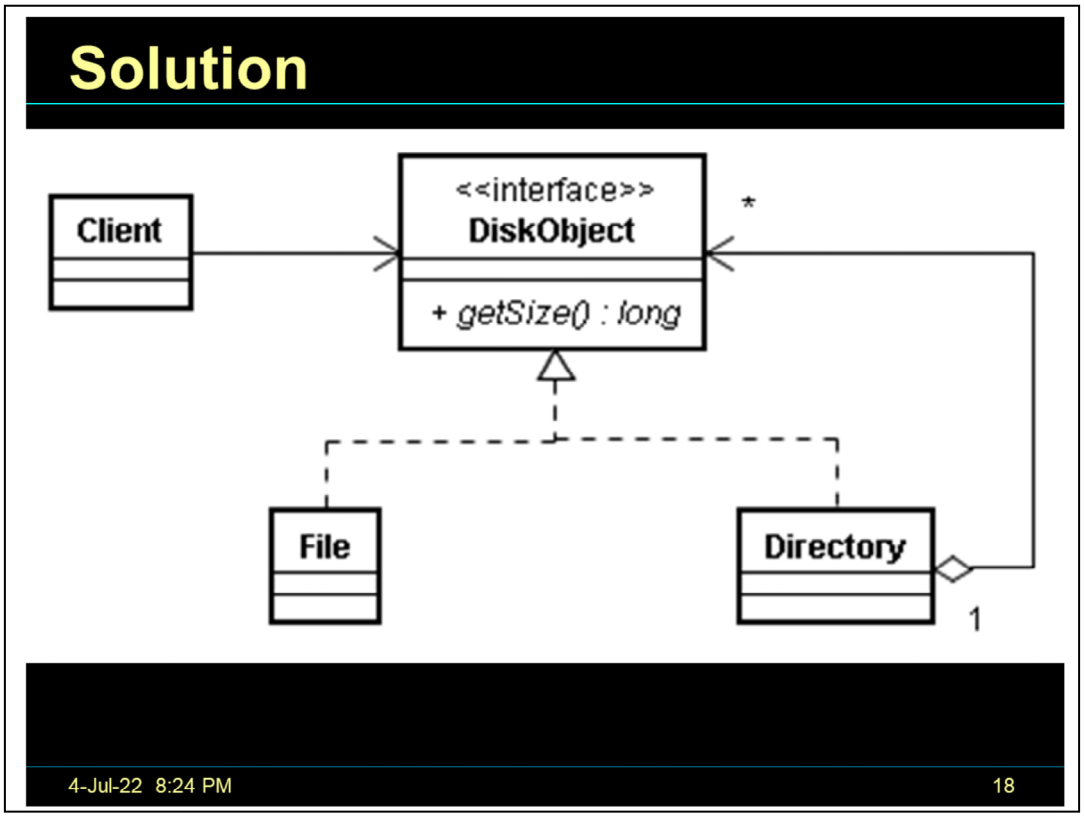
Composite DP

Problem

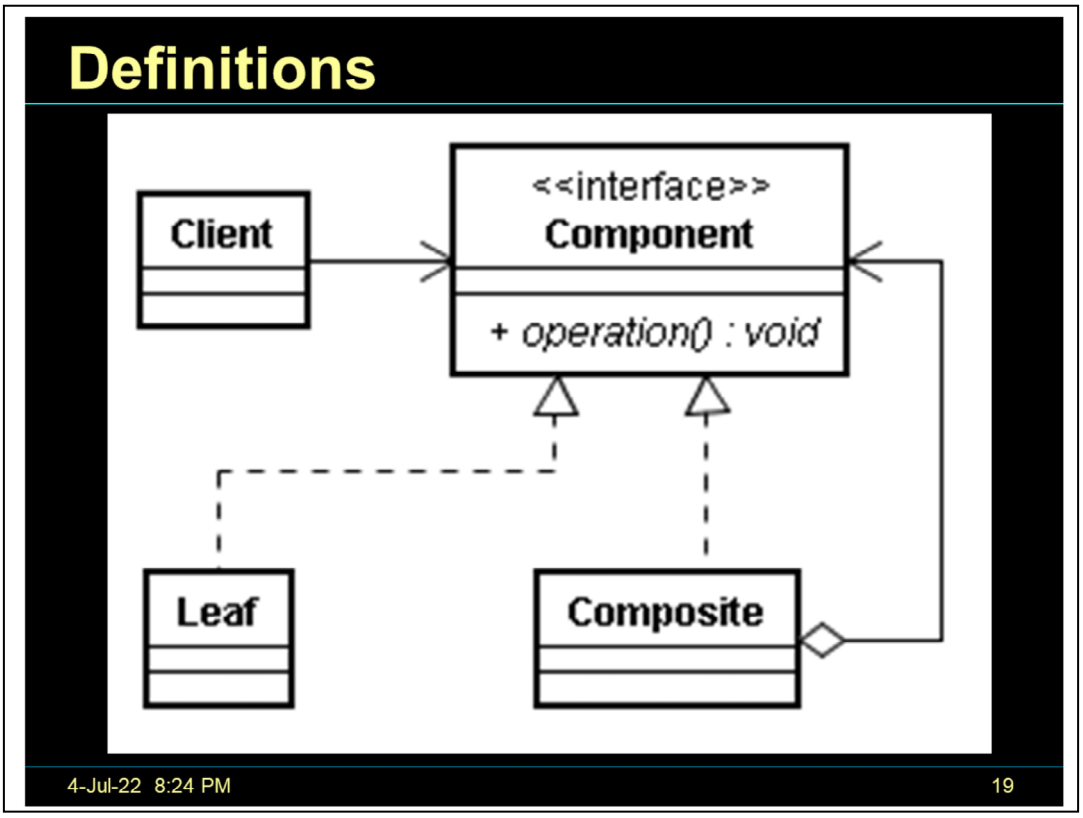
- We have a directory. The directory can contain many files and many directories.
  - Each sub-directory can again contain other files and directories.
  - Design classes to get the size of file or directory.
  - If we call the "getSize" function on a file, it should return the size of the file, but if it is for a directory then it should total the sizes of all files recursively and return the total.
  - We don't want to distinguish between files and directories, while calling the "getSize" function.



