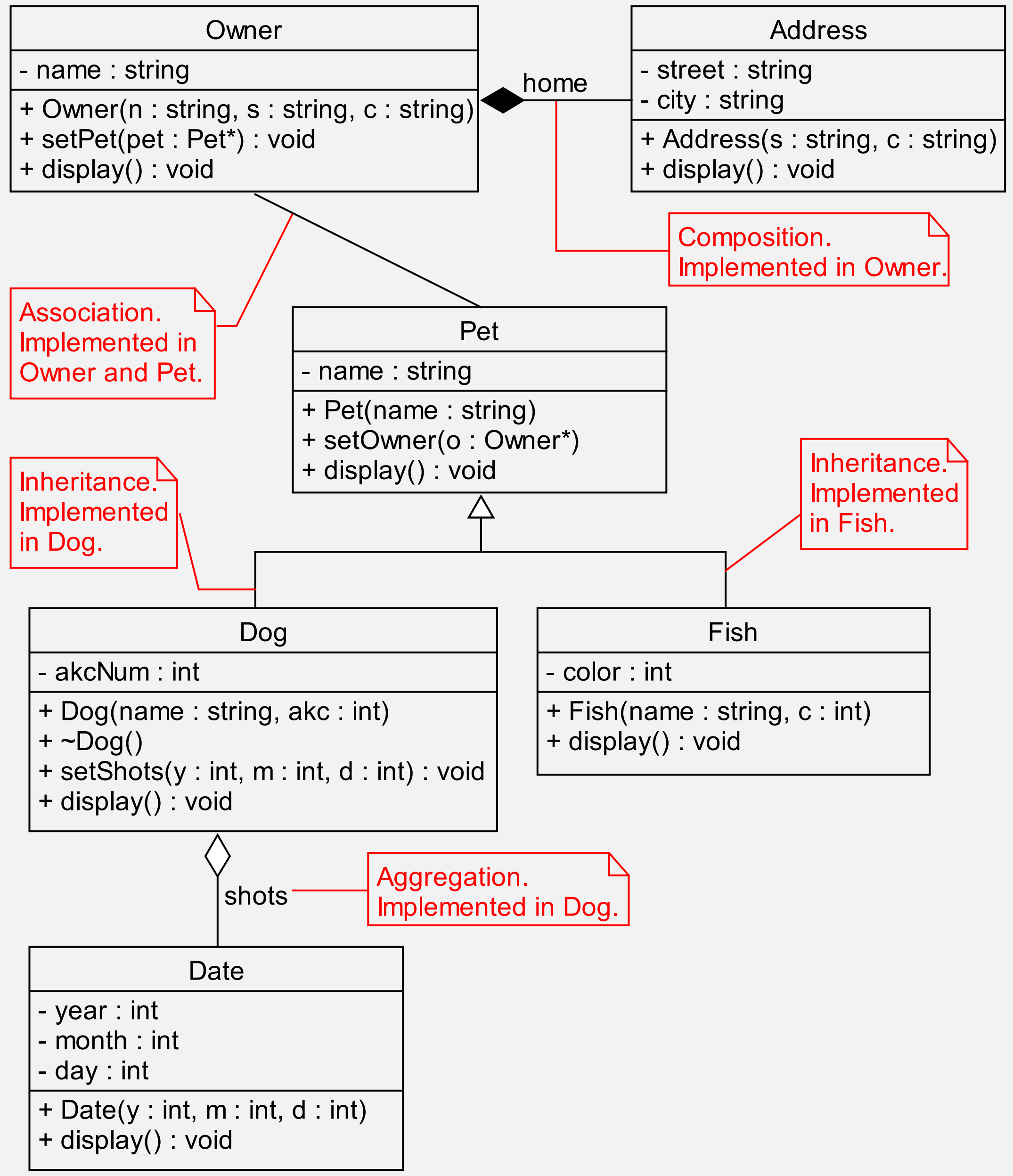
VET 1 EXAMPLE

Multi-class Example:

