Apache Commons io

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

The Apache Commons io are the components of the Apache Commons software, which are derived from Java API and provides various utility classes for common operations for File io covering wide range of use cases. It helps to avoid writing boilerplate code. This tutorial covers most of the topics required for a basic understanding of Apache Commons io and to get a feel of how it works.

Audience

This tutorial has been prepared for the beginners to help them understand the basic to advanced concepts related to Apache Commons io.

Prerequisites

Before you start practicing various types of examples given in this reference, we assume that you are already aware about the computer programs and computer programming languages.

Copyright & Disclaimer

Table of Contents

1. Apache Commons IO — Overview

Apache Commons IO library provides various utility classes for common operations for File IO covering wide range of use cases. It helps to avoid writing the boilerplate code.

Classes

Apache Commons IO library provides classes for following categories:

Utility classes

These classes which are under **org.apache.commons.io** package, provides file and string comparison. Following are some of the examples.

- **IOUtils** – Provides utility methods for reading, writing and copying files. The methods work with InputStream, OutputStream, Reader and Writer.
- **FilenameUtils** – Provides method to work with file names without using File Object. It works on different operating systems in similar way.
- **FileUtils** – Provides method to manipulate files like moving, opening, checking existence, reading of file etc. These methods use File Object.
- **IOCase** – Provides method for string manipulation and comparison.
- **FileSystemUtils** – Provides method to get the free space on a disk drive.
- **LineIterator** – Provides a flexible way to work with a line-based file.

Filter classes

Filter classes which are under **org.apache.commons.io.filefilter** package, provides methods to filter files based on logical criteria instead of string based tedious comparisons. Following are some of the examples.

- **NameFileFilter** – Filters file-names for a name.
- **WildcardFileFilter** – Filters files using the supplied wildcards.
- **SuffixFileFilter** – Filters files based on suffix. This is used in retrieving all the files of a particular type.
- **PrefixFileFilter** – Filters files based on prefix.
- **OrFileFilter** – Provides conditional OR logic across a list of file filters. Returns true, if any filter in the list return true. Otherwise, it returns false.

- **AndFileFilter** – Provides conditional and logic across a list of file filters. Returns false, if any filter in the list return false. Otherwise, it returns true.

File Monitor classes

File monitor classes which are under **org.apache.commons.io.monitor** package provides control to track changes in a specific file or folder and allows to do action accordingly on the changes. Following are some of the examples.

- **FileEntry** – Provides the state of a file or directory. File attributes at a point in time.
- **FileAlterationObserver** – Represents the state of files below a root directory, checks the filesystem and notifies listeners of create, change or delete events.
- **FileAlterationMonitor** – Represents a thread that spawns a monitoring thread triggering any registered FileAlterationObserver at a specified interval.

Comparator classes

File monitor classes which are under **org.apache.commons.io.comparator** package allow to compare and sort files and directories easily.

- **NameFileComparator** – Compare the names of two files.
- **SizeFileComparator** – Compare the size of two files.
- **LastModifiedFileComparator** – Compare the last modified dates of two files.

Stream classes

There are multiple implementation of InputStream under **org.apache.commons.io.input** package and of OutputStream under **org.apache.commons.io.output** package to do useful tasks on streams. Following are some of the examples.

- **NullOutputStream** – Absorbs all data sent with any error.
- **TeeOutputStream** – Sends output to two streams.
- **ByteArrayOutputStream** – Faster version of Java Development Kit (JDK) class.
- **CountingOutputStream** – Counts the number of bytes passed through the stream.
- **ProxyOutputStream** – Changes the calls to proxy stream.
- **LockableFileWriter** – A FileWriter to create lock files and allow simple cross thread file lock handling.

2. Apache Commons IO — Environment Setup

In this chapter, we will learn about the local environment setup of Apache Commons IO and how to set up the path of Commons IO for Windows 2000/XP, Windows 95/98/ME etc. We will also understand about some popular java editors and how to download Commons IO archive.

Local Environment Setup

If you are still willing to set up your environment for Java programming language, then this section will guide you on how to download and set up Java on your machine. Please follow the steps mentioned below to set up the environment.

Java Standard Edition (SE) is freely available from the link **Download Java,** which is available at https://www.oracle.com/java/technologies/oracle-java-archive-downloads.html. So, you download a version based on your operating system.

Follow the instructions to download Java and run the **.exe** to install Java on your machine. Once you have installed Java on your machine, you would need to set environment variables to point to correct installation directories.

Path for Windows 2000/XP

We are assuming that you have installed Java in **c:\Program Files\java\jdk** directory.

- Right-click on 'My Computer' and select 'Properties'.
- Click on the 'Environment variables' button under the 'Advanced' tab.
- Now, alter the 'Path' variable so that it also contains the path to the Java executable. Example, if the path is currently set to 'C:\WINDOWS\SYSTEM32', then change your path to read 'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'.

Path for Windows 95/98/ME

We are assuming that you have installed Java in **c:\Program Files\java\jdk** directory.

