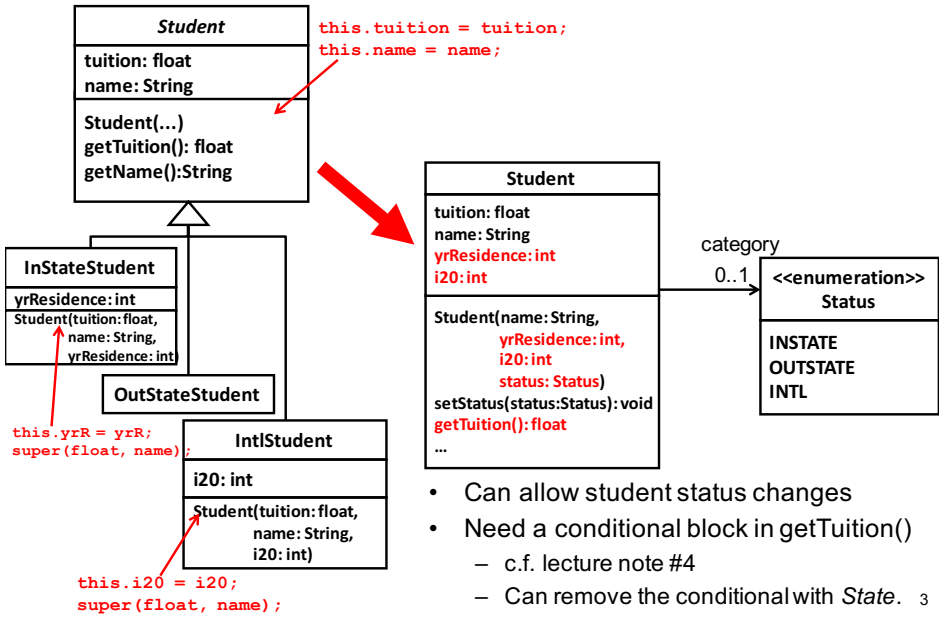
State Design Pattern

State Design Pattern

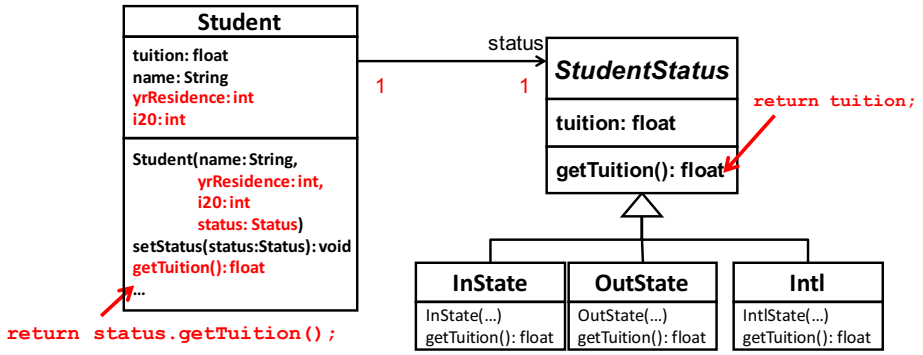
- Intent
  - Allow an object to change its behavior according to its state.

Eliminating Class Inheritance



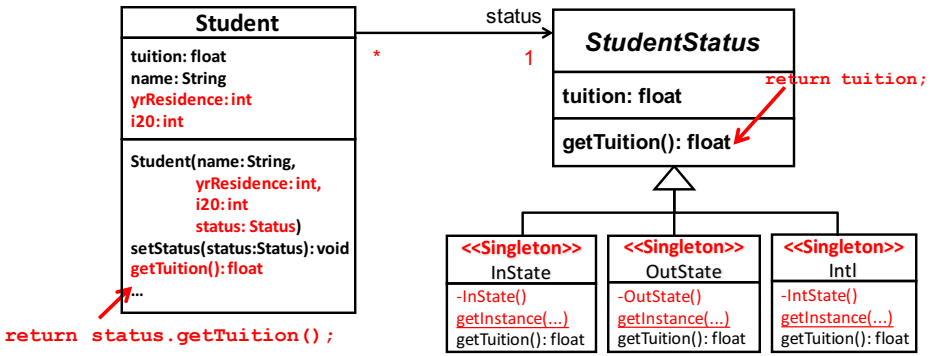
- Can allow student status changes
- Need a conditional block in getTuition()
  - c.f. lecture note #4
  - Can remove the conditional with *State*.

