# Part 1 — File Class (Deep Detail)

`java.io.File` class ka use hum karte hain:

✓ file create karne ke liye
✓ folder (directory) create karne ke liye
✓ file ka name/location check karne ke liye
✓ exist check
✓ file size
✓ read/write permissions
✓ file delete/rename
✓ directory ke files list karne ke liye

👉 **File class file ko open nahi karta — sirf representation hoti hai.**

# 1. File Object Create Karna (Very Important)

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

Ye code "data.txt" file **ko open nahi karta**
Bas **file ka path represent** karta hai.

# 2. File Create Karna

```
File f = new File("data.txt");
boolean created = f.createNewFile();

if(created)
    System.out.println("File created!");
else
    System.out.println("File already exists");
```

✓ Note: `createNewFile()` checked exception throw karta hai — FileNotFoundException nahi, **IOException**

# 3. Folder (Directory) Create Karna

```
File folder = new File("mydata");
folder.mkdir();  // create single folder
```

Multiple folder path create karne ke liye:

```
File folders = new File("a/b/c");
folders.mkdirs();  // creates full path
```

# 4. File Exists Check

```
if(f.exists()) {
    System.out.println("File exists");
} else {
    System.out.println("File does not exist");
}
```

# 5. File Delete Karna

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

if(f.delete())
    System.out.println("Deleted");
else
    System.out.println("Not deleted");
```

⚠ Note

- Delete permanent hota hai
- Recycle bin me nahi jata

# 6. File Rename Karna

```
File f1 = new File("old.txt");
File f2 = new File("new.txt");

f1.renameTo(f2);
```

# 7. File Ka Path

### Absolute Path:

```
f.getAbsolutePath();
```

### Relative Path:

```
f.getPath();
```

**Parent Directory:**

```
f.getParent();
```

# 8. File Ka Size (in bytes)

```
long size = f.length();
System.out.println("File size: " + size + " bytes");
```

# 9. Read/Write Permissions

```
f.canRead();
f.canWrite();
f.canExecute();
```

# 10. Directory Ke Sare Files List Karna

```
File dir = new File("mydata");
String[] files = dir.list();

for(String name : files) {
    System.out.println(name);
}
```

Ye sirf names deta hai.
Full File Objects chahiye to:

```
File[] files = dir.listFiles();

for(File file : files) {
    System.out.println(file.getName());
}
```

# Summary — File Class Kya Kya Kar Sakti Hai?

| Operation | Supported? |
|---|---|
| File create | ✓ |
| Folder create | ✓ |
| File delete | ✓ |

| | |
|---|---|
| Rename | ✓ |
| Check exist | ✓ |
| Read permissions | ✓ |
| Write permissions | ✓ |
| Get size | ✓ |
| Open file for reading | ❌ NO |
| Open file for writing | ❌ NO |

# Part 2 — FileReader (Deep Detail)

**FileReader** ka use hum tab karte hain jab hume **text file** se **characters** read karne hote hain.

✓ Character stream (Unicode support)

✓ Text files ke liye best

✓ Byte-by-byte nahi, character-by-character read karta hai

# 1. FileReader banana

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

❗ FileReader **exception throw** karta hai:

- FileNotFoundException
- IOException (indirectly)

Isliye code try-catch me hota hai.

# 2. Single Character Read Karna

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

int ch = fr.read();   // returns ASCII code
while(ch != -1) {
    System.out.print((char) ch); // convert to char
    ch = fr.read();
}

fr.close();
```

✓ `read()` ek character ka integer code deta hai

✓ `-1` ka matlab file end ho gayi

# 3. Buffer Array Use Karke Efficient Reading

Efficient + fast way:

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

char[] buffer = new char[100];
int length = fr.read(buffer);

while(length != -1) {
    System.out.print(new String(buffer, 0, length));
    length = fr.read(buffer);
}

fr.close();
```

✓ Faster

✓ Cleaner output

# 4. File Not Found Handling

```
try {
    FileReader fr = new FileReader("data.txt");
}
catch(FileNotFoundException e) {
    System.out.println("File not found!");
}
```

# 5. FileReader ke drawbacks

FileReader slow hota hai kyunki:

❌ characters ko direct disk se read karta hai

❌ har character ke liye disk access hota hai

**Solution:**

Use **BufferedReader**, jo cached buffer me data rakhta hai.

Hum BufferedReader agle topic me karenge.

# Mini Example — Full Program

```
import java.io.*;

class ReadFileDemo {
    public static void main(String[] args) {
       try {
```

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

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

            fr.close();
        }
        catch(Exception e) {
            System.out.println(e);
        }
    }
}
```

# Summary — FileReader

| Feature | Explanation |
|---|---|
| Purpose | Character reading |
| Reads | One char or char[] |
| Throws | FileNotFoundException, IOException |
| Speed | Slow (disk-level reading) |
| Best For | Small text files |

# Part 3 — BufferedReader (Deep Detail)

BufferedReader ka use large text files **fast read** karne ke liye hota hai.