- Edit the 'C:\autoexec.bat' file and add the following line at the end – 'SET PATH=%PATH%;C:\Program Files\java\jdk\bin'.

Path for Linux, UNIX, Solaris, FreeBSD

Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your shell documentation if you have trouble doing this.

Example, if you use bash as your shell, then you would add the following line to the end of your '.bashrc: export PATH=/path/to/java:$PATH'.

Popular Java Editors

To write your Java programs, you need a text editor. There are many sophisticated integrated development environment (IDEs) available in the market. But for now, you can consider one of the following:

- **Notepad** – On Windows machine you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.
- **Netbeans** – It is a Java IDE that is open-source and free which can be downloaded from https://www.netbeans.org/index.html.
- **Eclipse** – It is also a Java IDE developed by the eclipse open-source community and can be downloaded from https://www.eclipse.org/.

Download Common IO Archive

Download the latest version of Apache Common IO jar file from commons-io-2.8.0-bin.zip. At the time of writing this tutorial, we have downloaded **commons-io-2.8.0-bin.zip** and copied it into **C:\>Apache folder**.

OS	Archive name
Windows	commons-io-2.8.0-bin.zip
Linux	commons-io-2.8.0-bin.tar.gz
Mac	commons-io-2.8.0-bin.tar.gz

Apache Common IO Environment

Set the **APACHE_HOME** environment variable to point to the base directory location where, Apache jar is stored on your machine. Assuming, we have extracted **commons-io-2.8.0-bin.zip** in Apache folder on various Operating Systems as follows:

OS	Output
Windows	Set the environment variable APACHE_HOME to C:\Apache
Linux	export APACHE_HOME=/usr/local/Apache
Mac	export APACHE_HOME=/Library/Apache

Set CLASSPATH Variable

Set the **CLASSPATH** environment variable to point to the Common IO jar location. Assuming, you have stored **commons-io-2.8.0-bin.zip** in Apache folder on various Operating Systems as follows:

OS	Output
Windows	Set the environment variable CLASSPATH to %CLASSPATH%;%APACHE_HOME%\commons-io-2.8.0.jar;.;
Linux	export CLASSPATH=$CLASSPATH:$APACHE_HOME/commons-io-2.8.0.jar:.
Mac	export CLASSPATH=$CLASSPATH:$APACHE_HOME/commons-io-2.8.0.jar:.

3. Apache Commons IO — IOUtils

IOUtils provide utility methods for reading, writing and copying files. The methods work with InputStream, OutputStream, Reader and Writer.

Class Declaration

Following is the declaration for **org.apache.commons.io.IOUtils** Class:

```
public class IOUtils
   extends Object
```

Features of IOUtils

The features of IOUtils are given below:

- Provides static utility methods for input/output operations.
- toXXX() – reads data from a stream.
- write() – write data to a stream.
- copy() – copy all data to a stream to another stream.
- contentEquals – compare the contents of two streams.

Example of IOUtils Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

IOTester.java

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

import org.apache.commons.io.IOUtils;

public class IOTester {
   public static void main(String[] args) {
      try {
         //Using BufferedReader
         readUsingTraditionalWay();

         //Using IOUtils
         readUsingIOUtils();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   //reading a file using buffered reader line by line
   public static void readUsingTraditionalWay() throws IOException {
      try(BufferedReader bufferReader
         = new BufferedReader( new InputStreamReader(
            new FileInputStream("input.txt") ) )) {
         String line;
         while( ( line = bufferReader.readLine() ) != null ) {
            System.out.println( line );
         }
      }
   }

   //reading a file using IOUtils in one go
   public static void readUsingIOUtils() throws IOException {
      try(InputStream in = new FileInputStream("input.txt")) {
         System.out.println( IOUtils.toString( in , "UTF-8") );
      }
   }
}
```

Output

It will print the following result:

```
Welcome to TutorialsPoint. Simply Easy Learning.
Welcome to TutorialsPoint. Simply Easy Learning.
```

4. Apache Commons IO — FileUtils

FileUtils provide method to manipulates files like moving, opening, checking existence, reading of file etc. These methods use File Object.

Class Declaration

Following is the declaration for **org.apache.commons.io.FileUtils** Class:

```
public class FileUtils

   extends Object
```

Features of FileUtils

The features of FileUtils are stated below:

- Methods to write to a file.
- Methods to read from a file.
- Methods to make a directory including parent directories.
- Methods to copy files and directories.
- Methods to delete files and directories.
- Methods to convert to and from a URL.
- Methods to list files and directories by filter and extension.
- Methods to compare file content.
- Methods to file last changed date.
- Methods to calculate a checksum.

Example of FileUtils Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

IOTester.java

```
import java.io.File;
import java.io.IOException;
import java.nio.charset.Charset;


import org.apache.commons.io.FileUtils;
```

```
public class IOTester {
   public static void main(String[] args) {
      try {
         //Using FileUtils
         usingFileUtils();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingFileUtils() throws IOException {
      //get the file object
      File file = FileUtils.getFile("input.txt");

