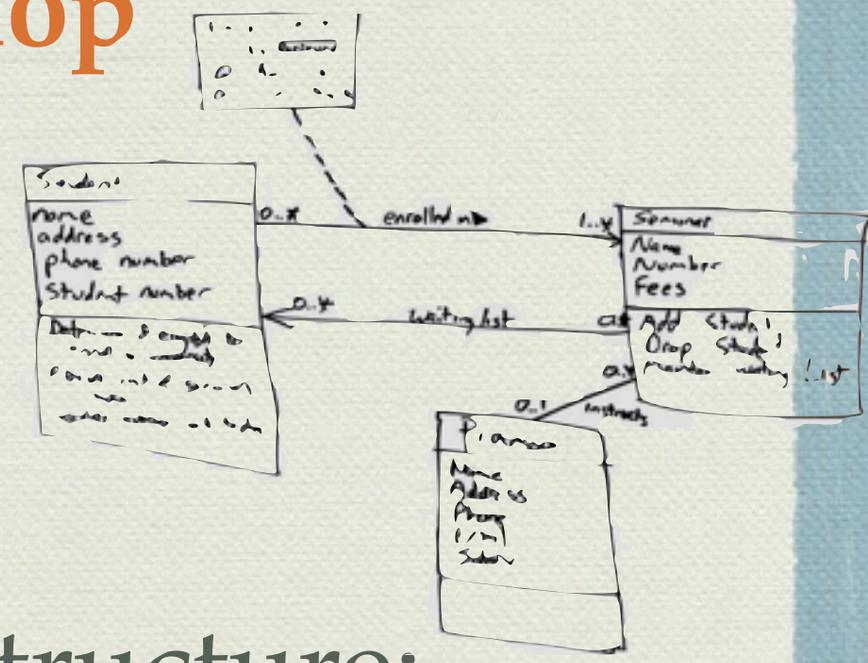
Software Engineering Workshop

Workshop 5



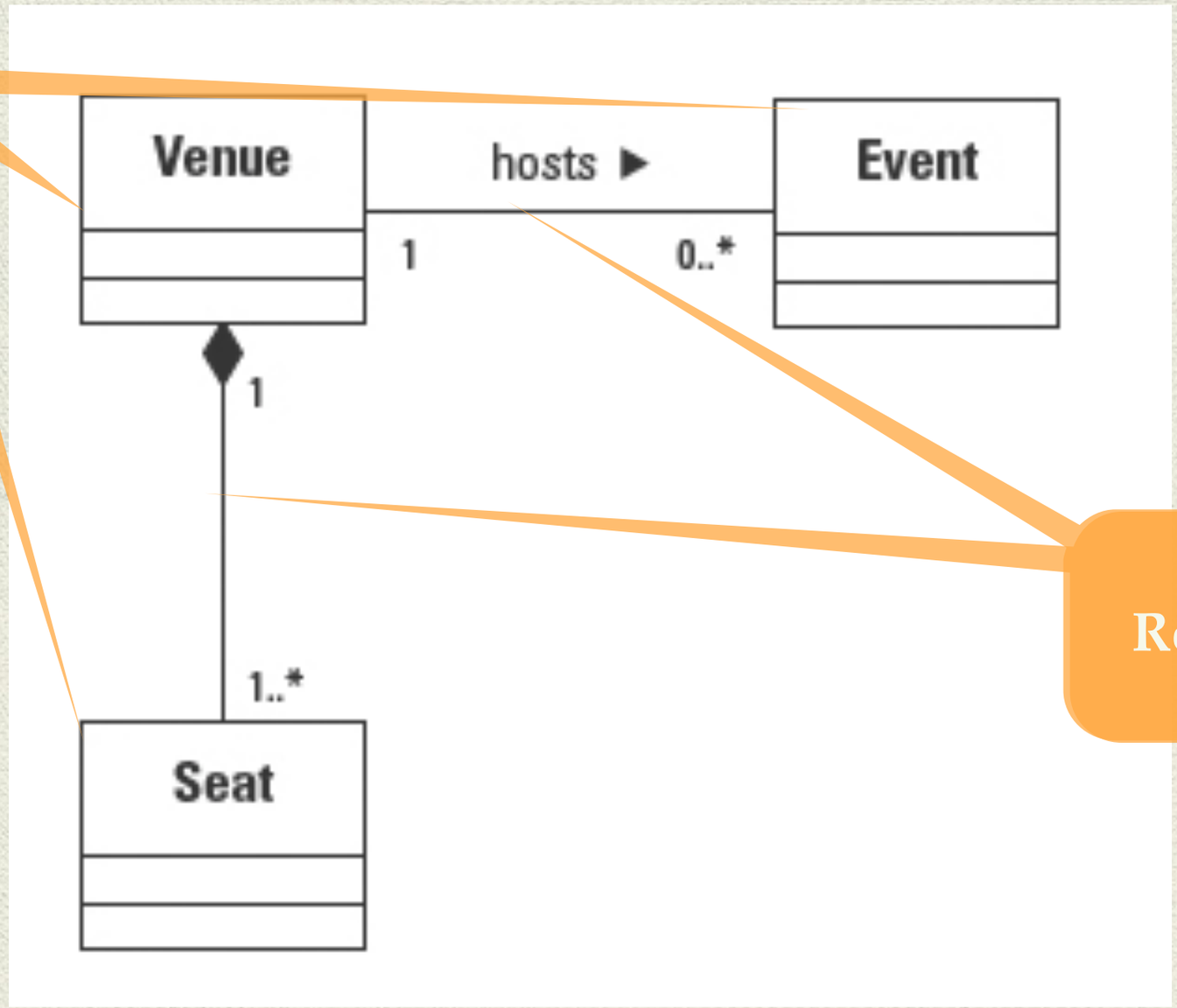
Modeling a System's Logical Structure: Class Diagrams (Part 2)

