



Java is a trademark of Sun Microsystems, Inc.

JavaOneSM

Hacking the File System with JDK™ Release 7

Alan Bateman
Sun Microsystems Inc.

Carl Quinn
Netflix Inc.

Agenda

- Introduction
- Path operations
- File operations
- Directories

Agenda

- Introduction
- Path operations
- File operations
- Directories
- Recursive operations
- File change notification
- File attributes
- Provider interface
- Conclusion

Introduction: Basic Concepts

- FileRef
 - Reference to a file

Introduction: Basic Concepts

- FileRef
 - Reference to a file
- Path
 - Locates a file using a system dependent path

Introduction: Basic Concepts

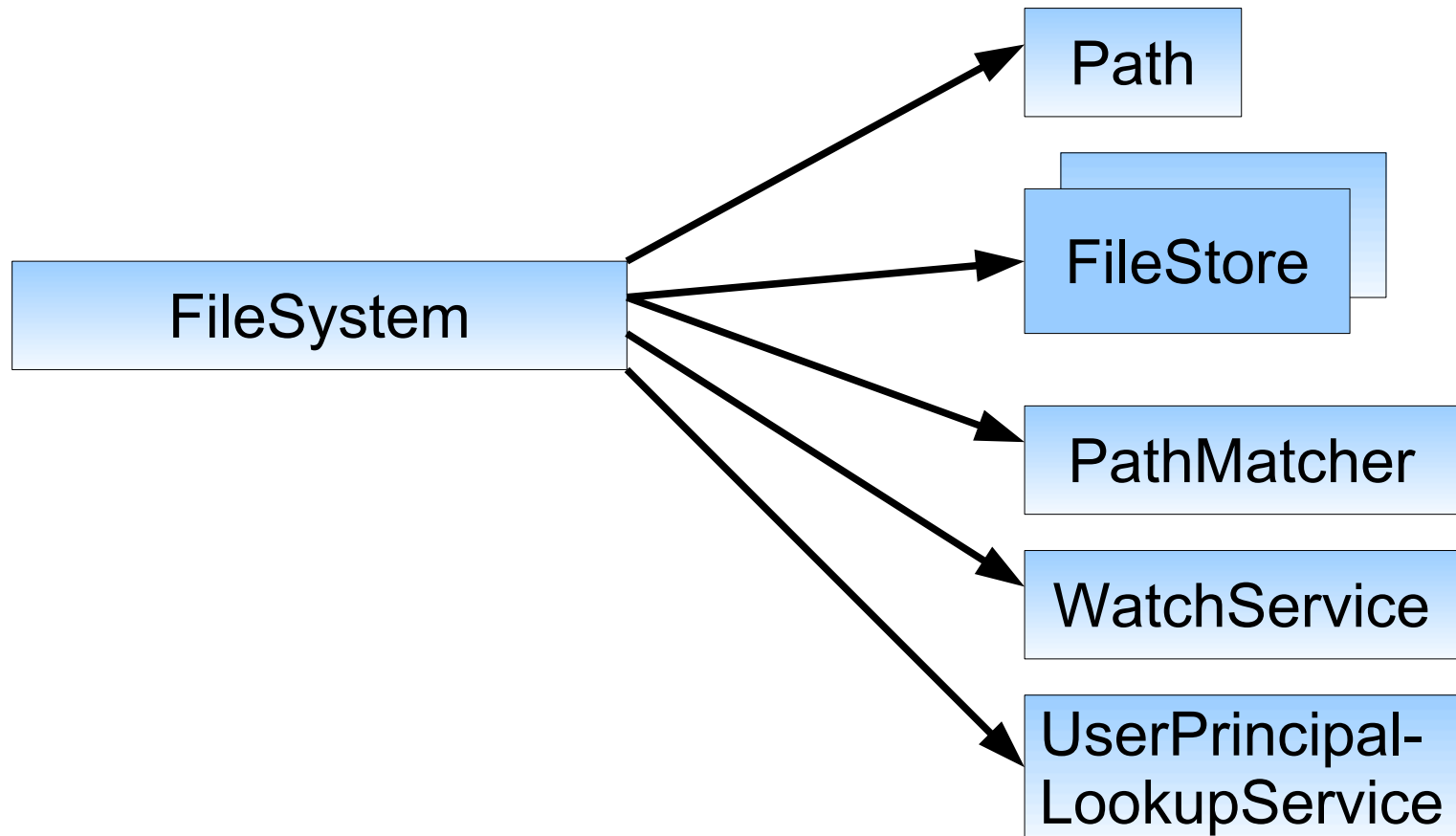
- FileRef
 - Reference to a file
- Path
 - Locates a file using a system dependent path
- FileSystem
 - Provides interface to file system
 - Factory for objects to access files and other objects in the file system.
 - Default file system for local/platform file system

