Files:

Filewriter will not get FilenotFoundexception, only Filereader will give filenotfoundexception.

→ used to refer or point to a file, in the given path. we can create a file using this class.

Filewriter:- used to write data into file, charaters by charater it will create a file if file doesn't exist. It will open the file if it already exists.

Bufferwriter:- used to write data into file, line by line. It is more efficent (faster) compared to Filewriter.

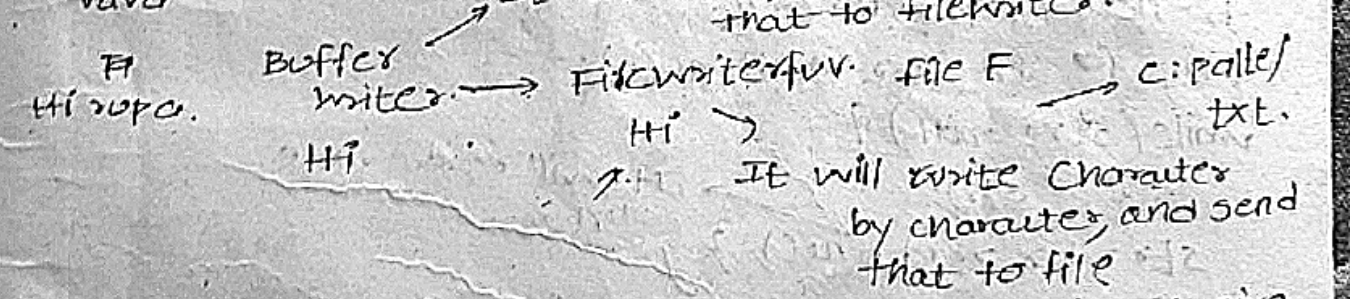
Filereader:- used to read data from file, charater by charater, it will throw FileNotFoundexception if file doesn't exist in given path.

Bufferreader():-

used to read data from file, line by line. It is more efficent (faster) compare to Filereader.

Files writing data into files:-

→ First we have to create object to that file, which contains filepath.




```
Package exception;                    → instead of this give
import java.io.FileNotFoundexception;   * import java.io.*;
Public class Sample { BufferWriter bw=Null;
    public static void main (String[] args) {
by  }   File f= new File ("D:/java/Exception.txt");
        Filewriter fw= new ~~File(f)~~ filewriter(f);
        BufferWriter bw= new bufferwrite(fw);
        bw.write("welcome to palle");
        bw.write("let us learn
        bw.flush();
            }
        catch(IoException e) {
            S.o.p("cannot write the file"); }
```

then new file will create and add

```
}
finally {
  try {
    f.close();
  catch(IOException e) {
    e.printStackTrace();
  }
}
```

→ when file is opened last, that will close at first -

```
package Exception;
import java.io.*;
Public class sample {
Public static void main(String[] args) {
  File f=null;
  FileReader fr=null;
BufferReader br=null;
try {
f=new File("D://java //Exception.txt");
fr=new FileReader(f);
String str= br.readLine();
while(str!=null) {
  S.O.P(str);
  str=br.readLine(); } }
```

→ It will read whole line

```
catch(FileNotFoundException e) {
  S.O.P("File doesn't exist");
  }
catch(IOException ie) {
    S.O.P("File may corrupted");
finally {
  try {
  br.close();
  fr.close(); }
  catch(IOException e) {
    S.O.P("file may not opened properly");
```

→ if(br!=null && fr!=null) {

Create object for these, in the follo...
Create object.

File f = new File("D:\FastFiles");

→ And there are 2-types of Path:→
→ Absolute path
→ Relative path.

Here definetly we have to use \\ or / for path.

```
import java.io.File;
public class AbsolutePathExample {
    public static void main(String[] args) {
        File file = new File("C:\\Users\\Rupa\\
        Documents\\example.txt");
        if (file.exists()) {
            S.O.P("File found at: " + file.getAbsolute
                Path());
        }
        else {
            S.O.P("File not found");
        }
    }
}
```

Relative Path:- Is a path related to the current working directory of your program. It is typically shorter and is useful when working within the directory structure of your project. The relative path does not include the root directory and is resolved from the directory where the java program is being executed.

→ If your program is running in.

C:\Users\Rupa\JavaProjects,

↓ relative path → files/example.txt.

```
File file = new File("files/example.txt");
if (file.exists()) {
    S.O.P("File found at:" + file.getAbsolute
        Path());
} else {
```

File f = new File("C:\Users\...\resources.txt");

S.o.P(f.createNewFile());

f.delete();

↓

same it will also return false and true

→ Here create NewFile, it will give true or false, if it is already there (ie if file the name is already there it will return false otherwise true).

