

## Composite Design Pattern

## Composite Design Pattern

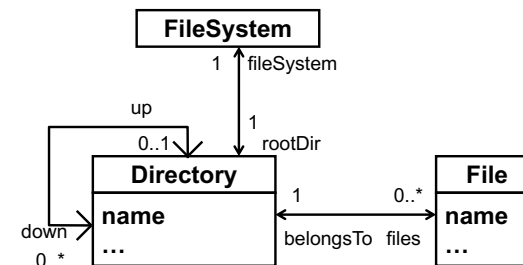
- Intent
  - Compose objects into a tree structure to represent a part-whole hierarchy.
  - Allow clients (of a tree) to treat individual objects and compositions of objects uniformly.

1

2

## A Design Exercise: File System

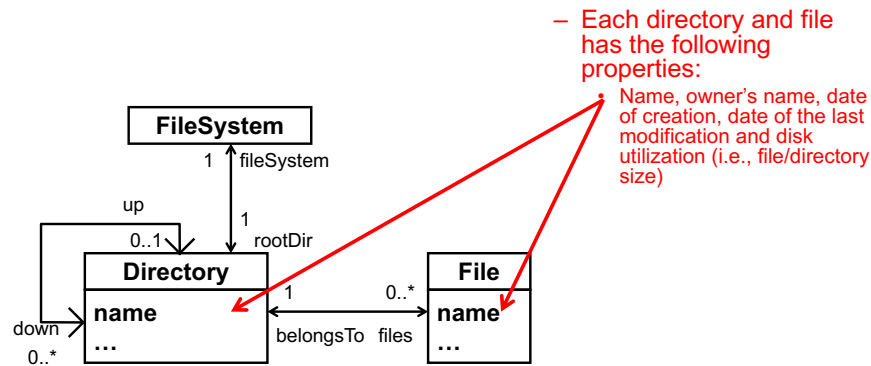
- A file system consists of directories and files.
- Each file exists in a particular directory.
- Each directory can contain multiple files.
- Directories form a tree structure.
  - Every directory has its parent directory, except the root directory.
  - Each directory can have multiple sub directories.
- Each directory and file has the following properties:
  - Name, owner's name, date of creation, date of the last revision and disk utilization (i.e., file/directory size)



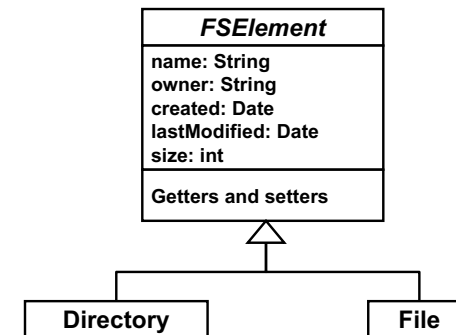
- A file system consists of directories and files.
- Each file exists in a particular directory.
- Each directory can contain multiple files.
- Directories form a tree structure.
  - Every directory has its parent directory, except the root directory.
  - Each directory can have multiple sub directories.
- Each directory and file has the following properties:
  - Name, owner's name, date of creation, date of the last revision and disk utilization (i.e., file/directory size)

3

4



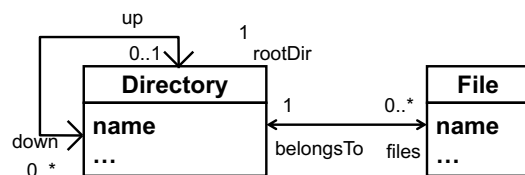
6



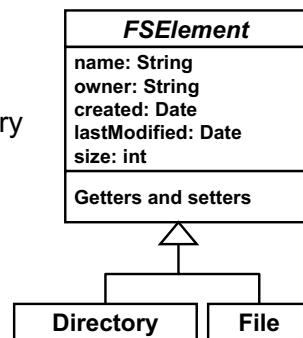
- A directory is never transformed to be a file.
- A file is never transformed to be a directory.

