CS 142
Inheritance II

Rhodes College

Announcements

- Program 2 has been assigned
  - Due Tuesday, February 10, 2015 by 11:55pm (on Moodle)
  - Assignment details on Course Website

Is-a versus has-a

- When an object of class A ***has a***n object of class B, use ***object composition***.
  - Class A will have a field (variable) of class B in its implementation
- When class A ***is a*** specific kind of another class B, use ***inheritance***
  - Class A will inherit from class B.

Is-a or has-a?

- Class A = Animal
  - Heart
  - Porcupine
  - Duck
- Class A= Phone
  - Cell Phone
  - Ringtone
  - Text Message
  - Landline

Automobile/Car Demo

- See vehicles.py and car_demo.py

Constructors with Inheritance

- Derived classes must create their own constructors if you want to initialize extra fields (variables).
- All classes must have a constructor.
  - If you don't write one, a default one with no arguments is generated (behind the scenes) for you.
- Every time an object of a class is constructed, a constructor must be called.
  - Default (no arguments) constructor is used unless otherwise specified.

More on Constructors

```
class Pet(object):
    def makeSound(self):
        print("Grr...")

p = Pet()
print(p)
p.makeSound()
```

This is all legal.

Creates a default constructor (no arguments) since no __init__ method defined.

Output:

```
<__main__.Pet object at 0x02D3EF90>
Grr...
```


More on Constructors

```
class Pet(object):
    def __init__(self, name):
        self.__name = name

    def makeSound(self):
        print("Grr...")

class Cat(Pet):
    def makeSound(self):
        print("Meow")

c2 = Cat("Abbey")
print(c2)
c2.makeSound()
```

In Python,

If base case has a constructor, it is called if none is specified in derived class. (This is not true in C++).

This is the same for every method. (you automatically inherit the base class methods.)

If you call a method on an object, the interpreter will look

- First in the object's class definition for that method – if found, it is used
- If not found, it will look for the method in the base class of the object.

Output:

```
<__main__.Cat object at 0x02D5EF90>
Meow
```

```
class Car(Automobile):
    # The __init__ method accepts arguments for the
    # car's make, model, mileage, price, and doors.

    def __init__(self, make, model, mileage, price, doors):
        # Call the superclass's __init__ method and pass
        # the required arguments. Note that we also have
        # to pass self as an argument.
        Automobile.__init__(self, make, model, mileage, price)

        # Initialize the __doors attribute.
        self.__doors = doors

    # The set_doors method is the mutator for the
    # __doors attribute.

    def set_doors(self, doors):
        self.__doors = doors

    # The get_doors method is the accessor for the
    # __doors attribute.

    def get_doors(self):
        return self.__doors
```

If you want to initialize variables not in the base class, you can write a derived class constructor

- **It must call the base class constructor first**
- Then initialize any new variables.
- This allows you to take advantage of the inheritance relationship

Using the Inheritance Relationship

```
# This program demonstrates the Car class.

import vehicles

def main():
    # Create an object from the Car class.
    # The car is a 2007 Audi with 12,500 miles, priced
    # at $21,500.00, and has 4 doors.
    used_car = vehicles.Car('Audi', 2007, 12500, 21500.00, 4)

    # Display the car's data.
    print('Make:', used_car.get_make())
    print('Model:', used_car.get_model())
    print('Mileage:', used_car.get_mileage())
    print('Price:', used_car.get_price())
    print('Number of doors:', used_car.get_doors())

# Call the main function.
main()
```

The Car class does not have a get_make() method or a get_model() method.

But this code works, since the Automobile (base class of Car) class does have those methods.

Overriding Methods

- Sometimes the derived class wants to have a specialized version of the code for a class method.

```
class Pet(object):
    def __init__(self, name):
        self.__name = name

    def makeSound(self):
        print("Grr...")

class Cat(Pet):
    def makeSound(self):
        print("Meow")

c2 = Cat("Abbey")
print(c2)
c2.makeSound()
```

Overrides the makeSound() method to alter the way the derived class behaves – very common.

Overriding Methods

```
petsList = [Dog("Spot"), Cat("Kitty"), Cat("Abbey"), Dog("Khan"), Pet("Tweety")]
for pet in petsList:
    print(pet.getName(), "says", end=' ')
    pet.makeSound()
```

Output:

```
Spot says Woof, woof!
Kitty says Meow
Abbey says Meow
Khan says Woof, woof!
Tweety says Grr...
```