It allows us to compose objects into tree structures to represent the part-whole hierarchies.

It lets clients treat individual objects and compositions of objects uniformly.

It can call methods for composites and leaves.



We define a function “operate” in the Component1 interface.

When this function is called for a File, it will perform the required task on the file.

When this function is called for a Directory, it will perform the required task on its children and Directory itself.

The component declares the interface for objects.

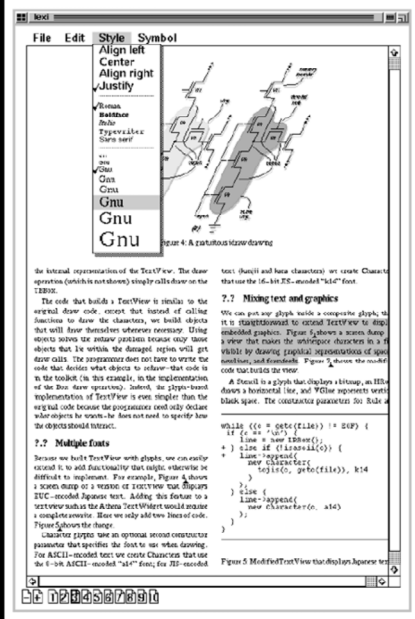
A Leaf represents primitive objects.

A leaf has no child nodes.

A Composite has a set of leaves and composite objects.

