Reading List.

Follow all the links listed explicitly on the slides and at least give them a read.

Some of the pages listed below, and some of the content linked from the slides have additional topics that were never explicitly brought up in class (you can tell, because no content related to them is present in the slides).
NOTE: Those "never covered" topics will *not* be present on the exam. If you are wondering whether something is relevant, apply this test first: *"is there any mention of it on the slides?"*. If you are still unsure, ask!!

Software Development Process

http://en.wikipedia.org/wiki/Software_development_process (and Waterfall/Spiral/XP/Scrum subpages)

Requirements/User Stories

http://en.wikipedia.org/wiki/User_story
http://en.wikipedia.org/wiki/Requirements_analysis

Software Design

http://en.wikipedia.org/wiki/Single_responsibility_principle
http://en.wikipedia.org/wiki/Software_design
Is Design Dead: http://martinfowler.com/articles/designDead.html
Practical UML: http://edn.embarcadero.com/article/31863
(and google all the modularity principles we discussed in lecture - wikipedia pages exist for each, as do Stack Overflow explanations)

Design — Review

same old concepts, but we're doing them again

NOTE:
the order in which we do things here is *not prescriptive* —
it is necessarily meandering and iterative.

Let's consider the problem from the sample midterm

You are creating a user interface that will display the contents of the filesystem. The filesystem contains folders and files, and each folder may contain other folders or files. When a change is made to the file system (e.g., new file or folder created, file or folder renamed, etc.) your user interface needs to update to reflect the change. Your user interface will be a window with a TreeViewer control. The TreeViewer control will display all of the files and folders.

How would this look as a user story?

You are creating a user interface that will display the contents of the filesystem. The filesystem contains folders and files, and each folder may contain other folders or files. When a change is made to the file system (e.g., new file or folder created, file or folder renamed, etc.) your user interface needs to update to reflect the change. Your user interface will be a window with a TreeViewer control. The TreeViewer control will display all of the files and folders.

value statement: As someone who wants to keep track of their hierarchy of folders and files, I want to be able to see a tree view that updates whenever a change is made to the system

acceptance criteria:

- the folders must be able to contain both folders and files
- the view must be organised in tree format reflecting the folder hierarchy
- the view must update when a change is made to the file system (new file or folder created, or file or folder renamed, etc)

I'm not necessarily going to use this user story again (I might, and probably will if I'm using Scrum), but re-stating the problem helps build my perspective.

I've noticed some detailed design stuff … let's capture that

You are creating a user interface that will display the contents of the filesystem. The filesystem contains folders and files, and each folder may contain other folders or files. When a change is made to the file system (e.g., new file or folder created, file or folder renamed, etc.) your user interface needs to update to reflect the change.

Your user interface will be a window with a TreeViewer control.
The TreeViewer control will display all of the files and folders.

things we learn for free — we need to:

- make a User Interface that "Is a" Window (so there's such thing as a window)
- make a TreeViewer control
- put the TreeViewer control into the User Interface
- The Tree Viewer Control will display all the files and folders

if I'm using Scrum, these might become tasks later?
Probably? Or will help me make some tasks?

value statement:
As someone who uses folders hierarchically, I want to be able to see a tree view that updates whenever a change is made to the system so I can keep track of my hierarchy of folders and files

acceptance criteria:

- the folders must be able to contain both folders and files
- the view must be organised in tree format reflecting the folder hierarchy
- the view must update when a change is made to the file system (new file or folder created, or file or folder renamed, etc)

Those things also give me some hints about responsibilities of these things I've found

You are creating a user interface that will display the contents of the filesystem. The filesystem contains folders and files, and each folder may contain other folders or files. When a change is made to the file system (e.g., new file or folder created, file or folder renamed, etc.) your user interface needs to update to reflect the change.
Your user interface will be a window with a TreeViewer control.
The TreeViewer control will display all of the files and folders.

Responsibilities we identify:
- The TreeViewer control displays files and folders
- The UserInterface window houses (and, probably passes information to?) the TreeViewer

things we learn for free — we need to:
- make a User Interface that "is a" window
- make a TreeViewer control
- put the TreeViewer control into the User Interface
- The Tree Viewer Control will display all the files and folders

value statement:
As someone who uses folders hierarchically, I want to be able to see a tree view that updates whenever a change is made to the system so I can keep track of my hierarchy of folders and files

acceptance criteria:
- the folders must be able to contain both folders and files
- the view must be organised in tree format reflecting the folder hierarchy
- the view must update when a change is made to the file system (new file or folder created, or file or folder renamed, etc)

Well I've eked out all I can from those last lines
Now let's revisit the next sentence and see if we can find entities and

You are creating a user interface that will display the contents of the filesystem. The filesystem contains folders and files, and each folder may contain other folders or files. When a change is made to the file system (e.g., new file or folder created, file or folder renamed, etc.) your user interface needs to update to reflect the change.
Your user interface will be a window with a TreeViewer control.
The TreeViewer control will display all of the files and folders.

Responsibilities we identified before:
- The TreeViewer control displays files and folders
- The UserInterface window houses (and, probably passes information to?) the TreeViewer

things we learn for free — we need to:
- make a User Interface that "is a" window
- make a TreeViewer control
- put the TreeViewer control into the User Interface
- The Tree Viewer Control will display all the files and folders

New stuff we see in Sentence 1:
- There's a UI (we knew that already)
- There's a filesystem (that's new!)
- and the UI *displays* the contents of the filesystem (corroborates our theory from before about the UI more or less using the TreeViewer)

value statement:
As someone who uses folders hierarchically, I want to be able to see a tree view that updates whenever a change is made to the system so I can keep track of my hierarchy of folders and files

acceptance criteria:
- the folders must be able to contain both folders and files
- the view must be organised in tree format reflecting the folder hierarchy
- the view must update when a change is made to the file system (new file or folder created, or file or folder renamed, etc)

This is getting messy. Let's rewrite what we know

1. You are creating a user interface that will display the contents of the filesystem.
2. The filesystem contains folders and files, and each folder may contain other folders or files.
3. When a change is made to the file system (e.g., new file or folder created, file or folder renamed, etc.) your user interface needs to update to reflect the change.
4. Your user interface will be a window with a TreeViewer control.
5. The TreeViewer control will display all of the files and folders.

Thing	What we know about it
User Interface	displays contents of the filesystem (probably using the TreeView control) is-a Window, has-a TreeView Control (that does the display)
Window	that acts like a window? need to think about this…
Tree Viewer Control	displays contents of the filesystem
Filesystem	there is one!
Folders	can have files or other folders, can be renamed
Files	have … stuff in them. who cares?

great … let's keep working through the sentences since we're not done with those …

things we learn for free — we need to:
- make a User Interface that "is a" window
- make a TreeViewer control
- put the TreeViewer control into the User Interface
- The Tree Viewer Control will display all the files and folders

Initially found responsibilities:
- The TreeViewer control displays files and folders
- The UserInterface window houses (and, probably passes information to?) the TreeViewer

Sentence 1
- There's a UI (we knew that already)
- There's a filesystem (that's new!)
- and the UI *displays* the contents of the filesystem (corroborates our theory from before about the UI more or less using the TreeViewer)

value statement:
As someone who uses folders hierarchically, I want to be able to see a tree view that updates whenever a change is made to the system so I can keep track of my hierarchy of folders and files

acceptance criteria:
- the folders must be able to contain both folders and files
- the view must be organised in tree format reflecting the folder hierarchy
- the view must update when a change is made to the file system (new file or folder created, or file or folder renamed, etc)

Now let's add to that…

1. *You are creating a user interface that will display the contents of the filesystem.*
2. ***The filesystem contains folders and files, and each folder may contain other folders or files.***
3. *When a change is made to the file system (e.g., new file or folder created, file or folder renamed, etc.) your user interface needs to update to reflect the change.*
4. *Your user interface will be a window with a TreeViewer control.*
5. *The TreeViewer control will display all of the files and folders.*

we already knew that folders could have other folders, but let's keep track of the fact that the filesystem is a hierarchical structure

Thing	What we know about it
User Interface	displays contents of the filesystem (probably using the TreeView control) is-a Window, has-a TreeView Control (that does the display)
Window	that acts like a window? need to think about this…
Tree Viewer Control	displays contents of the filesystem
Filesystem	**has a tree-like structure of folders and files**
Folders	can have files or other folders, can be renamed
Files	have … stuff in them. who cares?

things we learn for free — we need to:
- make a User Interface that "is a" window
- make a TreeViewer control
- put the TreeViewer control into the User Interface
- The Tree Viewer Control will display all the files and folders

Initially found responsibilities:
- The TreeViewer control displays files and folders
- The UserInterface window houses (and, probably passes information to?) the TreeViewer

Sentence 1
- There's a UI (we knew that already)
- There's a filesystem (that's new!)
- and the UI *displays* the contents of the filesystem (corroborates our theory from before about the UI more or less using the TreeViewer)

value statement:
As someone who uses folders hierarchically, I want to be able to see a tree view that updates whenever a change is made to the system so I can keep track of my hierarchy of folders and files

acceptance criteria:
- the folders must be able to contain both folders and files
- the view must be organised in tree format reflecting the folder hierarchy
- the view must update when a change is made to the file system (new file or folder created, or file or folder renamed, etc)

Still adding…

1. *You are creating a user interface that will display the contents of the filesystem.*
2. *The filesystem contains folders and files, and each folder may contain other folders or files.*
3. ***When a change is made to the file system (e.g., new file or folder created, file or folder renamed, etc.) your user interface needs to update to reflect the change.***
4. *Your user interface will be a window with a TreeViewer control.*
5. *The TreeViewer control will display all of the files and folders.*

who should do this? The TreeView Control only talks about displaying, but maybe it could also cover capturing user input. Then again maybe we should split that up we could make an InputListener. Let's do that for now, we can merge again later if we want to.