      //get the temp directory
      File tmpDir = FileUtils.getTempDirectory();

      System.out.println(tmpDir.getName());

      //copy file to temp directory
      FileUtils.copyFileToDirectory(file, tmpDir);

      //create a new file
      File newTempFile = FileUtils.getFile(tmpDir, file.getName());

      //get the content
      String data = FileUtils.readFileToString(newTempFile,
Charset.defaultCharset());


      //print the content
      System.out.println(data);
   }
}
```

Output

It will print the following result:

```
Temp
```

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

5. Apache Commons IO — FilenameUtils

FilenameUtils provide a method to work with file names without using File Object. It works on different operating systems in similar way. This class solve the problems when moving from a Windows based development machine to a Unix based production machine.

Class Declaration

Following is the declaration for **org.apache.commons.io.FilenameUtils** Class:

```
public class FilenameUtils
   extends Object
```

Features of FilenameUtils

This class defines six components within a filename. Consider an example location as **C:\dev\project\file.txt**. Then, the components are as follows:

- Prefix - C:\
- Relative Path - dev\project\
- Absolute path - C:\dev\project\
- Name - file.txt
- Base name - file
- Extension - txt

To identify a directory, add a separator to file name.

Example of FilenameUtils Class

An example of FilenameUtils Class is given below:

IOTester.java

```
import java.io.IOException;
import org.apache.commons.io.FilenameUtils;

public class IOTester {
   public static void main(String[] args) {
      try {
         //Using FilenameUtils
         usingFilenameUtils();

      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingFilenameUtils() throws IOException {
      String path = "C:\\dev\\project\\file.txt";
      System.out.println("Full Path: " +FilenameUtils.getFullPath(path));
      System.out.println("Relative Path: " +FilenameUtils.getPath(path));
      System.out.println("Prefix: " +FilenameUtils.getPrefix(path));
      System.out.println("Extension: " + FilenameUtils.getExtension(path));
      System.out.println("Base: " + FilenameUtils.getBaseName(path));
      System.out.println("Name: " + FilenameUtils.getName(path));

      String filename = "C:/commons/io/../lang/project.xml";
      System.out.println("Normalized Path: " +
FilenameUtils.normalize(filename));
   }
}
```

Output

It will print the following result:

```
Full Path: C:\dev\project\
Relative Path: dev\project\
Prefix: C:\
Extension: txt
Base: file
Name: file.txt
Normalized Path: C:\commons\lang\project.xml
```

6. Apache Commons IO — FileSystemUtils

FileSystemUtils provide a method to get the free space on a disk drive.

Class Declaration

Following is the declaration for **org.apache.commons.io.FileSystemUtils** Class:

```
public class FileSystemUtils
   extends Object
```

Example of FileSystemUtils Class

Given below is the example of FileSystemUtils Class:

IOTester.java

```
import java.io.IOException;


import org.apache.commons.io.FileSystemUtils;


public class IOTester {
   public static void main(String[] args) {
      try {
         System.out.println("Free Space " + FileSystemUtils.freeSpaceKb("C:/")
+ " Bytes");
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }
}
```

Output

It will print the following result:

```
Free Space 61355640 kb
```

7. Apache Commons IO — IOCase

Enumeration of IO case sensitivity. Different operating systems have different rules for case-sensitivity for file names. For example, Windows is case-insensitive for file naming while, Unix is case-sensitive.

IOCase captures that difference and provides an enumeration to control how the filename comparisons should be performed. It also provides methods to use the enumeration to perform comparisons.

Enum Declaration

Following is the declaration for **org.apache.commons.io.IOCase** Enum:

```
public enum IOCase
   extends Enum<IOCase> implements Serializable
```

Example of IOCase Enum

An example of IOCase Enum is given below:

IOTester.java

```
import java.io.IOException;
import org.apache.commons.io.IOCase;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingIOCase();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingIOCase() throws IOException {
      String text = "Welcome to TutorialsPoint. Simply Easy Learning.";
      String text1 = "WELCOME TO TUTORIALSPOINT. SIMPLY EASY LEARNING.";

      System.out.println("Ends with Learning (case sensitive): " +
      IOCase.SENSITIVE.checkEndsWith(text1, "Learning."));
      System.out.println("Ends with Learning (case insensitive): " +
      IOCase.INSENSITIVE.checkEndsWith(text1, "Learning."));

      System.out.println("Equality Check  (case sensitive): " +
      IOCase.SENSITIVE.checkEquals(text, text1));

      System.out.println("Equality Check  (case insensitive): " +
      IOCase.INSENSITIVE.checkEquals(text, text1));
   }
}
```

Output

It will print the following result:

```
Ends with Learning (case sensitive): false
Ends with Learning (case insensitive): true
Equality Check  (case sensitive): false
Equality Check  (case insensitive): true
```

8. Apache Commons IO — LineIterator

LineIterator provides a flexible way to work with a line-based file.

Class Declaration

Following is the declaration for **org.apache.commons.io.LineIterator** Class:

```
public class LineIterator
   extends Object implements Iterator<String>, Closeable
```

Example of LineIterator Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
Learn web technologies,
prepare exams,
code online,
all at one place.
```

IOTester.java

```
import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileUtils;
import org.apache.commons.io.LineIterator;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingLineIterator();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }


   public static void usingLineIterator() throws IOException {
      //get the file object
      File file = FileUtils.getFile("input.txt");

      try(LineIterator lineIterator = FileUtils.lineIterator(file)) {
         System.out.println("Contents of input.txt");
         while(lineIterator.hasNext()) {
            System.out.println(lineIterator.next());
         }
      }
   }
}
```

Output

It will print the following result:

```
Contents of input.txt
Welcome to TutorialsPoint. Simply Easy Learning.
Learn web technologies,
prepare exams,
code online,
all at one place.
```

9. Apache Commons IO — NameFileFilter

NameFileFilter filters the file-names for a name.

Class Declaration

Following is the declaration for **org.apache.commons.io.filefilter.NameFileFilter** Class:

```
public class NameFileFilter
   extends AbstractFileFilter implements Serializable
```

Example of NameFileFilter Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

Let's print all files and directories in the current directory and then filter a file whose name is Input.txt.

IOTester.java

```
import java.io.File;
import java.io.IOException;

import org.apache.commons.io.IOCase;
import org.apache.commons.io.filefilter.NameFileFilter;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingNameFileFilter();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingNameFileFilter() throws IOException {
      //get the current directory
      File currentDirectory = new File(".");

      //get names of all files and directory in current directory
      String[] files = currentDirectory.list();
      System.out.println("All files and Folders.\n");
      for( int i = 0; i < files.length; i++ ) {
         System.out.println(files[i]);
      }

      System.out.println("\nFile with name input.txt\n");
      String[] acceptedNames = {"input", "input.txt"};
      String[] filesNames = currentDirectory.list( new
NameFileFilter(acceptedNames, IOCase.INSENSITIVE) );

      for( int i = 0; i < filesNames.length; i++ ) {
         System.out.println(filesNames[i]);
      }
   }
}
```

Output

It will print the following result:

```
All files and Folders.


.classpath
.project
.settings
bin
input.txt
src


File with name input.txt


input.txt
```

10. Apache Commons IO — WildcardFileFilter

WildcardFileFilter filters the files by using the supplied wildcards.

Class Declaration

Following is the declaration for **org.apache.commons.io.filefilter.WildcardFileFilter** Class:

```
public class WildcardFileFilter
   extends AbstractFileFilter implements Serializable
```

Example of WildcardFileFilter Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

Let's print all files and directories in the current directory and then filter a file whose name ends with **t**.

IOTester.java

```
import java.io.File;
import java.io.IOException;

import org.apache.commons.io.filefilter.WildcardFileFilter;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingWildcardFileFilter();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingWildcardFileFilter() throws IOException {
      //get the current directory
      File currentDirectory = new File(".");

      //get names of all files and directory in current directory
      String[] files = currentDirectory.list();
      System.out.println("All files and Folders.\n");
      for( int i = 0; i < files.length; i++ ) {
         System.out.println(files[i]);
      }

      System.out.println("\nFile name ending with t.\n");
      String[] filesNames = currentDirectory.list( new WildcardFileFilter("*t")
);
      for( int i = 0; i < filesNames.length; i++ ) {
         System.out.println(filesNames[i]);
      }
   }
}
```

Output

It will print the following result:

```
All files and Folders.

.classpath
.project
.settings
bin
input.txt
src

File name ending with t

.project
input.txt
```

11. Apache Commons IO — SuffixFileFilter

SuffixFileFilter filters the files based on suffix. This is used in retrieving all the files of a particular type.

Class Declaration

Following is the declaration for **org.apache.commons.io.filefilter.SuffixFileFilter** Class:

```
public class SuffixFileFilter
   extends AbstractFileFilter implements Serializable
```

Example of SuffixFileFilter Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

Let's print all files and directories in the current directory and then filter a file with extension txt.

IOTester.java

```
import java.io.File;
import java.io.IOException;

import org.apache.commons.io.filefilter.SuffixFileFilter;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingSuffixFileFilter();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingSuffixFileFilter() throws IOException {
      //get the current directory
      File currentDirectory = new File(".");

      //get names of all files and directory in current directory
      String[] files = currentDirectory.list();
      System.out.println("All files and Folders.\n");
      for( int i = 0; i < files.length; i++ ) {
         System.out.println(files[i]);
      }

      System.out.println("\nFile with extenstion txt\n");
      String[] filesNames = currentDirectory.list( new SuffixFileFilter("txt")
);
      for( int i = 0; i < filesNames.length; i++ ) {
         System.out.println(filesNames[i]);
      }
   }
}
```

Output

It will print the following result:

```
All files and Folders.

.classpath
.project
.settings
bin
input.txt
src

File with extenstion txt

input.txt
```

12. Apache Commons IO — PrefixFileFilter

PrefixFileFilter filters the files which are based on prefix.

