**Directory and Filesystem Operations**

* This chapter is largely devoted to java.io.File class.
* The File class gives you the ability to list directories, obtain file status, rename and delete files on disk, create directories, and perform other filesystem operations.
* Note that many of the methods of this class attempt to modify the permanent file store, or disk file system, of the computer you run them on.

**Getting File Information**

**Problem**

* You need to know all you can about a given file on disk.

**Solution**

* Use a java.io.File object.

|  |  |  |
| --- | --- | --- |
| ***java.io.File methods*** | | |
| **Modifier and Type** | **Method** | **Description** |
| boolean | createNewFile() | It atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist. |
| boolean | canWrite() | It tests whether the application can modify the file denoted by this abstract pathname.String[] |
| boolean | canExecute() | It tests whether the application can execute the file denoted by this abstract pathname. |
| boolean | canRead() | It tests whether the application can read the file denoted by this abstract pathname. |
| boolean | isAbsolute() | It tests whether this abstract pathname is absolute. |
| boolean | isDirectory() | It tests whether the file denoted by this abstract pathname is a directory. |
| boolean | isFile() | It tests whether the file denoted by this abstract pathname is a normal file. |
| String | getName() | It returns the name of the file or directory denoted by this abstract pathname. |
| String | getParent() | It returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory. |
| boolean | mkdir() | It creates the directory named by this abstract pathname. |
| long | length() | This method returns the length of the file denoted by this abstract pathname. |
| long | lastModified() | This method returns the time that the file denoted by this abstract pathname was last modified. |
| String | getCanonicalPath() | This method returns the canonical pathname string of this abstract pathname. |
| boolean | exists() | True if something of that name exists |
| [String](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html)[] | [list](https://docs.oracle.com/javase/7/docs/api/java/io/File.html#list())() | Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname. |
| [File](https://docs.oracle.com/javase/7/docs/api/java/io/File.html)[] | [listFiles](https://docs.oracle.com/javase/7/docs/api/java/io/File.html#listFiles())() | Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname. |

**Example:** Java program to displays the property of a file and directory.

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.Date;

**public** **class** FileStatus {

**public** **static** **void** main(String[] argv) **throws** IOException {

File f=**new** File("C:\\Users\\Sangram\\Documents\\JavaDemo

\\MyFile.txt");

// Print File name

String n = f.getName();

System.***out***.println("File Name: " + n);

// Print full name

System.***out***.println("Canonical Name: " + f.getCanonicalPath());

// Print parent directory if possible

String p = f.getParent();

**if** (p != **null**) {

System.***out***.println("Parent Directory: " + p);

}

// Check if the file is readable

**if** (f.canRead()) {

System.***out***.println("File is Readable.");

}

// Check if the file is writable

**if** (f.canWrite()) {

System.***out***.println("File is Writable.");

}

// Report on the modification time.

Date d = **new** Date(f.lastModified());

System.***out***.println("Last Modified " + d);

// See if file, directory, or other. If file, print size.

**if** (f.isFile()) {

// Report on the file's size

System.***out***.println("File size is " + f.length() + " bytes.");

} **else** **if** (f.isDirectory()) {

System.***out***.println("It's a Directory");

} **else** {

System.***out***.println("It's Neither a File nor a Directory!");

}

}

}

**Output:**

File Name: MyFile.txt

Canonical Name: C:\Users\Sangram\Documents\JavaDemo\MyFile.txt

Parent Directory: C:\Users\Sangram\Documents\JavaDemo

File is Readable.

File is Writable.

Last Modified Thu May 21 22:47:27 IST 2020

File size is 69 bytes.

Creating a File

**Problem**

* You need to create a new file on disk, but you don’t want to write into it.

**Solution**

* Use a java.io.File object’s createNewFile() method.