Thing	What we know about it
User Interface	displays contents of the filesystem (probably using the TreeView control) is-a Window, has-a TreeView Control (that does the display)
Window	that acts like a window? need to think about this…
Tree Viewer Control	displays contents of the filesystem
Input Listener	**marshals changes to the filesystem. Tells the filesystem. Tells the UI to update**
Filesystem	has a tree-like structure of folders and files
Folders	can have files or other folders, can be renamed
Files	have … stuff in them. who cares?

things we learn for free — we need to:
- make a User Interface that "is a" window
- make a TreeViewer control
- put the TreeViewer control into the User Interface
- The Tree Viewer Control will display all the files and folders

Initially found responsibilities:
- The TreeViewer control displays files and folders
- The UserInterface window houses (and, probably passes information to?) the TreeViewer

Sentence 1
- There's a UI (we knew that already)
- There's a filesystem (that's new!)
- and the UI *displays* the contents of the filesystem (corroborates our theory from before about the UI more or less using the TreeViewer)

value statement:
As someone who uses folders hierarchically, I want to be able to see a tree view that updates whenever a change is made to the system so I can keep track of my hierarchy of folders and files

acceptance criteria:
- the folders must be able to contain both folders and files
- the view must be organised in tree format reflecting the folder hierarchy
- the view must update when a change is made to the file system (new file or folder created, or file or folder renamed, etc)

Oh - done with that description

1. You are creating a user interface that will display the contents of the filesystem.
2. The filesystem contains folders and files, and each folder may contain other folders or files.
3. When a change is made to the file system (e.g., new file or folder created, file or folder renamed, etc.) your user interface needs to update to reflect the change.
~~4. Your user interface will be a window with a TreeViewer control.~~
~~5. The TreeViewer control will display all of the files and folders.~~

we've already done these!

Thing	What we know about it
User Interface	displays contents of the filesystem (probably using the TreeView control) is-a Window, has-a TreeView Control (that does the display)
Window	that acts like a window? need to think about this…
Tree Viewer Control	displays contents of the filesystem
Input Listener	marshals changes to the filesystem. Tells the filesystem. Tells the UI to update
Filesystem	has a tree-like structure of folders and files
Folders	can have files or other folders, can be renamed
Files	have … stuff in them. who cares?

things we learn for free — we need to:
- make a User Interface that "is a" window
- make a TreeViewer control
- put the TreeViewer control into the User Interface
- The Tree Viewer Control will display all the files and folders

Initially found responsibilities:
- The TreeViewer control displays files and folders
- The UserInterface window houses (and, probably passes information to?) the TreeViewer

Sentence 1
- There's a UI (we knew that already)
- There's a filesystem (that's new!)
- and the UI *displays* the contents of the filesystem (corroborates our theory from before about the UI more or less using the TreeViewer)

value statement:
As someone who uses folders hierarchically, I want to be able to see a tree view that updates whenever a change is made to the system so I can keep track of my hierarchy of folders and files

acceptance criteria:
- the folders must be able to contain both folders and files
- the view must be organised in tree format reflecting the folder hierarchy
- the view must update when a change is made to the file system (new file or folder created, or file or folder renamed, etc)

What about that user story?

1. You are creating a user interface that will display the contents of the filesystem.
2. The filesystem contains folders and files, and each folder may contain other folders or files.
3. When a change is made to the file system (e.g., new file or folder created, file or folder renamed, etc.) your user interface needs to update to reflect the change.
4. Your user interface will be a window with a TreeViewer control.
5. The TreeViewer control will display all of the files and folders.

Thing	What we know about it
User Interface	displays contents of the filesystem (probably using the TreeView control) is-a Window, has-a TreeView Control (that does the display)
Window	that acts like a window? need to think about this…
Tree Viewer	displays contents of the filesystem **as a tree**
Input Listener	marshals changes to the filesystem. Tells the filesystem. Tells the UI to update
Filesystem	has a tree-like structure of folders and files
Folders	can have files or other folders, can be renamed
Files	have … stuff in them. who cares?

value statement:
As someone who uses folders hierarchically, I want to be able to see a **tree view** that updates whenever a change is made to the system so I can keep track of my hierarchy of folders and files

we had added some specification in translating to the user story, but it works in well, so let's keep it.

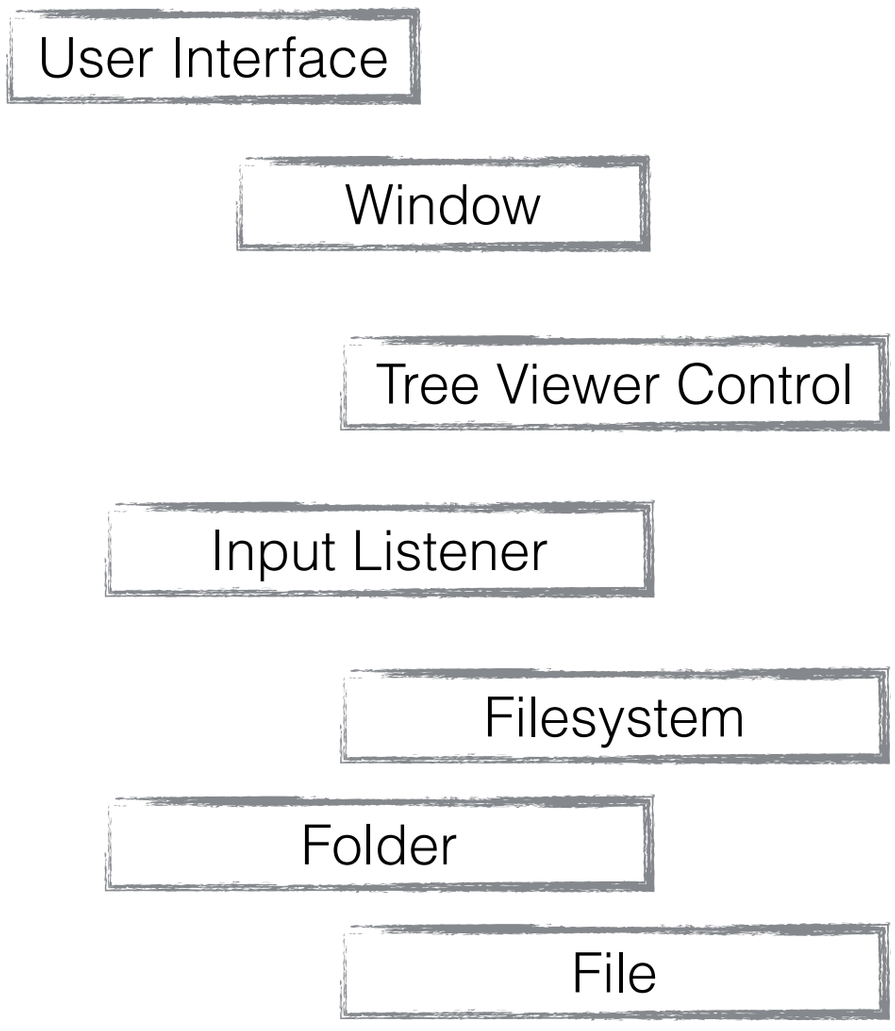
acceptance criteria:

- the folders must be able to contain both folders and files
- **the view must be organised in tree format reflecting the folder hierarchy**
- the view must update when a change is made to the file system (new file or folder created, or file or folder renamed, etc)

mainly we've got everything.

Okaaaaay, let's make a sketch?

Thing	What we know about it
User Interface	displays contents of the filesystem (probably using the TreeView control) is-a Window, has-a TreeView Control (that does the display)
Window	that acts like a window? need to think about this…
Tree Viewer Control	displays contents of the filesystem **as a tree**
Input Listener	marshals changes to the filesystem. Tells the filesystem. Tells the UI to update
Filesystem	has a tree-like structure of folders and files
Folders	can have files or other folders, can be renamed
Files	have … stuff in them. who cares?



Things are almost never plural, so I changed Folders to Folder and Files to File

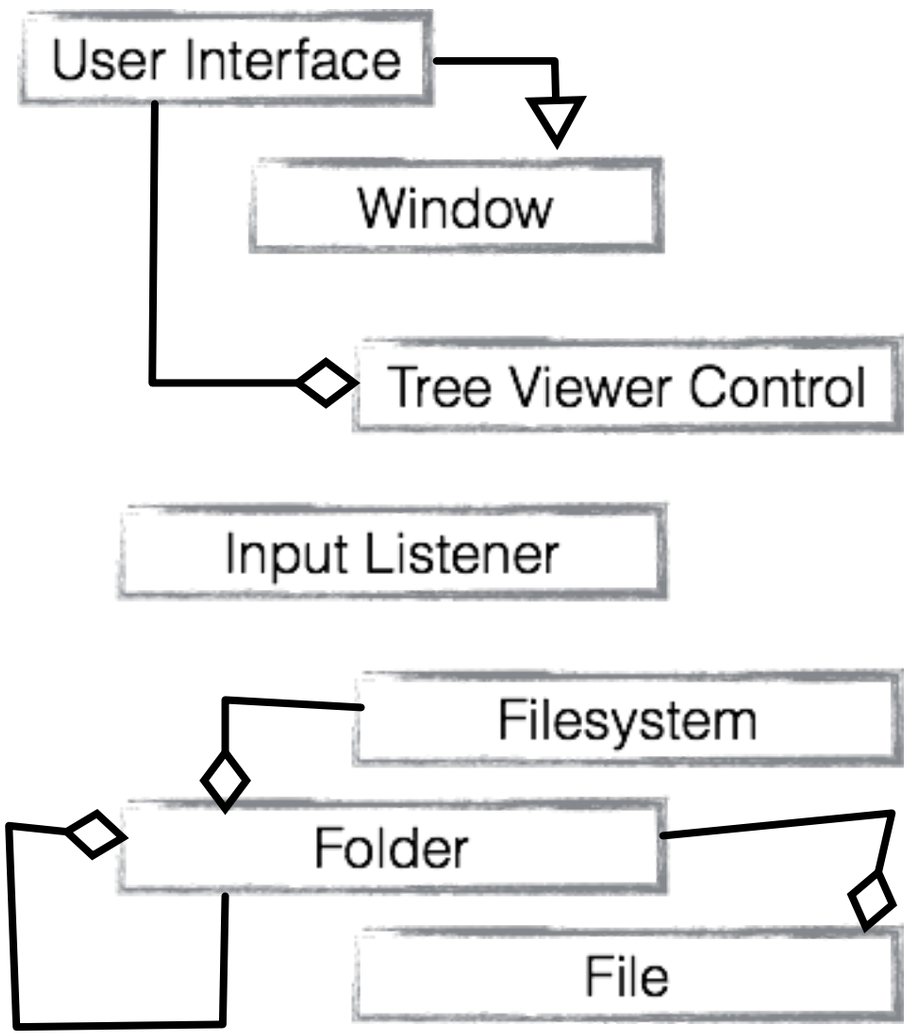
Let's look for easy to find aggregations, associations and generalisations

