4.Modeling a System's Logical Structure: Introducing Classes and Class Diagrams 类图

Shaoning Zeng, http://zsn.cc

提问时间

What we Learned?

- 3. Modeling System Workflows: Activity Diagrams
  - 3.1. Activity Diagram Essentials
  - 3.2. Activities and Actions
  - 3.3. Decisions and Merges
  - 3.4. Doing Multiple Tasks at the Same Time (forks and joins)
  - 3.5. Time Events
  - 3.6. Calling Other Activities
  - 3.7. Objects
  - 3.8. Sending and Receiving Signals
  - 3.9. Starting an Activity
  - 3.10. Ending Activities and Flows
  - 3.11. Partitions (or Swimlanes)
  - 3.12. Managing Complex Activity Diagrams

- Activity – Process – Flow | Path

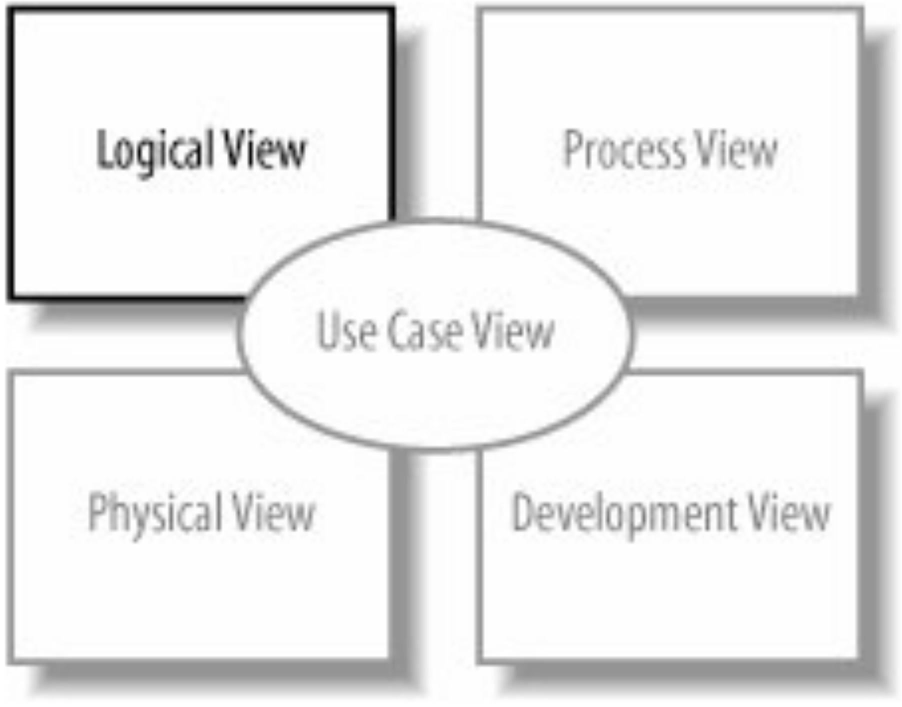
4. Modeling a System's Logical Structure: Introducing Classes and Class Diagrams

- 4.1. What Is a Class? 什么是类？
- 4.2. Getting Started with Classes in UML
- 4.3. Visibility 可见性
- 4.4. Class State: Attributes 属性
- 4.5. Class Behavior: Operations 操作
- 4.6. Static Parts of Your Classes

4. Modeling a System's Logical Structure: Introducing Classes and Class Diagrams

- A system's structure is made up of a collection of pieces often referred to as *objects*. 系统由对象构成
- Classes describe the different types of objects that your system can have, and class diagrams show these classes and their relationships (Chapter 5). 类是对象的种类
- Use cases describe the behavior of your system as a set of concerns. 用例描述系统行为
- Classes describe the different types of objects that are needed within your system to meet those concerns. 类描述系统完成这些行为所需要的对象种类

Figure 4-1. The Logical View on your model contains the abstract descriptions of your system's parts, including classes 逻辑视图

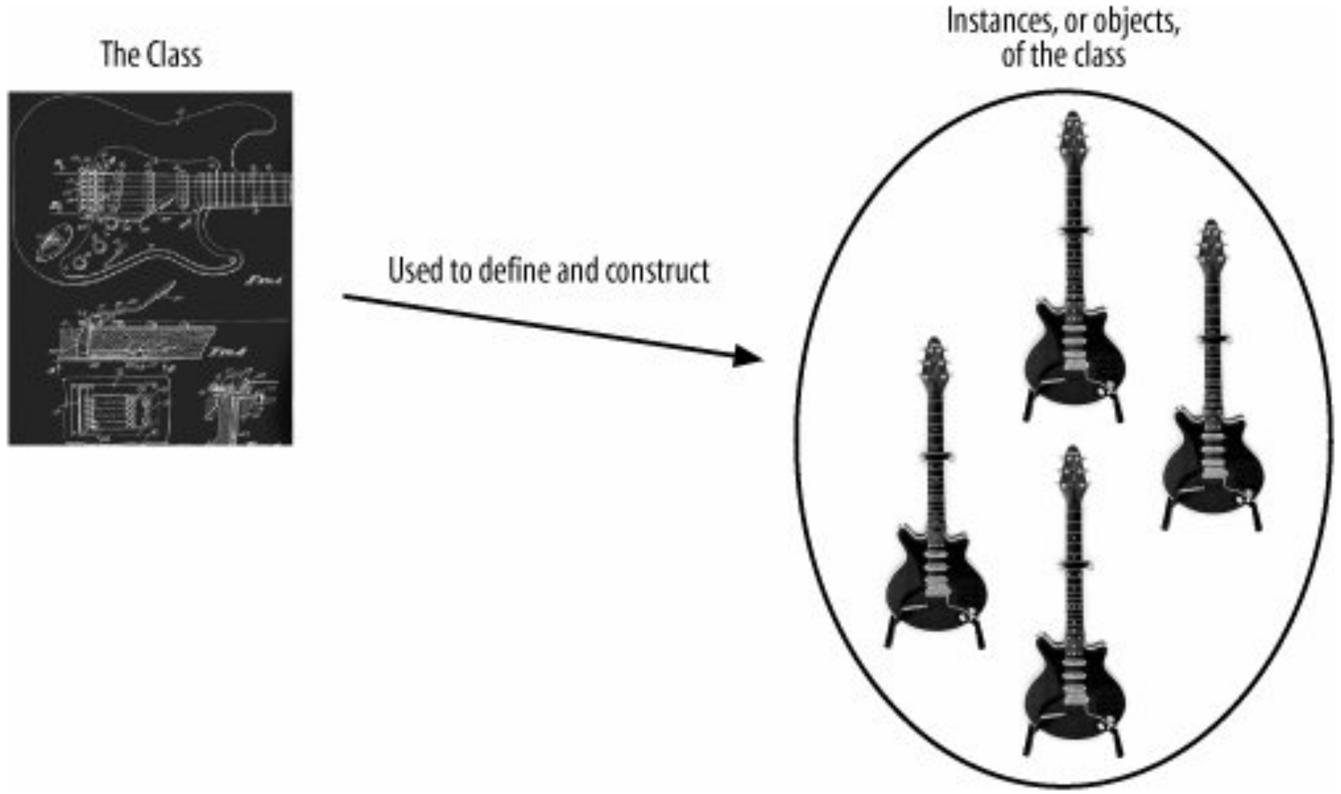


4.1. What Is a Class?

- It has an identity: it's the one I own. 标识|身份|ID
  - However, I'm not going to pretend that Burns made only one of this type of guitar and that it was just for me. I'm not that good a guitarist!
  - Burns as a company will make hundreds of this type of guitar or, to put it another way, this class of guitar.
- A class is a type of something. 类型
  - You can think of a class as being the blueprint out of which objects can be constructed

Figure 4-2. One of my guitars: a good example of an object

Figure 4-3. The class defines the main characteristics of the guitar; using the class, any number of guitar objects can be constructed


The Class
Used to define and construct
Instances, or objects, of the class

class and instance 类与实例

- Each guitar constructed from the class can be referred to as an instance or object of the class, and so my guitar is an instance of the Burns BMS Guitar class.
- A class's description will include two pieces of information: 类描述包括2个方面
  - the state information that objects of the class will contain 状态
  - and the behavior that they will support. 行为
- Attributes and operations are the mainstays of a class's description. 属性与操作
  - Together, they enable a class to describe a group of parts within your system that share common characteristics such as state represented by the class's attributes and behavior represented by the class's operations

Important OO characteristics

Abstraction

- 抽象

Encapsulation

- 封裝

4.1.1. Abstraction 抽象

- A class's definition contains the details about that class that are important to you and the system you are modeling.
  - For example, my BMS guitar might have a scratch (刮痕) on the back or several but if I am creating a class that will represent BMS guitars, do I need to add attributes that contain details about scratches?
  - I might if the class were to be used in a repair shop; however, if the class were to be used only in the factory system, then scratches are one detail that I can hopefully ignore.
- Discarding (摒弃) irrelevant details within a given context is called *abstraction*.

