Object-Oriented Inheritance & Polymorphism (Chapter 15)

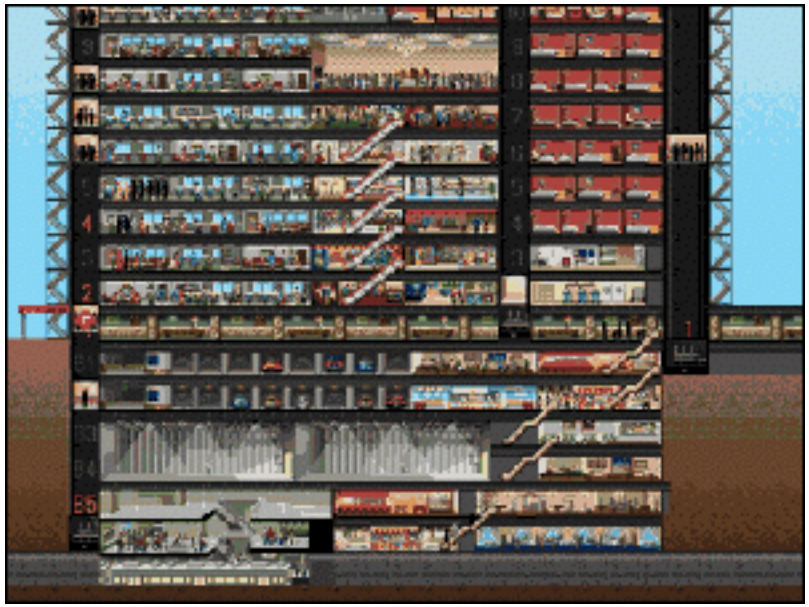
Topics

Understand the two important types of **object relationships** in **Object-Oriented Design**:

- **Object Composition**
- **Object Inheritance**

Object **Composition**

Imagine you are creating a simulation of a City?



To design a virtual **tower** we need to think about what **component** it has

Composition or "*HAS-A*" relationships

Tower

Elevators

Floors

Rooms

Windows

Doors

Translating Object Composition into C++ Code

Composition relationships translate to class **properties**

```
class Tower
{
  private:
       vector<Elevator> Elevators;
       vector<Floor> Floors;
  public:
       …
};
class Floor
{
   public:
        vector<Room> Rooms;
      ...
};
class Room
{};
```

Building objects using "**composition**" is good, but not sufficient …

- What about simulating **virtual people** working in our **virtual City**?
- There are **many different types** of "people".

