

Visitor Design Pattern

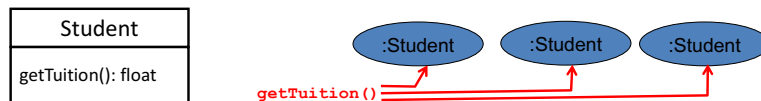
Visitor Design Pattern

- Intent
 - Separate (or decouple) a set of objects and the operations to be performed on those objects.

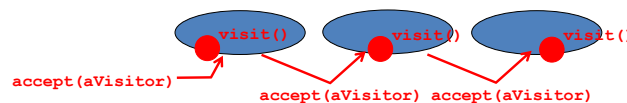
1

2

- In a traditional/normal design, if an operation is performed on some objects, it is defined (as a method) in a class(es) for those objects.



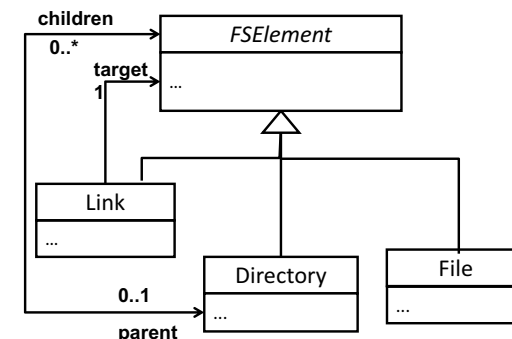
- With *Visitor*, the operation is defined in a visitor.



3

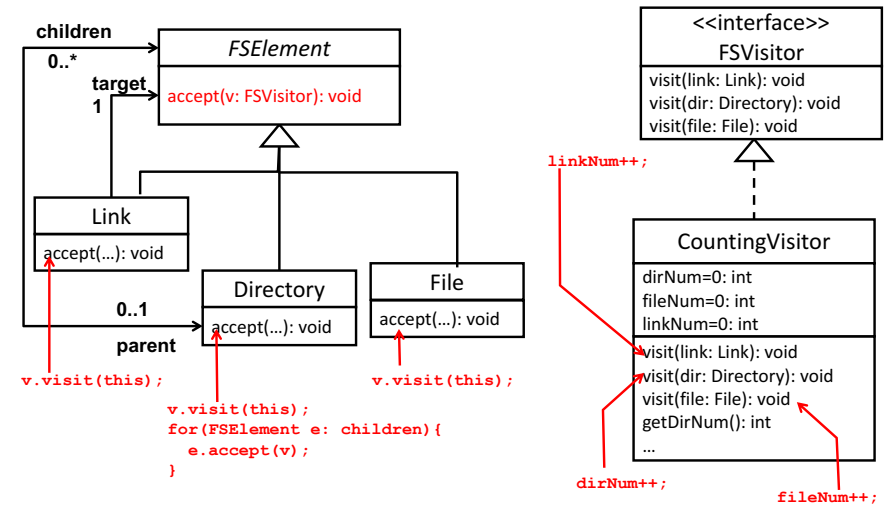
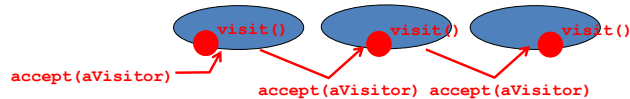
File System Examples (1/3)

