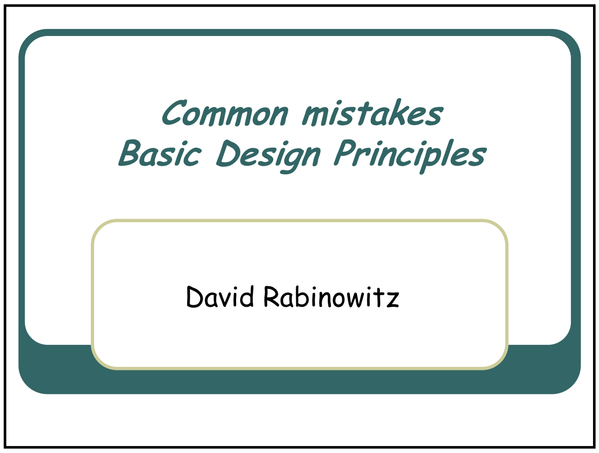
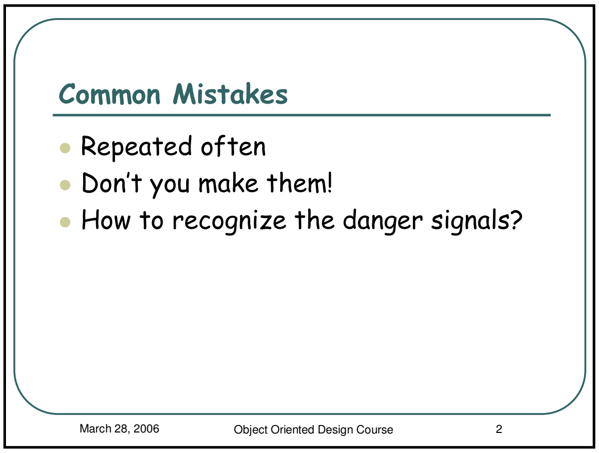


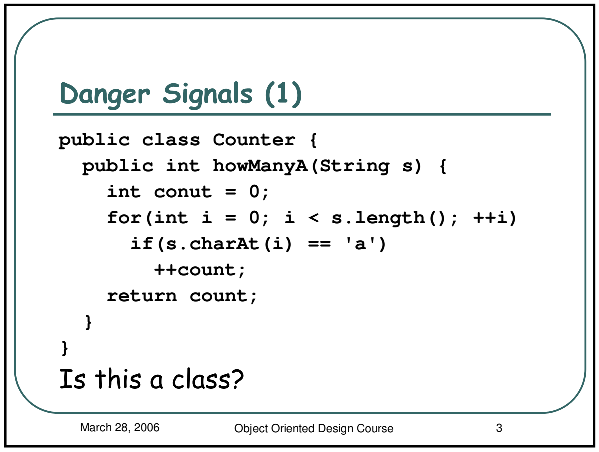
Common mistakes
Basic Design Principles

David Rabinowitz



Common Mistakes

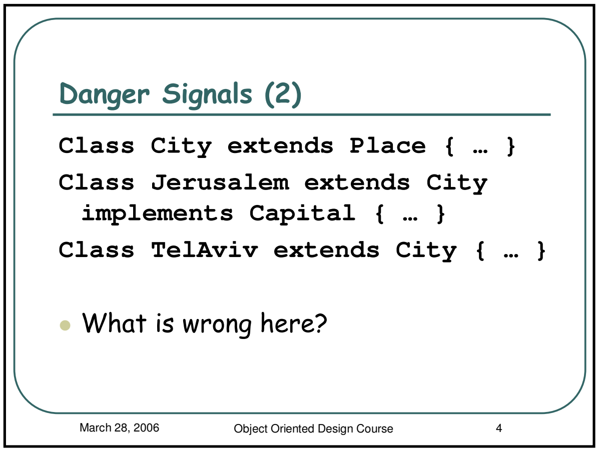
- Repeated often
- Don't you make them!
- How to recognize the danger signals?



