The **File class** deals directly with files and the file system. It describes the properties of a file itself.
A File object is used to obtain or manipulate the information associated with a disk file, such as the permissions, time, date, and directory path, and to navigate subdirectory hierarchies.

```
import java.io.File;
import java.util.Date;
public class FileDemo {
   public static void main(String args[]) {
      String path = "D:\\2018-19\\OOPJ\\OOPJSessional3\\folder1\\myfile1.txt";
      File f1 = new File(path);                  // path to a file
      if (f1.exists()) {
         System.out.println("File exists");
      } else {
         System.out.println("File not exists");
      }

      path = "D:\\2018-19\\OOPJ";            //path to a directory
      f1 = new File(path);
      if (f1.exists()) {
         System.out.println("File exists");
      } else {
         System.out.println("File not exists");
      }

     File f2 = new File("D:\\2018-19\\OOPJ\\OOPJSessional3\\folder1", "myfile1.txt");
      if (f2.exists()) {
         System.out.println("File exists");
      } else {
         System.out.println("File not exists");
      }

      File f3 = new File(new File("D:\\2018-19\\OOPJ\\OOPJSessional3\\folder1"),
"myfile1.txt");
      if (f3.exists()) {
         System.out.println("File exists");
      } else {
         System.out.println("File not exists");
      }
```

System.out.println("length() = " + f3.length()); **length() = 21**
System.out.println("canRead() = " + f3.canRead()); **canRead() = true**
System.out.println("canWrite() = " + f3.canWrite()); **canWrite() = true**
System.out.println("isDirectory() = " + f3.isDirectory()); **isDirectory() = false**
System.out.println("isDirectory() = " + f1.isDirectory()); **isDirectory() = true**
System.out.println("isFile() = " + f3.isFile()); **isFile() = true**
System.out.println("isHidden() = " + f3.isHidden()); **isHidden() = false**
System.out.println("lastModified() = " + new Date(f3.lastModified()));
lastModified() = Sat Sep 08 09:45:00 IST 2018

//this prints the list of all files and directories in current directory
File f = new File("**./**");
for (String s : **f.list()**) {
    System.out.println(s);
}
If not a directory then list() returns null.

System.out.println(f3.**getName()**); myfile1.txt
System.out.println(f3.**getParent()**); D:\2018-19\OOPJ\OOPJSessional3\folder1

System.out.println(f2.**getAbsolutePath()**);
D:\2018-19\OOPJ\OOPJSessional3\folder1\myfile1.txt
System.out.println(f3.**getPath()**);
D:\2018-19\OOPJ\OOPJSessional3\folder1\myfile1.txt

//moves the file after renaming to current directory
boolean renameTo = f3.**renameTo(new File("NewName.txt"));**
System.out.println(renameTo);

//renames the file
boolean renameTo = f3.**renameTo(new File("D:/2018-19/OOPJ/OOPJSessional3/folder1 /NewFile.txt"));**
System.out.println(renameTo);
If file already exists then does not rename.

//deletes a file if exists and deletes a directory if empty
boolean delete = f3.**delete();**
System.out.println(delete);

Java's stream-based I/O is built upon four abstract classes:
InputStream, OutputStream, Reader, and Writer.
InputStream and OutputStream are designed for byte streams. Reader and Writer are designed for character streams
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.

Object
- InputStream
  - FileInputStream
  - FilterInputStream
    - BufferedInputStream
- OutputStream
  - FileOutputStream
  - FilterOutputStream
    - BufferedOutputStream
    - PrintStream

- **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 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 an expression without having to cast it 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].

FileInputStream(String filePath)
FileInputStream(File fileObj)

FileOutputStream(String filePath)
FileOutputStream(File fileObj)
FileOutputStream(String filePath, boolean append)
FileOutputStream(File fileObj, boolean append)
If append is true, the file is opened in append mode.

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class DemoFileInputOutputStream {
	public static void main(String args[]) {
		FileOutputStream fos = null;
		try {
			fos = new FileOutputStream("file1.txt");
			for (int i = 0; i < 5; i++) {
				fos.write(i);
			}
		} catch (FileNotFoundException ex) {
		} catch (IOException ex) {
		} finally {
			try {
				fos.close();
			} catch (IOException ex) {
			}
		}
		FileInputStream fis = null;
		try {
			fis = new FileInputStream("file1.txt");
			int i;
			while ((i = fis.read()) != -1) {
				System.out.println(i);
			}
		} catch (FileNotFoundException ex) {
		} catch (IOException ex) {
		} finally {
			try {
				fis.close();
			} catch (IOException ex) {
			}
		}
	}
}
```

Output : 0 1 2 3 4

Output to file with byte[]

```
String s = "Hello World";
byte[] b = s.getBytes();
fos.write(b);
```

Input from file with byte[]

