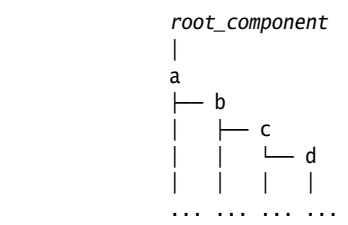
21.1 Characteristics of a Hierarchical File System

We first look at some characteristics of a hierarchical file system, and introduce the terminology used in this chapter.

Hierarchical File Systems

A *file system* allows persistent storage and organization of data as a *hierarchical* (or *tree*) structure on some external media. A tree structure has a *root component* (also called *root node*) at the top, under which other tree nodes represent files and directories. Each directory can have files and nested directories (often called *subdirectories*). An operative system can have multiple file systems, each of which is identifiable by its root component.

Below is an example of a file system, where the root component is platform dependent. On Unix-based platforms, the root component of the file system is denoted by the slash character (/). The root component for Windows-based platforms is a combination of a *volume* name (i.e., file system name), a colon (:), and a backslash character (\). For example, `C:\` designates the root component of the ubiquitous volume named `C` on a Windows-based platform.



When it is not necessary to distinguish between a file and a directory in the file system, we will use the term *directory entry* to mean both. Each directory entry in a hierarchical file system is uniquely identifiable by a *path* from the root component to the node representing the directory entry. The path of a directory entry in a file system is specified using the naming conventions of the host system, where *name elements* that comprise the path are separated by a platform-specific *name separator character* or *delimiter*. The name separator character for name elements on Unix-based platforms is the slash character (/), whereas on Windows-based platforms, it is the backslash character (\). For the most part, we will use conventions for Unix-based platforms.

Two examples of paths are given below, where each path has three name elements.

`/a/b/c`	on Unix-based platforms
`C:\a\b\c`	on Windows-based platforms

Absolute and Relative Paths

Directory entries can be referenced using both *absolute* and *relative* paths, but the path naming must follow the conventions of the host platform.

An absolute path starts with the platform-dependent root component of the file system, as in the examples above. All information is contained in an absolute path to reference the directory entry—that is, an absolute path uniquely identifies a directory entry in the file system.

A relative path is without designation of the root component and therefore requires additional path information to locate its directory entry in the file system. Typically, a relative path is interpreted in relation to the *current directory* (see the next subsection). Some examples of relative paths are given below, but they alone are not enough to uniquely identify a directory entry.

`c/d`	on Unix-based platforms
`c\d`	on Windows-based platforms

Note that in a path, the name elements are all parent directory names, except for the last name element, which can be either a file name or a directory name. The name of a directory entry does not distinguish whether it is a file or a directory in the file system. Although *file extensions* can be used in a file name for readability, it is immaterial for the file system. Care must also be exercised when choosing name elements, as characters allowed are usually platform dependent.

Java programs should not rely on system-specific path conventions. In the next section we will construct paths in a platform-independent way.

Current and Parent Directory Designators

A file system has a notion of a current directory which changes while traversing in the file system. The *current directory* is designated by the period character (`.`) and its *parent directory* by two periods (`..`). These designators can be used in constructing paths. Given that the current directory is `/a/b` in the directory tree shown earlier, Table 21.1 illustrates relative paths constructed using the current and the parent directory designators, and their corresponding absolute paths.

Table 21.1 *Using Current and Parent Directory Designators*

Relative path (Current directory: `/a/b`)	**Absolute path**
`./c/d`	`/a/b/c/d`
`.`	`/a/b`
`./..`	`/a`
`./../..`	`/` (i.e., the root component)

Symbolic Links

Apart from regular files, a file system may allow symbolic links to be created. A *symbolic link* (also called *a hard link*, *a shortcut*, or *an alias*) is a special file that acts as a reference to a directory entry, called the *target* of the symbolic link. Creating an *alias* of a directory entry in the file system is similar to creating a symbolic link. Using symbolic links is transparent to the user, as the file system takes care of using the target when a symbolic link is used in a file operation, with one caveat: Deleting the symbolic link does *not* delete the target. In many file operations, it is possible to indicate whether symbolic links should be followed or not.