

Java.io.File Class in Java

Difficulty Level : Medium Last Updated : 23 Jan, 2022

The File class is Java's representation of a file or directory pathname. Because file and directory names have different formats on different platforms, a simple string is not adequate to name them. The File class contains several methods for working with the pathname, deleting and renaming files, creating new directories, listing the contents of a directory, and determining several common attributes of files and directories.

- It is an abstract representation of files and directory pathnames.
- A pathname, whether abstract or in string form can be either absolute or relative. The parent of an abstract pathname may be obtained by invoking the `getParent()` method of this class.
- First of all, we should create the File class object by passing the filename or directory name to it. A file system may implement restrictions to certain operations on the actual file-system object, such as reading, writing, and executing. These restrictions are collectively known as access permissions.
- Instances of the File class are immutable; that is, once created, the abstract pathname represented by a File object will never change.

How to create a File Object?

A File object is created by passing in a string that represents the name of a file, a String, or another File object. For example,

```
File a = new File("/usr/local/bin/geeks");
```

This defines an abstract file name for the geeks file in the directory `/usr/local/bin`. This is an absolute abstract file name.

Constructors of File Class

- **File(File parent, String child):** Creates a new File instance from a parent abstract pathname and a child pathname string.
- **File(String pathname):** Creates a new File instance by converting the given pathname string into an abstract pathname.
- **File(String parent, String child):** Creates a new File instance from a parent pathname string and a child pathname string.
- **File(URI uri):** Creates a new File instance by converting the given file: URI into an abstract pathname.

Methods of File Class

S. No.	Method	Description	Return Type
1.	<u>canExecute()</u>	Tests whether the application can execute the file denoted by this abstract pathname.	boolean
2.	<u>canRead()</u>	Tests whether the application can read the file denoted by this abstract pathname.	boolean
3.	<u>canWrite()</u>	Tests whether the application can modify the file denoted by this abstract pathname.	boolean
4.	compareTo(File pathname)	Compares two abstract pathnames lexicographically.	int
5.	<u>createNewFile()</u>	Atomically creates a new, empty file named by this abstract pathname.	boolean
6.	<u>createTempFile(String prefix, String suffix)</u>	Creates an empty file in the default temporary-file directory.	File
7.	<u>delete()</u>	Deletes the file or directory denoted by this abstract pathname.	boolean
8.	equals(Object obj)	Tests this abstract pathname for equality with the given object.	boolean
9.	<u>exists()</u>	Tests whether the file or directory denoted by this abstract pathname exists.	boolean
10.	<u>getAbsolutePath()</u>	Returns the absolute pathname string of this abstract pathname.	String
11.	<u>list()</u>	Returns an array of strings naming the files and directories in the directory.	String[]
12.	<u>getFreeSpace()</u>	Returns the number of unallocated bytes in the partition.	long
13.	<u>getName()</u>	Returns the name of the file or directory denoted by this abstract pathname.	String

S. No.	Method	Description	Return Type
14.	<u>getParent()</u>	Returns the pathname string of this abstract pathname's parent.	String
15.	<u>getParentFile()</u>	Returns the abstract pathname of this abstract pathname's parent.	File
16.	<u>getPath()</u>	Converts this abstract pathname into a pathname string.	String
17.	<u>setReadOnly()</u>	Marks the file or directory named so that only read operations are allowed.	boolean
18.	<u>isDirectory()</u>	Tests whether the file denoted by this pathname is a directory.	boolean
19.	<u>isFile()</u>	Tests whether the file denoted by this abstract pathname is a normal file.	boolean
20.	<u>isHidden()</u>	Tests whether the file named by this abstract pathname is a hidden file.	boolean
21.	<u>length()</u>	Returns the length of the file denoted by this abstract pathname.	long
22.	<u>listFiles()</u>	Returns an array of abstract pathnames denoting the files in the directory.	File[]
23.	<u>mkdir()</u>	Creates the directory named by this abstract pathname.	boolean
24.	<u>renameTo(File dest)</u>	Renames the file denoted by this abstract pathname.	boolean
25.	<u>setExecutable(boolean executable)</u>	A convenience method to set the owner's execute permission.	boolean
26.	<u>setReadable(boolean readable)</u>	A convenience method to set the owner's read permission.	boolean
27.	<u>setReadable(boolean readable, boolean ownerOnly)</u>	Sets the owner's or everybody's read permission.	boolean