FileSystem as a factory

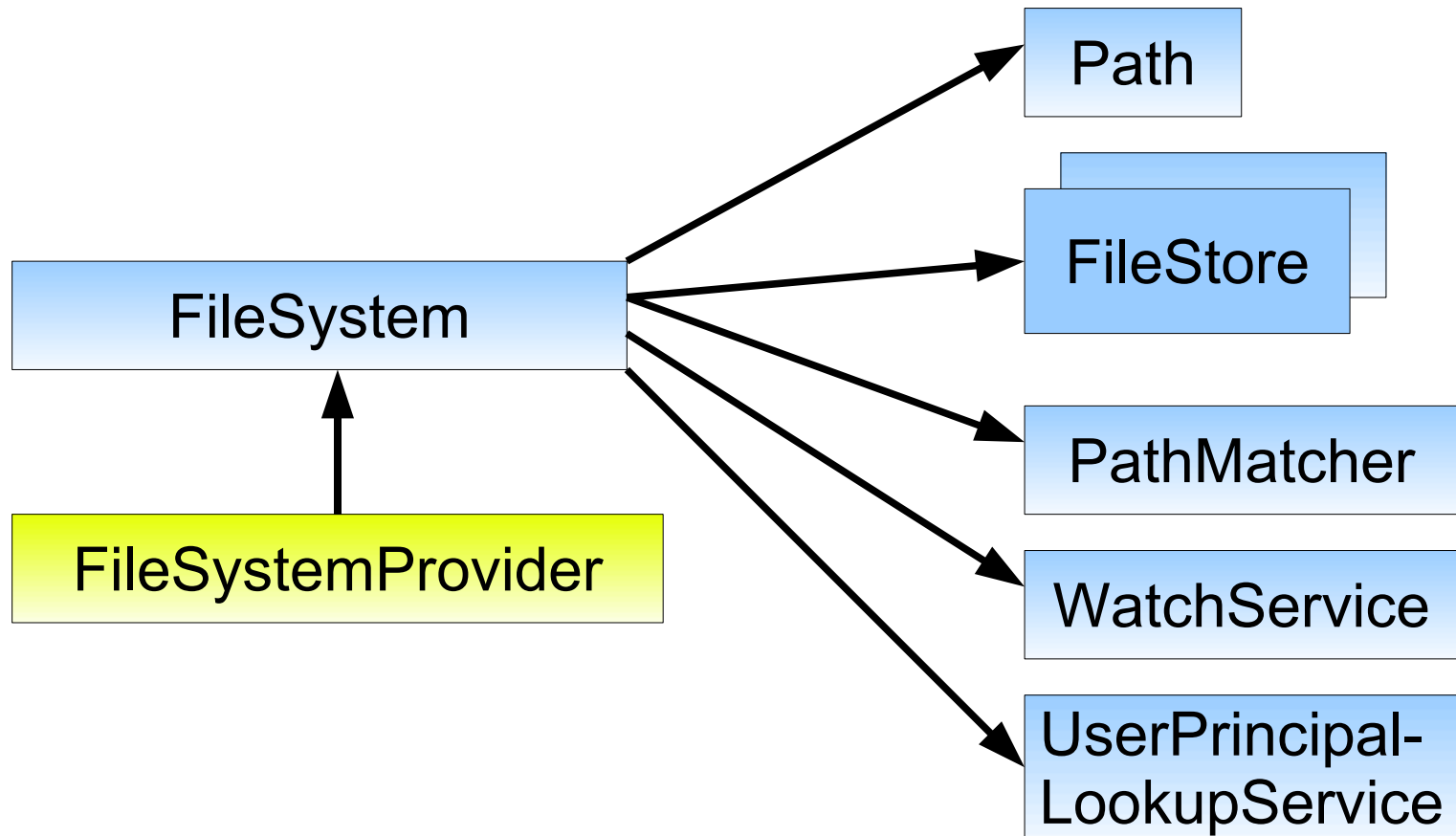
A light blue rectangular box with a thin black border, containing the text "FileSystem".