```
byte[] b = new byte[1024];
int read = fis.read(b);
System.out.println(read);              //11
System.out.println(b);                 //[B@659e0bfd
System.out.println(new String(b));//Hello World
```

Buffered Byte Streams

For the byte-oriented streams, a buffered stream extends a filtered stream class by attaching a memory buffer to the I/O stream. This buffer allows Java to do I/O operations on more than a byte at a time, thereby improving performance.

BufferedInputStream(InputStream *inputStream)*

BufferedOutputStream(OutputStream *outputStream*)

```
import java.io.*;
public class FileCopyWithBuffer {
	public static void main(String args[]) {
		String src = "file1.txt";
		String dest1 = "copyfile1.txt";
		String dest2 = "copy2file1.txt";
		long t1, t2;
		FileInputStream fis = null;
		FileOutputStream fos = null;
		try {
			fis = new FileInputStream(src);
			fos = new FileOutputStream(dest1);
			int i;
			t1 = System.nanoTime();
			while ((i = fis.read()) != -1)
				fos.write(i);
			t2 = System.nanoTime();
			System.out.println(t2 - t1);
		} catch (IOException ex) {
		} finally {
			try {
				fis.close();
				fos.close();
			} catch (IOException ex) {
			}
		}
```

```
		BufferedInputStream bis = null;
		BufferedOutputStream bos = null;
		try {
			bis = new BufferedInputStream(new FileInputStream(src));
			bos = new BufferedOutputStream(new FileOutputStream(dest2));
			int i;
			t1 = System.nanoTime();
			while ((i = bis.read()) != -1)
				bos.write(i);
			t2 = System.nanoTime();
			System.out.println(t2 - t1);
		} catch (FileNotFoundException ex) {
		} catch (IOException ex) {
		} finally {
			try {
				bis.close();
				bos.close();
			} catch (IOException ex) {
}	}	}	}
```

Finding content type using URLConnection class

```
import java.io.File;
import java.net.URLConnection;
public class CheckContentTypeDemo {
	public static void main(String args[]) {
		File f = new File("D:\\2018-19\\OOPJ\\OOPJSessional3\\folder1");
		for (String s : f.list()) {
		String path = "D:\\2018-19\\OOPJ\\OOPJSessional3\\folder1\\" + s;
		String g= URLConnection.guessContentTypeFromName(path);
		System.out.println(s + "  " + g);
}	}	}
```

```
myfile1.txt                                  text/plain
myfile_1.txt                                 text/plain
New Bitmap Image.bmp                         image/bmp
New Microsoft Office Excel Worksheet.xlsx    null
New Microsoft Office Word Document.docx      null
New WinRAR ZIP archive.zip                   application/zip
SamplePdf.pdf                                application/pdf
```

PrintStream defines several constructors. Some of them are:

1. **PrintStream(OutputStream *outputStream*)**

```
PrintStream ps = new PrintStream(System.out);
ps.println(1900);
ps.println("Hello Java");
ps.println("Welcome to Java");
PrintStream ps = null;
try {
        ps = new PrintStream(new FileOutputStream("mfile.txt"));
        ps.println(1900);
        ps.println("HelloJava");
        ps.println("Welcome to Java");
} catch (FileNotFoundException ex) {
} finally {
        ps.close();
}
```

2. **PrintStream(String outputFileName) throws FileNotFoundException**

```
PrintStream ps = null;
try {
        ps = new PrintStream("mynewfile.txt");
        ps.println(1900);
        ps.println("Hello Java");
        ps.println("Welcome to Java");
} catch (FileNotFoundException ex) {
} finally {
        ps.close();
}
```

3. **PrintStream(File outputFile) throws FileNotFoundException**

```
PrintStream ps = null;
try {
        ps = new PrintStream(new File("mysample.txt"));
        ps.println(1900);
        ps.println("Hello Java");
        ps.println("Welcome to Java");
} catch (FileNotFoundException ex) {
} finally {
        ps.close();
}
```

FileSplitter

```
import java.io.*;
public class FileSplitter {
	public static void main(String args[]) {
		try {
BufferedInputStream bis = new BufferedInputStream(new FileInputStream(
"myfile1.txt"));
int length = bis.available();
int n = length / 1024;
for (int i = 1; i <= n + 1; i++) {
	BufferedOutputStream bos = new BufferedOutputStream(new
	FileOutputStream("Part" + i));
	byte[] b = new byte[1024];
	int x = bis.read(b);
	if (x != -1)	bos.write(b);
	bos.close();
}
bis.close();
} catch (IOException ex) {
	System.out.println(ex);
} }}
```

FileMerger

```
import java.io.*;
public class FileMerger {
	public static void main(String args[]) {
		try {
BufferedOutputStream bos=new BufferedOutputStream(new
FileOutputStream("combo"));
String fn[] = {"Part1", "Part2", "Part3", "Part4", "Part5"};
for (String fn1 : fn) {
BufferedInputStream bis = new BufferedInputStream(new FileInputStream( fn1));
	int n;
	while ((n = bis.read()) != -1)		bos.write(n);
	bis.close();
}
bos.close();
}
catch (IOException ex) {	}}}
```

Convert ByteStream to CharStream

