File - IO

1. File
2. FileWriter
3. FileReader
4. BufferedWriter
5. BufferedReader
6. PrintWriter

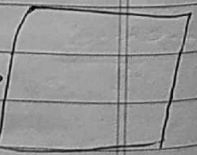
File

⇒ File f = new File("abc.txt");

This line won't create any physical file. First it will check is there any physical file available with this name. If it is available then f simply refers that file. If it is not available then we are just creating java file object to represent a name "abc.txt".

Ex.

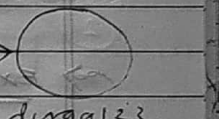
```
File f = new File("abc.txt");
s.o.p(f.exists());
f.createNewFile();
s.o.p(f.exists());
```



1st run	2nd run
false	true
true	true

⇒ We can use java File object to represent directory also

Ex. File f = new File("durga123");
s.o.p(f.exists());
f.mkdir();
s.o.p(f.exists());



o/p: false
true

Note: In UNIX everything is treated as a file. Java File IO concept is implemented based on UNIX OS, Hence Java File object can be used to represent both File and Directory

⇒ File class constructors

① File f = new File(String name);

Creates a java file object to represent name of the file/directory in CWD.

(2) File f = new File(String subdirname, String name);

creates a Java File object to represent name of the File/directory present in specified subdirectory

(3) File f = new File(File subdir, String name);

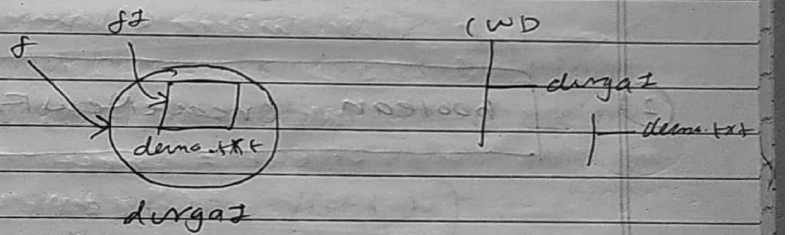
⇒ Ex-1 write code to create a file named with abc.txt in CWD.

```
File f = new File("abc.txt");
f.createNewFile();
```

CWD
|
abc.txt

⇒ Ex-2 write code to create a directory named with durga1 in CWD, and create a file named with demo.txt in that directory

```
File f = new File("durga1");
f.mkdir();
File f1 = new File("durga1", "demo.txt");
// File f1 = new File(f, "demo.txt");
f1.createNewFile();
```



⇒ Ex-3 write code to create a file demo.txt in E:\xyz folder

```
File f = new File("E:\\xyz",
                  "abc.txt");
f.createNewFile();
```

⇒ (Assume that E:\xyz folder is already available in system)

⇒ Important methods present in File class

(1) `boolean exists()`

Returns true if the specified file / directory available.

(2) `boolean createNewFile()`

First this method will check whether the specified file is available or not, if the file already available then this method will return false, If the file is not available then this method will create a physical file & returns true.

(3) `boolean mkdir()`

same functionality as above method, but for directory.

(4) `boolean isFile()`

Returns true if the specified file object pointing to physical file

(5) `boolean isDirectory()`

Returns true if the specified file object pointing to directory.

(6) `String[] list()`

This method returns the names of all files & subdirectories present in specified directory.

(7) `long length()`

Returns No. of characters present in specified file.

(8) `boolean delete()`

To delete a specified file / directory.

⇝ Ex. WAP to display the names of all File / Directories present in c:\durgaclasses.

```
import java.io.*;

class TestDir
{
    p s v main(Sm)
    {
        int count=0;
        File f = new File("c:\\durgaclasses");
        String[] s = f.list();

        for(String s1:s)
        {
            count++;
            s.o.p(s1);
        }
        s.o.p("Total Number:"+count);
    }
}
```

⇝ To display only Files

```
int count=0;
File f = new File("c:\\durgaclasses");
String[] s = f.list();

for(String s1:s)
{
    File f1 = new File(f,s1);

    if(f1.isFile())
    {
        count++;
        s.o.p(s1);
    }
}
s.o.p("Total Files:"+count);
```

(*) FileWriter

⇝ We can use FileWriter to write character data to the file.