Thing	What we know about it
User Interface	displays contents of the filesystem (probably using the TreeView control) is-a Window, has-a TreeView Control (that does the display)
Window	that acts like a window? need to think about this…
Tree Viewer Control	displays contents of the filesystem **as a tree**
Input Listener	marshals changes (still not sure how) to the filesystem. Tells the filesystem. Tells the UI to update
Filesystem	has a tree-like structure of folders and files
Folders	can have files or other folders, can be renamed
Files	have … stuff in them. who cares?

is-a

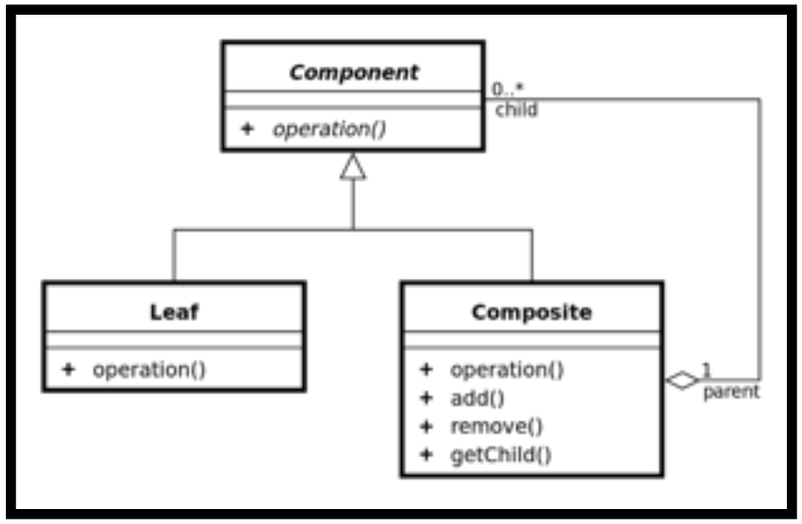
has-a

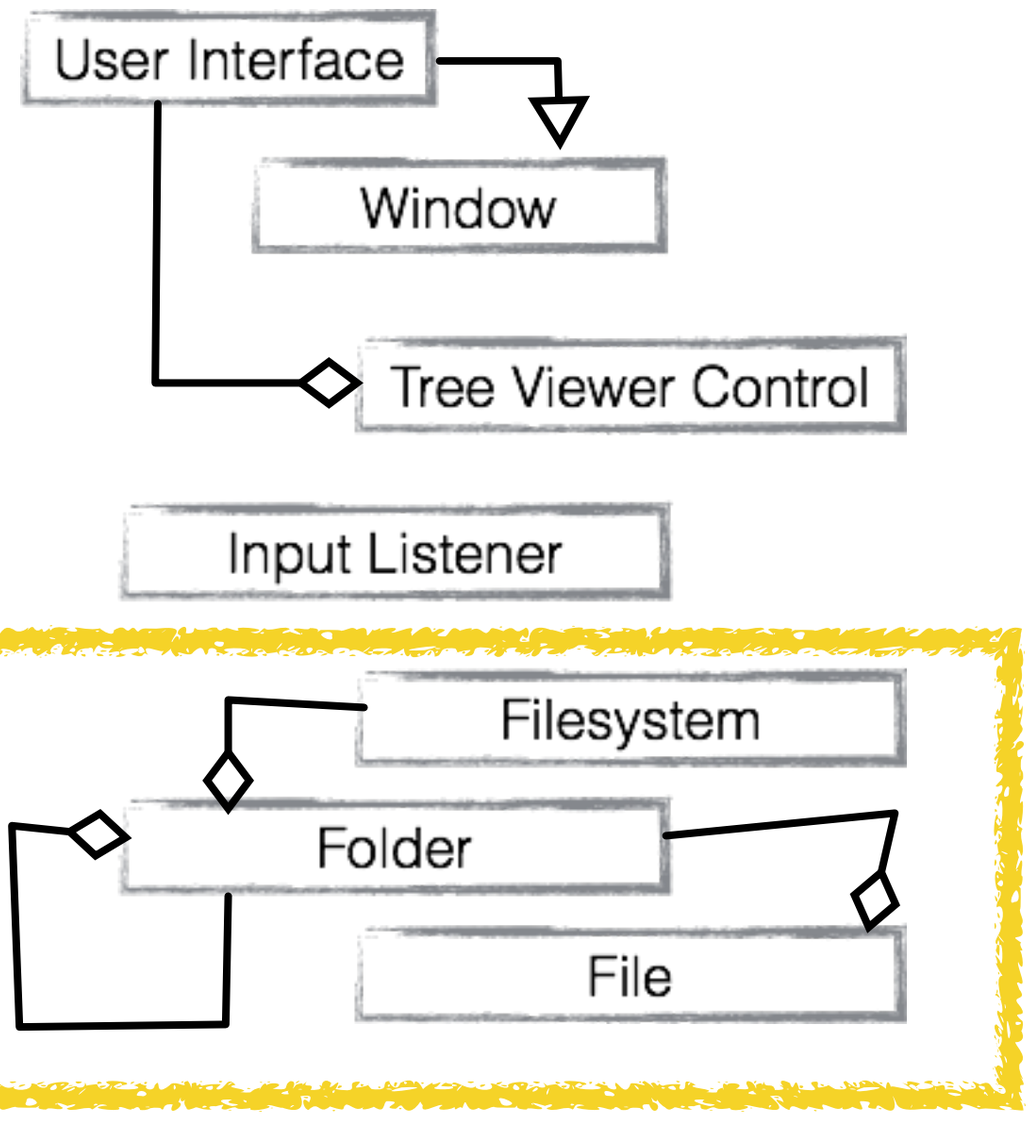
has-a



Stare at that for a bit … ooh

Thing	What we know about it
User Interface	displays contents of the filesystem (probably using the TreeView control) is-a Window, has-a TreeView Control (that does the display)
Window	that acts like a window? need to think about this…
Tree Viewer Control	displays contents of the filesystem **as a tree**
Input Listener	marshals changes (still not sure how) to the filesystem. Tells the filesystem. Tells the UI to update
Filesystem	has a tree-like structure of folders and files
Folders	can have files or other folders, can be renamed
Files	have … stuff in them. who cares?



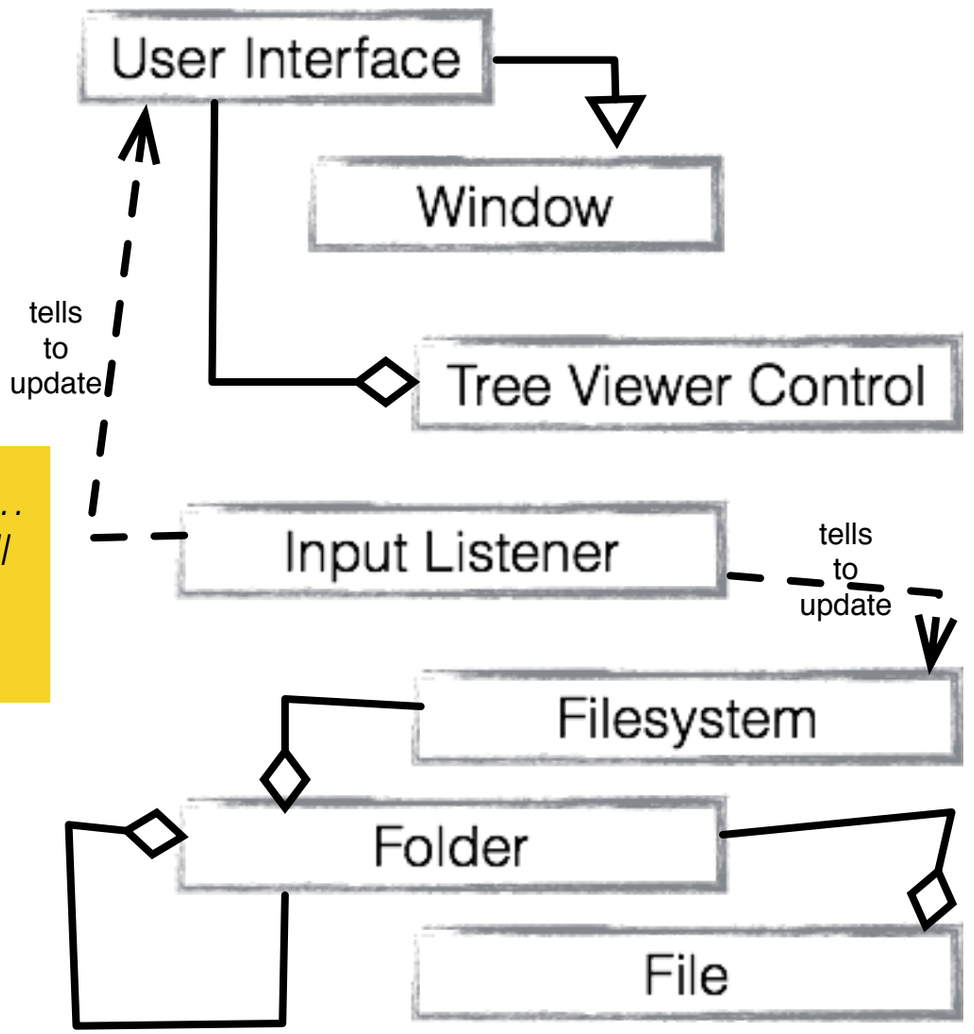


This reminds me of composite from 210. It's a bit different. Should I change it? Later?

