11. java.io package

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

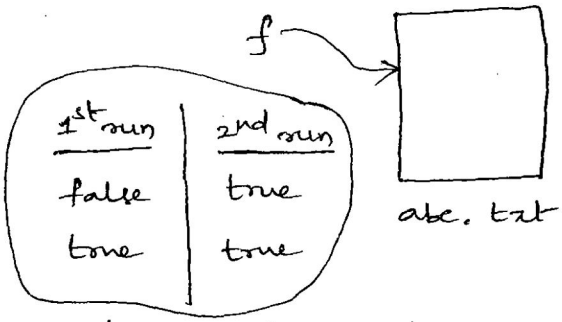
1. File :—

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

→ This line won't create any physical file.

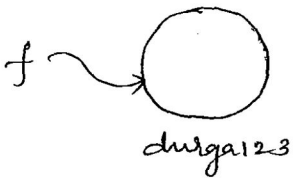
→ First it will check is there any physical file named with abc.txt is available or not, if it is available then f simply pointing to that file. If it is not available then it won't create any physical file just we are creating Java File object to represent the name abc.txt.

Ex:
```
File f = new File("abc.txt");
S.o.p(f.exists()); ⇒ o/p : false
f.createNewFile();
S.o.p(f.exists()); ⇒ o/p : true
```



→ Java File object can be used to represent directories also.

Ex:
```
File f = new File("durga123");
S.o.p(f.exists()); ⇒ o/p : false
f.mkdir();
S.o.p(f.exists()); ⇒ o/p : true
```



Note:— In UNIX, everything is a file. Java File IIo concept is implemented based on UNIX OS. Hence we can use Java File object

to represent both files & directories.

File class Constructors :—

1. File f = new File (String name);

Creates a Java File object to represent name of the file or directory in current working directory.

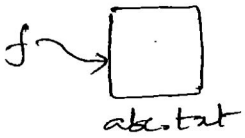
2. File f = new File (String subdir, String name);

Creates a Java File object to represent name of the file or directory in the specified subdirectory.

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

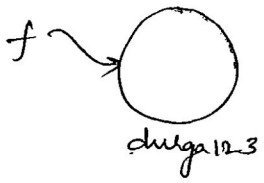
Ex①: Write code to create a file named with abc.txt in current working directory:—

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



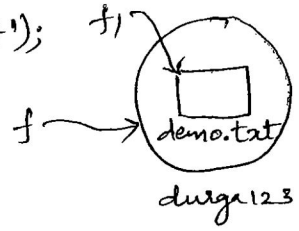
Ex②: Write code to create a directory named with durga123 in current working directory:—

```
File f = new File("durga123");
f.mkdir();
```

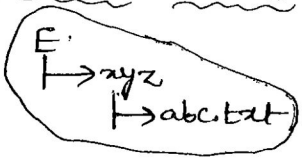


Ex③: Write code to create a directory named with durga123 in CWD and create a file named with demo.txt in that directory:—

```
// File f1 = new File("durga123", "demo.txt");
File f1 = new File(f, "demo.txt");
f1.createNewFile();
```



Ex④: write code to create a file named with abc.txt inside E:\\xyz folder.




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

→ Assume that E:\\xyz is already available in our system.

Important Methods of File class:—

1. boolean exists()

This method checks whether the physical file or directory present or not. If it is available returns ~~false~~ otherwise returns false.

2. boolean createNewFile()

First this method will check is there any physical file is already available or not with the required name. If it is already available returns false without creating any file. If it is not already available creates a new file and returns true.

3. boolean mkdir()

4. boolean isFile()

returns true if the File object represents a physical file.

5. boolean isDirectory()

6. String[] list()

It returns the names of all files and subdirectories present in specified sub directory.

7. long length()

It returns the no. of characters present in the file.

8. boolean delete()

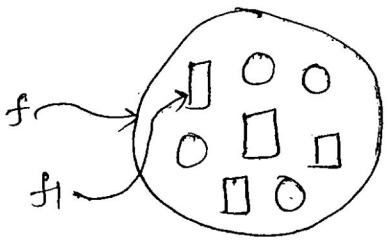
To delete a file or directory.

