Polymorphism

Outline

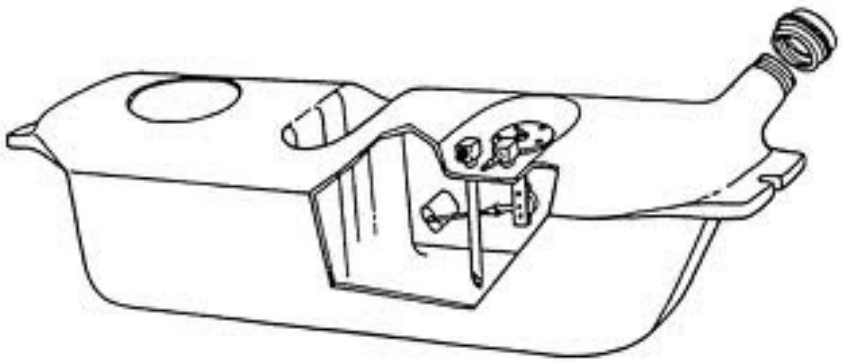
- Motivation
- Review of Object Oriented Design Concepts
- Definition of Polymorphism
- Example
- Summary

Motivation

- Code reusability
  - Add new functionalities to current code with slight or no modification.

Class and Object

- A class is the blueprint from which objects are made.
  - For example, class as blueprint of fuel tank of a car, while objects as the implemented fuel tanks.



Inheritance

- A subclass inherits all the *members* (data members and methods) from its superclass.
- Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

Inheritance II

- An instance method is a method without keyword **static** in the head of its definition.
- An instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass *overrides* the superclass's method.
  - For example, method display from superclass Question is overridden in subclass ChoiceQuestion to display options.
  - Method check_answer from Question is overridden in subclass NumericalQuestion (for approximation comparison).

Inheritance and Polymorphism

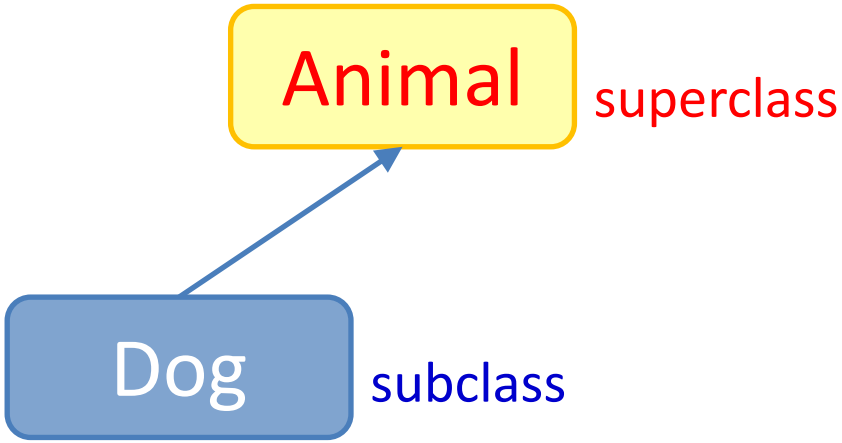
- Inheritance allows us to treat an object as an instance of its own class OR an instance of its base class.
- For example, let Dog be a subclass of Animal, then a dog diesl can be treated as a Dog object or an Animal object.
  - That is, diesl is a dog and can do whatever a dog can do (bark, entertain its master, …).
  - At the same time, diesl is an animal and can do whatever an animal can do (make sound, eat, …).

Inheritance and Polymorphism

- The ability to treat an object as an instance of its own class OR an instance of its base class allows many subclasses derived from the same base class to be treated as if they were one class.
- Methods **defined** for objects of base class can be **used** by objects of sub-classes.

Subclass vs. Superclass

- A class that is derived from another class is called a *subclass*.
- The class from which the subclass is derived is called a *superclass*.

