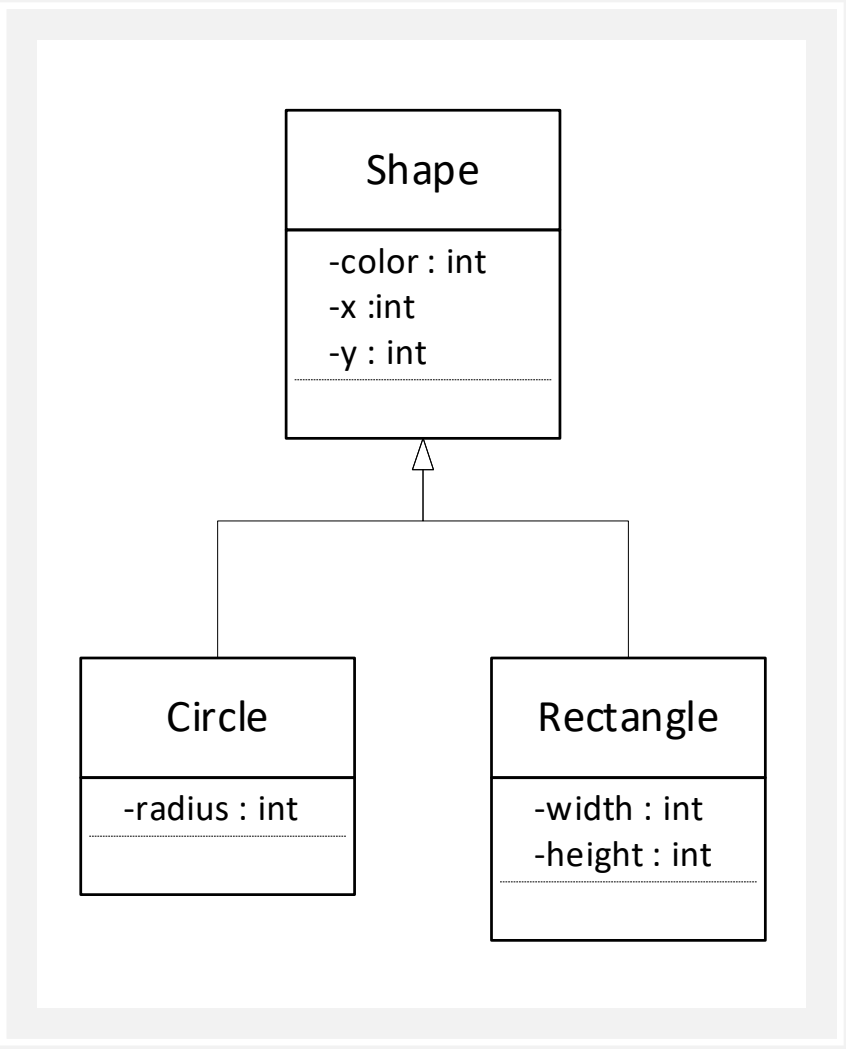
USING INHERITANCE

Accessing inherited features

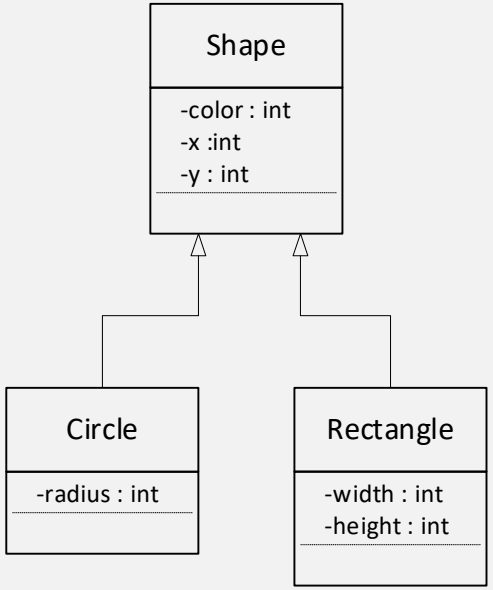
Delroy A. Brinkerhoff

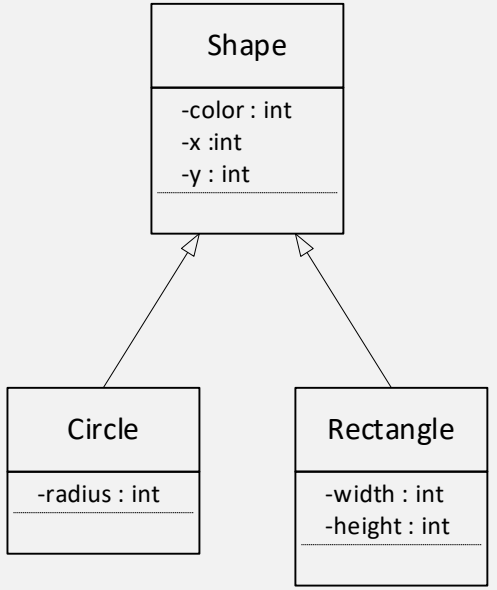
INHERITANCE EXAMPLE

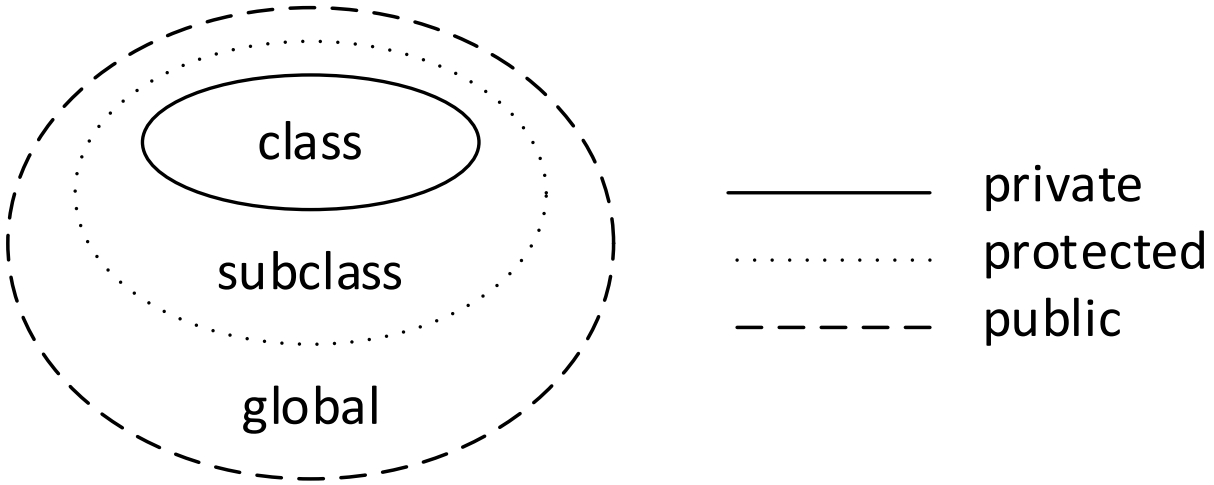
- A Circle is a Shape
  - inherits color, and an (x, y) location
  - has a radius that a Shape and a Rectangle don't
- A Rectangle is a Shape
  - inherits color, and an (x, y) location
  - has a width and height that a Shape and Circle don't