4.1.2. Encapsulation 封装

- According to the object-oriented approach to system development, for an object to be an object, it needs to contain both data attributes and the instructions that affect the data operations. 封装的是属性与操作
- This is the big difference between object orientation and other approaches to system development:
  - in OO, there is the concept of an object that contains, or encapsulates, both the data and the operations that work on that data.
    面向对象同时包含数据与操作
    其他方法则将数据与操作分离

4.1.2. Encapsulation (Cont.)

- Encapsulation enables a class to hide the inner details of how it works from the outside world like the electrics from the example guitar class and only expose the operations and data that it chooses to make accessible.
- Encapsulation is very important because with it, a class can change the way it works internally and as long as those internals are not visible to the rest of the system, those changes will have no effect on how the class is interacted with. 好处是内部变化不影响外部接口
  - This is a useful feature of the object-oriented approach because with the right classes, small changes to how those classes work internally shouldn't cause your system to break.

4.2. Getting Started with Classes in UML

- At its simplest, a class in UML is drawn as a rectangle split into up to three sections. 矩形=名称+属性+操作
  - The top section contains the name of the class,
  - the middle section contains the attributes or information that the class contains,
  - and the final section contains the operations that represent the behavior that the class exhibits.
- The attributes and operations sections are optional.
  - If the attributes and operations sections are not shown, it does not necessarily imply that they are empty, just that the diagram is perhaps easier to understand with that information hidden. 属性与操作是可选的，可以隐藏信息

Figure 4-4. Four different ways of showing a class using UML notation

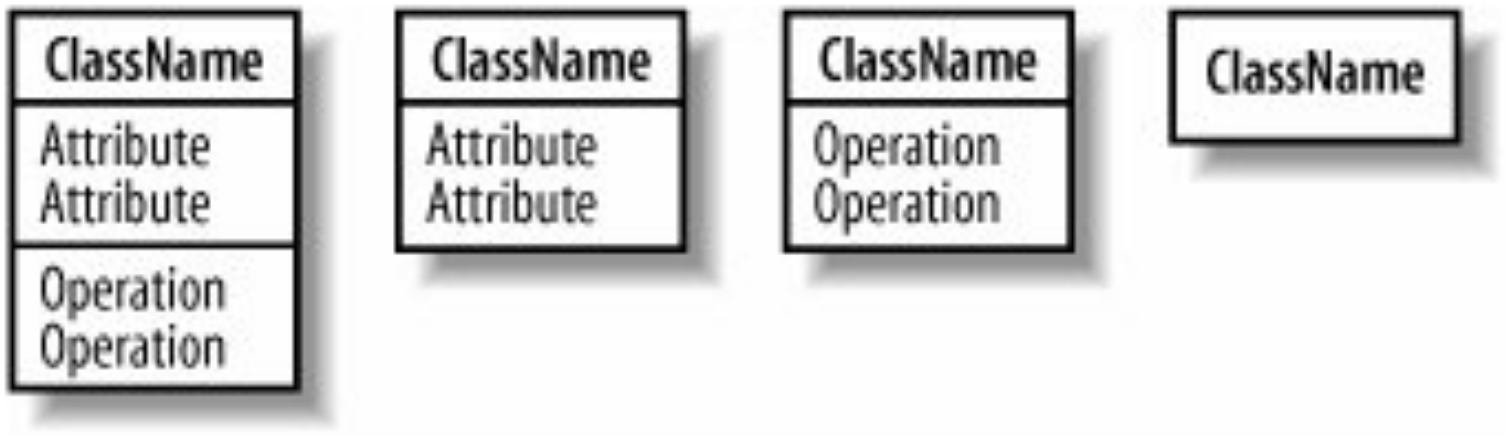
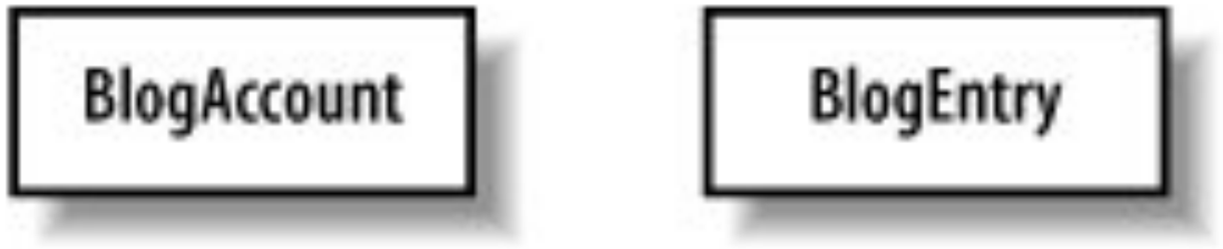


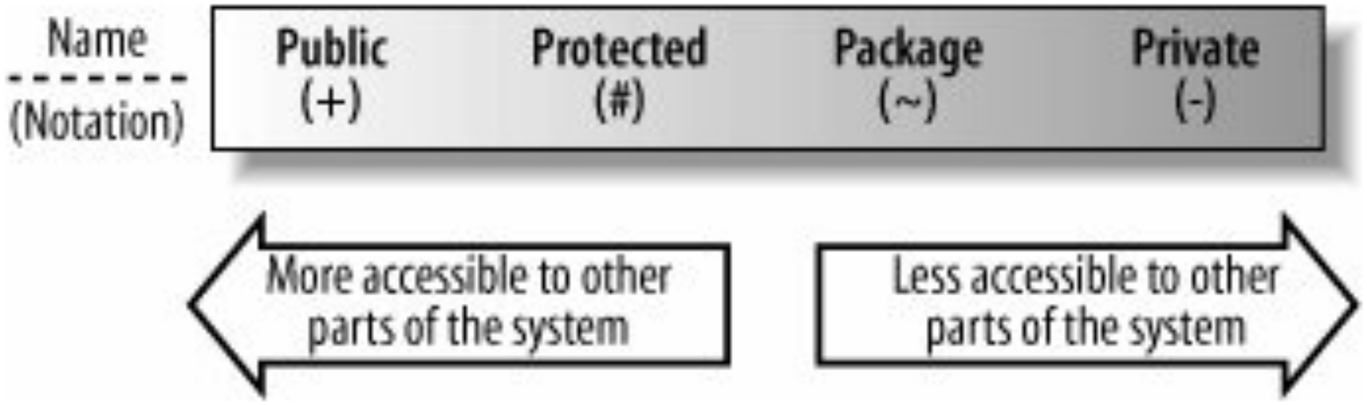
Figure 4-5. Two classes of objects have been identified in the CMS



