UML

or how to outline/describe your well-engineered software

Communicating Your Code

- Consider the band example. To describe the code we had to write it out.
  - Darned inefficient.
  - Hard to take in "what the code does" at a glance.

Is There A Better Way?

1. Hard to communicate code!

2. Is this a good design?
    1. Am I headed in the right direction?
    2. How do I know?
    3. High connectivity?
    4. Low Cohesion?

Help!

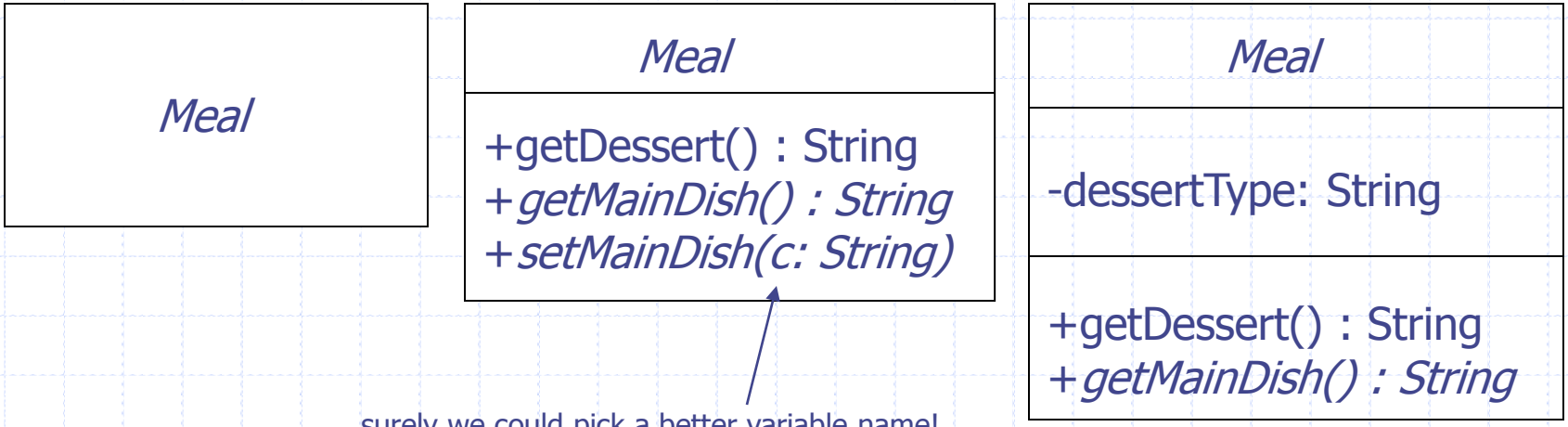
Unified Modeling Language

- Used to create diagrams of programs.
- Show relationships between objects.
  - shows the classes/objects
  - shows messaging between the objects

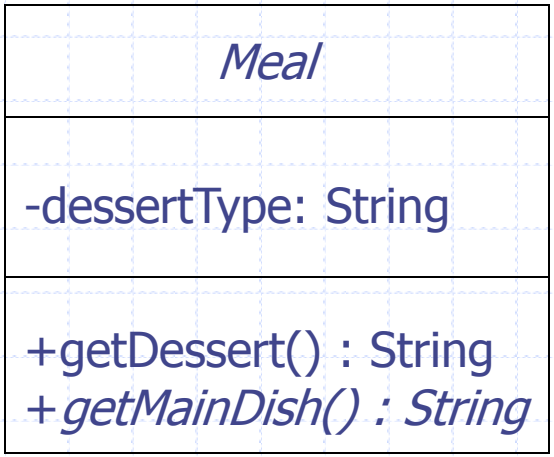
- Organizes thoughts when doing OO design
- **Communicates** your design to others
  - last example was hard to understand because just a bunch of code!

UML Class Diagrams

- Represent a class as a rectangle.
- Put name in Rectangle (*italic* if abstract)
- Put methods and variables in lower rectangles as needed (+ if public, - if private)



Class Diagrams: State and Behavior



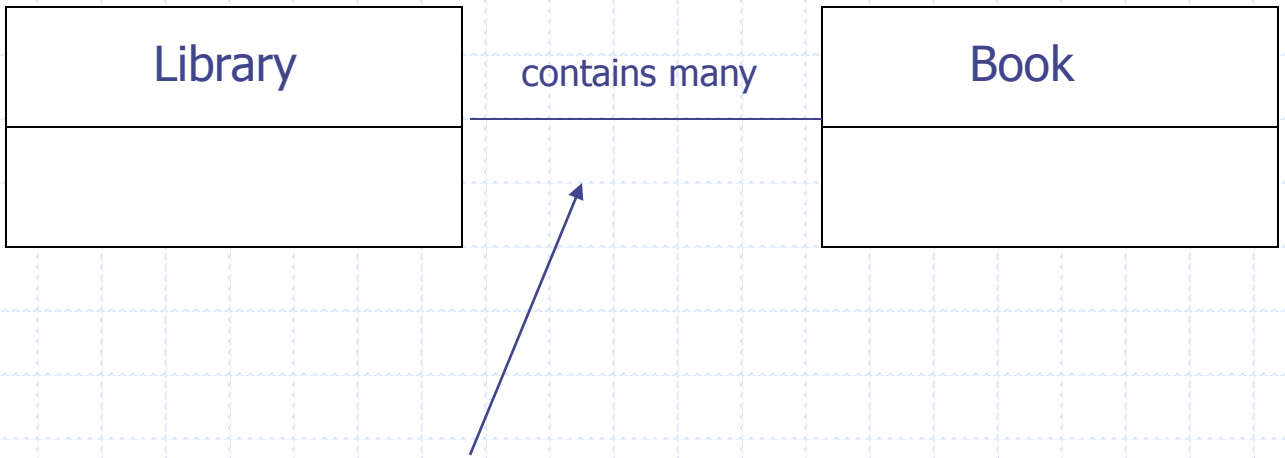
put STATE attributes here
(remember state?)

put BEHAVIOR here
(i.e., response to messages)

What about IDENTITY? Nope.
This is a class diagram, not an object diagram.

Class Diagrams: Association

- Draw a line to indicate a relationship.

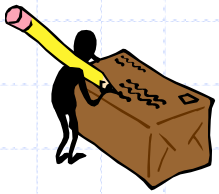


line indicates "association" between classes

frequently add a phrase to clarify the association

Class Diagrams: Association 2

- When do you draw an association?