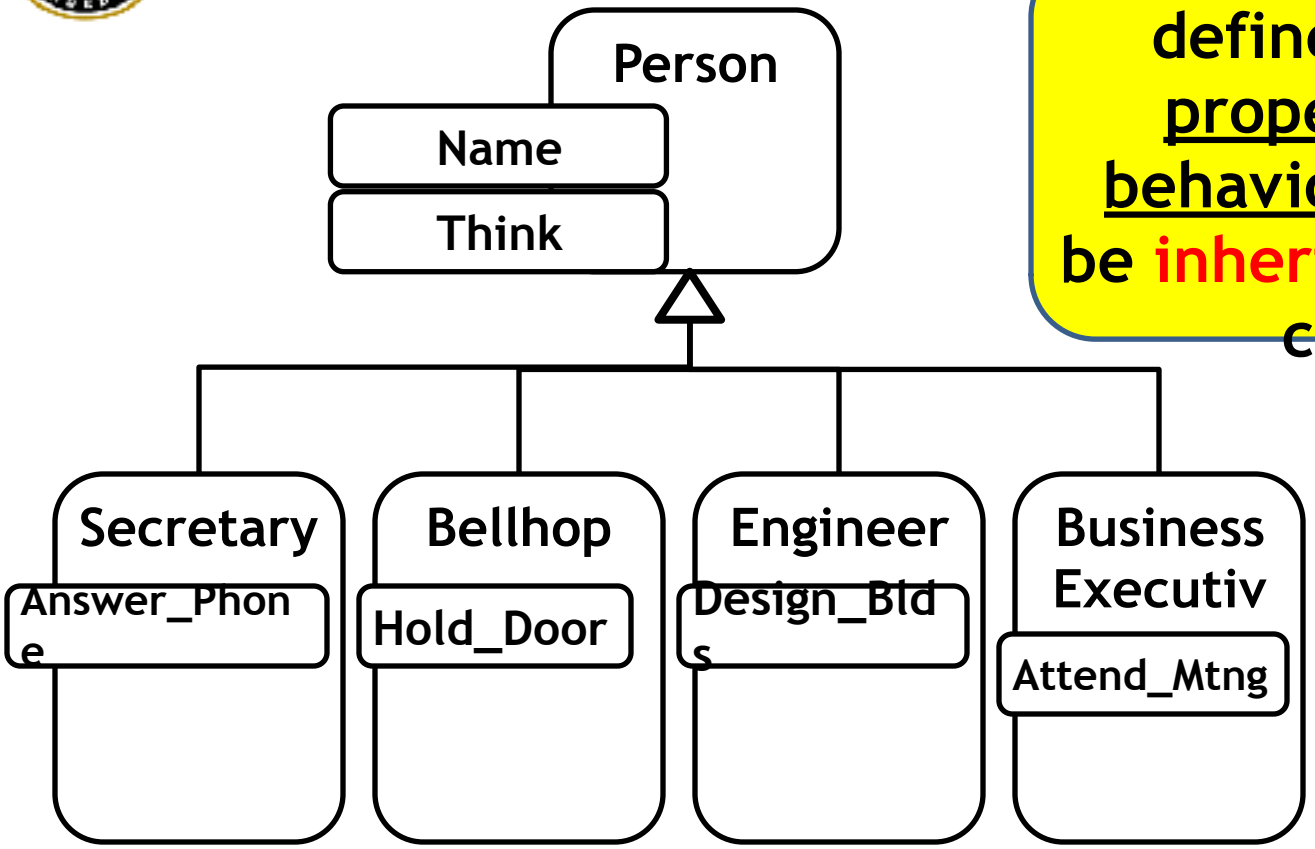
`secretary`

`Business Executive`

`Engineer`

Some properties are common, some are not.

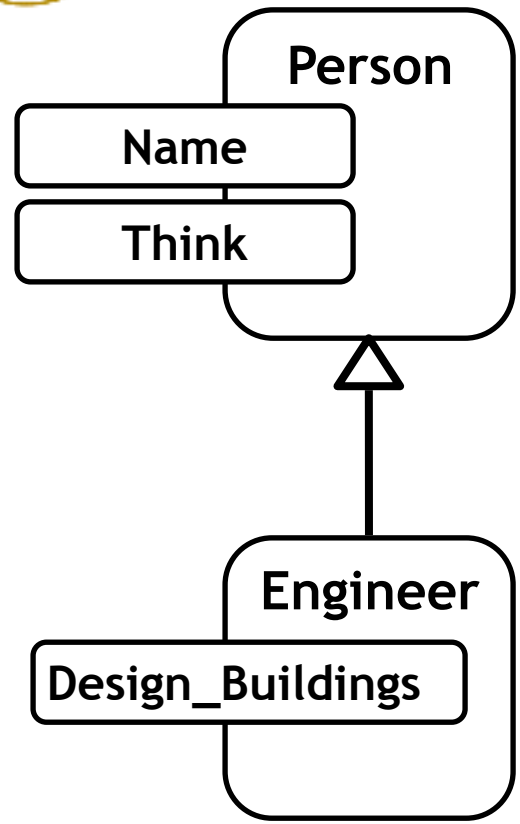
Object Inheritance



The "base class" define common properties and behaviors that can be inherited by other classes.

The "sub class" can "inherit" and "extend" the base class, adding methods that are specific to their own needs.

