

Features of OO Languages

I. Classes

Most OO languages have some sort of class construct, e.g., Java, C++, Simula, Smalltalk, CLOS (Common Lisp Object System)...

A class groups together data (attributes) and methods (behavior) that make up objects.

Example: C++

```
class Car {
private:
  int cyl;
  bool has_tires;
  Color color;
  Style body_style;
public:
  int get_num_cyl();
  bool has_tires();
  void set_color(Color c);
};
```

Attributes {

Behavior {

Chap 16

Classes

A class is instantiable. A class itself is not an object¹, but a class serves as a mold for building objects in your program - instantiable objects

Example: Java, C++

```
class Car {
  // attributes
  // behavior
};

...
Car c = new Car();
Car c1 = new Car();
Car c2 = new Car();
Car c3 = new Car();
```

Instantiate new object from class Car.

c is a reference to the new object.

Three more objects based on class Car

Observation: In most OO languages the class name also serves as a type name - for type checking.

¹ well, that is not true for all languages - CLOS, Smalltalk treat classes as objects themselves making them dynamic.

Inheritance

II. Inheritance

One of the goals of OO languages and OO programming is to represent real-world objects and concepts effectively in software.

We saw already that objects can be represented as classes in OO languages, but in the real-world objects are related.

Example:

a bird is an animal
a duck is a bird
a chicken is a bird
a turkey is a bird

The words in *italic* represent object classes.

In order to fulfill the promise of OO programming we need to be able to capture these kinds of object relationships in the programming language ⇒ Inheritance

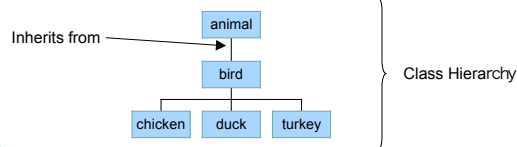
Inheritance

That the notion of inheritance is appropriate here can be shown by a simple linguistic trick:

a bird is an animal
a duck is a bird
a chicken is a bird
a turkey is a bird



a bird inherits from animal
a duck inherits from bird
a chicken inherits from bird
a turkey inherits from bird

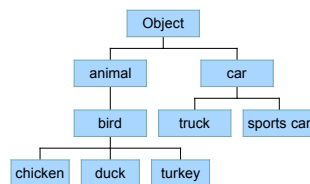


```
class Animal (...);  
class Bird extends Animal (...);  
class Chicken extends Bird (...);  
class Duck extends Bird (...);  
class Turkey extends Bird (...);
```

Inheritance: Observations

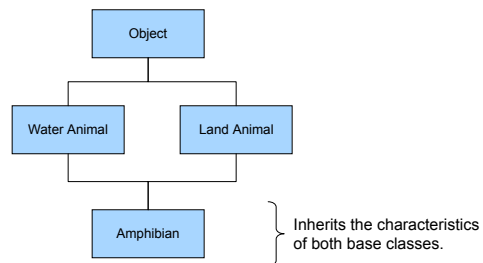
In most OO languages the derived class inherits both the attributes and the behavior from its super-class (or base class).

Some OO languages provide an implicit common root to user constructed class hierarchies - in Java all user defined base classes are implicitly derived from Object.



Multiple Inheritance

Some OO language (e.g. C++) allow for multiple inheritance.



Note: there are many technical challenges with multiple inheritance, the most important one is the "diamond inheritance" problem of base classes which results in replicated attributes and behaviors of the "diamond" base class - therefore, the designers of Java did not allow for multiple inheritance.

Encapsulation & Polymorphism

III. Classes and inheritance enable encapsulation and polymorphism.

a) Encapsulation

Encapsulation or information hiding is defined as the bundling of data with its methods.

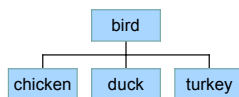
Classes provide this bundling in a natural way by mimicking natural objects - i.e. state and behavior go hand in hand. Usually the state is further protected with access restrictions such as public/private.

Encapsulation is important for programming in the large - if in large projects everything would be visible to everybody all the time, then programmers would easily become overwhelmed.

Encapsulation & Polymorphism

b) Polymorphism

Inheritance allows us to look at objects at different levels of abstract.



For any given object in this hierarchy we can talk about it in terms of either being a duck, chicken, or turkey, or we can just talk about it in terms of being a bird.

Polymorphic Programming

Example: polymorphic programming in Java, assume the previous bird class hierarchy.

```
Bird[] cage = new Bird[10];
...
cage[0] = new Duck();
cage[1] = new Turkey();
cage[2] = new Chicken();
cage[3] = new Chicken();
cage[4] = new Chicken();
...
// assume the Bird class has a 'feed' behavior
for (int i = 0; i < 5; i++)
    cage[i].feed();
```