**Example:** Java program to create a new file.

**import** java.io.File;

**import** java.io.IOException;

**public** **class** CreateFile {

**public** **static** **void** main(String[] args) {

**try** {

//creating a Java File instance

File file = **new** File("H:\\DemoJava\\javaFile123.txt");

//createNewFile(): Atomically creates

//a new, empty file. If file is already

//exist it show file exist message

**if** (file.createNewFile()) {

System.***out***.println("New File is created!");

} **else** {

System.***out***.println("File already exists.");

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**Output:**

New File is created!

Creating a Directory

* You can use the Java File class to create directories if they don't already exists.
* The File class contains the method mkdir().
* The mkdir() returns true if the directory was created, and false if not.

**Example:** Java program to create a new Directory

**import** java.io.File;

**public** **class** CreateDirectory {

**public** **static** **void** main(String[] args) {

//Creating a File object

File file = **new** File("H:\\JAVA\_DIR");

//Creating the directory

**boolean** bool = file.mkdirs();

**if**(bool){

System.***out***.println("Directory created successfully");

}

**else**{

System.***out***.println("Sorry couldnt create specified directory");

}

}

}

**Output:**

Directory created successfully

**Renaming a File or Directory**

**Problem**

* You need to change a file’s name on disk.

**Solution**

* Use a java.io.File object’s renameTo() method.

**Example:** Java program to rename a file

**import** java.io.File;

**public** **class** Rename {

**public** **static** **void** main(String[] args) {

//File objects for the existing name

File oldName = **new** File("H:\\DemoJava\\javaFile123.txt");

//File objects for the new name

File newName = **new** File("H:\\DemoJava\\JAVA.txt");

//renameTo(): Renames the file denoted

//by this abstract pathname. If file is

//already exist it show Error message

**if** (oldName.renameTo(newName))

System.***out***.println("Renamed successfully");

**else**

System.***out***.println("Error");

}

}

**Output:**

Renamed successfully

**Example:** Java program to rename a Directory

**import** java.io.File;

**public** **class** Rename {

**public** **static** **void** main(String[] args) {

//File objects for the existing name

File oldName = **new** File("H:\\DemoJava");

//File objects for the new name

File newName = **new** File("H:\\Java");

//renameTo(): Renames the Directory denoted

//by this abstract pathname. If Directory is

//already exist it show Error message

**if** (oldName.renameTo(newName))

System.***out***.println("Renamed successfully");

**else**

System.***out***.println("Error");

}

}

**Output:**

Renamed successfully

**Deleting a File and Directory**

**Problem**

* You need to delete one or more files from the disk.

**Solution**

* Use a java.io.File object’s delete() method.
* It deletes files (subject to permissions) and directories (subject to permissions and to the directory being empty).

**Example:** Java program to delete a file

**import** java.io.File;

**public** **class** DeleteFile {

**public** **static** **void** main(String[] args) {

//File objects for the new name

File file = **new** File("H:\\DemoJava\\JAVA.txt");

//file.delete(): Deletes the file or

//directory denoted by this abstract path name. If

//no such file display the failed to delete message

**if**(file.delete())

{

System.***out***.println("File deleted successfully");

}

**else**

{

System.***out***.println("Failed to delete the file");

}

}

}

**Output:**

File deleted successfully

**Example:** Java program to delete a Directory.

**import** java.io.File;

**public** **class** DeleteDirectory {

**public** **static** **void** main(String[] args) {

//File objects for the existing Directory name

File DirName = **new** File("H:\\DemoJava");

//Function for directory deletion

*deleteDir*(DirName);

System.***out***.println("The directory is deleted.");

}

**public** **static** **boolean** deleteDir(File dir){

//Returns an array of abstract pathnames

//denoting the files in the directory

//denoted by this abstract pathname.

