File System In Java:

“File f1 = new File(“abc.txt”);” the 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 “f1” simple refers that file. If it is not available then we are just creating java file object to represent the name “abc.txt” using the statement “f1.createNewFile();”.

```
import java.io.*;
class test
{
public static void main(String args[])
{
File f1 = new File("abc.text");
System.out.println(f1.exists());
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
false
```

```
import java.io.*;
class test
{
public static void main(String args[]) throws Exception
{
File f1 = new File("abc.txt");
System.out.println(f1.exists());
f1.createNewFile(); //create a new file named "abc.txt"
System.out.println(f1.exists());
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
false
true
```

We can use java file object to represent directory also. “File f2 = new File(“ABC”);” the statement is for reference to the folder named as “ABC”. “f2.mkdir();” the statement is used to create a folder/directory named as “ABC”.

```
import java.io.*;
class test
{
public static void main(String args[]) throws Exception
{
File f2 = new File("ABC");
System.out.println(f2.exists());
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
false
```

```
import java.io.*;
class test
{
public static void main(String args[]) throws Exception
{
File f2 = new File("ABC");
System.out.println(f2.exists());

f2.mkdir(); //create a directory named as "ABC";
System.out.println(f2.exists());
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
false
true
```

In UNIX OS everything is treated as a file. Java file io concept is implemented based on UNIX operating system. hence java file object can be used to represent both files and directories.

File Class Constructor:

“File f = new File(String name);” Creates java file object to represent name of the file or directory in current working directory.

“File ff = new File(String subdirname, String name);” Creates a java file object to represent name of the file or directory present in specified sub directory.

“File fff = new File(File subdir, String name);” if a directory is already referenced by any other object.

EXAMPLE 01:

WRITE CODE TO CREATE A FILE NAMED WITH ABC.txt IN CURRENT WORKING DIRECTORY.

File f = new File(“abc.txt”); f.createNewFile();

EXAMPLE 02:

WRITE CODE TO CREATE A DIRECTORY NAMED WITH “ATISH” IN CURRENT WORKING DIRECTORY AND CREATE A FILE NAMED WITH “demo.txt” IN THAT DIRECTORY.

File ff = new File(“ATISH”); ff.mkdir();

File ff1 = new File(“ATISH”, “demo.txt”);

File ff2 = new File(ff,”demo.txt”); ff2.createNewFile();

EXAMPLE 03:

WRITE CODE TO CREATE A FILE NAMED WITH “ABC.txt” in E: -> xyz folder.

File f3 = new File(“E: \\xyz”, “ABC.txt”);

F3.createNewFile();

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

Different File Methods:

boolean f.exists(); -> whether the file/directory is available or not. Returns true if the specified file or directory available.

boolean f.createNewFile(); -> create a new file. First this method will check whether the specified file is already available or not. If it is already available then this method returns false without creating any physical file. If the file is not already available then this method will creates a new file and returns true.

boolean f.mkdir(); -> create a new folder. First this method will check whether the specified folder is already available or not. If it is already available then this method returns false without creating any physical folder. If the folder is not already available then this method will creates a new folder and returns true.

boolean f.isFile(); -> check whether the thing is file or not. It returns true. If the specified file object pointing to physical file.

boolean f.isDirectory(); -> check whether the thing is directory or not. It returns true. If the specified folder object pointing to physical folder.

f.list(); -> also known as String[] list(); -> how many file and directory are there in a specific directory. This method returns the names of all files and subdirectory present in specified directory.

f.length(); -> also known as long length(); -> how many characters are there inside a file. Returns number of character present in specified file.

boolean f.delete(); -> delete file or the directory. To delete specified file or directory.

```
Int count = 0;
File f = new File("c:\\directory name");
String[] s = f.list();
for(String S1: s)      {
        count++;
        System.out.println(S1);    }
System.out.println("total number : "+counter);
```

Display File Names & Directory Names Using For Each Loop:

```
File fs = new File(f,s1);
If(fs.isFile())          {
Count++
SOP(s1);      }
SOP("TOTAL NUMBER OF FILES : "+count);
File fs = new File(f,s1);
If(fs.isDirectory())          {
Count++
SOP(s1);      }
SOP("TOTAL NUMBER OF FILES : "+count);
```

File Writer:

We can use file writer to write character data to the file.

FileWriter fw = new FileWriter(String filename);

FileWriter fw = new FileWriter(File fw);

The above two statements are meant for overriding of existing data. Instead of overriding if you want to append operation then we have to create filewriter by using the following constructors.

FileWriter fw = new FileWriter(String filename, boolean(true));

FileWriter fw = new FileWriter(file f, boolean append);

If the specified file is not already available then all the above constructors will create that file.

Write(int ch) -> this method is for write a single character. Write(char[] ch) -> this method is for write an array of character. Write(String s) -> this method is for write a string to the file. flush() this method is used for to give guaranty that total data including last character will be return to the file. close() this method is used for to close the writer.

In program filewriter can perform overriding of existing data. Instead of overriding if we want append operation then we have to create file writer object as follows.