Slides prepared by Marwah Alaofi

Announcement

- Class diagram is due in two weeks.
- The documentation will be due with sequence diagrams in week 14, make a start from now!

Classes



Relationships

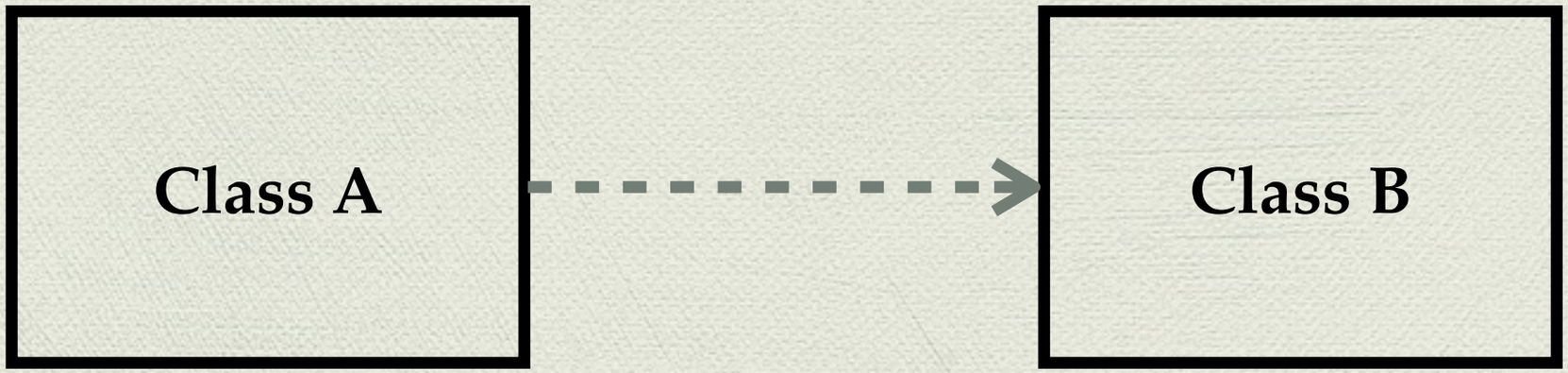
In today's workshop you'll learn about ..