Using *State*



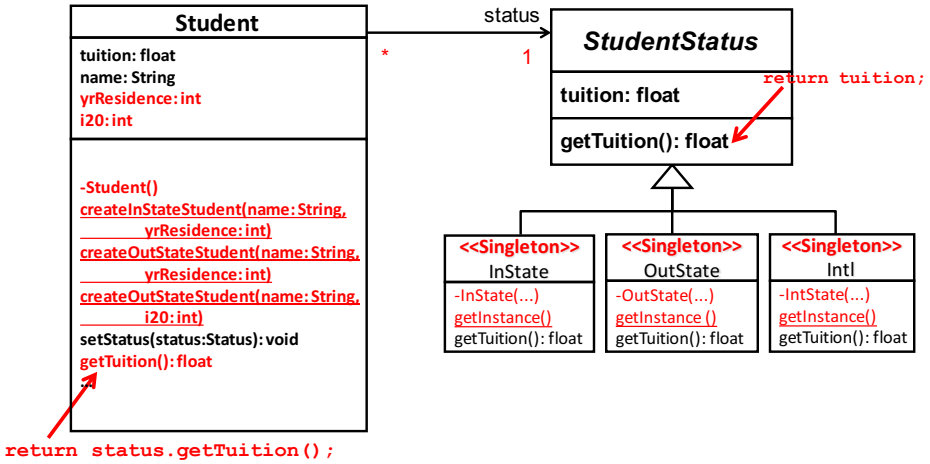

```
Student s1 = new Student( ..., new InState(...) );
s1.getTuition();
```

State Classes as Singleton




```
Student s1 = new Student( ..., InState.getInstance(...) );
s1.getTuition();
```


Adding Static Factory Methods

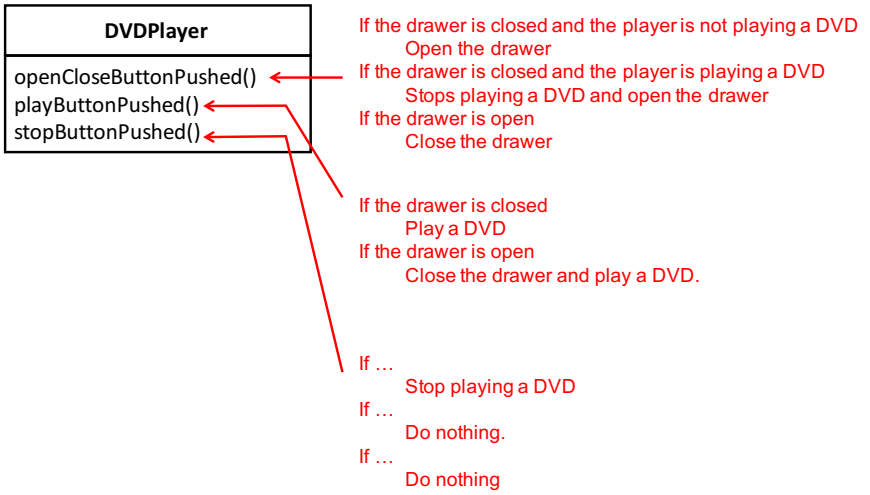



```
Student s1 = Student.createInStateStudent( "John Smith", 18 );
s1.getTuition();
```

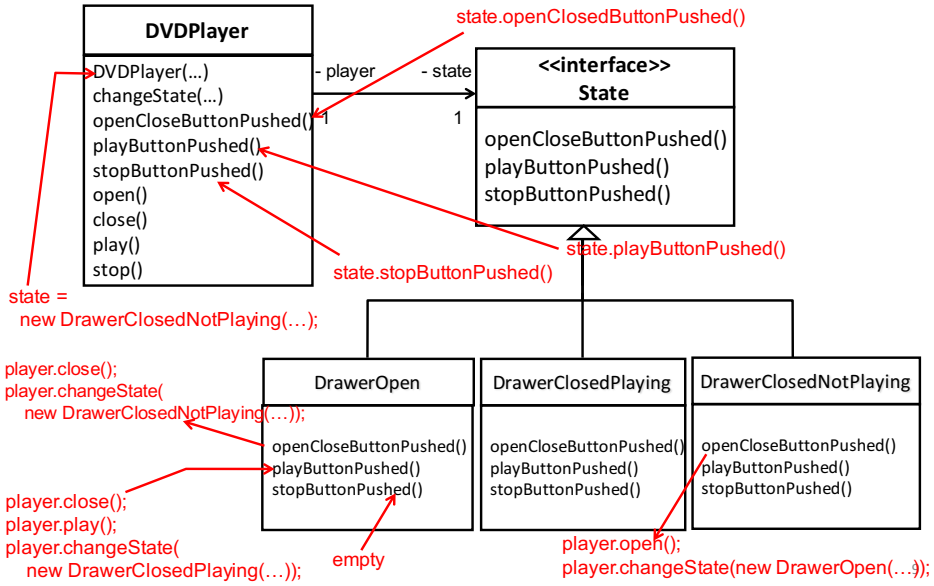

Another Example: DVD Player

- When the "open/close" button pushed,
  - Opens the drawer
    - If the drawer is closed and the player is not playing a DVD.
  - Stops playing a DVD and opens the drawer
    - if the drawer is closed and the player is playing a DVD.
  - Closes the drawer
    - if the drawer is open.
- When the "play" button pushed,
  - Plays a DVD
    - If the drawer is closed.
      - Displays an error message if the drawer is empty.
  - Closes the drawer and plays a DVD
    - If the drawer is open.
      - Displays an error message if the drawer is empty.

