

# Introduction to Inheritance

## Lecture 25 Section 15.1

Robb T. Koether

Hampden-Sydney College

Mon, Mar 26, 2018

- 1 Inheritance
- 2 The HAS-A Relation
- 3 The IS-A Relation
- 4 Assignment

# Outline

- 1 Inheritance
- 2 The HAS-A Relation
- 3 The IS-A Relation
- 4 Assignment

# Object-Oriented Programming

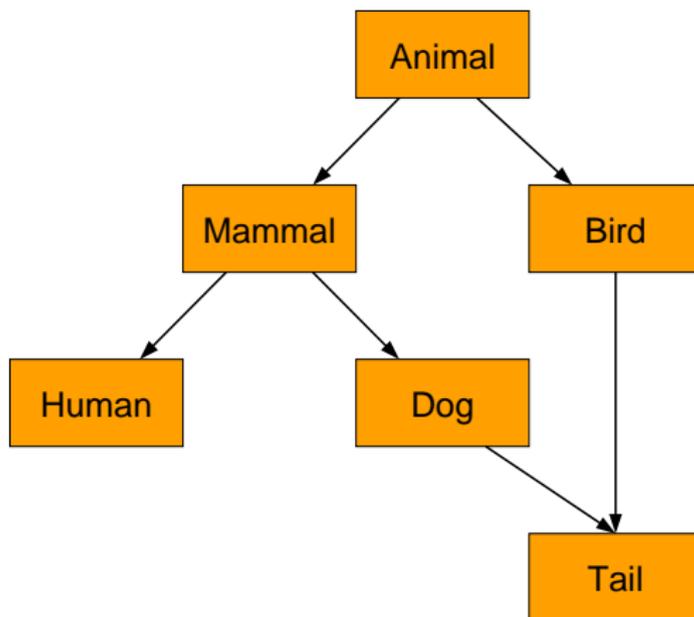
- The basic principle of object-oriented programming is to create objects and relationships between them that reflect reality.
- This should make the programming more intuitive.
- But it does take effort.

# Example

## Example (Relationships)

- What are the relationships among the following types of object?
  - human
  - mammal
  - dog
  - bird
  - animal
  - tail

# Hierarchy of Objects



# Inheritance in C++

- In C++ a new class may be **derived** from an existing class.
- The new class is called the **derived** class or the **subclass**.
- The original class is called the **base class** or the **superclass**.

# Inheritance in C++

- The subclass **inherits** all the data members and member functions of the base class.
- The subclass also has its own data members and member functions.
- Furthermore, the subclass may redefine inherited functions.

# Example

- We could derive the `Mammal` class from the `Animal` class.
- Does a mammal have all the attributes of animals in general?

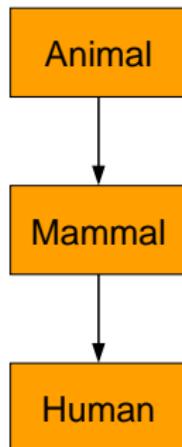
# Example

- We could derive the `Human` class from the `Mammal` class.
- Does a human have all the attributes of mammals in general?

# Example

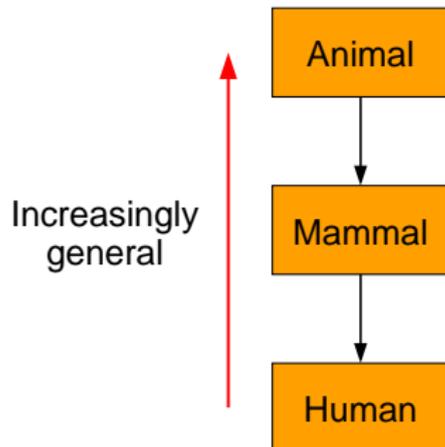
- Should we derive the `Tail` class from the `Dog` class?
- Does a tail have all the attributes of dogs in general?

