existing code, leaving everything that has been written so far unchanged.

• Is there a way to add new behavior to L-Just without touching any of the
  class and all appropriate methods to each class.

• Each time we want to compute something new, we have to edit each

What's The Matter?
ever again!

design of List one more method and never have to modify anything

- For List to execute any of these algorithms, we just need to add to the
  remove().

- The invariant behaviors are the methods find(), insert(), and

  The invariant behaviors are the methods find(), insert(), and

  Here, the variant behaviors are the infinitely many algorithms (i.e.,

  pattern (OOP #1).

- The key is to encapsulate the variant behaviors into a separate Union

  Toward a Solution...
ALL VISTORS AS WELL

If we have to modify the host union, then we will have to modify
stable and does not change. This extensibility only works if the taxonomy of the host union is
many external algorithms without changing any of the host union
extrinsic algorithms on the host permits the addition of inherently
appropriate visitor method.

This “decoupling” of the hosts structural behaviors from the
caller visitor and leaves it up to each of its concrete variants to call the
visitor and accept it (called the “hook”) to “accept” a
host union. The abstract host has a method (called the “hook”) to “accept” a
host union.

It has a separate method for each of the concrete variants of the
abstract visitor is usually defined as an interface in Java.

between two union patterns: a “host” union and a “visitor” union.

The Visitor pattern is a framework for communication and collaboration

The Visitor Pattern
that manufacture unique instances of themselves.

* All the "state-less" visitors should be singletons and are factories

that it wants.

An appropriate "factory" will provide the client with concrete visitors

- A client program, say StructClient, only deals with the structure

- In practice, the host union is encapsulated inside of another class, say

The Visitor Pattern (cont.)
public int getFinish() {
    public void setFinish(int finish);
    public static final int MATT = 0, GLOSSY = 1;
}

public interface Paintable extends Colorable {
    // Example
    // One interface can extend another interface.

    // Example
    // Implementations.

    // A set of method and constant declarations, without the method
    // What is an interface?

    // Declaring Interfaces

February 16, 2000
Comp 212
Using Interfaces

```

Example

- An interface is a reference type, just like a class.

    { }

    public int getcolor() { return -color; }

    public void setcolor(int color) {
        color = color;
    }

class Colorable point extends Point implements Colorable

    { }

    class Point { int x, y; }

Example

- In a class definition, we say that a class implements an interface.

    How do you use an interface?
```
Using Interfaces (cont.)

A class can implement one or more interfaces.