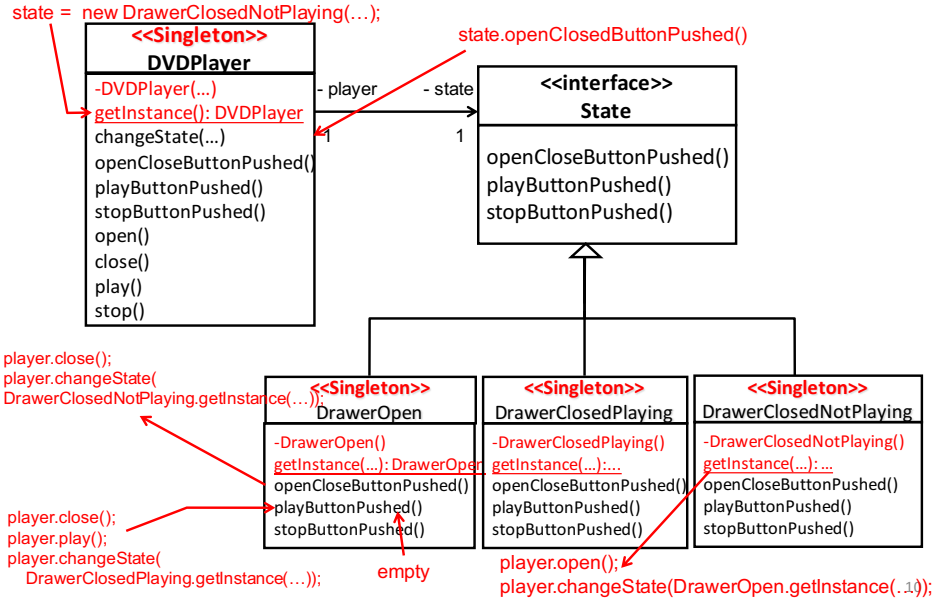
- When the "stop" button pushed
  - Stops playing a DVD
    - If the drawer is closed and the player is playing a DVD
  - Does nothing.
    - If the drawer is closed and the player is not playing a DVD.
  - Does nothing
    - If the drawer is open.



Defining States as Classes



State Classes as Singleton



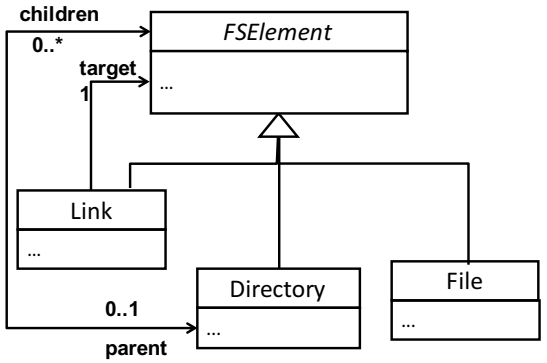
Visitor Design Pattern

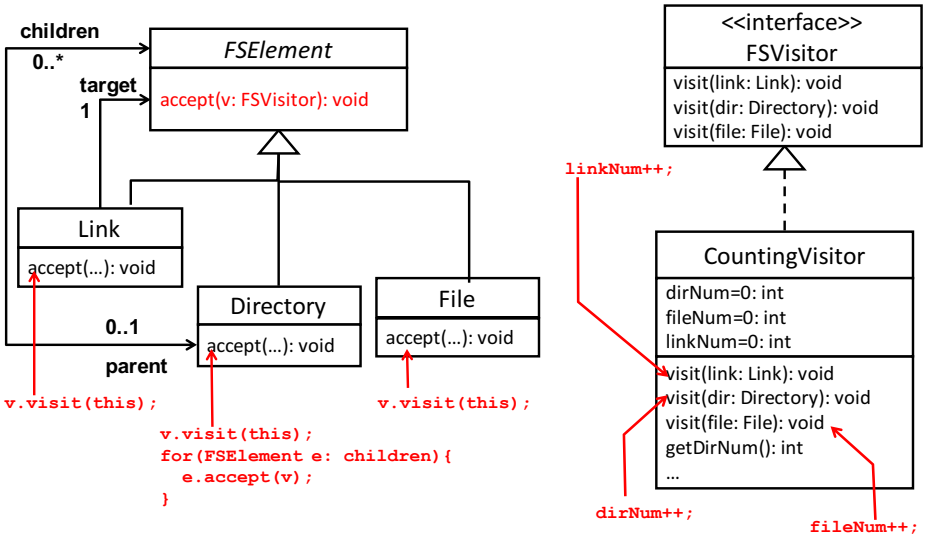
Visitor Design Pattern

- Intent
  - Decouple data structures (a set of objects) and operations (a set of operations to be performed on those objects).

File System Examples (1/4)

- Counting the number of directories, the number of files and the number links in a file system

