JAVA - FILE CLASS

Java File class represents the files and directory pathnames in an abstract manner. This class is used for creation of files and directories, file searching, file deletion etc.

The File object represents the actual file/directory on the disk. There are following constructors to create a File object:

Following syntax creates a new File instance from a parent abstract pathname and a child pathname string.

```
File(File parent, String child);
```

Following syntax creates a new File instance by converting the given pathname string into an abstract pathname.

```
File(String pathname)
```

Following syntax creates a new File instance from a parent pathname string and a child pathname string.

```
File(String parent, String child)
```

Following syntax creates a new File instance by converting the given file: URI into an abstract pathname.

```
File(URI uri)
```

Once you have *File* object in hand then there is a list of helper methods which can be used manipulate the files.

SN	Methods with Description
1	**public String getName()** Returns the name of the file or directory denoted by this abstract pathname.
2	**public String getParent()** Returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
3	**public File getParentFile()** Returns the abstract pathname of this abstract pathname's parent, or null if this pathname does not name a parent directory.
4	**public String getPath()** Converts this abstract pathname into a pathname string.
5	**public boolean isAbsolute()** Tests whether this abstract pathname is absolute. Returns true if this abstract pathname is absolute, false otherwise

6 **public String getAbsolutePath()**

Returns the absolute pathname string of this abstract pathname.

7 **public boolean canRead()**

Tests whether the application can read the file denoted by this abstract pathname. Returns true if and only if the file specified by this abstract pathname exists and can be read by the application; false otherwise.

8 **public boolean canWrite()**

Tests whether the application can modify to the file denoted by this abstract pathname. Returns true if and only if the file system actually contains a file denoted by this abstract pathname and the application is allowed to write to the file; false otherwise.

9 **public boolean exists()**

Tests whether the file or directory denoted by this abstract pathname exists. Returns true if and only if the file or directory denoted by this abstract pathname exists; false otherwise

10 **public boolean isDirectory()**

Tests whether the file denoted by this abstract pathname is a directory. Returns true if and only if the file denoted by this abstract pathname exists and is a directory; false otherwise.

11 **public boolean isFile()**

Tests whether the file denoted by this abstract pathname is a normal file. A file is normal if it is not a directory and, in addition, satisfies other system-dependent criteria. Any non-directory file created by a Java application is guaranteed to be a normal file. Returns true if and only if the file denoted by this abstract pathname exists and is a normal file; false otherwise

.

12 **public long lastModified()**

Returns the time that the file denoted by this abstract pathname was last modified. Returns a long value representing the time the file was last modified, measured in milliseconds since the epoch (00:00:00 GMT, January 1, 1970), or 0L if the file does not exist or if an I/O error occurs.

13 **public long length()**

Returns the length of the file denoted by this abstract pathname. The return value is unspecified if this pathname denotes a directory.

14 **public boolean createNewFile() throws IOException**

Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist. Returns true if the named file does not exist and was successfully created; false if the named file already exists.

15 **public boolean delete()**

Deletes the file or directory denoted by this abstract pathname. If this pathname denotes a directory, then the directory must be empty in order to be deleted. Returns true if and only if the file or directory is successfully deleted; false otherwise.

16 **public void deleteOnExit()**

Requests that the file or directory denoted by this abstract pathname be deleted when the virtual machine terminates.

17 **public String[] list()**

Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname.

18 **public String[] list(FilenameFilter filter)**

Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.

20 **public File[] listFiles()**

Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.

21 **public File[] listFiles(FileFilter filter)**

Returns an array of abstract pathnames denoting the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.

22 **public boolean mkdir()**

Creates the directory named by this abstract pathname. Returns true if and only if the directory was created; false otherwise.

23 **public boolean mkdirs()**

Creates the directory named by this abstract pathname, including any necessary but nonexistent parent directories. Returns true if and only if the directory was created, along with all necessary parent directories; false otherwise.

24 **public boolean renameTo(File dest)**

Renames the file denoted by this abstract pathname. Returns true if and only if the renaming succeeded; false otherwise.

25 **public boolean setLastModified(long time)**

Sets the last-modified time of the file or directory named by this abstract pathname. Returns true if and only if the operation succeeded; false otherwise.

26 **public boolean setReadOnly()**

Marks the file or directory named by this abstract pathname so that only read operations are allowed. Returns true if and only if the operation succeeded; false otherwise.

27 **public static File createTempFile(String prefix, String suffix, File directory) throws IOException**

Creates a new empty file in the specified directory, using the given prefix and suffix strings to generate its name. Returns an abstract pathname denoting a newly-created empty file.

28 **public static File createTempFile(String prefix, String suffix) throws**

IOException

Creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name. Invoking this method is equivalent to invoking createTempFile(prefix, suffix, null). Returns abstract pathname denoting a newly-created empty file.

29 **public int compareTo(File pathname)**

Compares two abstract pathnames lexicographically. Returns zero if the argument is equal to this abstract pathname, a value less than zero if this abstract pathname is lexicographically less than the argument, or a value greater than zero if this abstract pathname is lexicographically greater than the argument.

30 **public int compareTo(Object o)**

Compares this abstract pathname to another object. Returns zero if the argument is equal to this abstract pathname, a value less than zero if this abstract pathname is lexicographically less than the argument, or a value greater than zero if this abstract pathname is lexicographically greater than the argument.

31 **public boolean equals(Object obj)**

Tests this abstract pathname for equality with the given object. Returns true if and only if the argument is not null and is an abstract pathname that denotes the same file or directory as this abstract pathname.

32 **public String toString()**

Returns the pathname string of this abstract pathname. This is just the string returned by the getPath() method.

Example:

Following is the example to demonstrate File object:

```
package com.tutorialspoint;

import java.io.File;

public class FileDemo {
   public static void main(String[] args) {

      File f = null;
      String[] strs = {"test1.txt", "test2.txt"};
      try{
         // for each string in string array
         for(String s:strs )
         {
            // create new file
            f= new File(s);

            // true if the file is executable
            boolean bool = f.canExecute();

            // find the absolute path
            String a = f.getAbsolutePath();

            // prints absolute path
            System.out.print(a);

            // prints
            System.out.println(" is executable: "+ bool);
         }
```

```
      }catch(Exception e){
         // if any I/O error occurs
         e.printStackTrace();
      }
   }
}
```

Consider there is an executable file test1.txt and another file test2.txt is non executable in current directory, Let us compile and run the above program, this will produce the following result:

```
test1.txt is executable: true
test2.txt is executable: false
```