Inheritance

Creating a Class Hierarchy

CIS 15 : Spring 2007

Functionalia

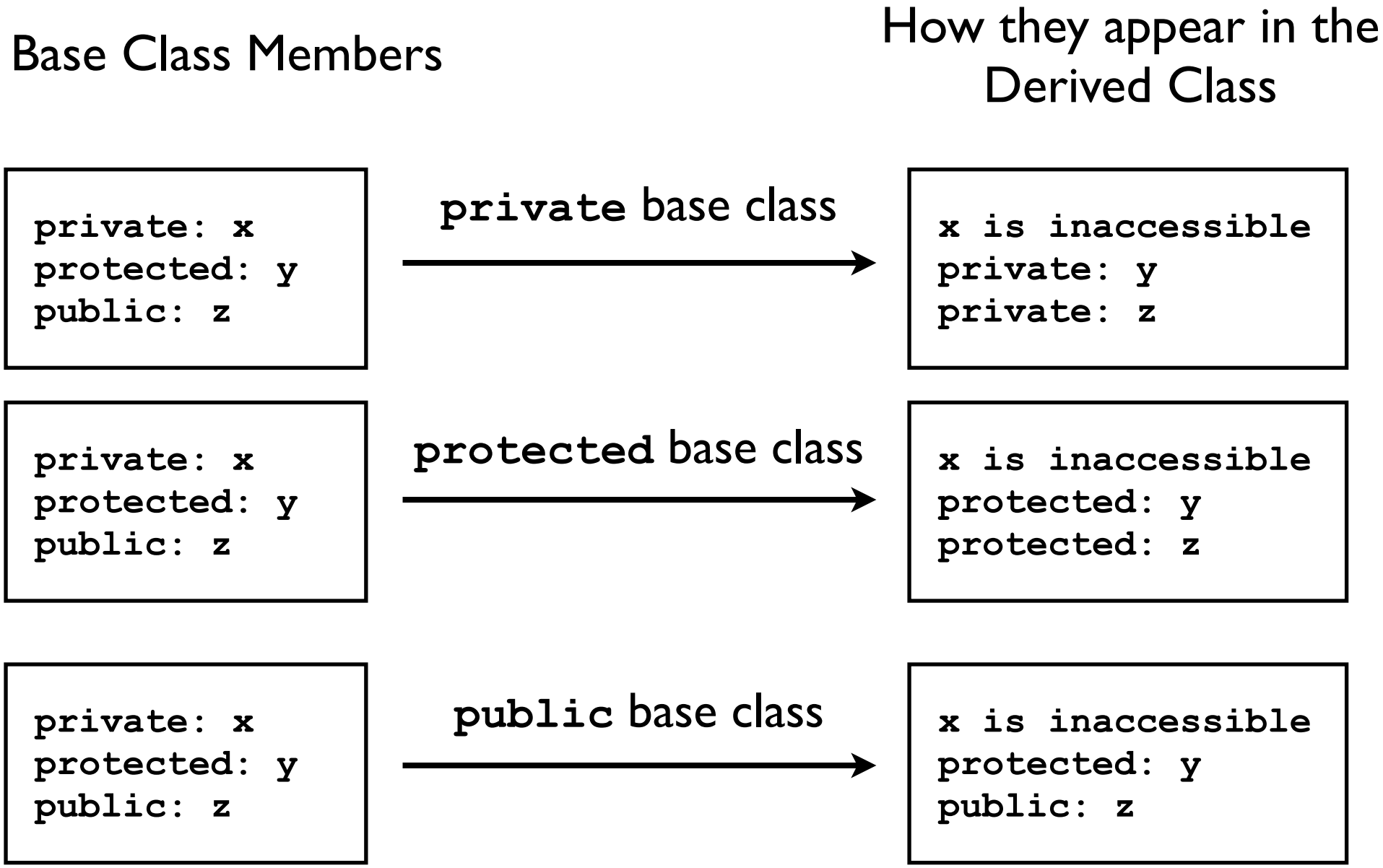
Midterm 2 NEXT Thursday

Review on Monday

Today:

- Quiz
- More Inheritance
- UML
- Multi-files

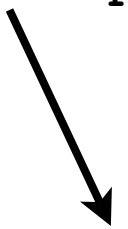
Base Class Access Specifiers



Base Class Access Specifiers

If no Access Specifier is given, the it is ***private*** by default.

```
class Circle : Shape
```



by default

```
class Circle : private Shape
```

Constructors and Destructors in Base and Derived Classes

```
class Base {
   public:
      Base() { cout << "Hello! Base Constructor." << endl; }
      ~Base() { cout << "Bye Bye! Base Destructor." << endl; }
};
class Derived : public Base {
   public:
      Derived() { cout << "Hello! Derived Constructor." << endl; }
      ~Derived() { cout << "Bye Bye! Derived Destructor." << endl; }
};
int main() {
   cout << "Create Derived Object" << endl;
   Derived d;
   cout << "Quit the program" << endl;
   return 0;
}
```

Constructors and Destructors in Base and Derived Classes

Base class's constructor is called ***before*** the derived class's constructor.