Okay … make more connections

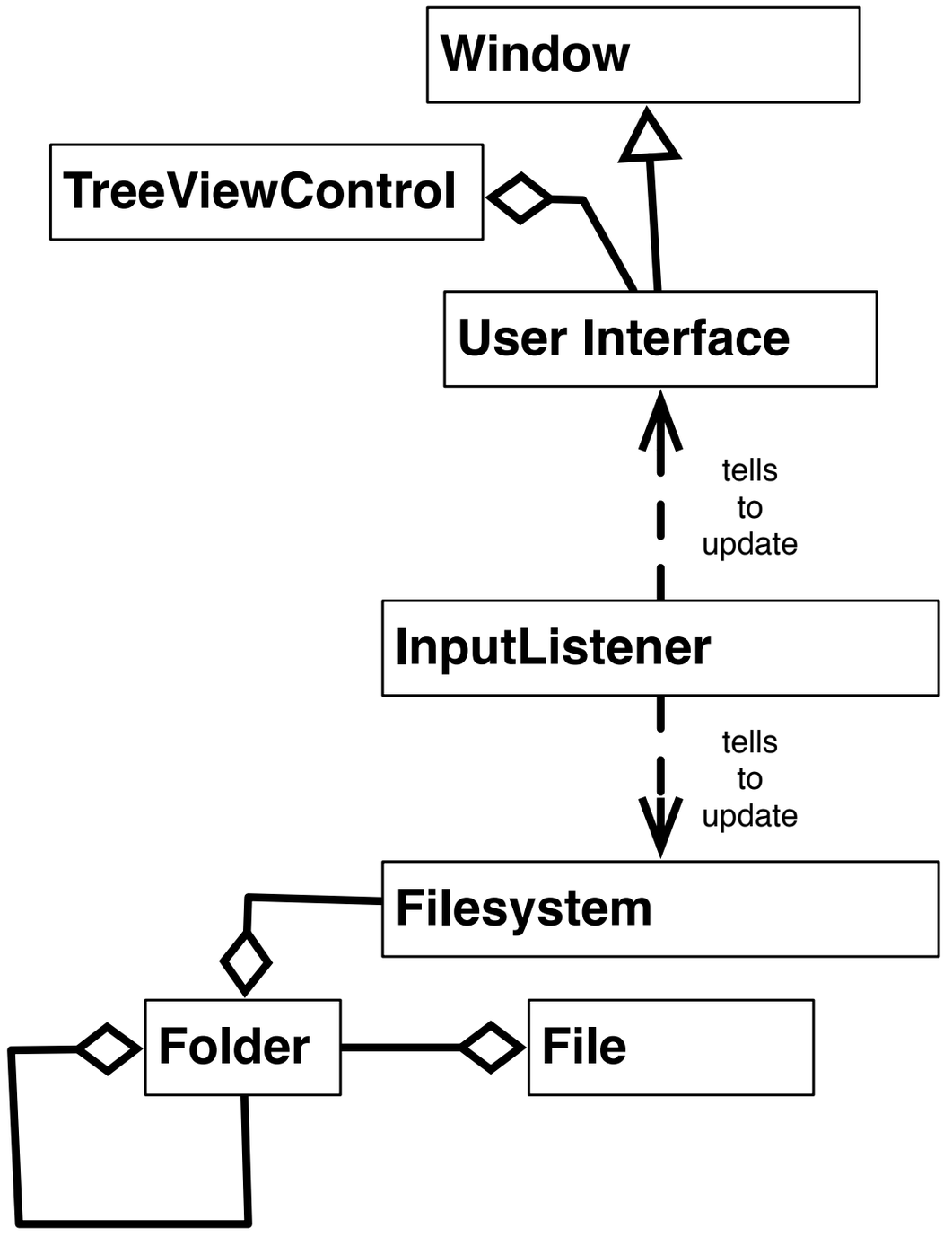
Thing	What we know about it
User Interface	displays contents of the filesystem (probably using the TreeView control) is-a Window, has-a TreeView Control (that does the display)
Window	that acts like a window? need to think about this…
Tree Viewer Control	displays contents of the filesystem as a tree
Input Listener	marshals changes (still not sure how) to the filesystem. **Tells the** filesystem. **Tells the** UI to update
Filesystem	has a tree-like structure of folders and files
Folders	can have files or other folders, can be renamed
Files	have … stuff in them. who cares?

pretty abstract … but may as well stick it on the diagram



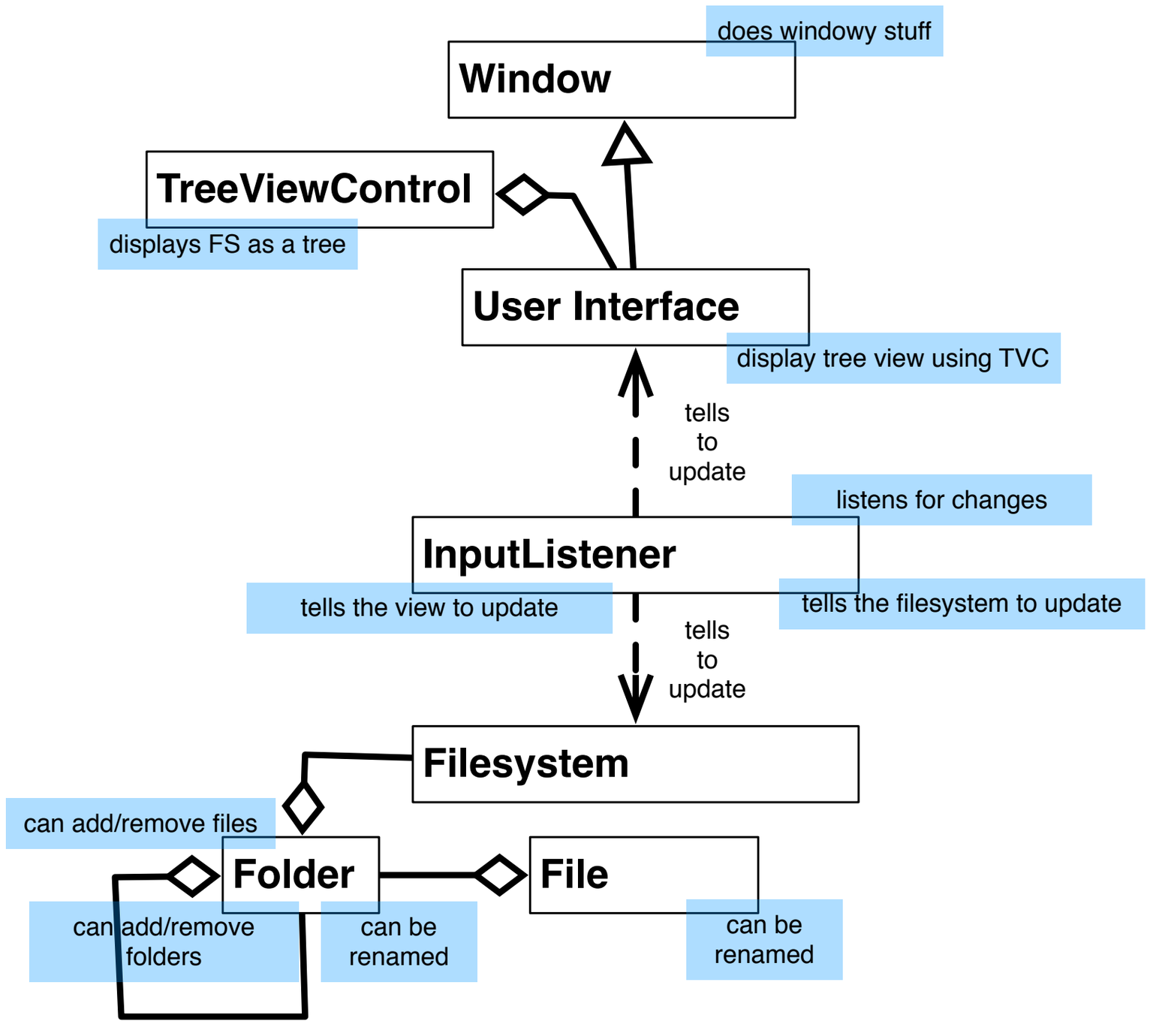
Re-draw for layout…

Thing	What we know about it
User Interface	displays contents of the filesystem (probably using the TreeView control) is-a Window, has-a TreeView Control (that does the display)
Window	that acts like a window? need to think about this…
Tree Viewer Control	displays contents of the filesystem as a tree
Input Listener	marshals changes (still not sure how) to the filesystem. Tells the filesystem. Tells the UI to update
Filesystem	has a tree-like structure of folders and files
Folders	can have files or other folders, can be renamed
Files	have … stuff in them. who cares?