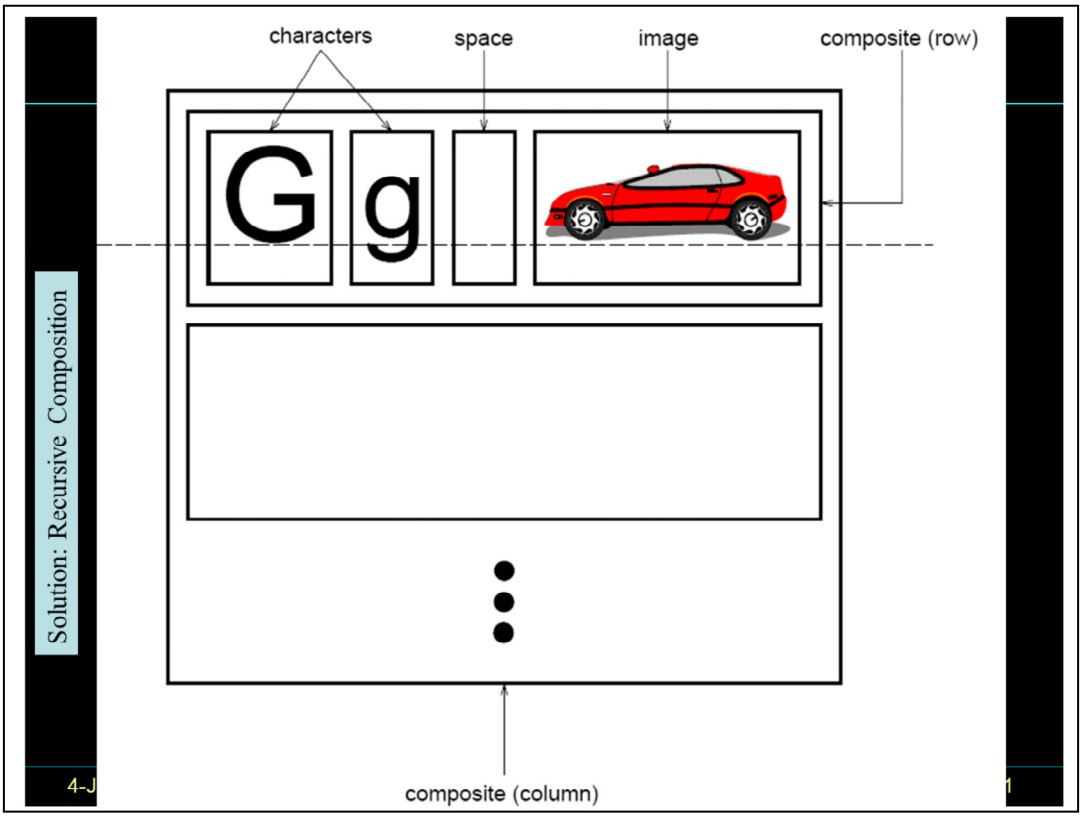
It stores child components.

Document Representation



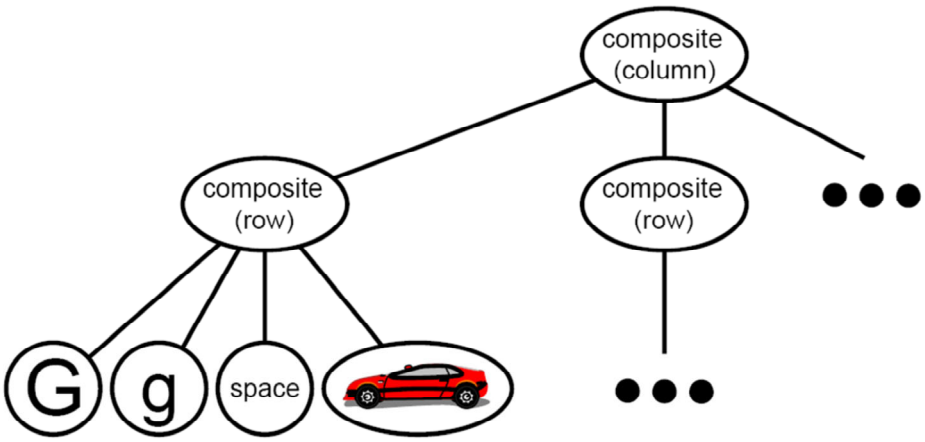
7 Design Problems:

- document structure
- formatting
- embellishment
- multiple look & feels
- multiple window systems
- user operations
- spelling checking & hyphenation

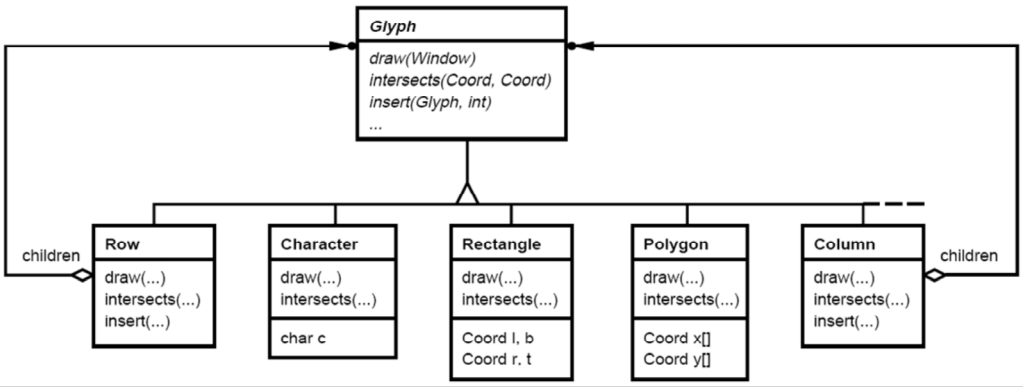

Solution: Recursive Composition
characters
space
image
composite (row)
G
g
composite (column)
4-J