- When an object of class A sends a message to an object of class B

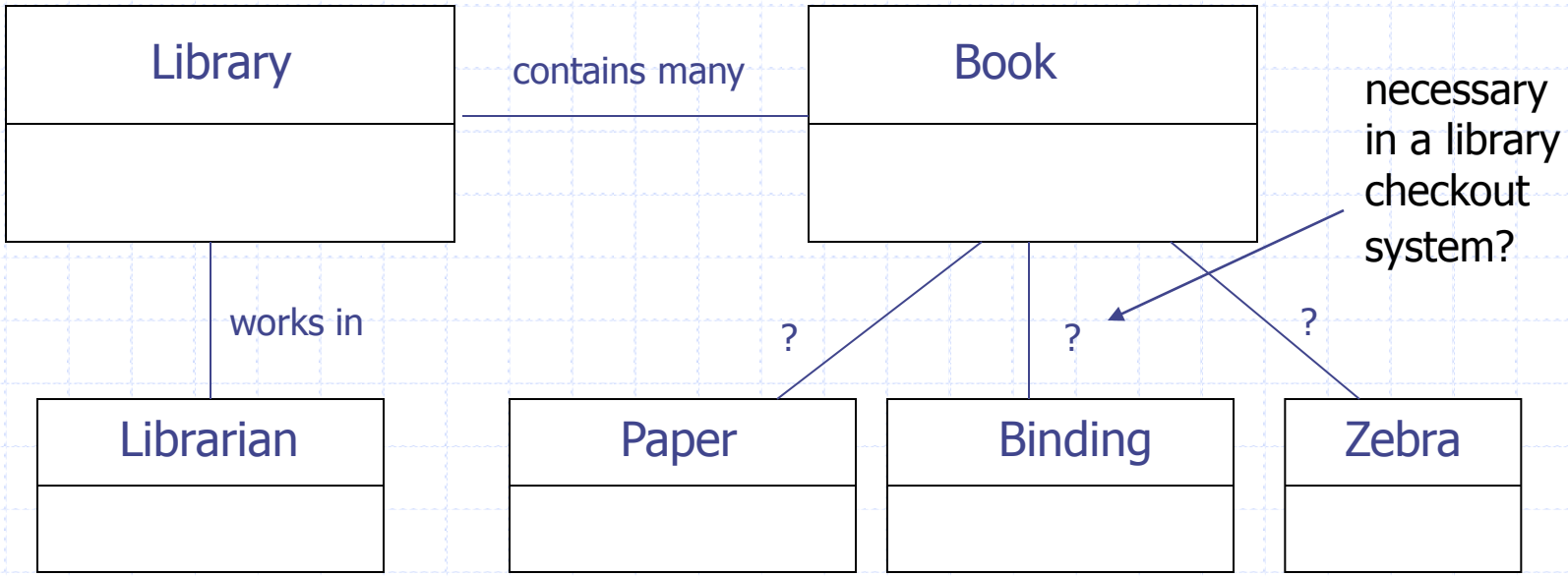
- When class A creates or contains an object of class B
  - if "B myObject = new B()" happens inside of class A

- When class A receives a message containing an object of class B as an argument.

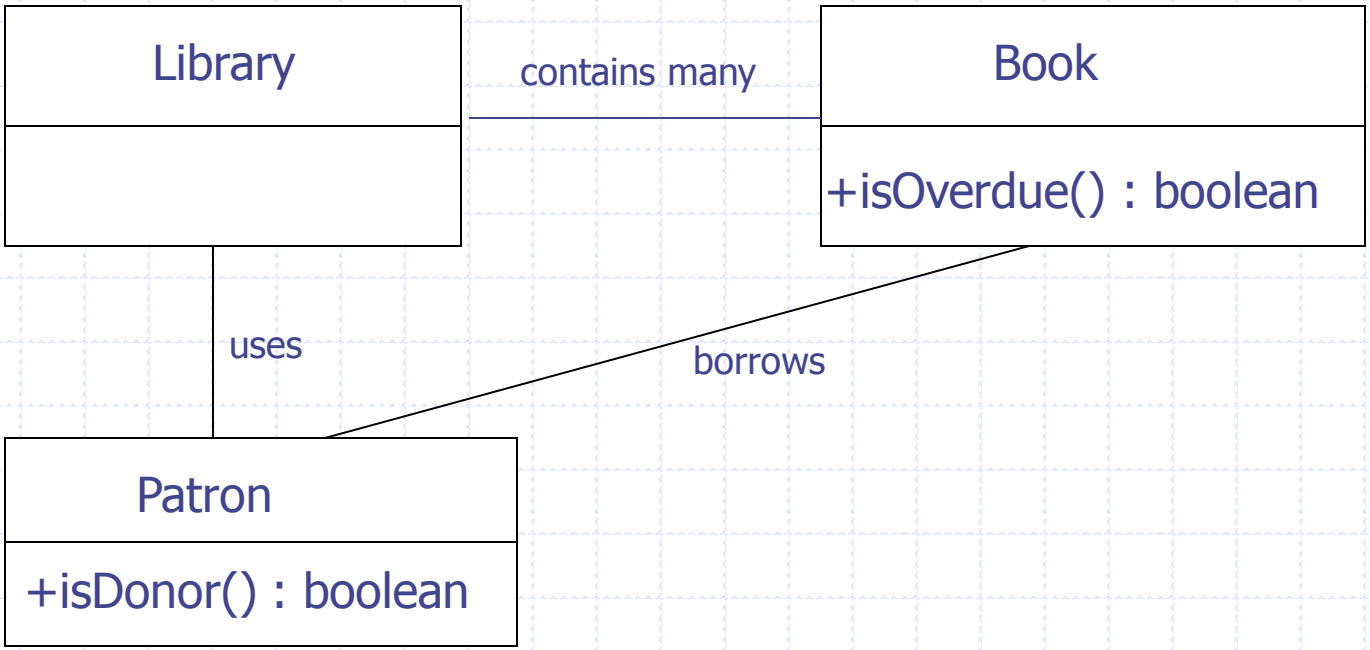
- When it is **not** confusing (don't draw one every bloody time you use a String or Character or Boolean…!)

Class Diagrams: Association 2

- In other words, draw an association when some object of class A has to know about an object of class B.