- Dependency relationships
- Association relationships
  - Aggregation relationships
  - Composition relationships
- Generalization Relationship

Relationships

- Classes do not live separately—they work together using different types of relationships.
- A class can be a type of another class—generalization—or it can contain objects of another class in various ways depending on how strong the relationship is between the two classes.

Dependency Relationship



- A *dependency* between two classes declares that a class needs to know about another class to use objects of that class.
- You show a dependency between classes using a dashed line with an arrow pointing from the dependent class (client) to the class that is used (supplier).
- Dependencies are typically read as "...uses a..."

Dependency Interpretation in Java

```
import B;
public class A
{
public void method1(B b)
{ // . . . }
public void method2()
{ B tempB = new B(); // . . . }
}
```

Either one of class B's uses, as a parameter to a method, or as a local instance reference inside a method, would be appropriate reflection of a UML dependency relationship.

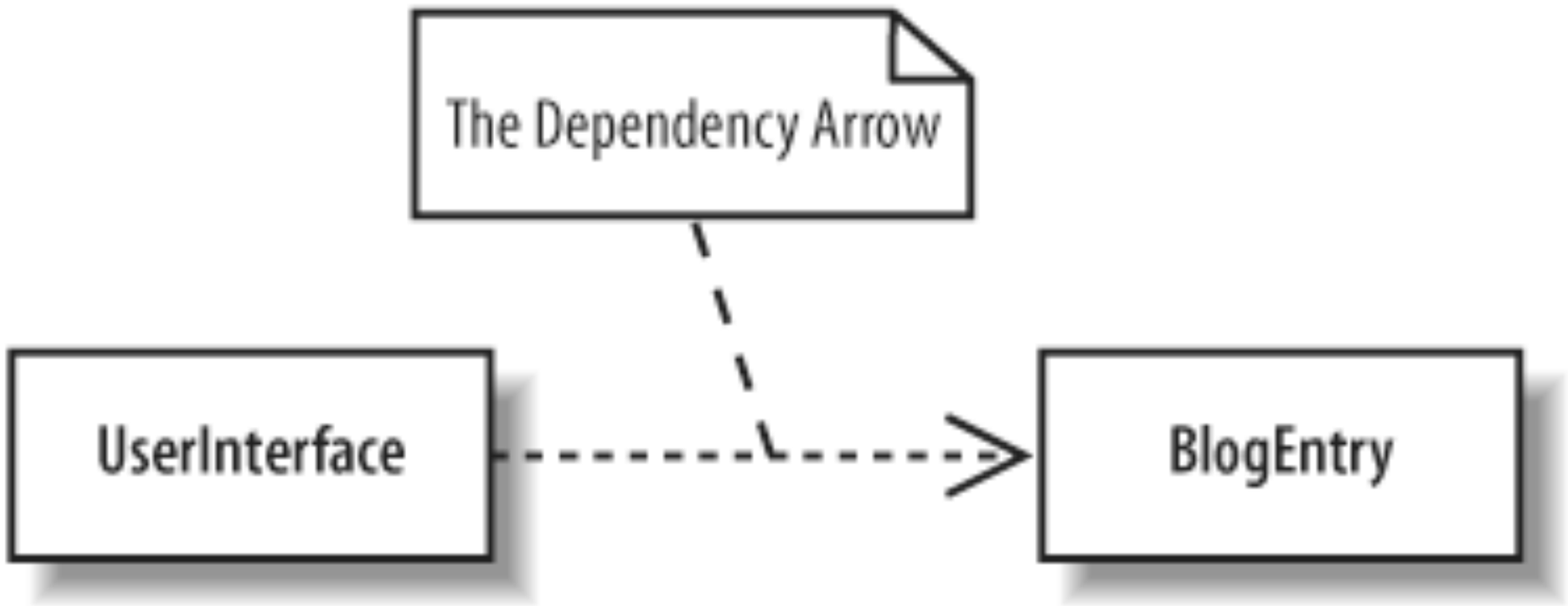
Dependency is the Weakest!

```
import B;
public class A
{
public void method1(B b)
{ // . . . }
public void method2()
{ B tempB = new B(); // . . . }
}
```

- The weakest relationship between classes is a dependency relationship.

