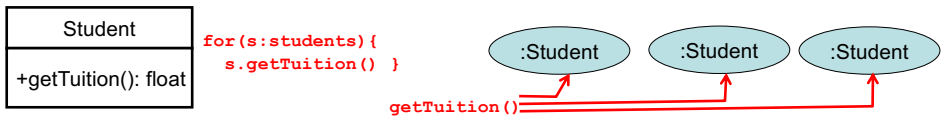
Visitor Design Pattern

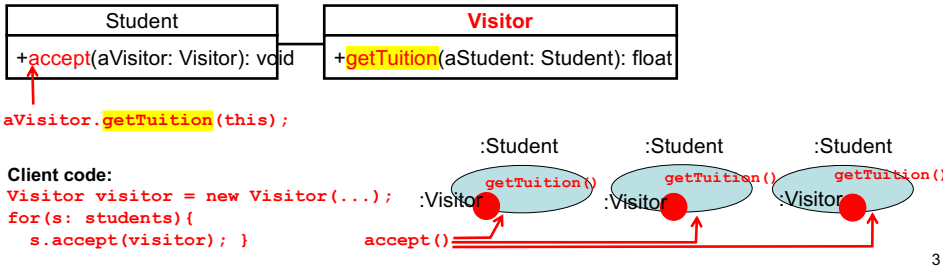
Visitor Design Pattern

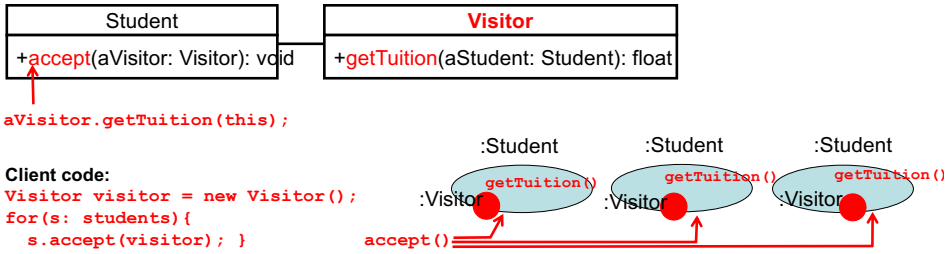
- Intent
  - Separate (or decouple) a set of objects and the operations to be performed on those objects.

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

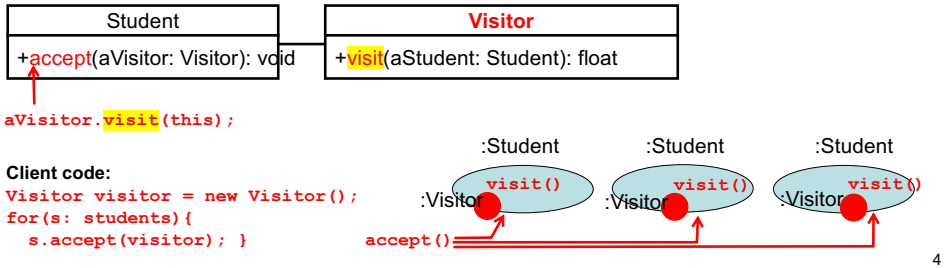


- With *Visitor*, the operation is defined as a method of a Visitor class.



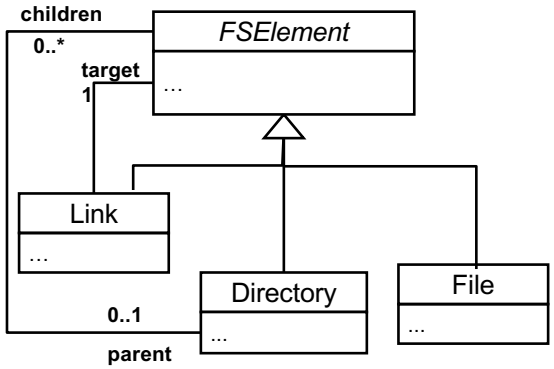


- A method(s) in a Visitor class are often named `visit()`.