UML and C++

Delroy A. Brinkerhoff

VET UML


Owner
- name : string
+ Owner(n : string, s : string, c : string)
+ setPet(pet : Pet*) : void
+ display() : void
Address
- street : string
- city : string
+ Address(s : string, c : string)
+ display() : void
home
Composition. Implemented in Owner.
Association. Implemented in Owner and Pet.
Pet
- name : string
+ Pet(name : string)
+ setOwner(o : Owner*)
+ display() : void
Inheritance. Implemented in Dog.
Inheritance. Implemented in Fish.
Dog
- akcNum : int
+ Dog(name : string, akc : int)
+ ~Dog()
+ setShots(y : int, m : int, d : int) : void
+ display() : void
Fish
- color : int
+ Fish(name : string, c : int)
+ display() : void
shots
Aggregation. Implemented in Dog.
Date
- year : int
- month : int
- day : int
+ Date(y : int, m : int, d : int)
+ display() : void

COMMON FEATURES
SAVING SLIDE SPACE

```
#pragma once;

#include "Pet.h"
#include "Date.h"

#include <string>
#include <iostream>
using namespace std;
```

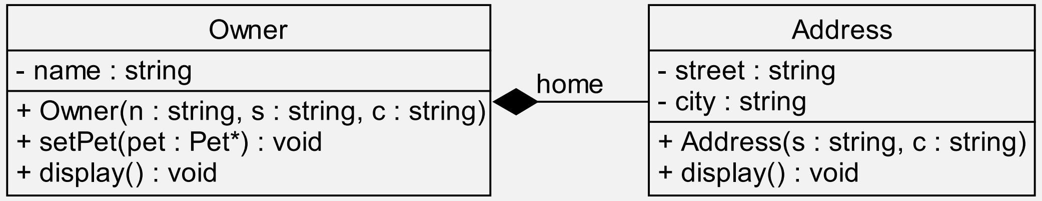
- Each class header file has a "pragma" directive
  - #ifndef / #define / #endif
- Most header files include other project headers
- All header files include two system headers

PART CLASSES

```
class Address
{
    private:
        string street;
        string city;

    public:
        Address(string s, string c)
            : street(s), city(c) {}

        void display()
        {
            cout << "Street: " << street <<
                " City: " << city << endl;
        }
};
```

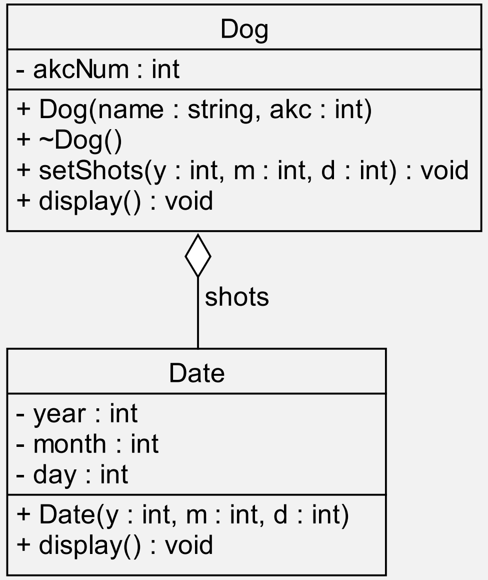
```
class Date
{
    private:
        int         year;
        int         month;
        int         day;

    public:
        Address(int y, int m, int d)
            : year(m), month(m), day(d) {}

        void display()
        {
            cout << year << "/" << month <<
                "/" << day << endl;
        }
};
```

THE OWNER / ADDRESS RELATIONSHIP COMPOSITION


Owner
- name : string
+ Owner(n : string, s : string, c : string)
+ setPet(pet : Pet*) : void
+ display() : void
home
Address
- street : string
- city : string
+ Address(s : string, c : string)
+ display() : void

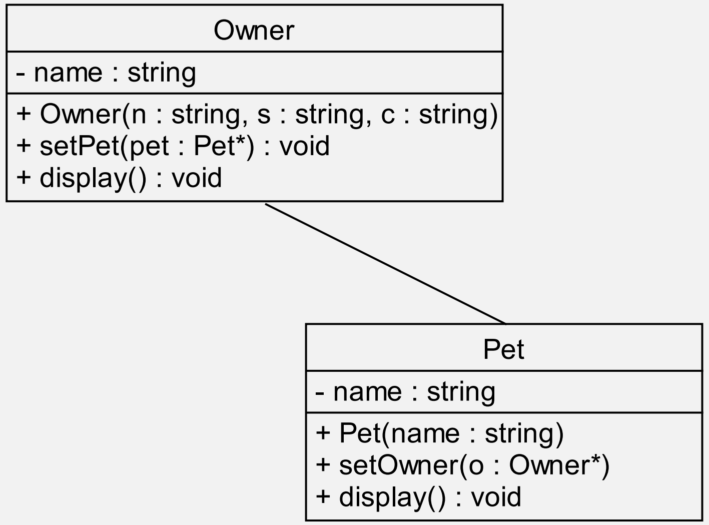
```
class Owner
{
    private:
        string   name;
        Address  home;

    public:
        Owner(string n, string s, string c)
            : name(n), home(s, c) {}

        void display()
        {
            cout << "Owner: " << name << endl;
            home.display();
        }
};
```

