

05/04/11

Java.io package

3/2

File I/O :-

1. File
2. FileWriter
3. FileReader
4. Buffered Reader
5. Buffered writer
6. PrintWriter.

(1) File :-

* File f = new File("abc.txt");

→ This line won't create any physical file, first it will check is there any file named with abc.txt is available or not.

→ If it is available then f simply pointing to that file.

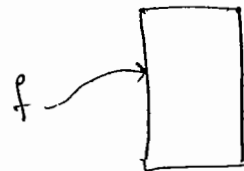
→ If it is not available then f represents just name of the file without creating any physical file.

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

```
S.o.pln(f.exists()); // false
```

```
f.createNewFile();
```

```
S.o.pln(f.exists()); // true
```



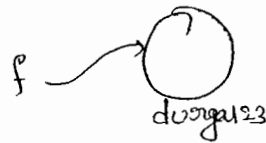
→ A Java file object can represent a directory also.

Ex:-
File f = new File("durga123");

S.o.pln(f.exists()); false

f.mkdir();

S.o.pln(f.exists()); true



Constructors:-

① File f = new File(String name);

→ Create a java file object to represent name of a file or directory.

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

→ To Create a file or directory present in some other sub-directory.

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

Ex:- write code to create a file named with abc.txt in current working directory.

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

f.createNewFile();

② w.c. to create a directory named with xyz in current working directory.

File f = new File("xyz");

f.mkdir();

③ w.c. to Create a directory named with duorgal23 in Current ^{3/3}
Working directory. in That directory create a file named with
abc.txt.

A) `File f = new File("duorgal23");`

`f.mkdir();`

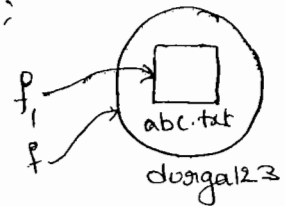
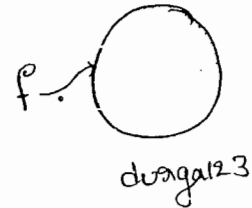
`File f1 = new File("duorgal23", "abc.txt");`

`f1.createNewFile();`

(or)

`File f1 = new File(f, "abc.txt");`

`f1.createNewFile();`



⇒ Important methods of File class :-

① `boolean exists();`

→ returns true if the physical file or directory present

② `boolean createNewFile();`

→ first this method will check wheather the Specified file is already available or not. if it is already available then this method returns false without Creating new file. if it is not already available then this method returns true after creating new file.

③ `boolean mkdir();`

④ `boolean isFile();`

(5) boolean isDirectory();

(6) String[] list();

→ it returns the names of all files & sub-directories present in the specified directory.

(7) boolean delete();

→ To delete a file or directory

(8) long length();

→ returns the no. of characters present in the specified file

Ex:- W.a.p to print the names of all files & sub-directories present in "D:\durga-classes".

```
import java.io.*;
```

```
class Test
```

```
{
```

```
    p.s.v.m(String[] args) throws Exception
```

```
{
```

```
    File f = new File("D:\\durga-classes");
```

```
    String[] s = f.list();
```

```
    for (String si : s)
```

```
    {
```

```
        s.o.println(si);
```

```
    }
```

```
}
```

3/4 (9) FileWriter :-

→ we can use `FileWriter` object to write character data to the file.

Constructors :-

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

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

→ The above 2 Constructors meant for overwriting. If we want to perform append instead of overwriting then we have to use the following Constructors.

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

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

→ If the Specified file is not already available then the above Constructors will Create that file.

Methods of FileWriter :-

① `write(int ch);`

To write a single character to the file.

② `write(char[] ch);`

To write an array of characters to the file.

③ `write(String s);`

To write a String to the file. <http://javabynataraj.blogspot.com> 368 of 401.

(4) flush() :-

→ to give the guarantee that last character of the data also return to the file.