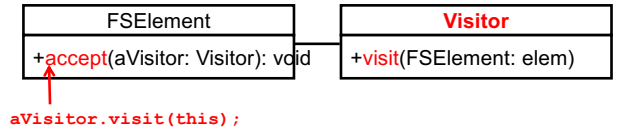


File System Examples (1)

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

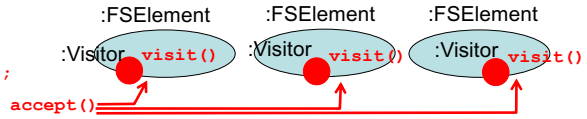


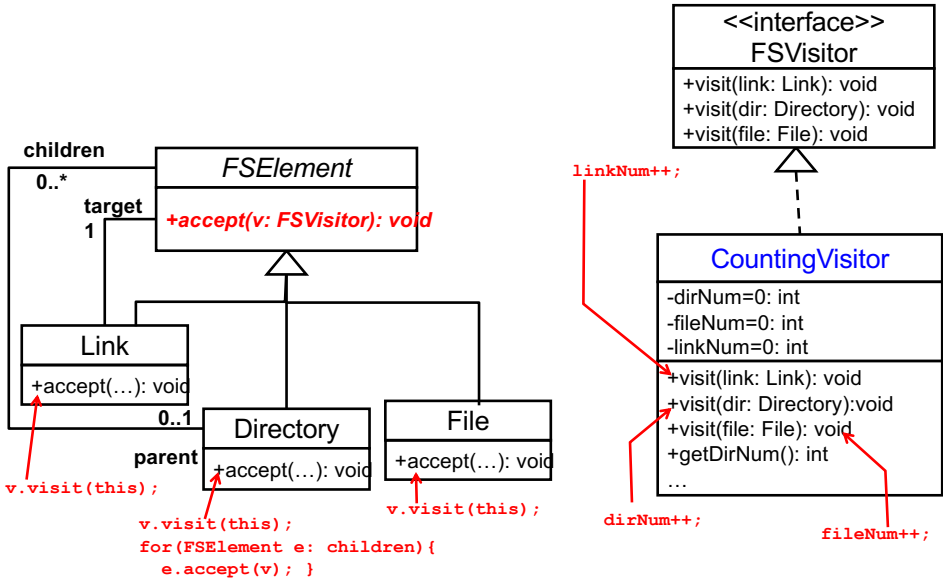
- With *Visitor*, an operation to count FS elements can be implemented as a method of the Visitor class.



Client code:

```
Visitor visitor = new Visitor();
for(e: FSElement){
  e.accept(visitor); }
```

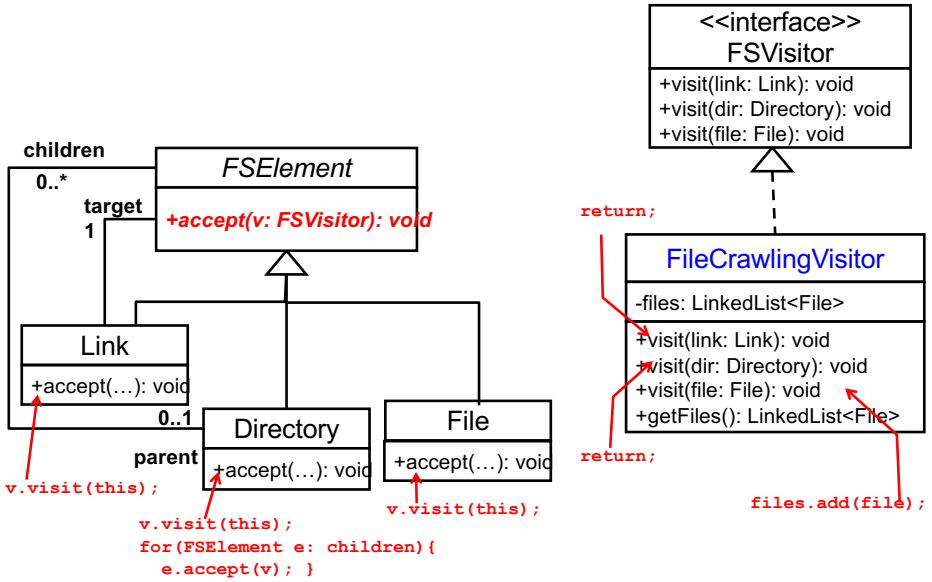





```
CountingVisitor visitor = new CountingVisitor();
rootDir.accept( visitor );
visitor.getDirNum(); visitor.getFileNum(); visitor.getLinkNum();
```