File[] files = dir.listFiles();

**if**(files != **null**){

**for**(File fileName : files){

**if**(fileName.isDirectory()){

*deleteDir*(fileName);

} **else** {

fileName.delete();

System.***err***.println("\*\* Deleted " + fileName

+ " \*\*");

}

}

}

**return** dir.delete();

}

}

**Output:**

\*\* Deleted H:\DemoJava\javaFile1.txt \*\*

\*\* Deleted H:\DemoJava\javaFile2.txt \*\*

\*\* Deleted H:\DemoJava\javaFile3.txt \*\*

The directory is deleted.

**Creating a Transient File**

**Problem**

* You need to create a file with a unique temporary filename, or arrange for a file to be deleted when your program is finished.

**Solution**

* Use a java.io.File object’s createTempFile() or deleteOnExit() method.

**File createTempFile() method**

* The **createTempFile()** function creates a temporary file in a given directory (if the directory is not mentioned then a default directory is selected).
* The function generates the filename by using the prefix and suffix passed as the parameters.
* If the suffix is null then the function uses “.tmp” as suffix. The function then returns the created file.
* **Syntax**:
  + *public static File createTempFile(String prefix, String suffix)*
    - **prefix** − The prefix string defines the file name; must be at least three characters long.
    - **suffix** − The suffix string defines the file's extension; if null the suffix ".tmp" will be used.
  + *public static File createTempFile(String prefix, String suffix, File directory)*
    - **prefix** − The prefix string defines the files name; must be at least three characters long.
    - **suffix** − The suffix string defines the file's extension; if null the suffix ".tmp" will be used.
    - **directory** − The directory in which the file is to be created. If the directory is not specified or a null value is passed then the function uses an default directory.

**File.deleteOnExit() method**

* The ***File.deleteOnExit()*** method also deletes the file or directory defined by abstract pathname.
* The deleteOnExit() method deletes files in reverse order.
* It deletes the file when JVM terminates. It does not return any value.
* Once the request has been made, it is not possible to cancel the request. So this method should be used with care.
* Usually, we use this method when we want to delete the temporary file.
* A temporary file is used to store the less important and temporary data, which should always be deleted when JVM terminates.
* If we want to delete the .temp file manually, we can use File.delete() method.

**Example:** Java programs illustrate the use of createTempFile() function and deleteOnExit() method.

**import** java.io.File;

**public** **class** CreateTempFile {

**public** **static** **void** main(String[] args) {

File f = **null**;

**try** {

System.***out***.println("\*\*\* Create Temporary file using

First Syntax \*\*\*");

// creates temporary file

f = File.*createTempFile*("tmp", ".txt");

// prints absolute path

System.***out***.println("File path: "+f.getAbsolutePath());

// deletes file when the virtual machine terminate

f.deleteOnExit();

// creates temporary file

f = File.*createTempFile*("tmp", **null**);

// prints absolute path

System.***out***.println("File path: "+f.getAbsolutePath());

// deletes file when the virtual machine terminate

f.deleteOnExit();

System.***out***.println(" ");

System.***out***.println("\*\*\* Create Temporary file using

Second Syntax \*\*\*");

// creates temporary file

f = File.*createTempFile*("tmp", ".txt", **new** File("D:/"));

// prints absolute path

System.***out***.println("File path: "+f.getAbsolutePath());

// deletes file when the virtual machine terminate

f.deleteOnExit();

// creates temporary file

f = File.*createTempFile*("tmp", **null**, **new** File("D:/"));

// prints absolute path

System.***out***.print("File path: "+f.getAbsolutePath());

// deletes file when the virtual machine terminates

f.deleteOnExit();

} **catch**(Exception e) {

// if any error occurs

e.printStackTrace();

}

}

}

**Output:**

\*\*\* Create Temporary file using First Syntax \*\*\*

File path: C:\Users\Sangram\AppData\Local\Temp\tmp11964534484036360606.txt

File path: C:\Users\Sangram\AppData\Local\Temp\tmp2999134140143943325.tmp

\*\*\* Create Temporary file using Second Syntax \*\*\*

File path: D:\tmp6661626881408204835.txt

File path: D:\tmp11537867165325446829.tmp

**Changing File Attributes**

**Problem**

* You want to change attributes of a file other than its name.