FileSystem

FileSystem as a factory



FileSystem as a factory



Path

- Immutable
- Create from path String or URI
- Defines methods to access and manipulate paths
- Defines methods to access files
- `File.toPath()` for interoperability

Path operations

- Accessing components
 - getName, getParent, getRoot, subpath
- Testing and comparing
 - startsWith, endsWith, equals, compareTo
- Combining
 - resolve (and inverse: relativize)

File operations

- I/O:
 - `newInputStream/newOutputStream` for stream I/O

File operations

- I/O:
 - `newInputStream/newOutputStream` for stream I/O
 - `newSeekableByteChannel` for channel I/O
 - `SeekableByteChannel = ByteChannel + file position`
 - cast to `FileChannel` for advanced operations such as file locking, memory mapped I/O, ...

File operations

- I/O:
 - `newInputStream/newOutputStream` for stream I/O
 - `newSeekableByteChannel` for channel I/O
 - `SeekableByteChannel = ByteChannel + file position`
 - cast to `FileChannel` for advanced operations such as file locking, memory mapped I/O, ...
- Other operations:
 - `createFile`, `createDirectory`, `delete`, `copyTo`, `moveTo`, `isSameFile`, `toRealPath`, `checkAccess` (`exists`, `notExists`), `createSymbolicLink`, `readSymbolicLink`, ...

Exceptions

- All methods that access file system may throw IOException
 - Specific exceptions for specific failures
 - Specific exceptions extend FileSystemException
- All other exceptions in API are unchecked

Directories

- DirectoryStream to iterate over the entries
 - Scales to large directories
 - Uses less resources
 - Smooth out response time for remote file systems
 - Provides handle to open directory
- Filter using glob, regex, or custom filter

Recursive Operations

- `Files.walkFileTree`
 - Walks a file tree rooted at a given starting file

Recursive Operations

- Files.walkFileTree
 - Walks a file tree rooted at a given starting file
 - Invoke FileVisitor method for each file/directory

```
interface FileVisitor<T> {  
    FileVisitResult preVisitDirectory(T dir);  
    FileVisitResult preVisitDirectoryFailed(T dir, IOException exc);  
    FileVisitResult visitFile(T file, BasicFileAttributes attrs);  
    FileVisitResult visitFileFailed(T file, IOException exc);  
    FileVisitResult postVisitDirectory(T dir, IOException exc);  
}
```

Recursive Operations

- `Files.walkFileTree`
 - Walks a file tree rooted at a given starting file
 - Invoke `FileVisitor` method for each file/directory
 - `SimpleFileVisitor` with default behavior