File System Examples (2)

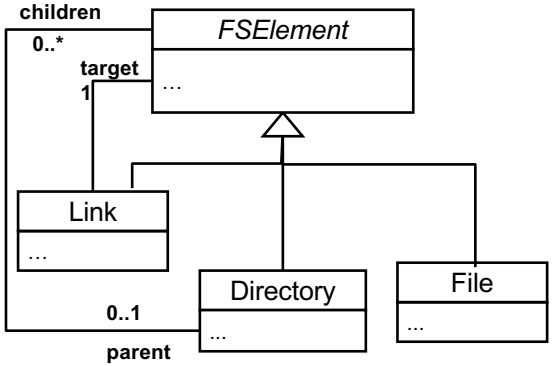
- Index files in a file system
  - c.f. OS indexing service
    - e.g., Windows indexing service and Mac/iOS Spotlight
  - Key functionalities
    - Crawl a file system to identify files
    - Extract and keep each file's metadata for later searches.
      - e.g., Path, name, size, creation time, owner's name, last-modified timestamp, checksum
- With *Visitor*, the file-crawling operation can be implemented as a method of the Visitor class.

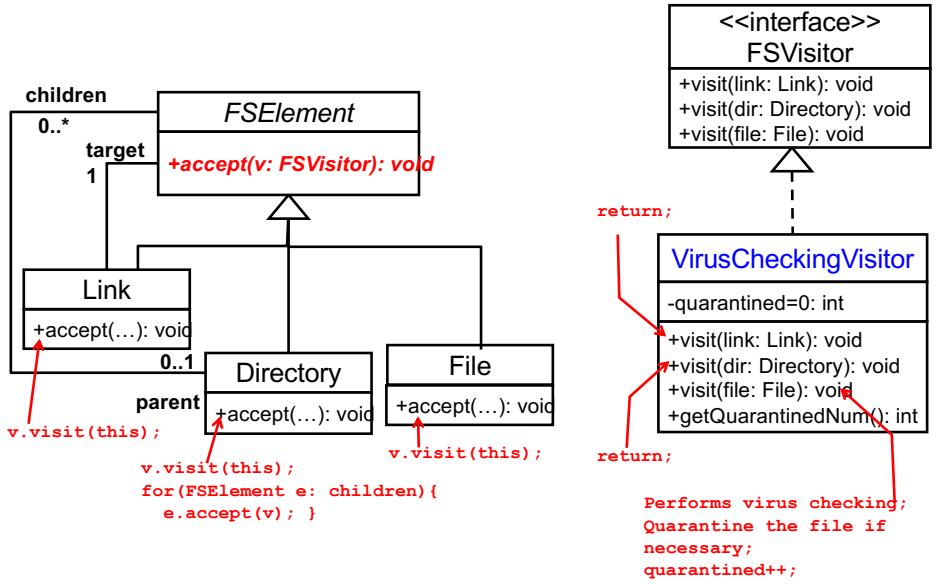



```
FileCrawlingVisitor visitor = new FileCrawlingVisitor();
rootDir.accept( visitor );
visitor.getFiles();
```


File System Examples (3)

- Perform virus check for each file in a file system
  - With *Visitor*, the virus-checking operation can be defined in a visitor.

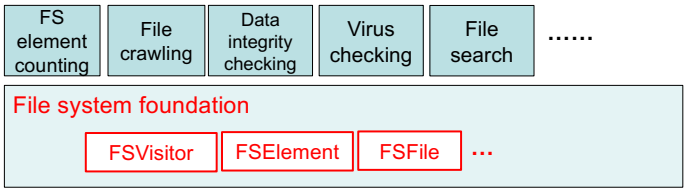





```
VirusCheckingVisitor visitor = new VirusCheckingVisitor();
rootDir.accept( visitor );
visitor.getQuarantinedNum();
```