Destructors are called in ***reverse*** order, with the derived class's destructor being called first.

```
Create Derived Object

Hello! Base Constructor.

Hello! Derived Constructor.

Quit the program

Bye Bye! Derived Destructor.

Bye Bye! Base Destructor.
```

Constructor/Destructors with Arguments

```
class Rectangle {
   private:
      double width;
      double length;
   public:
      Rectangle() { width = 0; length = 0; }
      Rectangle(double w, double len)
      { width = w;
        length = len; }
   };
```

Constructor/Destructors with Arguments

```
class Rectangle {
    private:
        double width;
        double length;
    public:
        Rectangle() { width = 0; length = 0; }
        Rectangle(double w, double len)
        { width = w;
          length = len; }
    };

class Cube : public Rectangle {
   private:
      double height;
      double volume;
   public:
      Cube() : Rectangle() { height = 0; volume = 0; }
      Cube(double w, double len, double h) : Rectangle(w, len)
      { height = h;
        volume = w * len * h; }
};
```

2 Constructors : No Parameter Constructor

```
class Rectangle {
    private:
        double width;
        double length;
    public:
        Rectangle() { width = 0; length = 0; }
        Rectangle(double w, double len)
        { width = w;
          length = len; }
    };

class Cube : public Rectangle {
   private:
      double height;
      double volume;
   public:
      Cube() : Rectangle() { height = 0; volume = 0; }
      Cube(double w, double len, double h) : Rectangle(w, len)
      { height = h;
        volume = w * len * h; }
};
```

2 Constructors : No Parameter Constructor

Declaration of the Cube

```
int main()
{
   Cube c();
}
```

2 Constructors : No Parameter Constructor

Declaration of the Cube

```
int main()
{
   Cube c();
}
```

Matches this Constructor

```
Cube() : Rectangle() { height = 0; volume = 0; }
```

2 Constructors : No Parameter Constructor

Declaration of the Cube

```
int main()
{
   Cube c();
}
```

Matches this Constructor

```
Cube() : Rectangle() { height = 0; volume = 0; }
```

Calls the Derived Constructor First

```
Rectangle() { width = 0; length = 0; }
```

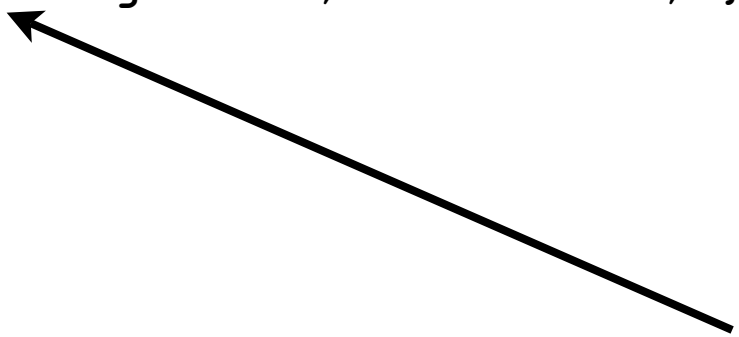
2 Constructors : No Parameter Constructor

Declaration of the Cube

```
int main()
{
   Cube c();
}
```

Matches this Constructor

```
Cube() : Rectangle() { height = 0; volume = 0; }
```



```
Rectangle() { width = 0; length = 0; }
```

Then Cube's Constructor is called.

2nd Constructor : With Parameters

```
class Rectangle {
    private:
        double width;
        double length;
    public:
        Rectangle() { width = 0; length = 0; }
        Rectangle(double w, double len)
        { width = w;
          length = len; }
    };

class Cube : public Rectangle {
    private:
        double height;
        double volume;
    public:
        Cube() : Rectangle() { height = 0; volume = 0; }
        Cube(double w, double len, double h) : Rectangle(w, len)
        { height = h;
          volume = w * len * h; }
};
```

Constructor with Parameters

Declaration of the Cube

```
int main()
{
   Cube c(10, 10, 10);
}
```

Matches this Constructor

```
Cube(double w, double len, double h) : Rectangle(w, len)
{ height = h;
  volume = w * len * h; }
```

Constructor with Parameters

Declaration of the Cube

```
int main()
{
   Cube c(10, 10, 10);
}
```

Matches this Constructor

```
Cube(double w, double len, double h) : Rectangle(w, len)
{ height = h;
  volume = w * len * h; }
```

Notice how the parameters are mapped.

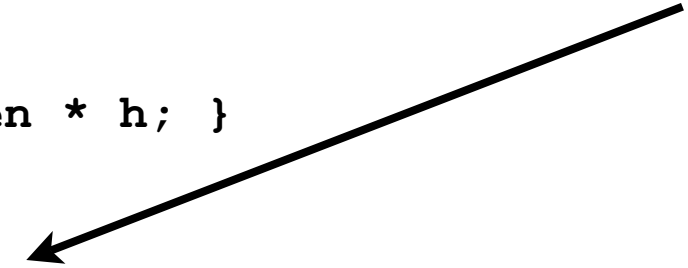
Constructor with Parameters

Declaration of the Cube

```
int main()
{
   Cube c(10, 10, 10);
}
```

Matches this Constructor

```
Cube(double w, double len, double h) : Rectangle(w, len)
{ height = h;
  volume = w * len * h; }
```



```
Rectangle(double w, double len)
{ width = w;
  length = len; }
```

Calls the Derived Constructor First

Constructor with Parameters

Declaration of the Cube

```
int main()
{
   Cube c(10, 10, 10);
}
```

Matches this Constructor

```
Cube(double w, double len, double h) : Rectangle(w, len)
{ height = h;
volume = w * len * h; }
```

```
Rectangle(double w, double len)
{ width = w;
  length = len; }
```

Then Cube's Constructor is called.