Class Declaration

Following is the declaration for **org.apache.commons.io.filefilter.PrefixFileFilter** Class:

```
public class PrefixFileFilter
   extends AbstractFileFilter implements Serializable
```

Example of PrefixFileFilter Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

Let's print all files and directories in the current directory and then, filter a file with name starting with input.

IOTester.java

```
import java.io.File;
import java.io.IOException;

import org.apache.commons.io.filefilter.PrefixFileFilter;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingPrefixFileFilter();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingPrefixFileFilter() throws IOException {
      //get the current directory
      File currentDirectory = new File(".");
      //get names of all files and directory in current directory
      String[] files = currentDirectory.list();
      System.out.println("All files and Folders.\n");
      for( int i = 0; i < files.length; i++ ) {
         System.out.println(files[i]);
      }

      System.out.println("\nFile starting with input\n");
      String[] filesNames = currentDirectory.list( new
PrefixFileFilter("input") );
      for( int i = 0; i < filesNames.length; i++ ) {
         System.out.println(filesNames[i]);
      }
   }
}
```

Output

It will print the following result:

```
All files and Folders.

.classpath
.project
.settings
bin
input.txt
src

File with extenstion txt

input.txt
```

13. Apache Commons IO — OrFileFilter

OrFileFilter provides conditional OR logic across a list of file filters. This returns true, if any filters in the list return true. Otherwise, it returns false.

Class Declaration

Following is the declaration for **org.apache.commons.io.filefilter.OrFileFilter** Class:

```
public class OrFileFilter
   extends AbstractFileFilter implements ConditionalFileFilter, Serializable
```

Example of OrFileFilter Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

Let's print all files and directories in the current directory and then, filter a file with name starting with . or ends with t.

IOTester.java

```
import java.io.File;
import java.io.IOException;

import org.apache.commons.io.filefilter.OrFileFilter;
import org.apache.commons.io.filefilter.PrefixFileFilter;
import org.apache.commons.io.filefilter.WildcardFileFilter;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingOrFileFilter();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingOrFileFilter() throws IOException {
      //get the current directory
      File currentDirectory = new File(".");

      //get names of all files and directory in current directory
      String[] files = currentDirectory.list();
      System.out.println("All files and Folders.\n");
      for( int i = 0; i < files.length; i++ ) {
         System.out.println(files[i]);
      }

      System.out.println("\nFile starting with . or ends with t\n");
      String[] filesNames = currentDirectory.list(
         new OrFileFilter(new PrefixFileFilter("."), new
WildcardFileFilter("*t")));
      for( int i = 0; i < filesNames.length; i++ ) {
         System.out.println(filesNames[i]);
      }
   }
}
```

Output

It will print the following result:

```
All files and Folders.

.classpath
.project
.settings
bin
input.txt
src

File starting with . or ends with t

.classpath
.project
.settings
input.txt
```

14. Apache Commons IO — AndFileFilter

AndFileFilter provides conditional and logic across a list of file filters. This returns true, if all filters in the list return true. Otherwise, it returns false.

Class Declaration

Following is the declaration for **org.apache.commons.io.filefilter.AndFileFilter** Class:

```
public class AndFileFilter
   extends AbstractFileFilter implements ConditionalFileFilter, Serializable
```

Example of AndFileFilter Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

Let's print all files and directories in the current directory and then, filter a file with name starting with . and ends with t.

IOTester.java

```
import java.io.File;
import java.io.IOException;

import org.apache.commons.io.filefilter.AndFileFilter;
import org.apache.commons.io.filefilter.PrefixFileFilter;
import org.apache.commons.io.filefilter.WildcardFileFilter;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingAndFileFilter();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingAndFileFilter() throws IOException {
      //get the current directory
      File currentDirectory = new File(".");

      //get names of all files and directory in current directory
      String[] files = currentDirectory.list();
      System.out.println("All files and Folders.\n");
      for( int i = 0; i < files.length; i++ ) {
         System.out.println(files[i]);
      }

      System.out.println("\nFile starting with . and ends with t\n");
      String[] filesNames = currentDirectory.list(
         new AndFileFilter(new PrefixFileFilter("."), new
WildcardFileFilter("*t")));
      for( int i = 0; i < filesNames.length; i++ ) {
         System.out.println(filesNames[i]);
      }
   }
}
```

Output

It will print the following result:

```
All files and Folders.

.classpath
.project
.settings
bin
input.txt
src

File starting with . or ends with t

.project
```

15. Apache Commons IO — FileEntry

FileEntry provides the state of a file or directory. File attributes at a point in time.

