6. Bringing Your Classes to Life: Object Diagrams

Shaoning Zeng, http://zsn.cc

What we learned?

- Chapter 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
- Chapter 5. Modeling a System's Logical Structure: Advanced Class Diagrams
  - 5.1. Class Relationships
  - 5.2. Constraints
  - 5.3. Abstract Classes
  - 5.4. Interfaces
  - 5.5. Templates

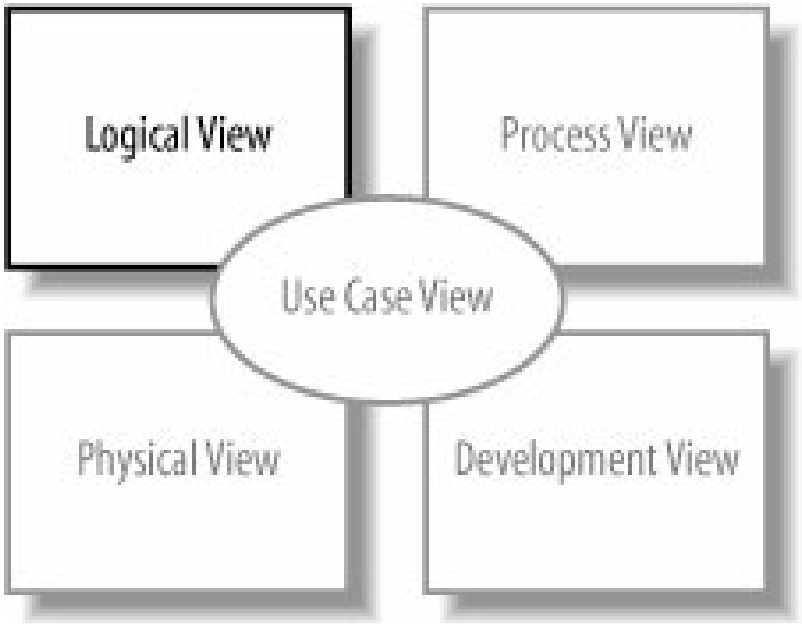
6. Bringing Your Classes to Life: Object Diagrams

- 6.1. Object Instances 对象实例
- 6.2. Links 链接
- 6.3. Binding Class Templates 绑定类模板

6. Bringing Your Classes to Life: Object Diagrams

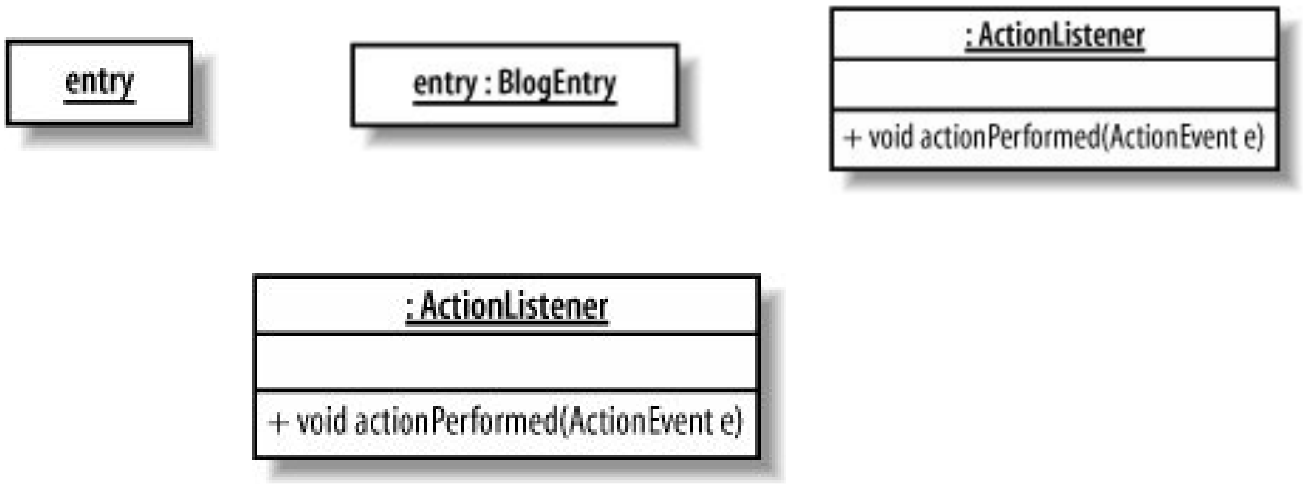
- Objects are at the heart of any object-oriented system at runtime.
- Compared to class diagrams, object diagram notation is very simple.
  - Despite having a fairly limited vocabulary, object diagrams are particularly useful when you want to describe how the objects within the system would work together in a particular scenario.
  - Class diagrams describe how all of the different types of objects within your system interact with each other.
  - They also draw attention to the many ways that the objects will exist and interact within your system at runtime.