S. No.	Method	Description	Return Type
28.	<u>setWritable(boolean writable)</u>	A convenience method to set the owner's write permission.	boolean
29.	toString()	Returns the pathname string of this abstract pathname.	String
30.	toURI()	Constructs a file URI that represents this abstract pathname.	URI

Example 1: Program to check if a file or directory physically exists or not.

```
// In this Java program, we accepts a file or directory name from
// command line arguments. Then the program will check if
// that file or directory physically exist or not and
// it displays the property of that file or directory.
```

```
import java.io.File;

// Displaying file property
class fileProperty {
    public static void main(String[] args)
    {

        // accept file name or directory name through
        // command line args
        String fname = args[0];

        // pass the filename or directory name to File
        // object
        File f = new File(fname);

        // apply File class methods on File object
        System.out.println("File name :" + f.getName());
        System.out.println("Path: " + f.getPath());
        System.out.println("Absolute path:"
            + f.getAbsolutePath());
        System.out.println("Parent:" + f.getParent());
        System.out.println("Exists :" + f.exists());

        if (f.exists()) {
            System.out.println("Is writable:"
                + f.canWrite());
        }
    }
}
```

```
        System.out.println("Is readable" + f.canRead());
        System.out.println("Is a directory:"
                           + f.isDirectory());
        System.out.println("File Size in bytes "
                           + f.length());
    }
}
```

Output:

```
File name :file.txt
Path: file.txt
Absolute path:C:\Users\akki\IdeaProjects\codewriting\src\file.txt
Parent:null
Exists :true
Is writable:true
Is readabletrue
Is a directory:false
File Size in bytes 20
```

Example 2: Program to display all the contents of a directory

Here we will accept a directory name from the keyboard and then display all the contents of the directory. For this purpose, `list()` method can be used as:

```
String arr[]=f.list();
```

In the preceding statement, the `list()` method causes all the directory entries copied into the array `arr[]`. Then pass these array elements `arr[i]` to the File object and test them to know if they represent a file or directory.

```
// Java Program to display all
// the contents of a directory
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
// Displaying the contents of a directory
class Contents {
    public static void main(String[] args)
        throws IOException
    {
        // enter the path and dirname from keyboard
        BufferedReader br = new BufferedReader(
            new InputStreamReader(System.in));

        System.out.println("Enter dirpath:");
        String dirpath = br.readLine();
        System.out.println("Enter the dirname");
        String dname = br.readLine();

        // create File object with dirpath and dname
        File f = new File(dirpath, dname);

        // if directory exists, then
        if (f.exists()) {
            // get the contents into arr[]
            // now arr[i] represent either a File or
            // Directory
            String arr[] = f.list();

            // find no. of entries in the directory
            int n = arr.length;

            // displaying the entries
            for (int i = 0; i < n; i++) {
                System.out.println(arr[i]);
                // create File object with the entry and
                // test if it is a file or directory
                File f1 = new File(arr[i]);
                if (f1.isFile())
                    System.out.println(": is a file");
                if (f1.isDirectory())
                    System.out.println(": is a directory");
            }
            System.out.println(
                "No of entries in this directory " + n);
        }
        else
            System.out.println("Directory not found");
    }
}
```

Output:

```
Enter dirpath:
C:\Users\akki\IdeaProjects\
Enter the dirname
codewriting
.idea
: is a directory
an1.txt
: is a file
codewriting.iml
: is a file
file.txt
: is a file
out
: is a directory
src
: is a directory
text
: is a file
No of entries in this directory 7
```

Related Post: [FileReader and FileWriter in Java](#)

This article is contributed by **Nishant Sharma**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.