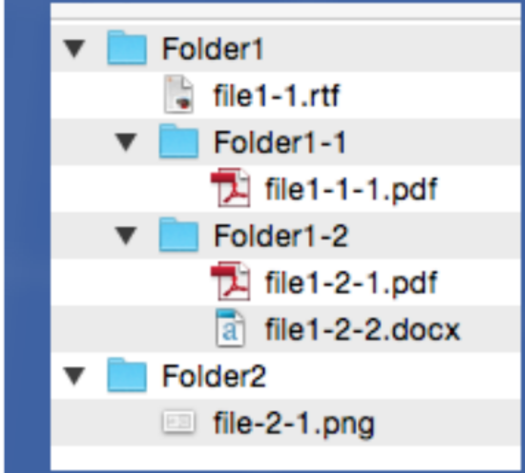
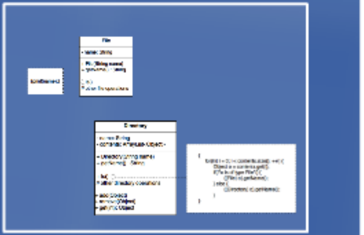
Think "Trees"...



- Folder1
  - file1-1.rtf
  - Folder1-1
    - file1-1-1.pdf
  - Folder1-2
    - file1-2-1.pdf
    - file1-2-2.docx
- Folder2
  - file-2-1.png



File system has *files* and *directories*

Directories can *contain* files & sub-directories

Several *operations* on files & folders have common *interface*

Composite

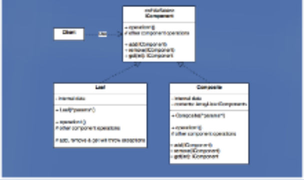
Compose objects into tree structures to represent *whole-part* hierarchies