4.3. Visibility 可见性

- Public
- Protected
- Package
- Private

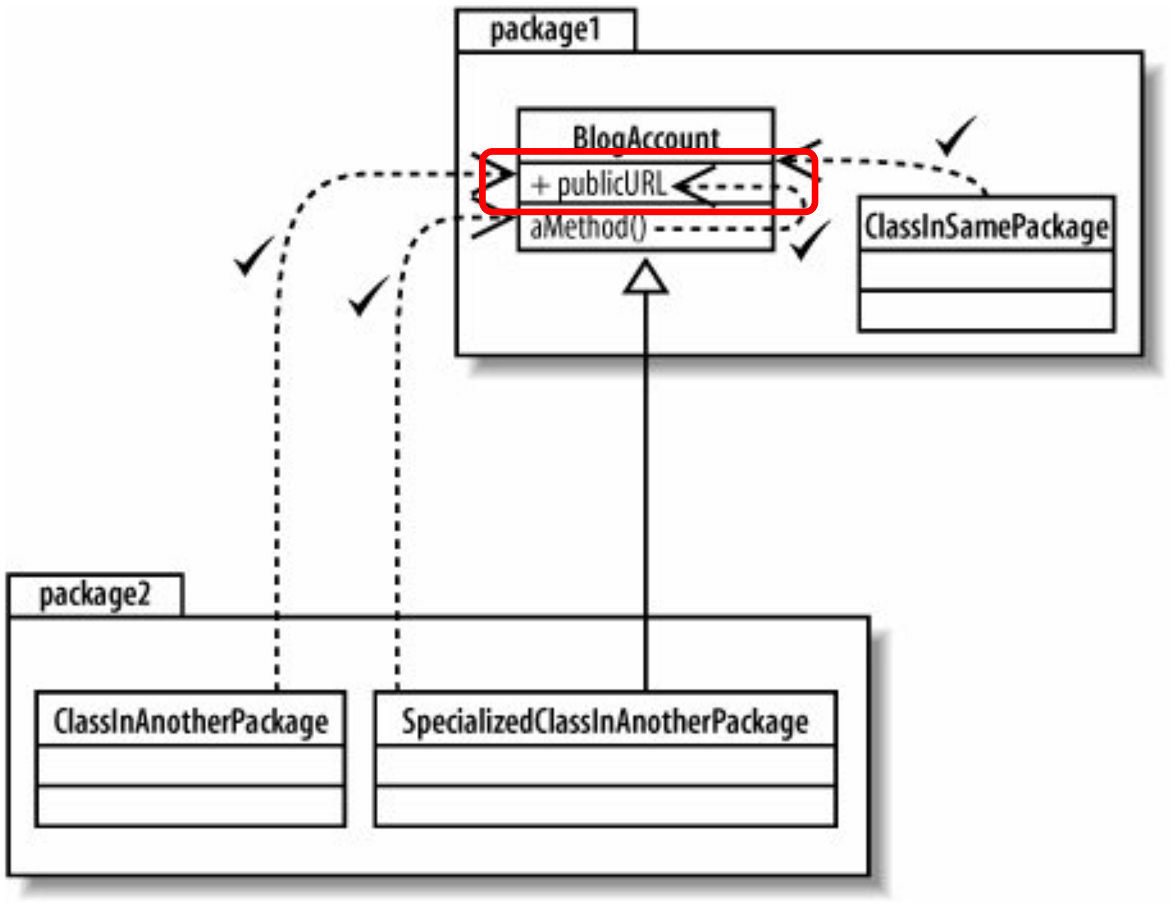
Figure 4-6. UML's four different visibility classifications 4种可见性的符号


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

4.3.1. Public Visibility 公开可见

- Starting with the most accessible of visibility characteristics, public visibility is specified using the plus (+) symbol before the associated attribute or operation.
  - Declare an attribute or operation public if you want it to be accessible directly by any other class.
- The collection of attributes and operations that are declared public on a class create that class's public interface. 接口
  - The public interface of a class consists of the attributes and operations that can be accessed and used by other classes. 使用
  - This means the public interface is the part of your class that other classes will depend on the most. 依赖
  - It is important that the public interface to your classes changes as little as possible to prevent unnecessary changes wherever your class is used. 不会变化

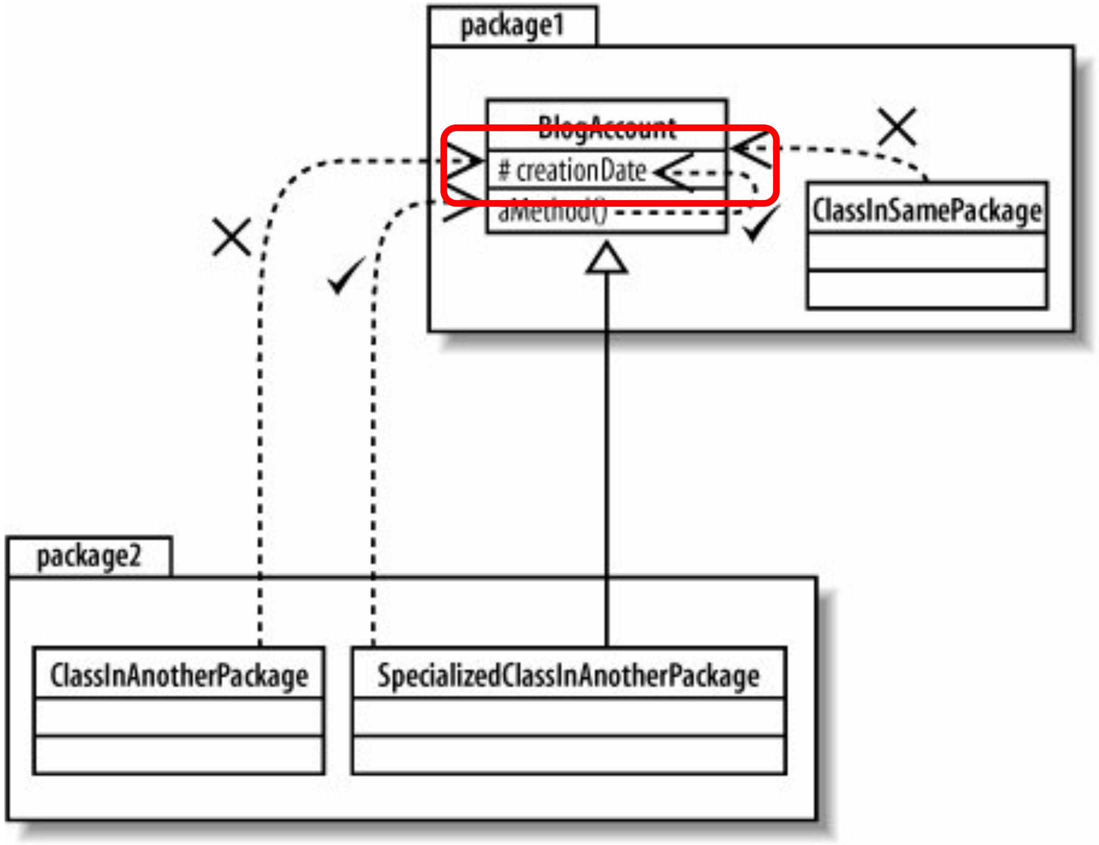
Figure 4-7. Using public visibility, any class within the model can access the public URL attribute



4.3.2. Protected Visibility 受保护

- Protected attributes and operations are specified using the hash (#) symbol and are more visible to the rest of your system than private attributes and operations, but are less visible than public.
- Declared 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. 继承
  - Protected elements cannot be accessed by a class that does not inherit from your class whether it's in the same package or not

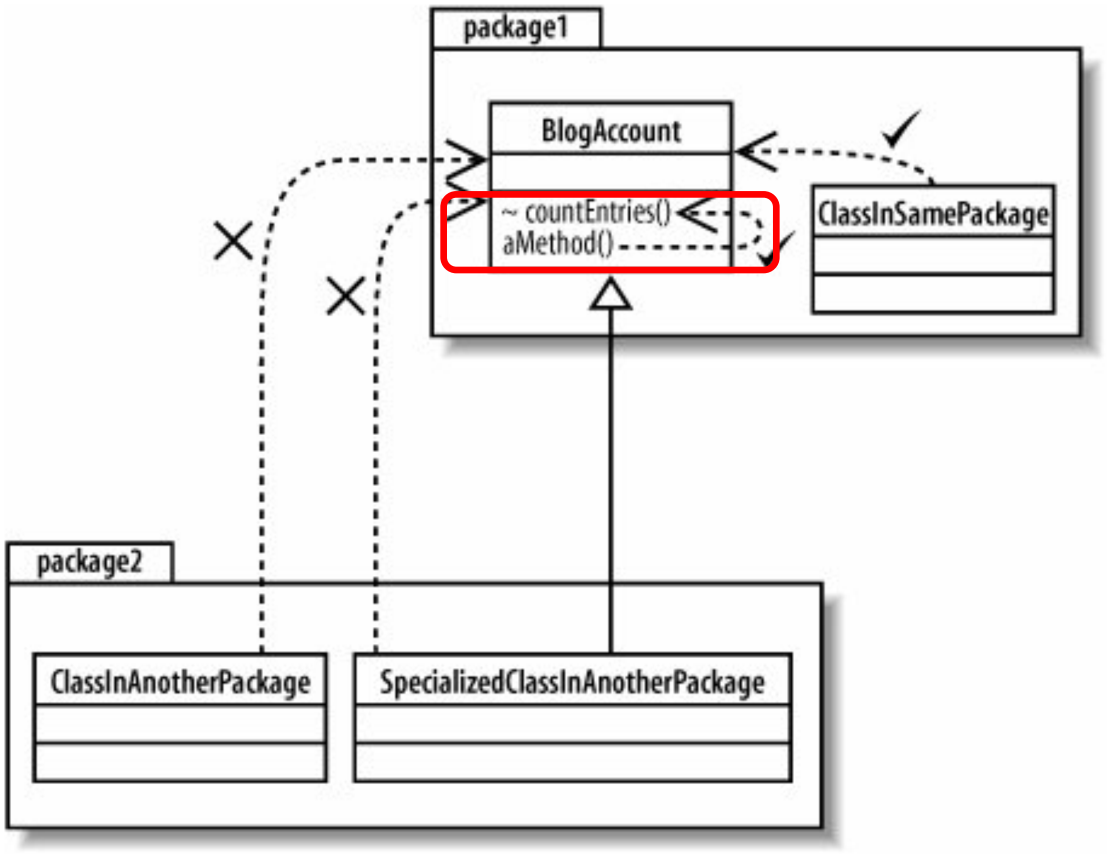
Figure 4-8. Any methods in the BlogAccount class or classes that inherit from the BlogAccount class can access the protected creationDate attribute



4.3.3. Package Visibility 包|友好|默认

- Package visibility, specified with a tilde (~), when applied to attributes and operations, sits in between protected and private.
  - As you'd expect, packages are the key factor in determining which classes can see an attribute or operation that is declared with package visibility .
- The rule is fairly simple: 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. Classes outside the package cannot access protected attributes or operations even if it's an inheriting class. 相同包可访问
  - In practice, package visibility is most useful when you want to declare a collection of methods and attributes across your classes that can only be used within your package.