Inheritance - "IS-A" Relationships

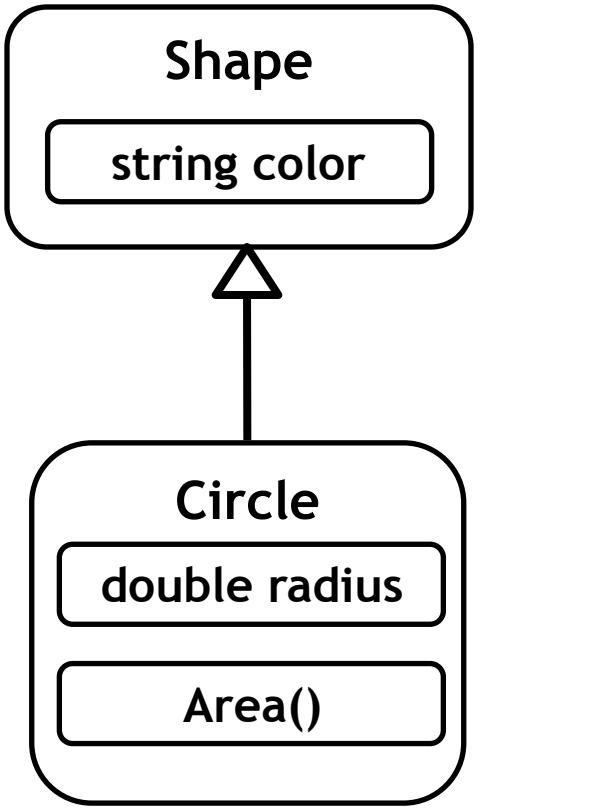


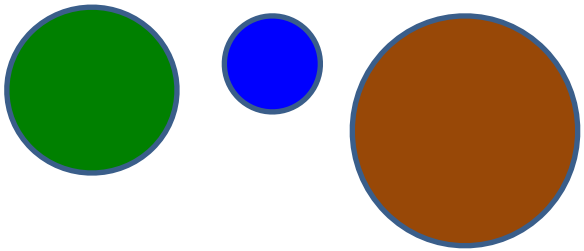
Engineer **IS-A** type of Person!

Person class is called:
base class,
parent class,
superclass

Engineer class is called:
derived class,
child class,
subclass

Another inheritance example





The Circle **Is-A** Shape!
The Circle class **inherits** and **extends** the Shape class

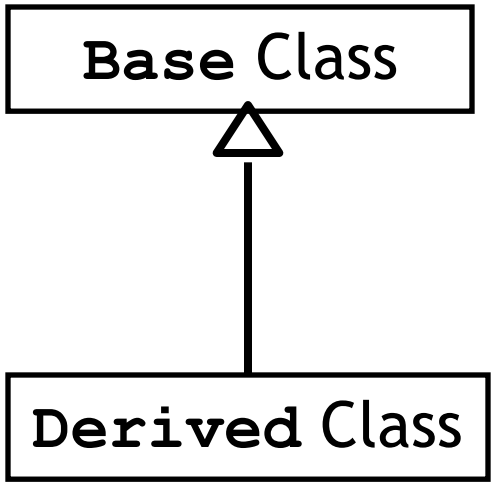
The Circle class is **derived** from the Shape **base** class

Circle class is a **child** of the **parent** Shape class.