- A dependent class briefly interacts with the target class but typically doesn't retain a relationship with it for any real length of time.

Dependency–Example



The UserInterface is dependent on the BlogEntry class because it will need to read the contents of a blog's entries to display them to the user.

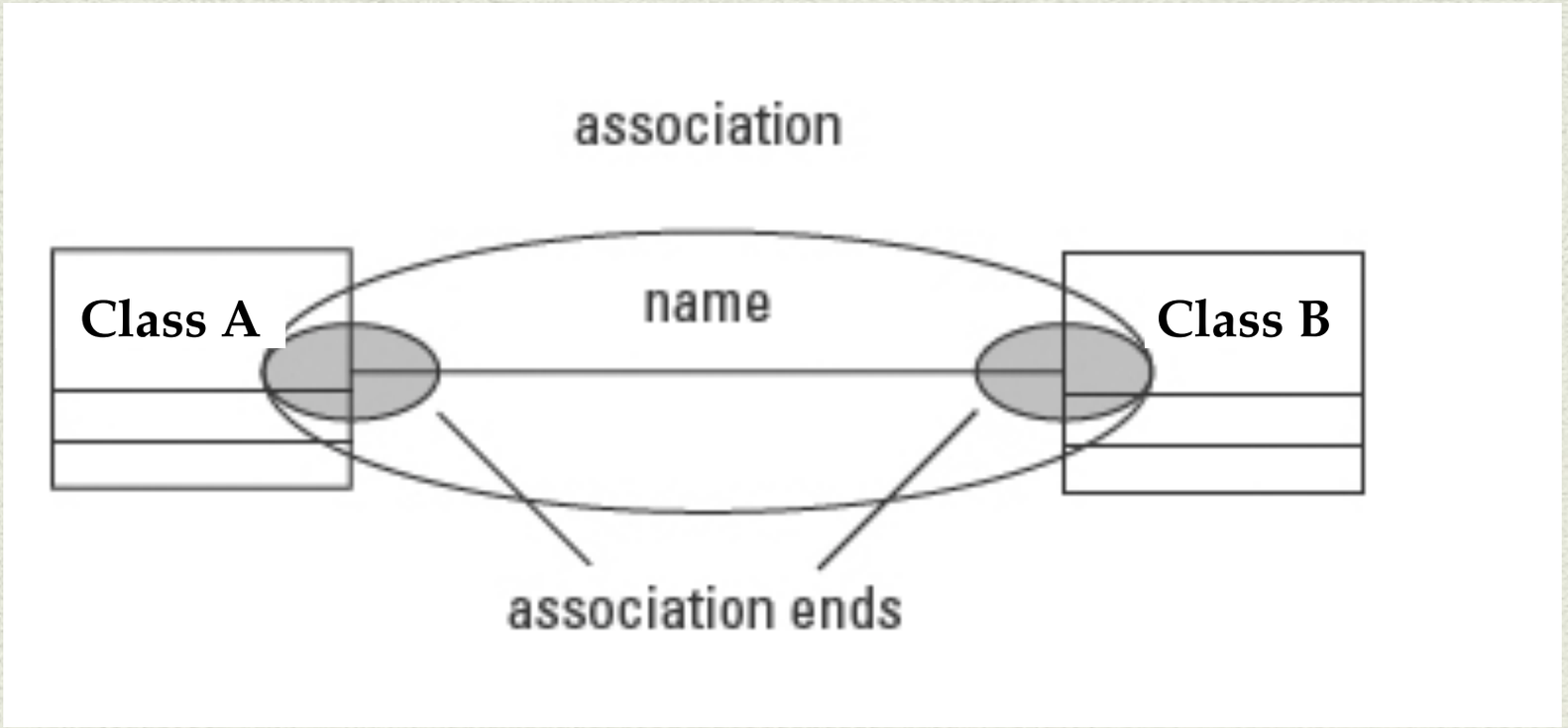
Common Use of Dependency Relationships

- The dependency relationship is often used when you have a class that is providing a set of general-purpose utility functions, such as in Java's regular expression (`java.util.regex`) and mathematics (`java.math`) packages.
- Classes depend on the `java.util.regex` and `java.math` classes to use the utilities that those classes offer.