Class Declaration

Following is the declaration for **org.apache.commons.io.monitor.FileEntry** Class:

```
public class FileEntry
   extends Object implements Serializable
```

Features of FileEntry

FileEntry class object provides the following file attributes at a point in time:

- getName() – file name.
- exists() – checks if file exists or not.
- isDirectory() – checks if file is a directory.
- lastModified() – gives last modified date time.
- listFiles() – gives content of directory.

Example of FileEntry Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

IOTester.java

```
import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileUtils;
import org.apache.commons.io.monitor.FileEntry;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingFileEntry();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingFileEntry() throws IOException {
      //get the file object
      File file = FileUtils.getFile("input.txt");

      FileEntry fileEntry = new FileEntry(file);

      System.out.println("Monitored File: " + fileEntry.getFile());
      System.out.println("File name: " + fileEntry.getName());
      System.out.println("Is Directory: " + fileEntry.isDirectory());
   }
}
```

Output

It will print the following result:

```
Monitored File: input.txt
File name: input.txt
Is Directory: false
```

16. Apache Commons IO — FileAlterationObserver

FileAlterationObserver represents the state of files below a root directory, checks the filesystem and notifies listeners of create, change or delete events.

Class Declaration

Following is the declaration for **org.apache.commons.io.monitor.FileAlterationObserver** Class:

```
public class FileAlterationObserver
   extends Object implements Serializable
```

Example of FileAlterationObserver Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

IOTester.java

```
import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileDeleteStrategy;
import org.apache.commons.io.FileUtils;
import org.apache.commons.io.monitor.FileAlterationListenerAdaptor;
import org.apache.commons.io.monitor.FileAlterationMonitor;
import org.apache.commons.io.monitor.FileAlterationObserver;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingFileAlterationObserver();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingFileAlterationObserver() throws IOException {
      //get the file object
      File inputFile = FileUtils.getFile("input.txt");
      String absolutePath = inputFile.getAbsolutePath();
      String parent =
absolutePath.substring(0,absolutePath.indexOf("input.txt"));
      File parentDirectory = FileUtils.getFile(parent);

      FileAlterationObserver observer = new
FileAlterationObserver(parentDirectory);

      observer.addListener(new FileAlterationListenerAdaptor() {

         @Override
         public void onDirectoryCreate(File file) {
            System.out.println("Folder created: " + file.getName());
         }

         @Override
         public void onDirectoryDelete(File file) {
            System.out.println("Folder deleted: " + file.getName());
         }

         @Override
         public void onFileCreate(File file) {
            System.out.println("File created: " + file.getName());
         }

         @Override
         public void onFileDelete(File file) {
            System.out.println("File deleted: " + file.getName());
         }
      });

      //create a monitor to check changes after every 500 ms
      FileAlterationMonitor monitor = new FileAlterationMonitor(500, observer);

      try {
         monitor.start();

         //create a new directory
         File newFolder = new File("test");
         File newFile = new File("test1");

         newFolder.mkdirs();
         Thread.sleep(1000);
         newFile.createNewFile();
         Thread.sleep(1000);
         FileDeleteStrategy.NORMAL.delete(newFolder);
         Thread.sleep(1000);
         FileDeleteStrategy.NORMAL.delete(newFile);
         Thread.sleep(1000);

         monitor.stop(10000);

      } catch(IOException e) {
         System.out.println(e.getMessage());
      } catch(InterruptedException e) {
         System.out.println(e.getMessage());
      } catch (Exception e) {
         System.out.println(e.getMessage());
      }
   }
}
```

Output

It will print the following result:

```
Folder created: test
File created: test1
Folder deleted: test
File deleted: test1
```

17. Apache Commons IO — FileAlterationMonitor

FileAlterationMonitor represents a thread that spawns a monitoring thread triggering any registered FileAlterationObserver at a specified interval.

Class Declaration

Following is the declaration for **org.apache.commons.io.monitor.FileAlterationMonitor** Class:

```
public final class FileAlterationMonitor
   extends Object implements Runnable
```

Example of FileAlterationMonitor Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