Figure 6-1. The Logical View of your model contains the abstract descriptions of your system's parts, including which objects exist within your system at runtime



6.1. Object Instances

- Object notation
  - an object is shown with a rectangle,
  - the title is underlined
  - add the class name to the object name
  - the anonymous object



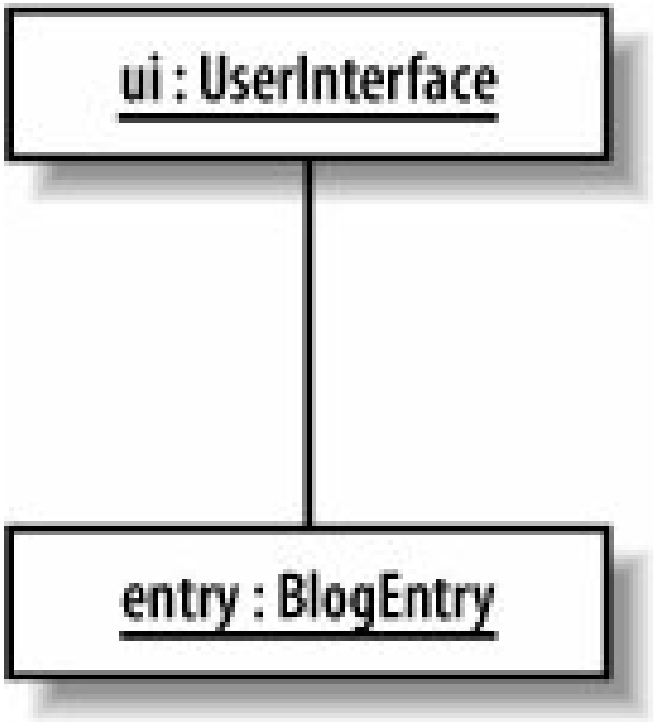
Example 8-1. Using an anonymous object in Java code to register an ActionListener

```
with a JButton
public void initialiseUI(  ) {
   //... Other method implementation code ...

   JButton button = new Jbutton("Submit");
   button.addActionListener(new ActionListener{
      public void actionPerformed(ActionEvent e)
      {
         System.out.println("The button was pressed so it's time to do something ...");
      }
   });

   //... Other method implementation code ...
}
```

6.2. Links

- Links between objects on an object diagram show that the two objects can communicate with each other. However, you can't just link any two objects together.
  - If you create a link between two objects, there must be a corresponding association between the classes.
- The links on an object diagram work in a similar fashion to links on a communication diagram (see Chapter 8).
  - However, unlike communication diagrams, the only additional piece of information that you can optionally add to a link is a label that indicates the purpose of the link, as shown in Figure 6-6.

Figure 6-5. Links are shown using a line between the two objects that are being linked



6.2.1. Links and Constraints

- Links between objects correspond to associations between the object's classes.

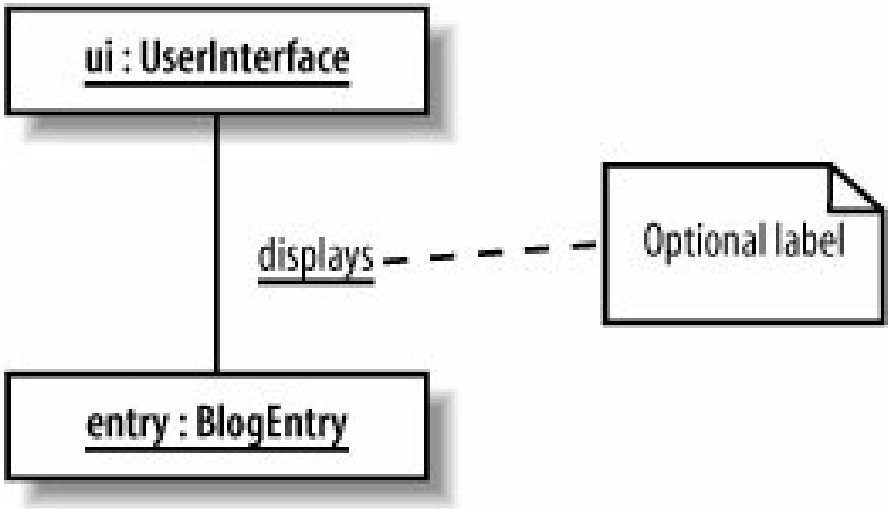


Figure 6-6. To play some tunes, a BlogEntry object is connected to a UserInterface object