⇒ **Constructors :-**

(1) `FileWriter fw = new FileWriter(String fname);`

(2) `FileWriter fw = new FileWriter(File f);`

→ The above FileWriters meant for overwriting of existing data.

→ Instead of overwriting, If we want to append the data to the file then we have to use following constructors:

(3) `FileWriter fw = new FileWriter (String fname, boolean append);`

(4) `FileWriter fw = new FileWriter (File f, boolean append);`

→ Note: If the specified file is not already available then all the above constructors will create that file

→ Important Methods in FileWriter:-

(1) `write (int ch)` To write single character

(2) `write (char[] ch)` To write Array of characters

(3) `write (String s)` To write string to a file.

(4) `flush()`

To give the guarantee that total data including last character written to the file

(5) `close()`

To close the FileWriter.

→ Ex.

```
import java.io.*;
class FileWriterDemo
{
    p s v main(Str) throws IOException
    {
        FileWriter fw = new
                FileWriter("abc.txt");
        fw.write(100);
        fw.write("orga in software");
        fw.write("\n");

        char[] ch1 = {'a','b','c'};
        fw.write(ch1);
        fw.write("\n");
```

```
        fw.flush();
        fw.close();
    }
}
```

O/P :

```
durga
software
abc
```

~~abcdxyz~~

⇝ In the above program, FileWriter can perform overwriting of existing data.

⇝ Instead of overwriting, If we want append operation then we have to create FileWriter as follows :

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

⇝ Note : The main problem with the FileWriter is, we have to insert line separator (\n) manually which is varied from system to system, so, it is difficulty for programmer, we can solve this problem using BufferedWriter & PrintWriter classes.

✳ FileReader

⇝ To read character data from file.

⇝ Constructors :

① `FileReader fr = new FileReader(String fname);`

② `FileReader fr = new FileReader(File f);`

⇝ Methods :

① `int read()`

⇝ It attempts to read next character from the file & returns its Unicode value.

⇝ If next character not available it returns -1.

⇝ As this method returns unicode (int) value, at a time of printing we have to perform Type-casting.

Ex. FileReader fr = new FileReader("abc.txt");

```
int i = fr.read();
while (i != -1)
{
    s.o.p((char)i);
    i = fr.read();
}
```

```
durga
software
abc
```

abc.txt

(2) `int read(char[] ch)`

→ It attempts to read enough characters from the file into char[] & returns no. of characters copied.

→ Ex. File f = new File("abc.txt");
char[] ch = new char[(int)f.length()];
FR fr = new FR(f);
fr.read(ch);

```
for (char ch1 : ch)
{
    s.o.p(ch1);
}
```

(3) `void close()`

Note: By using FileReader, we can read data character by character which is not convinient for programmer

→ Usage of FileWriter & FileReader is not recommanded because :-

- while writing data by FileWriter we have to insert line separator manually which is varied from system to system, which is difficult for programmers
- while Reading data by FileReader we can read data character by character which is not convinient to programmers.

→ To overcome these problems we should go for
BufferedWriter &
BufferedReader

BufferedWriter

→ we can use BufferedWriter to write character data to the file.

⇒ Constructors :-

1. `BufferedWriter bw = new BufferedWriter(Writer w);`

2. `BufferedWriter bw = new BufferedWriter(Writer w, int buffersize);`

Note : BufferedWriter can't communicate directly with the file, It can communicate with some Writer object.

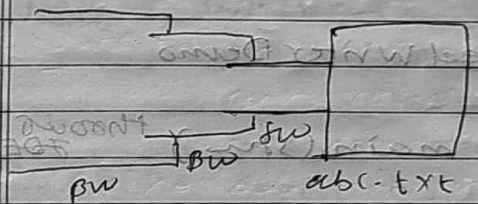
Qu Which of the following are valid ?