Ex: Write code to display the names of all files & subdirectories present in D:\durga_classes.

```
import java.io.*;
class Test
{
  p s v m(_) throws Exception
  {
    int count=0;
    File f=new File("D:\\durga_classes");
    String[] s=f.list();
    for(String s1 : s)
    {
      count++;
    } S.o.p(s1);
    S.o.p("The total number :"+count);
  }
}
```

To represent only file names :—

```
File f=new File("D:\\OCJP");
String[] s=f.list();
for(String s1 : s)
{
  File f1=new File(f, s1);
  if(f1.isFile())
  {
    S.o.p(s1);
  }
}
```



D:\\OCJP

To represent only directory names :—

→ In the above code replace isFile() method with isDirectory() method.

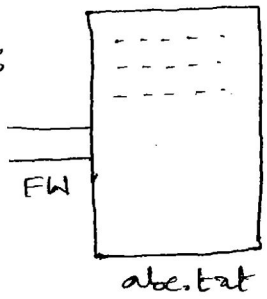
2) FileWriter :—

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

Constructors :—

① FileWriter fw=new FileWriter(String fname);

② FileWriter fw=new FileWriter(File f);



→ The above two constructors meant for overriding. Instead of overriding if we want append operation we have to use the following two 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.

Important 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)

4. flush()

To give guarantee that entire data including last character will be added to the file.

5. close().

Ex:
```
import java.io.*;
class FileWriterDemo
{
  P s v m(_) throws IOException
  {
    FW fw=new FW("abc.txt", true);
    fw.write(100);
    fw.write("durga\n softwareSolutions");
    fw.write('\n');
    char[] ch={'a', 'b', 'c'};
    fw.write(ch);
    fw.write('\n');
    fw.flush();
    fw.close();
  }
}
```

abc.txt

durga
SoftwareSolutions
abc

→ FileWriter is outdated concept.

3) FileReader class :—

→ We can use FileReader to read character data from the file.

Constructors :—

① FileReader fr=new FileReader(String filename);

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 the next character is not available then this method returns -1.

→ As this method returns unicode value at the time of retrieval we have to perform type casting.

Ex: S.o.p(fr.read()); ⇒ o/p : 100

S.o.p((char)fr.read); ⇒ o/p : d

② int read(char[] ch);

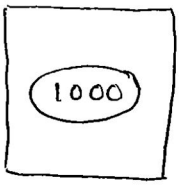
It attempts to read enough characters from the file into char array and return no. of characters copied from the file into array.

Ex: ① char[] ch=new char[10];

S.o.p(fr.read(ch)); ⇒ o/p : 10

② char[] ch=new char[10000];

S.o.p(fr.read(ch)); ⇒ o/p : 10000



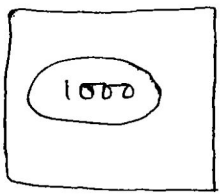
Good Programming Practice:—

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

char[] ch=new char[(int)f.length()];

fr.read(ch);

③ void close();



// Program to read the character data from the file:-

```
import java.io.*;
class FileReaderDemo
{
  p s v m(-) throws IOException.
  {
    File f=new File("abc.txt");
    FileReader fr=new FileReader(f);
    char[] ch=new char[(int)f.length()];
    fr.read(ch);
    for(char ch1:ch)
        S.o.p(ch1);
    S.o.p("*********");
    FileReader fr1=new FileReader("abc.txt");
    int i=fr1.read();
    while(i!=-1)
    {
      S.o.print((char)i);
      i=fr1.read();
    }
  }
}
```

Usage of FileWriter & FileReader is not not recommended becoz,

1. While writing data by FileWriter we have to insert line separator manually which is varied from system to system. It is difficult to the programmer.
2. By using FileReader we can read data character by character which is not convenient to the programmer.

→ To overcome above problems we should go for BufferedWriter and BufferedReader.