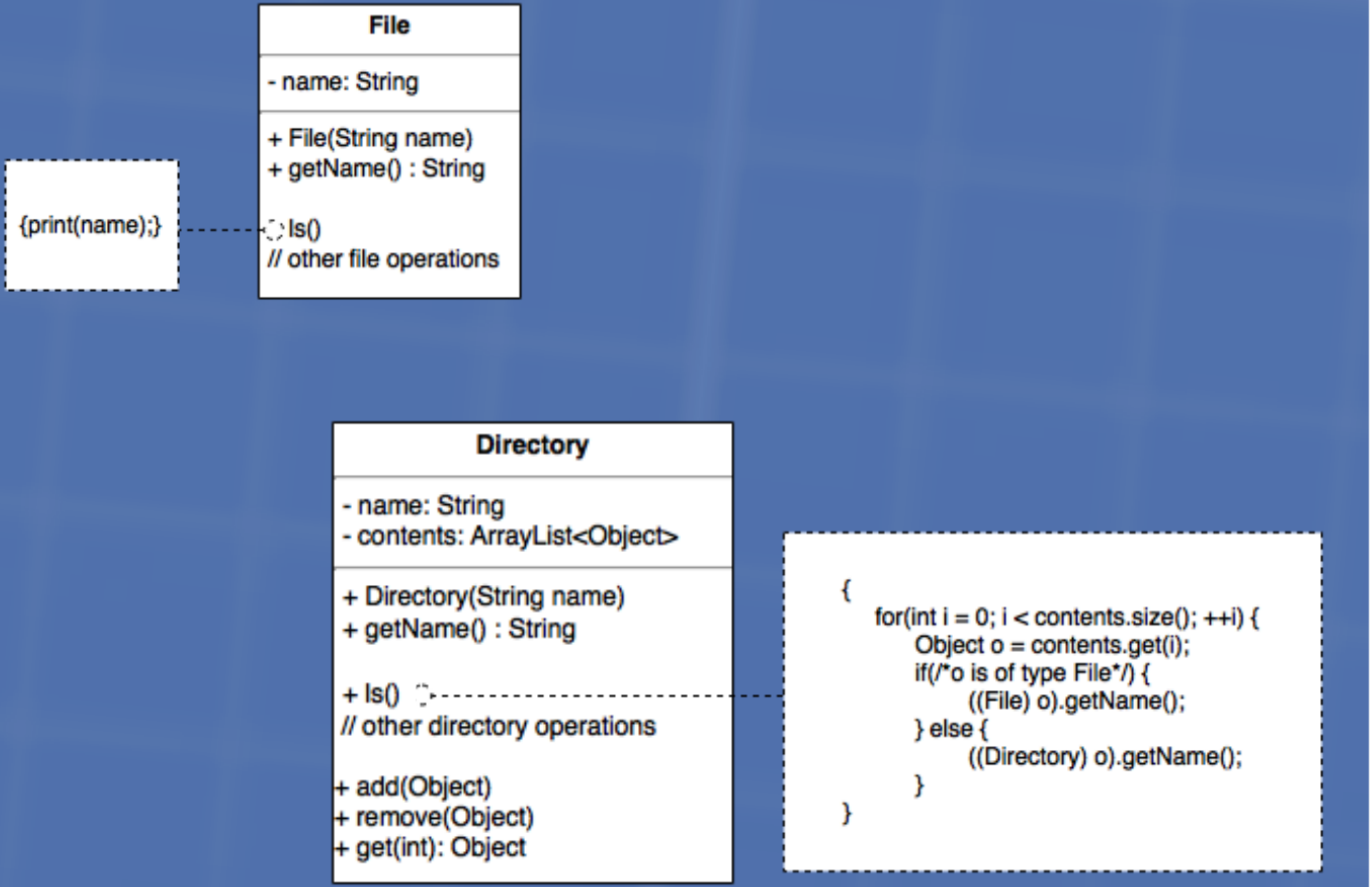
Recursive composition

Client can treat individual objects and compositions of objects *uniformly*

Trade off some violations for *transparency*

Design Skeleton





File

- name: String

+ File(String name)
+ getName() : String

ls()
// other file operations

```
{print(name);}
```

Directory

- name: String
- contents: ArrayList<Object>

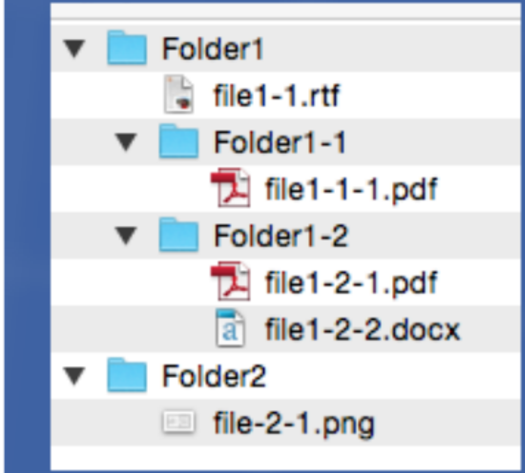
+ Directory(String name)
+ getName() : String

+ ls()
// other directory operations

+ add(Object)
+ remove(Object)
+ get(int): Object

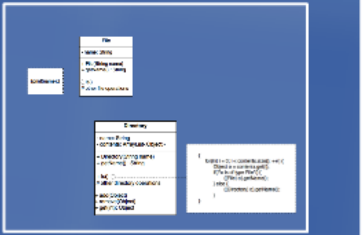
```
{
    for(int i = 0; i < contents.size(); ++i) {
        Object o = contents.get(i);
        if(/*o is of type File*/) {
            ((File) o).getName();
        } else {
            ((Directory) o).getName();
        }
}
```

Think "Trees"...



- Folder1
  - file1-1.rtf
  - Folder1-1
    - file1-1-1.pdf
  - Folder1-2
    - file1-2-1.pdf
    - file1-2-2.docx
- Folder2
  - file-2-1.png

File system has *files* and *directories*



Directories can *contain* files & sub-directories

Several *operations* on files & folders have common *interface*

Composite

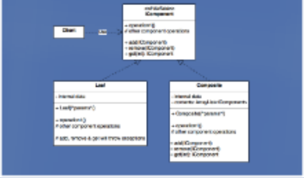
Compose objects into tree structures to represent *whole-part* hierarchies