✗ a) `BW bw = new BW("abc.txt");`

✗ b) `BW bw = new BW(new File("abc.txt"));`

✓ c) `BW bw = new BW(new FW("abc.txt"));`

✓ d) `BW bw = new BW(new BW(new FW("abc.txt")));`



⇒ Methods :-

1. write(int ch)
2. write(char[] ch)
3. write(String s)
4. flush()
5. close()

} common methods

6. `newLine()`
To insert a line separator

Q. When compared with FileWriter which of the following capability is extra in BufferedWriter?

Ans: 1) writing data to the file
2) close the file
3) flushing the file
4) Inserting a new line character.

⇒ Ex-

```
class BufferedWriterDemo
{
    p s v main(Str..) throws IOException
    {
        FileWriter fw = new
            FileWriter("abc.txt");
        BW bw = new BW(fw);
        bw.write(100);
        bw.newLine();

        char[] ch = {'a','b','c','d'};

        bw.write(ch);
        bw.nextLine();
        bw.write("durga");
        bw.nextLine();
        bw.write("software");
        bw.flush();
        bw.close();
    }
}
```

abc.txt
```
d
abcd
durga
software
```

Note: Whenever we are closing BW automatically internal FW will be closed & we are not required to close explicitly.

BufferedReader

⇒ we can use BufferedReader to read character data from file.

⇒ The main advantage of BufferedReader, when compared with FileReader is, we can read data line by line in addition to character by character.

⇒ BufferedReader can communicate via some Reader only, can't communicate directly.

Constructors :-

1. `BufferedReader br = new BufferedReader(Reader r);`
2. `BufferedReader br = new BufferedReader(Reader r, int buffersize);`

Methods :-

1. `int read()`
2. `int read(char[] ch)`
3. `void close()`

(1, 2, 3 → common methods)

4. `String readLine()`

It attempts to read next line from the file & returns it, if the next line not available then this method returns (null)

Ex:-

```
class BufferedReaderDemo
{
    p s v main(Str[] s)
    {
        FR fr = new FR("abc.txt");
        BR br = new BR(fr);
        String line = br.readLine();
        while(line != null)
        {
            S.o.p(line);
            line = br.readLine();
        }
        br.close();
    }
}
```

abc.txt
```
durga
software
solution
```

o/p =
```
durga
software
solution
```

Whenever we are closing Buffered Reader, automatically underlying Reader will be closed, we don't need to close explicitly.

⇒ The most enhanced Reader to read character data from file is BufferedReader.

✱ PrintWriter

⇒ It is the most enhanced writer to write character data to the file.

⇒ The main advantage of printWriter over FileWriter & BufferWriter is, we can write any type of primitive data directly to the file.

⇒ **Constructors :-**

(1)
```
PrintWriter pw = new
                 PrintWriter(String fname);
```

(2)
```
PrintWriter pw = new
                 PrintWriter(File f);
```

(3)
```
PrintWriter pw = new
                 PrintWriter(Writer w);
```

⇒ Note : PrintWriter can communicate directly with the File & can communicate via some writer also.

⇒ **Methods :-**

1. write(int ch)
2. write(char[] ch)
3. write(String s)
4. flush()
5. close()

+

print(char ch)	println(char ch)
print(int i)	println(int i)
print(double d)	println(double d)
print(boolean b)	println(boolean b)
print(String s)	println(String s)

⇒ Ex

```
class PrintWriterDemo
{
    p s v m main(Sm) throws IOException
    {
```

```
FileWriter fw = new FileWriter
                ("abc.txt");
PrintWriter pw = new PrintWriter(fw);

pw.write(100);
pw.println(100);
pw.println(true);
pw.println('c');
pw.println("durga");
pw.flush();
pw.close();

}
}
```

O/p:

```
d100
true
c
durga
```

abc.txt

Q. What is the difference between write(100) & print(100) ?

Ans: In the case of write(100) the corresponding character 'd' will be added to the file.

- But in the case of print(100), int value 100 will be added to the file directly.