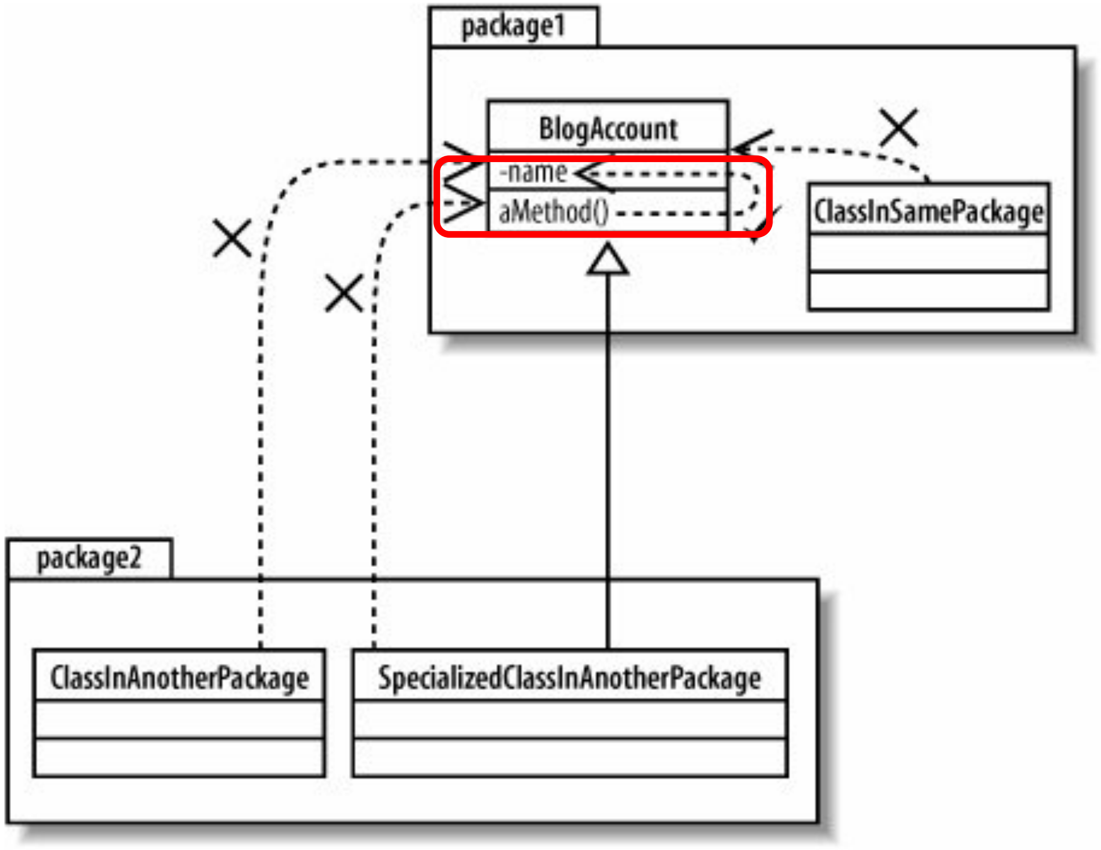
Figure 4-9. The countEntries operation can be called by any class in the same package as the BlogAccount class or by methods within the BlogAccount class itself



4.3.4. Private Visibility 私有可见

- Private visibility is the most tightly constrained type of visibility classification, and it is shown by adding a minus (-) symbol before the attribute or operation. 最严格的访问
  - 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.
- Private visibility is most useful if you have an attribute or operation that you want no other part of the system to depend on. 隐藏内部使用的细节
  - This might be the case if you intend to change an attribute or operation at a later time but don't want other classes with access to that element to be changed.

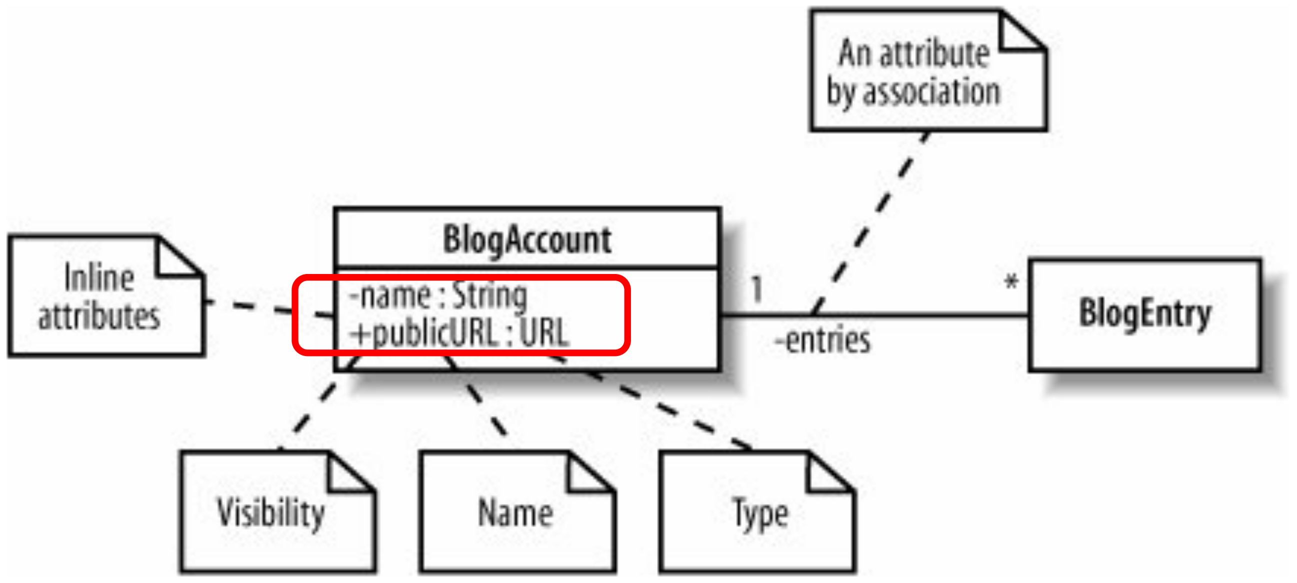
Figure 4-10. aMethod is part of the BlogAccount class, so it can access the private name attribute; no other class's methods can see the name attribute



4.4. Class State: Attributes 状态=属性 | 数据

- A class's attributes are the pieces of information that represent the state of an object.
- These 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. 内部属性， 关联属性

Figure 4-11. The BlogAccount class contains two inlined attributes, name and publicURL, as well as an attribute that is introduced by the association between the BlogAccount and BlogEntry classes



4.4.1. Name and Type 命名与类型

- An attribute's name can be any set of characters, but no two attributes in the same class can have the same name.
  - 属性不允许同名
- The type of attribute can vary depending on how the class will be implemented in your system but it is usually either a class,
  - 属性的类型
  - such as String, or a primitive type, such as an int in Java.