Association Relationships

- *Association* means that a class will actually contain a reference to an object, or objects, of the other class in the form of an attribute.
- Association is shown using a simple line connecting two classes.
- Associations are typically read as "...has a…".

Association Relationships



- A complete association definition is built with three parts: an association line between the classes and two association ends.

- The association line and its name define the identity and the purpose of the relationship. The association ends each define the rules about how the objects of the classes at each end may participate.

Association–Naming an Association

- The usual way to name an association is with a verb or verb phrase.
- The name only needs to be somewhere in the middle of the line between the two classes with a directional indicator to show the reader how to interpret the meaning of the association name.

Association–Navigability Arrow

- *Navigability* is often applied to an association relationship to describe which class contains the attribute that supports the relationship.
- You can explicitly forbid navigation from one class to another by placing a small `x` on the association line at the end of the class you can't navigate to

Navigation Examples in Java (Cont.)


```
import B;
public class A {
   private B b1;
   public B getB() {
      return b1;
   }
}
```

```
import A;
public class B {
 private A a1;
   public A getA() {
      return a1;
 }
}
```

Navigation Examples in Java (Cont.)


```
import B;
public class A {
   private B b1;
   public B getB() {
      return b1;
   }
}
```

```
public class B {
attributes
operations
}
```

Navigation Examples in Java (Cont.)


```
public class A {
Attributes
Operations
}
```

```
import A;
public class B {
 private A a1;
  public A getA() {
     return a1;
  }
}
```

Association–Multiplicity and Attribute Name

- Place the attribute name at the end of the association line and next to the class that it describes.

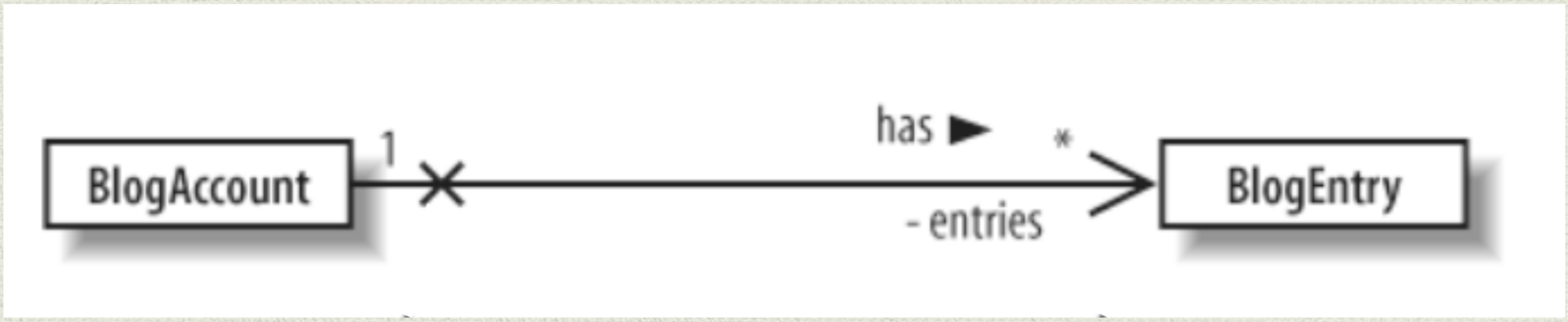
- You can express how many instances of a particular class are involved in a relationship. If you don't specify a value, a multiplicity of 1 is assumed.