Danger Signals (1)

```
public class Counter {
  public int howManyA(String s) {
    int conut = 0;
    for(int i = 0; i < s.length(); ++i)
      if(s.charAt(i) == 'a')
        ++count;
    return count;
  }
}
```

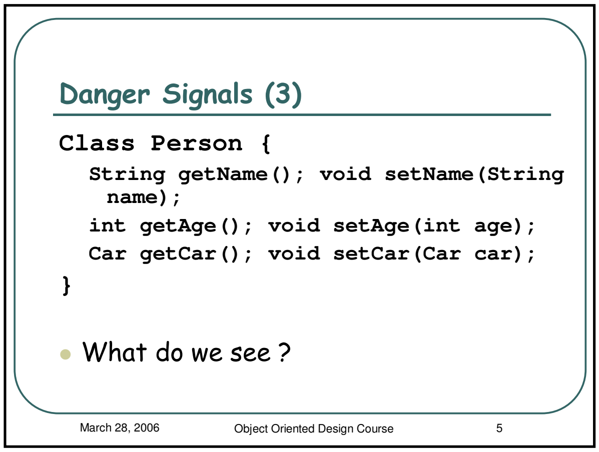
Is this a class?



Danger Signals (2)

```
Class City extends Place { … }
Class Jerusalem extends City
  implements Capital { … }
Class TelAviv extends City { … }
```

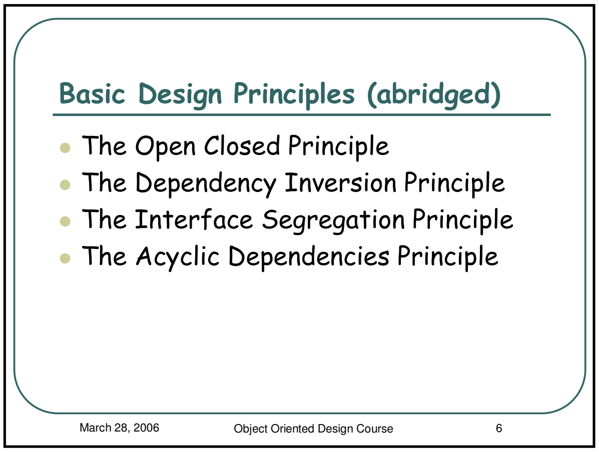
- What is wrong here?



Danger Signals (3)

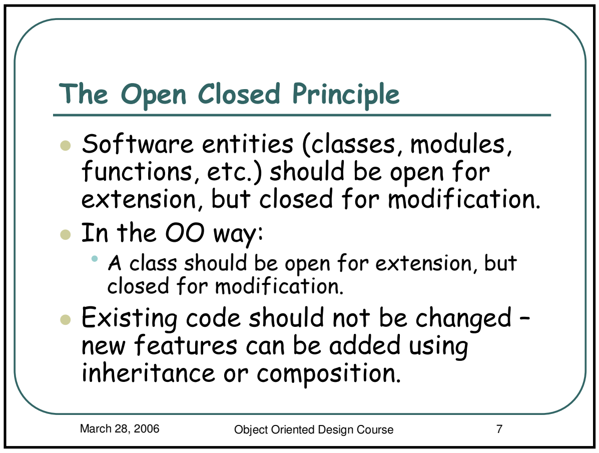
```
Class Person {
   String getName(); void setName(String
     name);
   int getAge(); void setAge(int age);
   Car getCar(); void setCar(Car car);
}
```

- What do we see ?



Basic Design Principles (abridged)

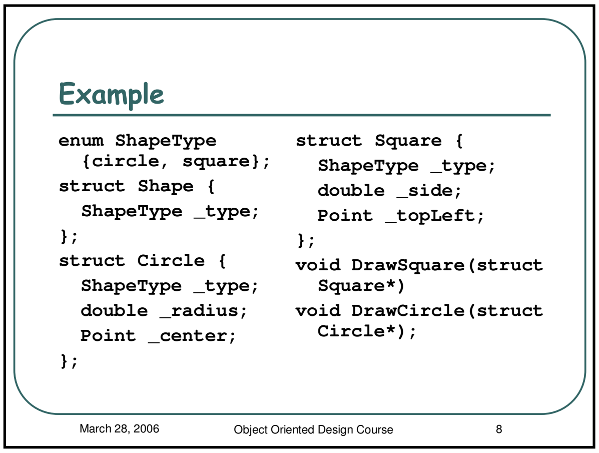
- The Open Closed Principle
- The Dependency Inversion Principle
- The Interface Segregation Principle
- The Acyclic Dependencies Principle



The Open Closed Principle

- Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.
- In the OO way:
  - A class should be open for extension, but closed for modification.
- Existing code should not be changed – new features can be added using inheritance or composition.