**Solution**

* Use setReadOnly() or setLastModified().

# Java.io.File.setReadOnly() Method

* The **java.io.File.setReadOnly()** method switches the file to read only mode and denies any write operations on the file.
* The **setReadOnly()** method is a part of [File](https://www.geeksforgeeks.org/file-class-in-java/) class.
* The setReadOnly() function marks the specified file or directory such that only read operations are allowed on the file or directory.
* The function returns **boolean data type**. The function returns true if the File object could be set as Read only else false.

**Example:** Java programs illustrate the use of **setReadOnly()** method

**import** java.io.File;

**public** **class** SetReadOnly {

**public** **static** **void** main(String[] args) {

**try** {

System.***out***.println("\*\*\* Before File Setting \*\*\*");

//Creates a new File instance

File file = **new** File("C:\\Users\\Sangram\\Documents\\JavaDemo\\FileOperation.txt");

//Atomically creates a new, empty file

file.createNewFile();

//Get the file name

String f = file.getName();

**if** (file.canRead())

System.***out***.println(f + " is Readable");

**else**

System.***out***.println(f + " is Not Readable");

**if** (file.canWrite())

System.***out***.println(f + " is Writable");

**else**

System.***out***.println(f + " is Not Writable");

file.setReadOnly();

System.***out***.println("");

System.***out***.println("\*\*\* After File Setting \*\*\*");

**if** (file.canRead())

System.***out***.println(f + " is Readable");

**else**

System.***out***.println(f + " is Not Readable");

**if** (file.canWrite())

System.***out***.println(f + " is Writable");

**else**

System.***out***.println(f + " is Not Writable");

} **catch**(Exception e) {

e.printStackTrace();

}

}

}

**Output:**

\*\*\* Before File Setting \*\*\*

FileOperation.txt is Readable

FileOperation.txt is Not Writable

\*\*\* After File Setting \*\*\*

FileOperation.txt is Readable

FileOperation.txt is Not Writable

**Java.io.File. setLastModified()**

* The **setLastModified()** method is a part of [File](https://www.geeksforgeeks.org/File-class-in-java/) class.
* The function sets the last modified time of the file or directory. The function sets the last modified value of the file in milliseconds.
* Syntax:
  + public boolean setLastModified(long time)
* This function accepts a long value as parameter which represents the new last modified time.
* The function returns a boolean value which states whether the new last modified time is set or not.

**Example:** Java programs illustrates the use of **setLastModified()** method

**import** java.io.File;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**public** **class** SetLastModifiedTime {

**public** **static** **void** main(String args[]) {

File f = **null**;

**boolean** bool = **false**;

**int** year, month, day;

**long** millisec;

Date dt;

**try** {

// create new File object

f = **new** File("C:\\Users\\Sangram\\Documents

\\JavaDemo\\abc.txt");

//Get the file name

String FName = f.getName();

System.***out***.println("\*\*\* Last Modification Time Before

Setting \*\*\*");

// Report on the modification time.

Date d = **new** Date(f.lastModified());

System.***out***.println(FName + " is Last Modified on " + d);

System.***out***.println("\*\*\* Last Modification Time After

Setting \*\*\*");

// date components

year = 2020;

month = 04;

day = 10;

// date in string

String sDate1 = day+"/"+month+"/"+year;

Date date1=**new**

SimpleDateFormat("dd/MM/yyyy")

.parse(sDate1);

// calculate milliseconds

millisec = date1.getTime();

// Check if the last modified time

// can be set to new value

**if** (f.setLastModified(millisec)) {

// Display that the last modified time

// is set as the function returned true

System.***out***.println("Last modified time of "

+ FName + " is set");

// last modified time

millisec = f.lastModified();

// calculate date object

dt = **new** Date(millisec);

// Print on the modification time.