THE DOG / DATE RELATIONSHIP AGGREGATION


Dog
- akcNum : int
+ Dog(name : string, akc : int)
+ ~Dog()
+ setShots(y : int, m : int, d : int) : void
+ display() : void
shots
Date
- year : int
- month : int
- day : int
+ Date(y : int, m : int, d : int)
+ display() : void

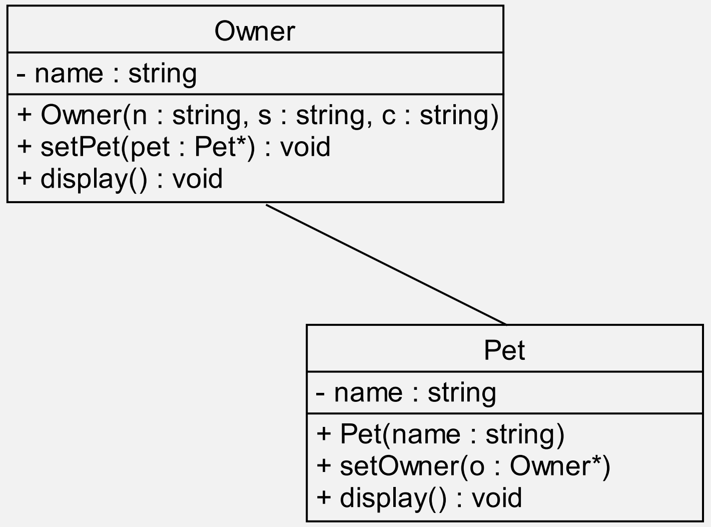
```
class Dog : public Pet
{
    private:
        Date*    shots = nullptr;

    public:
         ~Dog() { delete shots; }

        void setShots(int y, int m, int d)
        {
            if (shots != nullptr)
                delete shots;
            shots = new Date(y, m, d);
        }

        void display()
        {
            cout << "AKC#: " << akcNum << endl;
            if (shots != nullptr)
                shots->display();
        }
};
```

THE OWNER / PET RELATIONSHIP
OWNER SIDE OF ASSOCIATION



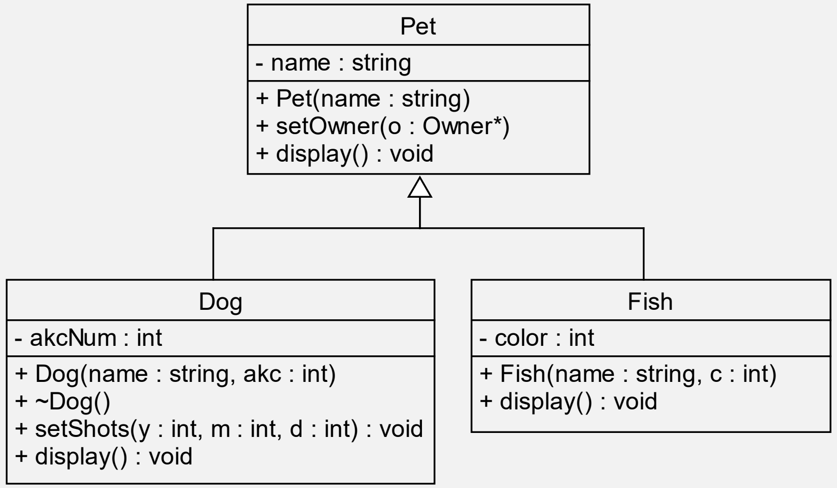
```
class Pet;

class Owner
{
    private:
        Pet*     myPet = nullptr;

    public:
        Owner(string n, string s, string c)
            : name(n), home(s, c) {}

        void setPet(Pet* p) { myPet = p; }

        void display()
        {
            cout << "Owner: " << name << endl;
            if (myPet != nullptr)
                myPet->display();
        }
};
```

