

Difference between File and FileReader classes in Java. What are the effects of using File and FileReader classes for reading or locating different file types like txt, json, csv, pdf, xlsx?

Below is a **clear, practical, interview-ready explanation** of File vs FileReader in Java, and **how they behave with different file types** (txt, json, csv, pdf, xlsx). I'll also point out **what works, what breaks, and why** – this is very important for real projects and interviews.

1. Core Difference (One-Line Summary)

Class	Purpose
File	Represents a file or directory path/location
FileReader	Reads text data (characters) from a file

- ➡ ☐ File does NOT read file content
- ➡ ☐ FileReader DOES read content – but only text

2. File Class (java.io.File)

What File really does

- Points to a file or folder on disk
- Works with **metadata**, not content

Typical uses

```
File file = new File("data.txt");

file.exists();
file.getName();
file.length();
file.isDirectory();
```

Important

- ❗ File **cannot read or write data**
 - ❗ It only **locates** the file
-

Effect of File on different file types

File Type	Can File handle it?	Explanation
.txt	☑ Yes	Just path + metadata
.json	☑ Yes	Format doesn't matter
.csv	☑ Yes	Same
.pdf	☑ Yes	Same
.xlsx	☑ Yes	Same

File doesn't care about file format

3. FileReader Class (java.io.FileReader)

What FileReader does

- Reads file character by character
- Uses default character encoding
- Meant for plain text

```
FileReader reader = new FileReader("data.txt");

int ch;

while ((ch = reader.read()) != -1) {
    System.out.print((char) ch);
}
reader.close();
```

Important Limitation

- FileReader is NOT for binary files

4. Effects of `FileReader` on Different File Types

☑ TEXT FILES (Safe)

File Type	Result	Why
.txt	☑ Correct	Plain text
.json	☑ Correct	Text-based
.csv	☑ Correct	Text-based

Example:

```
FileReader fr = new FileReader("data.json"); // OK
```

☑ BINARY FILES (Problematic)

File Type	Result	Why
.pdf	☑ Garbage output	Binary format
.xlsx	☑ Corrupted text	ZIP + binary
.png	☑ Unreadable	Binary

☑ You'll get random symbols or corrupted data

☑ No exception – but output is useless

5. WHY `FileReader` FAILS FOR PDF / XLSX?

Reason	Explanation
Binary format	PDF/XLSX are not characters
Character decoding	<code>FileReader</code> tries to convert bytes → chars
Encoding mismatch	Binary data ≠ text encoding

Reading binary files as text = data corruption

6. Correct Class Choices (VERY IMPORTANT)

✓ Text Files

FileReader
BufferedReader
Files.readAllLines()

✓ Binary Files

FileInputStream
BufferedInputStream

✓ Specialized Formats

File Type	Recommended API
.pdf	Apache PDFBox
.xlsx	Apache POI
.csv	OpenCSV
.json	Jackson / Gson

7. Interview-Friendly Comparison Table

Feature	File	FileReader
Reads content	<input type="checkbox"/> No	<input type="checkbox"/> Yes
Reads characters	<input type="checkbox"/>	<input type="checkbox"/>
Binary-safe	<input type="checkbox"/>	<input type="checkbox"/>
Format aware	<input type="checkbox"/>	<input type="checkbox"/>
Purpose	File metadata	Text reading
Works with PDF/XLSX	Location only	<input type="checkbox"/> Wrong choice

8. Common Interview Trap Question ☐

Q: Can FileReader read a PDF file?

Best Answer:

Yes, it can read it technically, but it will produce corrupted output because PDF is a binary file. FileReader should only be used for text-based files.

9. Simple Rule to Remember

File → WHERE the file is
FileReader → WHAT text is inside

If you want, I can also:

- Show **code failures with PDF/XLSX**
- Compare **FileReader vs BufferedReader vs FileInputStream**
- Explain **encoding issues (UTF-8 vs default charset)**

Just tell me.

File class Works with metadata, not content. What does it mean?

Great question – this line confuses many Java learners ☹
Let's break it down **very clearly and practically**.

What does “File works with metadata, not content” mean?

Simple meaning

File tells you information *about* a file, not the data *inside* the file.

What is metadata?