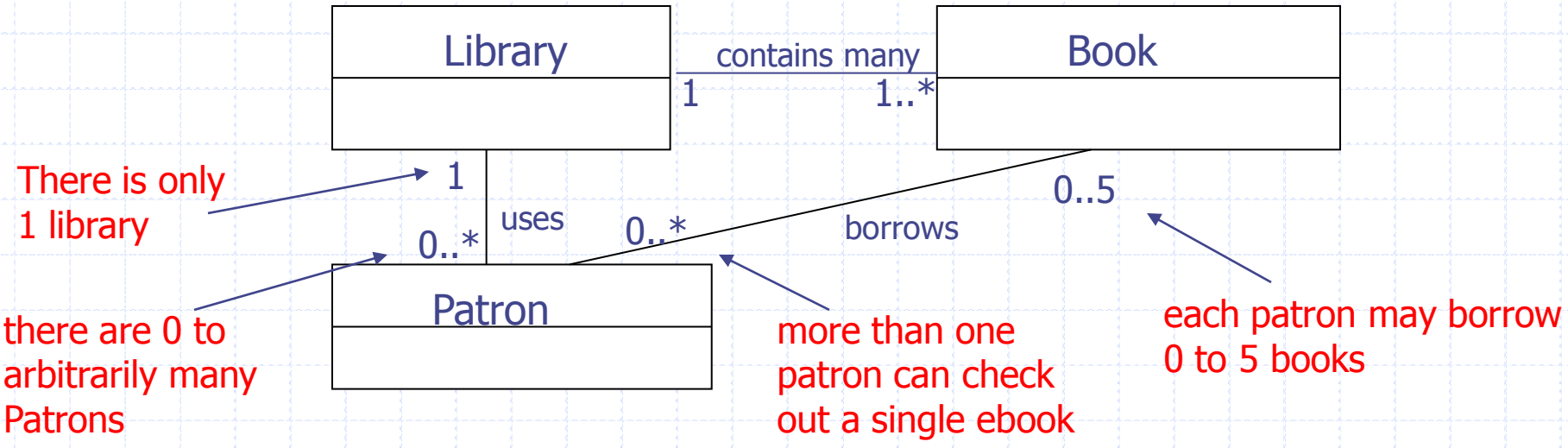
Class Diagrams: Association Example



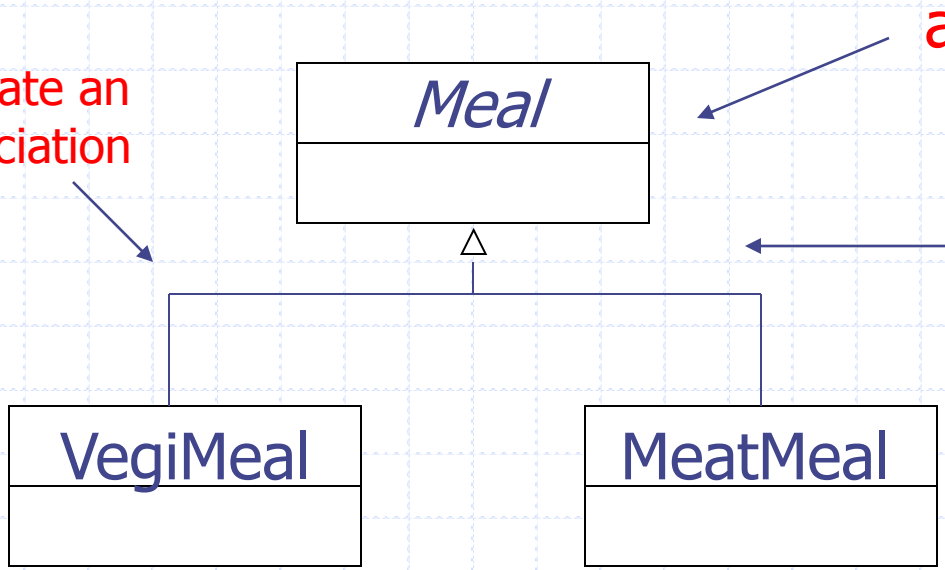
Class Diagrams: Multiplicity

- **Can be more specific about the association.**

1. How many objects will be instantiated from each class!
2. And/or how many of each are associated.



Class Diagrams: Inheritance (or Generalization)



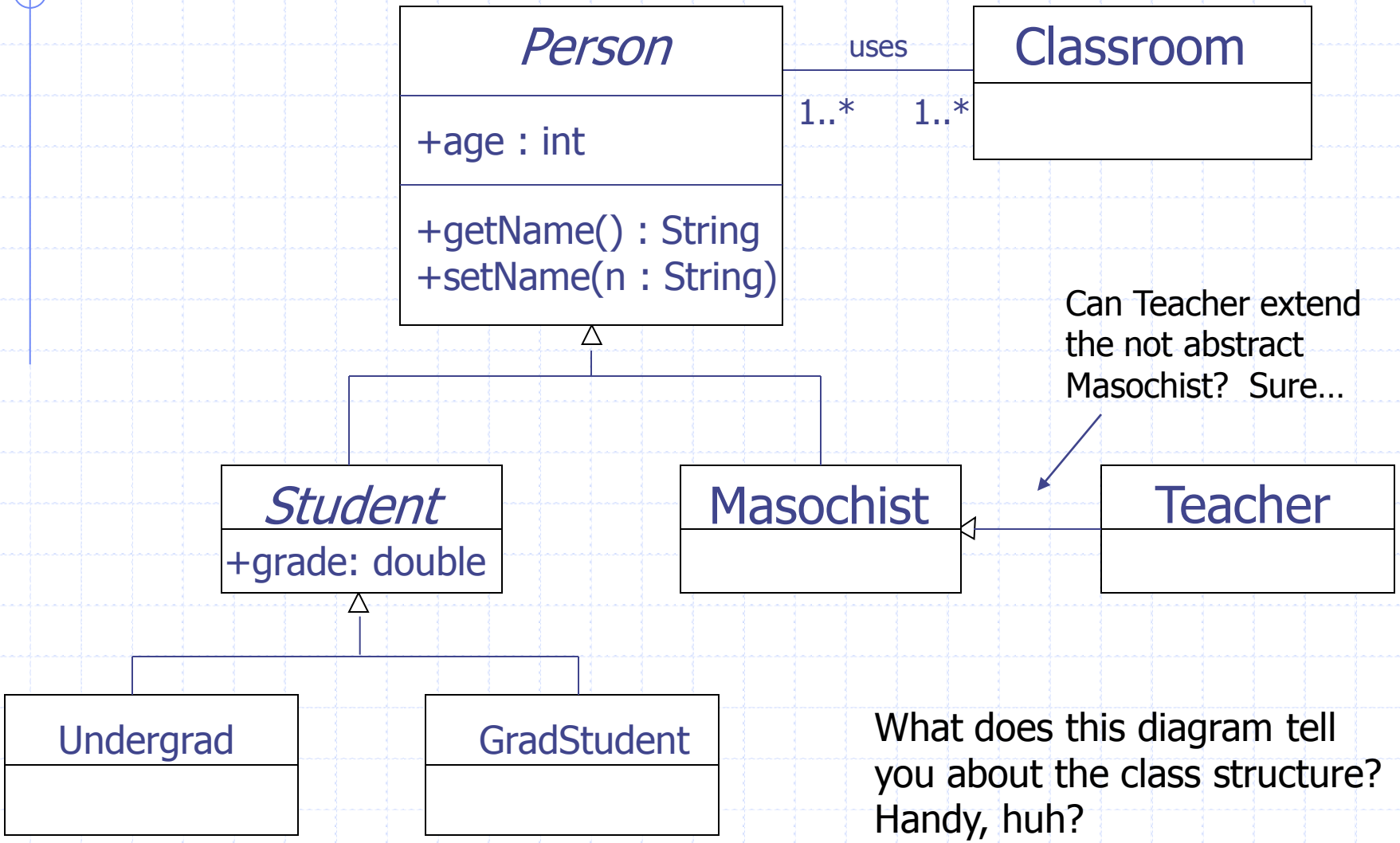
I can **see** the relationships between classes.