Figure 6-7. When a BlogEntry is added to a BlogAccount, it will be grouped under one or more categories; the Category class is associated with the relationship between a BlogEntry and a BlogAccount

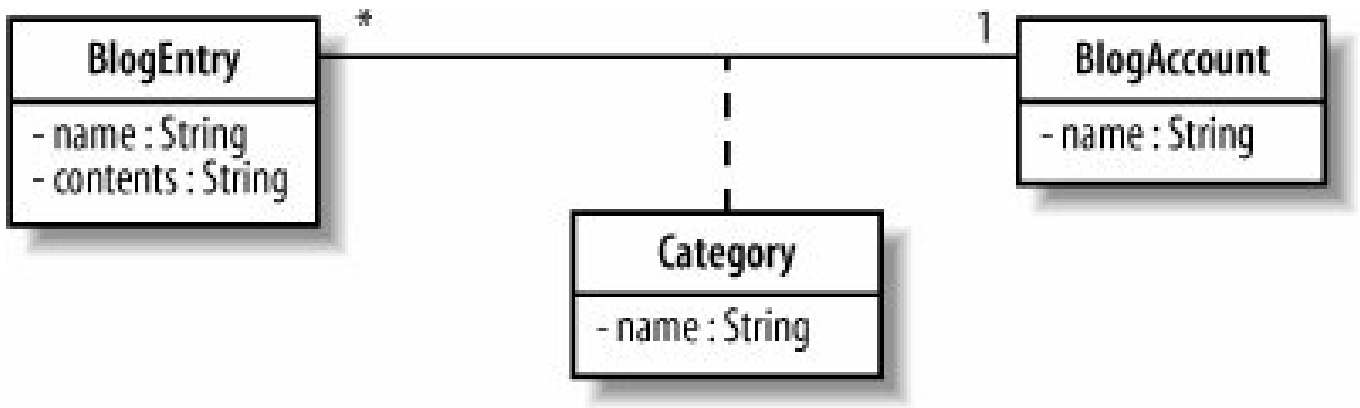


Figure 6-8. A BlogEntry can be associated with a BlogAccount and a set of categories, but there is no rule that states that a BlogEntry must be associated with a category