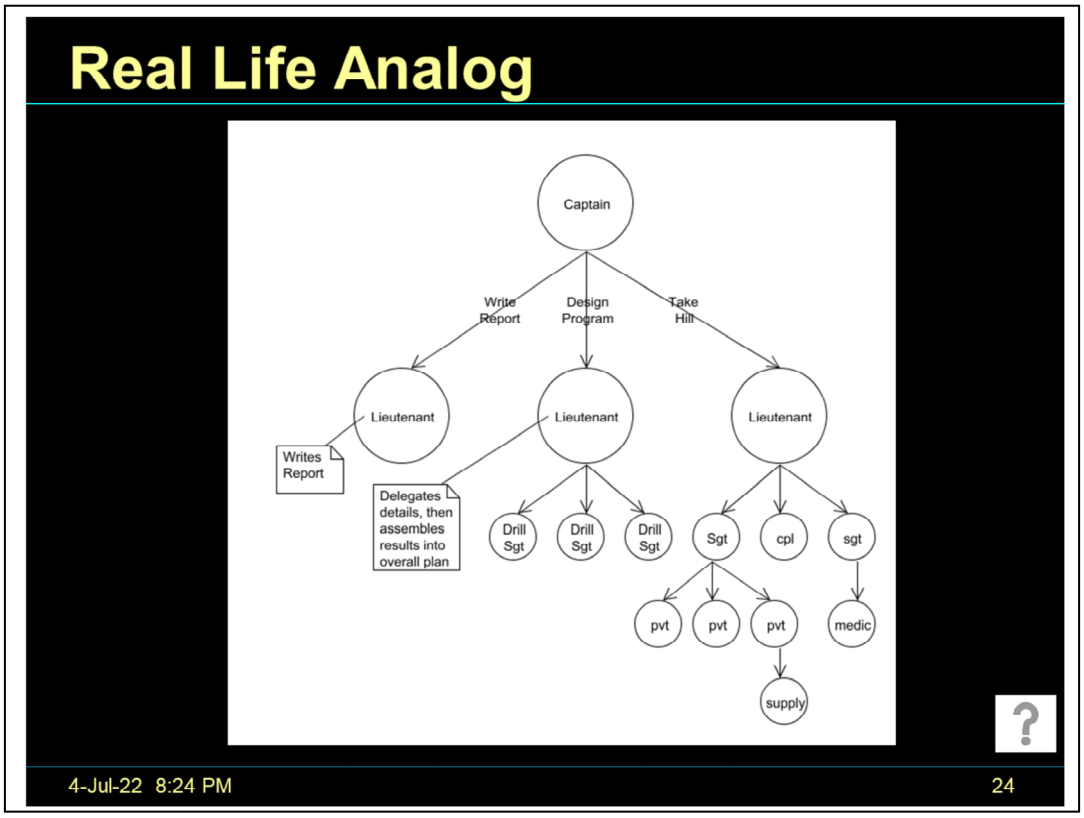
Composite DP

Object structure



Composite DP





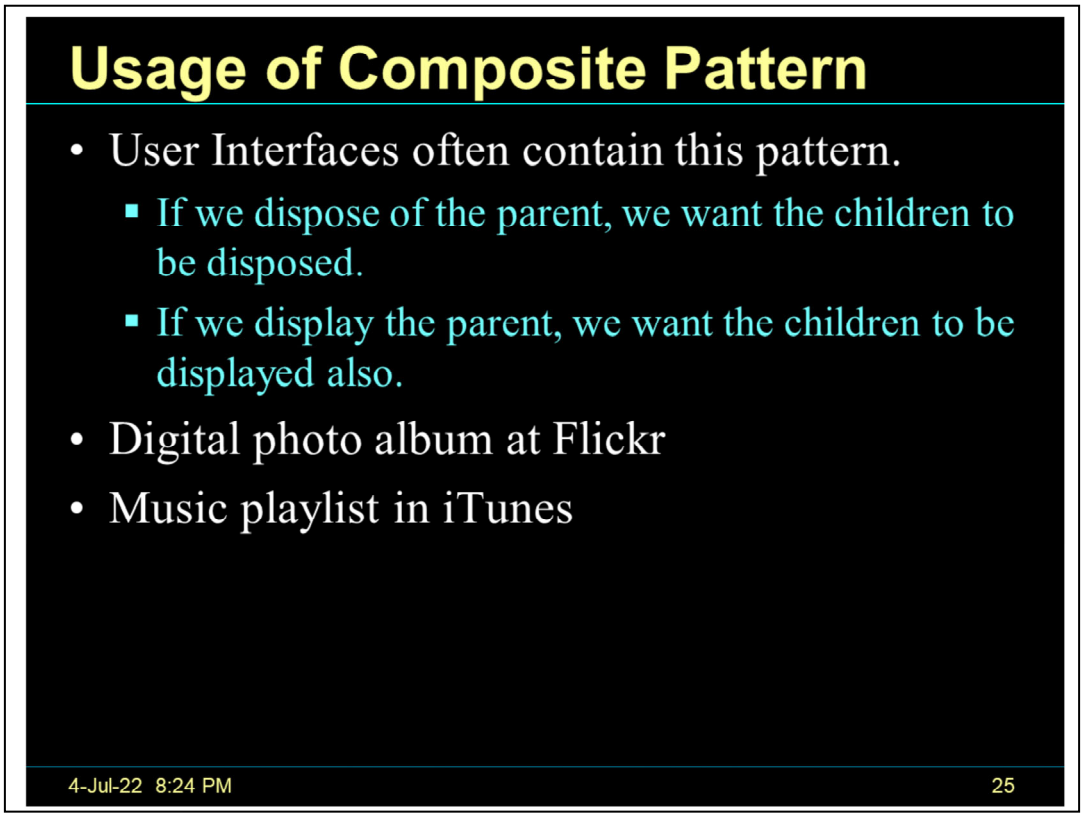
In the military, responsibilities flow from the top of a hierarchy down to the bottom in various ways. Let us assume a Captain is assigning responsibilities for a number of tasks:

She will order a Lieutenant to write up a set of reports for the pentagon. This Lieutenant will write the reports himself, with no further delegation.