Example 4-1. Java inline and by-association attributes

```
public class BlogAccount
{
   // The two inline attributes from Figure 4-11.
   private String name;
   private URL publicURL;

   // The single attribute by association, given the name 'entries'
   BlogEntries[] entries;

   // ...

}
```

4.4.2. Multiplicity 多样性

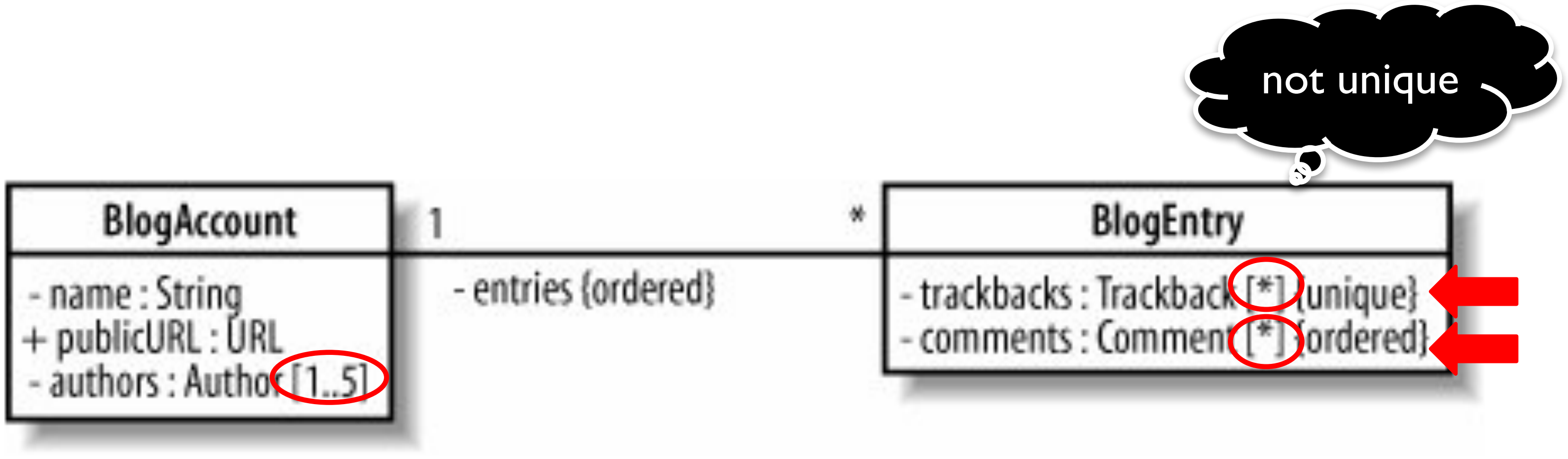

not unique
BlogAccount
- name : String
+ publicURL : URL
- authors : Author [1..5]
1
- entries {ordered}
*
BlogEntry
- trackbacks : Trackback [*] {unique}
- comments : Comment [*] {ordered}

Figure 4-12. Applying several flavors of attribute multiplicity to the attributes of the BlogAccount and BlogEntry classes

4.4.3. Attribute Properties 属性的属性

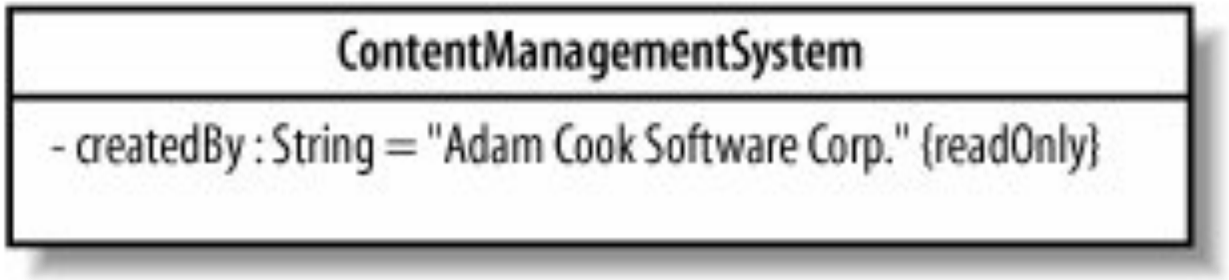

ContentManagementSystem
- createdBy : String = "Adam Cook Software Corp." {readOnly}

Figure 4-13. The createdBy attribute in the ContentManagementSystem class is given a default initial value and a property of readOnly so that the attribute cannot be changed throughout the lifetime of the system

```
public class ContentManagementSystem
{
   private final String createdBy = "Adam Cook Software Corp.";
}
```

Example 4-2. Final attributes in Java are often referred to as constants since they keep the same constant value that they are initially set up with for their entire lifetime

4.4.4. Inline Attributes Versus Attributes by Association 内部属性与关联属性

Inline Attributes

- The diagram is neater and easier to manage with more room for other information when the attributes are specified inline with the class box.

Association

- the diagram quickly becomes busy and that's just to show the associations, never mind all of the other relationships that classes can have.

Figure 4-14. The MyClass class has five attributes, and they are all shown using associations

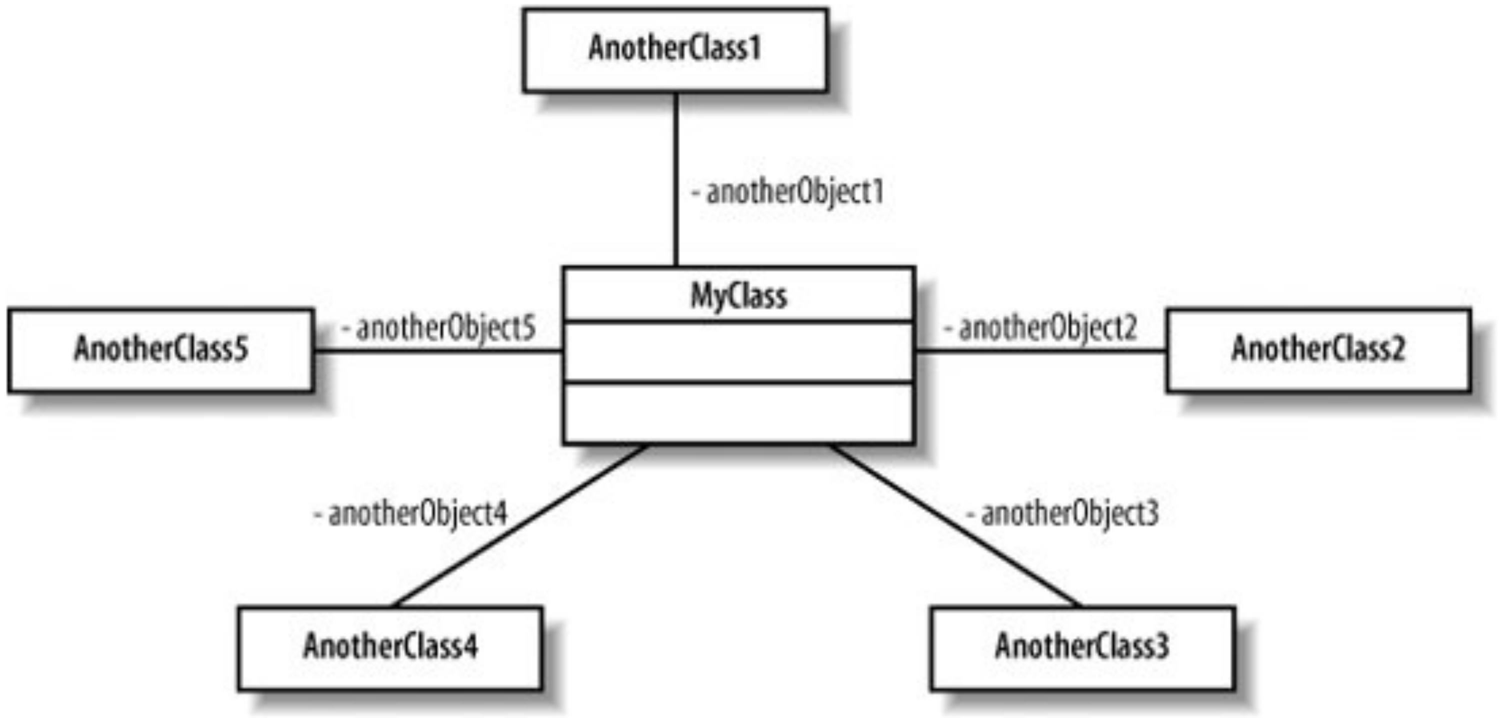
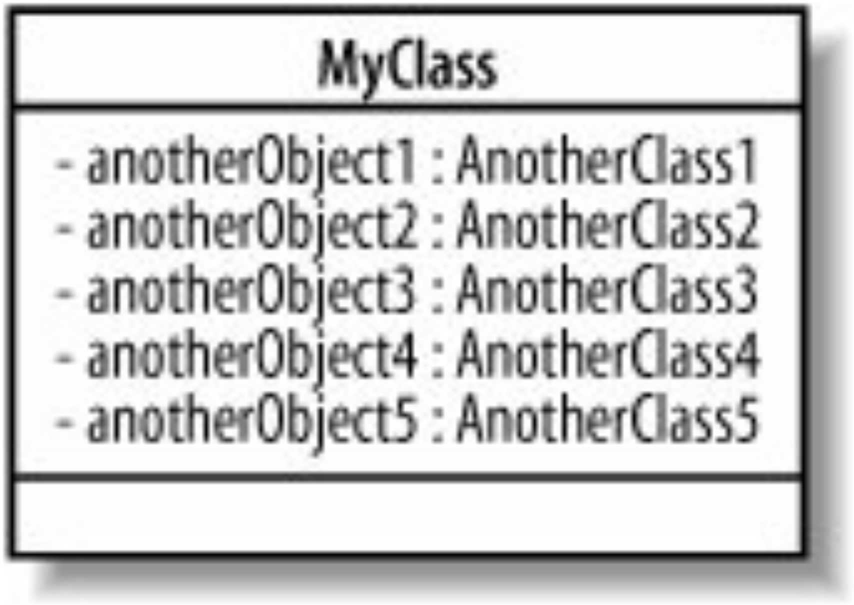