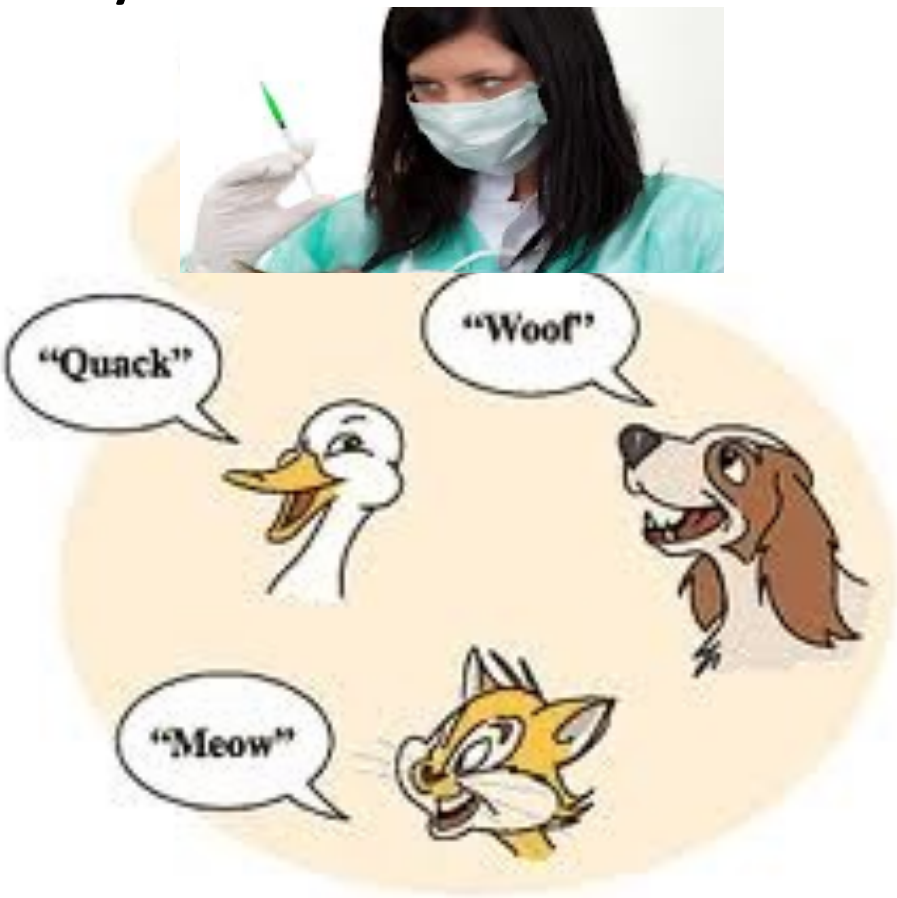


Polymorphism

- For the same message ("Make a sound!"), objects of different subclasses give different responses.

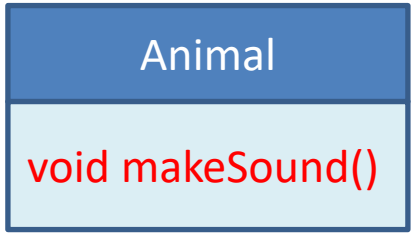
Vet as “Animal talker”

- When he/she gives shots to animals, they cry in their own way.



Class Animal

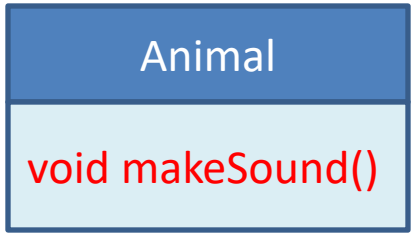
- For class Animal, we only care about the following:
  - Each animal can make a sound.
- What methods should be in Animal class?



Code of Animal.hpp

```
#ifndef ANIMAL_H_  //include guard
#define ANIMAL_H_
class Animal
{
public:
   virtual void makeSound();
       //add keyword virtual when declaring a method in
       //super class that will be overridden in subclass.
};
#endif
```

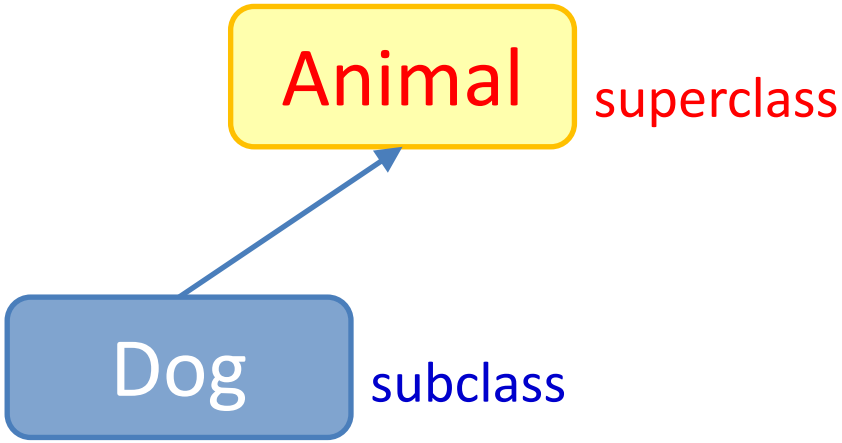
Code of Animal.cpp




```
#include "Animal.hpp"

void Animal::makeSound()
//When define a method,
//no virtual is needed.
{    //A general animal can be any
     //species, we cannot tell what
     //sound it may make. Hence we
     //use empty method body.
}
```

Extend Subclass from Superclass

- Subclasses share some of the same functionality of the superclass.
  - As an animal, dog can make a sound.
- Subclasses can define their own unique behaviors.
  - Dog is the animal that woofs.