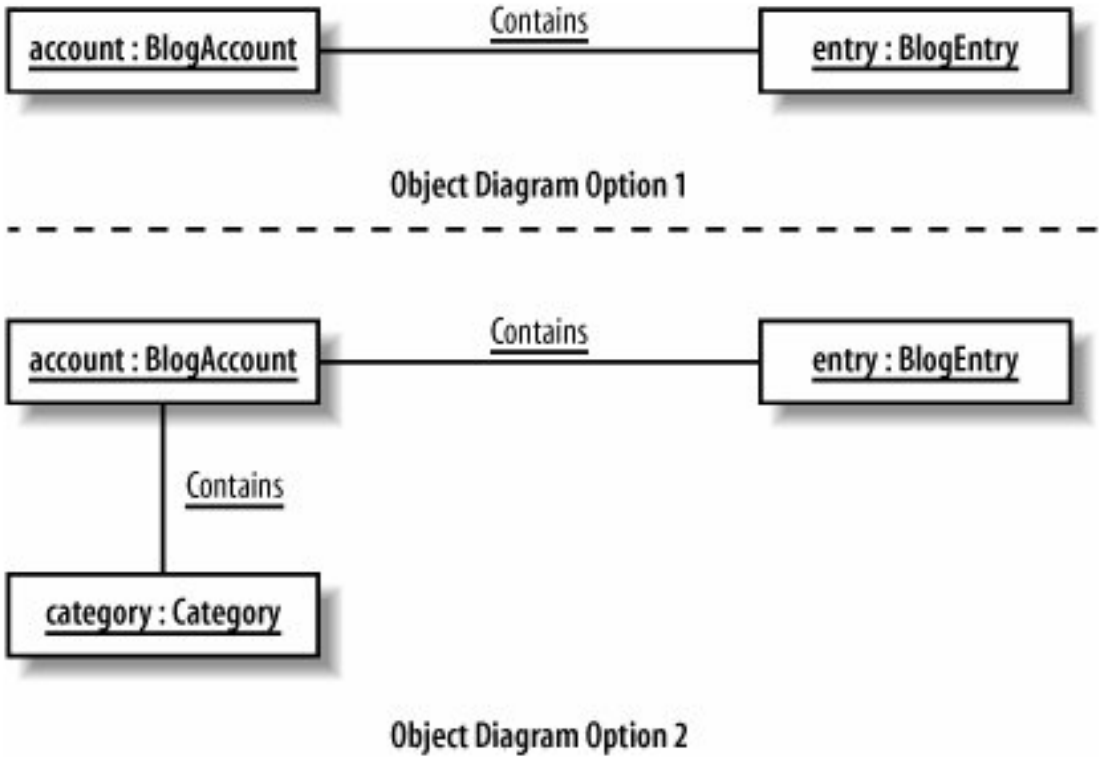


Figure 6-9. The constraint states that within the context of the BlogEntry, a BlogEntry object should be associated with a category and that the association should not be null; in other words, for a BlogEntry to be added to a BlogAccount, it must have a category assigned

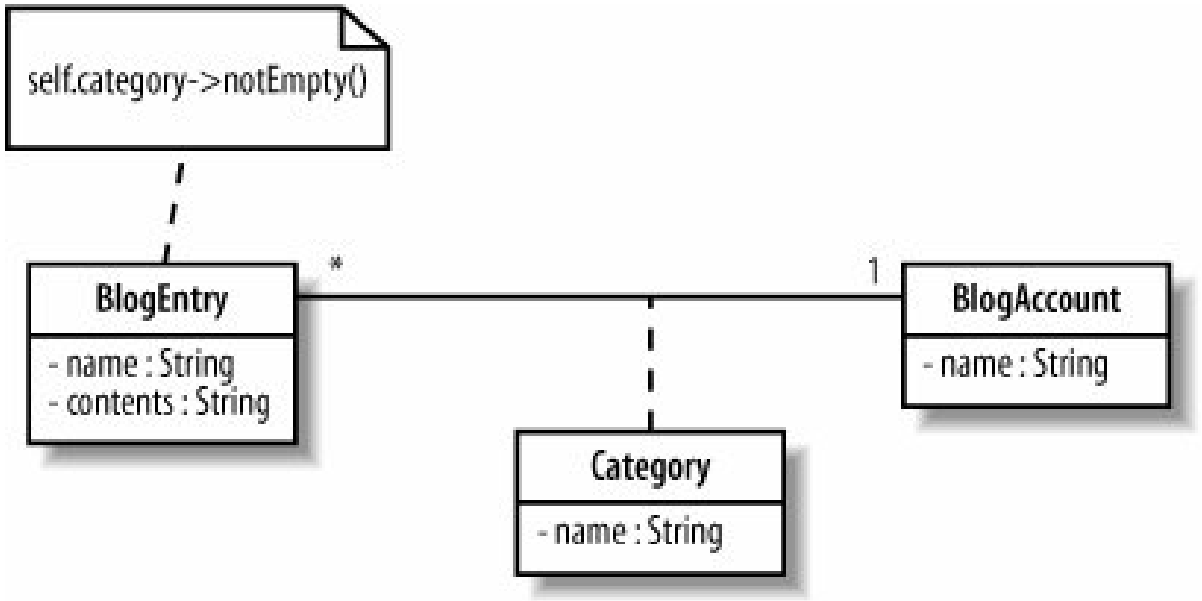
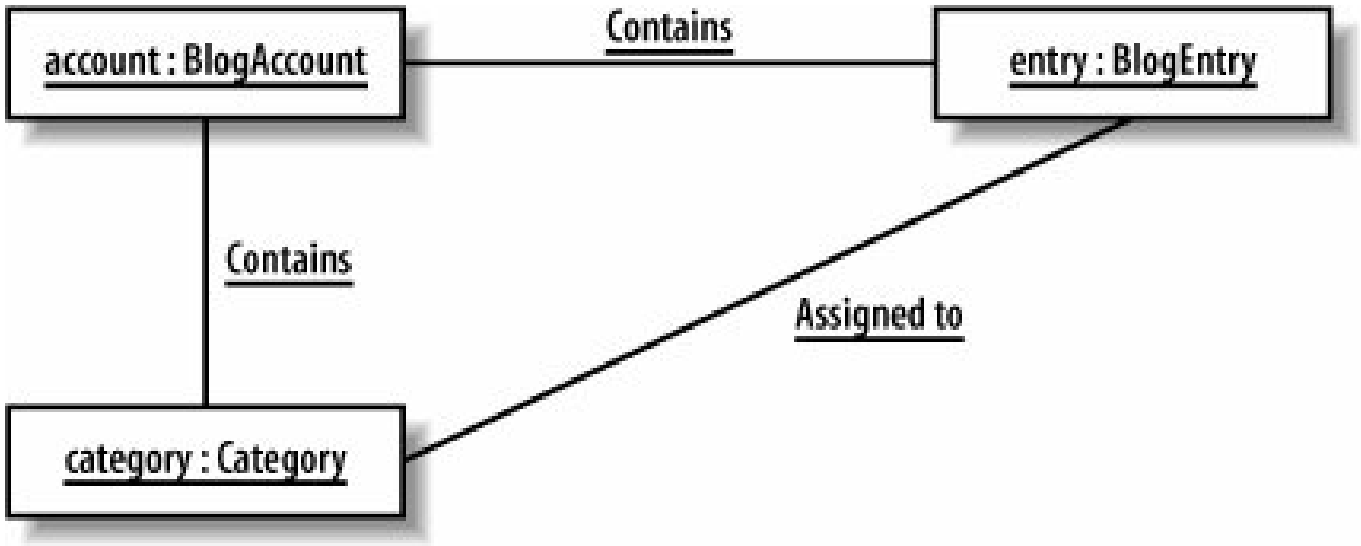