7

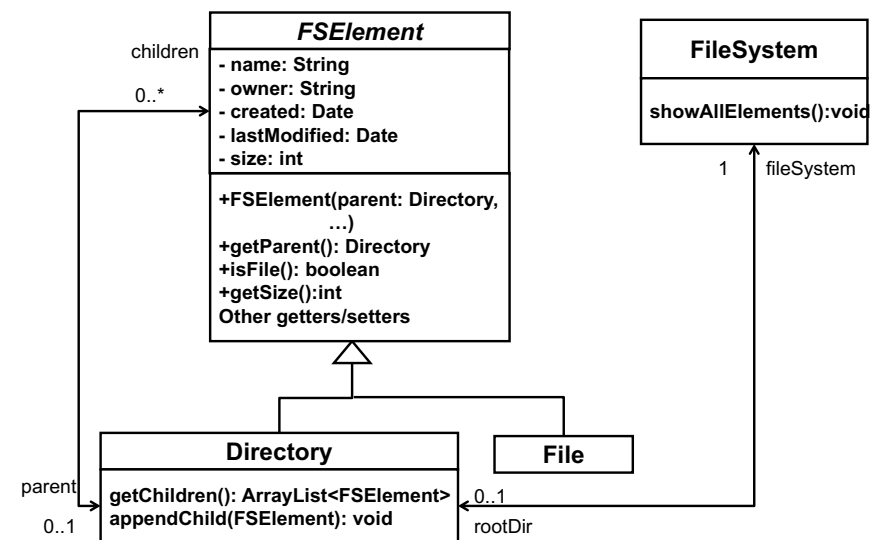
## Using Composite...



- How can we design directory-to-directory structures?
- How can we design file-to-directory structures?



