Defining Interfaces

What is an interface?

• A set of method and constant declarations, without the method

Example *

One interface can extend another interface.

Example *

Implementations.

• Public interface Colorable

  { *

    public int getTexture();
    public void setTexture(int texture);
    public static final int MATT = 0, GLOSSY = 1;
  }

  public interface Paintable extends Colorable
Using Interfaces
A class can implement one or more interfaces.

Using Interfaces (cont.)
In contrast, we defined the host as an abstract method.

- Method objects only.
- Method objects only.

For example, they would define the methods of `ILlist` as follows:

```
1. Object for Empty (Empty list, object host, object input) to act on
2. Object for NonEmpty (NonEmpty list, object host, object input) to act on
```

Our definition of the `Visitor` Pattern differs from the (standard) definition presented by the GoF book ("Design Patterns").
NETList and thus can access all package private elements. * LsFactory resides in the same package as EmptyList and

to instantiate and initialize concrete list objects.

If checks for valid input before calling on the appropriate constructors

NETList objects:

- A factory class, LsFactory, is provided to build EmptyList and NETList are

- For this reason, the constructors for EmptyList and NETList are

- it is good software engineering practice to shield clients from the details of correctly manipulating concrete instances of a list.

Software Engineering Issues
via their public behaviors. Since AllList, EmptyList, NELList, Persistent, and ListFactory are all public classes, they can be directly manipulated by any client. The clients to suit their needs.

The concrete visitor classes are usually in different packages created by system without rewriting/recompiling any of the existing code.

Any client can develop any concrete visitor to add on to the existing package to the client.

AllList, EmptyList, NELList, Persistent, and ListFactory in one deliver the developer of this List/Visitor framework would deliver.

In practice, the developer of this List/Visitor framework would deliver for any concrete visitor to use them.

- As a consequence, AllList and all of its subclasses must be public in

- Each of the visitor's methods explicitly specifies what concrete subclass

Software Engineering Issues (cont.)
methods and thus violates this principle.

In the proceeding version of the polymonial/visitor framework, the visitor interface requires a specific concrete subclass of Abstract for each of its

• It is good software engineering practice to program at the highest level

Software Engineering Issues (cont.)
abstract host.

abstraction by making the visitor interface depend only on the

* We can promote the standard visitor pattern to a higher level of
  * They need not know about the concrete subclasses of ArrayList.
  * And since the visitors only deal with the public methods of the host,
    * ArrayList.

and NELList have the same public methods as their abstract superclasses

- We can achieve this goal because in our current design, EmptyList
  - ArrayList without changing any of the clients' code.

- This will allow us more flexibility in modifying our implementation of

and made package private.

the clients: EmptyList and NELList should be hidden from the clients.

- We would like to hide more of the details of the implementation from

Software Engineering Issues (cont.)
made by the proper concrete subclasses, reducing code complexity.

- Polymorphism will ensure that, at run time, the proper calls will be

  Everything else remains the same.

- NESTED objects only.
  2. Object forNonempty(AllList host, Object input) to act on
    EmptyList objects only, and
  1. Object forEmpty(AllList host, Object input) to act on

  Implementations to require AllList as a host instead:
  Iterator and the corresponding method signatures of all of its concrete

- The only change we need to make is to redefine the visitor interface

A Variant of the Visitor Pattern
code

the implementation of \texttt{AllList} without affecting any of the existing client classes \texttt{AllList} to all of its clients, in particular its visitors, we can change

By hiding the details of implementation and exposing only the abstract

\textbf{A Variant of the Visitor Pattern (cont.)}