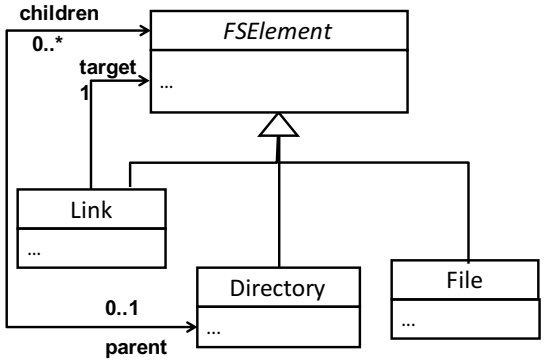


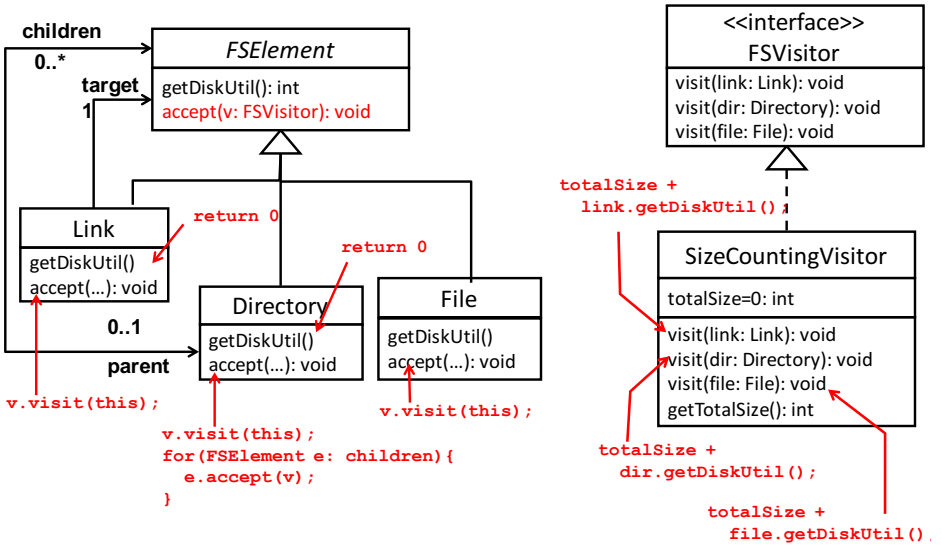



```
CountingVisitor visitor = new CountingVisitor();
rootDir.accept( visitor );
visitor.getDirNum(); visitor.getFileNum(); visitor.getLinkNum();
```


File System Examples (2/4)

- Counting the total disk utilization in a file system

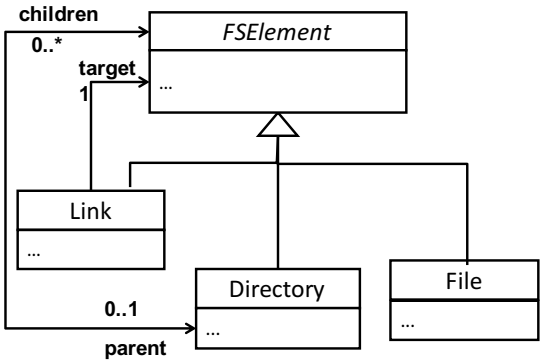


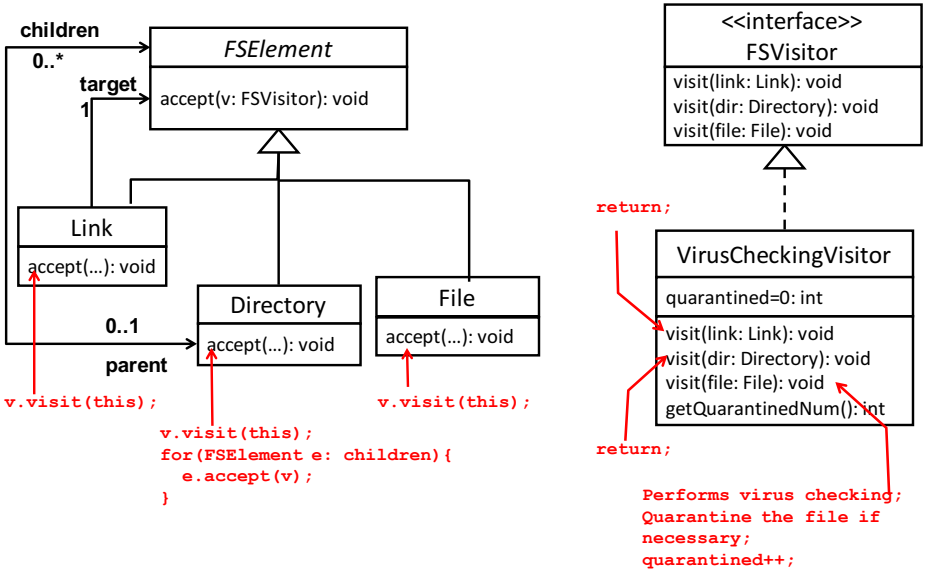



```
SizeCountingVisitor visitor = new SizeCountingVisitor();
rootDir.accept( visitor );
visitor.getTotalSize();
```

File System Examples (3/4)