Class Diagrams: Inheritance Example

- Suppose I give you the following classes:
  - Student
  - Person
  - Masochist
  - Undergrad,
  - GradStudent
  - Teacher
  - Classroom
- Can you tell me how these are related?
  - What inherits from what? Is there more than one possibility?
  - Which ones are abstract?
- Use UML!

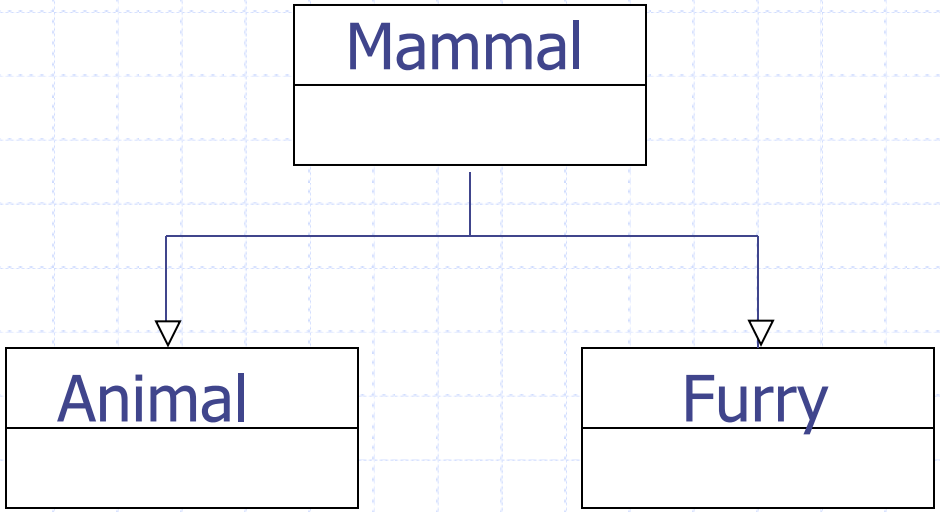
You can only guess at my intent!

Class Diagrams: Inheritance Example (cont.)



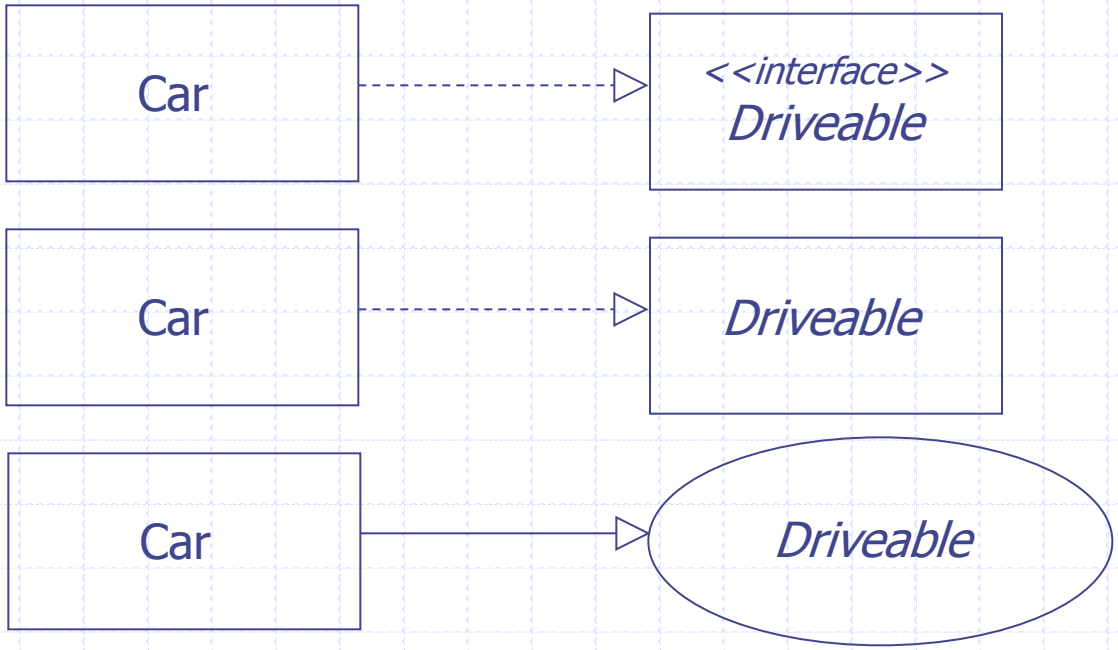
Class Diagrams: Multiple Inheritance

- Why are you punishing yourself?
  - Ok, if you insist…



Class Diagrams: Interfaces

- An interface is generally shown in one of three ways.
  - You may see them all.
  - Use your favorite.