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



4

- With *Visitor*, an operation to count FS elements is defined in a visitor.



```

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

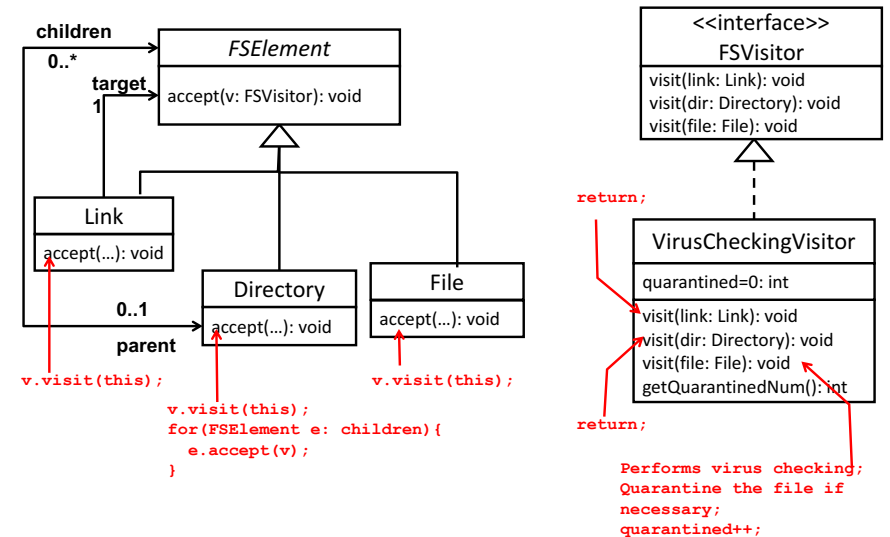
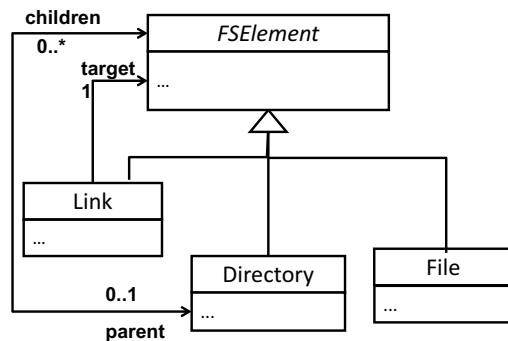
```

5

6

File System Examples (2/3)

- Virus checking for each file
- File indexing



```

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

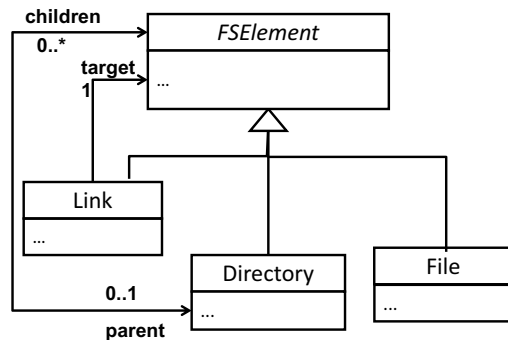
```

9

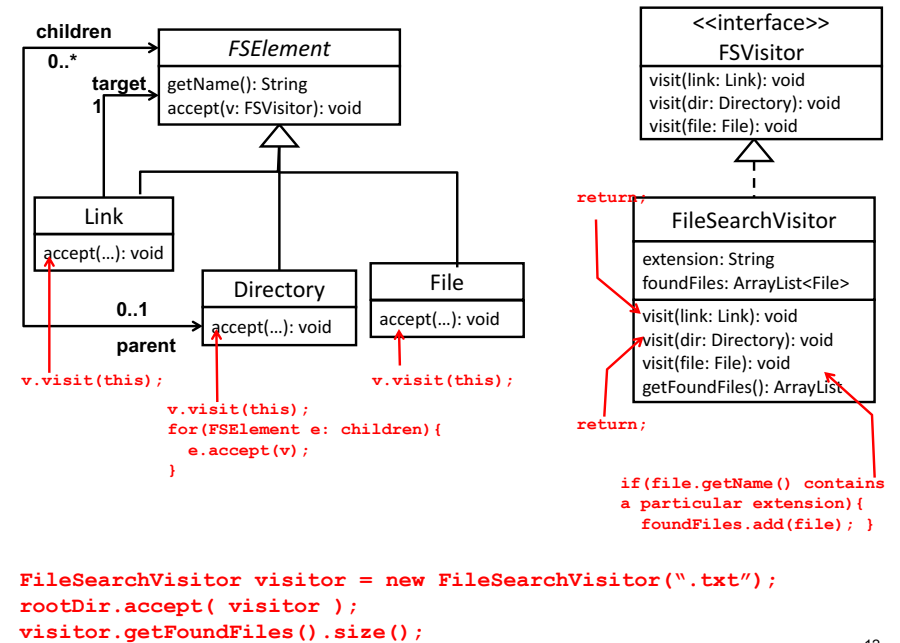
10

File System Examples (3/3)

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



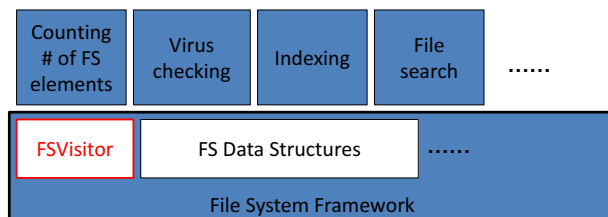
11



12

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 remain intact.