- Virus checking for each file
- File indexing

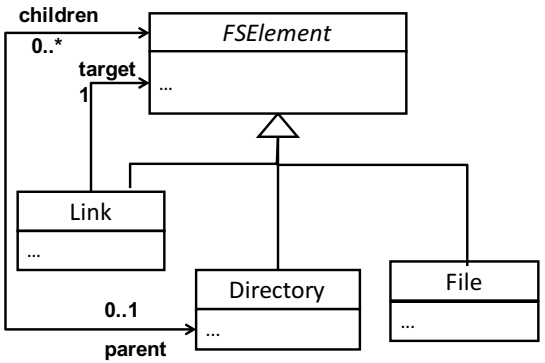


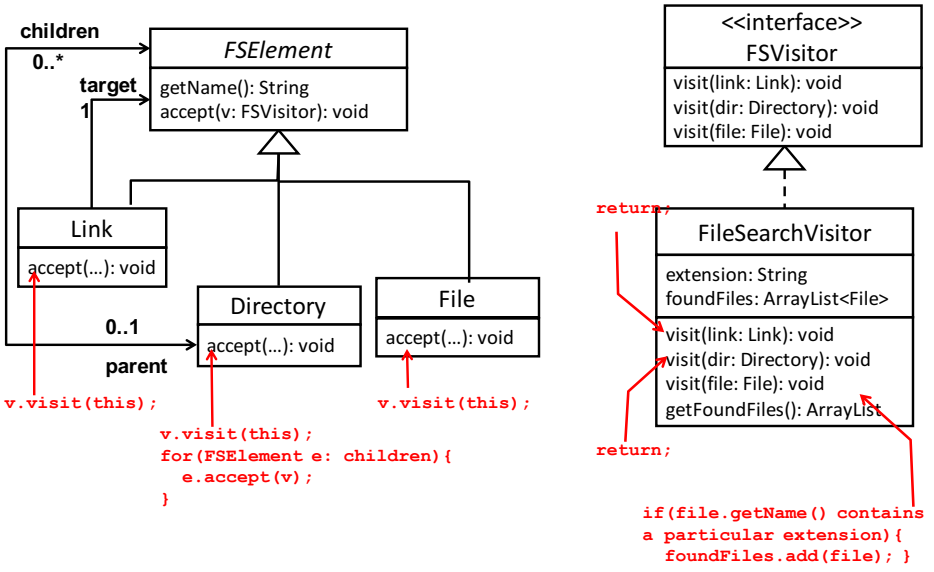



```
VirusCheckingVisitor visitor = new VirusCheckingVisitor();
rootDir.accept( visitor );
visitor.getQuarantinedNum();
```


File System Examples (4/4)

- File search
  - Searching/identifying files that have a particular extension
    - e.g., *.txt, *.jpg

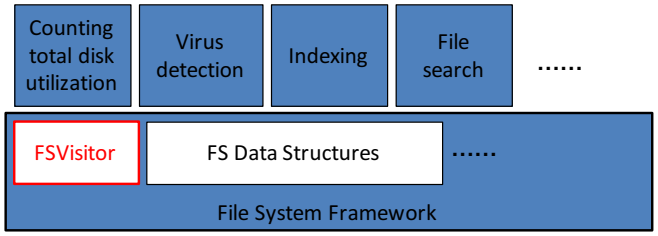





```
FileSearchVisitor visitor = new FileSearchVisitor(".txt");
rootDir.accept( visitor );
visitor.getFoundFiles().size();
```

What's the Point?

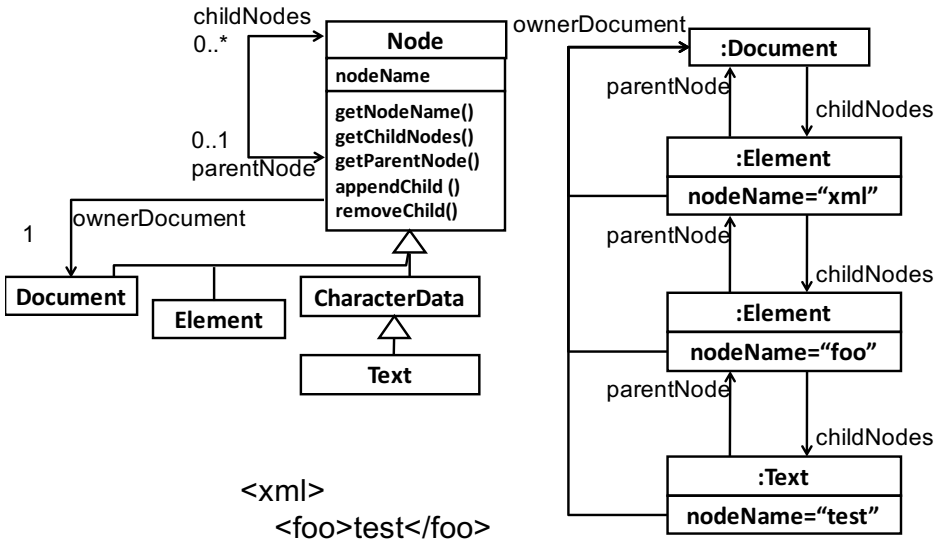
- Separating foundation data structures and the operations performed on those data structures.
  - It is easy to add, modify and remove operations.
  - Data structures can keep intact.



HW16-1

- Implement FSVisitor and three visitor classes.
  - You can choose what visitor classes to implement.