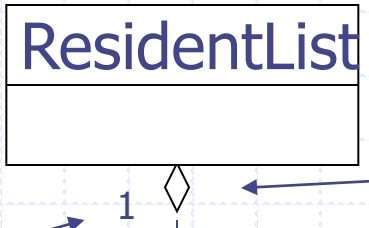
Class Diagrams: Aggregation

- **Aggregation** is when a class is part of another class.
  - A Building contains People
  - An Airport contains Aircraft
  - A Library contains Books
- But it could contain none at all (empty building).
- AND People, Aircraft, and Books don't disappear if we delete the Building, Airport, and Library. (We say the airport *doesn't own* the aircraft.)

Class Diagrams: Aggregation

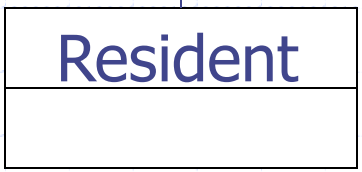
- Aggregation is usually a collection of things.
  - ResidentList
  - Forest (aggregates trees)
- Not required to show aggregation in UML – only if it make an association clearer!

Class Diagrams: Aggregation (Example)



1

1..*



open diamonds indicate "aggregation."
A **Resident** "is a part of" a **ResidentList**.
(Or there are many **Resident**s in a **ResidentList.)**

lines indicate "association"

means there are anywhere from
1 to an infinite number (*) of Residents

means there is just one (1) ResidentList

Class Diagrams: Aggregation Code Example 1

```
public class Lunch
{
    private Sandwich sandwich;

    public Lunch(Sandwich sandwich)
    {
        this.sandwich = sandwich;
    }
}
```

Lunch contains a Sandwich.

The sandwich is created outside the Lunch. So it does not go away when the Lunch goes away. Hence aggregation!

Class Diagrams: Aggregation Code Example 2

```
public class Lunch
{
    private List meal = new ArrayList();

    public void addLunchItem(Object lunchItem)
    {
        meal.add(lunchItem);
    }

    public void removeLunchItem(Object lunchItem)
    {
        meal.remove(lunchItem);
    }
}
```

Lunch contains a list of meal items.

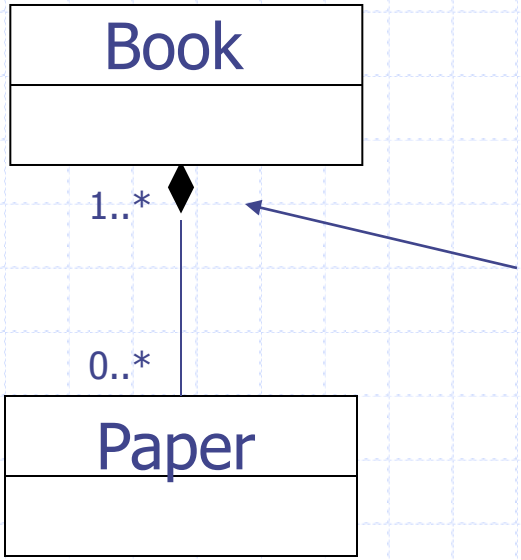
The lunchItems are created outside the Lunch. So lunchItems do not go away when the Lunch goes away. Hence aggregation!

By the way, can you tell me why the uses of List and Object are good engineering? (Casting!)

Class Diagrams: Composition

- Very similar to aggregation, but in **composition** the parts disappear if we erase the whole.
  - Book contains Pages
    - Deleting the Book class would eliminate the instances of the Pages as well.
    - We say that "the Book *wholly owns* the Pages"
    - The Pages have no reason to exist without the book.

Class Diagrams: Composition Example



Solid diamond indicates composition.

So what's this diagram mean?

Class Diagrams: Solid or Open Diamond?

- A mnemonic:
  - When you erase a solid diamond, you have to erase the border and the interior.
    - just as when you erase the composite class you have to erase the parts as well.
  - When you erase an open diamond, you only have to erase the border.
    - just as when you erase the aggregate class, the insides don't have to be erased.

Class Diagrams: Composition Code Example

```
public class Ocean
{
    private Water water;

    public Ocean()
    {
        water = new Water();
    }
}
```

The Water is created **inside** the Ocean. So it goes away when the Ocean goes away. Hence composition!