March 28, 2006 Object Oriented Design Course 7

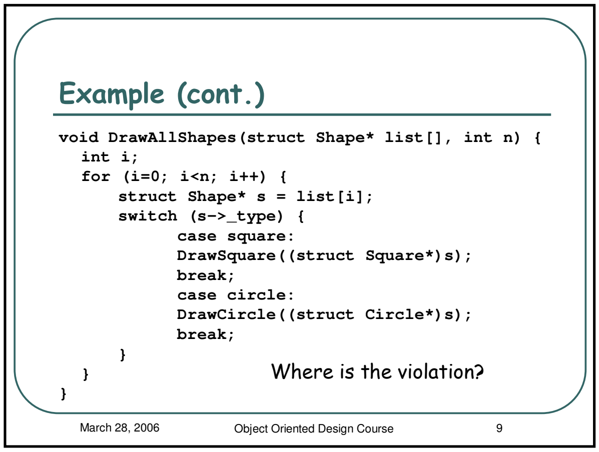


Example

```
enum ShapeType
  {circle, square};
struct Shape {
  ShapeType _type;
};
struct Circle {
  ShapeType _type;
  double _radius;
  Point _center;
};
```

```
struct Square {
  ShapeType _type;
  double _side;
  Point _topLeft;
};
void DrawSquare(struct
  Square*)
void DrawCircle(struct
  Circle*);
```

March 28, 2006 Object Oriented Design Course 8

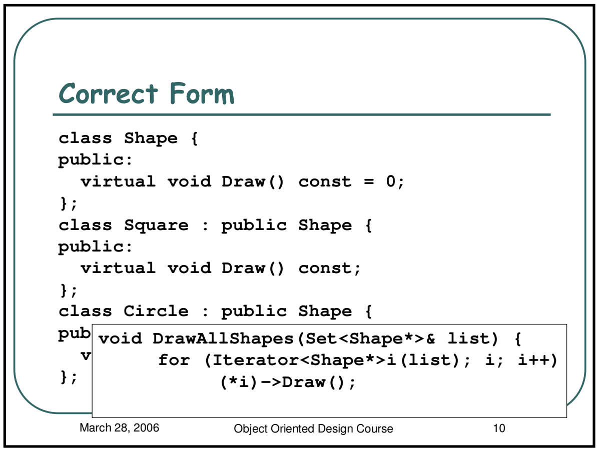


Example (cont.)

```
void DrawAllShapes(struct Shape* list[], int n) {
  int i;
  for (i=0; i<n; i++) {
      struct Shape* s = list[i];
      switch (s->_type) {
           case square:
           DrawSquare((struct Square*)s);
           break;
           case circle:
           DrawCircle((struct Circle*)s);
           break;
      }
  }
}
```

Where is the violation?

March 28, 2006 Object Oriented Design Course 9

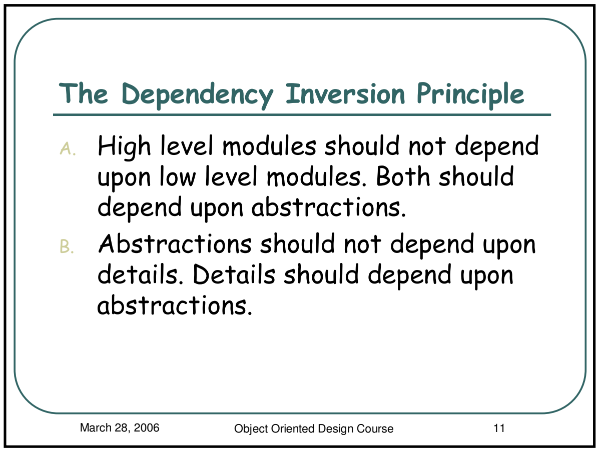


Correct Form

```
class Shape {
public:
  virtual void Draw() const = 0;
};
class Square : public Shape {
public:
  virtual void Draw() const;
};
class Circle : public Shape {
pub
  v
};
```

```
void DrawAllShapes(Set<Shape*>& list) {
     for (Iterator<Shape*>i(list); i; i++)
          (*i)->Draw();
```