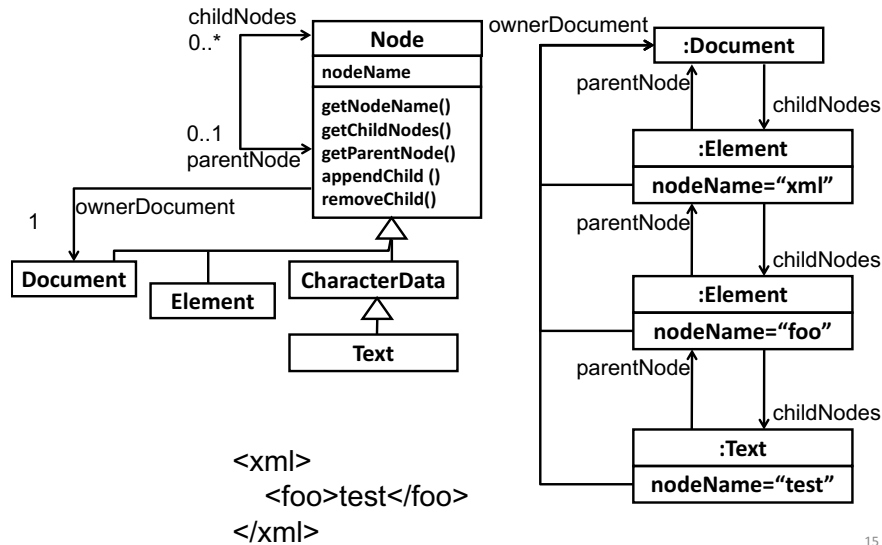
13

HW 10

- Implement FSVisor and three visitor classes.
- Due: December 24 (Sun) midnight
 - c.f. Dec 16 to 22: Final exam period

14

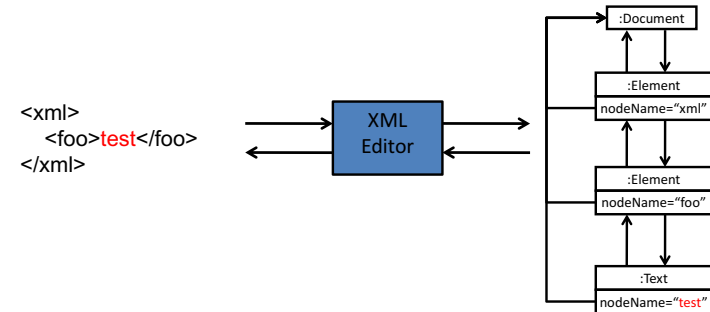
Another Example with DOM



15

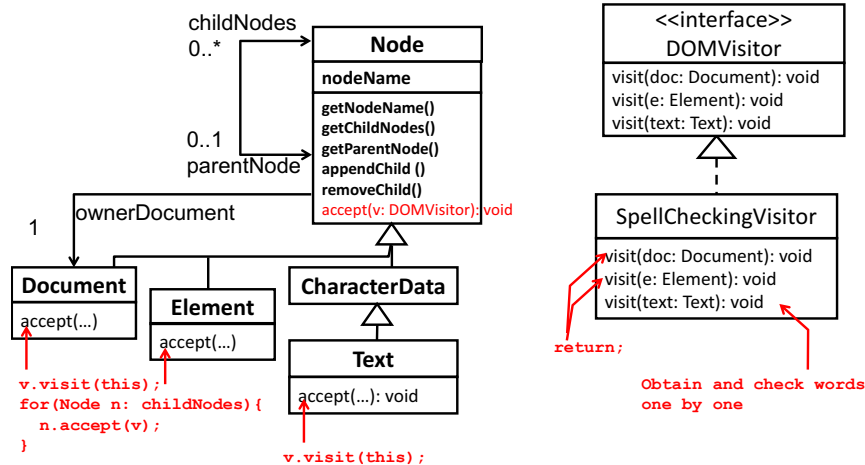
Spelling Checker in an XML Editor

- 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.