8



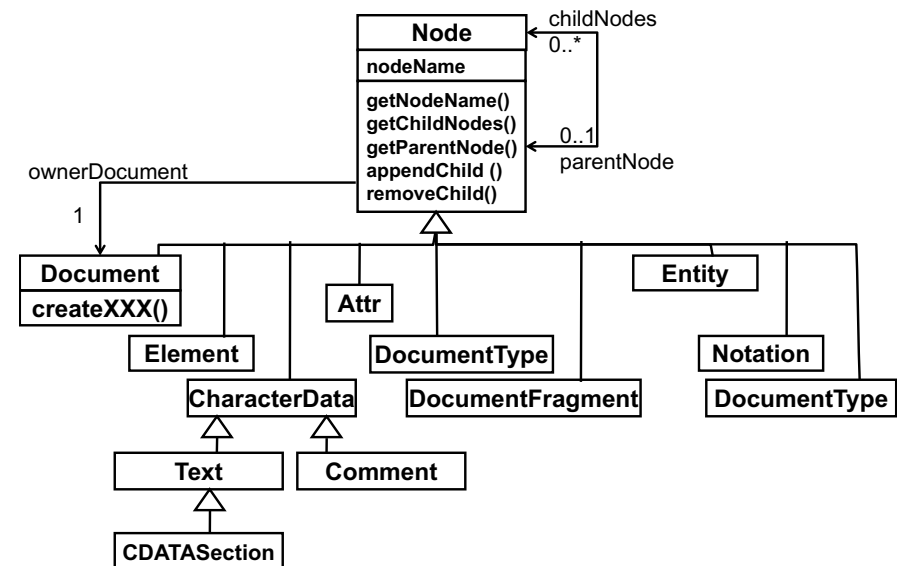
9

## Another Example: Document Object Model (DOM)

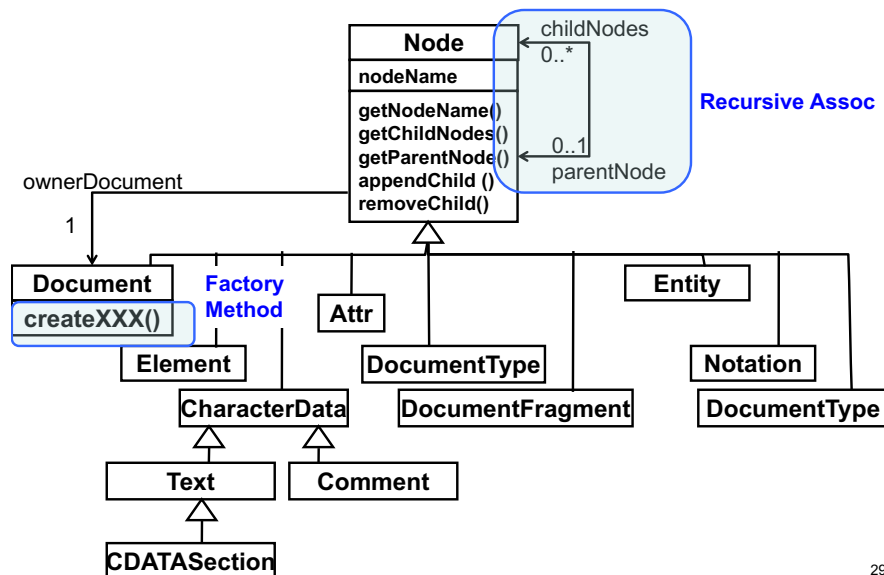
- Document Object Model (DOM)
  - A parser interface for XML parsers.
  - Specification: [www.w3c.org](http://www.w3c.org)
    - Level 1, DOM Core
  - Implementations:
    - Has been implemented by many libraries/frameworks.
    - Has been implemented by virtually all major languages.
    - e.g., Java API (javax.xml)

27

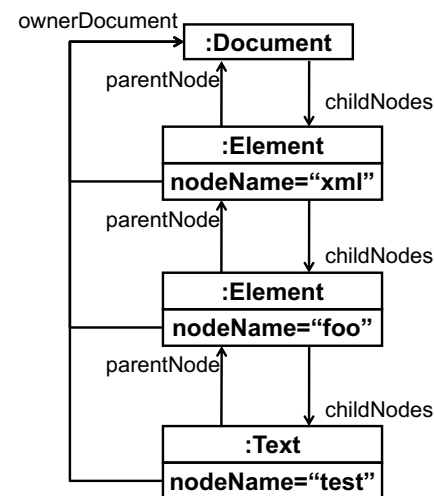
## Document Object Model (DOM)



## Example Instances of DOM Classes



29



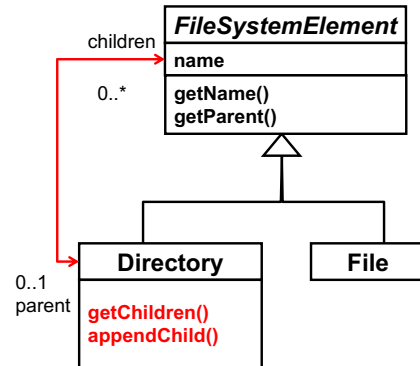
```

<xml>
  <foo>test</foo>
</xml>
  
```

30

## Two Variants of Composite

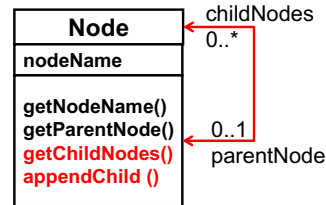
### Variant #1



Directory encapsulates child-handling methods.

File does not have child-handling methods.

### Variant #2

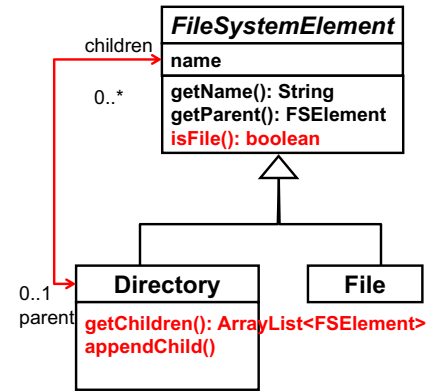


Node can represent directories and files.

Both directories and files have child-handling methods.

31

## Variant #1

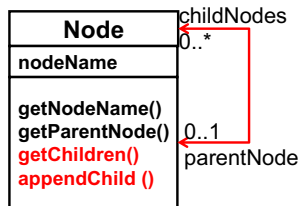


Directory encapsulates child-handling methods. File does not have child-handling methods.

- Pros
  - No need to implement unnecessary (i.e., child-handling) methods in File.
  - Type safe
- Cons
  - User/client code has to be aware of the type of an FSElement (Directory or File).
    - Need downcasting.
  - Iterator itr = aDir.getChildren().iterator(); while( itr.hasNext() ){ FSElement elem = itr.next(); if( !elem.isFile() ){ ((Directory) elem).getChildren(); } else{ ... } }
  - User/client code has to be modified when new subclasses are added.