Figure 4-15. The MyClass class's five attributes shown inline within the class box


MyClass
- anotherObject1 : AnotherClass1
- anotherObject2 : AnotherClass2
- anotherObject3 : AnotherClass3
- anotherObject4 : AnotherClass4
- anotherObject5 : AnotherClass5

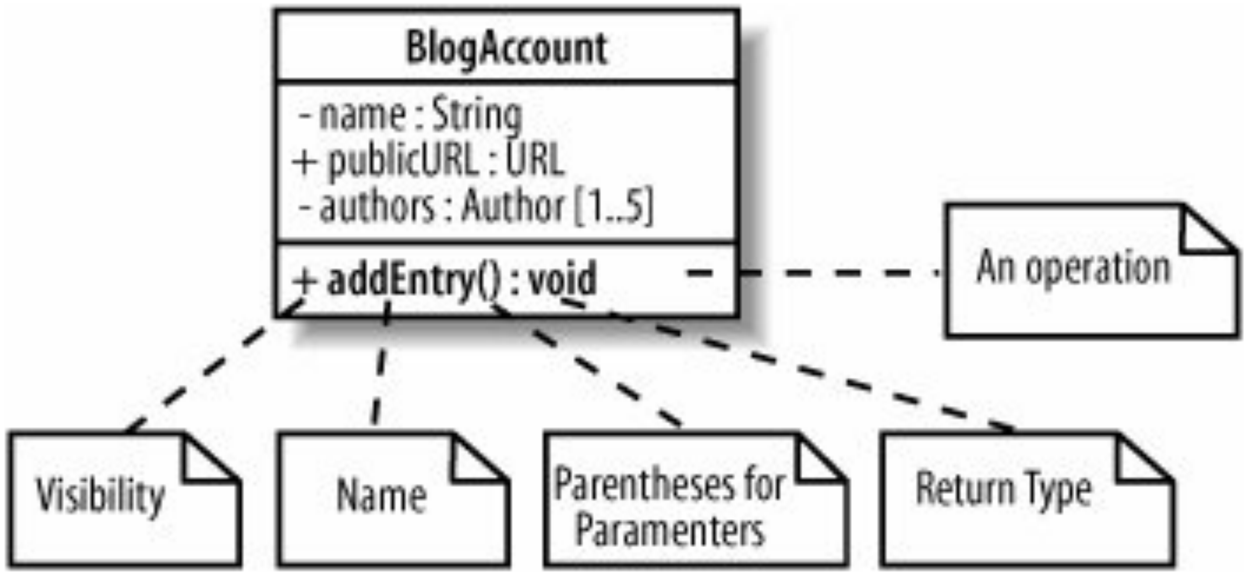
4.5. Class Behavior: Operations 行为=操作

- A class‘s operations describe what a class can do but not necessarily how it is going to do it.
  - 做什么，不说明怎么做
  - An operation is more like a promise or a minimal contract that declares that a class will contain some behavior that does what the operation says it will do.
    - 操作就像是承诺或契约
  - The collection of all the operations that a class contains should totally encompass all of the behavior that the class contains, including all the work that maintains the class's attributes and possibly some additional behavior that is closely associated with the class.
    - 操作代表了类的所有行为

4.5. Class Behavior: Operations 行为=操作

- 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
  - UML的操作要素：可见性+名称+(参数)+返回类型

Figure 4-16. Adding a new operation to a class allows other classes to add a BlogEntry to a BlogAccount



4.5.1. Parameters 参数

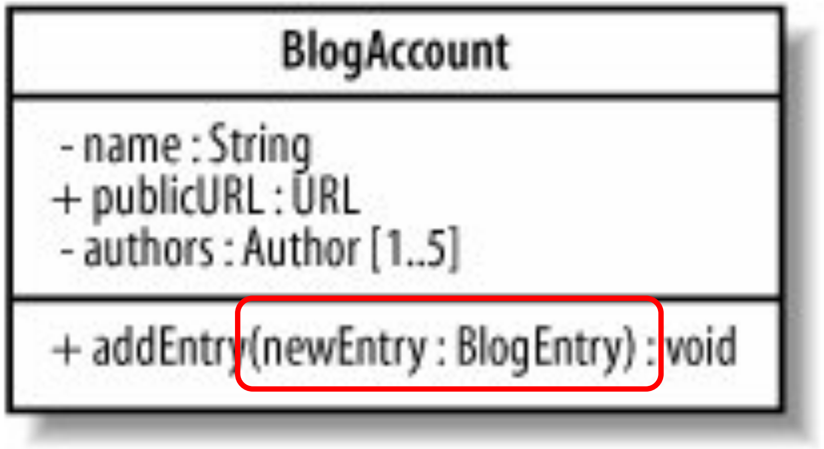

BlogAccount
- name : String
+ publicURL : URL
- authors : Author [1..5]
+ addEntry(newEntry : BlogEntry) : void

Figure 4-17. Adding a new parameter to the addEntry operation saves a bit of embarrassment when it comes to implementing this class; at least the addEntry operation will now know which entry to add to the blog!

4.5.1. Parameters

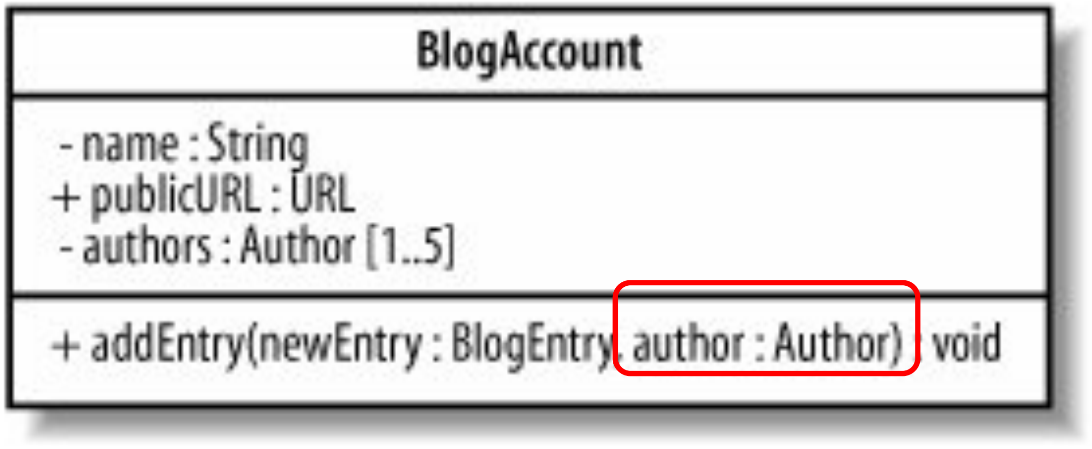
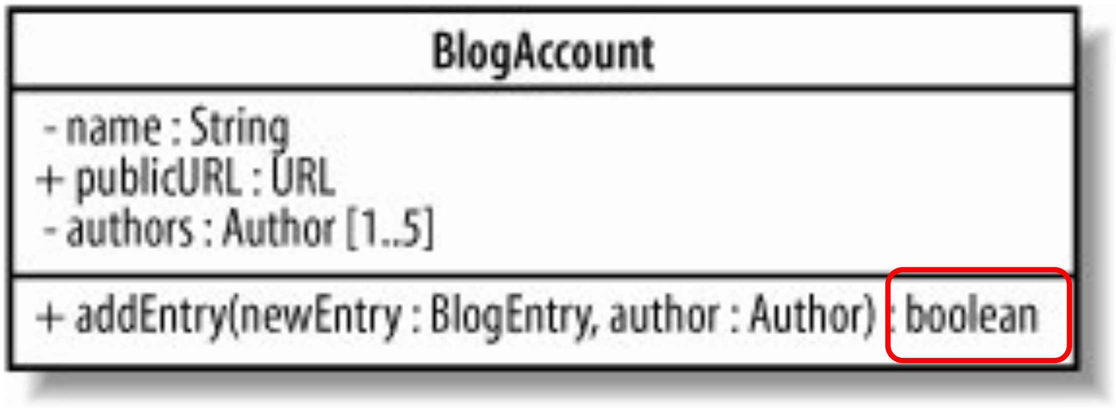

BlogAccount
- name : String
+ publicURL : URL
- authors : Author [1..5]
+ addEntry(newEntry : BlogEntry, author : Author) : void

Figure 4-18. As well as passing the new blog entry that is to be added, by adding another parameter, we can also indicate which author wrote the entry.

