commonly accepted rule of thumb that attributes should always be private and only in extreme cases opened to direct access by using something more visible.
- The exception to this rule is when you need to share your class's attribute with classes that inherit from your class.
- In this case, it is common to use protected.
- In well-designed OO systems, attributes are usually private or protected, but very rarely public.

Attribute Name

- An attribute's name can be any set of characters, but no two attributes in the same class can have the same name.
- One of the primary aims of modeling your system is to communicate your design to others so make sure that the name accurately describes what is being named.
- Check to make sure that the name meets the conventions of the programming language.

Attribute Type

- The type of attribute can vary depending on how the class will be implemented in your system but it is usually either a class, such as String, or a primitive type, such as an int in Java.

Multiplicity

- Sometimes an attribute will represent more than one object.
- In fact, an attribute could represent any number of objects of its type.
- **This is like declaring that an attribute is an array.**
- Multiplicity allows you to specify that an attribute actually represents a collection of objects, and it can be applied to both inline and attributes by association.

Multiplicity (Cont.)

- Multiplicity is modeled as a value expression.
- When multiplicity is used in an inline attribute, the value expression is enclosed within square brackets ([]).
- When multiplicity is used to adorn a diagram notation like an association, it has no enclosing brackets.
- **Multiplicity can express a range of values, a specific value, a range without limit.**

Multiplicity (Cont.)

- **Range of values**

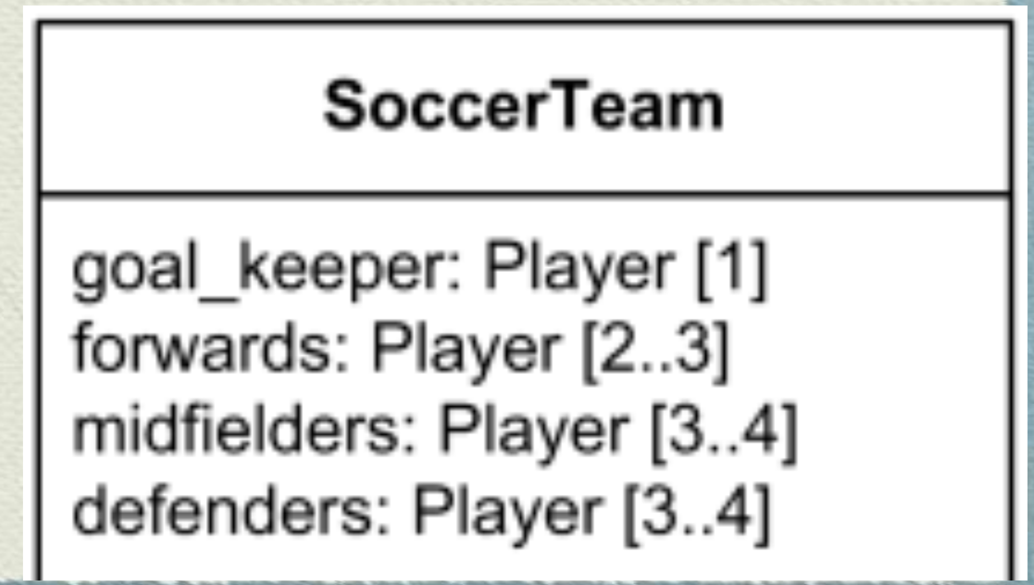
A range of values includes a lower and an upper value separated by two periods, as in [0..5] or 0..5.

- **Specific values**

When the upper and lower values in a range are the same, the UML allows the use of the upper value by itself, as in [2] or 2

- **Range without limit**

When the upper value is unknown or unspecified, the UML uses the asterisk (*) in place of a number value, as in [1..*], which means one or more.



SoccerTeam

goal_keeper: Player [1]
forwards: Player [2..3]
midfielders: Player [3..4]
defenders: Player [3..4]

Attribute Properties

- There is also a set of properties that can be applied to attributes to completely describe an attribute's characteristics.