Association Example

```
public class BlogAccount {

    :
    private BlogEntry[] entries;
    :
}
```

```
public class BlogEntry
{
//Attributes and Methods
declared here ...
}
```



Another Example

- A tweet has unknown number of comments.

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.

Aggregation Relationships

- A special case of association that models **a whole-part relationship** between an aggregate (the whole) and its parts.

- It is used to indicate that a class actually ***owns but may share*** **objects of another class. Aggregations are usually read as "...owns a..."**

- Aggregation is shown by using an **empty diamond arrowhead next to the owning class.**

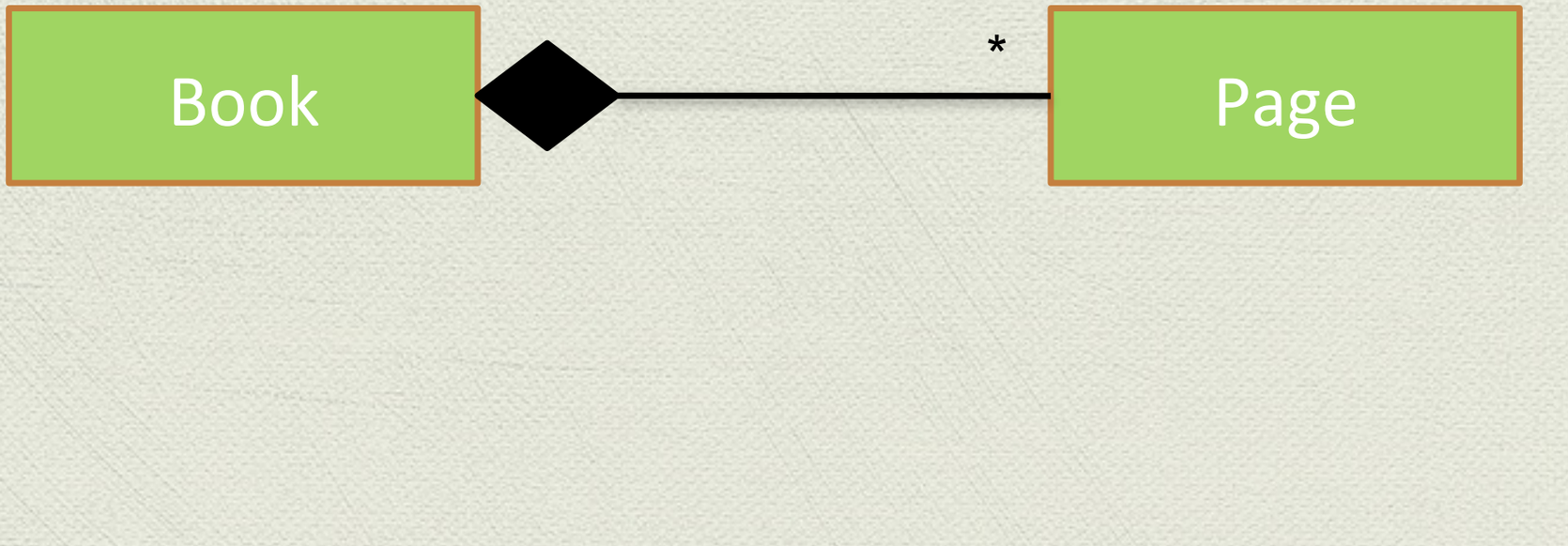
Aggregation Relationships

"**aggregation** has no semantics beyond that of a regular association. It is, as Jim Rumbaugh puts it, a modeling placebo. People can, and do, use it - but there are no standard meanings for it. So if you see it, you should inquire as to what the author means by it. I would advise not using it yourself without some form of explanation" -Martine Fowler.