She will order another Lieutenant to create a training program for the upcoming maneuvers. This Lieutenant will actually delegate the specifics to a set of Drill Sergeants, then assemble their work into a plan, which he'll report back to the Captain.

She will order a third Lieutenant to take hill 403. This Lieutenant will delegate this responsibility to a set of Sergeants, who will delegate further to Corporals, Privates and other resources (medics, an armor squad), each of which may delegate further, in a very complex arrangement

In each case, the Captain will "order" a single Lieutenant, and the variations in how these orders are accomplished will be encapsulated.



Iterator & composite:

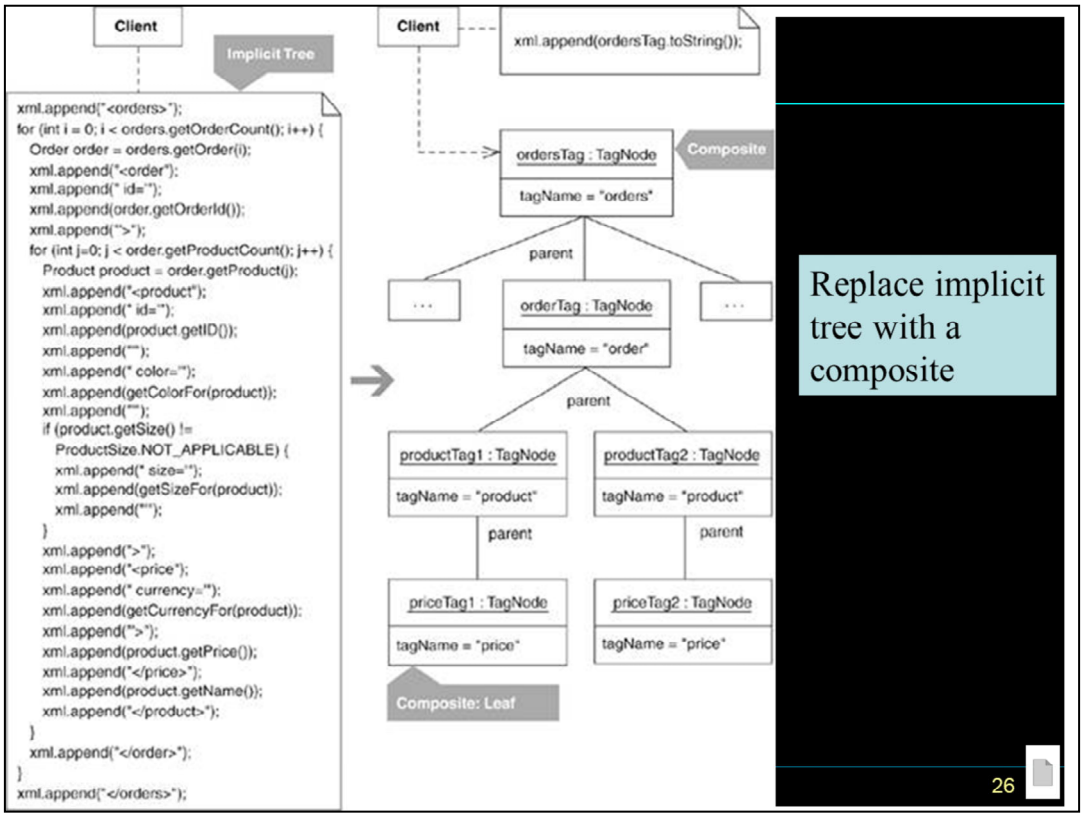
The getIterator function

in the File class returns an Iterator with one element.

for the Directory class gives an Iterator that will have all the files recursively one after another.

The getIterator function is not a part of this DP.

It is optional but is needed sometimes.