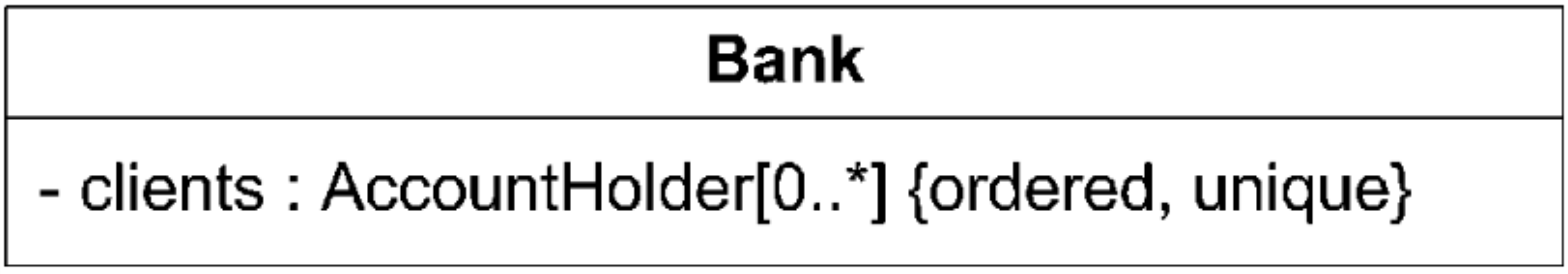
- Examples of properties

  - readOnly: Specifies that the attribute may not be modified once the initial value is set.

  - ordered: An attribute with a multiplicity greater than 1 can be specified to be *ordered*.

  - unique: an attribute with multiplicity greater than 1 may be required to be *unique*.

Attribute Properties–Example



Bank

- clients : AccountHolder[0..*] {ordered, unique}

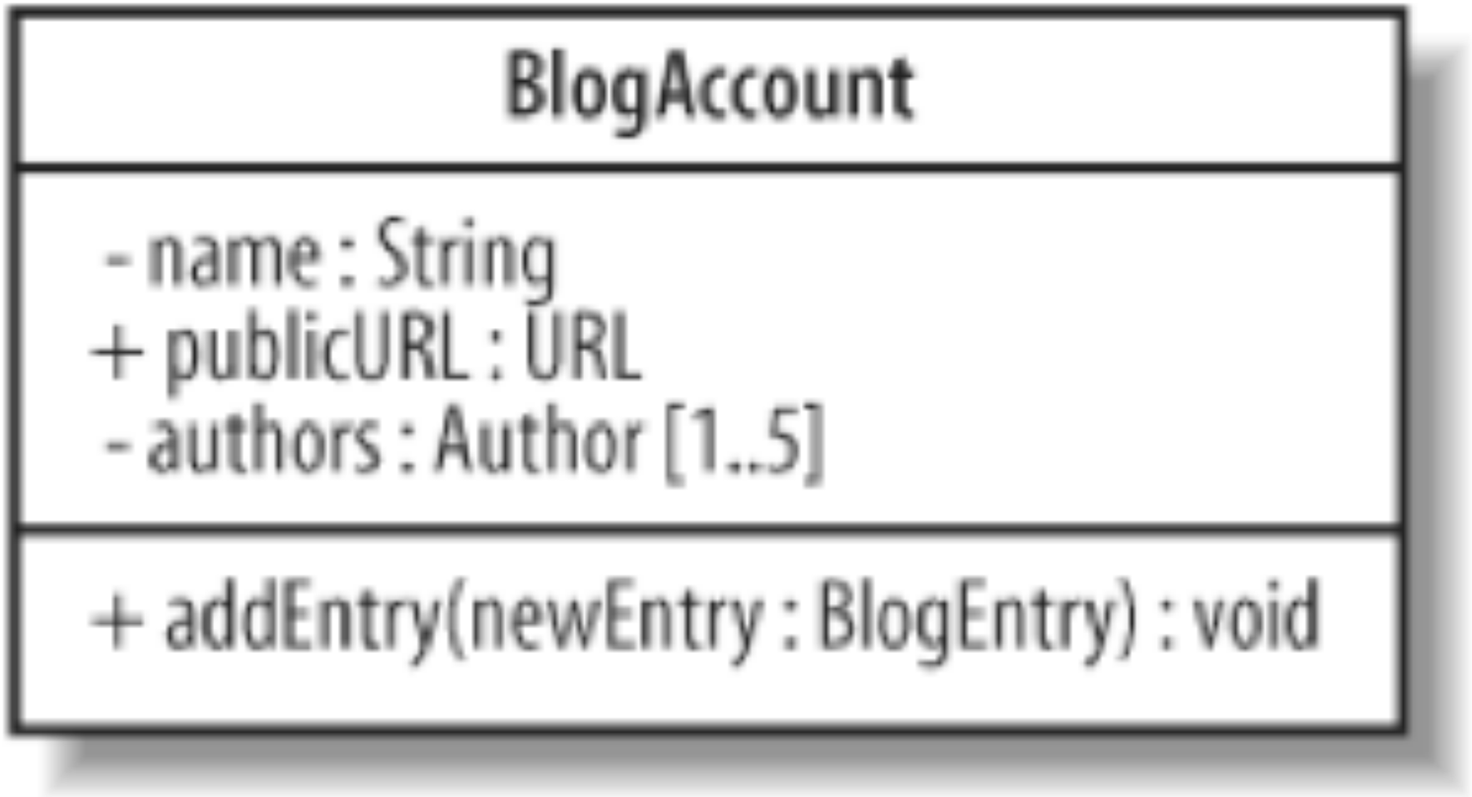
Modeling Operations

- **Operations are the actions that a class knows to carry out**.
- Operations most obviously correspond to the methods on a class.
- **The full UML syntax for operations is:**

visibility name (parameter-list) : return-type {property-string}

- Operations in UML are specified on a class diagram with a signature that is at minimum made up of **a visibility property, a name, a pair of parentheses** in which any parameters that are needed for the operation to do its job can be supplied, and **a return type**.