- Circle and Rectangle can't *directly* access color or x and y
- radius, width, and height don't apply to all subclasses



ALTERNATE INHERITANCE STYLES





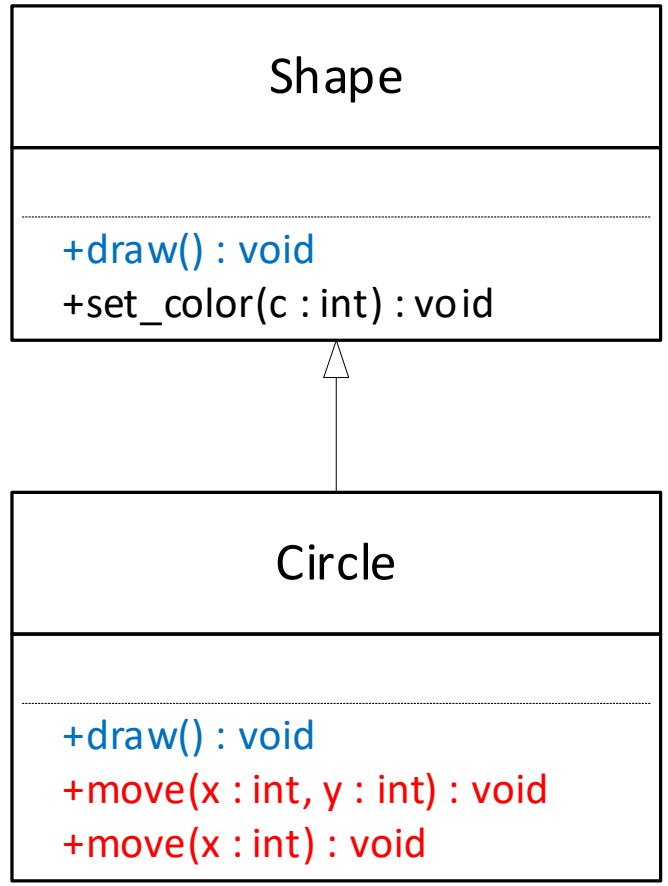


PROTECTED

- "protected" keyword is only meaningful with inheritance
- Subclass can directly access protected superclass features
- protected is an intermediate level of accessibility
  - less restrictive than "private"
  - more restrictive than "public"

