

15.3 Using NIO Classes and Interfaces to Get File and Directory Information

Interfaces `Path` and `DirectoryStream` and classes

`Paths` and `Files` (all from package `java.nio.file`) are useful for retrieving information about files and directories on disk:

- `Path` interface—Objects of classes that implement `Path` represent the location of a file or directory. `Path` objects do not open files or provide any file-processing capabilities. Class `File` (package `java.io`) also is used commonly for this purpose.
- `Paths` class—Provides `static` methods used to get a `Path` object representing a file or directory location.
- `Files` class—Provides `static` methods for common file and directory manipulations, such as copying files; creating and deleting files and directories; getting information about files and directories; reading the contents of files; getting objects that allow you to manipulate the contents of files and directories; and more.
- `DirectoryStream` interface—Objects of classes that implement this interface enable a program to iterate through the contents of a directory.

Creating Path Objects

You'll use class `static` method `get` of class `Paths` to convert a `String` representing a file's or directory's location

into a **Path** object. You can then use the methods of interface **Path** and class **Files** to determine information about the specified file or directory. We discuss several such methods momentarily. For complete lists of their methods, visit:

<http://docs.oracle.com/javase/8/docs/api/java/nio/file/Path.html>
<http://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html>



Absolute vs. Relative Paths

A file or directory’s path specifies its location on disk. The path includes some or all of the directories leading to the file or directory. An **absolute path** contains *all* directories, starting with the **root directory**, that lead to a specific file or directory. Every file or directory on a particular disk drive has the *same* root directory in its path. A **relative path** is “relative” to another directory—for example, a path relative to the directory in which the application began executing.

Getting Path Objects from URIs

An overloaded version of **Files static** method **get** uses a **URI** object to locate the file or directory. A **Uniform Resource Identifier (URI)** is a more general form of the **Uniform Resource Locators (URLs)** that are used to locate

websites. For example, the URL
`http://www.deitel.com/` is the URL for the Deitel & Associates website. URIs for locating files vary across operating systems. On Windows platforms, the URI

`file:///C:/data.txt`



identifies the file `data.txt` stored in the root directory of the C: drive. On UNIX/Linux platforms, the URI

`file:/home/student/data.txt`



identifies the file `data.txt` stored in the `home` directory of the user `student`.

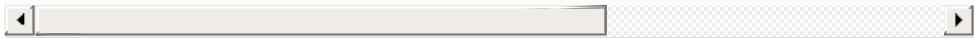
Example: Getting File and Directory Information

Figure 15.2 prompts the user to enter a file or directory name, then uses classes `Paths`, `Path`, `Files` and `DirectoryStream` to output information about that file or directory.

```
1 // Fig. 15.2: FileAndDirectoryInfo.java
2 // File class used to obtain file and directory
3 import java.io.IOException;
4 import java.nio.file.DirectoryStream;
```

```
5   import java.nio.file.Files;
6   import java.nio.file.Path;
7   import java.nio.file.Paths;
8   import java.util.Scanner;
9
10  public class FileAndDirectoryInfo {
11      public static void main(String[] args) throws
12          Scanner input = new Scanner(System.in);
13
14      System.out.println("Enter file or director
15
16      // create Path object based on user input
17      Path path = Paths.get(input.nextLine());
18
19      if (Files.exists(path)) { // if path exist
20          // display file (or directory) informat
21          System.out.printf("%n%s exists%n", path
22          System.out.printf("%s a directory%n",
23              Files.isDirectory(path) ? "Is" : "Is
24          System.out.printf("%s an absolute path%
25              path.isAbsolute() ? "Is" : "Is not")
26          System.out.printf("Last modified: %s%n"
27              Files.getLastModifiedTime(path));
28          System.out.printf("Size: %s%n", Files.s
29          System.out.printf("Path: %s%n", path);
30          System.out.printf("Absolute path: %s%n"
31
32          if (Files.isDirectory(path)) { // output
33              System.out.printf("%nDirectory conte
34
35          // object for iterating through a di
36          DirectoryStream<Path> directoryStrea
37          Files.newDirectoryStream(path);
38
39          for (Path p : directoryStream){
40              System.out.println(p);
41          }
42      }
43
44      else { // not file or directory, output er
```

```
45         System.out.printf("%s does not exist%n"
46             }
47     } // end main
48 } // end class FileInfoAndDirectory
```