Modeling Operations–Example



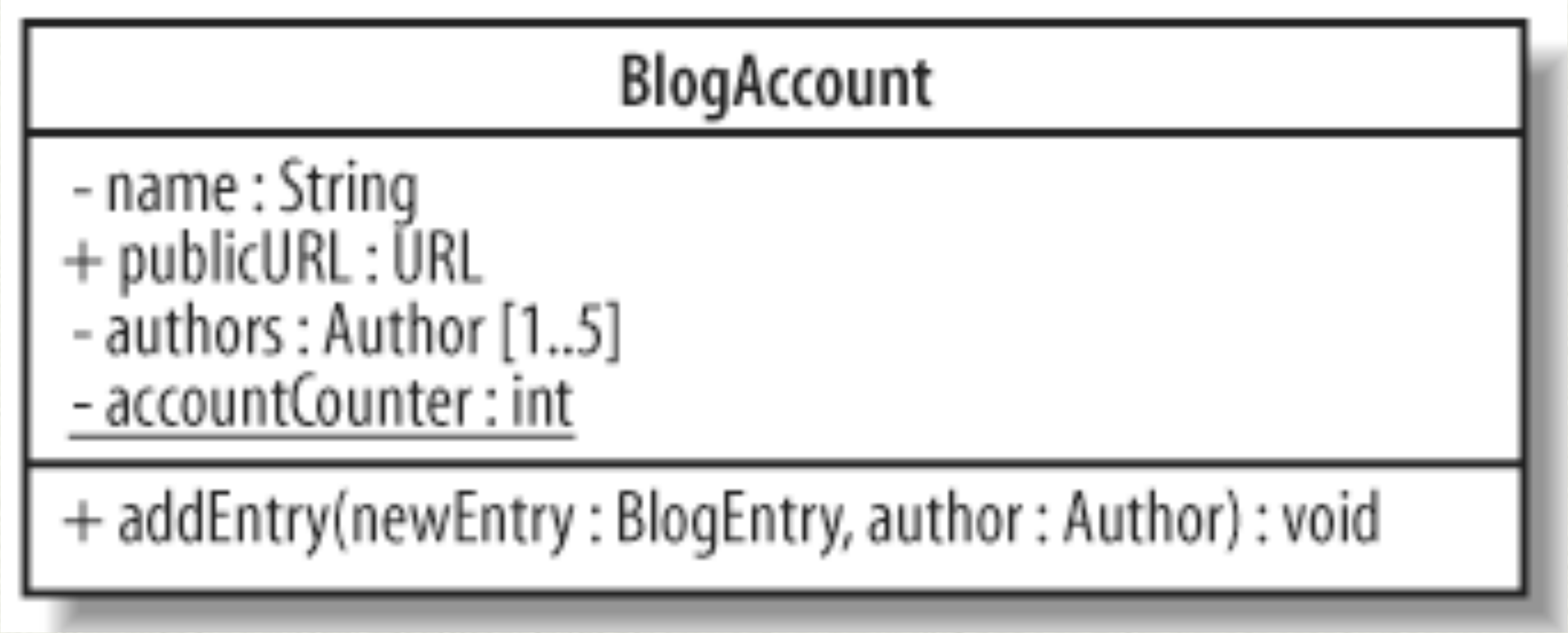
BlogAccount

- name : String
+ publicURL : URL
- authors : Author [1..5]

+ addEntry(newEntry : BlogEntry) : void

Static Parts of the Class

- In UML, operations, attributes, and even classes themselves can be declared static.
- You represent static parts by underlining them.



BlogAccount

- name : String
+ publicURL : URL
- authors : Author [1..5]
- accountCounter : int

+ addEntry(newEntry : BlogEntry, author : Author) : void

References

- Fowler, M. (2004). UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional.
- Miles, R and Hamilton, K. (2006) Learning UML 2.0. Sebastopol: O'Reilly Media, Inc.
- Pender, T (2003). UML Bible. John Wiley & Sons, Inc., New York, NY.
- Pilone, D., & Pitman, N. (2006). *UML 2.0 in a nutshell*. O’Reilly.