CS 617 Object Oriented Systems Lecture 8 Inheritance, Reuse, Polymorphism, Dynamic Binding 3:30-5:00 pm Mon, Jan 28

Rushikesh K Joshi

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Outline

Outline

Is Conceptual Compatibility Enforced?

The models of inheritance in OOPLs do not enforce conceptual compatibility between a subclass and its superclass

Some Examples

- A set of mathematical functions in a class, use the class as superclass to avoid call indirections or additional receiver names
- An implementation as a superclass e.g. an Array used inside a class implementing LIFO abstraction
- When a whole component needs exactly one instance of each of its components

What happens to the visibilities of members in superclasses in the private inheritance model?

Impact of Private Inheritance on Member Visibility

X: superclass
Y: subclass
Z: an independent class using an object reference of type Y
YY: subclass of subclass

- Private members of X are visible in X, not in Y, not from Z
- Protected members of X are visible in X, in Y, not from Z
- Public members of X are visible in X, in Y, not from YY,Z
- Private and Protected members of X are not visible in YY

Extended Inheritance Model: Protected Inheritance

X: superclass
Y: subclass
Z: an independent class using an object reference of type Y
YY: subclass of subclass

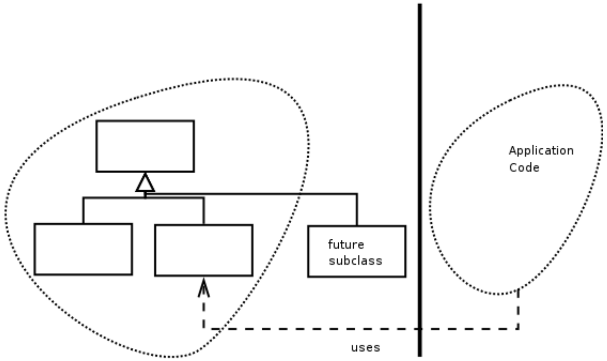
- Private members of X are visible in X, not in Y, not from Z
- Protected members of X are visible in X, in Y, not from Z
- Public members of X are visible in X, in Y, not from Z
- Private members of X are not visible in YY, but protected and public members of X are

The Model of Private Inheritance

- It’s known that inheritance is being used for non-conceptual reasons
- Derived class does not export base class’s interface
- Derived class uses implementation of base class
- If inheritance was not be used, what alternative design would you choose?

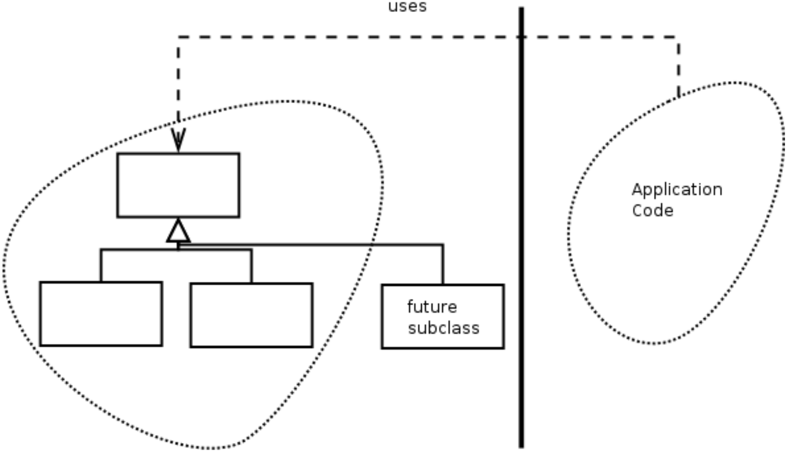
Outline

Reuse Through Extension and Refinements



Outline

Towards Higher Reuse through Polymorphism



Dynamic Binding and Polymorphism I

```
class A {
public:
        virtual void f () { cout « "A.f "; };
        virtual void g () { cout « "A.g "; };
        virtual void h () { cout « "A.h "; };
        virtual void k () { cout « "A.k "; };
};
class B : public A {
public:
        virtual void g () { cout « "B.g "; };
        virtual void h () { cout « "B.h "; };
};
class C : public B {
public:
        virtual void h () { cout « "C.h "; };
        virtual void k () { cout « "C.k "; };
};
```

Dynamic Binding and Polymorphism II

```
main () {

C *cp = new C;
B* bp = cp;
A* a1 = cp;
A* a2 = bp;
A* a3 = new B;
        cp->f(); cp->g(); cp->h(); cp->k();
        bp->f(); bp->g(); bp->h(); bp->k();
        a1->f(); a1->g(); a1->h(); a1->k();
        a2->f(); a2->g(); a2->h(); a2->k();
        a3->f(); a3->g(); a3->h(); a3->k();
}
```