IT602: Object-Oriented Programming

Lecture - 15

Files and Streams

Arpit Rana

5th April 2022

Input and Output

The **`java.io`** package provides an extensive library of classes for dealing with input and output.

Java provides ***streams*** that implement sequential access of data.

There are two kinds of streams: ***byte streams*** and ***character streams*** (a.k.a. *binary streams* and *text streams*, respectively).

- An input stream is an object that an application can use to read a sequence of data, and
- An output stream is an object that an application can use to write a sequence of data.
- An input stream acts as a source of data, and an output stream acts as a destination of data.

Input and Output

The following entities can act as both input and output streams:

- An array of bytes or characters
- A file
- A *pipe* (a mechanism by which a program can communicate data to another program during execution)
- A network connection

The **`java.io`** package also provides a general interface to interact with the file system of the host platform.

The **`File`** Class

The **`File`** class provides a general machine-independent interface for the file system of the underlying platform.

- A **`File`** object represents the pathname of a file or directory in the host file system.
- An application can use the functionality provided by the **`File`** class for handling files and directories in the file system.
- The **`File`** class is not meant for handling the contents of files. For that purpose, there are the `FileInputStream` and `FileOutputStream` classes,

The **`File`** Class

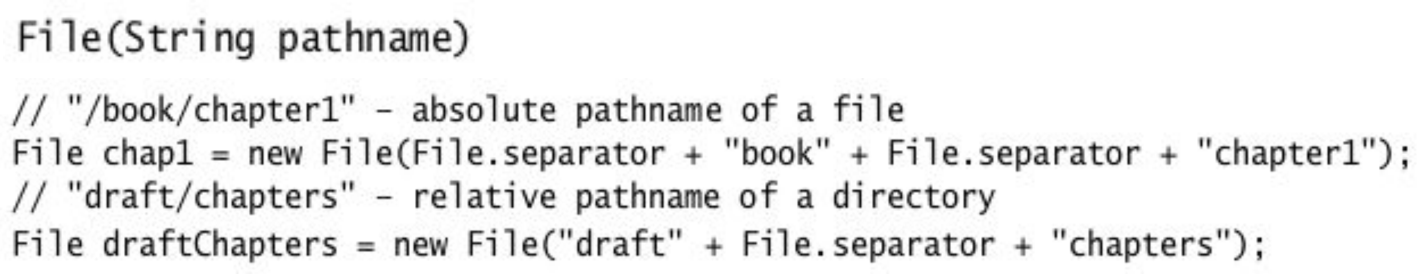
The File class has various constructors for associating a file or a directory pathname to an object of the File class.

- Creating a **`File`** object does not mean creation of any file or directory based on the pathname specified.
- A **`File`** instance, called the *abstract pathname*, is a representation of the pathname of a file and directory.
- The pathname cannot be changed once the **`File`** object is created.

The **`File`** Class

The pathname (of a file or a directory) can be an absolute pathname or a pathname relative to the current directory.

- An empty string as argument results in an abstract pathname for the current directory.



```
File(String pathname)

// "/book/chapter1" – absolute pathname of a file
File chap1 = new File(File.separator + "book" + File.separator + "chapter1");
// "draft/chapters" – relative pathname of a directory
File draftChapters = new File("draft" + File.separator + "chapters");
```

The **`File`** Class

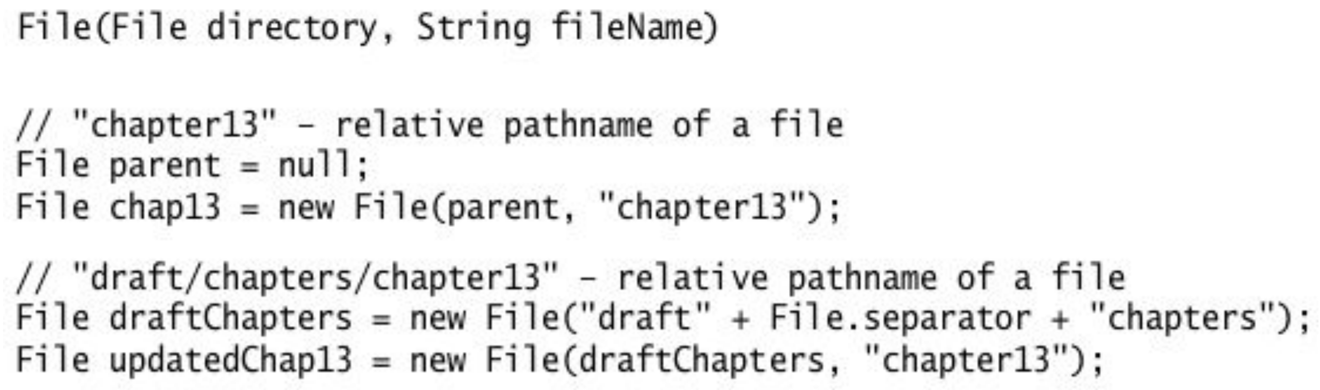
- This creates a `File` object whose pathname is as follows: `directoryPathname + separator + fileName.`

```
File(String directoryPathname, String fileName)
```

```
// "/book/chapter1" - absolute pathname of a file
File updatedChap1 = new File(File.separator + "book", "chapter1");
```

