Lecture 5.3

Object Oriented Concepts

Overriding Basics

- A derived class can use the methods of its base class(es), or it can override them
- The method in the derived class must have exactly the **same signature** as the base class method to override.
- The signature is number and type of arguments and the constantness (const, non- const) of the method.
- The **return type** must match the base class method to override.

Example of same signature

```
class Pet { …
   void makePetJump (int times) { }
};
class Dog { …
   void makePetJump (int times) { }
};
```


Overriding Basics (cont)

- When an object of the base class is used, the base class method is called. When an object of the subclass is used, its version of the method is used if the method is overridden.

Example

```
int main ( ) {
   Animal myAnimal;
   Duck myDuck;

   myAnimal.walk( ); // This will display
                     //“Animal Walking”
   myDuck.walk( ); //This will display “Waddle”
```


Overriding an Overloaded method

- What is an **overloaded** method?
- A method defined more than once with the same name but different signature.
- For instance, suppose the Pet class had defined several speak methods.

```
void speak();
void speak(string s);
void speak(string s, int loudness);
```


Overriding an Overloaded method (cont)

- If the subclass, Cat, defined only

```
void speak();
```

- speak() would be overridden.

  speak(string s) and

  speak(string s, int loudness) would be **hidden**.

Overriding an Overloaded method (cont2)

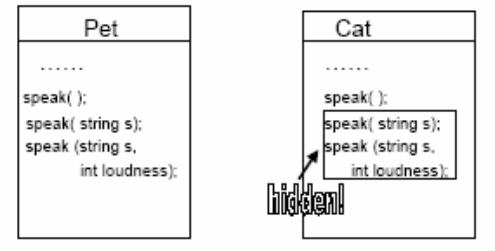
- using Cat fluffly; we could call:

```
fluffy.speak();
```

- But the following would cause compilation errors.

```
fluffy.speak("Hello");
fluffy.speak("Hello", 10);
```

Overriding Overloaded methods (cont3)



Important!

- If you override an overloaded base class method
- either override every one of the overloads, or carefully consider why you are not.

Polymorphism

- Definition: The ability for objects of different classes **related by inheritance** to respond differently to the overriden member call.
- ***Virtual*** keyword enables polymorphism.

Example:

```
class base{
    public:
      void override() //NOTE: this is NOT a virtual method
        {cout<<"override() from base is called"<<endl;}
      virtual void virtual_override()
        {cout<<"virtual_override() from base is called"<<endl;}
};
```


Polymorphism example (cont)

```
class derived: public base{
    public:
      void override()
        {cout<<"override() from derived is called"<<endl;}
      virtual void virtual_override()
        {cout<<"virtual_override() from derived is called"<<endl;}
};
```

Polymorphism example (cont)

```
void  without_virtual_polymorphism(base * b){ b->override();}
void  virtual_polymorphism(base * b){ b->virtual_override();}

int main()
{
   derived d1;
   without_virtual_polymorphism(&d1); //no polymoriphism
   virtual_polymorphism(&d1); //polymorphism
   return 0;
}
```

NOTE: polymorphism works only though pointers or pass by reference, NOT though pass by value.

Brief info about Abstract Classes

- Cannot be instantiated.
- You can't declare an object of that type.
- A class is made abstract by declaring one or more of its virtual function to be "pure". A pure virtual function is one with an initializer of = 0 in its declaration as in

  virtual float speak() = 0;
- Hence a derived class must override pure virtual function. If function implementation of pure virtual function in derived class is not provided than derived class would also become abstract class. (why?)

Example

If Pet is abstract you could NOT declare:

Pet myPet;