32

## Variant #2



Node can represent directories and files.

Both directories and files have child-handling methods.

- Pros
  - User/client code does not have to know the type of a file system element.
    - No need to do downcasting.
  - Iterator itr = aNode.getChildren().iterator(); while( itr.hasNext() ){ Node elem = itr.next(); elem.getName(); elem.getChildren(); } // No if statement here.
  - User/client code can be intact even if new types of nodes are added.
  - Less number of classes
- Cons
  - Need to implement child-handling methods in a relatively ugly way.
    - Generate an error.
    - Throw an exception.
  - Lower modularity and lower type safety
  - Many methods/variables in a single class

33

## Which Variant to Use?

- How often do you expect to change the structure of classes?
  - Adding and removing subclasses of FSElement?
  - Often
    - Variant #2 would make more sense.
      - User/client code can be independent from the changes in subclasses.
  - Rare
    - Variant #1 would make sense too.
      - More type safe.
      - Less needs for error handling.

34

## A Design Decision/Rationale in DOM

- Class structure may often change as the DOM specification evolves.
  - The structure of XML documents can change/evolve independently from DOM's API design.
    - due to future updates in the XML specification.
  - Backward compatibility is important for user/client code.
- DOM designers chose variant #2 .

35

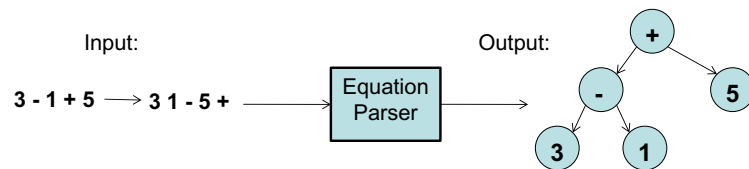
## A Design Decision/Rationale in FS

- The variety of subclasses is very limited.
  - Changes are rare on those subclasses.
- Variant #1 makes more sense.

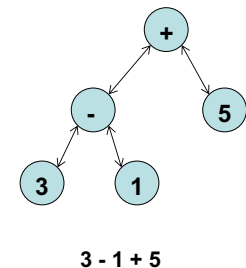
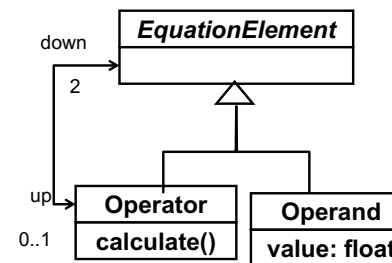
36

## An Example of Composite

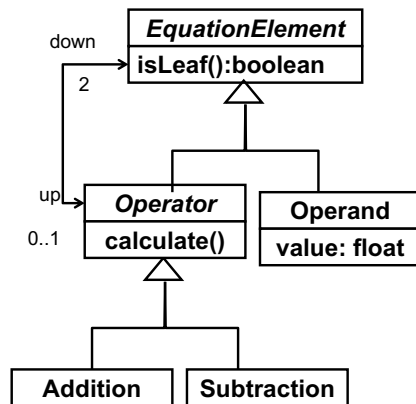
- Assume you are coding a parser to parse equations in a textual form.
  - Input: textual/string representation of an equation
  - Output: equivalent in-memory representation
    - Tree structure
      - its leaf nodes represent operands
      - the root and intermediate nodes represent operators.



37



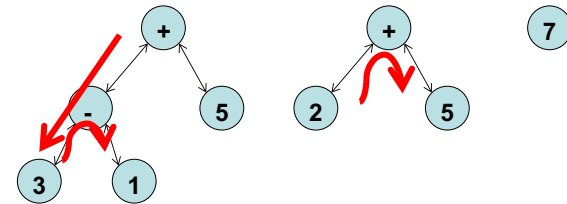
38



- Conditional statements can be eliminated in `calculate()`

39

- This parser requires a depth-first traversal policy.
  - Starts with the “deepest” and “left-most” leaf node
  - Traverse all nodes in the same layer
  - Goes up to a higher layer



40