16

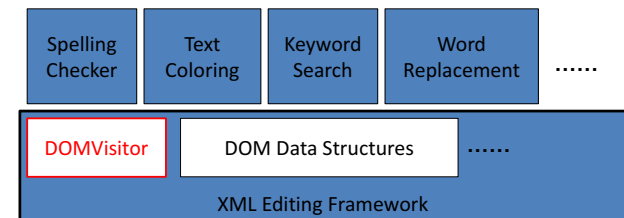
Spelling Checker as a Visitor



17

Other Potential Visitors

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



18

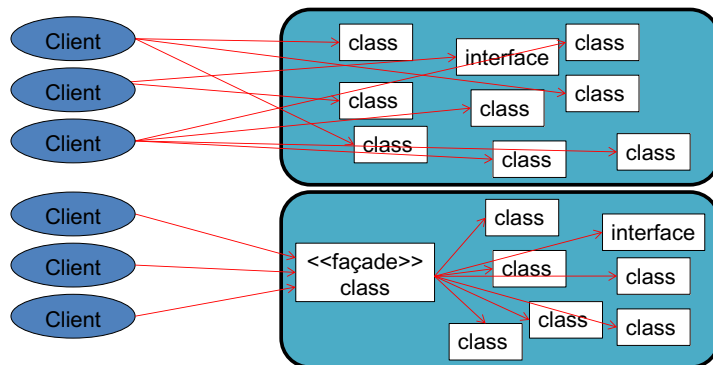
Applicability of *Visitor*

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

Façade Design Pattern

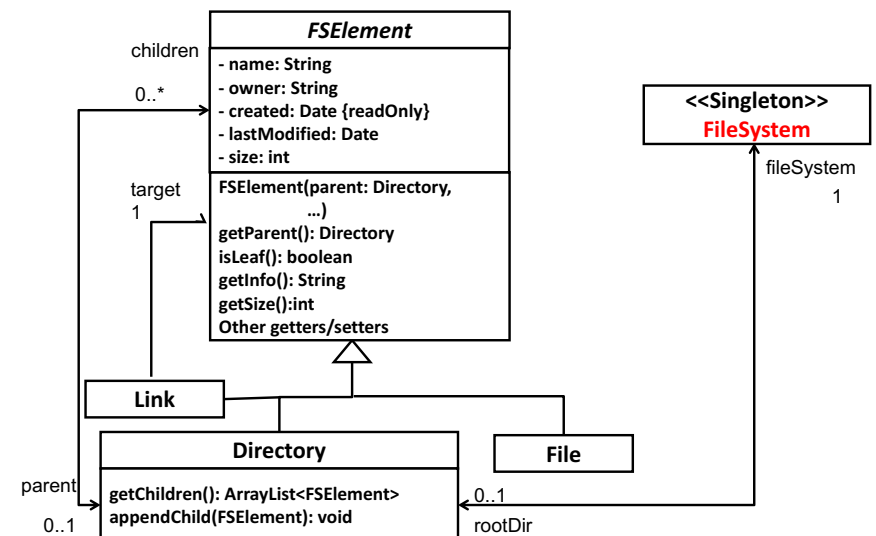
Façade

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



19

File System

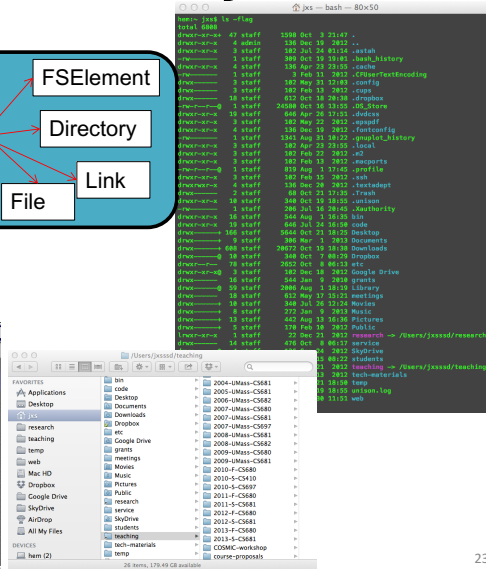
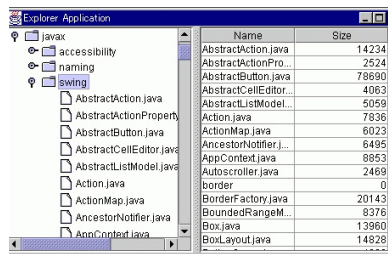
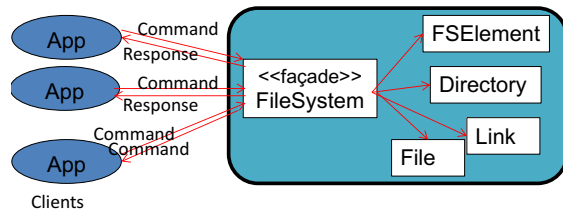