# General Guidelines



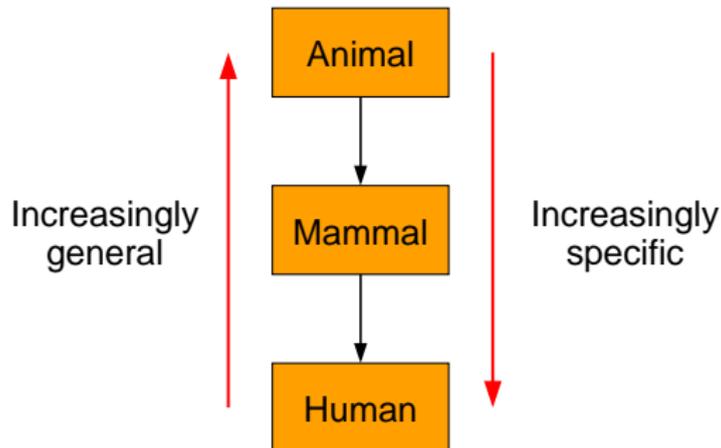
Generally speaking, the hierarchy should go from the most general class at the top down to the most specific class at the bottom.

# General Guidelines



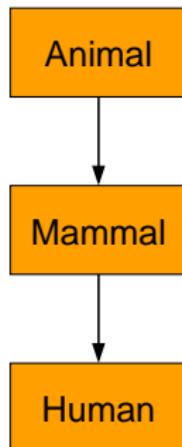
Generally speaking, the hierarchy should go from the most general class at the top down to the most specific class at the bottom.

# General Guidelines



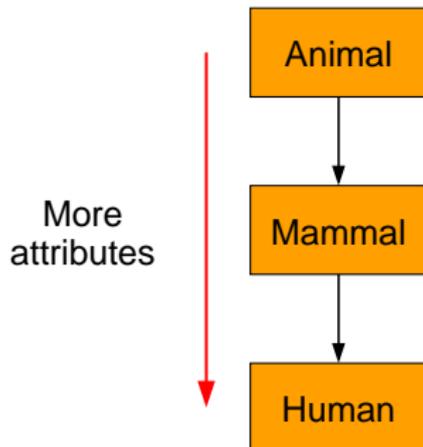
Generally speaking, the hierarchy should go from the most general class at the top down to the most specific class at the bottom.

# General Guidelines



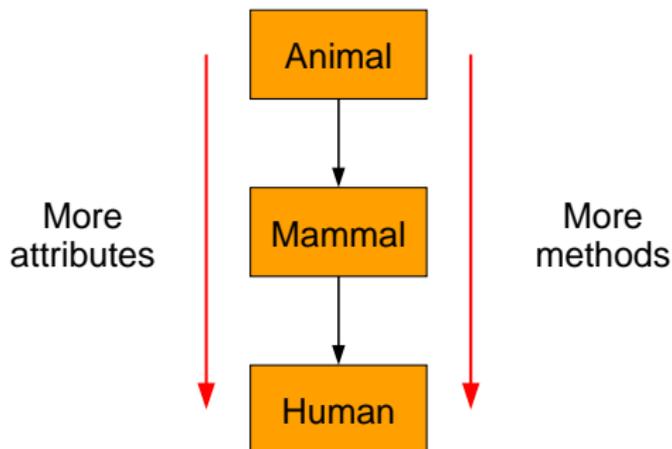
Generally speaking, the classes lower in the hierarchy should have all the attributes of the classes higher in the hierarchy, as well as additional attributes of their own.

# General Guidelines



Generally speaking, the classes lower in the hierarchy should have all the attributes of the classes higher in the hierarchy, as well as additional attributes of their own.