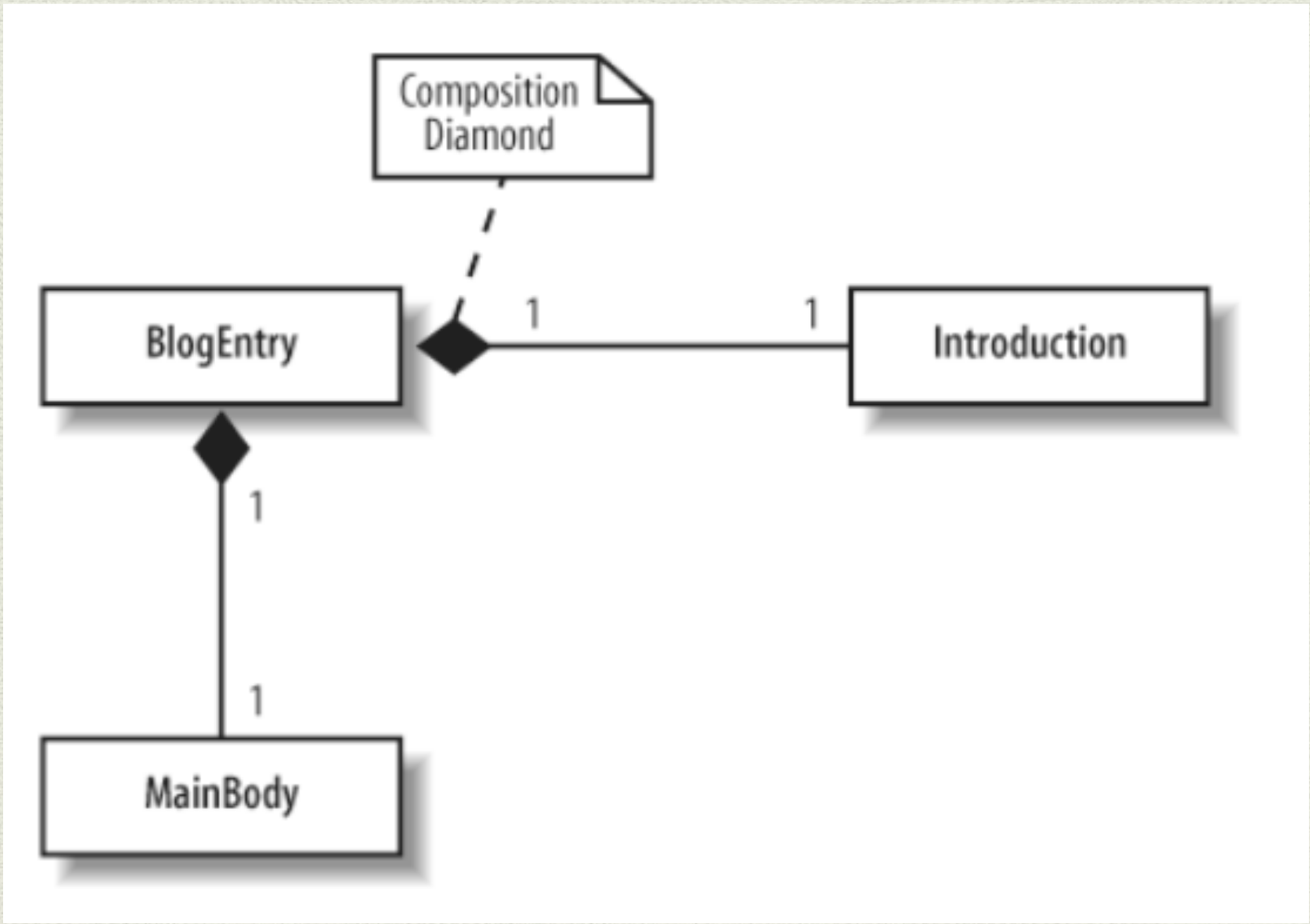
Composition Relationships

- A strong form of aggregation.
- It is a whole-part relationship between a composite (the whole) and its parts, in which the parts must belong only to one whole and the whole is responsible for creating and destroying its parts when it created or destroyed.
- Composition is shown using a **filled diamond** attached to the class that represents the whole.