```
import java.io.*;
public class BytetoCharStream {
        public static void main(String args[]) {
                try {
OutputStreamWriter osw = new OutputStreamWriter(new
FileOutputStream("file1.txt"));
                        String s = "hello";
                        osw.write(s);
                        osw.close();
InputStreamReader isr = new InputStreamReader(new FileInputStream("file1.txt"));
                        char[] buffer = new char[100];
                        while (isr.read(buffer) != -1) {
                                System.out.println(new String(buffer));
                        }
                } catch (FileNotFoundException ex) {
                } catch (IOException ex) {
}       }       }
```

Character Stream Classes

Program1:

```
import java.io.*;
public class CharacterStreamDemo{
        public static void main(String args[])       {
                try ( FileWriter fw = new FileWriter("File1.txt");
                      FileReader fr = new FileReader("File1.txt") ){
                        fw.write("My first character stream file");
                        fw.close();
                        int c;
                        while((c = fr.read()) != -1)
                                System.out.print((char) c);
                } catch(IOException e) {
                        System.out.println("I/O Error: " + e);
                }
        }
}
```

Program 2:

```
public class ReaderWriterCopyBufferDemo {
	public static void main(String args[]) {
		String src = "file1.txt";
		String dest1 = "copyfile1.txt";
		String dest2 = "copy2file1.txt";
		long t1, t2;
		try {
			FileReader fis = new FileReader(src);
			//FileWriter fos = new FileWriter(dest1);
				int i;
			t1 = System.nanoTime();
			while ((i = fis.read()) != -1) {
				System.out.println(i);
				// fos.write(i);
			}
			t2 = System.nanoTime();
			System.out.println(t2 - t1);
		} catch (IOException ex) {
		}
		try {
			BufferedReader fis = new BufferedReader(new FileReader(src));
			//BufferedWriter fos = new BufferedWriter(new
FileWriter(dest2));
			int i;
			t1 = System.nanoTime();
			while ((i = fis.read()) != -1) {
				System.out.println(i);
				// fos.write(i);
			}
			t2 = System.nanoTime();
			System.out.println(t2 - t1);
		} catch (FileNotFoundException ex) {
		} catch (IOException ex) {
}	}	}
```

BufferedReader has a method readLine() to read a line terminated by newline character.

BufferedWriter has a method writeLine() that adds a newline character to the intended writer.

StreamTokenixer

```
import java.io.*;
public class DemoStreamTokenizer {
	public static void main(String args[]) {
	try {
	 StreamTokenizer st = new StreamTokenizer(new FileReader("myfile.txt"));
	 while ( st.nextToken() != StreamTokenizer.TT_EOF) {
		switch (st.ttype) {
case StreamTokenizer.TT_NUMBER:
	System.out.println("Number : " + st.nval + "\tline no. " + st.lineno());
break;
case StreamTokenizer.TT_WORD:
	System.out.println("Word : " + st.sval + "\tline no. " + st.lineno());
	break;
default:
	System.out.println("Token : " + (char) st.ttype + "\tline no. " + st.lineno());
}}} catch (IOException ex) {
}}}
```

RandomAccessFile

```
import java.io.*;
public class DemoRandomAccessFile {
	public static void main(String args[]){
		try {
			String path = "myfile.txt";
			RandomAccessFile raf = new RandomAccessFile(path,"rw");
			String s[]={"abc","def","ghi"};
			for(String a : s) {
				byte b[]=a.getBytes();
				raf.write(b);
			}
			raf.seek(0);
			 int i;
			while((i=raf.read())!=-1)
				System.out.print((char)i);
		} catch (IOException ex) {
}	}	}
```

public class Student implements Serializable java.io.Serializable

```
import java.io.*;
public class ListofStudents {
	public static void main(String args[]) {
		Student s1 = new Student();
		s1.setId(1);
		s1.setName("abc");

		String filename = "StudentRecord.txt";
		FileOutputStream fos = null;
		ObjectOutputStream oos = null;
		try {
			fos = new FileOutputStream(filename);
			oos = new ObjectOutputStream(fos);
			oos.writeObject(s1);
			oos.writeObject(s1);
			fos.close();
			oos.close();
		} catch (IOException ex) {
		}

		FileInputStream fis = null;
		ObjectInputStream ois = null;
		try {
			fis = new FileInputStream(filename);
			ois = new ObjectInputStream(fis);
			Object obj;
			while ((obj = ois.readObject()) != null) {
				if (obj.getClass().getSimpleName().equals("Student")) {
					System.out.println(((Student) obj).getName());
				}
			}
			fis.close();
			ois.close();
		} catch (IOException ex) {
		} catch (ClassNotFoundException ex) {
		}
	}
}
```