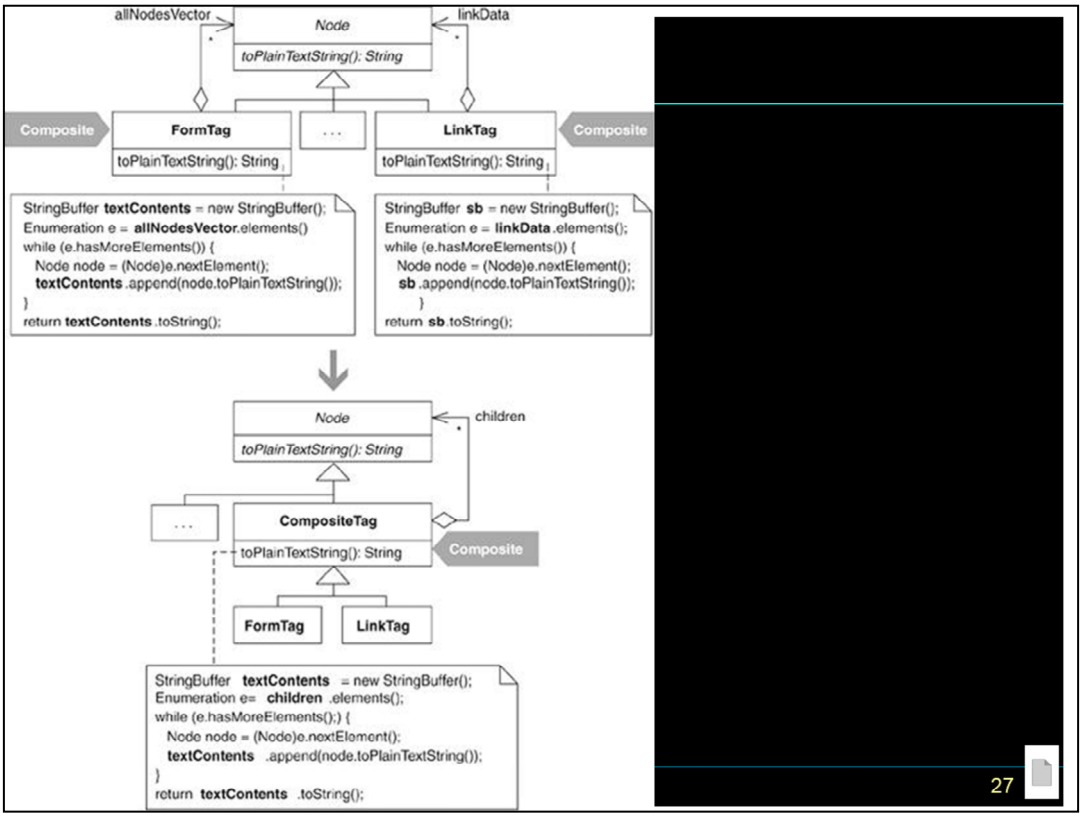
Data or code forms an implicit tree when it's not explicitly structured as a tree but may be represented as a tree. For example, the code that creates the XML data in the previous code sketch outputs values like this:

```
String expectedResult =
  "<orders>" +
    "<order id='321'>" +
      "<product id='f1234' color='red' size='medium'>" +
        "<price currency='USD'>" +
          "8.95" +
        "</price>" +
        "Fire Truck" +
      "</product>" +
      "<product id='p1112' color='red'>" +
        "<price currency='USD'>" +
          "230.0" +
        "</price>" +
        "Toy Porshe Convertible" +
      "</product>" +
    "</order>" +
  "</orders>";
```

==========

Benefits and Liabilities

+ Encapsulates repetitive instructions like formatting, adding, or removing nodes.
+ Provides a generalized way to handle a proliferation of similar logic.
+ Simplifies construction responsibilities of a client.
– Complicates a design when it's simpler to construct implicit trees.



Subclasses in a hierarchy implement the same Composite.
Extract a superclass that implements the Composite.

Benefits and Liabilities

+ Eliminates duplicated child-storage and child-handling logic.
+ Effectively communicates that child-handling logic may be inherited.

Assignment

- Q41