March 28, 2006 Object Oriented Design Course 10

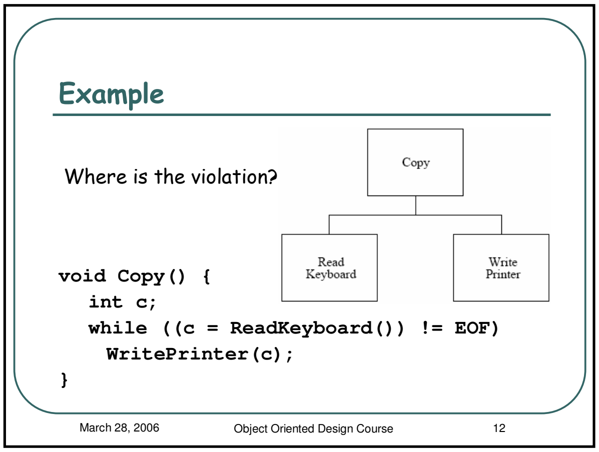


The Dependency Inversion Principle

A. High level modules should not depend upon low level modules. Both should depend upon abstractions.

B. Abstractions should not depend upon details. Details should depend upon abstractions.

March 28, 2006 Object Oriented Design Course 11



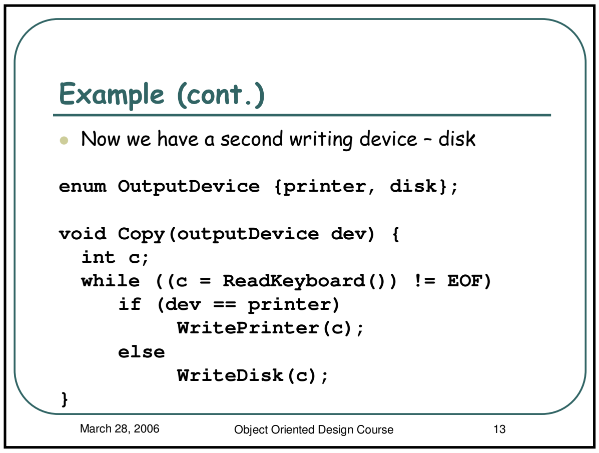
Example

Where is the violation?

Copy
Read Keyboard
Write Printer

```
void Copy() {
   int c;
   while ((c = ReadKeyboard()) != EOF)
     WritePrinter(c);
}
```

March 28, 2006 Object Oriented Design Course 12

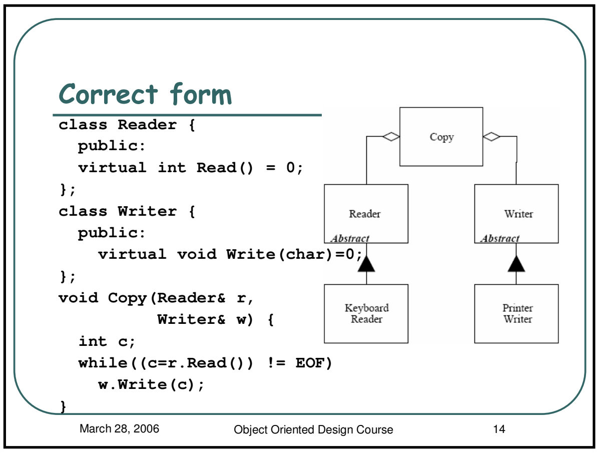


Example (cont.)

- Now we have a second writing device – disk

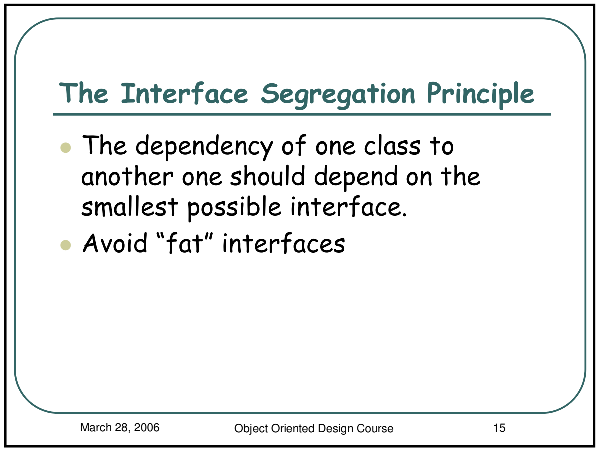
```
enum OutputDevice {printer, disk};

void Copy(outputDevice dev) {
  int c;
  while ((c = ReadKeyboard()) != EOF)
     if (dev == printer)
          WritePrinter(c);
     else
          WriteDisk(c);
}
```