System.***out***.println(FName + " is (new) Last

Modified on " + dt);

}

**else** {

// Display that the last modified time

// cannot be set as the function returned false

System.***out***.println("Last modified time cannot be set");

}

} **catch**(Exception e) {

// if any error occurs

e.printStackTrace();

}

}

}

**Output:**

\*\*\* Last Modification Time Before Setting \*\*\*

abc.txt is Last Modified on Wed May 27 00:28:04 IST 2020

\*\*\* Last Modification Time After Setting \*\*\*

Last modified time of abc.txt is set

abc.txt is (new) Last Modified on Fri Apr 10 00:00:00 IST 2020

**Listing a Directory**

**Problem**

* You need to list the filesystem entries named in a directory.

**Solution**

* Use a java.io.File object’s list() or listFiles() method.

# list() method

* The **list()** method is a part of [File](https://www.geeksforgeeks.org/File-class-in-java/) class.
* The java.io.File.list() returns the array of files and directories in the directory defined by this abstract path name.
* The method returns null, if the abstract pathname does not denote a directory.
* Syntax:
  + *public String[] list()*
    - This function does not have any parameter
  + *public String[] list(FilenameFilter f)*
    - This function takes FilenameFilter object as parameter
  + The function returns a string array, or null value if the file object is file.
  + To list the filesystem entities named in the current directory, just write:
    - String[] list = new File(".").list( )

**Example:** Java program to demonstrate the use of list() function to find all the files and directories in a given directory and current directory.

**import** java.io.File;

**public** **class** ListingDirectory {

**public** **static** **void** main(String[] args) {

// try-catch block to handle exceptions

**try** {

// Create a file object

File f = **new** File("C:\\Users\\Sangram\\Documents");

// Get all the names of the files/Directory

// present in the given directory

String[] DirList = f.list();

System.***out***.println("Files/Directories are in the given

directory:");

// Display the names of the files

**for** (String dir : DirList) {

System.***out***.println(dir);

}

System.***out***.println();

// Get list of names from current directory

String[] Dirs = **new** java.io.File(".").list();

System.***out***.println("Files/Directories are in the current

directory:");

// Display the names of the files

**for** (String dirname : Dirs) {

System.***out***.println(dirname);

}

}

**catch** (Exception e) {

System.***err***.println(e.getMessage());

}

}

}

**Output:**

Files/Directories are in the given directory:

desktop.ini

JavaDemo

My Music

My Pictures

My Videos

Files/Directories are in the current directory:

.classpath

.project

.settings

bin

src

**Example:** Java program to demonstrate the use of *list(FilenameFilter f)* function to find all the files and directories in a given directory whose names start with “My ”.

**import** java.io.File;

**import** java.io.FilenameFilter;

**public** **class** ListingDirectory {

**public** **static** **void** main(String[] args) {

// try-catch block to handle exceptions

**try** {

// Create a file object

File f = **new** File("C:\\Users\\Sangram\\Documents");

// Create a FilenameFilter

FilenameFilter filter = **new** FilenameFilter() {

**public** **boolean** accept(File f, String name) {

**return** name.startsWith("My");

}

};

// Get all the names of the directory

// present in the given directory

// and whose names start with "My"

String[] DirList = f.list(filter);

System.***out***.println("Files/Directories are in the given

directory:");

// Display the names of the files

**for** (String dir : DirList) {

System.***out***.println(dir);

}

}

**catch** (Exception e) {

System.***err***.println(e.getMessage());

}

}

}

**Output:**

Files/Directories are in the given directory:

My Music

My Pictures

My Videos

# listFiles() method

* The **listFiles()** method is a part of [File](https://www.geeksforgeeks.org/File-class-in-java/) class.
* The function returns an array of Files denoting the files in a given abstract pathname if the path name is a directory else returns null.
* Syntax:
  + public File[] listFiles()
    - This function does not have any parameter.
  + public File[] listFiles(FilenameFilter f)
    - This function takes FilenameFilter object as parameter.
  + public File[] listFiles(FileFilter f)
    - This function takes FileFilter object as parameter.
  + The function returns a File array, or null value if the file object is a file.
  + To get an array of already constructed File objects rather than Strings, use:
    - File[] list = new File(".").listFiles( );

**Example:** Java program to demonstrate the use of listFiles() method to find all the files and directories in a given directory and current directory.