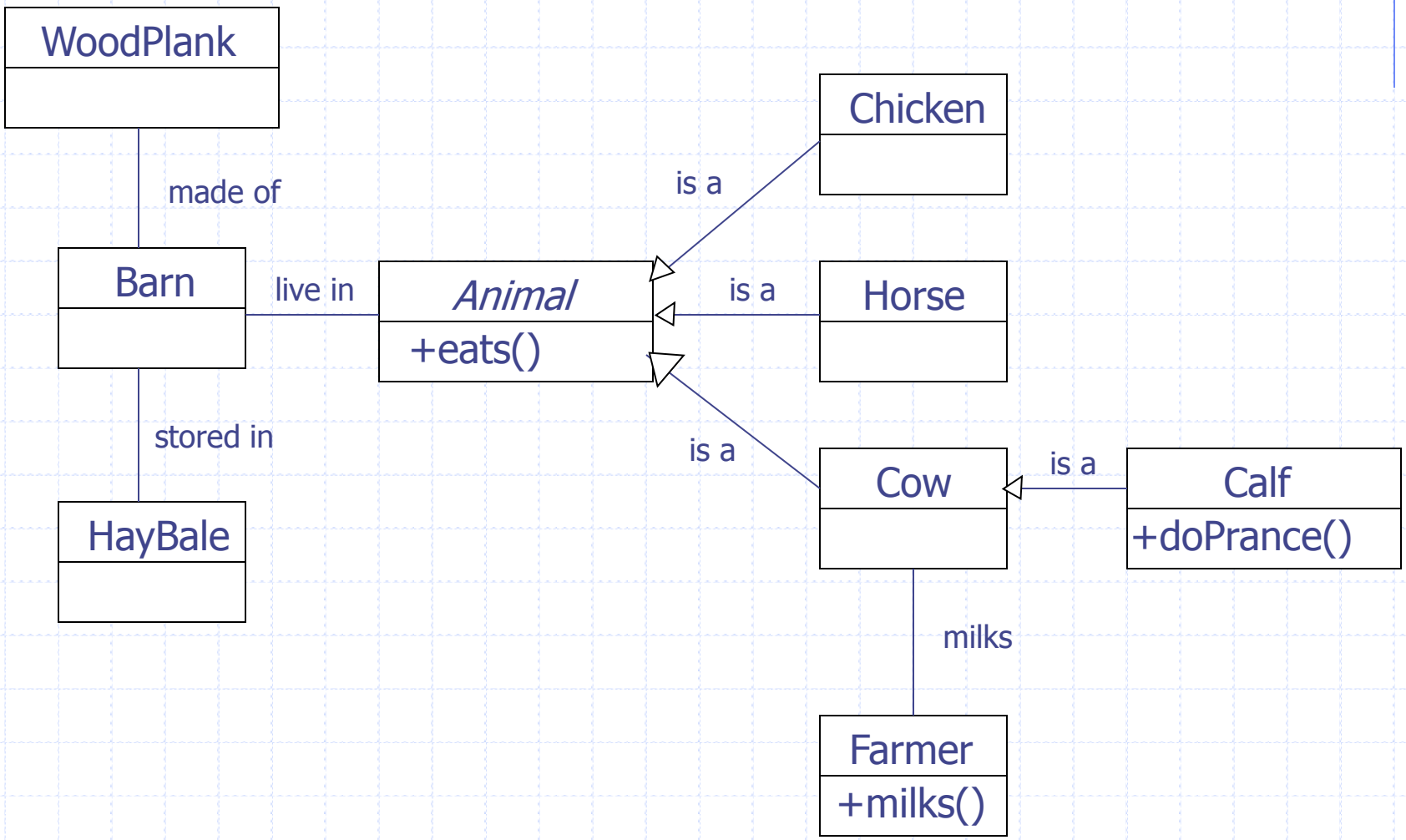
Barn Example

Draw a class diagram for a barn that contains some animals such as chickens, horses, cows, and calfs. A farmer milks any cows in the barn, and the animals eat hay bales that are stored in the barn. The barn is made of wood planks.

Barn Example: Classes

- Potential classes
  - Barn, Animal, Chicken, Horse, Cow, Calf, HayBale, WoodPlanks, Farmer
    - Calf and Cow same class? Calf extends Cow (implies has additional behavior)? Just set youth attribute in class Cow? (Depends on intent of the customer that ordered the "barnyard" product.)
    - Chicken, Horse, Cow extend Animal
- Potential methods
  - milks, eats

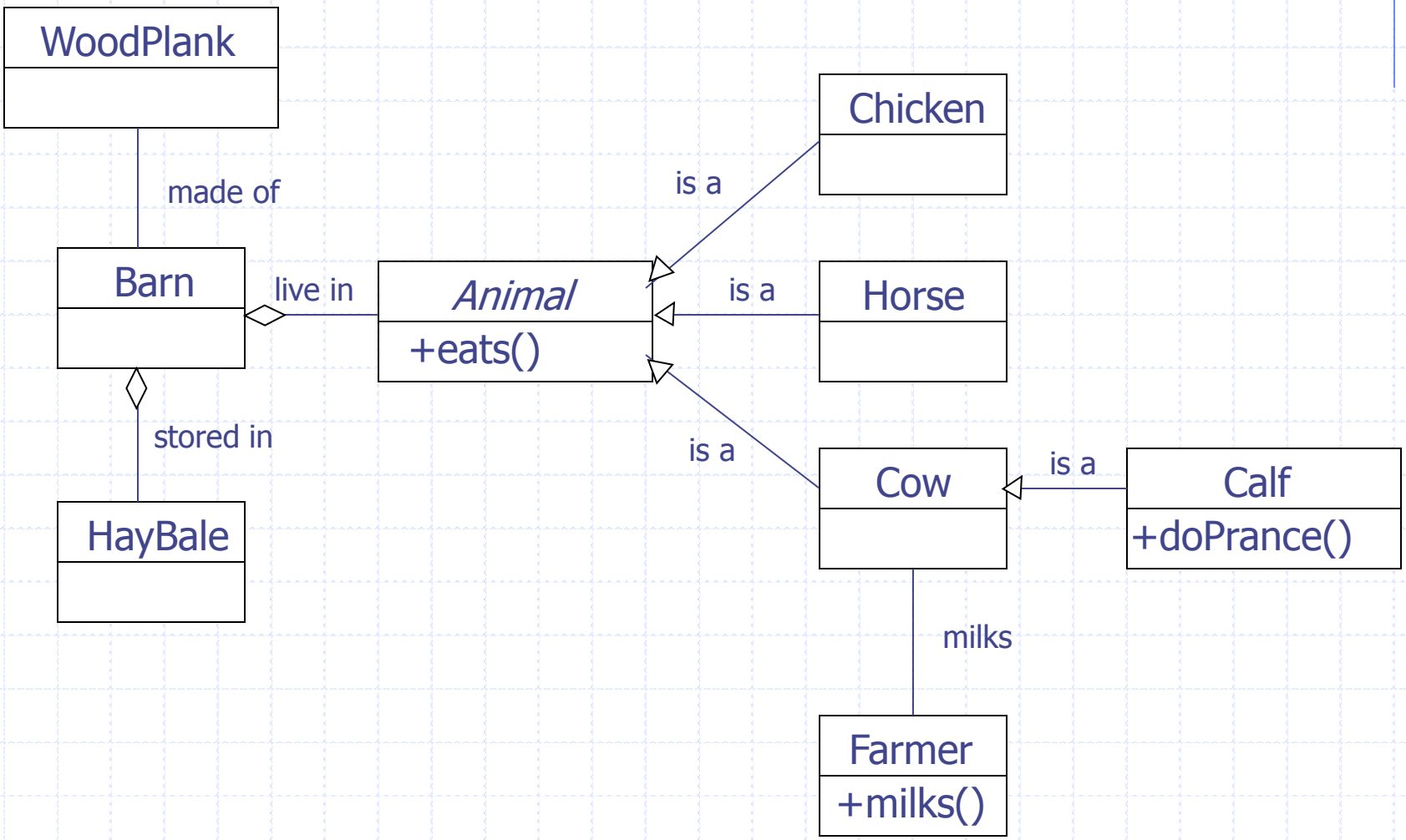
Barn Example: Class Associations (And UML)



Barn Example: Class Aggregation

- *Barn* contains the *Animals* and the *HayBales*. That's a clue to the UML structure.
- **Aggregation**! The Animals and HayBales won't disappear if we get rid of the Barn.