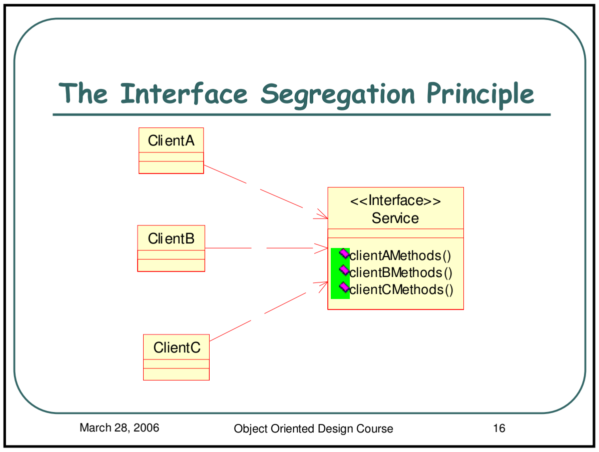
Correct form

```
class Reader {
  public:
  virtual int Read() = 0;
};
class Writer {
  public:
    virtual void Write(char)=0;
};
void Copy(Reader& r,
          Writer& w) {
  int c;
  while((c=r.Read()) != EOF)
    w.Write(c);
}
```

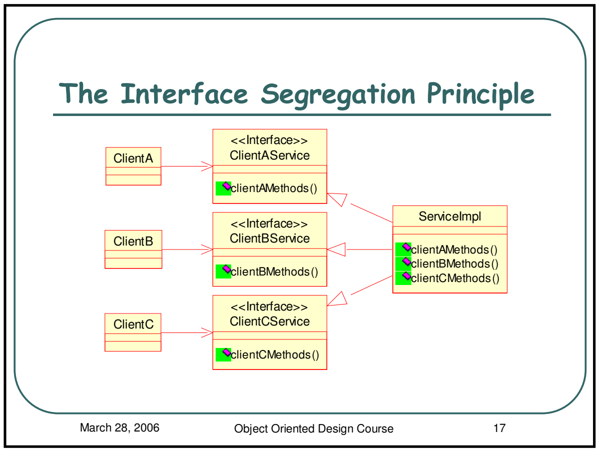


The Interface Segregation Principle

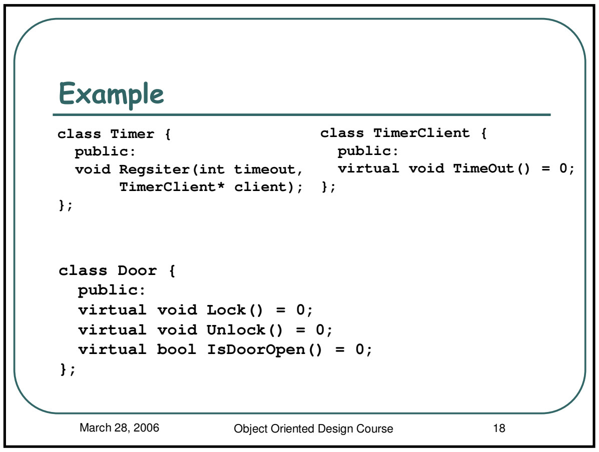
- The dependency of one class to another one should depend on the smallest possible interface.
- Avoid "fat" interfaces



