Object-Oriented Design

Object-Oriented Programming in C++

Roadmap

- Four main themes:
  - Well-Behaved Objects ✓
  - **Object-Oriented Design**
  - Generic Programming & STL
  - API Design
- Other
  - Miscellaneous ✓
  - New C++ standard (2011)

Object-Oriented Design

- **How?**
  - Inheritance
  - Virtual functions
  - Polymorphic behavior
- **When?**
  - Is (public) inheritance appropriate?
- **What?**
  - Exactly are we inheriting?

```
class Base
{
public:
  virtual void foo() {
    // Implementation ...
  }
};

class Derived: public Base
{
public:
  virtual void foo() {
   // A different impl. ...
  }
};
```

A Few Questions

- What is a rectangle ?

- What is a square ?

- Is square a rectangle ?

```
class Rectangle { . . . };
class Square : public Rectangle { . . . };
```

Common Pitfalls

- Newcomers to OO programming tend to overuse inheritance.

"If all you have is a hammer, everything looks like a nail."

- Proverb

What?

- Public inheritance is used do model "is-a" relationships.
- Public inheritance is used do model "is-a" relationships.
- Public inheritance is used do model "is-a" relationships
  - **... only**.
- **Probably the most important rule of OO design.**

SM [32]

Public Inheritance

```
class Base
{
public:
  virtual void foo() {
    // Implementation ...
  }
};

class Derived: public Base
{
public:
  virtual void foo() {
   // A different impl. ...
  }
};
```

- Base is a more general type, Derived is a specialization
- Every object of type Derived is also an object of type Base
  - (not vice versa)
- Anywhere an object of type Base can be used, an object of type Derived can be used as well.
  - (not vice versa)

Examples

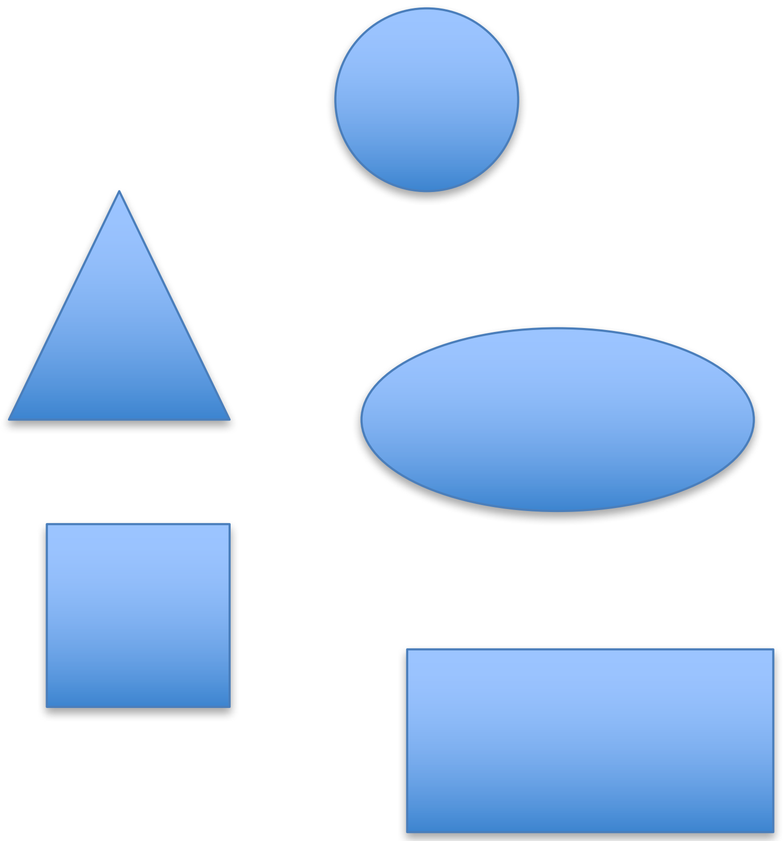
```
class Person { . . . };
class Student : public Person { . . . };
```

```
class Dog { . . . };
class Schnauzer : public Dog { . . . };
class MiniatureSchnauzer : public Schnauzer { . . . };
class StandardSchnauzer  : public Schnauzer { . . . };
class GiantSchnauzer     : public Schnauzer { . . . };
```