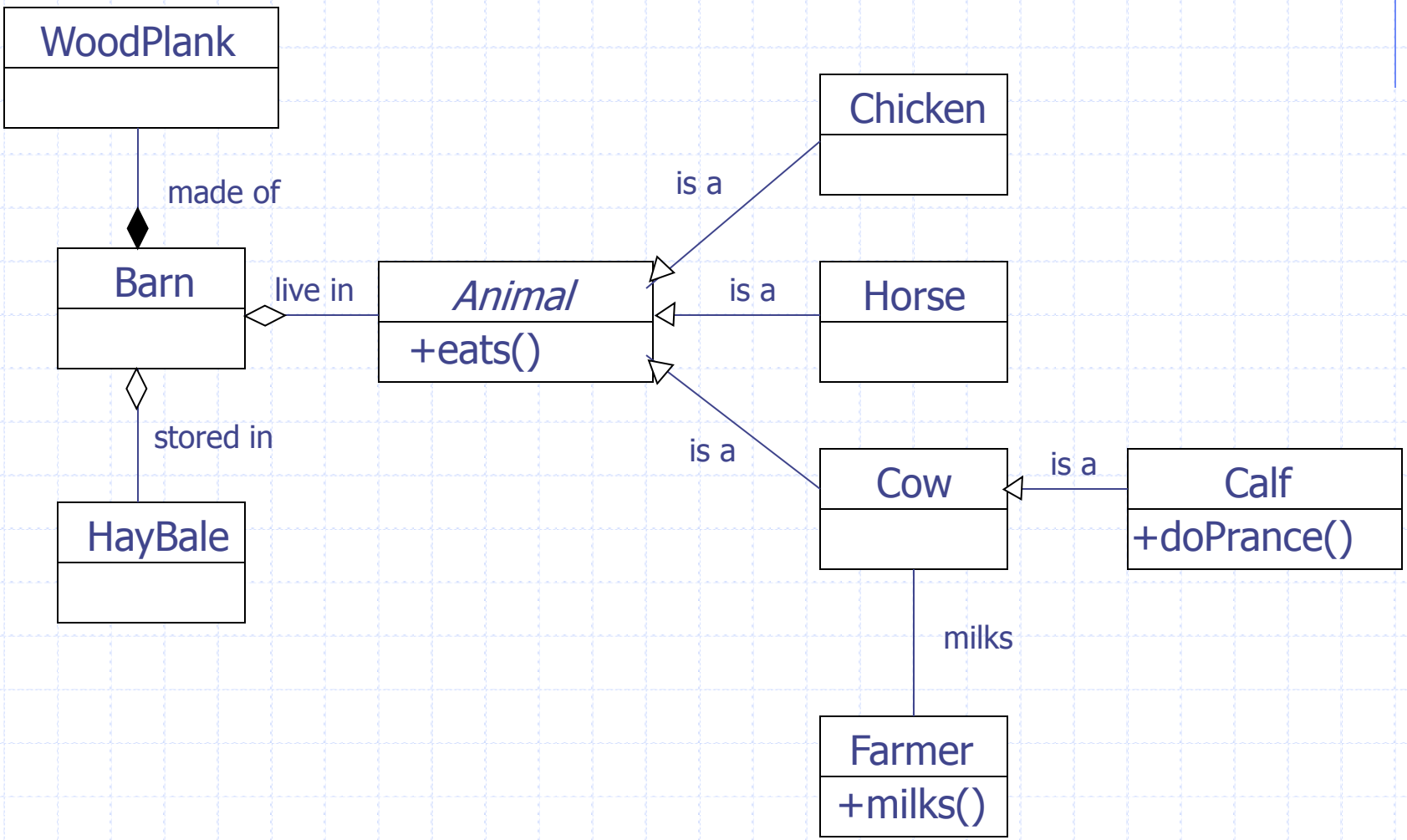
Barn Example: Revised UML



Barn Example: Class Composition

- The WoodPlanks serve no purpose except to be part of the structure of the Barn. That's a clue to the UML structure!
- **Composition**! These instances of the WoodPlanks will disappear if the Barn is removed.