Recursive composition

Client can treat individual objects and compositions of objects *uniformly*

Trade off some violations for *transparency*

Design Skeleton



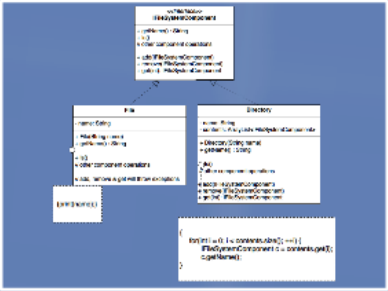
Composite

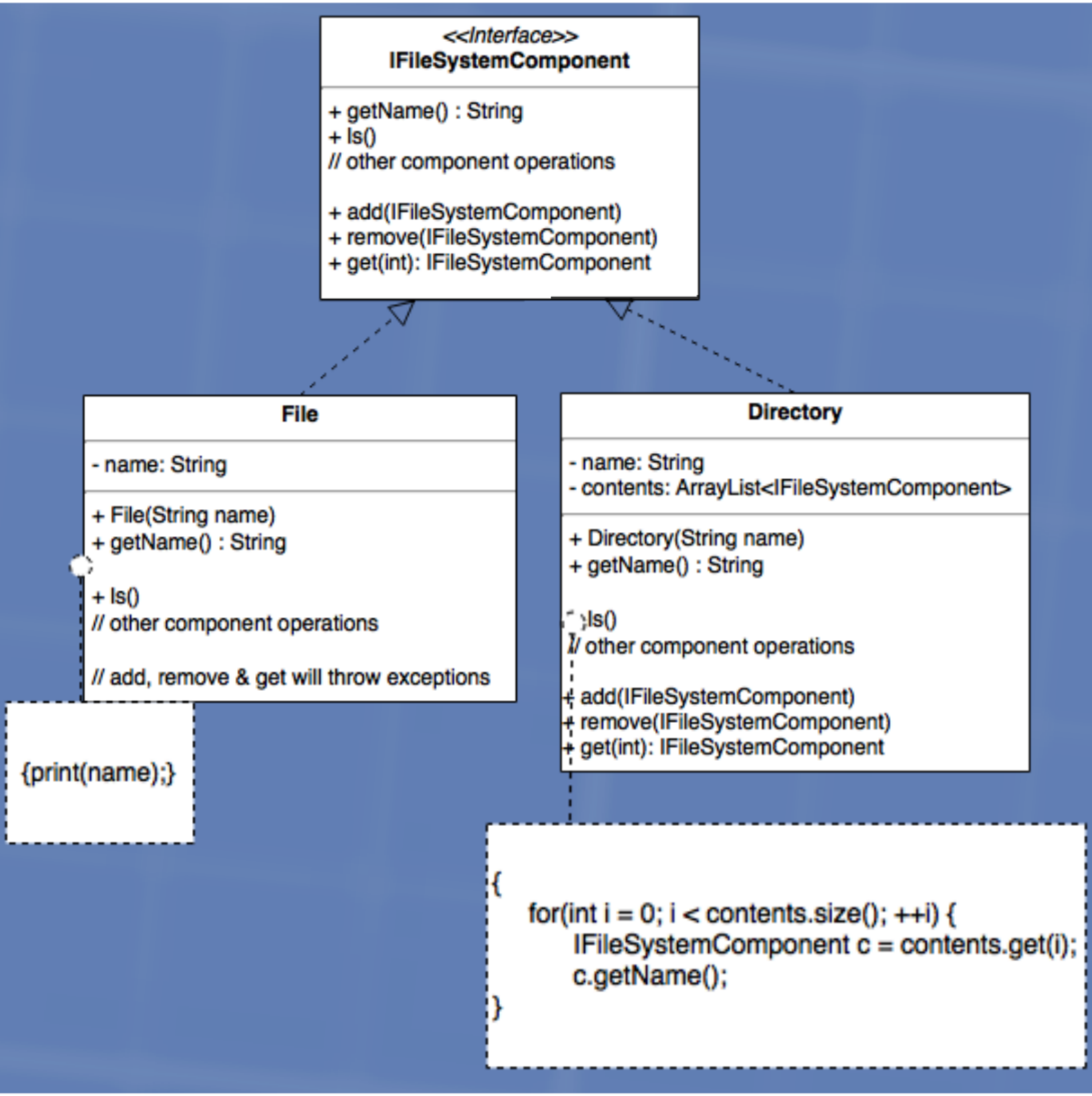
Compose objects into tree structures to represent *whole-part* hierarchies

Recursive composition

Client can treat individual objects and compositions of objects *uniformly*

Trade off some violations for *transparency*





<<Interface>>
IFileSystemComponent

+ getName() : String
+ ls()
// other component operations

+ add(IFileSystemComponent)
+ remove(IFileSystemComponent)
+ get(int): IFileSystemComponent