(5) close() :-

Ex:- demo program for the FileWriter.

```
import java.io.*;
```

```
class FileDemo2
```

```
{
```

```
    p.s.v.m(String[] args)
```

```
{
```

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

```
    fw.write(100); // adding a single character
```

```
    fw.write("vagaIn softwareSolutions");
```

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

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

```
    fw.write(ch);
```

```
    fw.write('\n');
```

```
    fw.flush();
```

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

```
}
```

→ appending

o/p:-
100 → d
dvaga
SoftwareSolutions
abc

(3) - FileReader :-

3/5

→ we can use `FileReader` to read character data from the file

Constructors :-

1. `FileReader fr = new FileReader(String name);`
2. `FileReader fr = new FileReader(File f);`

Methods of FileReader :-

(i) `int read();` :-

* It attempts to read next character from the file and return its Unicode value.

* If the next character is not available, then this method returns `-1`.

(ii) `int read(char[] ch);` :-

* It attempts to read enough characters from the file into the char array & returns the no. of characters which are copied from file to the `char[]`.

(iii) `close();`

Ex:- on FileReader

```
import java.io.*;
```

```
class FileReadDemo
```

```
{
```

```
    p.s.v.m (String[] args) throws IOException
```

```
File f = new File("wc.txt")
```

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

```
S.o.pln(fr.read()); //Unicode of first character
```

```
Char[] ch2 = new Char[(int) (f.length())];
```

```
fr.read(ch2); // file data copied to array
```

```
for (Char c: ch2)
```

```
{
```

```
    System.out.print(c);
```

```
}
```

```
S.o.pln("**** -----");
```

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

```
int i = fr.read();
```

```
while (i != -1)
```

```
{
```

```
    S.o.pln((Char) i);
```

```
    i = fr.read();
```

```
}
```

```
}
```


3/6

• Usage of FileWriter & FileReader is not recommended because :-

- 1) While writing data by FileWriter we have to insert line separators manually which is a bigger headache to the programmer.
- 2) By using FileReader we can read data character by character which is not convenient ~~write~~ ^{to the} programmer.
- 3) To resolve these problems SUN people introduced BufferedWriter & BufferedReader classes.

(ii) BufferedWriter :-

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

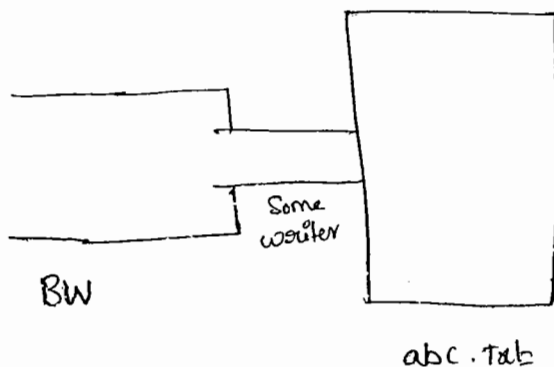
Constructors :-

➔ `BufferedWriter bw = new BufferedWriter(Writer w);`

➔ `BufferedWriter bwc = new BufferedWriter(Writer w, int buffersize);`

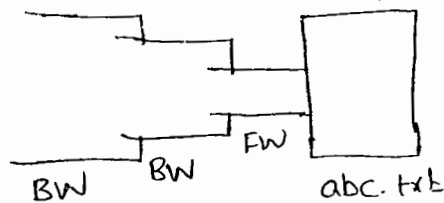
Note!

• BufferedWriter never communicates directly with the file. Compulsorily it should communicate via some writer object only.



Q) which of the following are valid.

- ✗ ① `BufferedWriter bw = new BufferedWriter("abc.txt");`
- ✗ ② `BufferedWriter bw = new BW(new File("abc.txt"));`
- ✓ ③ `BufferedWriter bw = new BufferedWriter(new FileWriter("abc.txt"));`
- ✓ ④ `BW bw = new BW(new BW(new FW(new File("abc.txt"))));`



Important methods of BufferedWriter :-

- ① `write(int ch)`
- ② `write(char[] ch)`
- ③ `write(String s)`
- ④ `flush()`
- ⑤ `close()`
- ⑥ `newLine();` :- to insert a newline character

Q:- When Compared with `FileWriter` which of the following Capability is available as a Separate method in `BufferedWriter`.

- ✗ ① writing data to the file. ✓ ② inserting a line separator.
- ③ flushing the stream.
- ③ closing the stream.