S.o.P(f.delete());

S.o.P(f.exists()); → true or false

S.o.P(f.isHidden()); → if file hidden, it will return true otherwise false.

S.o.P(f.canWrite()); → If it is false, then we will set setWritable as true.

f.setWritable(true); →

S.o.P(f.canWrite());

S.o.P(Arrays.toString(f.list())); → It will list the files.

S.o.P(Arrays.toString(f.listFiles()));

↓

It will give in a path manner.

S.o.P(f.mkdir()); → It will create folder

S.O.P(f.mkdirs()); → It will create another file

if only one file is there, then we use dir.

am → Cancel file.

S.o.P(f.lastModified()); → but it will just numbers.

S.o.P(new Date(f.lastModified()));

S.o.P(f.isDirectory()); → true or false

S.o.P(f.isFile());

Different ways of reading the data from text files?

- FileInputStream
- Scanner
- FileReader
- BufferedReader.

(1)

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
public class Program17
    public static void main(String[] args)
            throws IOException.
```

↙

Here definitely we have to throw or else we

there are more chances to get exceptions
We have to mention exception.

```
File file = new File("./sample.txt");
if(!file.exists())
    file.createNewFile();
FileInputStream fis = new FileInputStream(file);
S.o.p(fis.read());
```

↓ ⟶ here we get

Here read, it read only one character, if character is not their it will return -1.

if we want character way then → ((char) fis.read());

```
int asciiCode;
while((asciiCode = fis.read()) != -1) {
    S.o.P((char)asciiCode); }
```

→ These stream i.e inputstream, outputstream like tap water we close after usage, same way with Stream So we have to close these.

```
fis.close();
```

Scanner:-

```
Scanner scanner = new Scanner(file);
```

(We already declare file path, so we are using this)

Scanner.nextLine() → to read line by line characters
scanner.hasNext()

```
while(scanner.hasNext()) {
    S.o.P(scanner.next());
}
scanner.close();
}
```

FileReader:-

```
import java.io.FileReader;
import java.io.IoException;
public class FileReaderExample {
```

```
try(FileReader fileReader = new FileReader
    ("file.txt")) {
    int data;
    while((data = fileReader.read()) != -1) {
        s.o.p(char(data));
    }
    catch(IOException e) {
        e.printStackTrace(); } }
}
```

Buffereader:-

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class BufferedReadExample {
    public static void main(String[] args) {
        try (BufferedReader
        bufferedReader = new
            BufferedReader(new FileReader
            ("file.txt"))) {
            String line;
while((line = bufferedReader.readLine()) !=
    null) {
    s.o.p(line);
    }
catch(IOException e) {
    e.printStackTrace()
}
}
}
```

Write:- → FileOutputStream
↳ FileWriter
↳ BufferedWriter
↳ outputStreamWriter
↳ PrintWriter

1. FileOutputStream:-

```
import java.io.FileOutputStream;
import java.io.outputStreamWriter;
import java.io.IOException;
```

```
public class FileOutputStream {
    public static void main(String[] args) {
        try(FileOutputStream fileOutput = new
FileOutputStream("output.txt");
        OutputStreamWriter outputStreamWriter =
            new OutputStreamWriter(fileOutput)) {
            String data = "Hello, World! \n This is a
                test.";
            outputStreamWriter.write(data);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

→ As usual syntax, now for inputstream, same for outputstream.

properties file:-

→ Here we store in a key value pair manner.

→ File Name and path:-

getName():→ returns the name of the file or directory

getPath():- returns the path of the file as a string.

getAbsolutePath()→ returns the absolute path of the file as a string.

getCanonicalPath():- returns the canonical path, resolving symbolic links if any.

File existence and type:-

exists():→ checks if the file or directory exists.

isFile():→ checks if the file object represents a file.

isDirectory():→ checks if the file object represents a directory.

File properties:-

length()→ returns the length of the file in bytes (returns 0 for directories).

lastModified:- returns the last modified time in milliseconds since the epoch.

canRead()- checks if the file is readable

canWrite()- checks if the file is writable

canExecute()- checks if the file is executable.

File manipulation:-

delete(): - Deletes the file or directory (must be empty for directories).
mkdir(): - Creates a directory
mkdirs(): - Creates a directory along with any necessary parent directories

Directory Handling:-

list() → Returns an array of file names in the directory as string values.
listFiles(): - Returns an array of File objects for each file/directory in the directory.
renameTo(File dest): Renames the file to the specified destination file.

File path checks:-

isAbsolute(): - Checks if the file path is absolute.
getParent(): - Returns the parent directory of the file as a string.
toURI: - Converts file path to a URI.