Translating inheritance into C++

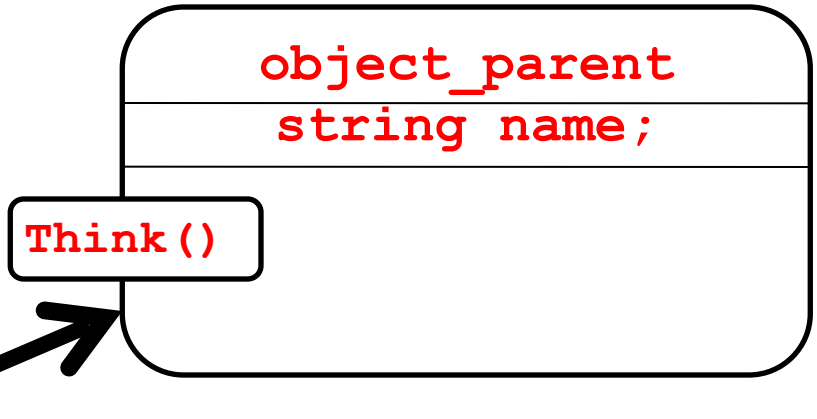
```
// Base class
class Base
{
};

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



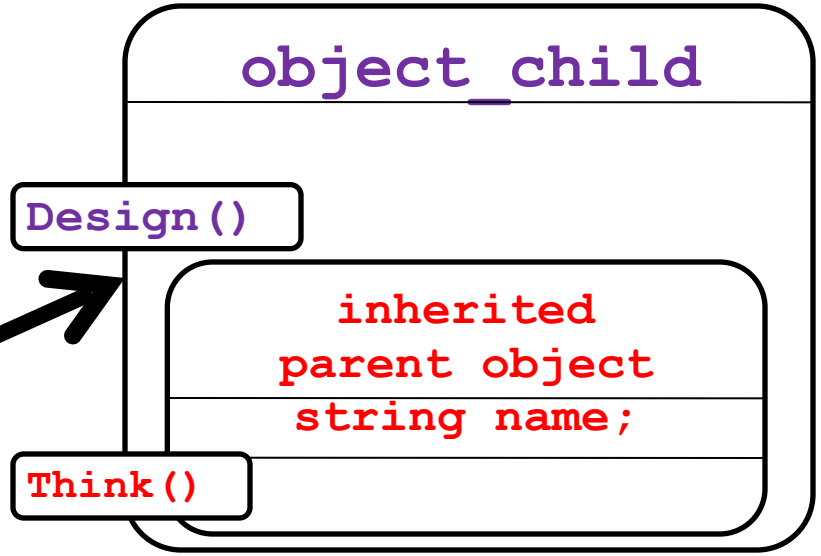
Assume we have a Person class. We can make an object of type Person

```
class Person
{ private:
   string name;
   public:
   void Think();
};
Person object_parent;
```



If we define an **Engineer** class to **inherit** from the **Person** class, an Engineer object will also include a Person object!