4.5.2. Return Types 返回类型



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

Figure 4-19. The addEntry(..) operation now returns a Boolean indicating whether the entry was successfully added

4.5.2. Return Types

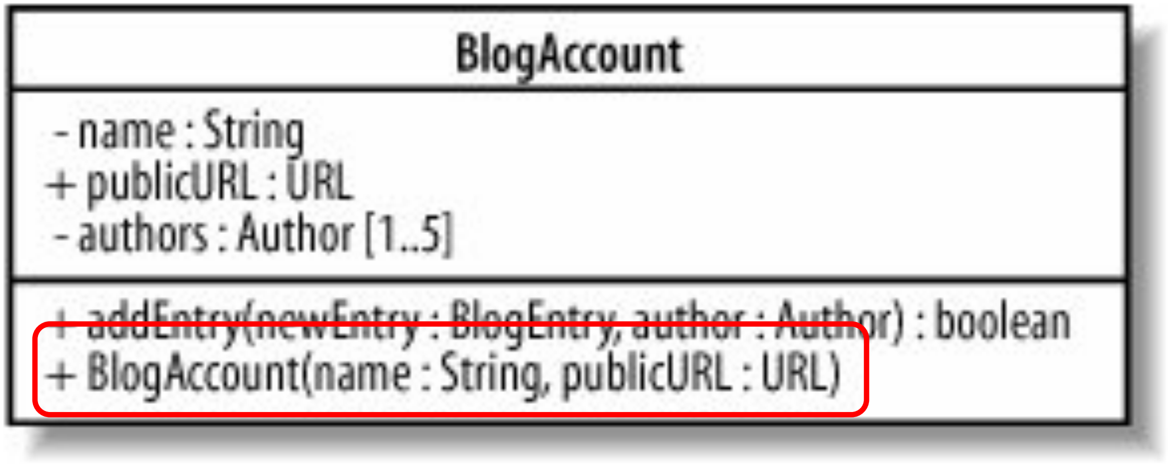

BlogAccount
- name : String
+ publicURL : URL
- authors : Author [1..5]
+ addEntry(newEntry : BlogEntry, author : Author) : boolean
+ BlogAccount(name : String, publicURL : URL)

Figure 4-20. The BlogAccount(..) constructor must always return an instance of BlogAccount, so there is no need to explicitly show a return type

4.6. Static Parts of Your Classes 静态成员

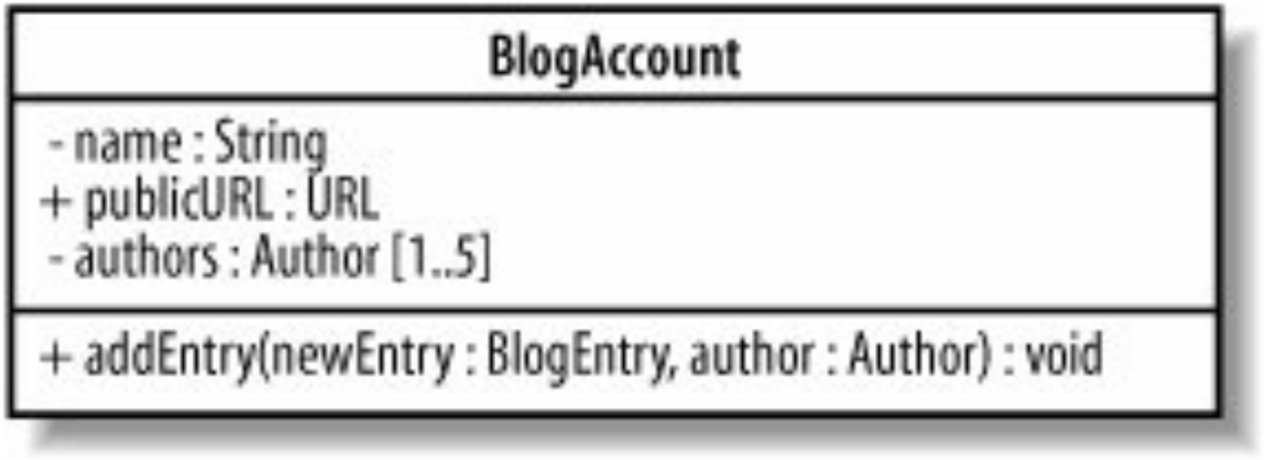

BlogAccount
- name : String
+ publicURL : URL
- authors : Author [1..5]
+ addEntry(newEntry : BlogEntry, author : Author) : void

Figure 4-21. The BlogAccount class is made up of three regular attributes and one regular operation

Figure 4-22. Both account1 and account2 contain and exhibit their own copy of all the regular non-static attributes and operations declared on the BlogAccount class

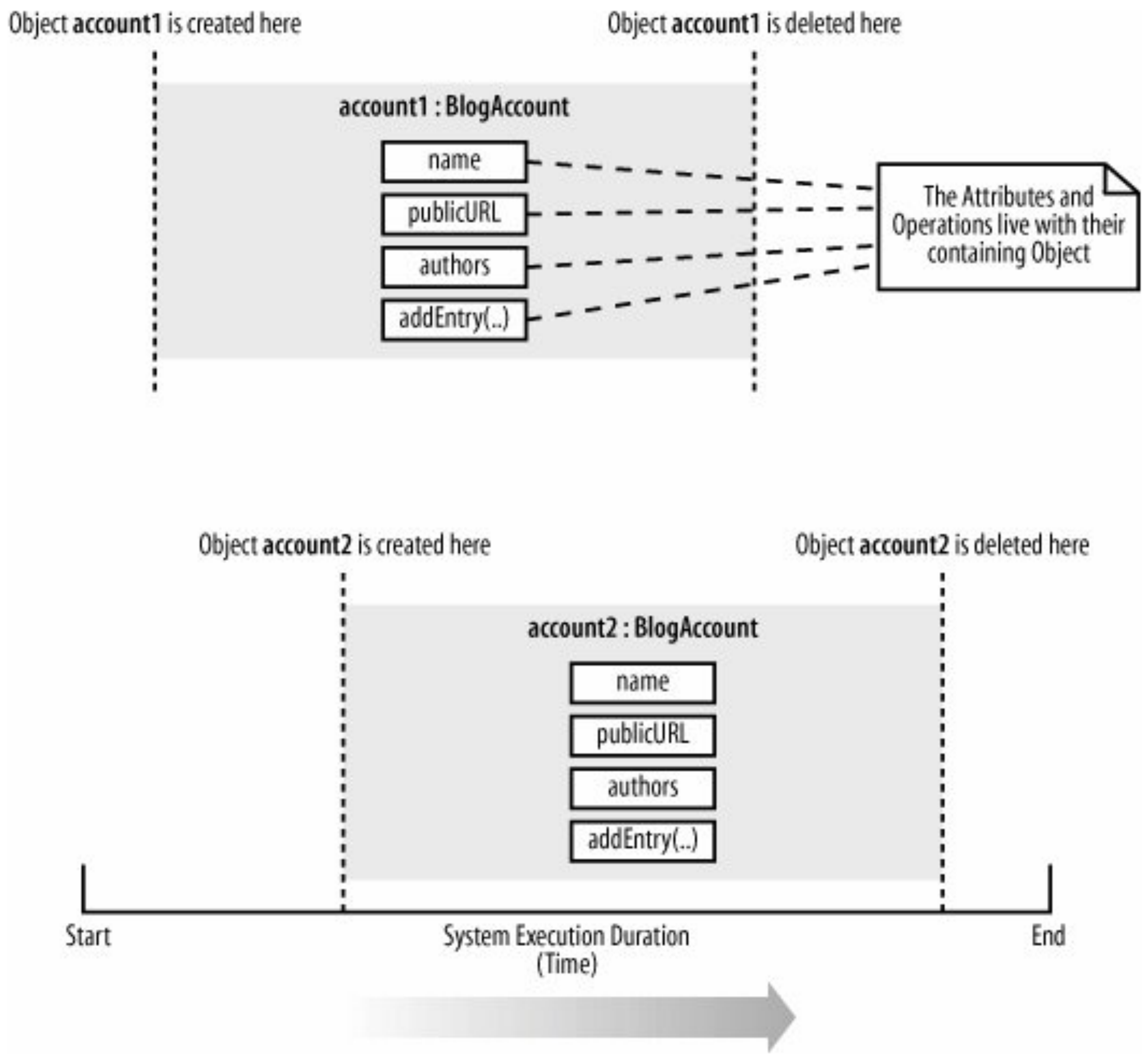

Object account1 is created here
Object account1 is deleted here
account1 : BlogAccount
name
publicURL
authors
addEntry(..)
The Attributes and Operations live with their containing Object
Object account2 is created here
Object account2 is deleted here
account2 : BlogAccount
name
publicURL
authors
addEntry(..)
Start
System Execution Duration (Time)
End