→ **Note:** The most enhanced writer to write character data to the file is "PrintWriter".

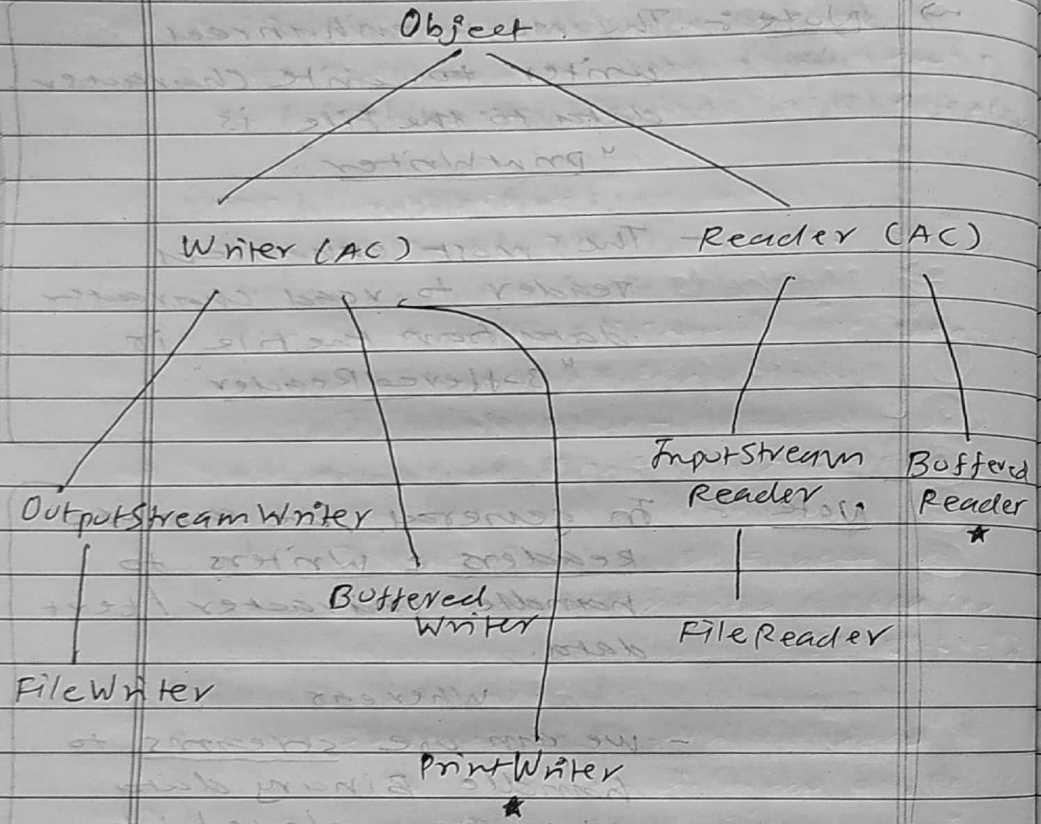
- The most enhanced reader to read character data from the file is "BufferedReader"

Note: In general, we can use Readers & Writers to handle character / text data.

Whereas

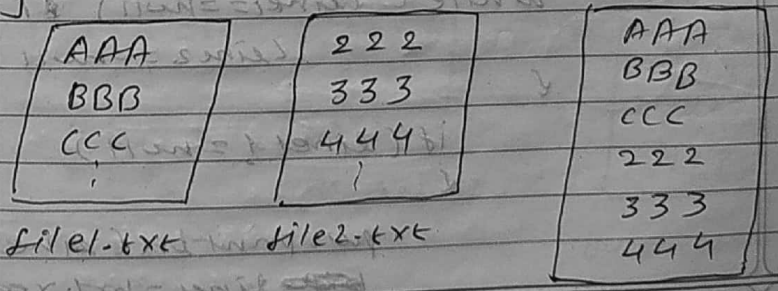
- we can use streams to handle Binary data like Images/pdfs/video.. etc.

- we can use "OutputStream" to write Binary data to the file & "InputStream" to read Binary data from file.