Constructors

```
class Cube : public Rectangle {
   private:
      double height;
      double volume;
   public:
      Cube() : Rectangle() { height = 0; volume = 0; }
      Cube(double w, double len, double h);
};
```

Use the ':' only when defining the member function.

```
Cube::Cube(double w, double len, double h) : Rectangle(w, len)
{
   height = h;
   volume = w * len * h;
}
```

Base class has no default constructor

```
class Rectangle {
   private:
      double width;
      double length;
   public:
      Rectangle(double w, double len)
      { width = w;
        length = len; }
};
```

Base class has no default constructor

```
class Rectangle {
   private:
      double width;
      double length;
   public:
      Rectangle(double w, double len)
      { width = w;
        length = len; }
};
```

Derived Class Constructor must call some Constructor from the Base Class.

```
class Cube : public Rectangle {
...
   public:
      Cube() : Rectangle(10.0, 10.0) { height = 0; volume = 0; }
...
}
```

Remember? This was a problem.

```
int main()
{
  Circle c;
  c.setRadius(10);
  c.setArea(157);

  cout << c.getArea() << endl;
  cout << c.getRadius() << endl;
}
```

157
10

Does not update **radius**.

```
class Shape {
    private:
        double area;
    public:
        void setArea(double a)
            { area = a; }
        double getArea()
            { return area; }
};

class Circle : public Shape {
    private:
        double radius;
    public:
        void setRadius(double r)
            { radius = r;
              setArea(3.14 * r * r); }

        double getRadius()
            { return radius; }
};
```

Circle class can ***redefine*** Shape classes function `setArea(...)`

```
class Circle : public Shape {
   private:
      double radius;
   public:
      void setRadius(double r)
         { radius = r;
           setArea(3.14 * r * r); }

      double getRadius()
         { return radius; }

};
```

Redefined `setArea(...)` solves for the new radius

```
class Circle : public Shape {
   private:
      double radius;
   public:
      void setRadius(double r)
         { radius = r;
           setArea(3.14 * r * r); }

      double getRadius()
         { return radius; }

      void setArea(double a)
      {
         radius = sqrt(3.14 / a); // #include <math>
      }
};
```

But! New `setArea()` is responsible for updating the `area`

```
class Circle : public Shape {
   private:
      double radius;
   public:
      void setRadius(double r)
         { radius = r;
           setArea(3.14 * r * r); }

      double getRadius()
         { return radius; }

      void setArea(double a)
      {
         radius = sqrt(3.14 / a); // #include <math>
         Shape::setArea(a);
      }
};
```

Calls the Base Class setArea(...)

But wait, we're not done yet!

```
class Circle : public Shape {
   private:
      double radius;
   public:
      void setRadius(double r)
         { radius = r;
           setArea(3.14 * r * r); }

      double getRadius()
         { return radius; }

      void setArea(double a)
      {
         radius = sqrt(3.14 / a); // #include <math>
         Shape::setArea(a);
      }
};
```

Call Shape's `setArea(...)`

```
class Circle : public Shape {
   private:
      double radius;
   public:
      void setRadius(double r)
         { radius = r;
           setArea(3.14 * r * r); }

      double getRadius()
         { return radius; }

      void setArea(double a)
      {
         radius = sqrt(3.14 / a); // #include <math>
         Shape::setArea(a);
      }
};
```

Call Shape's `setArea(...)`

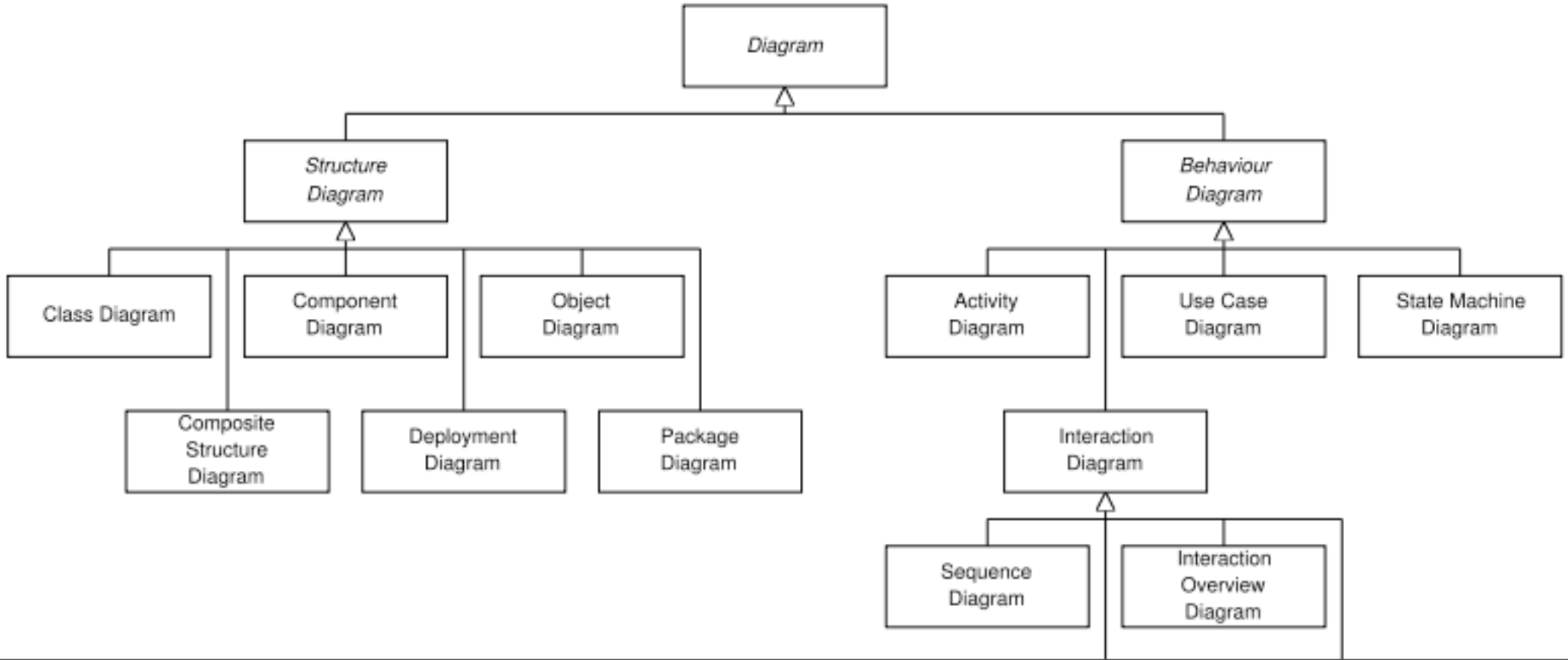
```
class Circle : public Shape {
   private:
      double radius;
   public:
      void setRadius(double r)
         { radius = r;
           Shape::setArea(3.14 * r * r); }

      double getRadius()
         { return radius; }

      void setArea(double a)
      {
         radius = sqrt(3.14 / a); // #include <math>
         Shape::setArea(a);
      }
};
```