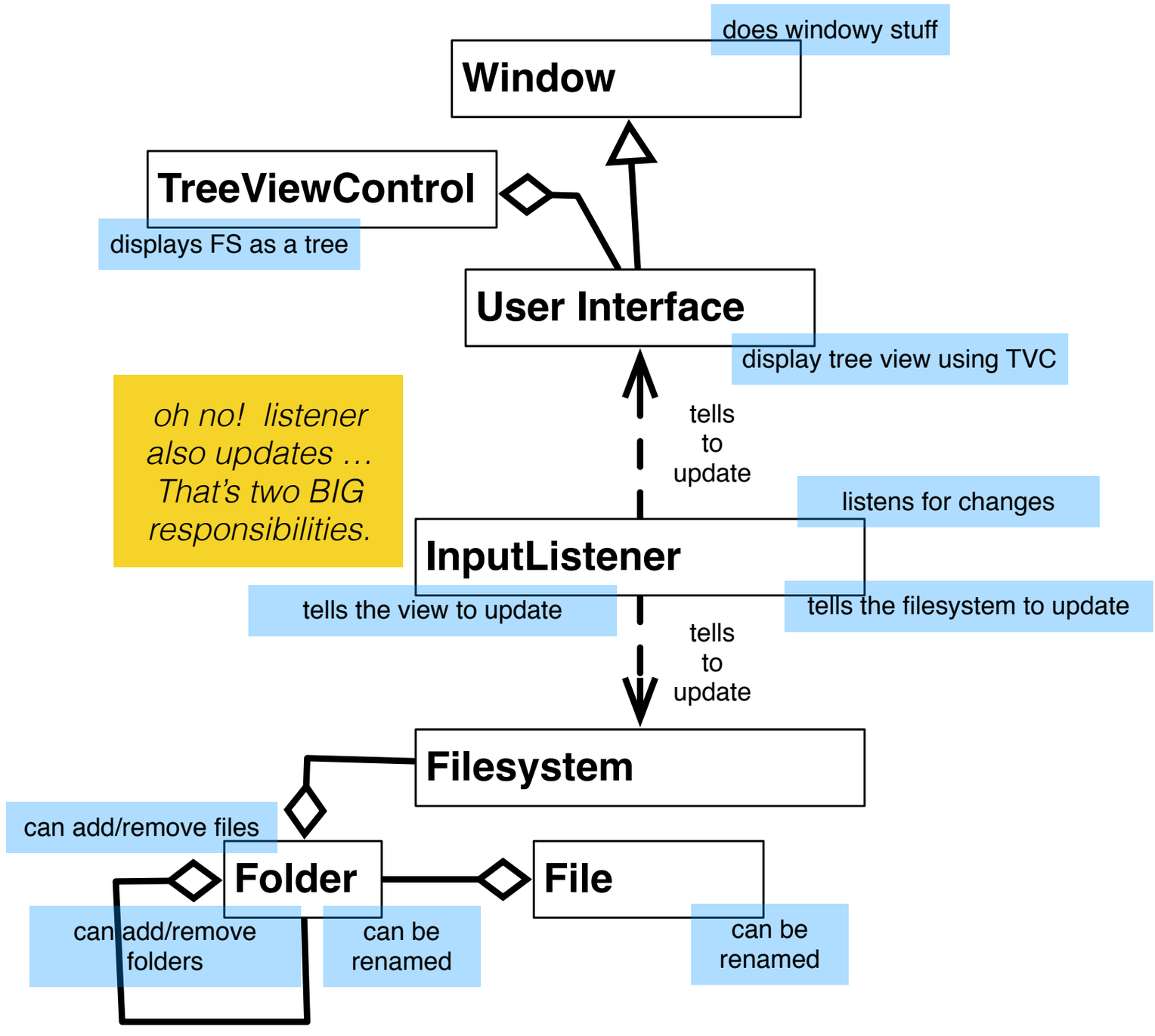
Add some verbs for responsibilities

Thing	What we know about it
User Interface	displays contents of the filesystem (probably using the TreeView control) is-a Window, has-a TreeView Control (that does the display)
Window	that acts like a window? need to think about this…
Tree Viewer Control	displays contents of the filesystem as a tree
Input Listener	marshals changes (still not sure how) to the filesystem. Tells the filesystem. Tells the UI to update
Filesystem	has a tree-like structure of folders and files
Folders	can have files or other folders, can be renamed
Files	have … stuff in them. who cares?



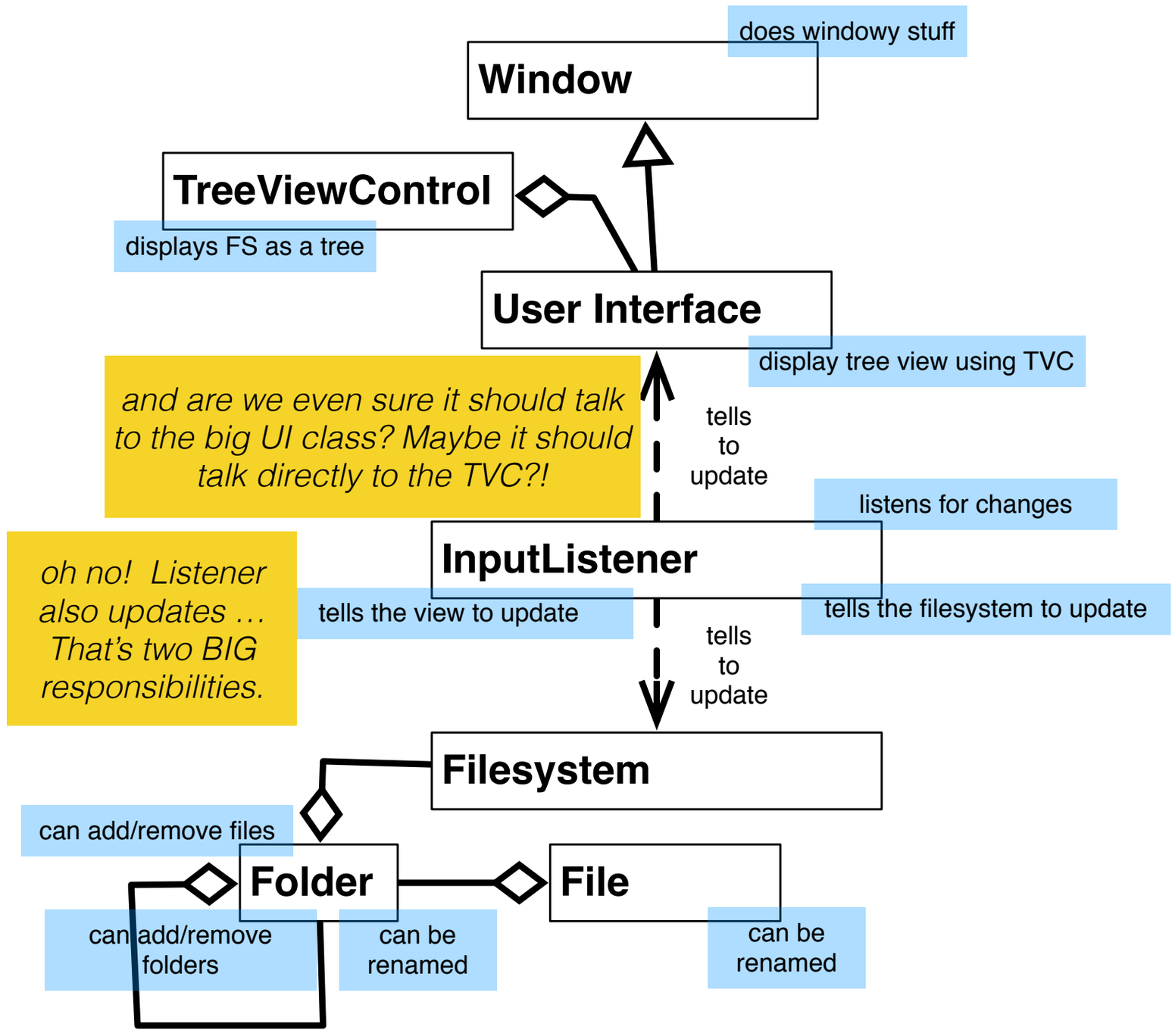
Stare at that for a bit….

Thing	What we know about it
User Interface	displays contents of the filesystem (probably using the TreeView control) is-a Window, has-a TreeView Control (that does the display)
Window	that acts like a window? need to think about this…
Tree Viewer Control	displays contents of the filesystem as a tree
Input Listener	marshals changes (still not sure how) to the filesystem. Tells the filesystem. Tells the UI to update
Filesystem	has a tree-like structure of folders and files
Folders	can have files or other folders, can be renamed
Files	have … stuff in them. who cares?

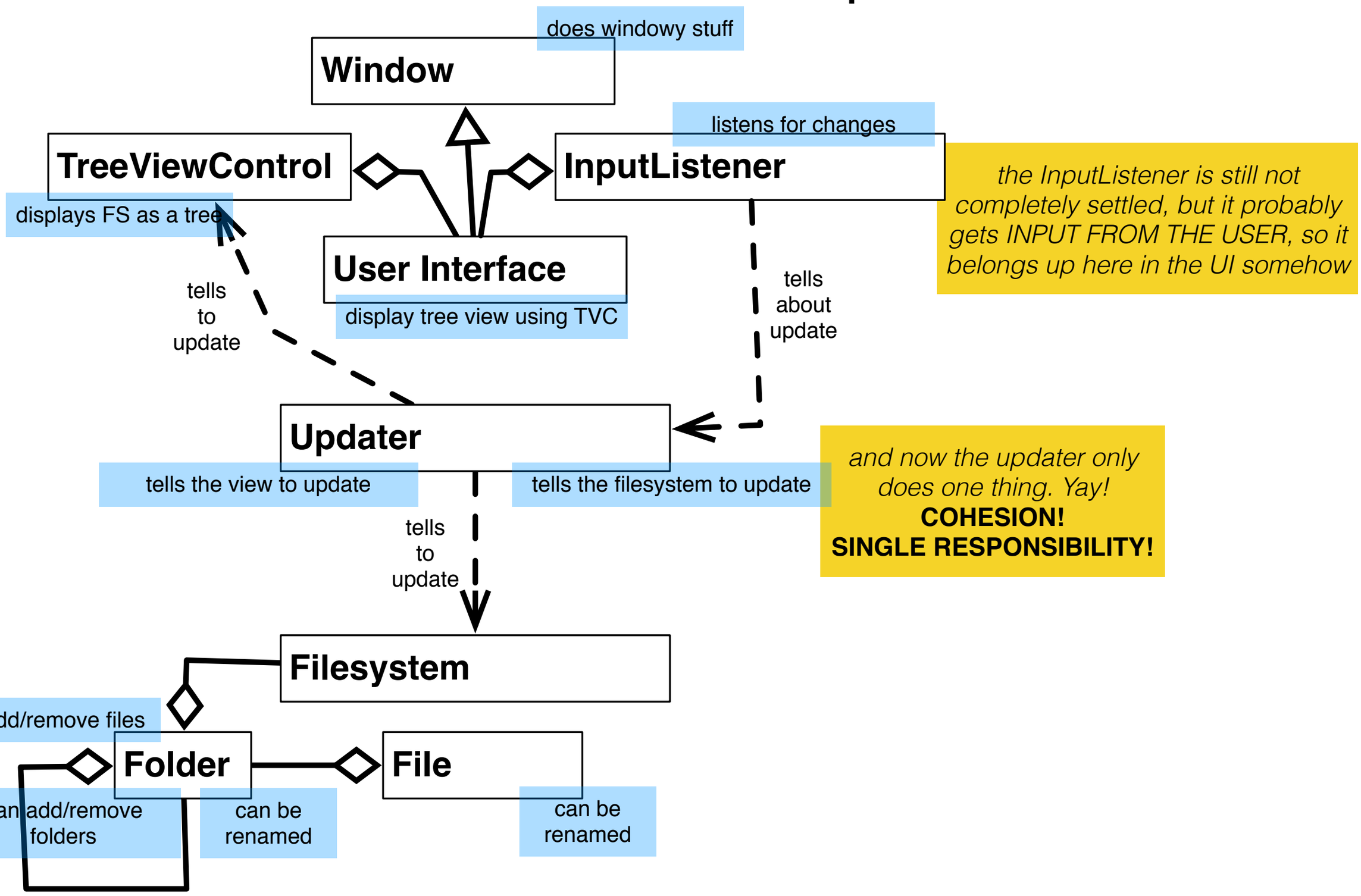


Stare at that for a bit….