Ex:-

317

```
import java.io.*;
```

```
class BufferedWriterDemo
```

```
{
```

```
    P.S.V.M (String[] args) throws Exception
```

```
{
```

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

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

```
    BufferedWriter bw = new BufferedWriter(fw);
```

```
    bw.write(100);
```

```
    bw.newLine();
```

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

```
    bw.write(ch);
```

```
    bw.newLine();
```

```
    bw.write("dunga");
```

```
    bw.newLine();
```

```
    bw.write("Software Solutions");
```

```
    bw.flush();
```

```
    bw.close();
```

y

%p:-

```
d
abcd
dunga
Software Solutions
```

wc.txt

Note:- When ever we are closing BufferedWriter automatically underlying writers will be closed

```
BW.close(); | fw.close(); | fw.close();
              |              | bw.close();
```

✓

x

<http://javabynataraj.blogspot.com> 374 of 401.

(iv) BufferedReader :-

- * The main Advantage of BufferedReader over FileReader is we can read the data Line by Line instead of Reading Character by character. This approach improves performance of the System by reducing the no. of read operations.

Constructors :-

- (i) `BufferedReader br = new BufferedReader(Reader r);`
- (ii) `BufferedReader (Reader r, int buffer size);`

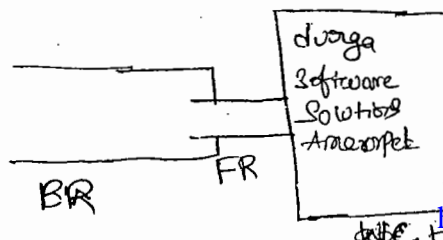
Note:-

- * BufferedReader can't communicate directly with the file Compulsary it should communicate via some reader object.

Important methods :-

- ① `int read();`
- ② `int read(char[] ch);`
- ③ `close();`
- ④ `String readLine();`

- * It attempts to find the next line & if the next line is available then it returns it, otherwise it returns null.



Ex:- import java.io.*;

3/8

class Buffered

{

p.s.v.m(String[] args) throws Exception

{

FileReader fr = new FileReader("wc.txt");

BufferedReader br = new BufferedReader(fr);

String line = br.readLine();

while(line != null)

{

S.o.pln(line);

line = br.readLine();

}

br.close();

}

}

o/p:-

doorga
Software
Solutions

-Amrakesh

Note:-

* when ever you are closing BufferedReader (underlying Readers will be closed).

PrintWriter() :-

* This is the most enhanced writer to write character data to file. By using FileWriter & BufferedWriter we can write only character data but by using PrintWriter we can write any primitive data-types to the file.

Constructors:-

- ① `PrintWriter pw = new PrintWriter(String name);`
- ② `PrintWriter pw = new PrintWriter(File f);`
- ③ `PrintWriter pw = new PrintWriter(Writer w);`

Methods:-

① <code>write(int ch)</code>	<code>Print(char ch)</code>	<code>println(char ch)</code>
② <code>write(char[] ch)</code>	<code>print(int i)</code>	<code>println(int i)</code>
③ <code>write(String s)</code>	<code>print(long l)</code>	<code>println(long l)</code>
④ <code>flush()</code>	<code>print(double d)</code>	<code>println(double d)</code>
⑤ <code>close()</code>	<code>print(String s)</code>	<code>println(String s)</code>
	<code>print(char[] ch)</code>	<code>println(char[] ch)</code>
	⋮	⋮

ex:-

```
import java.io.*;
```

```
class PrintWriterDemo1
```

```
{
```

```
    p.s.v.m(String[] args) throws IOException
```

```
{
```

FileWriter fw = new FileWriter("wc.txt");

PrintWriter pw = new PrintWriter(fw);

pw.write(100); // 100

pw.println(100); // 100

pw.println(true); // true

pw.println('c'); // c

pw.println("durga"); // durga

pw.flush();

pw.close();

o/p:-

```
100
true
c
durga
```

wc.txt

Q:- what is the diff. b/w the following

(a) pw.write(100);

(b) pw.println(100);