UML

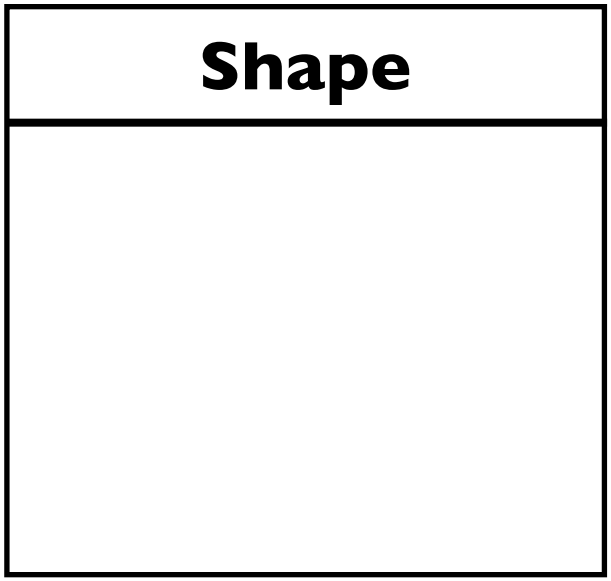
UML is the **U**nified **M**odeling **L**anguage.

It is a graphical language with a general purpose approach to ***object modeling***

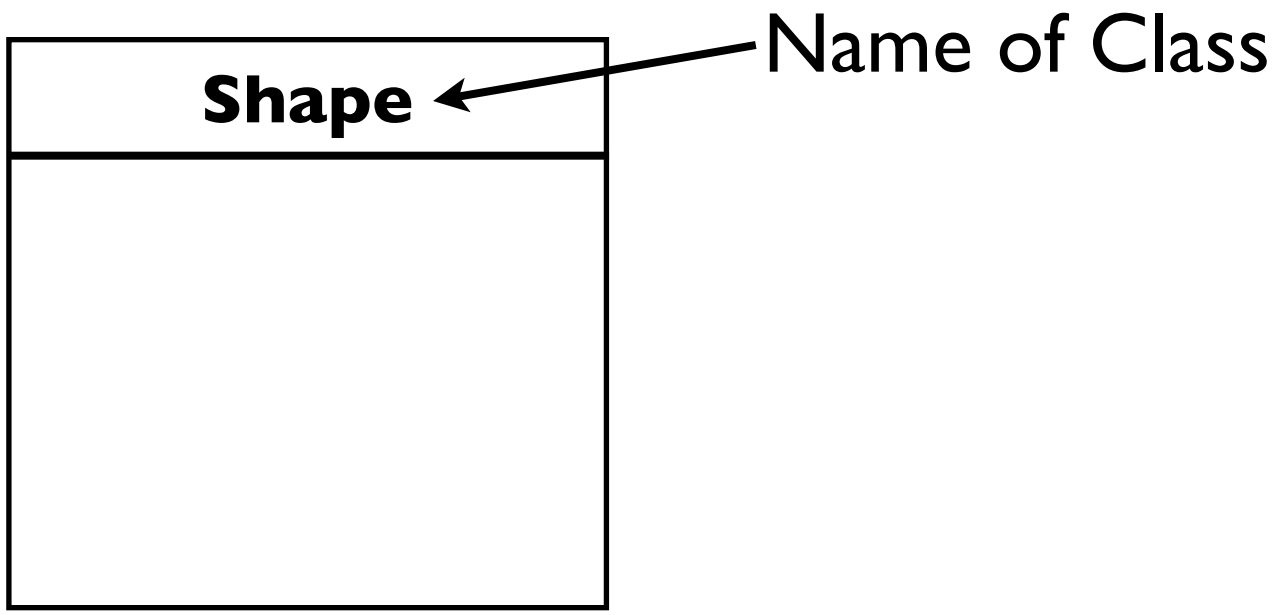
There are many components to UML: we will look at the Class Relationship component.