Figure 4-23. An attribute or operation is made static in UML by underlining it; the accountCounter attribute will be used to keep a running count of the number of objects created from the BlogAccount class

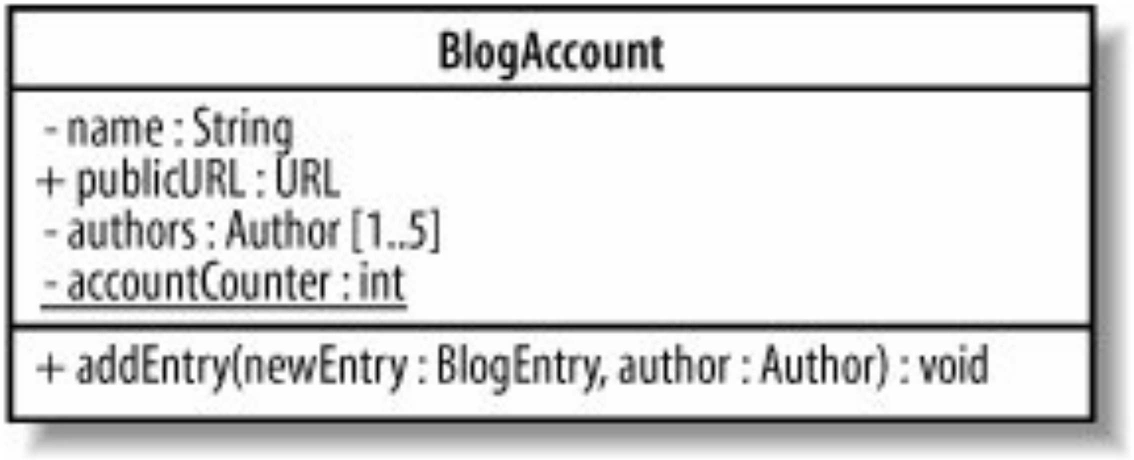
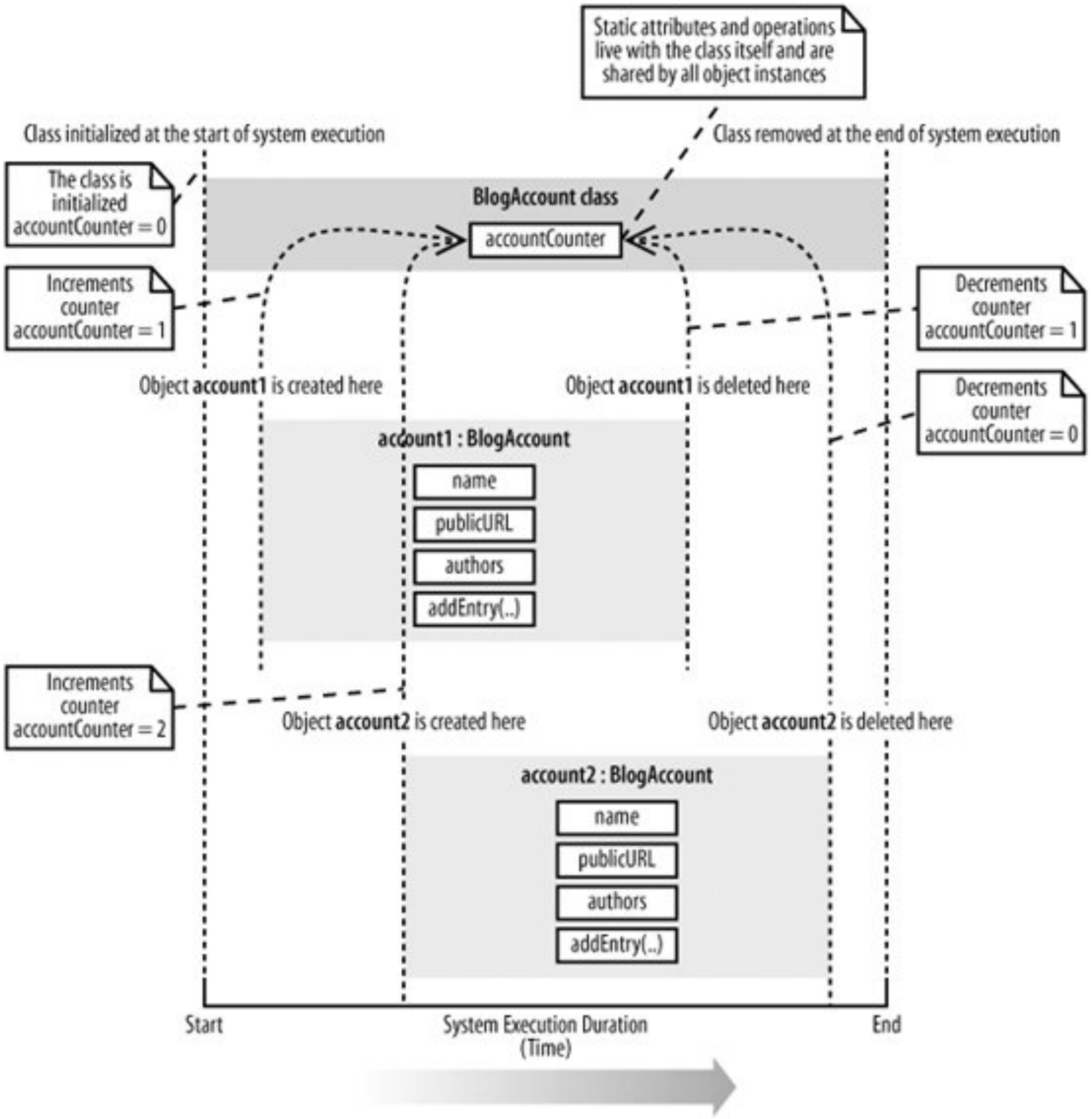

BlogAccount
- name : String
+ publicURL : URL
- authors : Author [1..5]
- accountCounter : int
+ addEntry(newEntry : BlogEntry, author : Author) : void

Figure 4-24. The static accountController attribute is sh to
kee
obj


Static attributes and operations live with the class itself and are shared by all object instances
Class initialized at the start of system execution
Class removed at the end of system execution
The class is initialized accountCounter = 0
BlogAccount class
accountCounter
Increments counter accountCounter = 1
Decrements counter accountCounter = 1
Object account1 is created here
Object account1 is deleted here
Decrements counter accountCounter = 0
account1 : BlogAccount
name
publicURL
authors
addEntry(..)
Increments counter accountCounter = 2
Object account2 is created here
Object account2 is deleted here
account2 : BlogAccount
name
publicURL
authors
addEntry(..)
Start
System Execution Duration (Time)
End

Summary

- 4. Modeling a System's Logical Structure: Introducing Classes and Class Diagrams
  - 4.1. What Is a Class?
  - 4.2. Getting Started with Classes in UML
  - 4.3. Visibility
  - 4.4. Class State: Attributes
  - 4.5. Class Behavior: Operations
  - 4.6. Static Parts of Your Classes

What's Next?

- Chapter 5. Modeling a System's Logical Structure: Advanced Class Diagrams
  - 5.1. Class Relationships 类的关系
  - 5.2. Constraints 约束
  - 5.3. Abstract Classes 抽象类
  - 5.4. Interfaces 接口
  - 5.5. Templates 模板

See you …