Dog.hpp

```
#ifndef DOG_H_
#define DOG_H_
#include "Animal.hpp"
class Dog : public Animal
{
public:
   virtual void makeSound();
      //override makeSound method from super class
      //Animal. Keyword virtual is optional but encouraged.
};
#endif
```

This method overrides makeSound method in superclass Animal.

Code of Class Dog

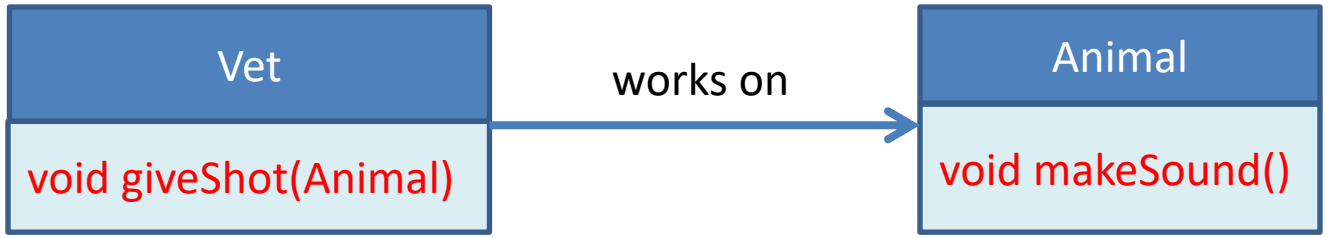
```
//Dog is an animal that woofs.
#include "Dog.hpp"
#include <iostream>
using namespace std;

void Dog::makeSound()
{
    cout << "Dog goes woof." << endl;
}
```

Code of Class Cat

```
//Cat is an animal that
//make sound meows.
//You define code for Cat.hpp and
//Cat.cpp
```

Vet

- A vet gives shots to animals.
  - Warning: Vet is not a subclass of Animal.
- As a result, an animal given a shot makes sound.
- For simplicity, we do not define any data member (like name or certification) for vet.



Vet.hpp

```
#ifndef VET_H_
#define VET_H_
#include "Animal.hpp"

class Vet
{
public:
    void giveShot(Animal* beast);
};
#endif
```

Keys:

- giveShot method applies to Animal, a superclass.
- Must be a pointer to super class to avoid information slicing for subclasses.

Code of Vet.cpp

Key: giveShot method applies to Animal *, a pointer to superclass.

```
#include "Vet.hpp"
#include "Animal.hpp"

void Vet::giveShot(Animal* beast)
{    //When vet gives shot to an
     //animal, it makes a sound.
    beast->makeSound();
}
```

When Vet give shots to animals

```
int main()
{   const int SIZE = 2;
    Animal* animals[SIZE];
    animals[0] = new Dog();
    animals[1] = new Cat();

    Vet v;
    for (int i = 0; i < SIZE; i++)
        v.giveShot(animals[i]);
```

A dog can be used as an animal.

Continue of VetAndAnimals

```
//Do not forget to release dynamic memory.
for (int i = 0; i < SIZE; i++)
{
    delete animals[i];
    animals[i] = 0;
}
}
```

Structure of project vetAnimal

The project contains the following files:

Animal.hpp	Animal.cpp
Cat.hpp	Cat.cpp
Dog.hpp	Dog.cpp
Vet.hpp	Vet.cpp

VetAndAnimals.cpp (contain main method)

makefile

Give a shot to Duck

- Extend a species (say Duck) from Animal.
- Override makeSound method in Class Duck.
- Now giveShot method applies to a duck.
- When a duck is given a shot, it quacks.

Polymorphism increases code reusability

- When we extend subclass Duck from Animal class, we do not need to modify Vet class:
  - giveShot method works for any animal, including a duck, or whatever newer species that the designer of Vet class may not foresee when writing the code.
- With polymorphism, we write code for a superclass that can handle **ALL** future subclasses.