Defining a Class in UML

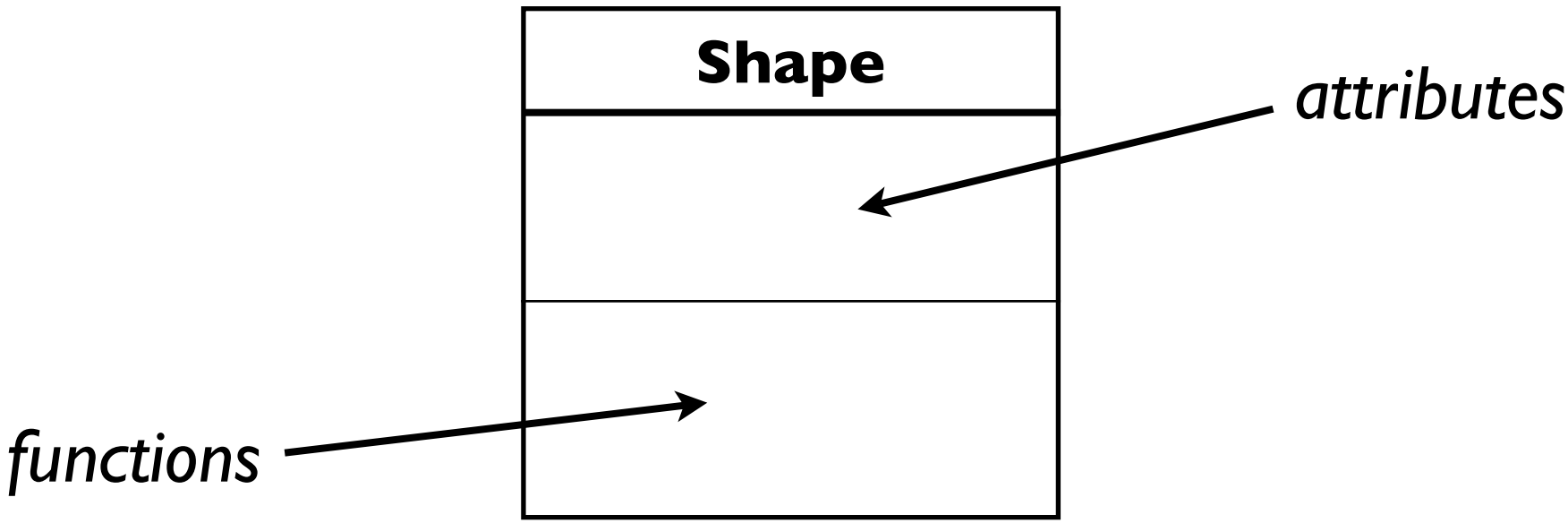


Defining a Class in UML



One class = One Box

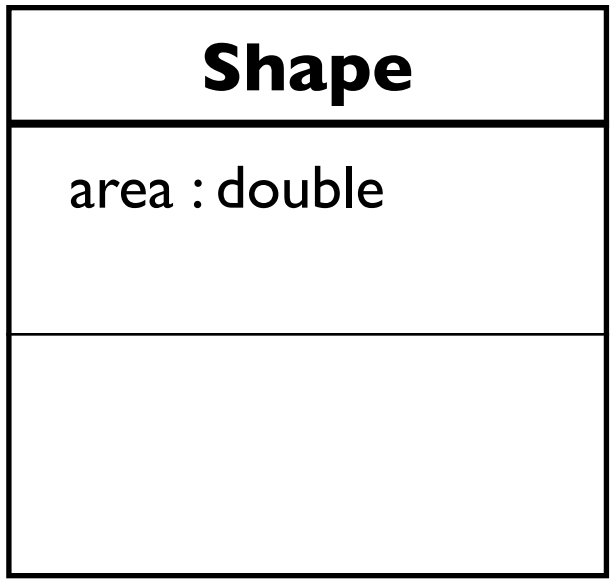
Defining a Class in UML



Class has two sections

1. Attributes (i.e. member variables)
2. Methods (i.e. member functions)

Defining a Class in UML

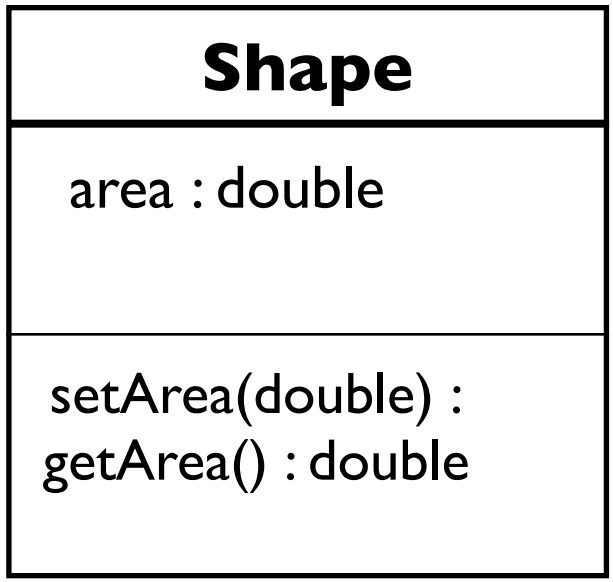


Attributes are defined:

name : *type*

(backwards from C/C++ style)