Composition Relationships– Examples

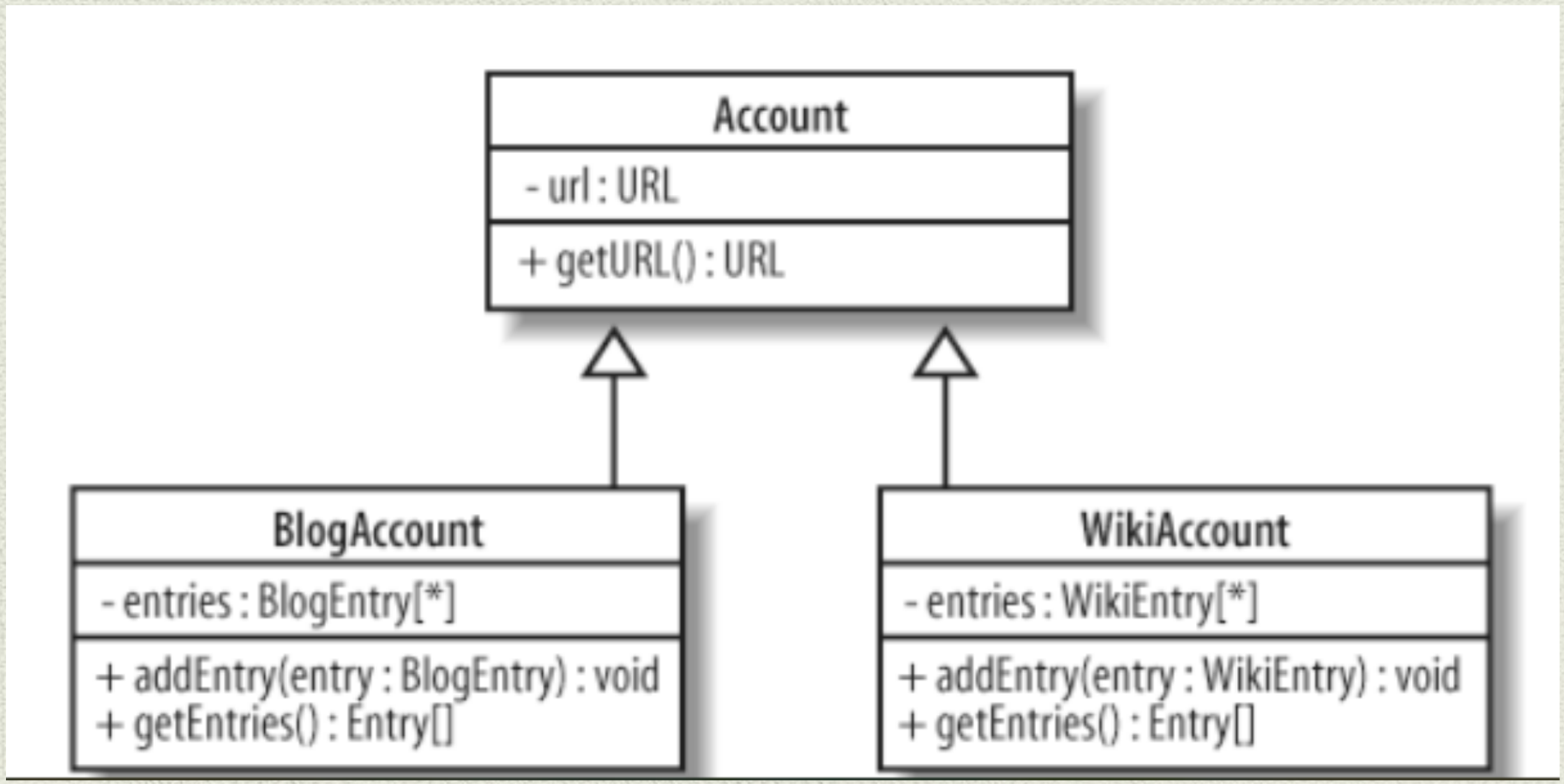


Composition Relationships– Another Example



Generalization Relationships

- In UML, the generalization arrow is used to show that a class is a type of another class



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.