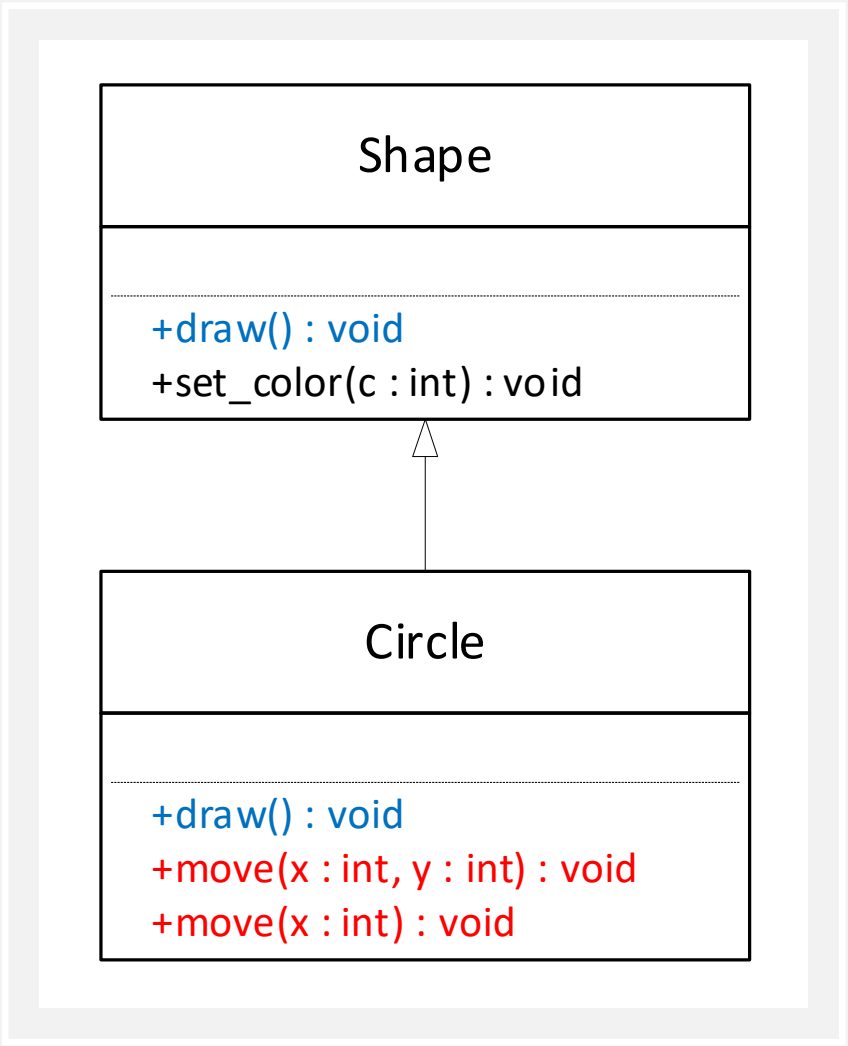
OVERLOADED FUNCTIONS

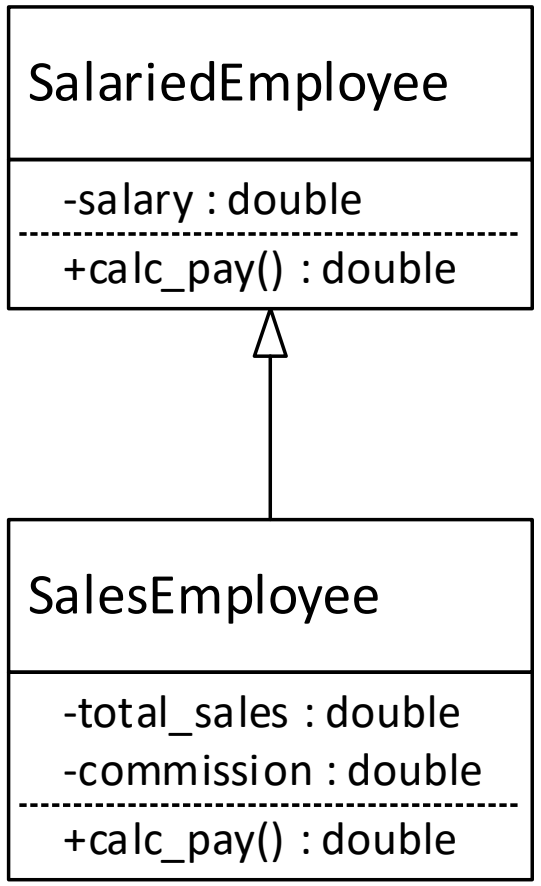
- Have the same name
- Must have unique argument lists
- May have different return types
- Are defined in the same scope
  - Member and non-member functions
  - Functions defined in a class or namespace have the same scope
  - Functions defined in different classes have different scope
- Illustrated in red