Defining a Class in UML

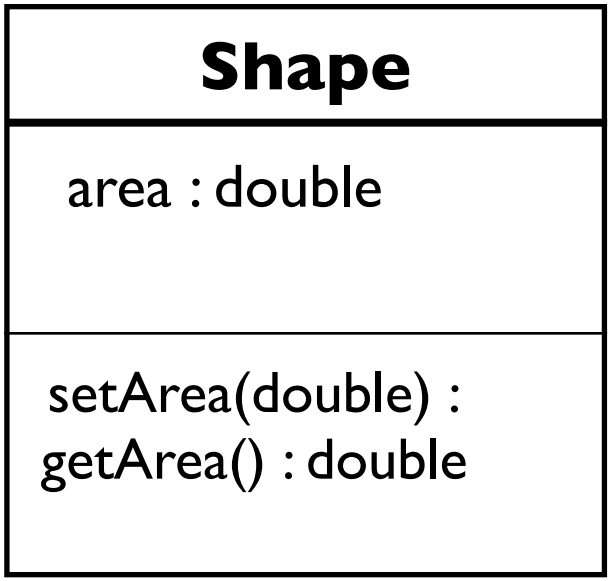


Methods are defined:

name : *return type*

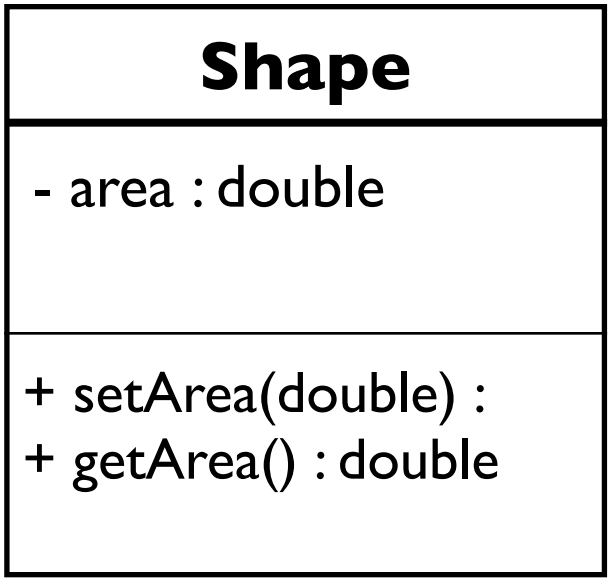
(leave *return type* blank for void)

Defining a Class in UML



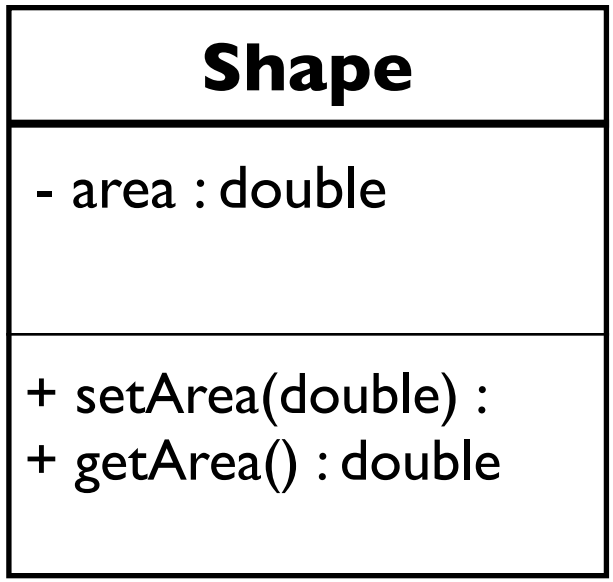
What else are we missing from our model of the Shape class?