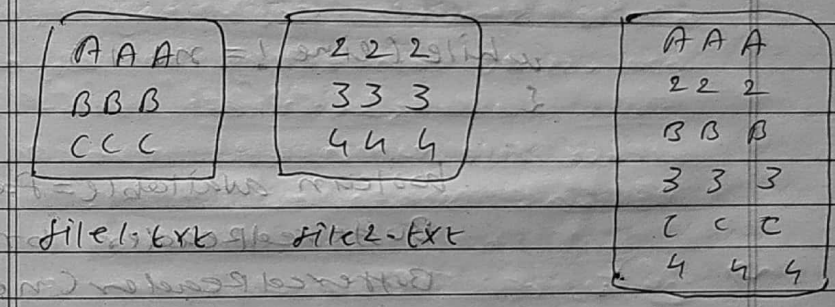
Ex. WAP to merge data from two file into a third file.

```
import java.io.*;
class FileMerger
{
    p s v main (Str) throws IOException
    {
        PrintWriter pw = new
            PrintWriter("file3.txt");
        BufferedReader br = new
            BufferedReader(new
                FileReader("file1.txt"));
        String line = br.readLine();
        while (line != null)
        {
            pw.println(line);
            line = br.readLine();
        }

        br = new BufferedReader(new
                FileReader("file2.txt");
        line = br.readLine();
        while (line != null)
        {
            pw.println(line);
            line = br.readLine();
        }

        pw.flush();
        br.close();
        pw.close();
    }
}
```

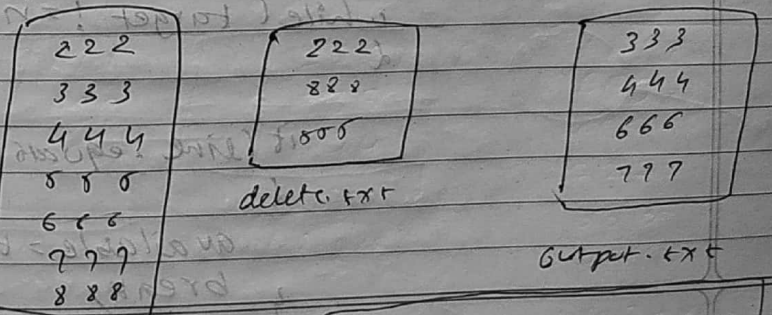


Ex- WAP to perform file merge operation where merging should be done line by line alternatively.

```
import java.io.*;
class FileMerger2
{
    p s v main (Str[]) throws IOException
    {
        PrintWriter pw = new
            PrintWriter ("file3.txt");

        BufferedReader br1 = new
            BufferedReader (new FileReader
                ("file1.txt"));

        BufferedReader br2 = new
            BufferedReader (new FileReader
                ("file2.txt"));

        String line1 = br1.readLine();
        String line2 = br2.readLine();

        while ( (line1 != null) ||
                (line2 != null) )
        {
            if (line1 != null)
            {
                pw.println(line1);
                line1 = br1.readLine();
            }
            if (line2 != null)
            {
                pw.println(line2);
                line2 = br2.readLine();
            }
        }
        pw.flush();
        br1.close();
        br2.close();
        pw.close();
    }
}
```



Ex- WAP to perform file extraction operation.



```
import java.io.*;
class FileExtractor
{
    p s v main(Strn) throws IOException
    {
        PrintWriter pw = new
            PrintWriter("output.txt");

        BufferedReader br1 = new
            BufferedReader(new
                FileReader("input.txt"));

        String line = br1.readLine();

        while(line != null)
        {
            boolean available=false;
            BufferedReader br = new
            BufferedReader(new
                FileReader("delete.txt"));

            String target = br2.readLine();

            while( target != null)
            {

                if(line.equals(target))
                {
                    available = true;
                    break;
                }
                target = br2.readLine();
            }
            if (available == false)
            {
                pw.println(line);
            }
            line = br.readLine();
        }
        pw.flush();
    }
}
```

End of File Flo

Start of Serialization