The Interface Segregation Principle



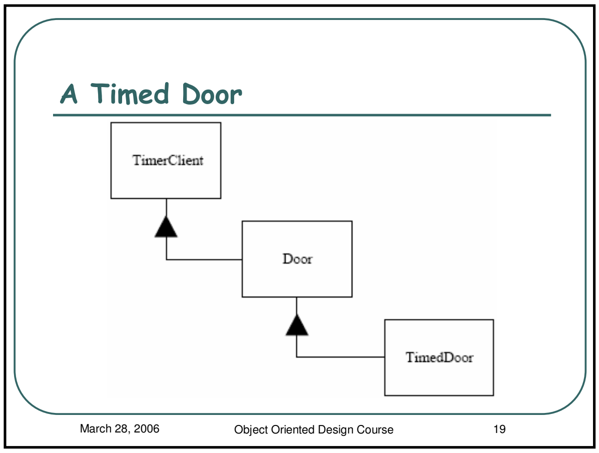
The Interface Segregation Principle



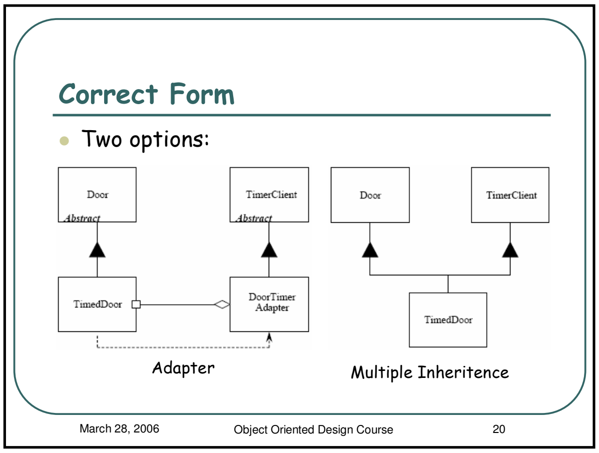
Example

```
class Timer {                    class TimerClient {
  public:                          public:
  void Regsiter(int timeout,       virtual void TimeOut() = 0;
        TimerClient* client);    };
};


class Door {
  public:
  virtual void Lock() = 0;
  virtual void Unlock() = 0;
  virtual bool IsDoorOpen() = 0;
};
```



A Timed Door

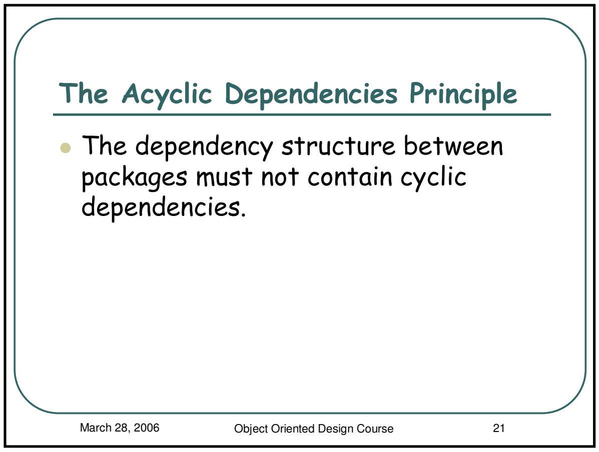


Correct Form

- Two options:

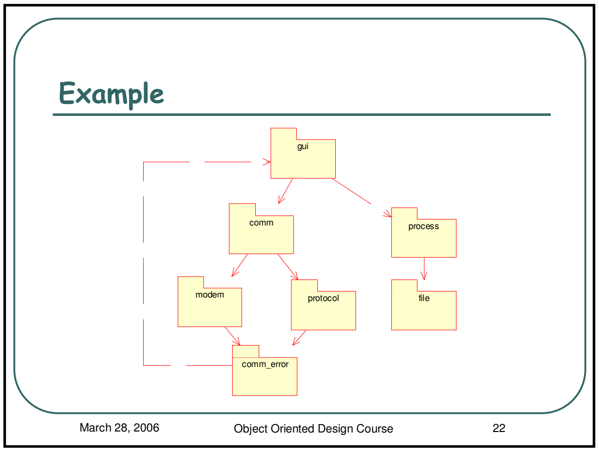
Adapter

Multiple Inheritence

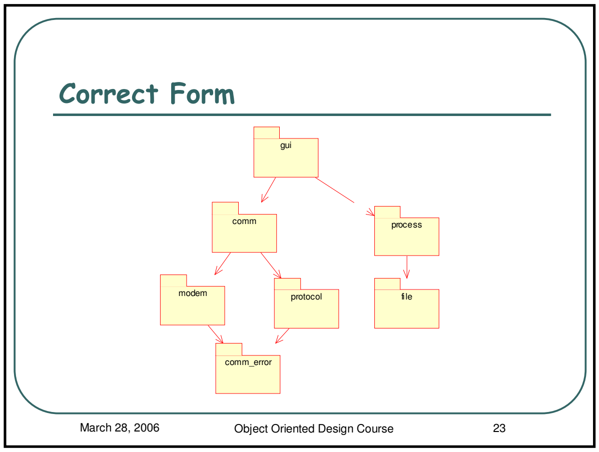


The Acyclic Dependencies Principle

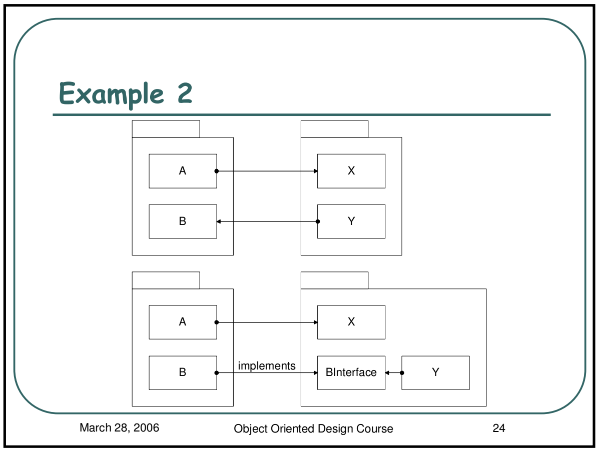
- The dependency structure between packages must not contain cyclic dependencies.



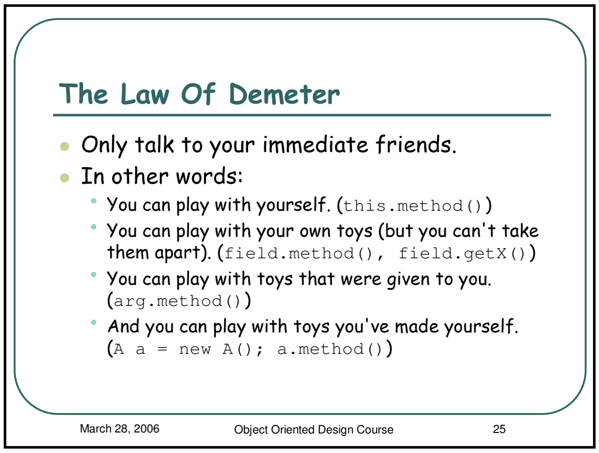
Example



Correct Form



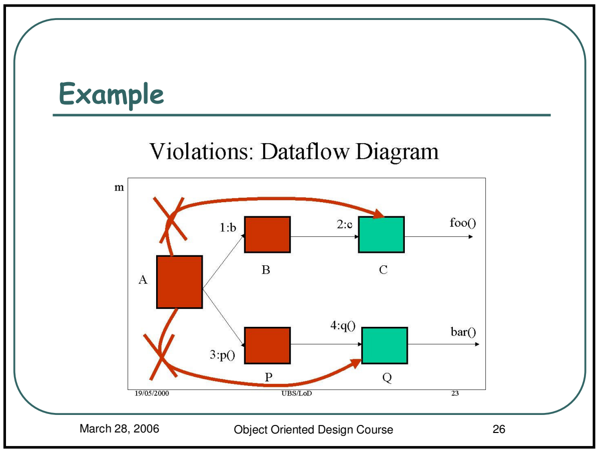
Example 2



The Law Of Demeter

- Only talk to your immediate friends.
- In other words:
  - You can play with yourself. (`this.method()`)
  - You can play with your own toys (but you can't take them apart). (`field.method()`, `field.getX()`)
  - You can play with toys that were given to you. (`arg.method()`)
  - And you can play with toys you've made yourself. (`A a = new A(); a.method()`)

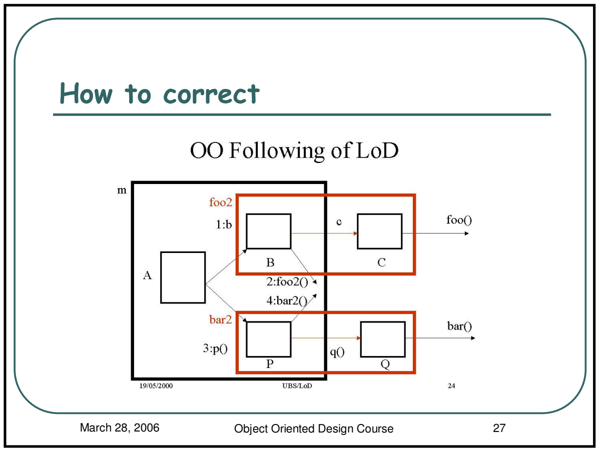
March 28, 2006 Object Oriented Design Course 25