```
class Engineer : public Person
{
   public:
   void Design();
};
Engineer object_child;
```



Check it out

- Clone from https://github.com/ptucker/PersonsAndEngineers.git

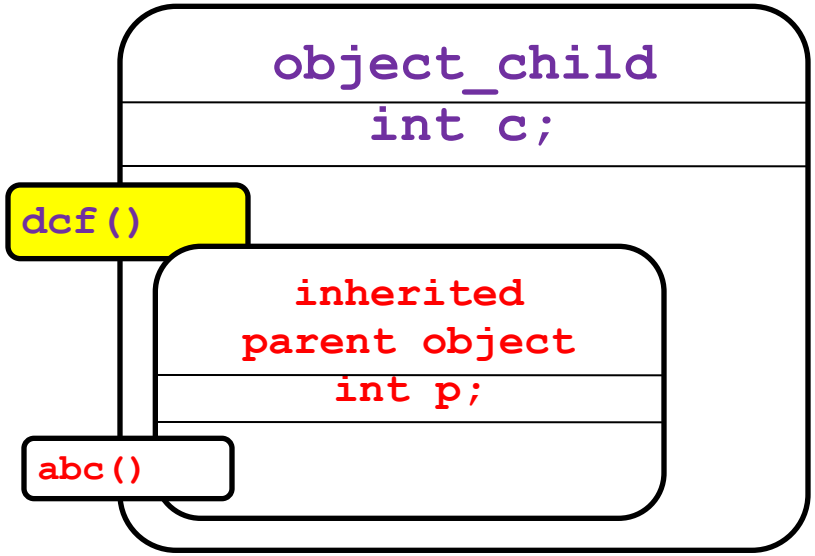
- Compile and try it out
- Add the other three person sub-types
  - Secretary
  - Bellhop
  - Business executive

- Hang onto this work for our next session

Access specifiers in the **base class** control the accessibility of properties/ methods in the **derived class**:

1) `public`
   a) can be accessed **by** derived class
   b) can be accessed **through** derived class
2) `private`
   a) cannot be accessed **by** derived class,
   b) cannot be accessed **through** derived class
3) `protected`
   a) can be accessed **by** derived class
   b) cannot be accessed **through** derived class

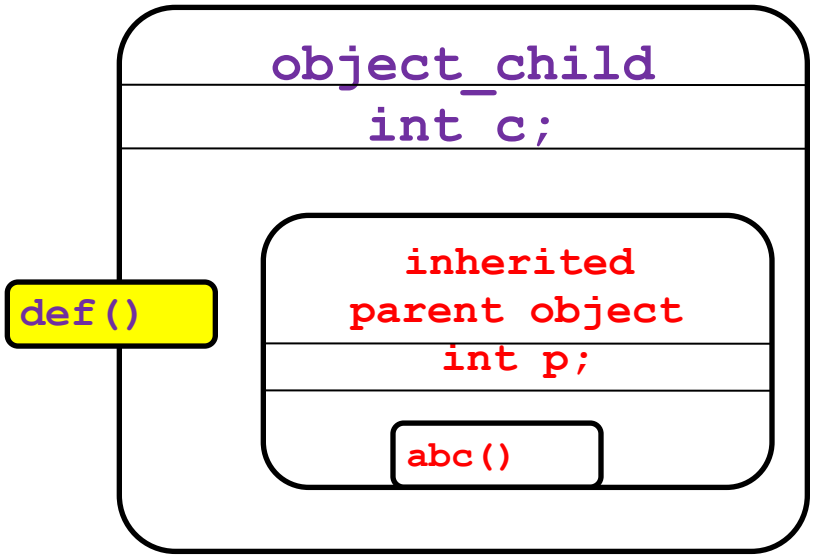
Public parent::abc()



In this example **abc()** IS

1. accessible by the child object
2. accessible through child object

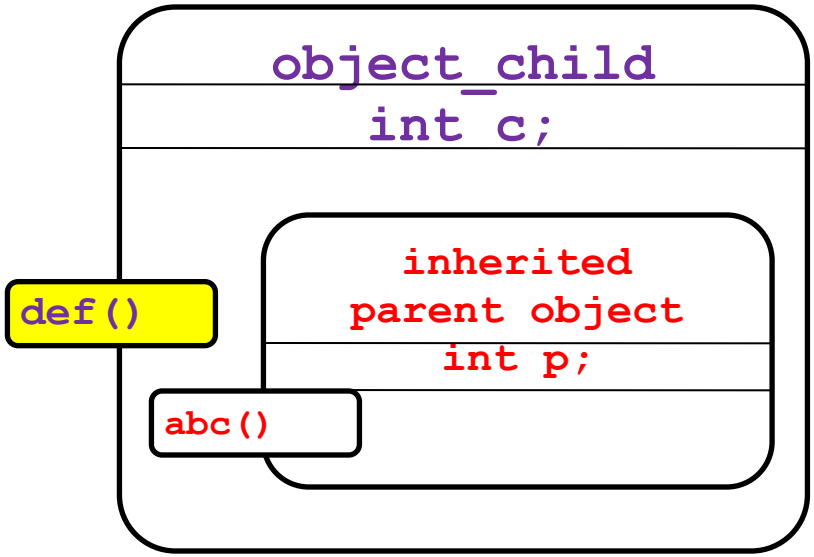