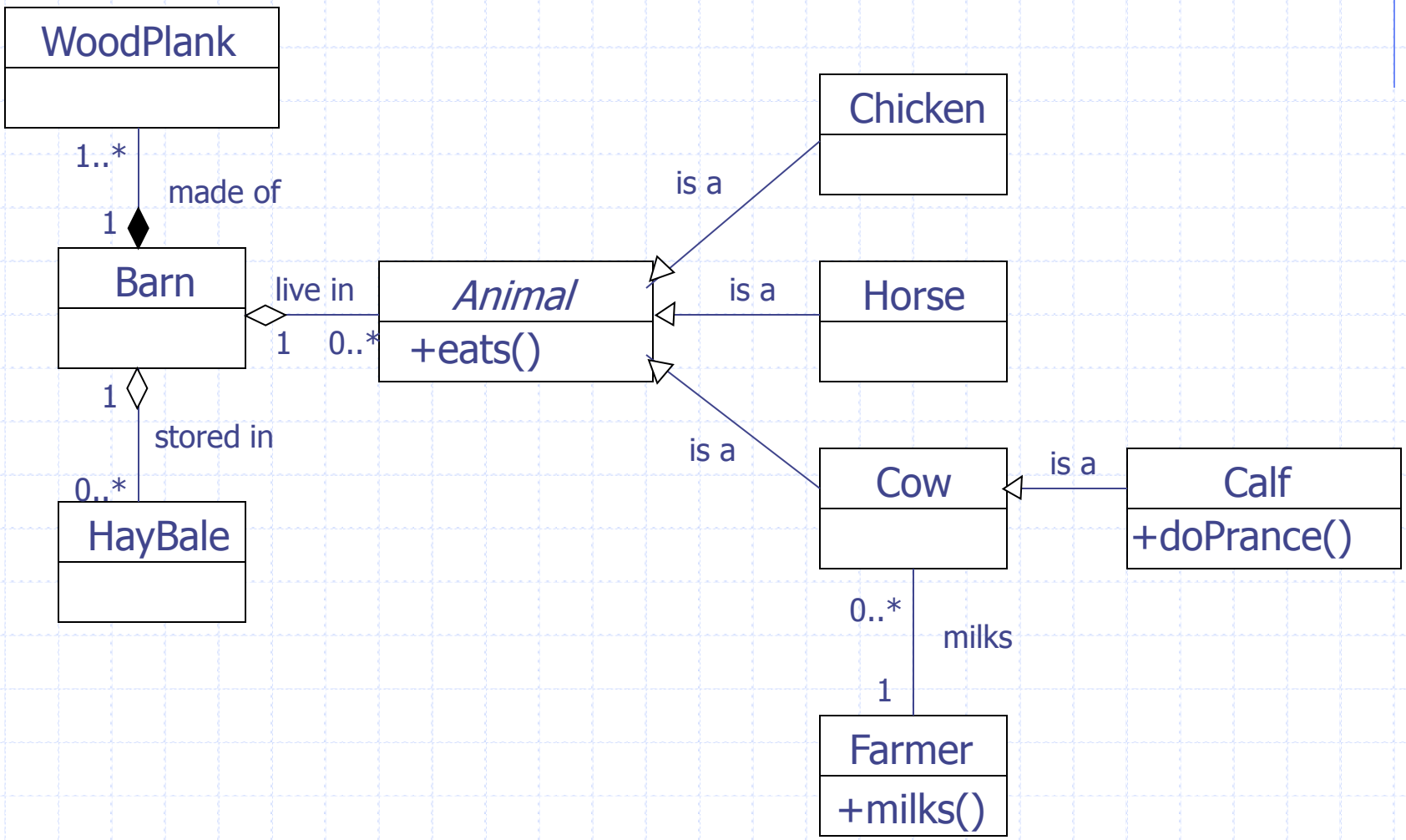
Barn Example: Revised UML



Barn Example: Class Multiplicities

- There is only one Barn (1), but there are an unspecified number of Animals (0..*) and HayBales (0..*).
- The barn is made of at least one and probably many planks (1..*).
- There is only one Farmer, but he may interact (by sending "milking messages") with many Cows (0..*).

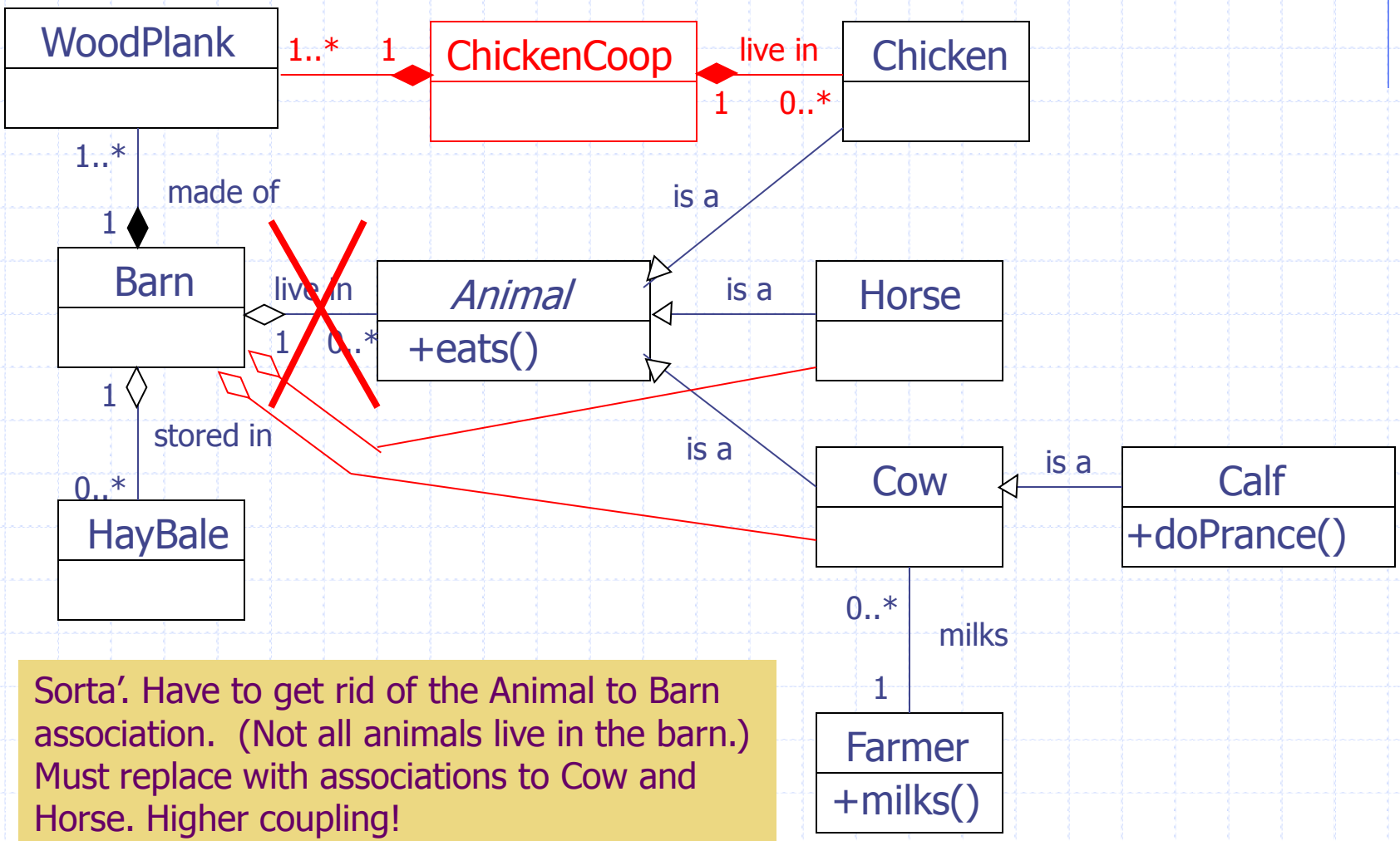
Barn Example: Revised UML



Barnyard Changes!

- What if the farmer now builds a chicken coop made of 100 wood planks? And the chickens live in the coop?
- new class **ChickenCoop**.
- **Composition** is WoodPlanks with **multiplicity** 100.
- Chicken **association** changes.

Did We Do a Good Job Engineering the Original Barnyard?



Sorta'. Have to get rid of the Animal to Barn association. (Not all animals live in the barn.) Must replace with associations to Cow and Horse. Higher coupling!

Better Engineering of Barnyard