# General Guidelines

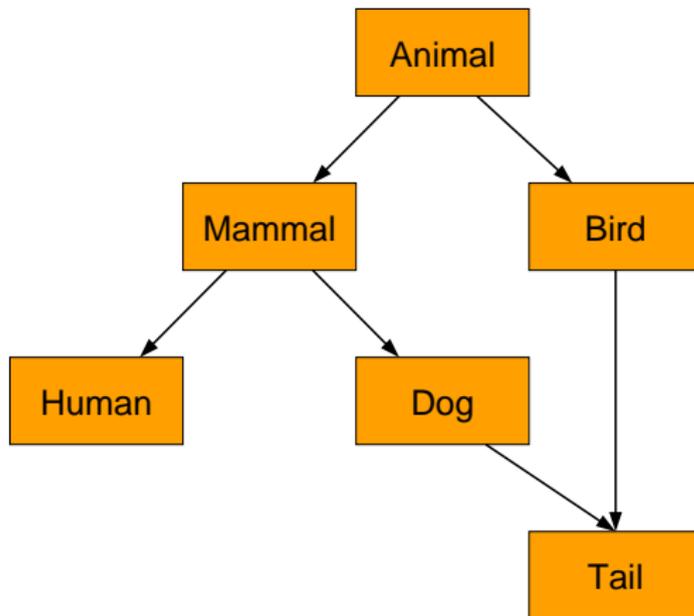


Generally speaking, the classes lower in the hierarchy should have all the attributes of the classes higher in the hierarchy, as well as additional attributes of their own.

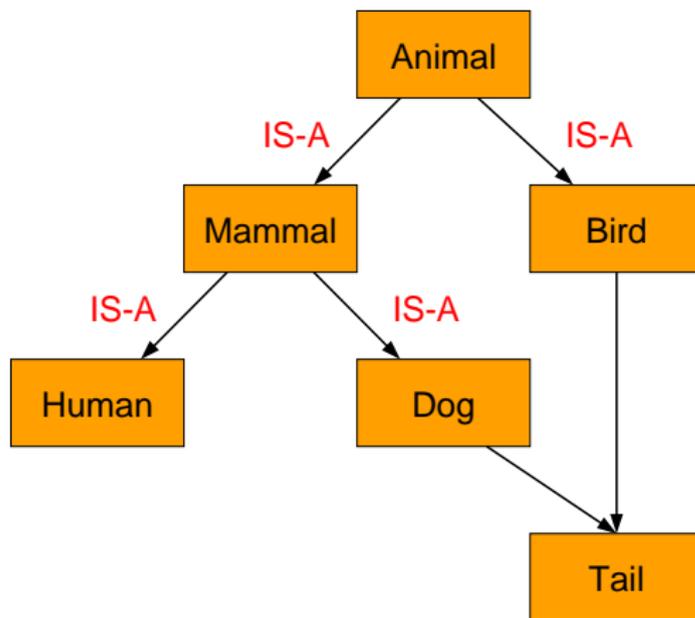
# Design Decisions

- It is not always easy to decide when to derive one class from another.
- The decision should reflect the way we think about the classes.
- Two helpful relations
  - IS-A
  - HAS-A

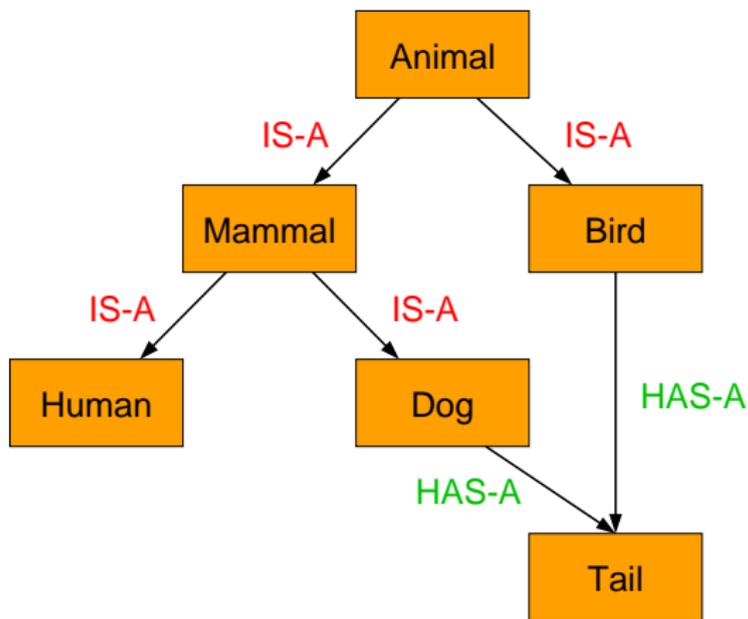
# Hierarchy of Objects



# Hierarchy of Objects



# Hierarchy of Objects



# Outline

1 Inheritance

**2 The HAS-A Relation**

3 The IS-A Relation

4 Assignment

# The HAS-A Relation

- If an object of class A HAS a component that is an object of class B, then the HAS-A relation holds from class A to class B.
- If every class A object HAS-A class B component, then an object of class B should be a **data member** of class A.