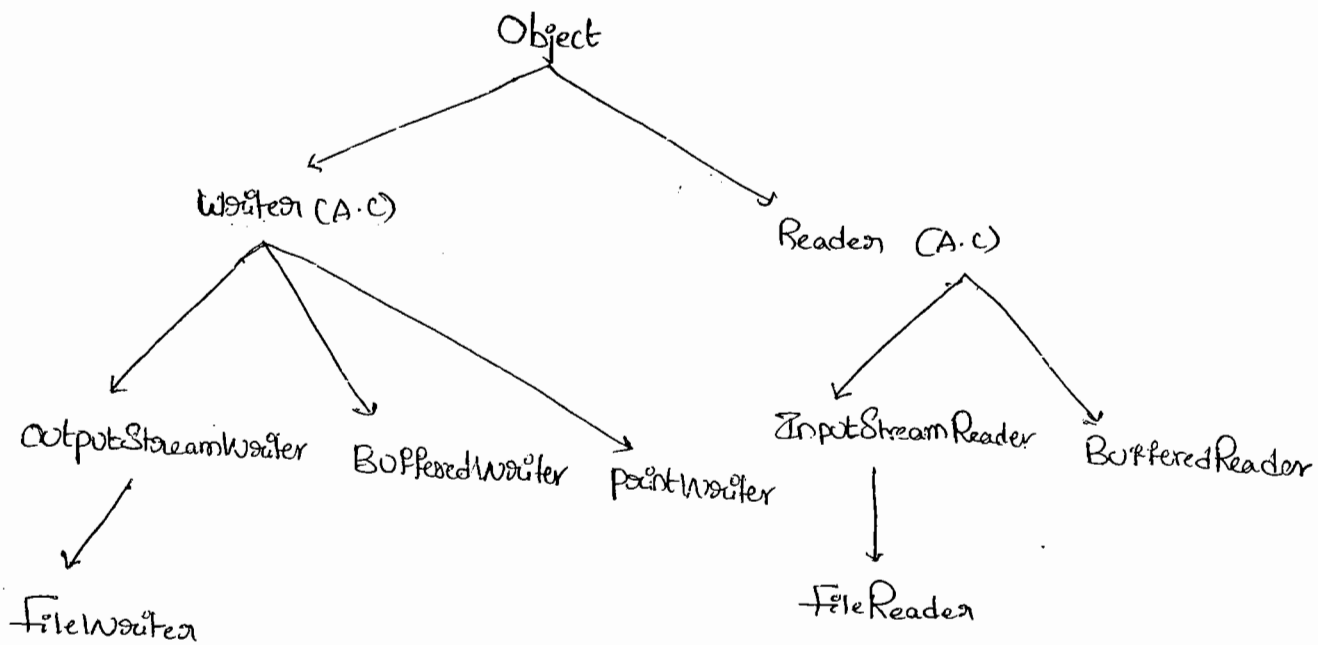
Ans:-

pw.write(100);

→ In this case the corresponding character 1 will be added to the file

pw.println(100)

→ In this case the corresponding int value 100 directly will be added to the file



Note:-

- * Readers & writers meant for handling character data (any primitive data type) +
- * To handle Binary data (like images, movie files, jar files) we should go for Streams.
- * We can use InputStream to Read Binary data & OutputStream to write a Binary data.
- * We can use ObjectInputStream & ObjectOutputStream to read & write Objects to a file respectively (Serialization).
- * The most Enhanced writer to write character data is PrintWriter whereas the most Enhanced Reader to read character data is BufferedReader.

* W.a.p to merge data from two files into a 3rd file. 321

file3.txt = file1.txt + file2.txt

```
import java.io.*;
```

```
class FileMerger
```

```
{
```

```
    P.S. v.m (String[] args) throws Exception
```

```
{
```

```
    PrintWriter pw = new PrintWriter("output.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();
```