20

21

22

HW 11

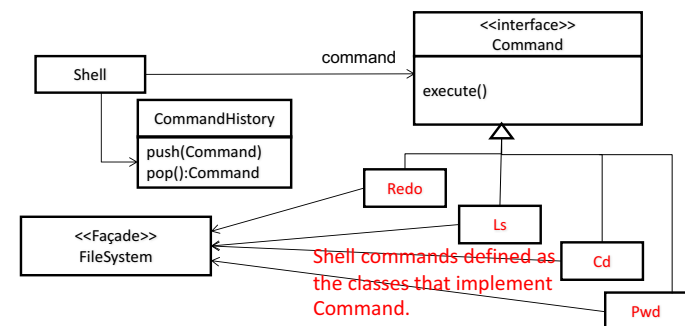


- Implement a shell for your FS elements.
 - NOT GUI shell, but CUI (character UI) shell just like a Unix/Windows terminal.
- Implement individual shell commands with *Command*.
- Implement FileSystem as *Façade*.
- Implement a “pluggable” sorting feature with Comparator (*Strategy*).

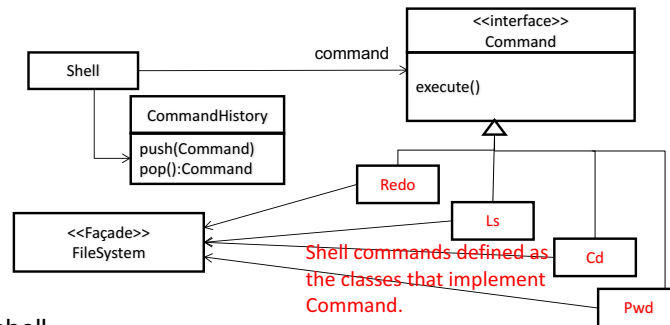
Designing FS Commands with *Command*

- Shell accepts the following commands:
 - 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
 - mv <dir/file> <destination dir>
 - Move a directory/file to the destination directory
 - cp <dir/file> <destination dir>
 - Copy a directory/file to the destination directory
 - chown
 - Change the owner of a file/directory
 - history
 - Print a sequence of previously-executed commands.
 - redo
 - Redo the most recently-executed command.
 - sort
 - Sort directories and files in the current directory

- *Why Command?*
 - There exist several (potentially many) clients/apps 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



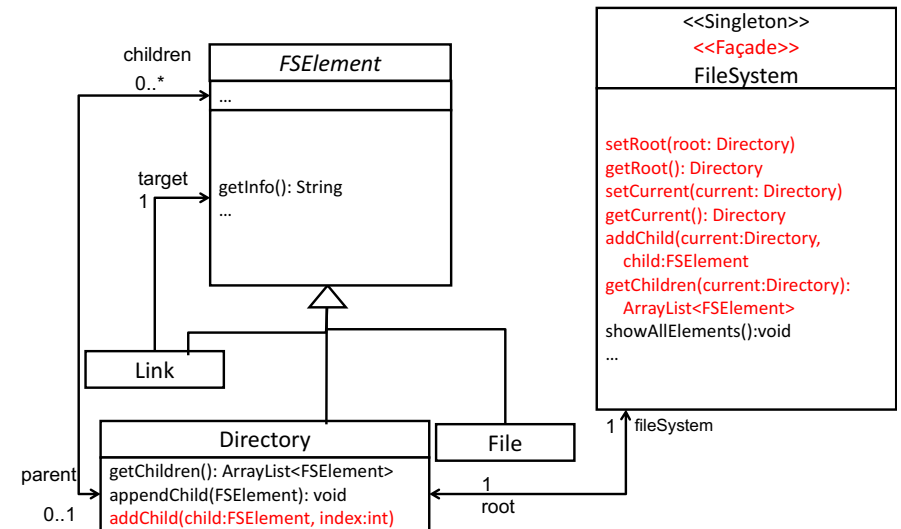
Designing FileSystem as *Façade*



- The shell
 - Receives a command (e.g. “cd” command) from the user,
 - Creates an instance of a corresponding Command class (e.g. Cd), and
 - Calls execute() on the instance.
 - execute() calls a method(s) of FileSystem.
 - Interacts with FileSystem through execute().
 - That is, FileSystem serves as Façade.