4. BufferedWriter :—

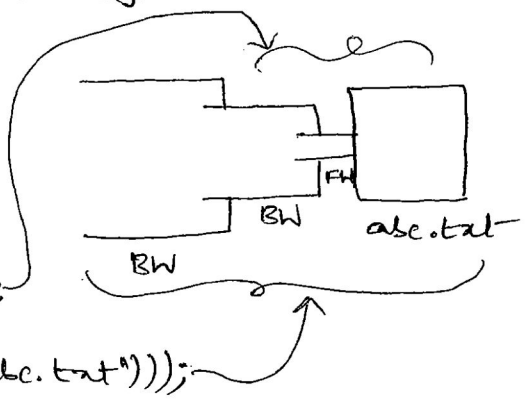
→ We can use BufferedWriter to write character data to the file.

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 should communicate via some Writer object.

*** Q: Which of the following are valid?



✗ ① BW bw = new BW("abc.txt");

✗ ② BW bw = new BW(new File("abc.txt"));

✓ ③ BW bw = new BW(new FW("abc.txt"));

✓ ④ BW bw = new BW(new BW(new FW("abc.txt")));

Methods :—

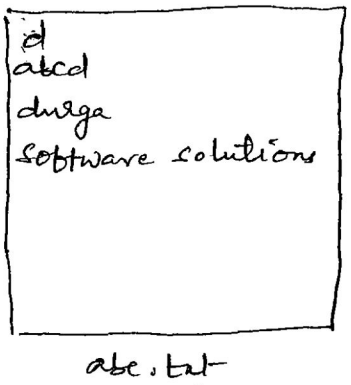
1. write(int ch)
2. write(char[] ch)
3. write(String s)
4. flush()
5. close()
* 6. newLine() → To insert a line separator

*** Q ⇒ When compared with FileWriter which of the following extra capability is available in method form of BufferedWriter?

① writing data to the file
② close the file
③ flushing the file
④ inserting a new line character ✓

Ex:
```
import java.io.*;
class BufferedWriterDemo
{
    p s v m(-) throws IOE
    {
        FW fw=new FW("abc.txt");
        BW bw=new BW(fw);
        bw.write(100);
        bw.newLine();
        char[] ch1={'a','b','c','d'};
        bw.write(ch1);
        bw.newLine();
        bw.write("durga");
        bw.newLine();
        bw.write("Software solutions");
        bw.flush();
    } bw.close();
}
```

O/P:



abc.txt

Note:- Whenever we are closing BufferedWriter automatically underlying Writers will be closed & we are not required to close explicitly.

ex: `bw.close();` ✓ | `fw.close();` ✗ | `bw.close();` `fw.close();` ✗

5. BufferedReader:—

→ We can use BufferedReader to read character data from the file.

→ The main advantage of BufferedReader over FileReader is we can read data line by line instead of character by character which is more convenient to the programmer.

Constructors:—

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

*** **Note:-** BufferedReader can't communicate with a file, it can communicate by a some Reader object.

Methods:-

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

*4. String readLine()

It attempts to read next line from the file & returns it. If the next line is not available then we will get null.

Ex:

```
import java.io.*;
class BufferedReaderDemo
{
   p s v m(-) throws Exception
   {
      FR fr=new FR("abc.txt");
      BR br=new BR(fr);
      String line=br.readLine();
      while(line!=null)
      {
         S.o.p(line);        ------> o/p : durga
         line=br.readLine();             software
      }                                  solutions
      br.close();
   }
}
```

abc.txt:

```
durga
software
solutions
```

Note:- Whenever we are closing BufferedReader automatically underlying Readers will be closed and we are not required to close explicitly.

6. PrintWriter:-

→ The most enhanced writer to write character data to the file is PrintWriter.

→ The main advantage of PrintWriter 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 either directly to the file or via Writer object.

Methods :—

1. write(int ch);
2. write(char[] ch);
3. write(String s);
4. flush();
5. close();
6. print(char ch);
7. print(int i);
8. print(double d);
9. print(String s);
10. print(boolean b);
11. println(char ch);
12. println(int i);
13. println(double d);
14. println(String s);
15. println(boolean b);

Ex:
```
import java.io.*;
class PrintWriterDemo
{
  P s v m(_)throws IOE
  {
    FW fw=new FW("abc.txt");
    PW pw=new PW(fw);
    pw.write(100);
    pw.println(100);
    pw.println(true);
    pw.println('c');
    pw.println("durga");
    pw.flush();
  } pw.close();
```

O/P:

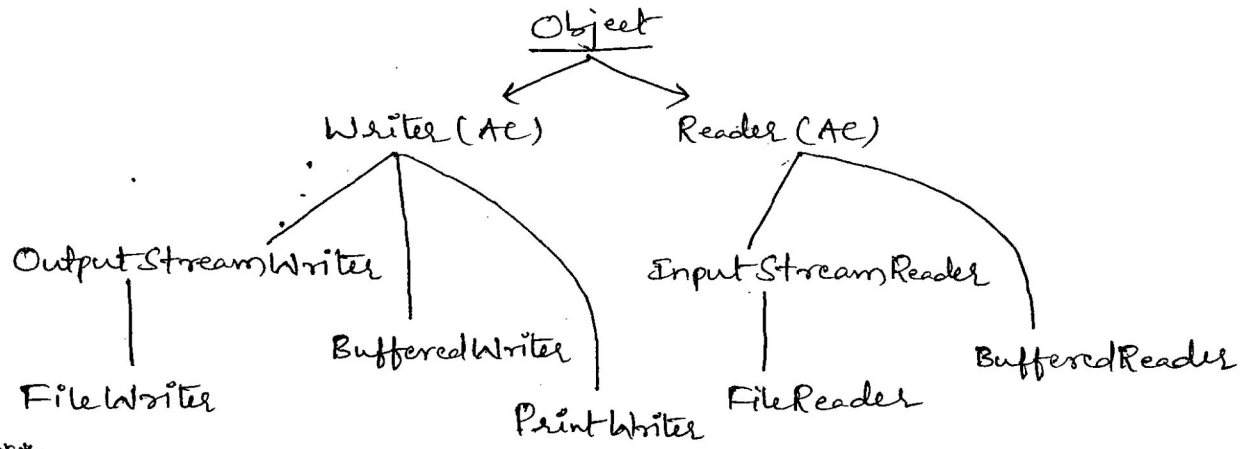
abc.txt

Q: What is the difference blw pw.write(100) and pw.print(100)?

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

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

Note:- The most enhanced Reader to read character data from the file is BufferedReader where as the most enhanced Writer to write character data to the file is PrintWriter.



*** Note:- In general we can use Readers & Writers to handle text data.

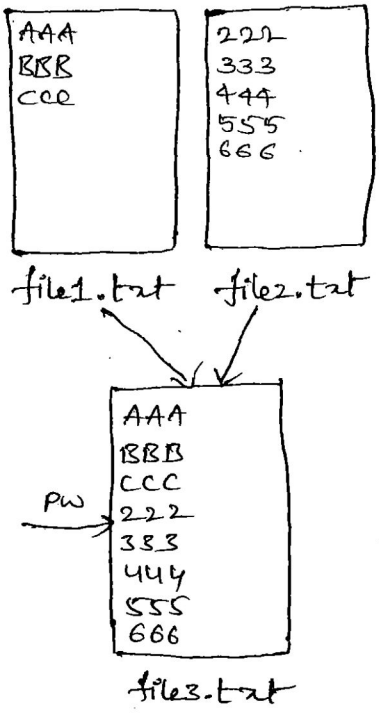
But we can use Streams to handle binary data (like images, video files, audio files, jar files etc.).

→ We can use OutputStream to write binary data to the file where as InputStream to read binary data from the file.

// Write a program to merge data from two files into a third file.

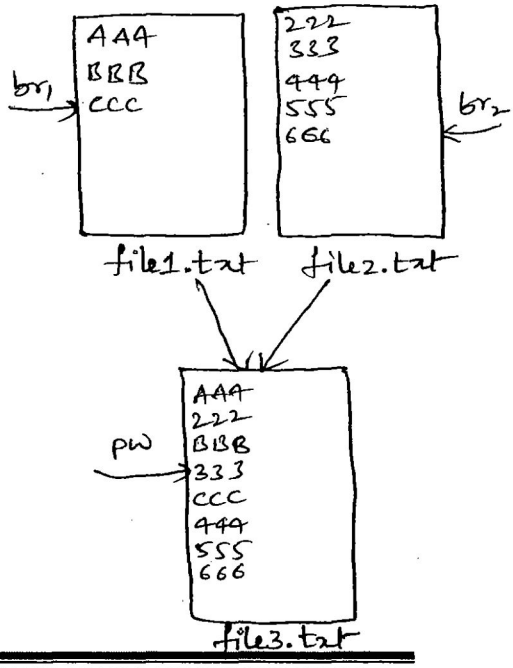
```
import java.io.*;
class FileMerger
{
  p s v m(-) throws Exception
  {
    PW pw=new PW("file3.txt");
```

```
BR br=new BR(new FR("file1.txt"));
String line=br.readLine();
while(line!=null)
{
  pw.println(line);
  line=br.readLine();
}
br=new BR(new FR("file2.txt"));
line=br.readLine();
while(line!=null)
{
  pw.println(line);
  line=br.readLine();
}
pw.flush();
br.close();
pw.close();
}}
```