THE OWNER / PET RELATIONSHIP
PET SIDE OF ASSOCIATION



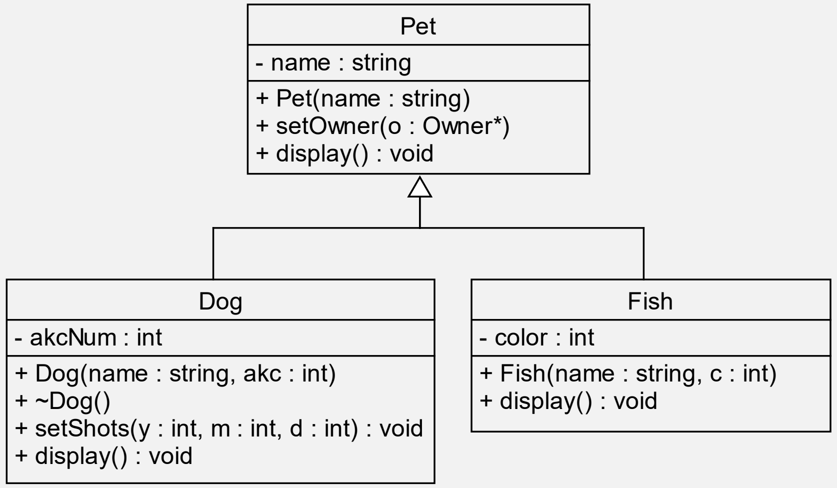
```
class Owner;

class Pet
{
    private:
        string   name;
        Owner*   owner = nullptr;

    public:
        Pet(string n) : name(n) {}

        void setOwner(Owner o) { owner = o; }

        void display()
         {
            cout << "Pet: " << name << endl;
        }
};
```

INHERITANCE (1)
THE PET SUPERCLASS



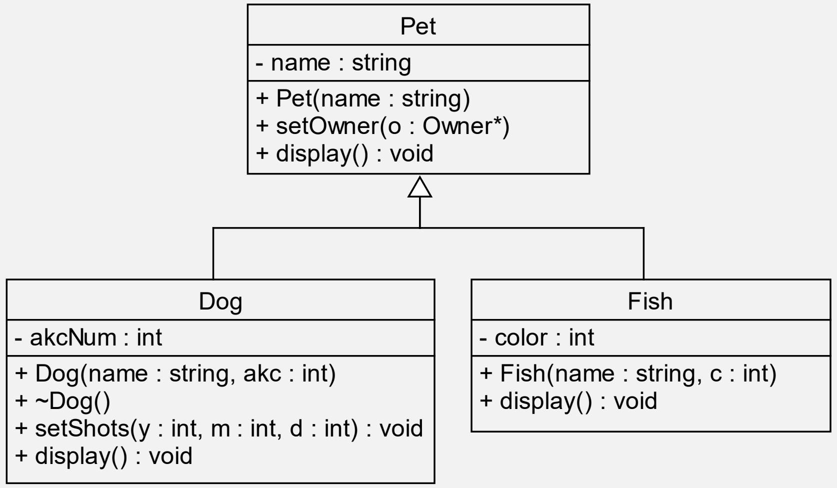
```
class Pet
{
    private:
        string   name;

    public:
        Pet(string n) : name(n) {}

        void display()
         {
            cout << "Pet: " << name << endl;
        }
};
```

INHERITANCE (2)
THE DOG SUBCLASS



```
class Dog : public Pet
{
    private:
        int       akcNum;

    public:
        Dog(string name, int akc)
            : Pet(name), akcNum(akc) {}

        void display()
        {
            Pet::display();
            cout << "AKC#: " << akcNum << endl;
            if (shots != nullptr)
                shots->display();
        }
};
```

INHERITANCE (3)
THE FISH SUBCLASS



```
class Fish : public Pet
{
    private:
        int      color;

    public:
        Fish(string name, int c)
            : Pet(name), color(c) {}

        void display()
        {
            Pet::display();
            cout << "Fish color: " <<
                color << endl;
        }
};
```

BUILDING THE OBJECTS
MAIN

```
#include "Owner.h"
#include "Dog.h"
using namespace std;

int main()
{
        Dog      myPet("Dogbert", 300);
        Owner    theOwner("Dilbert", "115 Elm St.", "Ogden");

        myPet.setShots(2000, 9, 1);

        myPet.setOwner(&theOwner);
        theOwner.setPet(&myPet);

        theOwner.display();

        return 0;
}
```