Private parent::abc()



In this example **abc()** is

1. not accessible by the child object.
2. not accessible through the child object

Protected parent::abc()



In this example **abc()** is

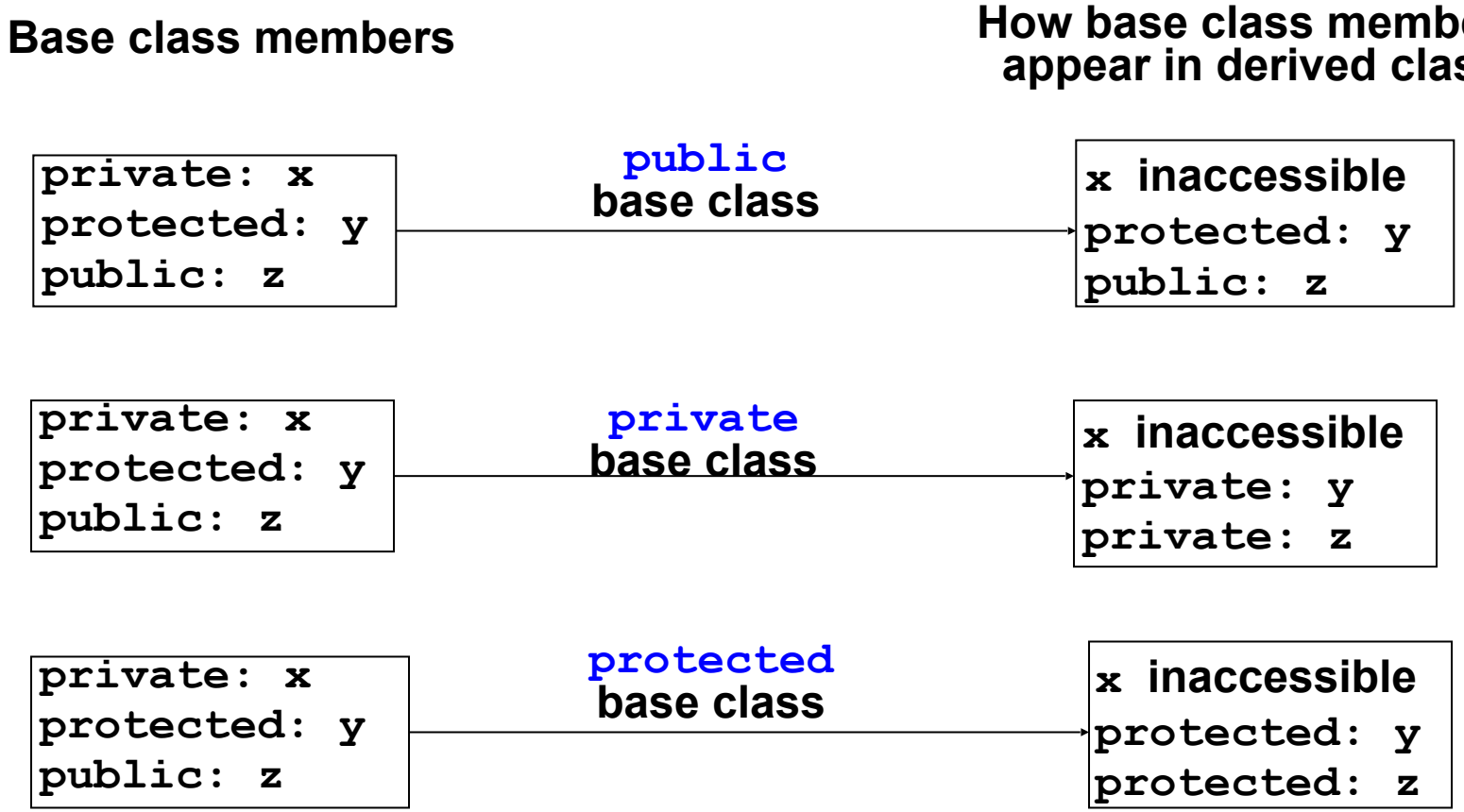
1. accessible by the child object.
2. not accessible through the child object.

We can also control access of the base class members through the derived class by the **type of inheritance**

```
class Child : public Parent
{
};
```

base class inheritance can be one of public, protected or private

Effect of **type of Inheritance**



Let’s again check it out

- Suppose Engineer needed access to _name
  - Let’s try changing the access level of _name

You can call the **base class constructor** in the **derived class constructor**.

```
class Person {
    private:
        string _name
    public:
        Person(string name){
            this->_name = name;
        };
};

class Engineer : public Person {
    public:
    Engineer(string name): Person(name)
    {}
};
```

Inheritance + Composition