Java File class

In Java, File class is used for the representation of files or directory pathname. Because of the names of files and directory have different formats on a different platform. The path of a file can be absolute or relative. There are many methods of file classes which can be used for creating, reading, deleting or renaming or re edit a file.

This class extends Object class and implements Serializable interface. Declaration of the class is following.

Declaration

public class File extends Object implements Serializable, Comparable<File>

The File class is located into **java.io** package so we need to import the package to use its functionality.

Following are the fields of File Class.

|  |  |  |
| --- | --- | --- |
| **S.no** | **Field** | **Description** |
| 1 | pathSeparator | It is represented as a string and it is used to separate the  characters of the path. |
| 2 | pathSeparatorChar | It is used as a path separator character. |
| 3 | separator | It is represented as a string and it is used to separate the characters  of the path. It is a default name separator. |
| 4 | separatorChar  FileInputStream  FileOutputStream | It is represented as a string and it is used to separate the characters of the path.It is a default name separator. |

Following are the Constructors of File Class.

1. File(File parent, String child)

2. File(String pathname)

3. File(String parent, String child)

4. File(URI uri)

Following are the methods of File Class.

|  |  |  |
| --- | --- | --- |
| **S.no.** | **Method** | **Description** |
| 1 | createTempFile(String prefix, String suffix) | It is used to create an empty file. |
| 2 | createNewFile() | It is used for creating a new file, which is  empty and has an abstract pathname. |
| 3 | canWrite() | It is used to check whether the application can  modify a file which has an abstract path. |
| 4 | canExecute() | It is used to check whether the application can  execute a file which has an abstract path. |
| 5 | canRead() | It is used to check whether the application can read a file which has an abstract path |
| 6 | isAbsolute() | It is used to check whether the abstract pathname is absolute or not. |
| 7 | isDirectory() | It is used to check whether the file with abstract pathname is a directory. |
| 8 | isFile() | It is used to check whether the file with abstract  pathname is a file. |
| 9 | getName() | It is used to get the name of the file. |
| 10 | getParent() | It is used to get the name of the Parent file |
| 11 | toPath() | It is used to get objects of java.nio.file.Path. |
| 12 | toURI() | It is used to create a URL of a file with abstract pathname. |
| 13 | listFiles() | It is used for getting an array of the abstract  pathname. |
| 14 | getFressSpace() | It is used for getting the number of unallocated  bytes. |
| 15 | list(FilenameFilter filter) | It is used for getting an array of string with the  name of the file which have abstract pathname. |
| 16 | mkdir() | It is used to create directory name. |

Example : Creating a File

Lets start working with file by creating a new file. To create a file, we are using createNewFile method that takes filename as an argument. This method creates a new file if file is not already exist. See the example below.

import java.io.File;

import java.io.IOException;

public class FileCreateDemo1

{

public static void main(String[] args)

{

try

{

File Obj = new File("FileDemo.txt");

if (Obj.createNewFile()) {

System.out.println("\*\*\*\*\*\*File created\*\*\*\*\*\*");

System.out.println("Name of the file = " + Obj.getName());

}

else{

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

}

}

catch (IOException e){

e.printStackTrace();

}

}

}

Example: Writing in a file

After creating a file, now will write data to the file. To write data, a method of Filewrite class is used. It will write the data but may throw exception so make sure to handle the excecptions as well.

import java.io.FileWriter;

import java.io.IOException;

public class FileWriteDemo1

{

public static void main(String[] args)

{

try

{

FileWriter obj = new FileWriter("FileDemo.txt");

obj.write("Welcome to abc.com");

obj.close();

System.out.println("File is Updated.");

}

catch (IOException e)

{

e.printStackTrace();

}

}

}

Example : Reading a file

To read data from the file we used File and Scenner classes both are used to handle input and output of system resources. Here we are using two method hasNextLine() and nextLine() to read the data sequentially.

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class FileReadDemo1

{

public static void main(String[] args)

{

try

{

File Obj = new File("FileDemo.txt");

Scanner obj1 = new Scanner(Obj);

while (obj1.hasNextLine())

{

String obj2 = obj1.nextLine();

System.out.println(obj2);

}

obj1.close();

}catch (FileNotFoundException e)

{

e.printStackTrace();

}

}

}

Example : Copying a file

Lets take another example to copy data of one file to another. Here we are using fileinput and output streams to read and write data. Although this is just a procedure to copy one file data to another whereas Java provides built-in methods to direct copy one file data to another file. See the below example.

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class FileCopyDemo1 {

public static void main(String[] args) {

FileInputStream a = null;

FileOutputStream b = null;

try {

File obj\_in = new File("FileDemo.txt");

File obj\_out = new File("FileDemo1.txt");

a = new FileInputStream(obj\_in);

b = new FileOutputStream(obj\_out);

byte[] buffer = new byte[1024];

int length;

while ((length = a.read(buffer)) > 0) {

b.write(buffer, 0, length);

}

a.close();

b.close();

System.out.println("File copied successfully!!");

} catch (IOException e) {

e.printStackTrace();

}

}

}

File permissions

We can check permissions like: reading, writing, deleting etc of a file by using the built-in methods. The File class provides methods canRead(), canWrite() etc to check whether the operation is permissible or not.

import java.io.\*;

public class FilePermissionDemo1 {

public static void main(String[] args) {

File a = new File("FileDemo1.txt");

boolean b = a.exists();

if (b == true) {

System.out.println("Executable: " + a.canExecute());

System.out.println("Readable: " + a.canRead());

System.out.println("Writable: " + a.canWrite());

} else {

System.out.println("File not found.");

}

}

}

Retrieving file information

This is one of the important part of file handling, getting metadata of a file is necessary to keep the information about the file like: type of file, location of file, permissions of file etc. in this example, we are using some built-in methods of File class to get information about the file. See the below example

import java.io.File;

public class FileInfoDemo1

{

public static void main(String[] args)

{

File Obj = new File("FileDemo1.txt");

if (Obj.exists())

{

System.out.println("File name= " + Obj.getName());

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

System.out.println("Absolute path= " + Obj.getAbsolutePath());

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

System.out.println("Writeable= " + Obj.canWrite());

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

System.out.println("Readable= " + Obj.canRead());

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

System.out.println("File size in bytes= " + Obj.length());

}

else

{

System.out.println("file does not exist.");

}

}

}

Deleting a file

In case, we need to delete a file then Java provides the method delete() that helps to delete a file using a code. In this example, we are deleting file. The method returns a boolean value to ensure that file has deleted successfully.

import java.io.File;

public class FileDeleteDemo1 {

public static void main(String[] args) {

File Obj = new File("FileDemo.txt");

if (Obj.delete()) {

System.out.println(Obj.getName() + " has been deleted");

} else {

System.out.println("Failed");

}

}

}