The **`File`** Class

- If the directory argument is `null` , the resulting **`File`** object represents a file in the current directory.
- If the directory argument is not `null`, it creates a **`File`** object that represents a file in the given directory.
    - The pathname of the file is then the pathname of the directory **`File`** `object + separator + fileName`



```
File(File directory, String fileName)

// "chapter13" - relative pathname of a file
File parent = null;
File chap13 = new File(parent, "chapter13");

// "draft/chapters/chapter13" - relative pathname of a file
File draftChapters = new File("draft" + File.separator + "chapters");
File updatedChap13 = new File(draftChapters, "chapter13");
```

The **`File`** Class

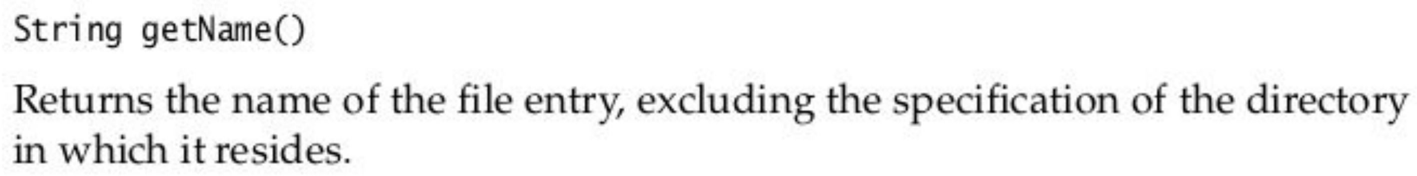
An object of the **`File`** class provides a handle to a file or directory in the file system, and can be used to create, rename, and delete the entry.

A **`File`** object can also be used to query the file system for information about a file or directory:

- whether the entry exists
- whether the **`File`** object represents a file or directory
- get and set read, write, or execute permissions for the entry
- get pathname information about the file or directory
- list all entries under a directory in the file system

Querying the File System

The File class provides a number of methods for obtaining the platform-dependent representation of a pathname and its components.



```
String getName()
```

Returns the name of the file entry, excluding the specification of the directory in which it resides.

- On Unix, the name part of "/book/chapters/one" is "one" .
- On Windows platforms, the name part of "c:\java\bin\javac" is "javac" .
- On the Macintosh, the name part of "HD:java-tools:javac" is "javac" .

Querying the File System

The File class provides a number of methods for obtaining the platform-dependent representation of a pathname and its components.

```
String getPath()

String getAbsolutePath()

String getParent()

long lastModified()

long length()
```

File or Directory Existence

A **`File`** object is created using a pathname.

Whether this pathname denotes an entry that actually exists in the file system can be checked using the `exists()` method:

```
boolean exists()
```

Since a File object can represent a file or a directory, the following methods can be used to distinguish whether a given File object represents a file or a directory, respectively:

```
boolean isFile()
boolean isDirectory()
```

File and Directory Permissions

Write, read and execute permissions can be set by calling the following methods.

```
boolean setReadable(boolean readable)
boolean setReadable(boolean readable, boolean owner)

boolean setWritable(boolean writable)
boolean setWritable(boolean writable, boolean owner)

boolean setExecutable(boolean executable)
boolean setExecutable(boolean executable, boolean owner)
```

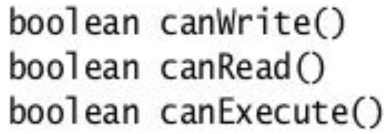
These methods throw a `SecurityException` if permission cannot be changed.

If the first argument is true, the operation permission is set; otherwise it is cleared.

If the second argument is true, the permission only affects the owner; otherwise it affects all users.

File and Directory Permissions

To check whether the specified file has write, read, or execute permissions, the following methods can be used.



```
boolean canWrite()
boolean canRead()
boolean canExecute()
```

They throw a `SecurityException` if general access is not allowed, i.e., the application is not even allowed to check whether it can read, write or execute a file.

Creating New Files and Directories

The **`File`** class can be used to create files and directories.

A file can be created whose pathname is specified in a **`File`** object using the following method:

```
boolean createNewFile() throws IOException
```

It creates a new, empty file named by the abstract pathname if, and only if, a file with this name does not already exist.

The returned value is `true` if the file was successfully created, `false` if the file already exists.

Any I/O error results in an `IOException.`

Renaming and Deleting Files and Directories

Renaming Files and Directories

A file or a directory can be renamed, using the following method which takes the new pathname from its argument. It throws a `SecurityException` if access is denied.

boolean renameTo(File dest)

Deleting Files and Directories

A file or a directory can be deleted using the following method. In the case of a directory, it must be empty before it can be deleted. It throws a `SecurityException` if access is denied.

boolean delete()

IT602: Object-Oriented Programming

Next lecture -

Byte Streams