Defining a Class in UML



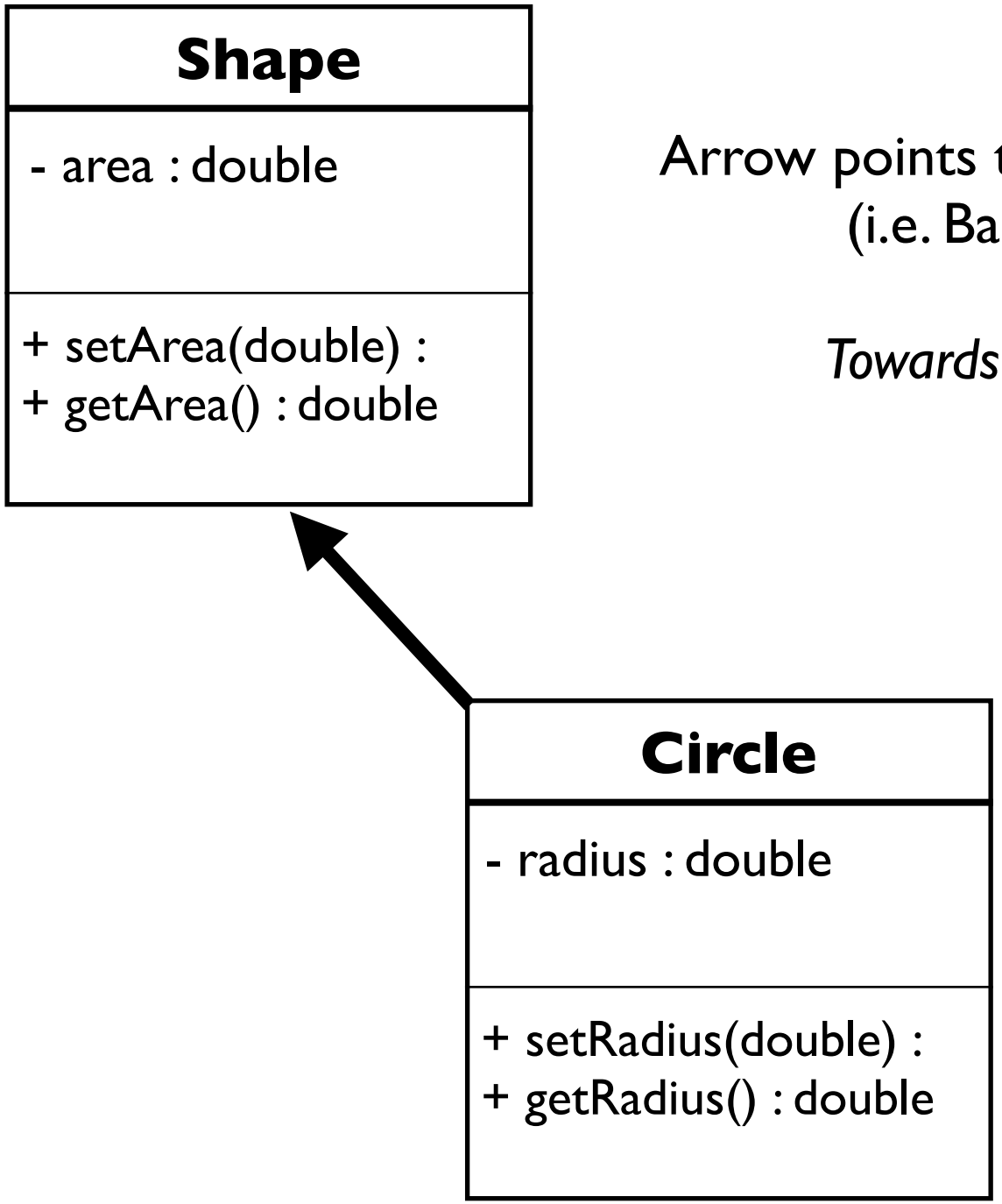
Access Level for the Members is specified by a punctuation right before the name.

Defining a Class in UML



Punctuation	Access Level
+	public
-	private
#	protected

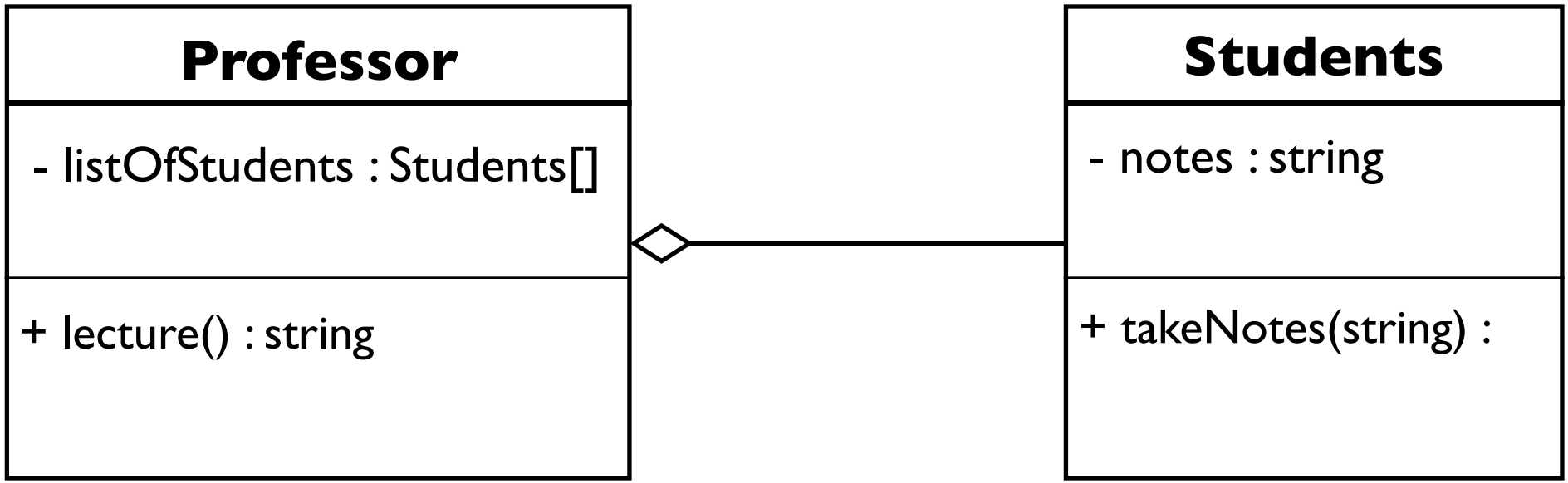
Relationships (*is-a*) : Inheritance



Arrow points to the super-class (i.e. Base Class).