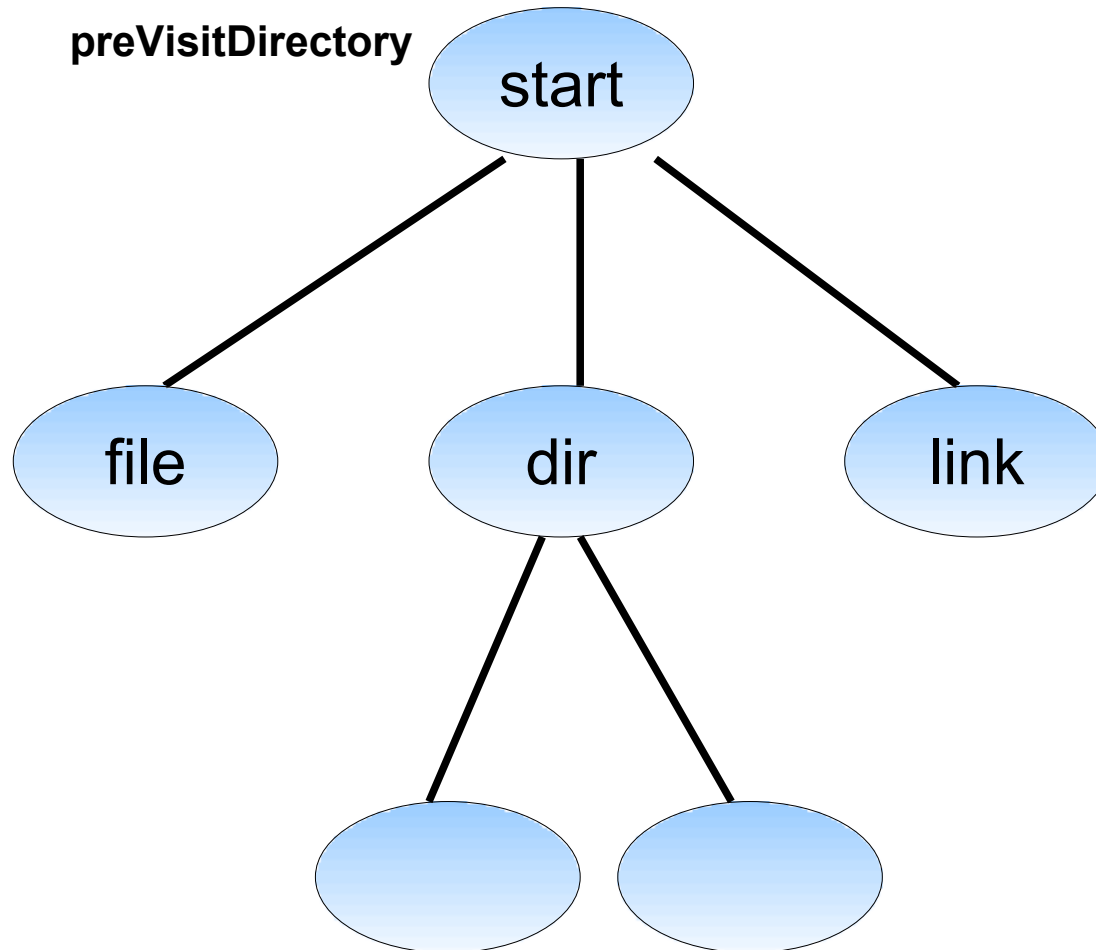
Recursive Operations

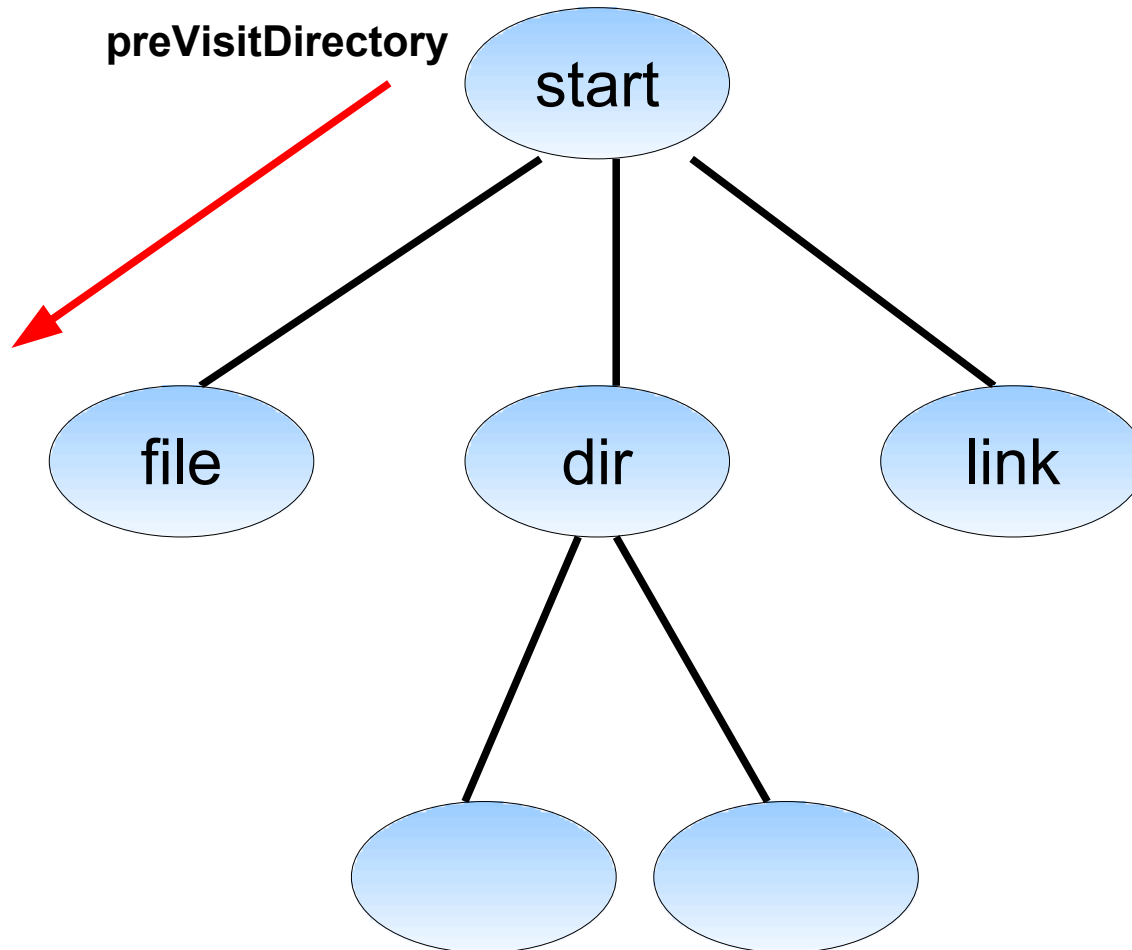
- Files.walkFileTree
 - Walks a file tree rooted at a given starting file
 - Invoke FileVisitor method for each file/directory
 - SimpleFileVisitor with default behavior
 - Return value controls iteration