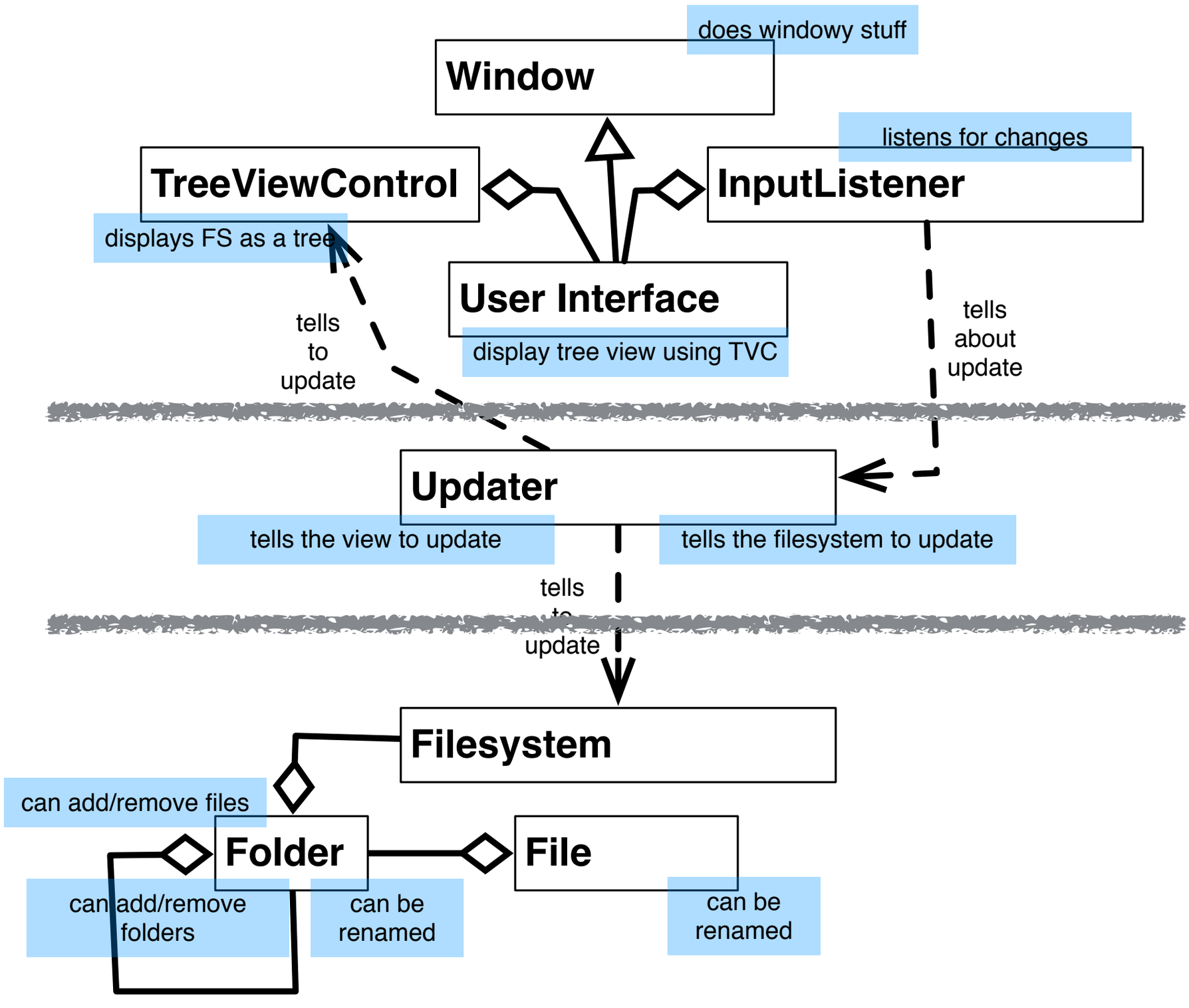
Thing	What we know about it
User Interface	displays contents of the filesystem (probably using the TreeView control) is-a Window, has-a TreeView Control (that does the display)
Window	that acts like a window? need to think about this…
Tree Viewer Control	displays contents of the filesystem as a tree
Input Listener	marshals changes (still not sure how) to the filesystem. Tells the filesystem. Tells the UI to update
Filesystem	has a tree-like structure of folders and files
Folders	can have files or other folders, can be renamed
Files	have … stuff in them. who cares?



Let's split Listener into Listener and Updater



Huh. That's nice…somewhat layered.



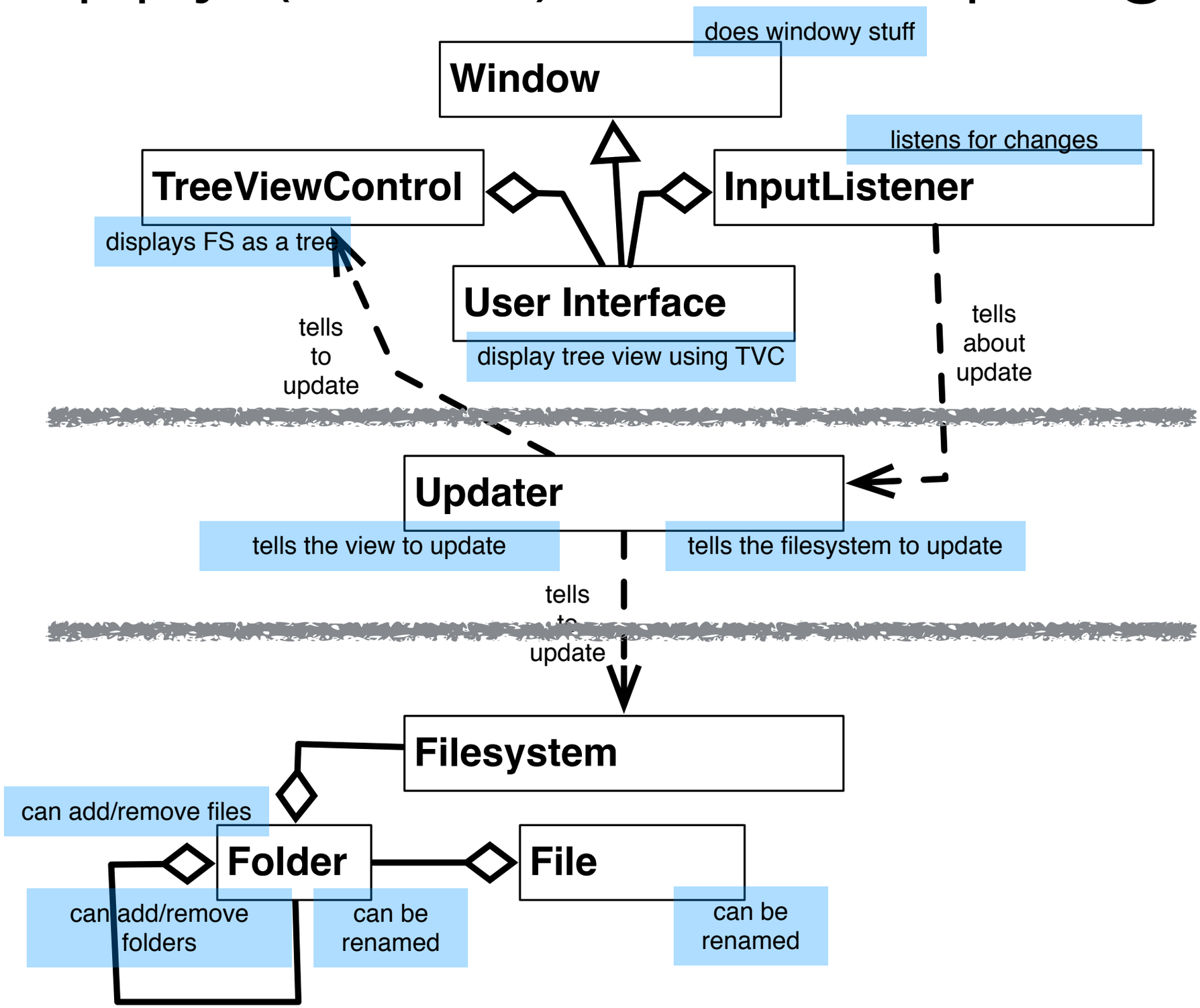
***MOST** user triggered systems end up looking something like this.*