**import** java.io.File;

**public** **class** ListingDirectory {

**public** **static** **void** main(String[] args) {

// try-catch block to handle exceptions

**try** {

// Create a file object

File f = **new** File("C:\\Users\\Sangram\\Documents");

// Get all the names of the files/Directory

// present in the given directory

File[] DirList = f.listFiles();

System.***out***.println("Files/Directories are in the given

directory:");

// Display the names of the files/Directories

**for** (**int** i = 0; i < DirList.length; i++) {

System.***out***.println(DirList[i].getName());

}

System.***out***.println();

// Get list of names from current directory

File[] Dirs = **new** File(".").listFiles( );

System.***out***.println("Files/Directories are in the given

directory:");

// Display the names of the files/Directories

**for** (**int** j = 0; j < Dirs.length; j++) {

System.***out***.println(Dirs[j].getName());

}

}

**catch** (Exception e) {

System.***err***.println(e.getMessage());

}

}

}

**Output:**

Files/Directories are in the given directory:

desktop.ini

JavaDemo

My Music

My Pictures

My Videos

Files/Directories are in the given directory:

.classpath

.project

.settings

bin

src

**Example:** Java program to demonstrate the use of listFiles*(FilenameFilter f)* function to find all the files and directories in a given directory whose names start with “out”.

**import** java.io.File;

**import** java.io.FilenameFilter;

**public** **class** ListingDirectory {

**public** **static** **void** main(String[] args) {

// try-catch block to handle exceptions

**try** {

// Create a file object

File f = **new** File("C:\\Users\\Sangram\\Documents

\\JavaDemo");

// Get all the names of the files

// present in the given directory

// Create a FilenameFilter

FilenameFilter filter = **new** FilenameFilter() {

**public** **boolean** accept(File f, String name) {

**return** name.startsWith("out");

}

};

// Get all the names of the directory

// present in the given directory

// and whose names start with "out"

File[] DirList = f.listFiles(filter);

System.***out***.println("Files/Directories are in the given

directory:");

// Display the names of the files/Directories

**for** (**int** i = 0; i < DirList.length; i++) {

System.***out***.println(DirList[i].getName());

}

}

**catch** (Exception e) {

System.***err***.println(e.getMessage());

}

}

}

**Output:**

Files/Directories are in the given directory:

output.txt

output1.txt

output2.txt

**Example:** Java program to demonstrate the use of listFiles(FileFilter f) method to find all the files and directories in a given directory which are text files.

**import** java.io.File;

**import** java.io.FileFilter;

**public** **class** ListingDirectory {

**public** **static** **void** main(String[] args) {

// try-catch block to handle exceptions

**try** {

// Create a file object

File f = **new** File("C:\\Users\\Sangram\\Documents

\\JavaDemo");

// Get all the names of the files

// present in the given directory

// Create a FileFilter

FileFilter filter = **new** FileFilter() {

**public** **boolean** accept(File f) {

**return** f.getName().endsWith("txt");

}

};

// Get all the names of the files present

// in the given directory

// which are text files

File[] files = f.listFiles(filter);

System.***out***.println("Files are:");

// Display the names of the files

**for** (**int** i = 0; i < files.length; i++) {

System.***out***.println(files[i].getName());

}

}

**catch** (Exception e) {

System.***err***.println(e.getMessage());

}

}

}

**Output:**

Files are:

abc.txt

BinaryInput.txt

BinaryOutput.txt

demofile.txt

MyFile.txt

MyInputFile.txt

MyOutputFile.txt

MyOutputFile1.txt

output.txt

output1.txt

output2.txt

**Getting the Directory Roots**

**Problem**

You want to know about the top-level directories, such as C:\ and D:\ on Windows.