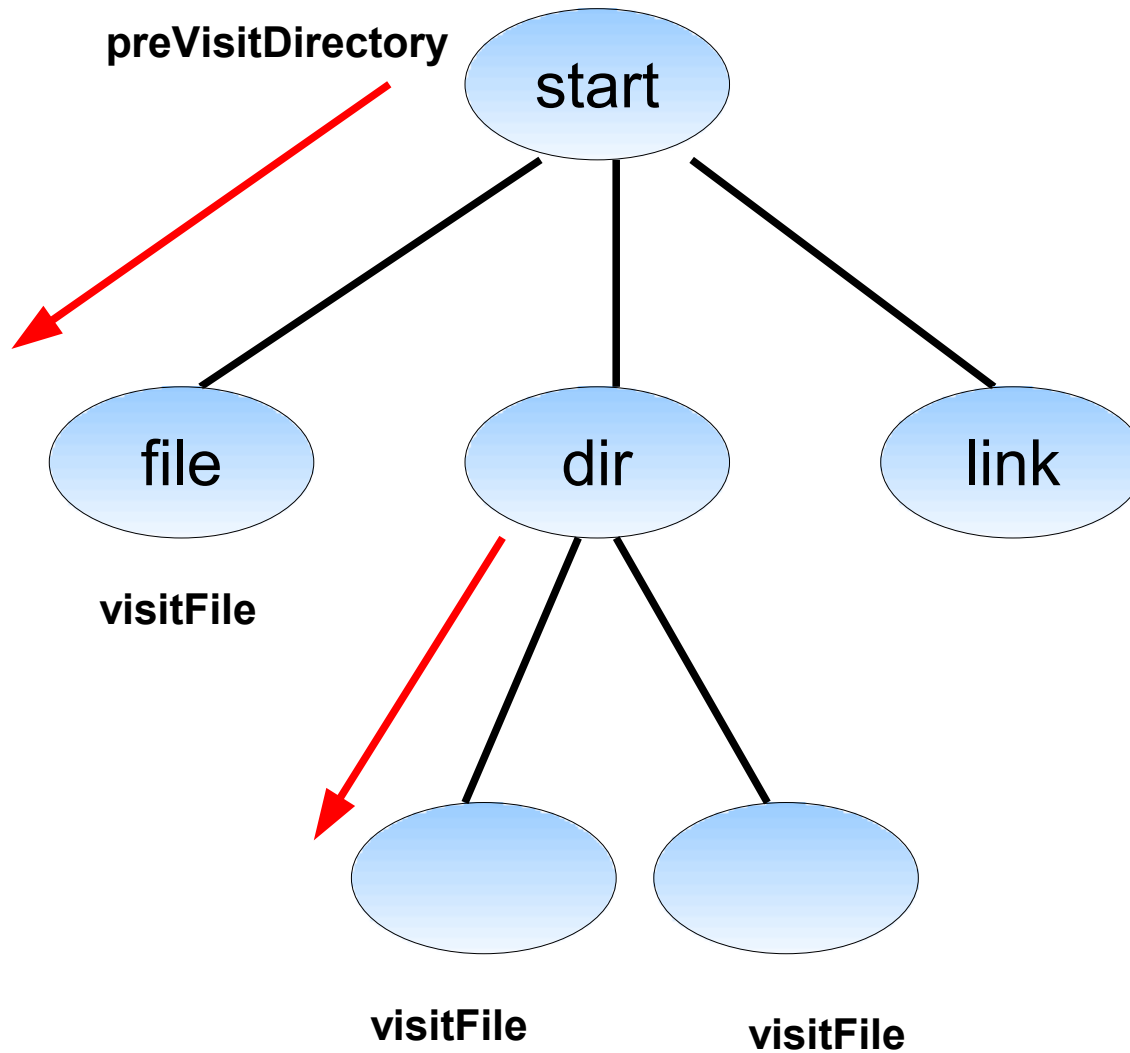
```
enum FileVisitResult {  
    CONTINUE,  
    TERMINATE,  
    SKIP_SUBTREE,  
    SKIP_SIBLINGS  
}
```