💡 BufferedReader = Fast Character Reader (with internal buffer)

# 1. BufferedReader kyu use hota hai?

**FileReader slow hota hai** kyunki:

- har character read karte waqt disk se access hota hai
- disk access = slowest operation

**BufferedReader fast hota hai** kyunki:

- ek baar me large chunk of data memory buffer me load karta hai
- waha se fast-fast characters read hota hai

# 2. BufferedReader ka constructor

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

Yani BufferedReader → FileReader ko wrap karta hai.

# 3. Reading Methods (VERY Important)

### 1️⃣ read() → one character

```
int ch = br.read();
```

### 2️⃣ readLine() → whole line (BEST METHOD)

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

✓ readLine() **poori line** ek saath deta hai
✓ newline (\n) return nahi hoti
✓ return: String
✓ EOF (end of file) → returns **null**

# 4. Full Program — Read Full File Line-by-Line

```
import java.io.*;

class ReadBufferedDemo {
    public static void main(String[] args) {
        try {
            BufferedReader br = new BufferedReader(new
FileReader("test.txt"));

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

            br.close();
        }
        catch(Exception e) {
```

```
            System.out.println(e);
        }
    }
}
```

✓ Fast
✓ Clean
✓ Industry code

# 5. try-with-resources (Modern Way)

Java 7+ me recommended:

```
try(BufferedReader br = new BufferedReader(new
FileReader("test.txt"))) {

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

} catch(Exception e) {
    e.printStackTrace();
}
```

✓ Close karna automatic
✓ Clean code
✓ No resource leak

# 6. BufferedReader vs FileReader (Interview Table)

| Feature | FileReader | BufferedReader |
|---|---|---|
| Speed | Slow | Fast |
| Reads | Char / Char[] | Char / Char[] / **Line** |
| Buffering | No | YES |
| Best For | Small text files | Large files |
| Industry Use | Rarely | Always |

# Summary

BufferedReader:

✓ fastest character reading tool
✓ line-by-line reading
✓ most used in real projects

# Part 4 — FileWriter (Deep Detail)

`FileWriter` ka use **text file me character write** karne ke liye hota hai.

# 1. FileWriter object banana

```
FileWriter fw = new FileWriter("output.txt");
```

✓ Agar file **nahi hai** → automatically **ban jaati hai**
✓ Agar file **hai** → overwrite ho jaati hai

# 2. FileWriter me write() methods

### 1 Single character write

```
fw.write('A');
```

### 2 String write

```
fw.write("Hello World");
```

### 3 Char array write

```
char[] data = {'a','b','c'};
fw.write(data);
```

### 4 Part of String write

```
fw.write("Java Programming", 5, 11);
```

# 3. FileWriter close karna IMPORTANT

```
fw.close();
```

✓ close() buffer flush karta hai (remaining data file me daalta hai)
✓ agar close() nahi karoge → data incomplete write ho sakta hai!

# Important: Append Mode

By default file overwrite hoti hai.
File me **add (append)** karne ke liye:

```
FileWriter fw = new FileWriter("data.txt", true);
```

# Example — Writing to File

```
import java.io.*;

class WriteFileDemo {
    public static void main(String[] args) {
       try {
            FileWriter fw = new FileWriter("test.txt");

            fw.write("Hello Java!\n");
            fw.write("Writing to file using FileWriter.");

            fw.close();
        }
        catch(Exception e) {
            System.out.println(e);
        }
    }
}
```

# Part 5 — BufferedWriter (Deep Detail)

BufferedWriter writing ko **fastest** banata hai.

💡 BufferedWriter = fast write with internal buffer

# 1. BufferedWriter object

```
BufferedWriter bw = new BufferedWriter(new FileWriter("test.txt"));
```

# 2. write(), newLine(), flush()

### 1️⃣ write()

```
bw.write("Hello BufferedWriter!");
```

### 2️⃣ newLine() → platform-independent newline
(MAC, Windows, Linux sab me safe)
`bw.newLine();`

### 3️⃣ flush()
Buffer ko forcefully file me push karta hai:
`bw.flush();`

# Example — BufferedWriter Program

```
import java.io.*;

class BufferWriteDemo {
    public static void main(String[] args) {
        try {
            BufferedWriter bw = new BufferedWriter(new
FileWriter("test.txt"));

            bw.write("Hello from BufferedWriter!");
            bw.newLine();
            bw.write("This is fast and efficient.");
            bw.close();   // flush automatically hota hai
        }
        catch(Exception e) {
            System.out.println(e);
        }
    }
}
```

# FileWriter vs BufferedWriter (Interview Table)

| Feature | FileWriter | BufferedWriter |
|---|---|---|
| Speed | Medium | Very Fast |
| Buffering | No | YES |
| Methods | write() | write(), newLine(), flush() |
| Best Use | Small writing | Logging, large files, enterprise apps |

# Summary

✓ FileWriter — basic writing
✓ BufferedWriter — fast writing
✓ newLine() — safest newline
✓ append mode — `"file", true`