27

Example (not Complete) Methods in FileSystem



28

An Example Interaction among User, Shell and FileSystem

- The shell
 - prints out a prompt like “>”,
 - lets the user enter a command and parses it,
 - Assume the user enters “cd ...” as a command.
 - Creates an instance of Cd, and
 - Calls execute() on the instance.
- execute()
 - implements the logic of a command by calling a method(s) of FileSystem, and
 - execute() of the Cd class
 - Checks if the destination directory exists by calling getChildren(), etc. and moves to the destination by calling setCurrent().
 - calls setCurrent(getRoot()) if “cd” command has no parameters.
 - returns any output message to Shell.

29

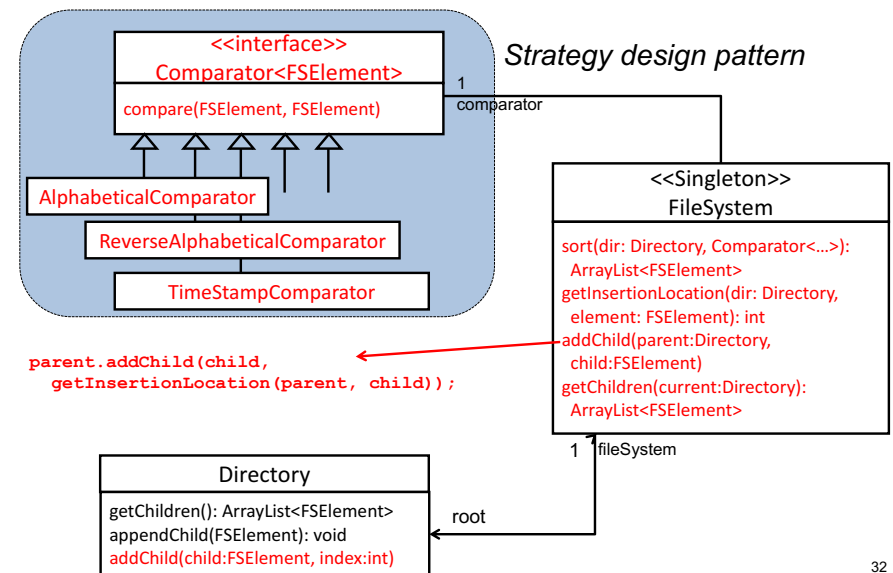
Sorting FS Elements

- Example sorting policies
 - Alphabetical
 - Reverse alphabetical
 - Timestamp-based (e.g. “last-modified date”-based)
 - Element kind based (e.g. directories listed first followed by files and links, file type based)

30

Sorting FS Elements with Comparator

- It is not a good idea to hardcode sorting logic in Directory.
 - Whenever a new sorting policy is required, you need to modify Directory.
- Better idea: Make Directory open-ended for various sorting policies (i.e., make Directory loosely-coupled from sorting policies)
 - Allow each FS user to select a sorting policy dynamically
 - Allow FS developers to add new sorting policies in a maintainable manner.
 - Have them add extra code (classes) rather than modify Directory.
- Solution: Use *Strategy* (Comparator).



31

32

Extra Commands to be Implemented

- addChild() always follows the default (alphabetical) sorting policy.
 - Directory always retains alphabetically-sorted FS elements.
 - getChildren() returns alphabetically-sorted elements.
- sort(Directory, Comparator<FSElement>) re-sorts FS elements based on a custom (non-default) 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)

33

- Support extra command that are associated with your visitors.
 - c.f. HW 9
 - e.g., count, viruscheck, search
 - Add extra methods in FileSystem to create and run visitors.
- All previous HW solutions for file system development must be integrated into a single code base.
- Unit tests are required for all classes.

34

- Due: December 24 (Sun) midnight
 - c.f. Dec 16 to 22: Final exam period