Another Example with DOM

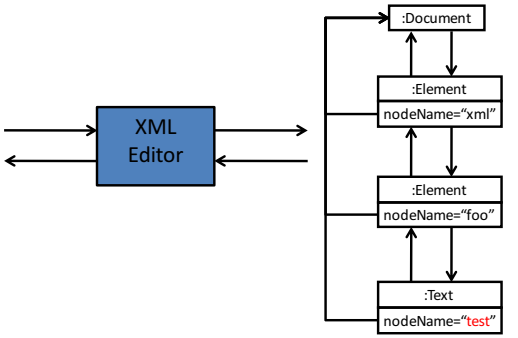



```
<xml>
   <foo>test</foo>
</xml>
```

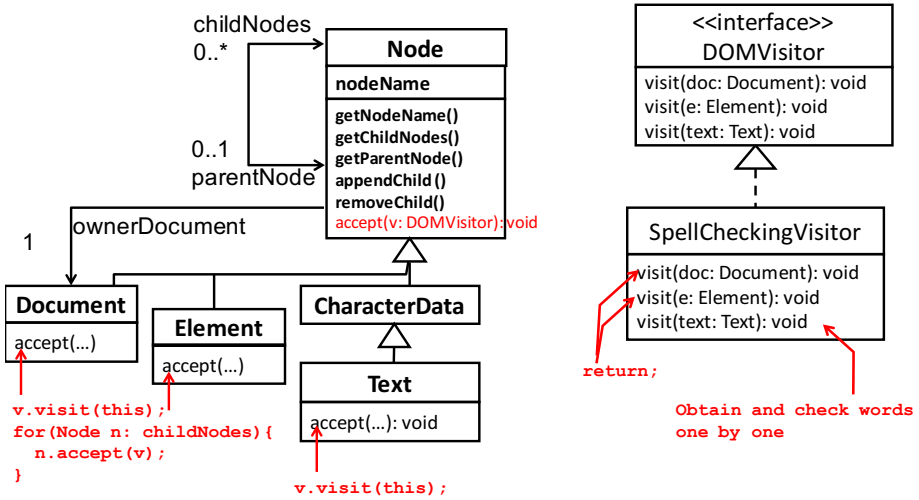

Spelling Checker in an XML Editor

- Imagine an XML editor that
  - Reads/imports an XML file, parses it and build its in-memory representation in DOM
  - Allows the user to check the spelling of each word in "Text" elements.

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

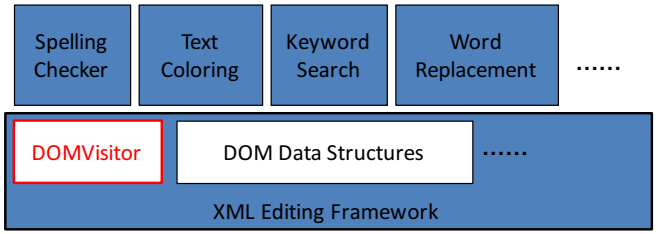


Spelling Checker as a Visitor



Other Potential Visitors

- Many other visitors can be defined.
  - Any features/operations that are applied to a set of objects.



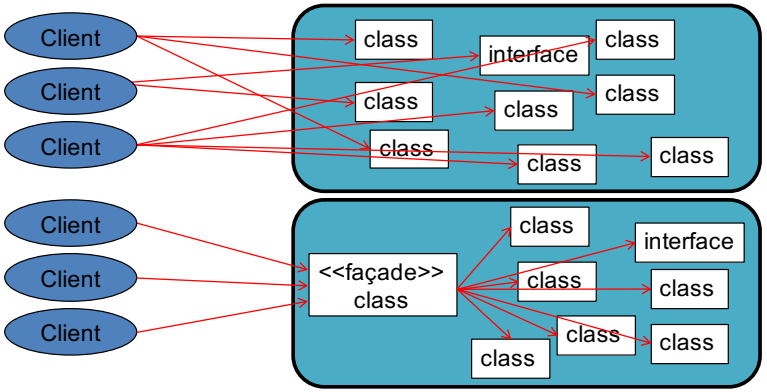
Applicability of *Visitor*

- *Visitor* can be applied to any collection of objects, not limited to *Composite*-based tree structures.
  - List, graph, etc. etc.