OVERRIDDEN FUNCTIONS

- Have the same name
- Are defined in two or more classes related by inheritance
  - Does not apply to non-member functions
- Must have identical argument lists and return type
- Subclass functions must have equal or greater accessibility
  - superclass: protected; subclass: protected or public
  - superclass: public; subclass: public
- Illustrated in blue





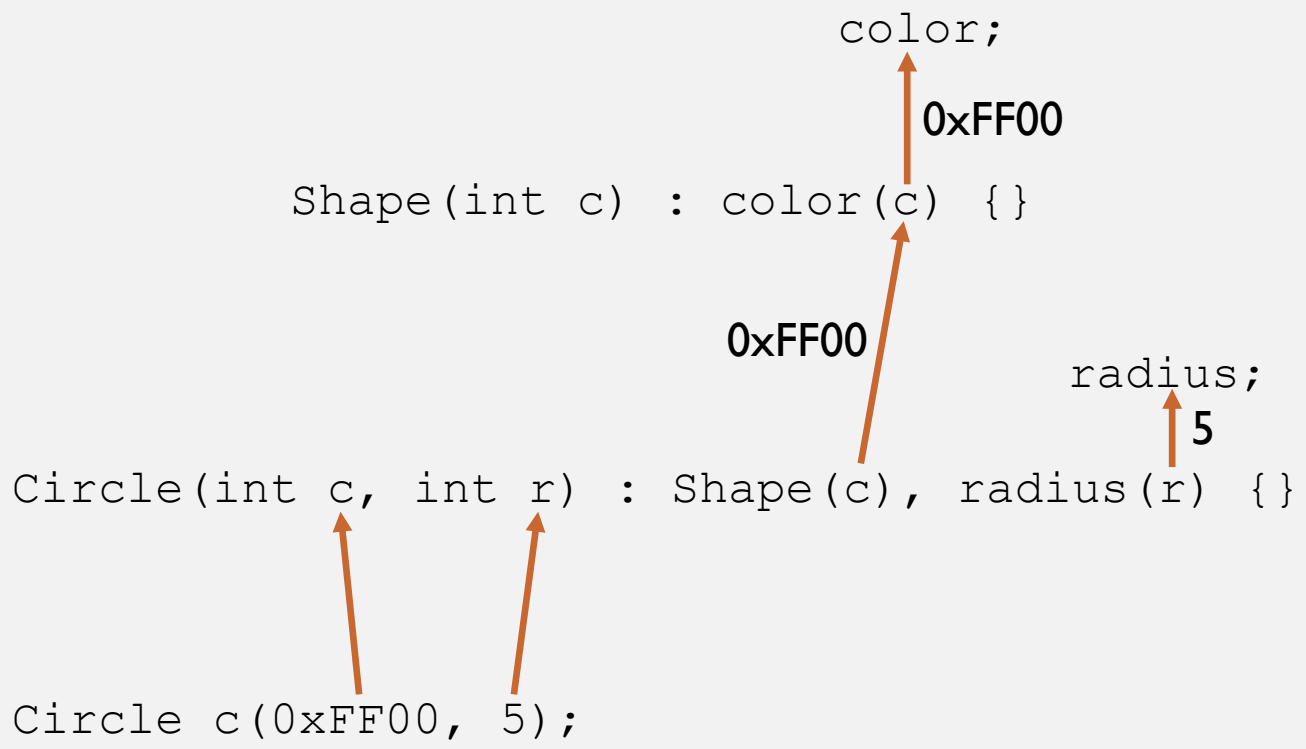
ACCESSING INHERITED PRIVATE DATA: USING A PUBLIC INTERFACE

```
SalariedEmployee
{
    private:
        double salary;

    public:
        double calc_pay()
        {
            return salary / 24;
        }
};
```

CHAINING OVERRIDDEN FUNCTIONS

```
SalesEmployee : public SalariedEmployee
{
    private:
        double total_sales;
        double commission;

    private:
        double calc_pay()
        {
            return SalariedEmployee::calc_pay() + total_sales * commission;
        }
};
```

CONSTRUCTING OBJECTS



CALLING AN OVERRIDDEN FUNCTION

- `c.draw();`
  - Calls the Circle draw function
- `c.Shape::draw();`
  - Calls the Shape draw function
- When a subclass object calls an overridden function, it calls the subclass function unless the superclass function is specifically selected

```
void draw()
{
        . . . .;
        Shape::draw();
        . . . .;
}
```

- An overriding function in a subclass may call the superclass that it overrides, and the scope resolution operator is necessary

CALLING AN INHERITED FUNCTION

- `c.set_color(0xFF);`
  - Circle does not override set_color
  - Calls the Shape set_color function
- When a subclass object calls an inherited function, it calls the superclass function, and the call is indistinguishable from a call made to a subclass member function