UI on top

some logic in the middle

data on the bottom

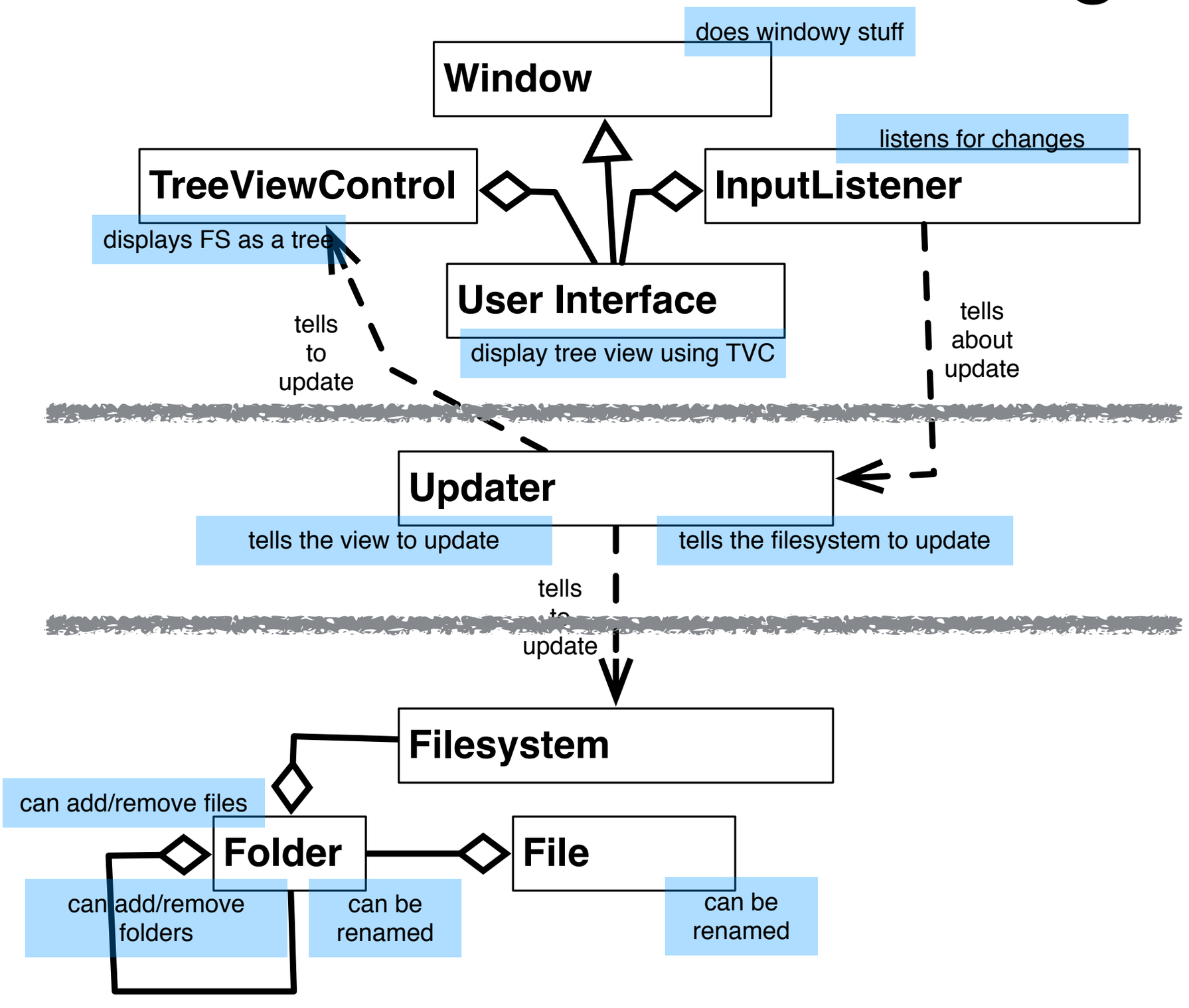
The law of Demeter looks somewhat happy (so far), and coupling looks low



no arrows from UI straight to the File System

each class is behaviourally connected to just one thing!
And containment connections are also minimal

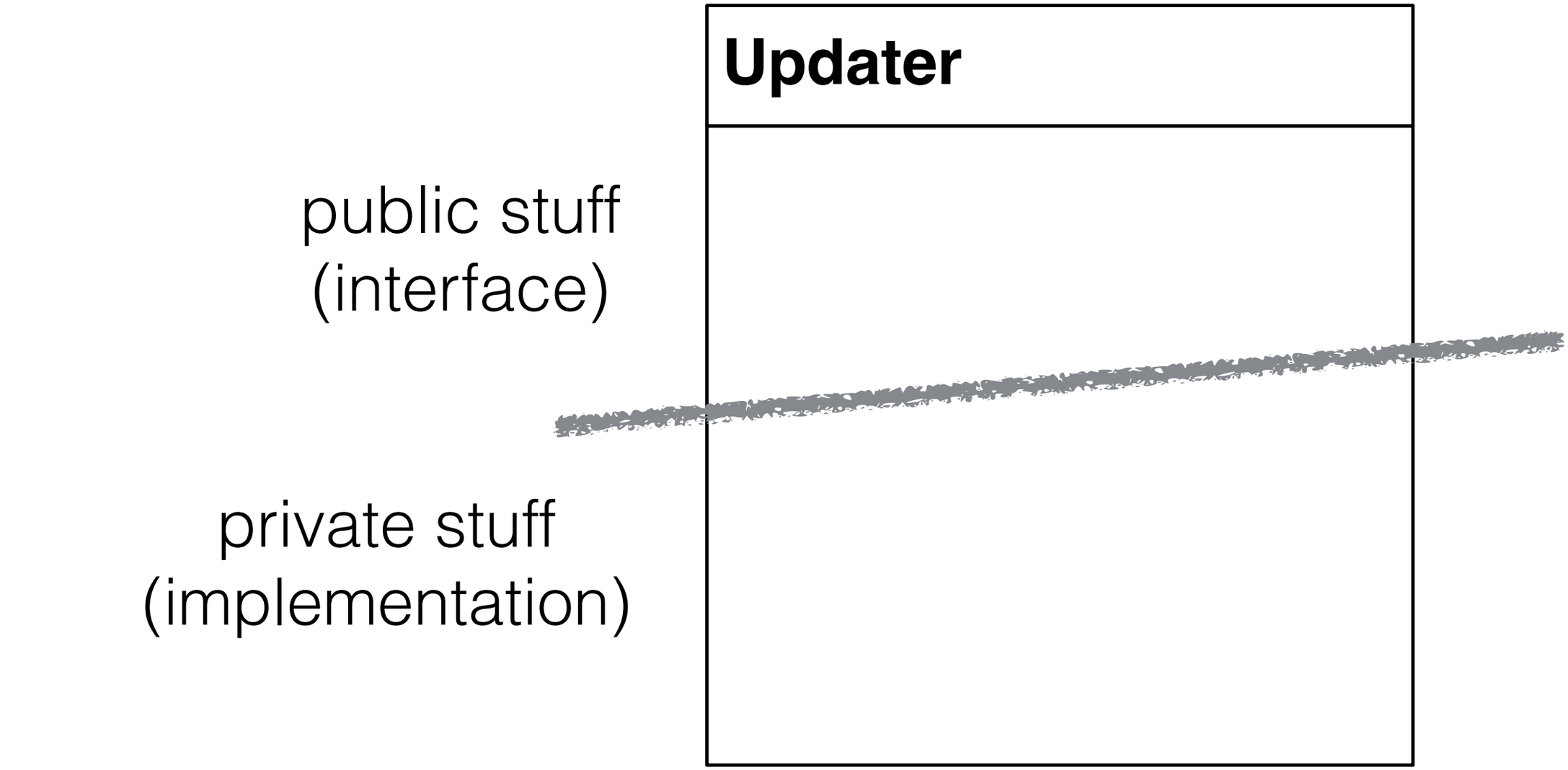
Okay …. so …. what about … Information hiding?



but but but what about information hiding? How do we get that?

We need to drill deeper to take that into consideration. Let's take a closer look at Updater.

But what about Information Hiding?



But what about Information Hiding?

If everything is up here, we have a very bad iceberg!!

public stuff
(interface)

private stuff
(implementation)

Updater

handleNewUpdate( )
updateView( )
updateFilesystem( )

and no one should be calling these directly anyway!

Let's move some stuff around!

But what about Information Hiding?

this is a lot less tippy

+ public stuff (interface)

- private stuff (implementation)

+ means public
- means private

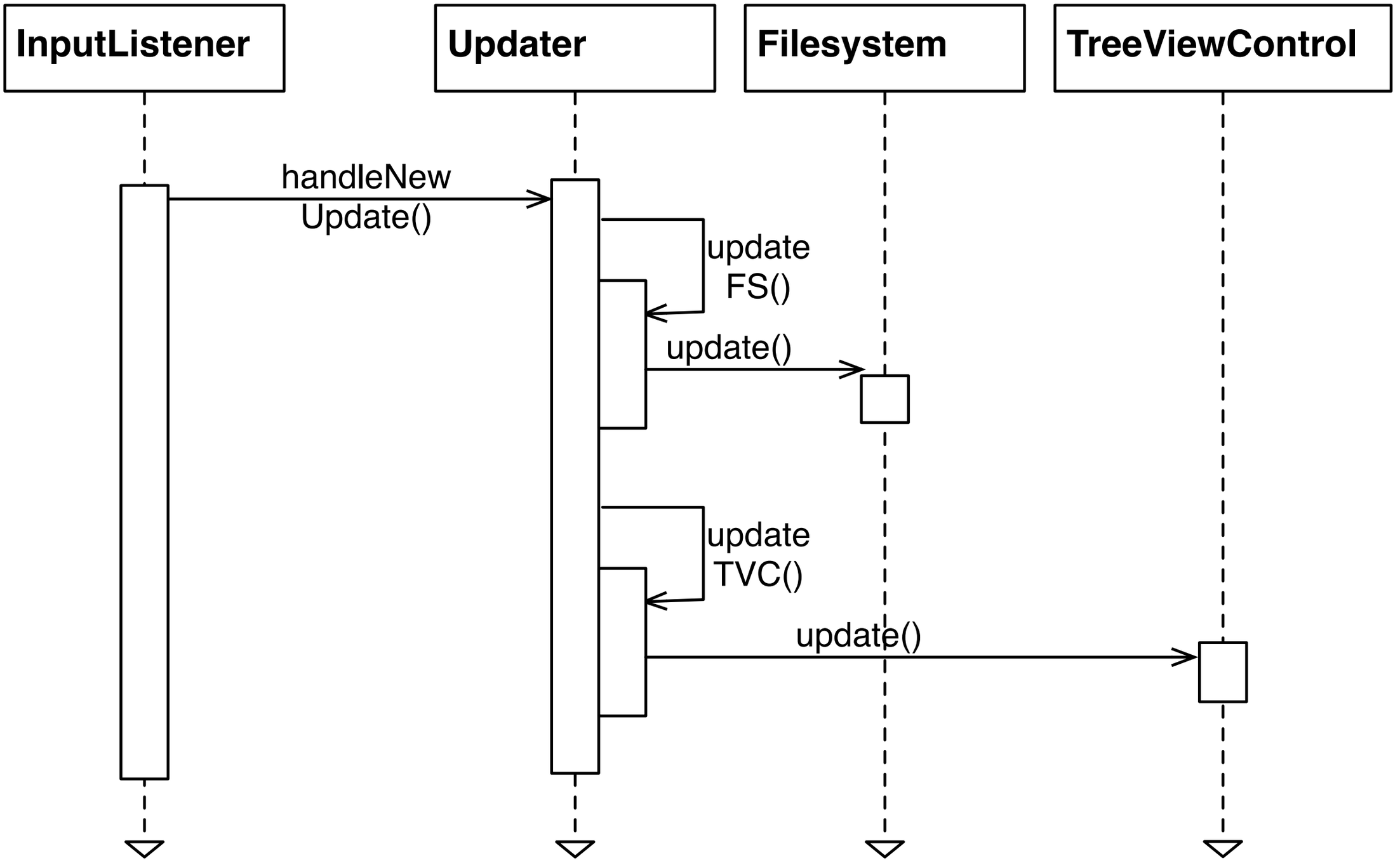
Updater

+ handleNewUpdate( )