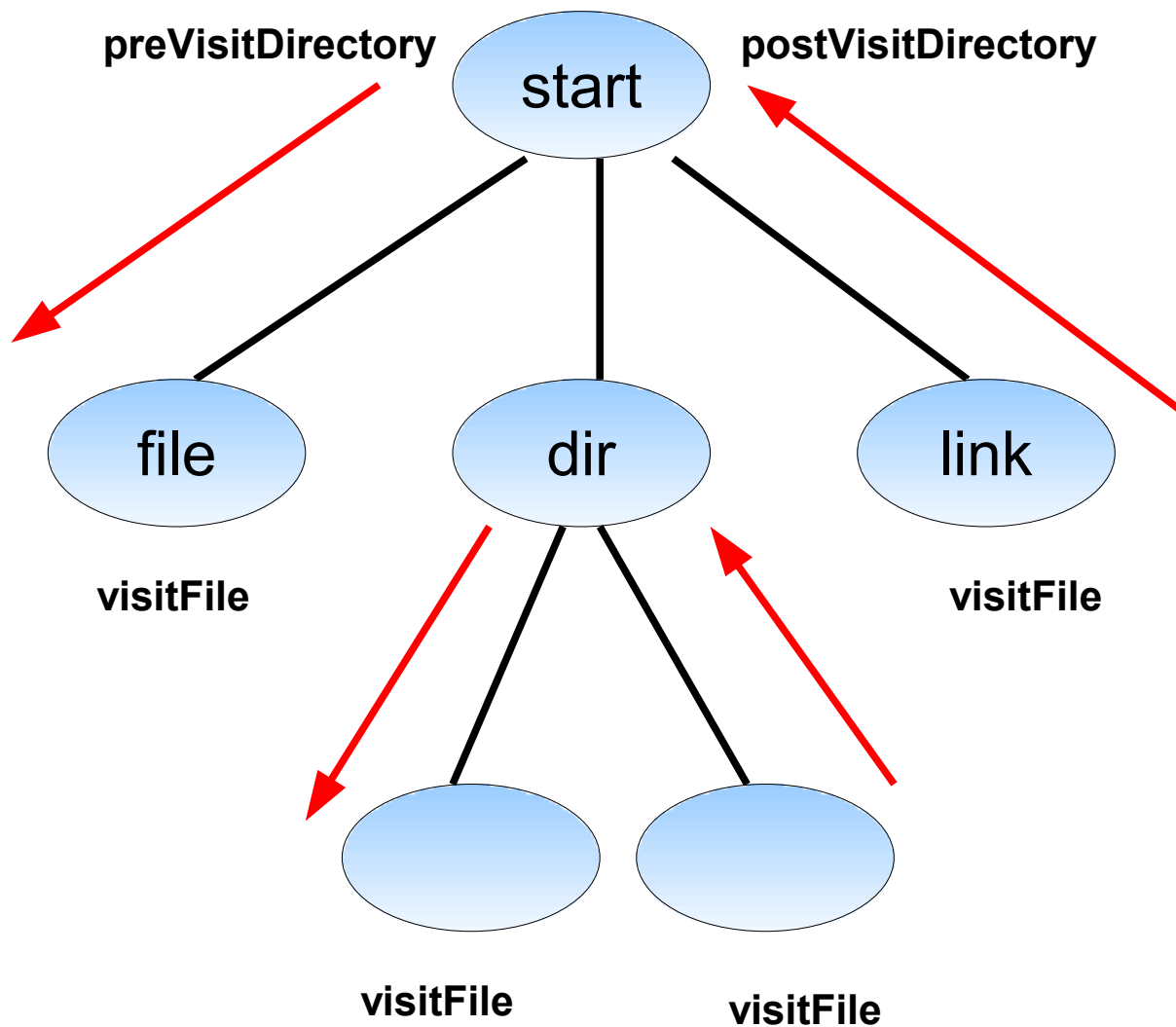
Recursive Operations

- `Files.walkFileTree`
 - Walks a file tree rooted at a given starting file
 - Invoke `FileVisitor` method for each file/directory
 - `SimpleFileVisitor` with default behavior
 - Return value controls iteration
 - Options control if sym links are followed
 - Not followed by default
 - When following then cycles are detected and reported









File Change Notification

- Improve performance of applications that are forced to poll the file system today

File Change Notification

- Improve performance of applications that are forced to poll the file system today
- WatchService
 - Watch registered objects for events and changes
 - Makes use of native event facility where available
 - Supports concurrent processing of events

File Change Notification

- Improve performance of applications that are forced to poll the file system today
- WatchService
 - Watch registered objects for events and changes
 - Makes use of native event facility where available
 - Supports concurrent processing of events
- Path implements Watchable
 - Register directory to get events when entries are created, deleted, or modified