File

- name: String

+ File(String name)
+ getName() : String

+ ls()
// other component operations

// add, remove & get will throw exceptions

```
{print(name);}
```

Directory

- name: String
- contents: ArrayList<IFileSystemComponent>

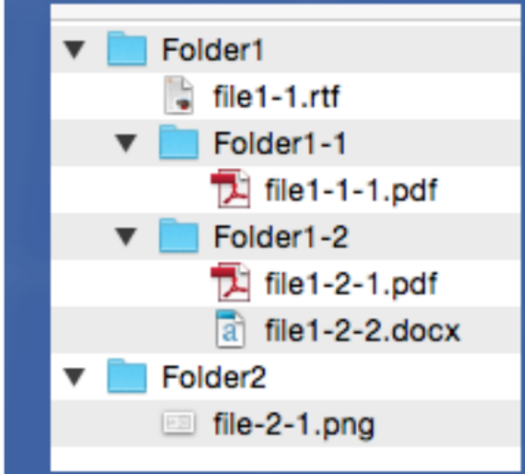
+ Directory(String name)
+ getName() : String

+ ls()
// other component operations

+ add(IFileSystemComponent)
+ remove(IFileSystemComponent)
+ get(int): IFileSystemComponent

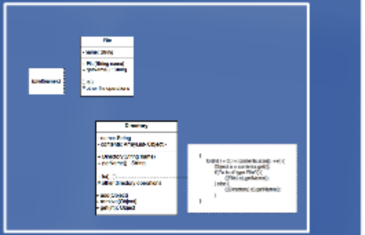
```
{
    for(int i = 0; i < contents.size(); ++i) {
        IFileSystemComponent c = contents.get(i);
        c.getName();
}
```

Think "Trees"...



Folder1
file1-1.rtf
Folder1-1
file1-1-1.pdf
Folder1-2
file1-2-1.pdf
file1-2-2.docx
Folder2
file-2-1.png