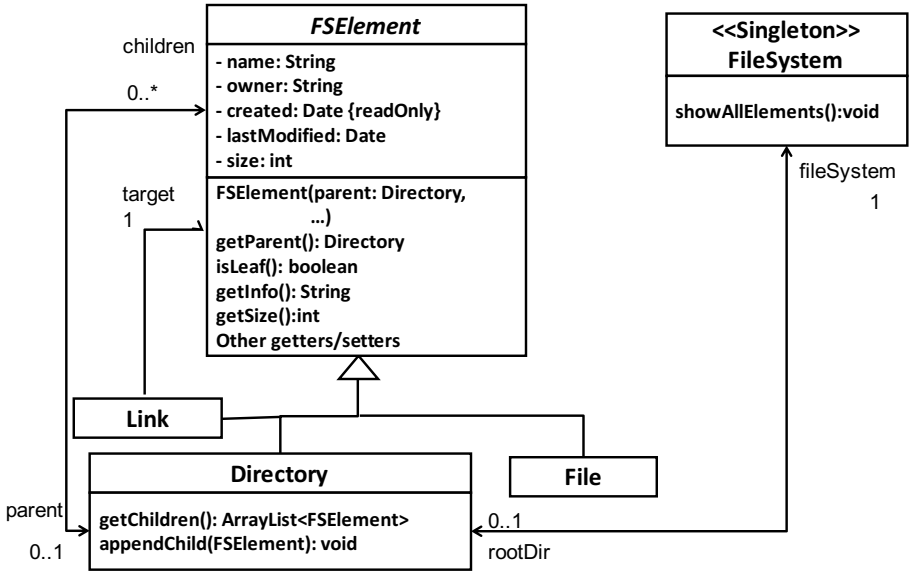
Façade Design Pattern

Façade

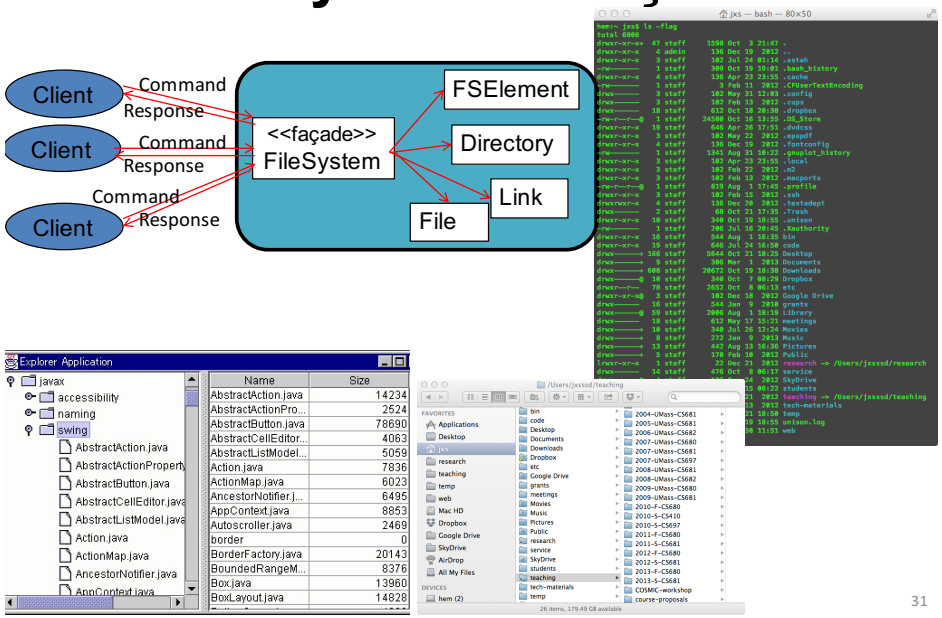
- Intent
  - Provide a unified interface (or primary point of contact) to a set of data structures (subsystems) in a system.
  - Define a higher-level interface that makes those data structures (subsystems) easier to use.



File System

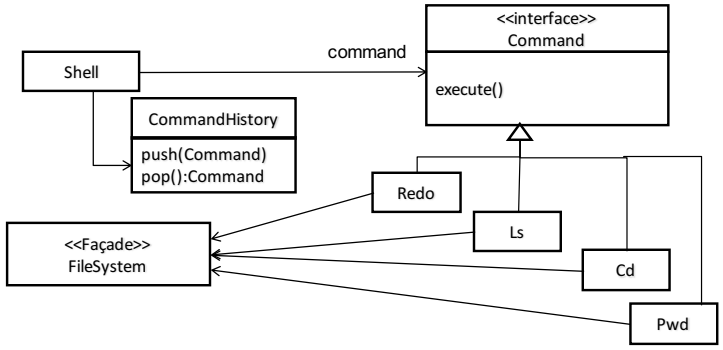


FileSystem as *Façade*

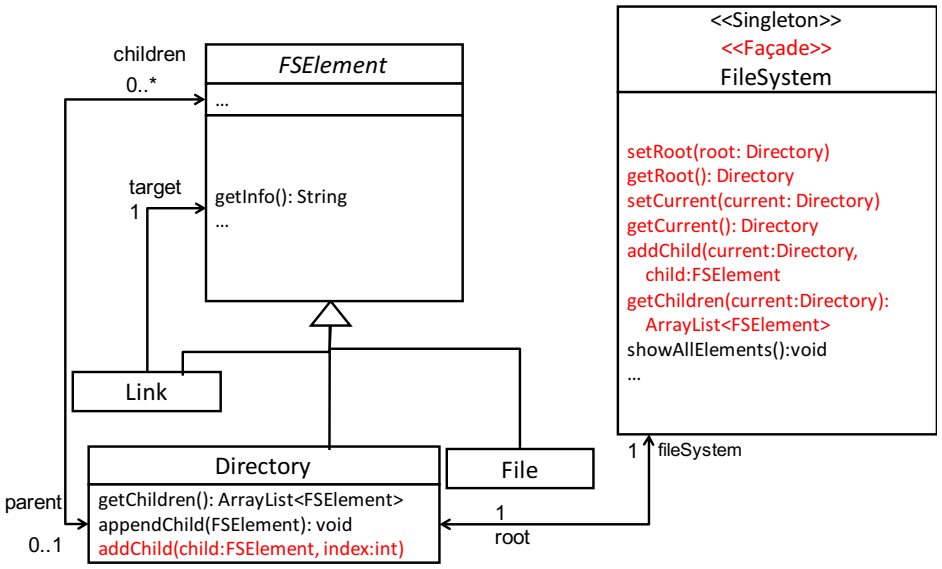


Designing FS Commands with *Command*

- There exist several (potentially many) clients for a command.
- Each command has relevant arguments/options.
- New commands are often added.
- Existing commands are often modified/updated.
- Need to record/log command history.
  - "history" command, "up" arrow