Example

Violations: Dataflow Diagram

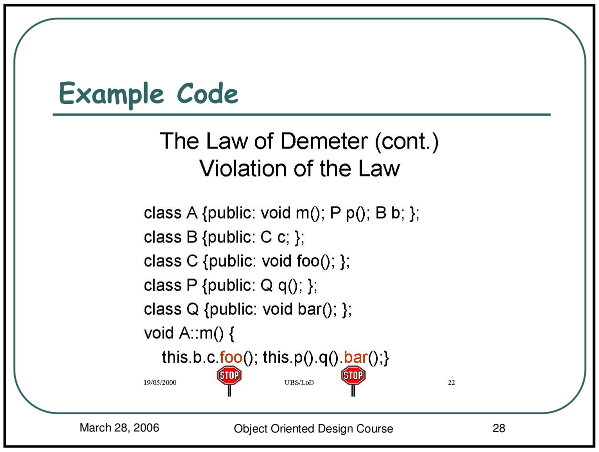
March 28, 2006 Object Oriented Design Course 26



How to correct

OO Following of LoD

March 28, 2006 Object Oriented Design Course 27



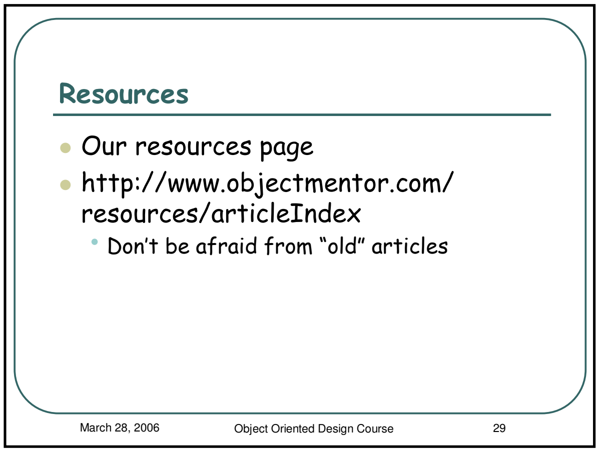
Example Code

The Law of Demeter (cont.)
Violation of the Law

```
class A {public: void m(); P p(); B b; };
class B {public: C c; };
class C {public: void foo(); };
class P {public: Q q(); };
class Q {public: void bar(); };
void A::m() {
   this.b.c.foo(); this.p().q().bar();}
```

19/05/2000
STOP
UBS/LoD
STOP
22

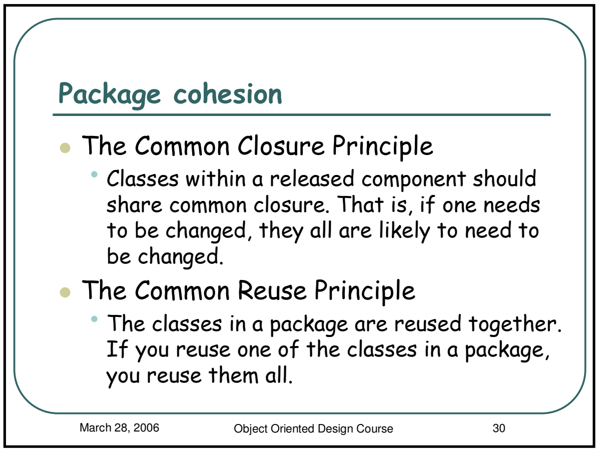
March 28, 2006 Object Oriented Design Course 28



Resources

- Our resources page
- http://www.objectmentor.com/resources/articleIndex
  - Don't be afraid from "old" articles

March 28, 2006 Object Oriented Design Course 29



Package cohesion

- The Common Closure Principle
  - Classes within a released component should share common closure. That is, if one needs to be changed, they all are likely to need to be changed.
- The Common Reuse Principle
  - The classes in a package are reused together. If you reuse one of the classes in a package, you reuse them all.

March 28, 2006 Object Oriented Design Course 30