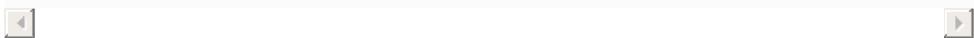
Enter file or directory name:

c:\examples\ch15

```
    ch15 exists
    Is a directory
    Is an absolute path
Last modified: 2013-11-08T19:50:00.838256Z
    Size: 4096
    Path: c:\examples\ch15
    Absolute path: c:\examples\ch15
```

Directory contents:

```
    C:\examples\ch15\fig15_02
    C:\examples\ch15\fig15_12_13
    C:\examples\ch15\SerializationApps
    C:\examples\ch15\TextFileApps
```



Enter file or directory name:

C:\examples\ch15\fig15_02\FileInfoAndDirectory.java

```
FileInfoAndDirectory.java exists
    Is not a directory
    Is an absolute path
Last modified: 2013-11-08T19:59:01.848255Z
    Size: 2952
Path: C:\examples\ch15\fig15_02\FileInfoAndDirectory.java
Absolute path: C:\examples\ch15\fig15_02\FileInfoAndDirectory.java
```



Fig. 15.2

`File` class used to obtain file and directory information.

The program begins by prompting the user for a file or directory (line 14). Line 17 inputs the filename or directory name and passes it to `Paths static` method `get`, which converts the `String` to a `Path`. Line 19 invokes `Files static` method `exists`, which receives a `Path` and determines whether it exists (either as a file or as a directory) on disk.

If the name does not exist, control proceeds to line 45, which displays a message containing the `Path's String` representation followed by “does not exist.” Otherwise, lines 21–42 execute:

- `Path` method `getFileName` (line 21) gets the `String` name of the file or directory without any location information.
- `Files static` method `isDirectory` (line 23) receives a `Path` and returns a `boolean` indicating whether that `Path` represents a directory on disk.
- `Path` method `isAbsolute` (line 25) returns a `boolean` indicating whether that `Path` represents an absolute path to a file or directory.
- `Files static` method `getLastModifiedTime` (line 27) receives a `Path` and returns a `FileTime` (package `java.nio.file.attribute`) indicating when the file was last modified. The program outputs the `FileTime`'s default `String` representation.
- `Files static` method `size` (line 28) receives a `Path` and returns a `long` representing the number of bytes in the file or directory. For directories, the value returned is platform specific.

- `Path` method `toString` (called implicitly at line 29) returns a `String` representing the `Path`.
- `Path` method `toAbsolutePath` (line 30) converts the `Path` on which it's called to an absolute path.

If the `Path` represents a directory (line 32), lines 36–37 use `Files static` method `newDirectoryStream` to get a `DirectoryStream<Path>` containing `Path` objects for the directory's contents. Lines 39–41 display the `String` representation of each `Path` in the `DirectoryStream<Path>`. Note that `DirectoryStream` is a generic type like `ArrayList` ([Section 7.16](#)).

The first output of this program demonstrates a `Path` for the folder containing this chapter's examples. The second output demonstrates a `Path` for this example's source-code file. In both cases, we specified an absolute path.



Error-Prevention Tip

15.1

Once you've confirmed that a `Path` exists, it's still possible that the methods demonstrated in [Fig. 15.2](#) will throw `IOExceptions`. For example, the file or directory represented by the `Path` could be deleted from the system after the call to `Files` method `exists` and before the other statements in lines 21–42 execute. Industrial strength file- and

directory-processing programs require extensive exception handling to deal with such possibilities.

Separator Characters

A **separator character** is used to separate directories and files in a path. On a Windows computer, the *separator character* is a backslash (\). On a Linux or macOS system, it's a forward slash (/). Java processes both characters identically in a pathname. For example, if we were to use the path

```
c:\Program Files\Java\jdk1.6.0_11\demo/jfc
```



which employs each separator character, Java would still process the path properly.



Good Programming Practice 15.1

When building Strings that represent path information, use File.separator to obtain the local computer's proper separator character rather than explicitly using / or \. This constant is a String consisting of one character—the proper separator for the system.



Common Programming Error 15.1

Using \ as a directory separator rather than \\ in a string literal is a logic error. A single \ indicates that the \ followed by the next character represents an escape sequence. Use \\ to insert a \ in a string literal.