FileSystem as *Façade*



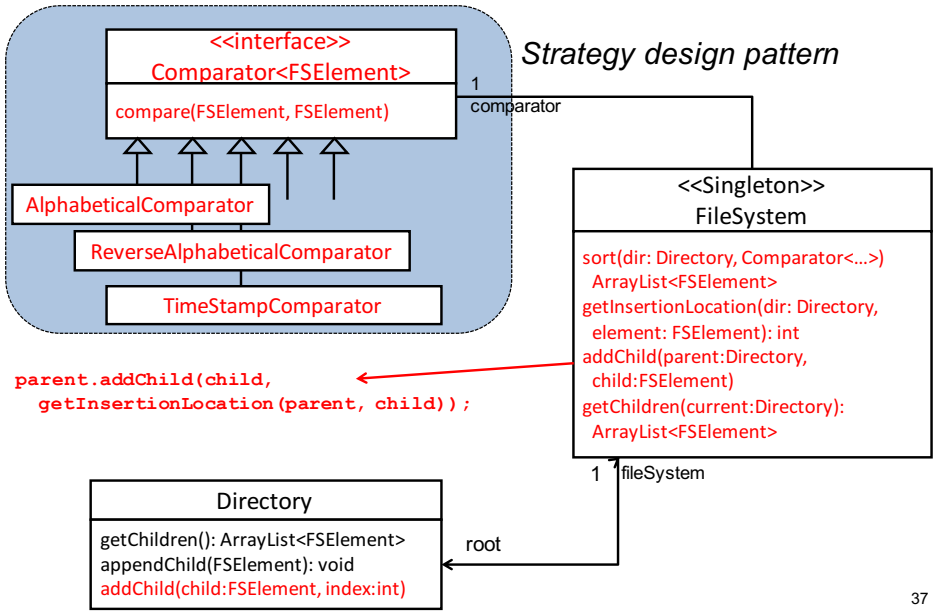
- Shell accepts the following commands, just like a Unix/Windows terminal.
  - pwd
    - Print the current working directory.
  - cd <dir name>
    - Change the current directory to the specified directory. Accept a relative (not absolute) directory name. Accept ".." (move to the parent directory of the current directory.)
  - cd
    - Change the current directory to the root directory.
  - ls
    - Print the name of every file, directory and link in the current directory.
  - dir
    - Print the information (i.e., kind, name, size and owner) of every file, directory and link in the current directory.
  - dir <dir/file name>
    - Print the specified directory's/file's information. Accept relative (not absolute) directory name. Accept ".."
  - mkdir <dir name>
    - Make the specified directory in the current directory.
  - rmdir <dir name>
    - Remove the specified directory in the current directory.
  - ln <target (real) dir/file> <link (alias) dir/file>
    - Make a link
  - history
    - Print a sequence of previously-executed commands.
  - redo
    - Redo the most recently-executed command.
  - sort
    - Sort directories and files in the current directory
  - chown
    - Change the owner of a file/directory
- Implement any extra ones as you like [optional]; e.g., mv, cp, rm, etc.
- Implement any command options as you like [optional]

- Shell
  - prints out a prompt like ">"
  - have the user enter a command, parse it,
  - instantiate a corresponding command class, and
  - call execute() on that command class instance.
- execute()
  - implements the logic of a command by calling a method(s) in FileSystem, and
    - Cd.execute()
      - Checks if the destination directory exists by calling getChildren(), etc. and moves to the destination by calling setCurrent().
      - calls setCurrent(getRoot()) if a "cd" command has no parameter.
  - returns any necessary output message to Shell.

Why is "index" introduced in addChild()?

- To sort FS elements.
  - Soring policies: alphabetical, reverse alphabetical, timestamp-based ("last-modified"-based), element kind based (dir v.s. file, file type based), etc.
- It is not a good idea to embed/couple sorting logic in/with Directory.
  - Better idea: Make Directory open-ended for various sorting policies (Decouple Directory from sorting policies)
    - Allow the FS user to select a sorting policy dynamically
    - Allow the FS developer to add new sorting policy in the future.

Soring FS Elements with Comparator



HW16-2

- Implement FileSystem as *Façade.*
- Implement individual shell commands with *Command.*
- Implement a "pluggable" soring feature with Comparator (*Strategy*).
  - addChild() always follows the default (alphabetical) soring policy.
    - getChildren() always returns alphabetically-sorted elements.
  - sort(Directory, Comparator<FSElement>) follows a custom sorting policy, which is indicated by the second parameter, and returns re-sorted elements.
    - Directory does not have to retain the re-sorted elements.
    - Implement at least one custom sorting policy (e.g., timestamp-based)

No Individual Project in CS680

- HWs only.
- Grading factors
  - Homework (90%)
  - Quizzes (10%)
    - Occasionally, at the beginning of a lecture
- HWs will be due at December 25 (Fri) midnight. No deadline extensions. No exceptions.
  - If you miss the deadline, you will receive an INC temporarily.