# Example

## Example (HAS-A Relation)

- A `Car` has an `Engine`.
- A `Car` has `Wheels`.
- The HAS-A relation holds from classes `Engine` and `Wheel` to class `Car`.

# Example

## Example (HAS-A Relation)

```
class Engine;  
class Wheel;  
class Car  
{  
  // Data members  
  Engine e;  
  Wheel w[4];  
};
```

# Outline

- 1 Inheritance
- 2 The HAS-A Relation
- 3 The IS-A Relation**
- 4 Assignment

# The IS-A Relation

- If an object of class A IS an object of a more general class B, then the IS-A relation holds from class A to class B.
- If every class A object IS-A class B object, then class A should be a **derived class** of class B.

# Example

## Example (IS-A Relation)

- A `Car` is a `MotorVehicle`.
- A `MotorVehicle` is a `Vehicle`.
- The IS-A relation holds from class `Car` to class `MotorVehicle` and from `MotorVehicle` to `Vehicle`.

# Example

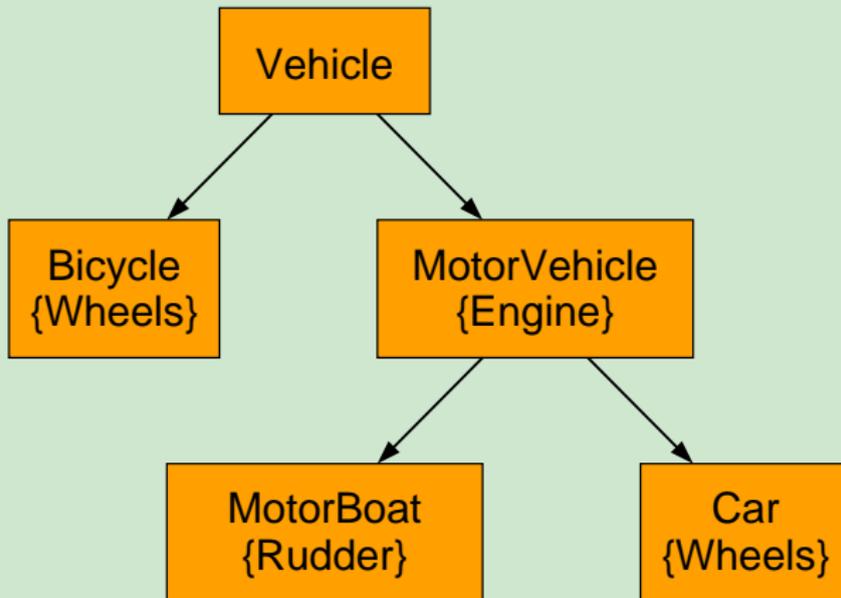
## Example (IS-A Relation)

```
class Vehicle;
class MotorVehicle : public Vehicle
{
    Engine e;
};
class Car : public MotorVehicle;
{
    Wheel w[4];
};
class Boat : public Vehicle;
{
    Rudder r;
};
```

- Where would a `Motorboat` class fit in the hierarchy?

# Example

## Example (IS-A Relation)



## Example

- Arrange the following classes in natural hierarchies.
  - List
  - ArrayList
  - LinkedList
  - LinkedListNode
  - LinkedListwTail
  - DoublyLinkedList
  - DoublyLinkedListNode
  - CircLinkedList

# Outline

- 1 Inheritance
- 2 The HAS-A Relation
- 3 The IS-A Relation
- 4 Assignment**

# Assignment

## Assignment

- Read Section 15.1.