WatchService API

- WatchKey represents registration of a watchable object with a WatchService.

```
// register
WatchKey key = file.register(watcher, ENTRY_CREATE, ENTRY_MODIFY);

// cancel registration
key.cancel();
```

WatchService API

- WatchKey represents registration of a watchable object with a WatchService.
- WatchKey signalled and queued when event detected
- WatchService poll or take methods used to retrieve signalled key.

```
class WatchService {  
    WatchKey take();  
    WatchKey poll();  
    WatchKey poll(long timeout, TimeUnit unit);  
}
```

WatchService API

- WatchKey represents registration of a watchable object with a WatchService.
- WatchKey signalled and queued when event detected
- WatchService poll or take methods used to retrieve signalled keys
- Process events
- WatchKey reset method returns key to ready state

File Attributes

- Meta-data associated with file
- Long standing requests from applications

File Attributes

- Meta-data associated with file
- Long standing requests from applications
 - File owner
 - Permissions
 - Timestamps
 - DOS attributes
 - Extended attributes
 - ...

File Attributes

- Meta-data associated with file
- Long standing requests from applications
- Highly file system specific

File Attributes

- Group related attributes
- Define *view* of those attributes
 - Typesafe methods to access attributes in group
 - Bulk access where applicable
 - Converts to/from file system representation
- Implementation requires to support basic view
 - Attributes common to most file systems
 - BasicFileAttributes in examples
- Implementation may support others

File Attributes

- `java.nio.file.attribute` package
- basic view: `BasicFileAttributeView`
- posix view: `PosixFileAttributeView`
- acl view: `AclFileAttributeView`
 - Based on NFSv4 ACL model
- owner view: `FileOwnerAttributeView`
- user-defined view: `UserDefinedFileAttributeView`

Attributes class makes it easy

```
BasicFileAttributes attrs =  
    Attributes.readBasicFileAttributes(file);
```

```
BasicFileAttributes attrs =  
    Attributes.readBasicFileAttributes(file, NOFOLLOW_LINKS);
```

```
FileTime lastModifiedTime = ...  
Attributes.setLastModifiedTime(file, lastModifiedTime);
```

```
PosixFileAttributes attrs =  
    Attributes.readPosixFileAttributes(file);
```

```
Set<PosixFilePermission> perms = ...  
Attributes.setPosixFilePermissions(file, perms);
```

```
UserPrincipal owner = Attributes.getOwner(file);
```

File Attributes

- Dynamic access
 - treat attributes as name/value pairs

```
boolean isSymbolicLink
    (Boolean)file.getAttribute("isSymbolicLink", NOFOLLOW_LINKS);

Set<PosixFilePermission> perms = ...
file.setAttribute("posix:permissions", perms);

Map<String,?> attrs = file.readAttributes("basic:*");
```

File Attributes

- Dynamic access
 - treat attributes as name/value pairs
 - loose type safety
- Ask FileSystem for the set of supported views
- Ask underlying FileStore if it supports view
- Set initial attributes when creating files
 - important for security related attributes

```
FileAttribute<Set<PosixFilePermission>> perms = ...  
f.createFile(perms);
```

Provider interface

- Used to develop and deploy custom file systems
 - Desktop file system
 - Memory file system
- Provider identified by URI scheme
- Factory for FileSystem instances
- Default provider required (**file:///**)
- Can replace or interpose on default provider
 - Create virtual file system
 - java.io uses default provider

Conclusion

- Finally address shortcomings of `java.io.File`
- Easy to use for basic and common operations
- Provides access to more advanced facilities for applications that require it
- Extensible via provider mechanism

More information

- BOF-5087: All Things I/O with JDK™ Release 7, Thursday @ 6:30pm, Gateway 102-103.
- OpenJDK New I/O Project
 - <http://openjdk.java.net/projects/nio>
- Blogs
 - <http://blogs.sun.com/alanb>



JavaOneSM

Thank You

Alan Bateman
Sun Microsystems Inc.

Carl Quinn
Netflix Inc.

