Inheritance

***DogsAndCats* example**

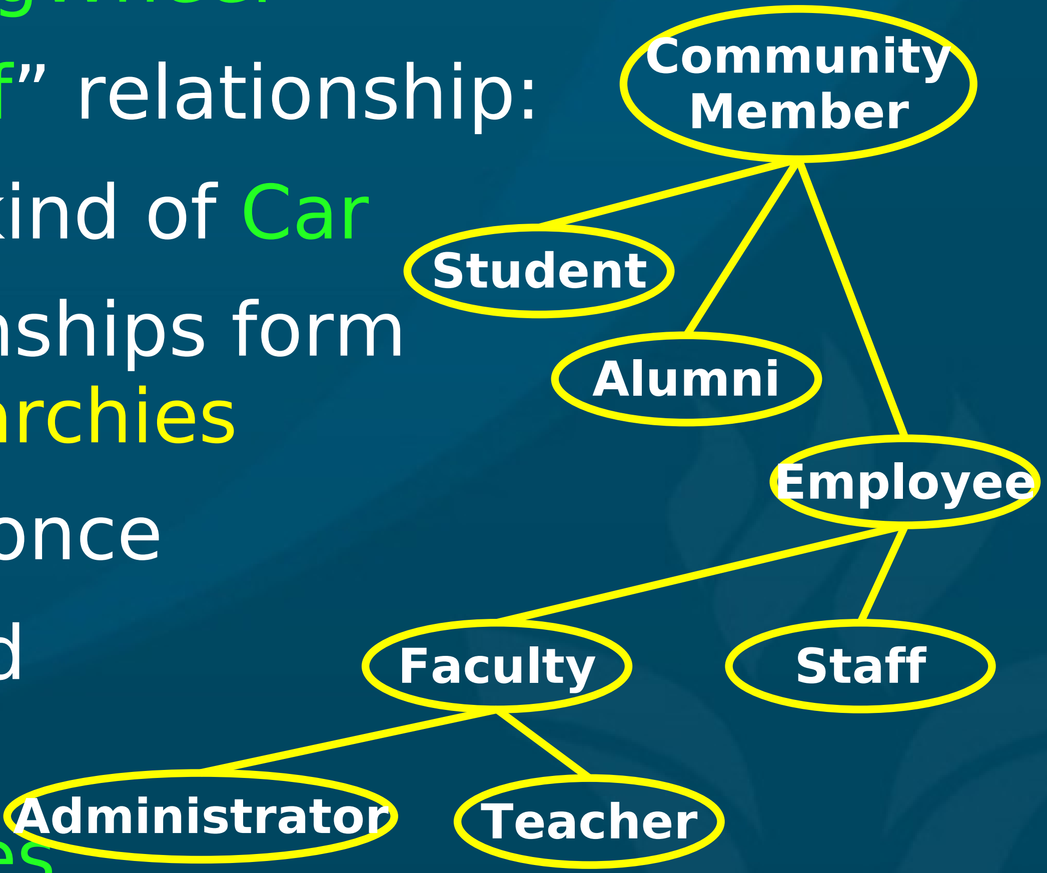
28 Jan 2009
CMPT166
Dr. Sean Ho
Trinity Western University

Why use inheritance?

- Reusability
  - Create new classes from existing ones
    - Absorb attributes and behaviours
    - Add new capabilities
- Polymorphism
    - Enable developers to write programs with a general design
    - A single program can handle a variety of existing and future classes
    - Aids in extending program, adding new capabilities

Superclasses and subclasses

- Attribute: "has a" relationship:
  - A Car has a steeringWheel
- Subclass: "is a kind of" relationship:
  - A Convertible is a kind of Car
  - Inheritance relationships form tree-like class hierarchies
- Polymorphism: write once
  - changeOil() method
  - works on all Cars, not just Convertibles



Subclassing in C++

- When declaring a class, indicate its superclass (parent):
    - class Dog : public Pet { ....
  - A Dog is a kind of Pet
  - Inherits everything Pet has
  - Can add Dog-specific attribs/methods
- Inherit as public
  - So all public members of Pet stay public
  - Otherwise they become private in Dog

public/protected/private

- Recall that protected means:
    - inaccessible to outside world
    - but accessible to methods in a subclass
- So any protected member of Pet is accessible to Dog (but not private members)
- Rule of thumb: make all attributes private or protected by default
    - Write set/get functions as needed

Note: default parameters

- Methods may have default values for tail-end parameters:
    - **void say(string msg = “Hello!”) {**
        - **cout << msg << endl;**
    - **}**
- Useful for constructors:
    - **class Stack {**
        - **Stack(int size = 0);**
    - **}**
    - **Stack myStack(5);**
    - **Stack yourStack();**

Overloading functions

- We've seen operators like '<<' that have different meanings depending on the type of the operands
  - What does '<<' do on ints? On ofstreams?
- This is called overloading
- We can overload functions using multiple definitions with different parameter lists:
  - int dbl(int x) return 2*x;
  - float dbl(float x) return 2.0*x;
  - string dbl(string x) return x+x;
- Overloading vs. default parameters?

Constructors

- When an object (variable) is instantiated (created) in a block, its memory is allocated and its constructor is called
    - In C++, constructor is always called
    - Destructor is called when object disappears
- Constructor of a subclass should call the superclass constructor first:
        - public Dog() : Pet() { ....
    - Initialize Pet stuff first, then Dog-specific

Upcasting

- A reference to an instance of a subclass may also be treated as an instance of the superclass
    - class Dog : public Pet { …
    - Dog fido
  - Every Dog is also a Pet
- Pointer to fido:
    - Pet* myPetPtr = &fido;
  - This assignment works!
  - "forgets" the object is a Dog, only thinks of it as a generic Pet

Virtual methods

- A subclass can redefine a method defined by the superclass
  - Every Pet knows how to speak()
  - But Dogs speak() differently from Cats
  - Subclasses overload speak()
- Flag the method as virtual in the superclass
- Late binding: which version of speak() to use?
  - Decided at run-time
- Polymorphism: same code works on several different types, all subclasses of the same parent