IOTester.java

```
import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileDeleteStrategy;
import org.apache.commons.io.FileUtils;
import org.apache.commons.io.monitor.FileAlterationListenerAdaptor;
import org.apache.commons.io.monitor.FileAlterationMonitor;
import org.apache.commons.io.monitor.FileAlterationObserver;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingFileAlterationMonitor();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingFileAlterationMonitor() throws IOException {
      //get the file object
      File inputFile = FileUtils.getFile("input.txt");
      String absolutePath = inputFile.getAbsolutePath();
      String parent =
absolutePath.substring(0,absolutePath.indexOf("input.txt"));
      File parentDirectory = FileUtils.getFile(parent);

      FileAlterationObserver observer = new
FileAlterationObserver(parentDirectory);

      observer.addListener(new FileAlterationListenerAdaptor(){

         @Override
         public void onDirectoryCreate(File file) {
            System.out.println("Folder created: " + file.getName());
         }

         @Override
         public void onDirectoryDelete(File file) {
            System.out.println("Folder deleted: " + file.getName());
         }

         @Override
         public void onFileCreate(File file) {
            System.out.println("File created: " + file.getName());
         }

         @Override
         public void onFileDelete(File file) {
            System.out.println("File deleted: " + file.getName());
         }
      });

      //create a monitor to check changes after every 500 ms
      FileAlterationMonitor monitor = new FileAlterationMonitor(500, observer);

      try {
         monitor.start();

         //create a new directory
         File newFolder = new File("test");
         File newFile = new File("test1");

         newFolder.mkdirs();
         Thread.sleep(1000);
         newFile.createNewFile();
         Thread.sleep(1000);
         FileDeleteStrategy.NORMAL.delete(newFolder);
         Thread.sleep(1000);
         FileDeleteStrategy.NORMAL.delete(newFile);
         Thread.sleep(1000);

         monitor.stop(10000);

      } catch(IOException e) {
         System.out.println(e.getMessage());
      } catch(InterruptedException e) {
         System.out.println(e.getMessage());
      } catch (Exception e) {
         System.out.println(e.getMessage());
      }
   }
}
```

Output

It will print the following result:

```
Folder created: test
File created: test1
Folder deleted: test
File deleted: test1
```

18. Apache Commons IO — NameFileComparator

NameFileComparator compare the names of two files. It can be used to sort lists or arrays of files by using their name either in a case-sensitive, case-insensitive or system dependent case sensitive way.

Class Declaration

Following is the declaration for **org.apache.commons.io.comparator.NameFileComparator** Class:

```
public class NameFileComparator
   extends Object implements Serializable
```

Example of NameFileComparator Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

IOTester.java

```
import java.io.File;
import java.io.FileFilter;
import java.io.IOException;

import org.apache.commons.io.IOCase;
import org.apache.commons.io.comparator.NameFileComparator;
import org.apache.commons.io.filefilter.FileFileFilter;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingNameFileComparator();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }


   public static void usingNameFileComparator() throws IOException {
      //get the current directory
      File currentDirectory = new File(".");


      NameFileComparator comparator = new
NameFileComparator(IOCase.INSENSITIVE);


      File[] sortedFiles =
comparator.sort(currentDirectory.listFiles((FileFilter)FileFileFilter.FILE));


      System.out.println("Sorted By Name: ");
      for(File file:sortedFiles) {
         System.out.println(file.getName());
      }
   }
}
```

Output

It will print the following result:

```
Sorted By Name:
.classpath
.project
input.txt
```

19. Apache Commons IO — SizeFileComparator

SizeFileComparator compare the sizes of two files/directory. It can be used to sort lists or arrays of files by using their size or directories based on their number of children.

Class Declaration

Following is the declaration for **org.apache.commons.io.comparator.SizeFileComparator** Class:

```
public class SizeFileComparator
   extends Object implements Serializable
```

Example of SizeFileComparator Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

IOTester.java

```
import java.io.File;
import java.io.FileFilter;
import java.io.IOException;

import org.apache.commons.io.comparator.SizeFileComparator;
import org.apache.commons.io.filefilter.FileFileFilter;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingSizeFileComparator();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingSizeFileComparator() throws IOException {
      //get the current directory
      File currentDirectory = new File(".");

      SizeFileComparator comparator = new SizeFileComparator();

      File[] sortedFiles =
comparator.sort(currentDirectory.listFiles((FileFilter)FileFileFilter.FILE));

      System.out.println("Sorted By Size: ");
      for(File file:sortedFiles) {
         System.out.println(file.getName() + ", size(kb) :" + file.length());
      }
   }
}
```

Output

It will print the following result:

```
Sorted By Size:
input.txt, size:124
.project, size:382
.classpath, size:441
```

20. Apache Commons IO — LastModifiedFileComparator

LastModifiedFileComparator compare the last modified dates of two files/directory. It can be used to sort lists or arrays of files/directories using their last modified dates.

Class Declaration

Following is the declaration for **org.apache.commons.io.comparator.LastModifiedFileComparator** Class:

```
public class LastModifiedFileComparator
   extends Object implements Serializable
```

Example of LastModifiedFileComparator Class

Here is the input file we need to parse:

```
Welcome to TutorialsPoint. Simply Easy Learning.
```

IOTester.java

```
import java.io.File;
import java.io.FileFilter;
import java.io.IOException;
import java.util.Date;

import org.apache.commons.io.comparator.LastModifiedFileComparator;
import org.apache.commons.io.filefilter.FileFileFilter;

public class IOTester {
   public static void main(String[] args) {
      try {
         usingLastModifiedFileComparator();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingLastModifiedFileComparator() throws IOException {
      //get the current directory
      File currentDirectory = new File(".");


      LastModifiedFileComparator comparator = new LastModifiedFileComparator();


      File[] sortedFiles =
comparator.sort(currentDirectory.listFiles((FileFilter)FileFileFilter.FILE));


      System.out.println("Sorted By Last Modified date: ");
      for(File file:sortedFiles) {
         System.out.println(file.getName() + ", Modified on: " + new
Date(file.lastModified()));
      }
   }
}
```

