Outline

Inheritance
- "is-a" relationship
- Superclass, subclass
- Access specifiers

Construction
- Order
- Initializer list review
- Destruction

3 Uses:

Reuse

Extension

Specialization
Virtual functions

# Inheritance

- Child class & parent class
- Different method behavior dep. on class
- Extending classes

parent Class

```
class Animal {

};
```

child Class

access specifier

```
class Dog : public Animal {
public:
    void bark();
};
```

parent / child
super / sub
base / derived

"is-a" relationship

"A Dog is an Animal"

"always"

"A square is a Rectangle"

derived — base

✗ `class Rectangle : public Square`

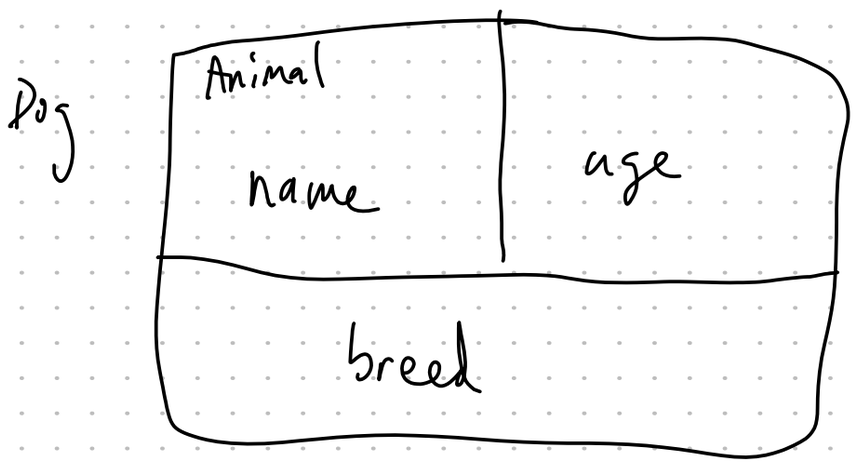
- Derived class automatically gets all of the base class's member variables and methods
- Derived class must appear after base class

| Access specifier | private | protected | public |
|---|---|---|---|
| Who can use methods / mem. vars | ONLY this class | this class and all classes deriving from it. | everyone |

Inheritance captures "is-a" relationship

- Reuse (not a crucial selling point, we can accomplish the same w/ composition)

```
class Dog {
    Animal inner; // composition
}
```




- Extension
  - Adding member vars. or methods to an existing class.
- Specialization
  - Changing the base classes method's behavior

Recap

- Construction order: Base then Derived
- Destruction order: Derived the Base
- Similar to if you had a Base object as a member in Derived (composition)

# Virtual Functions

## Pure Virtual Functions: (PVF)

```
void make_noise() = 0;
```

- When a class has a PVF it is called "abstract"
  - We cannot create instances of abstract classes
  - If we want to make instances of Dog, we must impl the PVF
    Otherwise, Dog is also abstract