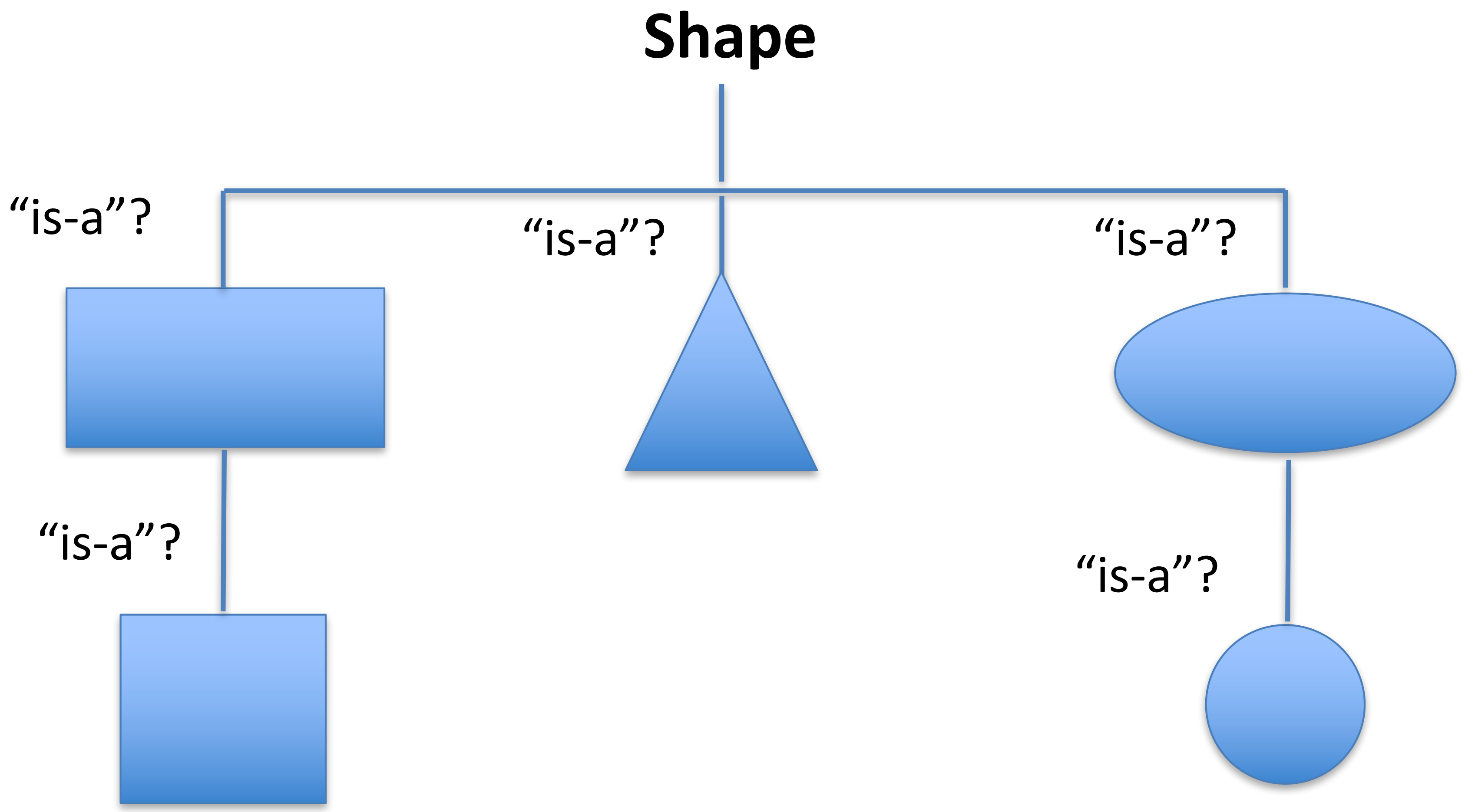
```
class Bird { . . . };
class Eagle  : public Bird { . . . };
class Penguin: public Bird{ . . . };
```

Example with Shapes

- Shapes
  - Rectangle
  - Ellipse
  - Triangle
  - Square
  - Circle
- Your task:
  - Create an inheritance hierarchy rooted in Shape.



Example (cont.)



Our intuitive notion of "is-a" is not the same as OO "is-a"

Liskov Substitution Principle (LSP)

- What is proper public inheritance?
  - *It should always be possible to substitute a base class for a derived class without any change in behavior. (LSP)*

- *For example, should be able to use*
  - *a Student where a Person is used, or*
  - *a Penguin for a Bird, etc.*

LSP in Practice

```
class Circle : public Ellipse {
Public:
    //Override setMajorRadius/setMinorRadius to set both, and ..
    void setRadius( float r ) {
        setMajorRadius( r );
    }
    float getRadius() const { return getMajorRadius(); }
};


void TestEllipse( Ellipse &e ) {
    e.setMajorRadius( 10 );
    e.setMinorRadius( 20 );
    assert(e.getMajorRadius()==10 && e.getMinorRadius()==20);
}
. . .
Ellispe e;
Circle  c;
TestEllipse(e);
TestEllipse(c);
```

Must be OK for LSP to hold.

However, changes behavior; thus not proper public inheritance.

Let’s Ask Questions Again

- What is a rectangle ?

- What is a square ?

- Is square a rectangle ?
- Well,
  - Yes, not necessarily (in the OO sense).
  - No, not necessarily (in the OO-sense)
  - Necessarily? Is is then sometimes?

Depends on our Design

- For example, what if our program needs not change objects once they are constructed.
  - e.g. no changing of Radius
- Does the LSP now hold for?
  - ellipse/circle
  - rectangle/square
- If that is the case
  - public inheritance is fine
  - If not, we must model the relationship differently.

Composition

- Often more appropriate to use composition.

```
class Circle {
public:
    ...
    void setRadius( float r ) {
        el_.setMajorRadius( r );
        el_.setMinorRadius( r );
    }
    float getRadius() const {return el_.getMajorRadius();}
private:
    Ellispe el_;
};
```

- Models “has-a” (or “implemented-in-terms-of”) relationships.

Exercises

- How do you think it is best to model the OO relationships between?
  - Person and Student
  - Rectangle and Square
  - Bird and Penguin
  - DFA and NFA
  - …

Questions?

- I have one final question for you?
- Is a square (object) a rectangle (object) ?

Summary

- How to model relationships between classes?
  - Use public inheritance for "is-a" (in the OO sense) relationships only, i.e. make sure passes the LSP.
    - Do not use public inheritance otherwise.
  - Use composition for "is-implemented-in-terms-of" relationships.
    - Private inheritance also models "is-implemented-in-terms-of", but use judiciously.
      - Choose which? Use composition whenever you can and private inheritance whenever you must.

SM [39]

  - Prefer composition to inheritance