Metadata = data about the file

It includes things like:

- File name
- File path
- File size
- Whether it exists
- Whether it is a file or directory
- Last modified time
- Read / write / execute permissions

Metadata ≠ actual text or data stored in the file

Example: Real-world analogy

Imagine a book:

Book Part	Java Equivalent
Book cover (title, pages, weight)	Metadata
Text written inside pages	Content

➡ ☐ File reads the cover

➡ ☐ FileReader reads the pages

Example 1: Using `File` (Metadata only)

```
File file = new File("data.txt");

System.out.println(file.getName());           // data.txt
System.out.println(file.getAbsolutePath());
System.out.println(file.length());            // size in bytes
System.out.println(file.exists());            // true/false
System.out.println(file.isFile());            // true
```

No method above reads file text

What `File` CANNOT do ❓

```
File file = new File("data.txt");
file.read();           // ❓ No such method
file.getContent();     // ❓ No such method
```

Because reading content is NOT `File`'s responsibility

Example 2: Reading content (Needs another class)

```
FileReader reader = new FileReader(file);
BufferedReader br = new BufferedReader(reader);

String line = br.readLine(); // Reads ACTUAL text inside file
```

✓ Now you are reading **content**, not metadata.

File class behavior with different file types

File Type	What File can do
.txt	Show size, name, path
.json	Same
.csv	Same
.pdf	Same
.xlsx	Same

File doesn't care about file format

Why Java designed it this way?

Separation of responsibilities

Class	Responsibility
File	Locate & describe a file
Reader / InputStream	Read file data
Writer / OutputStream	Write file data

This makes:

- Code cleaner
 - APIs safer
 - Binary & text handling separate
-

Very common interview question ?

Q: Can we read a file using only File?

Correct answer:

No. File only provides metadata. To read content, we need classes like FileReader, BufferedReader, or FileInputStream.

One-line memory trick

File = file info
Reader / Stream = file data

=====

Below is a **clear, practical, and interview-oriented comparison** of **FileReader VS BufferedReader VS FileInputStream**, with **use-cases, file types, and performance effects**.

1. One-Line Purpose of Each

Class	One-line purpose
FileReader	Reads characters from a text file
BufferedReader	Reads text efficiently using buffering
FileInputStream	Reads raw bytes (binary data)

2. What exactly do they read?

Class	Reads	Data Type
FileReader	Characters	Text
BufferedReader	Characters (buffered)	Text
FileInputStream	Bytes	Binary

3. Detailed Comparison Table (Very Important)

Feature	FileReader	BufferedReader	FileInputStream
Reads content	☒ Yes	☒ Yes	☒ Yes
Data unit	Character	Character	Byte
Buffering	☒ No	☒ Yes	☒ No
Performance	Slow	Fast	Medium
Binary safe	☒ No	☒ No	☒ Yes
Text friendly	☒ Yes	☒ Best	☒ No
Reads line by line	☒ No	☒ Yes	☒ No
File type support	Text only	Text only	All files

4. File Type Compatibility (CRITICAL)

File Type	FileReader	BufferedReader	FileInputStream
.txt	☒	☒	☒ (manual conversion)
.json	☒	☒	☒
.csv	☒	☒	☒
.pdf	☒	☒	☒
.xlsx	☒	☒	☒
.png	☒	☒	☒

☒ = Possible, but not convenient

☒ = Wrong choice

5. Code Examples (Side by Side)

❏ **FileReader (Simple text reading)**

```
FileReader fr = new FileReader("data.txt");

int ch;

while ((ch = fr.read()) != -1) {
    System.out.print((char) ch);
}

fr.close();

# Reads one character at a time → slower
```

❏ **BufferedReader (Best for text files)**

```
BufferedReader br = new BufferedReader(new FileReader("data.txt"));

String line;

while ((line = br.readLine()) != null) {
    System.out.println(line);
}

br.close();

# Reads line by line → fast & memory-efficient
```

❏ **FileInputStream (Binary files)**

```
FileInputStream fis = new FileInputStream("file.pdf");

int b;

while ((b = fis.read()) != -1) {
    System.out.print(b);
}

fis.close();

# Reads raw bytes, not text
```

6. Why BufferedReader is Faster ?

Reason	Explanation
Buffer	Reads chunks, not single chars
Fewer I/O calls	Better disk efficiency
readLine()	Easy line handling

`BufferedReader` wraps `FileReader`

7. Common Interview Traps [🔗](#)

Can `BufferedReader` read a PDF?

Answer:

No. `BufferedReader` still reads characters. PDF is binary.

Can `FileInputStream` read a text file?

Answer:

Yes, but it reads bytes – you must manually convert bytes to characters.

8. Which One Should I Use? (Rule of Thumb)

Plain text (txt, json, csv) → `BufferedReader`
Small text file → `FileReader`
Binary file (pdf, xlsx, image) → `FileInputStream`

9. Interview-Perfect Summary (Say This)

`FileReader` reads characters without buffering.
`BufferedReader` adds buffering and line-by-line reading, making it faster for text files.
`FileInputStream` reads raw bytes and is used for binary files like PDF and Excel.