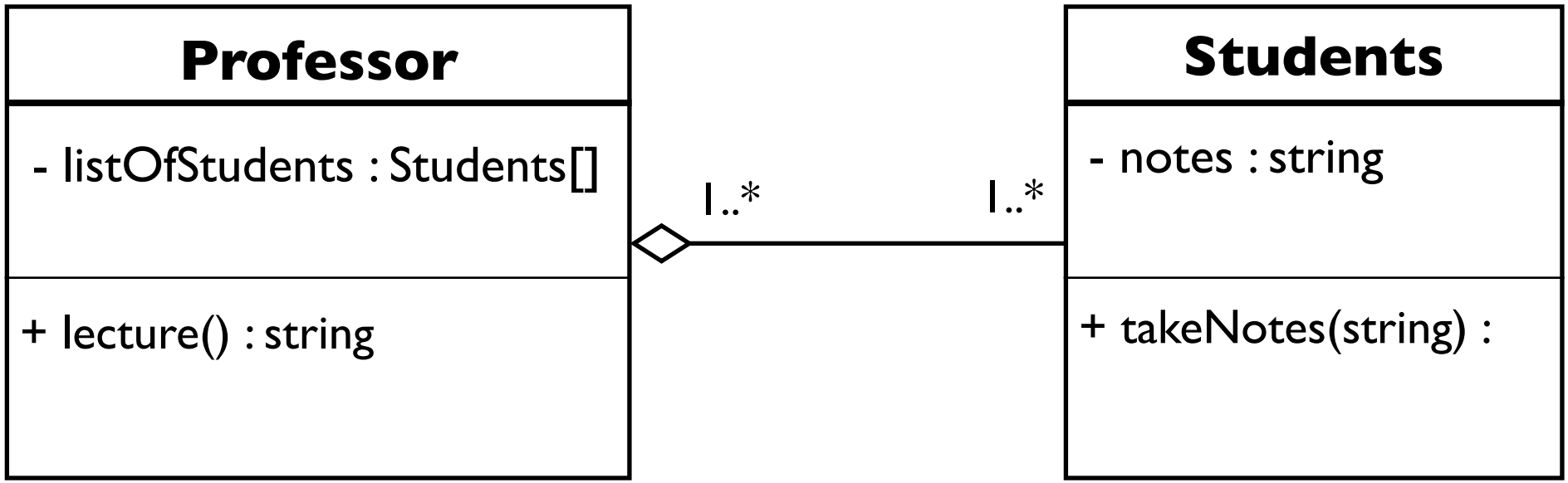
Towards Generality.

Relationships (*has-a*) : Aggregation



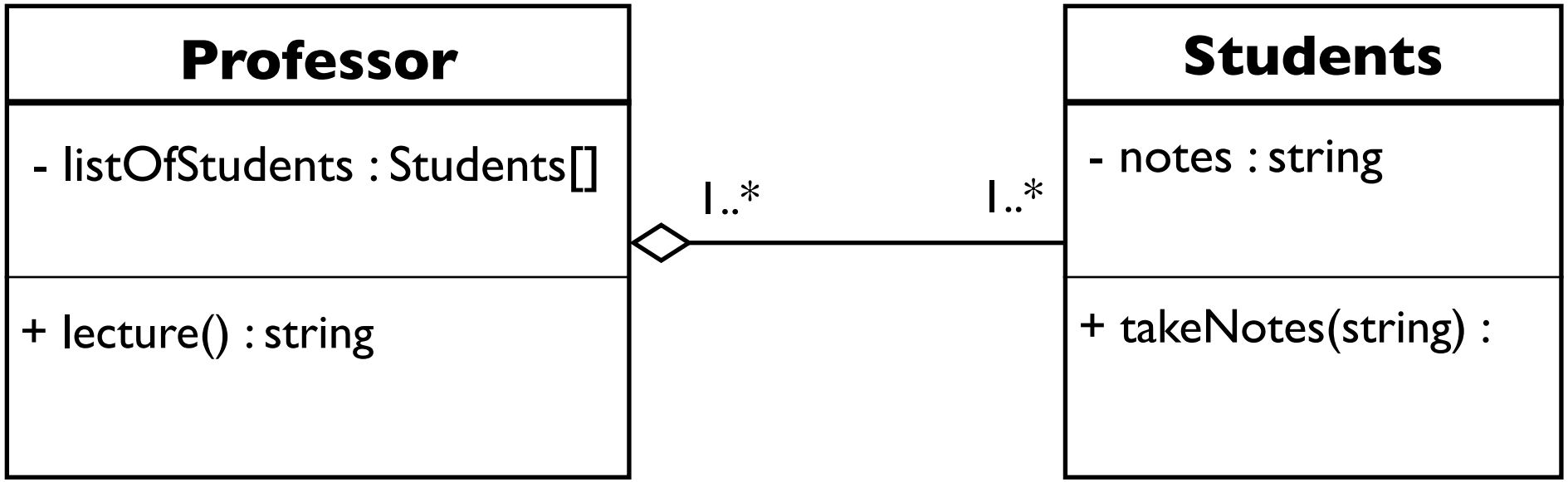
Aggregation is indicated by the diamond shaped arrow.

Relationships (*has-a*) : Aggregation



Multiplicity - number of objects that participate in the relationship
(Commonly used in describing Database topologies : a.k.a. *cardinality*)

Relationships (*has-a*) : Aggregation



Multiplicity	Description
0..1	No instances, or one instance (optional)
1	Exactly one instance
0..* or *	Zero or more instances
1..*	One or more instances (at least one)

Extend your Cat class through a UML Diagram.

```
class Cat
{
```

```
};
```

3 Member Variables
Constructor
Destructor
Accessor/Mutator Functions.