**Solution**

Use the static method File.listRoots().

# listRoots() method

* The **listRoots()** method is a part of [File](https://www.geeksforgeeks.org/file-class-in-java/) class.

* The listRoots() function returns the root directories of all the available file System roots.
* It is guaranteed that the pathname of any file on the system will begin with any one of these roots.
* The function returns **File array**, which contains all the file system roots.

**Example:** Java program to demonstrate the use of File.listRoots() method.

**import** java.io.File;

**public** **class** ListRoot {

**public** **static** **void** main(String[] args) {

// Get list of roots names

File root[] = File.*listRoots*();

// check if the root is null or not

**if** (root != **null**) {

System.***out***.println("The List of Roots are: ");

// Print the list

**for** (File dr : drives) {

System.***out***.println(dr);

}

}

**else** {

System.***out***.println("There are no roots");

}

}

}

**Output:**

The List of Roots are:

C:\

D:\

**Using Path instead of File**

**Problem**

* You need more capability than the standard File class. You need to move, copy, delete, and otherwise work on files with a minimum of coding.

**Solution**

* Consider using the Path class, an intended replacement for File, and the Files class.

**java.nio.file.Path**

* Path is the particular location of an entity such as file or a directory in a file system so that one can search and access it at that particular location.
* Path is an interface which is introduced in Java NIO file package during Java version 7, and is the representation of location in particular file system.
* As path interface is in Java NIO package so it get its qualified name as java.nio.file.Path.
* In general path of an entity could be of two types:
  + absolute path
    - It is the location address from the root to the entity where it locates
  + relative path
    - It is the location address which is relative to some other path.

**Example:**

**import** java.io.File;

**import** java.nio.file.Path;

**import** java.nio.file.Paths;

**public** **class** PathDemo {

**public** **static** **void** main(String[] args) {

//Converts a path string, or a sequence of strings

//that when joined form a path string, to a Path.

Path path = Paths.*get*("C:\\Users\\Sangram\\Documents

\\JavaDemo\\abc.txt");

System.***out***.println("Relative path: " + path);

//Returns a Path object representing

//the absolute path of this path.

Path absolute = path.toAbsolutePath();

System.***out***.println("Absolute path: " + absolute);

//Returns a name element of this path as a Path object.

System.***out***.println("Name: " + path.getFileName());

//Returns the root component

System.***out***.println("Root: " + path.getRoot());

//Returns the parent path

System.***out***.println("Parent: " + path.getParent());

//Returns the number of name elements in the path.

System.***out***.println("Name Count: " + path.getNameCount());

//Returns a name element of this path as a Path object.

System.***out***.println("First Directory: " + path.getName(0));

//Returns a relative Path that is a subsequence

//of the name elements of this path.

System.***out***.println("Sub Path: " + path.subpath(0, 2));

//Returns the string representation of this path.

System.***out***.println(path.toString());

// call toFile() to get

// File object from path

File file = path.toFile();

// print file details

System.***out***.println("File Name:" + file.getName());

**if** (!file.exists()) {

System.***out***.println("This is a path.");

} **else** {

System.***out***.println("This is a file.");

}

//Get the path of file.

System.***out***.println("Path: " + file.toPath());

}

}

**Output:**

Relative path: C:\Users\Sangram\Documents\JavaDemo\abc.txt

Absolute path: C:\Users\Sangram\Documents\JavaDemo\abc.txt

Name: abc.txt

Root: C:\

Parent: C:\Users\Sangram\Documents\JavaDemo

Name Count: 5

First Directory: Users

Sub Path: Users\Sangram

C:\Users\Sangram\Documents\JavaDemo\abc.txt

File Name:abc.txt

This is a file.

Path: C:\Users\Sangram\Documents\JavaDemo\abc.txt