INHERITANCE

INITIALIZATION LIST

- comp3_init_list.cpp
- init_list_pointer.cpp (uses pointers but no shared pointer)
- Init_shared_ptr.cpp (uses shared pointers)
- Why do you thing it may be beneficial to do this?

COMPOSITION VS INHERITANCE

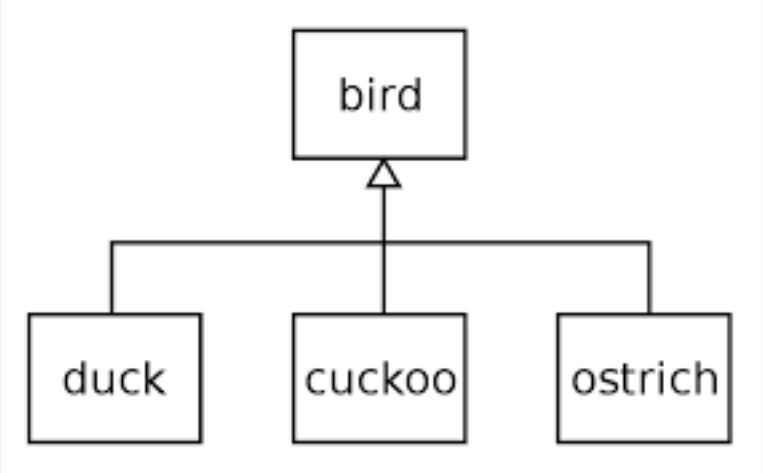
- Aggregation and Composition models a “has-a” relationship
- Now we will talk about inheritance which models the “is-a” relationship between classes

INHERITANCE

- Inheritance is a way of creating a new class by starting with an existing class and adding new members
- The new class can replace or extend the functionality of the existing class

IS-A RELATIONSHIP

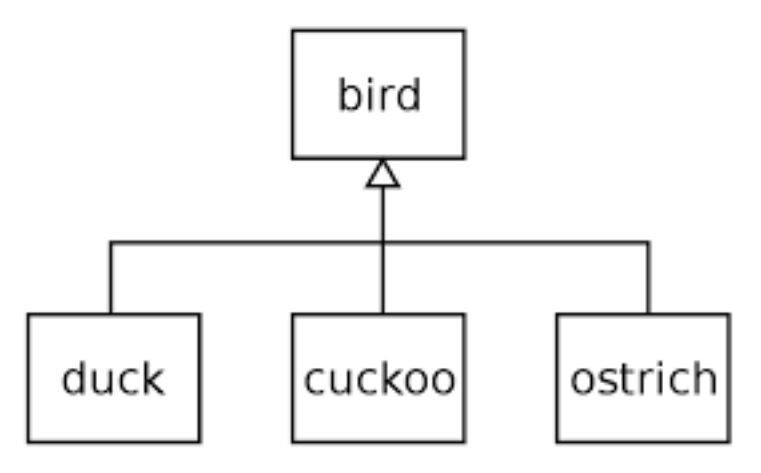
- When one object is a specialized version of another object, there is an *is-a* relationship between the two objects
  - a poodle “is-a” dog
  - a student “is-a” person
  - an instructor ”is-a” person
  - a car ”is-a” vehicle
  - a rectangle “is-a” shape



- Each of the above are specialized objects of a more general object
  - Ex. There are many dogs with basic characteristics but the poodle is a dog with special characteristics specific to poodles
  - This is the same with the other examples

INHERITANCE TERMINOLOGY

- Base class
  - The existing class – dog, vehicle, person, shape
  - A.k.a. – parent class, superclass
  - More broad in nature
- Derived class
  - The new class – poodle, car, stude
  rectangle
  - A.k.a. – child class, subclass
  - More specialized



Base Class

Derived Classes

INHERITANCE SYNTAX AND NOTATION

```
class Base
{
   //some stuff
};
class Derived : public Base
{
   //some stuff
};
```

This indicates Derived inherits from Base

INHERITANCE OF MEMBERS

```
class Parent
{
  int a;
  void basef();
};
class Child : public Parent
{
  int c;
  void derivedf();
};
```

Base class access specification. This determines the type of access for the inherited

Parent members

```
int a;
void basef();
```

Child has parent members and it's own members

```
int a; void basef();
//this makes the child more specialized
int c; void derivedf();
```

INHERITANCE AND CONSTRUCTORS

- When and how do the constructors get called with respect to inheritance
  - simple_inherit.cpp inheritance.h

PROTECTED MEMBERS

- **protected member**: A class member labeled **`protected`** is accessible to member functions of derived classes as well as to member functions of the same class
- Like **`private`**, except accessible to members functions of derived classes
- Example:
  - driver_inherit1.cpp, inheritance1.h, inheritance1.cpp

CLASS ACCESS SPECIFIERS

- **Base class access specification** determines how **`private`**, **`protected`**, and **`public`** members of base class can be accessed by derived classes

Access specifier

- Ex.

```
class Faculty : public Person
{
    some stuff
};
```

BASE CLASS ACCESS SPECIFICATION

Base class members

How base class members appear in the derived class.

private: x protected: y public: z	→	Private	private: x private: y private: z
private: x protected: y public: z	→	Protected	private: x protected: y protected: z
private: x protected: y public: z	→	Public	private: x protected: y public: z

INHERITANCE AND OVERRIDING A FUNCTION

- A derived class can override a member function of its base class by defining a derived class member function with the same name and parameter list.

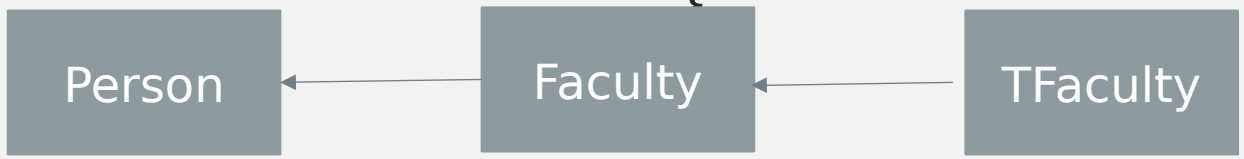
- overriding.cpp

OVERRIDING BASE CLASS FUNCTION

```
class Person
{
   protected:
      string name;
   public:
      Person ()
     Person(string
name)
    string getName()
const
}
```

```
class Faculty: public
Person
{
    private:
      Discipline
department;
    public:
      Faculty(string n,
disciplined):Person(n)
      { department = d;}
}
```

```
class Tfaculty : public Faculty
{
   private:
      string title;
   public:
      Tfaculty(string n, Discipine d,
string title) : Faculty(n, d) { }
      void setTitle(string t)
      string getName() const { }
}
```



driver_inherit3.cpp
inheritance3.h

This example shows overriding and the order constructors are called.

CONSTRUCTORS, DESTRUCTORS, AND INHERITANCE

- When an object of a derived class is being instantiated, the base class constructor is called before the derived class constructor. When the object is destroyed, the derived class destructor is called before the base class destructor.
- BaseDemo.cpp