self.updateView( )
self.updateFileSystem

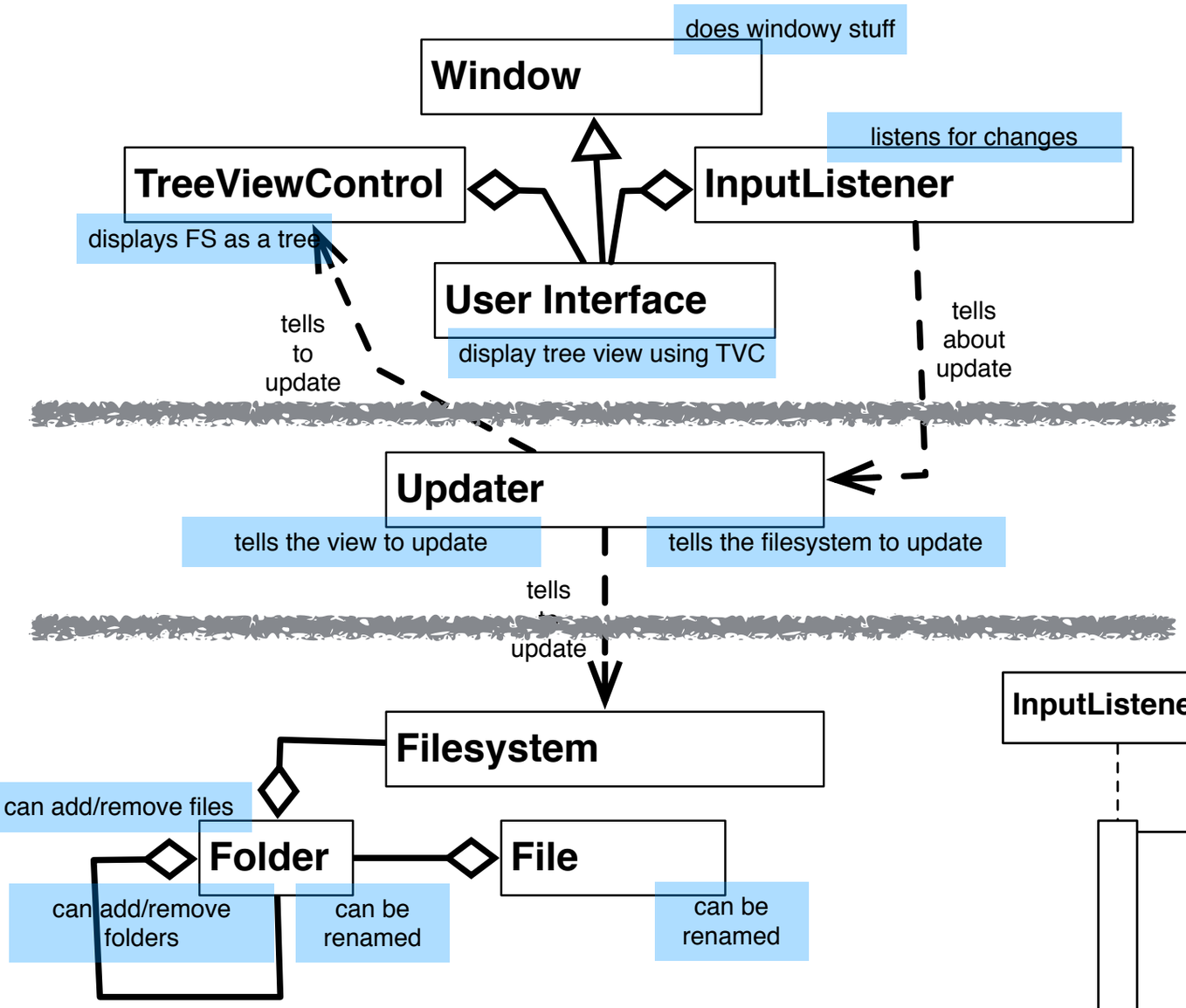
- updateView( )
- updateFilesystem( )

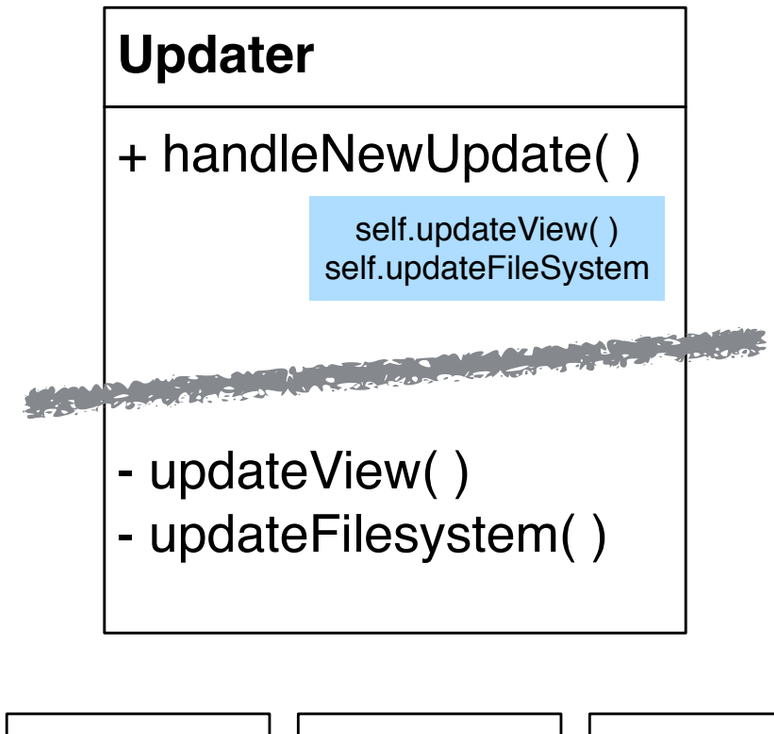
And we can see the information hiding in a sequence diagram

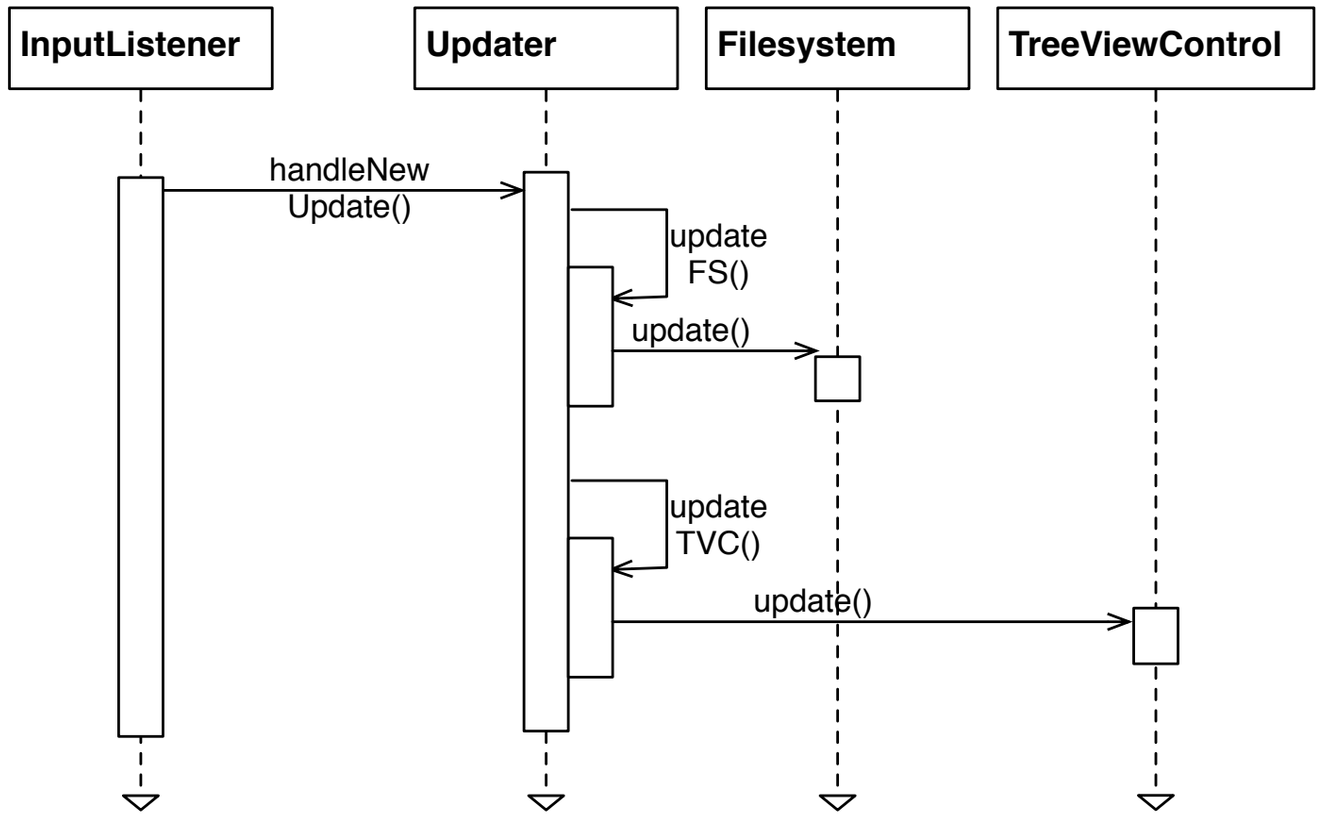


Nice! Next we would keep going, and make public/private splits for each class

Questions!


does windowy stuff
Window
listens for changes
TreeViewControl
InputListener
displays FS as a tree
User Interface
display tree view using TVC
tells to update
tells about update
Updater
tells the view to update
tells the filesystem to update
tells update
Filesystem
can add/remove files
Folder
File
can add/remove folders
can be renamed
can be renamed


Updater
+ handleNewUpdate( )
self.updateView( )
self.updateFileSystem
- updateView( )
- updateFilesystem( )


InputListener
Updater
Filesystem
TreeViewControl
handleNew Update()
update FS()
update()
update TVC()
update()