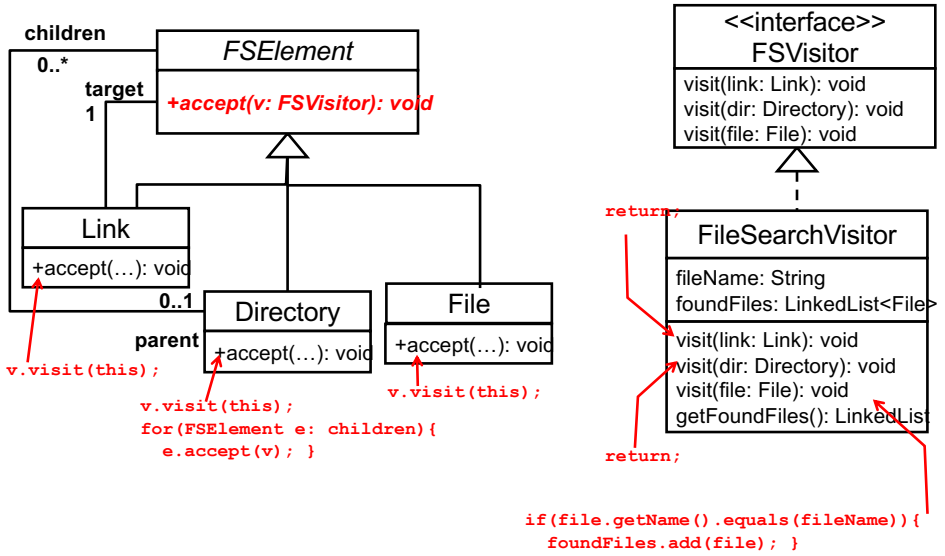

What's the Point?

- *Visitor* can separate FS data structures and the operations to be performed on those data structures.
  - Allows those operations to be pluggable.
  - Makes it easy to add, modify and remove those operations without changing FS data structures.



HW 8

- Define `FSVisitor`, `FSElement`, `Directory`, `File`, `Link` and `FileSystem` in the `umbcs680.hw08.fs` package.
- Implement `FSVisitor` with 3 visitor classes in an extra package: `umbcs680.hw08.util`
  - `CountingVisitor`
  - `FileCrawlingVisitor`
  - `FileSearchVisitor`
    - Find a file with its name
- Use the 3 visitors with an example FS structure that you have used in HW 6 and 7.




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


HW 9

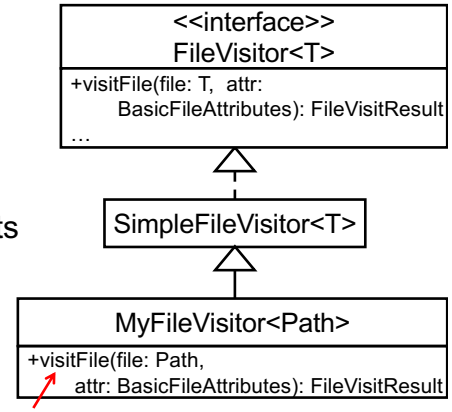
- Make YOUR OWN HW assignment and provide a solution to it.
- Come up with an example of the *Composite* design pattern in a particular application.
  - Do NOT use an example covered in CS680.
- Implement it yourself.
- Turn in:
  - Your implementation (code)
  - A short readme.txt file that explains what kind of app you consider and how your code implements the *Composite* design pattern.
  - Test code and an Ant script.

Applicability of *Visitor*

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

Visitor in Java API

- **`FileVisitor<T>`** and **`SimpleFileVisitor<T>`** in Java NIO (New I/O) (**`java.nio`**)
  - A visitor for files.
    - In **`java.nio.file`**
  - **`visitFile(file, attr)`**
    - Invoked when a visitor visits a **`file`**.
    - **`attr`**: a set of attributes (metadata) of the **`file`**
    - **`Path`**: Represents a path. See Appendix.




```
System.out.println(file);
System.out.println(attr.size());
System.out.println(attr.creationTime());
...
```


- **`java.nio.file.Files`**
  - A utility class (i.e., a set of static methods) to process a file/directory.
- **`Files.walkFileTree()`**
  - Visits each file in a file tree and calls **`visitFile()`** on a visitor.
    - `static Path walkFileTree(Path start, FileVisitor<Path> visitor)`
-
```
Path aDir = ...;
Files.walkFileTree( aDir, new MyFileVisitor<Path>() );
```