Assignment

On

File I/O Classes and Method

Course Name: Advance net-based JAVA

Course code: SE-409

Submitted To:

Md. Safaet Hossain
Associate Professor and Head
City University

Submitted by:

MD.Rasel Hossain
ID: 163432521
Batch-43rd
Department of CSE

Submission Date: 30/09/2020

The Java Input/Output Classes:

BufferedInputStream	FileWriter
PipedInputStream	BufferedOutputStream
FilterInputStream	PipedOutputStream
BufferedReader	FilterOutputStream
PipedReader	BufferedWriter
FilterReader	PipedWriter
ByteArrayInputStream	FilterWriter
PrintStream	ByteArrayOutputStream
InputStream	PrintWriter
CharArrayReader	InputStreamReader
PushbackInputStream	CharArrayWriter
LineNumberReader	PushbackReader
DataInputStream	ObjectInputStream
RandomAccessFile	DataOutputStream
ObjectInputStream.GetField	Reader
File	ObjectOutputStream
SequenceInputStream	FileDescriptor
ObjectOutputStream.PutField	SerializablePermission
FileInputStream	ObjectStreamClass
StreamTokenizer	FileOutputStream
ObjectStreamField	StringReader
FilePermission	OutputStream
StringWriter	FileReader
OutputStreamWriter	Writer

File Method

File defines many methods that obtain the standard properties of a **File** object. For example, **getName( )** returns the name of the file, **getParent( )** returns the name of the parent directory, and **exists( )** returns **true** if the file exists, **false** if it does not. The **File** class, however, is not symmetrical. By this, we mean that there are many methods that allow you

to *examine* the properties of a simple file object, but no corresponding function exists to change those attributes. The following example demonstrates several of the **File** methods:

```
// Demonstrate File.
 import java.io.File;

class FileDemo {
            static void p(String s) {
              System.out.println(s);
}

public static void main(String args[]) {
File f1 = new File("/java/COPYRIGHT");
p("File Name: " + f1.getName());
p("Path: " + f1.getPath());
p("Abs Path: " + f1.getAbsolutePath());
p("Parent: " + f1.getParent());
p(f1.exists() ? "exists" : "does not exist");
p(f1.canWrite() ? "is writeable" : "is not writeable");
p(f1.canRead() ? "is readable" : "is not readable");
p("is " + (f1.isDirectory() ? "" : "not" + " a directory"));
p(f1.isFile() ? "is normal file" : "might be a named pipe");
p(f1.isAbsolute() ? "is absolute" : "is not absolute");
p("File last modified: " + f1.lastModified());
p("File size: " + f1.length() + " Bytes");
}
}
```

When run this program, we see something similar to the following:

```
File Name: COPYRIGHT
Path: /java/COPYRIGHT
Abs Path: /java/COPYRIGHT
Parent: /java
exists
is writeable
is readable
is not a directory
is normal file
is absolute
```

Most of the **File** methods are self-explanatory. **isFile( )** and **isAbsolute( )** are not. **isFile( )** returns **true** if called on a file and **false** if called on a directory. Also, **isFile( )** returns **false** or some special files, such as device drivers and named pipes, so this method can be used to make sure the file will behave as a file. The **isAbsolute( )** method returns **true** if the file has an absolute path and **false** if its path is relative. **File** also includes two useful utility methods. The first is **renameTo( )**, shown here:

boolean renameTo(File *newName*)

Here, the filename specified by *newName* becomes the new name of the invoking **File** Object. It will return **true** upon success and **false** if the file cannot be renamed (if you Either attempt to rename a file so that it moves from one directory to another or use an Existing filename, for example).The second utility method is **delete( )**, which deletes the disk file represented by the Path of the invoking **File** object. It is shown here:

boolean delete( )

also use **delete( )** to delete a directory if the directory is empty. **delete( )** returns **true** if it deletes the file and **false** if the file cannot be removed.Here are some other **File** methods that you will find helpful. (They were added by Java 2.)

Method	Description
void deleteOnExit( )	Removes the file associated with the invoking object when the Java Virtual Machine terminates.
boolean isHidden( )	Returns **true** if the invoking file is hidden. Returns **false** otherwise.
boolean setLastModified(long *millisec)*	Sets the time stamp on the invoking file to that specified by *millisec*, which is the number of milliseconds from January 1, 1970, Coordinated Universal Time (UTC).
boolean setReadOnly( )	Sets the invoking file to read-only. Also, because **File** supports

The Stream Classes

Java’s stream-based I/O is built upon four abstract classes: **InputStream**, **OutputStream**, **Reader**, and **Writer**. a programs perform their I/O operations through concrete subclasses, the top-level classes define the basic functionality common to all stream classes **InputStream** and **OutputStream** are designed for byte streams. **Reader** and **Writer** are designed for character streams. The byte stream classes and the character stream classes form separate hierarchies. In general, you should use the character stream classes when working with characters or strings, and use the byte stream classes when working with bytes or other binary objects.

<u>The Byte Streams:</u>

The byte stream classes provide a rich environment for handling byte-oriented I/O. A byte stream can be used with any type of object, including binary data. This versatility makes byte streams important to many types of programs. Since the byte stream classes are topped by **InputStream** and **OutputStream**,

<u>InputStream:</u>

InputStream is an abstract class that defines Java’s model of streaming byte input. All of the methods in this class will throw an **IOException** on error conditions. Table shows the methods in **InputStream**.

Method	Description
int available( )	Returns the number of bytes of input currently available for reading.
void close( )	Closes the input source. Further read attempts will generate an **IOException**.
void mark(int *numBytes*)	Places a mark at the current point in the input stream that will remain valid until *numBytes* bytes are read.
boolean markSupported( )	Returns **true** if **mark( )**/**reset( )** are supported by the invoking stream.
int read( )	Returns an integer representation of the next available byte of input. –1 is returned when the end of the file is encountered
int read(byte *buffer*[ ])	Attempts to read up to *buffer.length* bytes into *buffer* and returns the actual number of bytes that were successfully read. –1 is returned when the end of the file is encountered.
int read(byte *buffer*[ ], int *offset*, int *numBytes*)	Attempts to read up to *numBytes* bytes into *buffer* starting at *buffer*[*offset*], returning the number of bytes successfully read. –1 is returned when the end of the file is encountered.
void reset( )	Resets the input pointer to the previously set mark.
long skip(long *numBytes*)	Ignores (that is, skips) *numBytes* bytes of input, returning the number of bytes actually ignored.

OutputStream:

OutputStream is an abstract class that defines streaming byte output. All of the methods in this class return a **void** value and throw an **IOException** in the case of errors. Table shows the methods in **OutputStream**.

Method	Description
void close( )	Closes the output stream. Further write attempts will generate an **IOException**.
void flush( )	Finalizes the output state so that any buffers are cleared. That is, it flushes the output buffers.
void write(int *b*)	Writes a single byte to an output stream. Note that the parameter is an **int**, which allows you to call **write( )** with expressions without having to cast them back to **byte**.
void write(byte *buffer*[ ])	Writes a complete array of bytes to an output stream.
void write(byte *buffer*[ ], int *offset*, int *numBytes*)	Writes a subrange of *numBytes* bytes from the array *buffer*, beginning at *buffer*[*offset*].