Figure 6-10. The self.category->notEmpty( ) constraint affects the options available when you create an object diagram, ensuring that an entry is associated with a category for it to be added to a BlogAccount



6.3. Binding Class Templates

- In Chapter 5, we saw how the parameters declared on class templates could be realized using subclassing on a class diagram.
  - Although this approach works fine, the real power of templates comes when you bind template parameters at runtime.
  - To do this, you take a template and tell it the types that its parameters are going to be as it is constructed into an object.
- Object diagrams are ideal for modeling how runtime binding takes place. 运行时绑定
  - When you use runtime template parameter binding, you are really talking about objects rather than classes,
  - so you can't really model this information on a regular class diagram.

Figure 6-11. The ListOfThings collection can store and remove any class of object to which the E parameter is bound

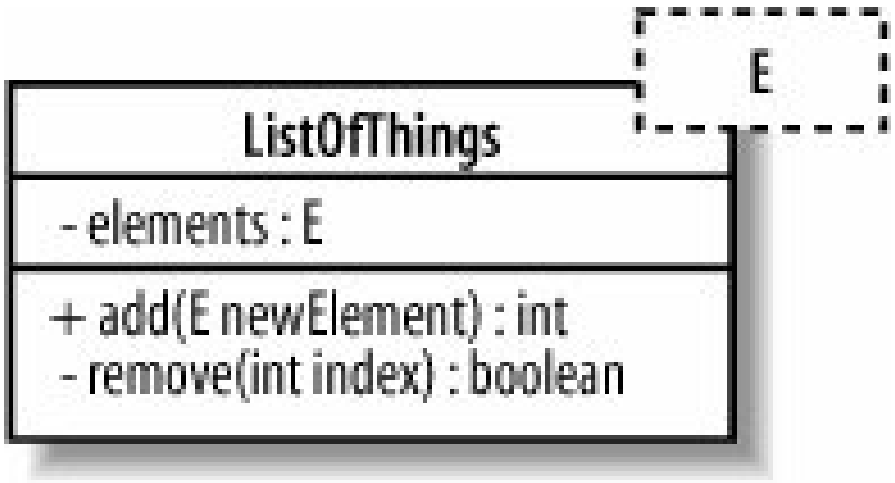


Figure 6-12. The listOfBlogEntries reuses the generic ListOfThings template, binding the E parameter to the BlogEntry class to store only objects of the BlogEntry class

Example 8-2. Using Java 5 generics to implement the ListOfThings template and a runtime binding to the ListOfBlogEntries

```
public class ListOfThings<E> {

   // We're cheating a bit here; Java actually already has a List template
   // and so we're using that for our ListOfThings templates operations.
   private List[E] elements;

   public ListOfThings {
      elements = new ArrayList<E>(  );
   }

   public int add(E object) {
      return elements.add(object);
   }

   public E remove(int index) {
      return elements.remove(index);
   }
}

public class Application {

   public static void main(String[] args) {
      // Binding the E parameter on the ListOfThings template to a Musician class
      // to create a ListOfThings that will only store Musician objects.
      ListOfThings <BlogEntry>listOfBlogEntries= new ListOfThings<BlogEntry>(  );
   }
}
```

See you …