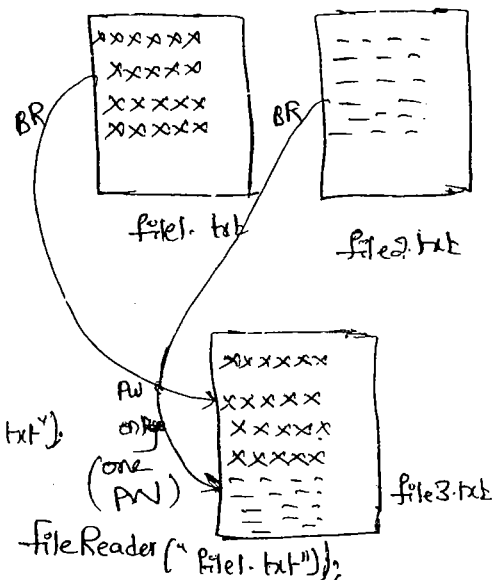
```
}
```

```
    pw.flush();
```

```
    br.close();
```

```
    pw.close();
```

```
}
```



Ex2:-

w.a.p to merge data from 2 files into a 3rd file but merging should be done line by line alternatively.

```
import java.io.*;
```

```
class FileMerger2
```

```
{  
    p.s.v.m(String[] args) throws IOException
```

```
{
```

```
    PrintWriter pw = new PrintWriter("output.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();
```

```
    pw.close();
```

```
    br1.close();  
    br2.close();
```

```
}
```

3:-

w.a.p to merge data from two files into a 3rd file but merging should be done para by para. assume that there is a Blank line B/w Every 2 paras?

4:-

w.a.p to delete duplicates from the given i/p file?

```
import java.io.*;
```

```
class DuplicateElimination
```

```
{
    p.s.v.m(String[] args) throws Exception
```

```
{
```

```
    BufferedReader br1 = new BufferedReader(new FileReader("i/p.txt"));
```

```
    PrintWriter pw = new PrintWriter("o/p.txt");
```

```
    String line = br1.readLine();
```

```
    while(line != null)
```

```
    {
```

```
        boolean available = false;
```

```
        BR br2 = new BR(new FR("o/p.txt"));
```

```
        String target = br2.readLine();
```

```
        while(target != null)
```

```
        {
```

```
            if(line.equals(target))
```

```
            {
                available = true;
```

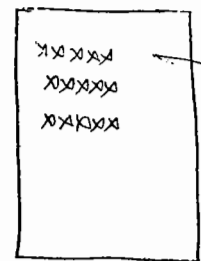
```
                break;
```

```
            }
```

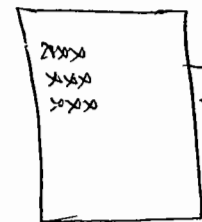
```
            target = br2.readLine();
```

```
        }
```

```
    }
```



input.txt



output.txt

```
if (available == false)
```

```
{  
    pw.println(line);
```

```
    pw.flush();
```

```
}
```

```
line = br.readLine();
```

```
{
```

```
    pw.flush();
```

```
    br.close();
```

```
    br2.close();
```

```
    pw.close();
```

```
}
```

```
}
```

⑤ W.a.p to perform File Extraction (result.txt = total.txt - delete.txt)

```
import java.io.*;
```

```
class FileExtractor
```

```
{
```

```
    p.s.v. m (String[] args) throws Exceptions
```

```
{
```

```
    BufferedReader br1 = new BufferedReader(new FileReader  
                                                ("mobile.txt"));
```

```
    PrintWriter pw = new PrintWriter("output.txt");
```

```
    String line = br1.readLine();
```

```
    while (line != null)
```

```
{
```

```
        boolean available = false;
```

```
        BR br2 = new BR(new BR("delete.txt"));
```

String target = br2.readLine();

while (target != null)

durgajobs.com

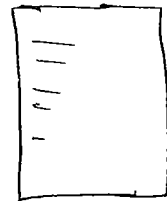
```

↓
if (line.equals(target))
↓
    available = true;
    break;
}
target = br2.readLine();
}
if (available == false)
↓
    pw.println(line);
}
line = br1.readLine();
}
pw.flush();
br1.close();
br2.close();
pw.close();
}
}

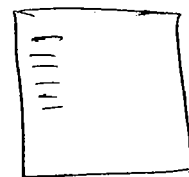
```



total.txt



delete.txt



result.txt

heretic = a person who holds controversial beliefs, especially contrary to religion, ^{opposite} privileged argument.
Profession etc.