// write a program to merge data from 2 files into a third file where merging should be done line by line alternatively.

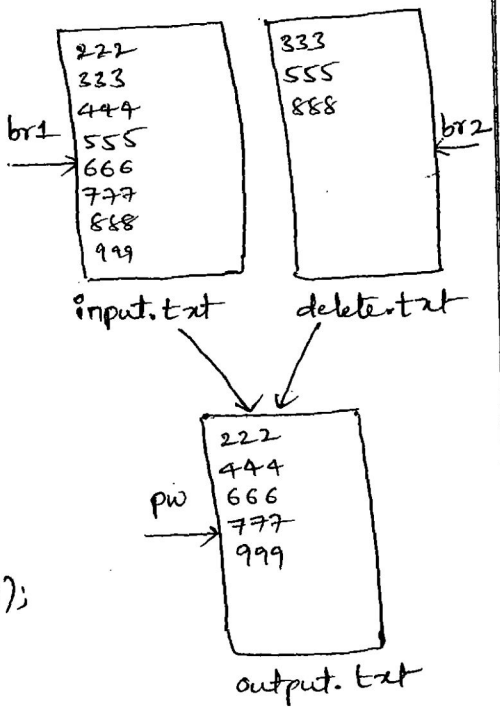
```
import java.io.*;
class FileMerge1
{
  P s v m(-) throws Exception
  {
    PW pw=new PW("file3.txt");
    BR br1=new BR(new FR("file1.txt"));
    BR br2=new BR(new FR("file2.txt"));
    String line1=br1.readLine();
    String line2=br2.readLine();
    while((line1!=null)||(line2!=null))
    {
```



```
        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();
      }
    }
```

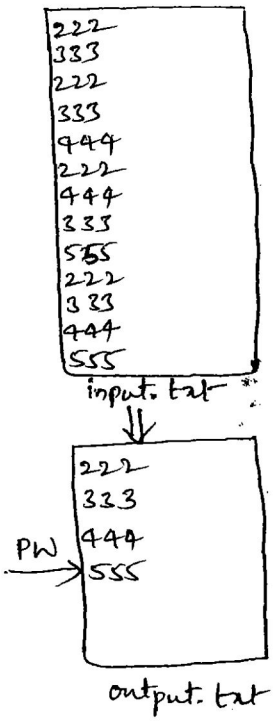
// Write a program to perform file extraction operation.

```
import java.io.*;
class FileExtractor
{
  p s v m(-) throws Exception
  {
    PW pw=new PW("output.txt");
    BR br1=new BR(new FR("input.txt"));
    String line=br1.readLine();
    while(line!=null)
    {
      boolean available=false;
      BR br2=new BR(new FR("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=br1.readLine();
      }
      pw.flush();
    }
  }
```

//Write a Java program to remove duplicates from the given input file.




```
import java.io.*;
class DuplicateEliminator
{
  P s v m(-)throws Exception
  {
    PW pw=new PW("output.txt");
    BR br1=new BR(new FR("input.txt"));
    String line=br1.readLine();
    while (line!=null)
    {
      boolean available=false;
      BR br2=new BR(new FR("output.txt"));
      String target=br2.readLine();
      while (target!=null)
      {
        if(line.equals(target))
        {
          available=true;
        } break;
```

```
            target=br2.readLine();
        }
        if(available==false)
        {
            pw.println(line);
            pw.flush();
        }
        line=br1.readLine();
    }
  }
}
```