File system has *files* and *directories*



Directories can *contain* files & sub-directories

Several *operations* on files & folders have common *interface*

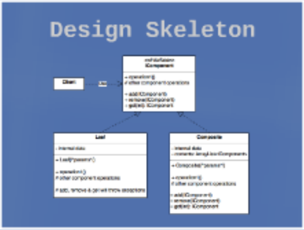
Composite

Compose objects into tree structures to represent *whole-part* hierarchies

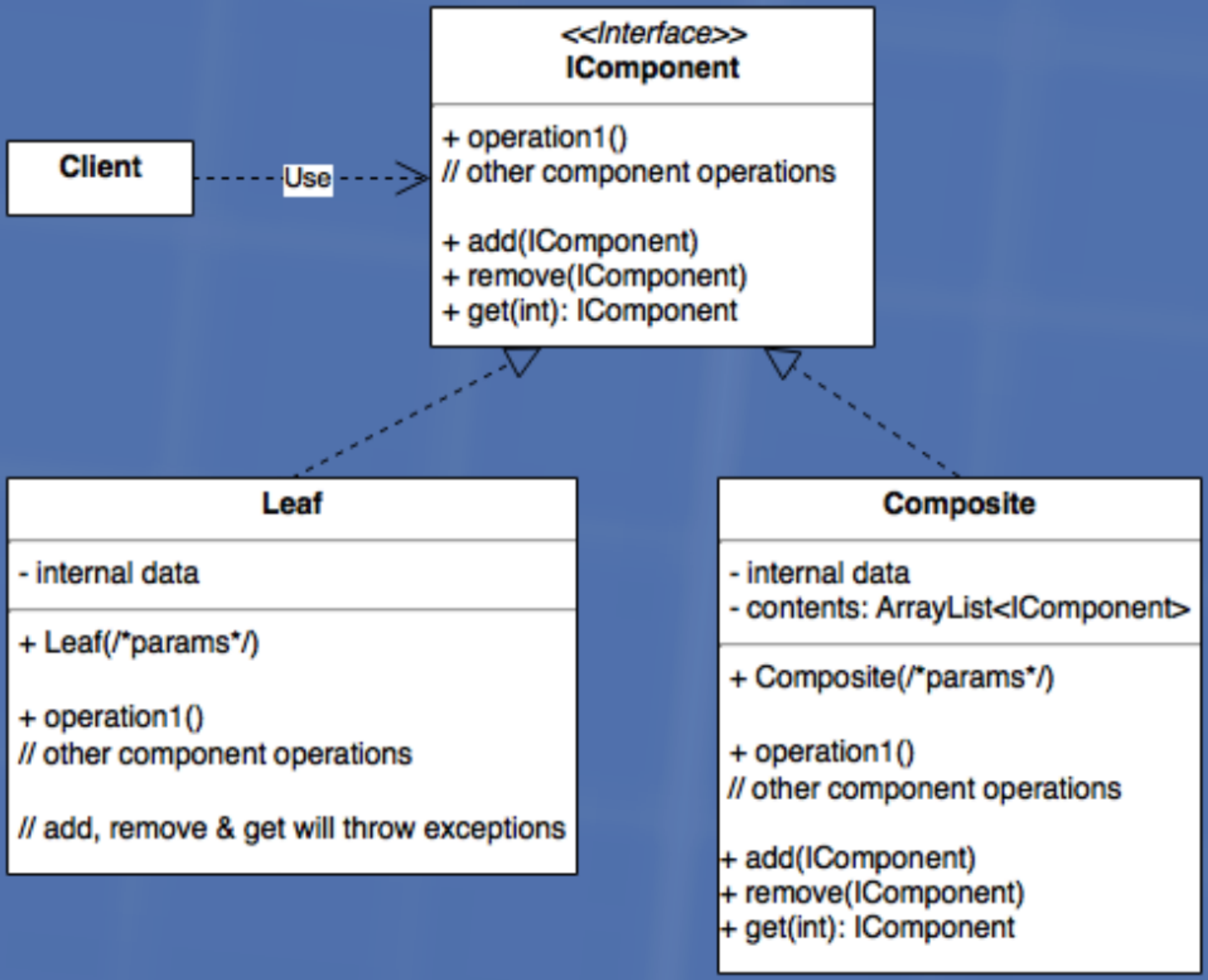
Recursive composition

Client can treat individual objects and compositions of objects *uniformly*

Trade off some violations for *transparency*



Design Skeleton



<<Interface>>
IComponent

+ operation1()
// other component operations

+ add(IComponent)
+ remove(IComponent)
+ get(int): IComponent

Client

Leaf

- internal data

+ Leaf(/*params*/)

+ operation1()
// other component operations

// add, remove & get will throw exceptions

Composite

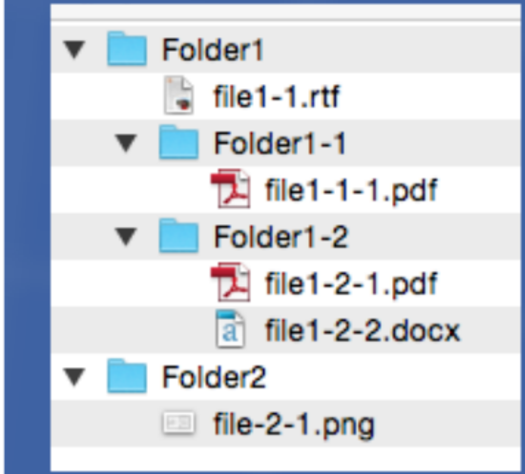
- internal data
- contents: ArrayList<IComponent>

+ Composite(/*params*/)

+ operation1()
// other component operations

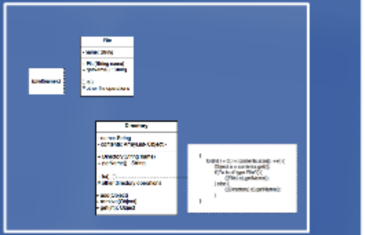
+ add(IComponent)
+ remove(IComponent)
+ get(int): IComponent

Think "Trees"...



- Folder1
  - file1-1.rtf
  - Folder1-1
    - file1-1-1.pdf
  - Folder1-2
    - file1-2-1.pdf
    - file1-2-2.docx
- Folder2
  - file-2-1.png

File system has *files* and *directories*



Directories can *contain* files & sub-directories

Several *operations* on files & folders have common *interface*

Composite

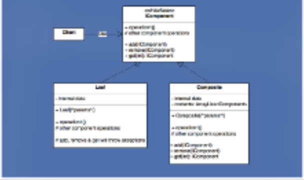
Compose objects into tree structures to represent *whole-part* hierarchies

Recursive composition

Client can treat individual objects and compositions of objects *uniformly*

Trade off some violations for *transparency*

Design Skeleton



<<interface>>
IComponent
+ operation()
// other component operations
+ add(IComponent)
+ remove(IComponent)
+ get(int): IComponent
Client
Use
Leaf
- internal data
+ Leaf("params")
+ operation()
// other component operations
// add, remove & get will throw exceptions
Composite
- internal data
- contents: ArrayList<IComponent>
+ Composite("params")
+ operation()
// other component operations
+ add(IComponent)
+ remove(IComponent)
+ get(int): IComponent