Output

It will print the following result:

```
Sorted By Last Modified date:
.project, Modified on: Sat Dec 05 16:46:49 IST 2020
.classpath, Modified on: Sat Dec 05 16:51:17 IST 2020
input.txt, Modified on: Sat Dec 05 17:19:49 IST 2020
```

21. Apache Commons IO — TeeInputStream

It is an InputStream proxy that transparently writes a copy of all bytes read from the proxy stream to a given OutputStream. The proxy input stream is closed, when the close() method on this proxy is called. It can be used to operate two streams collectively at a time.

Class Declaration

Following is the declaration for **org.apache.commons.io.input.TeeInputStream** Class:

```
public class TeeInputStream
   extends ProxyInputStream
```

Example of TeeInputStream Class

In this example, closing a TeeInputStream closes the TeeInputStream as well as TeeOutputStream objects.

IOTester.java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;

import org.apache.commons.io.input.TeeInputStream;
import org.apache.commons.io.output.TeeOutputStream;

public class IOTester {

   private static final String SAMPLE = "Welcome to TutorialsPoint. Simply Easy
Learning.";
   public static void main(String[] args) {
      try {
         usingTeeInputStream();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingTeeInputStream() throws IOException {
      TeeInputStream teeInputStream = null;
      TeeOutputStream teeOutputStream = null;

      try {
         ByteArrayInputStream inputStream = new
ByteArrayInputStream(SAMPLE.getBytes("US-ASCII"));
         ByteArrayOutputStream outputStream1 = new ByteArrayOutputStream();
         ByteArrayOutputStream outputStream2 = new ByteArrayOutputStream();

         teeOutputStream = new TeeOutputStream(outputStream1, outputStream2);
         teeInputStream = new TeeInputStream(inputStream, teeOutputStream,
true);
         teeInputStream.read(new byte[SAMPLE.length()]);

         System.out.println("Output stream 1: " + outputStream1.toString());
         System.out.println("Output stream 2: " + outputStream2.toString());

      } catch (IOException e) {
         System.out.println(e.getMessage());
      } finally {
         //teeIn.close() closes teeIn and teeOut which in turn closes the out1
and out2.
         try {
            teeInputStream.close();
         } catch (IOException e) {
            System.out.println(e.getMessage());
         }
      }
   }
}
```

Output

It will print the following result:

```
Output stream 1: Welcome to TutorialsPoint. Simply Easy Learning.
Output stream 2: Welcome to TutorialsPoint. Simply Easy Learning.
```

22. Apache Commons IO — TeeOutputStream

TeeOutputStream splits OutputStream. It is named after the unix 'tee' command. It allows a stream to be branched to two streams.

Class Declaration

Following is the declaration for **org.apache.commons.io.output.TeeOutputStream** Class:

```
public class TeeOutputStream
   extends ProxyOutputStream
```

Example of TeeOutputStream Class

In this example, TeeOutputStream accepts two output streams as parameter and passing data to TeeOutputStream set data to both output streams.

IOTester.java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;

import org.apache.commons.io.input.TeeInputStream;
import org.apache.commons.io.output.TeeOutputStream;

public class IOTester {

   private static final String SAMPLE = "Welcome to TutorialsPoint. Simply Easy
Learning.";

   public static void main(String[] args) {
      try {
         usingTeeInputStream();
      } catch(IOException e) {
         System.out.println(e.getMessage());
      }
   }

   public static void usingTeeInputStream() throws IOException {
      TeeInputStream teeInputStream = null;
      TeeOutputStream teeOutputStream = null;

      try {
         ByteArrayInputStream inputStream = new
ByteArrayInputStream(SAMPLE.getBytes("US-ASCII"));
         ByteArrayOutputStream outputStream1 = new ByteArrayOutputStream();
         ByteArrayOutputStream outputStream2 = new ByteArrayOutputStream();

         teeOutputStream = new TeeOutputStream(outputStream1, outputStream2);
         teeInputStream = new TeeInputStream(inputStream, teeOutputStream,
true);
         teeInputStream.read(new byte[SAMPLE.length()]);

         System.out.println("Output stream 1: " + outputStream1.toString());
         System.out.println("Output stream 2: " + outputStream2.toString());

      } catch (IOException e) {
         System.out.println(e.getMessage());
      } finally {
         //teeIn.close() closes teeIn and teeOut which in turn closes the out1
and out2.
         try {
            teeInputStream.close();
         } catch (IOException e) {
            System.out.println(e.getMessage());
         }
      }
   }
}
```

Output

It will print the following result:

```
Output stream 1: Welcome to TutorialsPoint. Simply Easy Learning.
Output stream 2: Welcome to TutorialsPoint. Simply Easy Learning.
```