THE MAIN PROBLEM WITH FILE WRITER IS WE HAVE TO INSERT LINE SEPARATOR "\n" MANUALLY WHICH IS VARIED FROM SYSTEM TO SYSTEM. IT IS DIFFICULTY TO THE PROGRAMMER. WE CAN SOLVE THIS PROBLEM BY USING "BUFFERED WRITER" & "PRINT WRITER CLASSES".

File Reader:

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

Constructors:

FileReader fr = new FileReader(String filename);

FileReader fr = new FileReader(file, f);

int read(); the statement attempts to read next character from the file and returns it's Unicode value. if the next character is not available then this method returns "-1". As this method returns Unicode value(int value), at the time of printing we have to perform typecasting.

```
import java.io.*;
class test
{
public static void main(String args[]) throws Exception
{
FileReader fr = new FileReader("abc.txt");
int i = fr.read();
while(i != -1)
{
System.out.println((char)i);
i = fr.read();
}

}
}
```

Int read(char[] ch); it attempts to read enough characters from the file into char array and returns number of characters copied from the file. void close(); to close the reading operation.

```
import java.io.*;
class test
{
public static void main(String args[]) throws Exception
{
File f = new File("abc.txt");
char[] ch = new char[(int)f.length()];
        //if the length is with in int range
FileReader fr = new FileReader(f);
fr.read(ch);
for(char ch1: ch)
{
SOP(ch1);
}

}
}
```

BY USING FILEREADER WE CAN READ DATA CHARACTER BY CHARACTER WHICH IS NOT CONVIENT TO THE PROGRAMMER.

USAGE OF FILE WRITER AND FILE READER IS NOT RECOMMENDED BECAUSE WHILE WRITING DATA BY FILEWRITER WE HAVE TO INSERT LINE SEPARATOR "\N" MANUALLY WHICH IS VARRIED FROM SYSTEM TO SYSTEM. IT IS DIFFICULT TO THE PROGRAMMER. BY USING FILEREADER WE CAN READ DATA CHARACTER BY CHARACTER, WHICH IS NOT CONVINENT TO THE

PROGRAMMER.TO OVERCOME THESE PROBLEM WE SHOULD GO FOR BUFFEREDWRITER & BUFFEREDREADER.

BufferedWriter:

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

CONSTRUCTORS:

BufferedWriter BW = new BufferedWriter(any writer w); //w is reference to any FileWriter

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

BufferedWriter can't communicate directly with the file. it can communicate via some writer object.

Q. WHICH OF THE FOLLOWING ARE VALID?

1. BufferedWriter bw = new BufferedWriter("abc.txt"); INVALID
2. BufferedWriter bw = new BufferedWriter(new file("abc.txt")); INVALID
3. BufferedWriter bw = new BufferedWriter(new fw("abc.txt")); VALID
4. BufferedWriter bw = new BufferedWriter(new BW(new fw("abc.txt"))); VALID

BufferedWriter Methods:

1. write(int ch); to write a single character -> bw.write(100); 100 = d in char
2. write(char[] ch); to write array of character
3. write(String s); to write string to file
4. flush();
5. close();
6. newLine(); to insert a new line separator. *bw.newLine();*

Q. WHEN COMPARED WITH FILEWRITER WHICH OF THE FOLLOWING CAPABILITY AVAILABLE EXTRA IN METHOD FORM IN BUFFEREDWRITER?

1. Writing data to the file
2. Close the file
3. flushing the file
4. *inserting a new line character -> answer.*

WHENEVER WE ARE CLOSING BUFFEREDWRITER AUTOMATICALLY INTERNAL FILEWRITER WILL BE CLOSED. WE ARE NOT REQUIRED TO CLOSE EXPLICITLY.

BufferedReader:

We can used bufferedreader to read character data from the file. The main advantage of bufferedreader when compared with filereader is we can read data line by line in addition to character by character.

Constructor:

1. BufferedReader br = new BufferedReader(reader r);

2. BufferedReader br = new BufferedReader(reader r, int buffered size);

BUFFEREDREADER CAN'T COMMUNICATE DIRECTLY WITH THE FILE AND IT CAN COMMUNICATE VIA SOME READER OBJECT.

BufferedReader Methods():

1. int read();

2. int read(char[] ch);

3. void close();

4. String readLine(); it attempts to read next line from the file and returns it. If the next line is not available then this method returns null.

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

BufferedReader br = new BufferedReader(fr);

String line  = br.readLine();

While(line != null)

{

System.out.println(line);

Line = br.readLine();

} br.close();
```

WHENEVER WE ARE CLOSING BUFFEREDREADER AUTOMATICALLY UNDERLYING FILE READER WILL BE CLOSED AND WE ARE NOT REQUIRED TO CLOSE EXPLICITLY. THE MOST ENHANCED READER TO READ CHARACTER DATA FROM THE FILE IS BUFFEREDREADER.

PrintWriter():

It is the most enhanced writer to write character data to the file. the main advantage of PrintWriter over FileWriter and BufferedWriter is we can write any type of primitive data directly to the file.

Constructors:

1. PrintWriter pw = new PrintWriter (String file name);

2. PrintWriter pw = new PrintWriter(String f);

3. PrintWriter pw = new PrintWriter(writer w);

PrintWriter can communicate directly with the file and can communicate via some writer object also.

PrintWriter Methods():

1. write(int ch); -> single char value
2. write(String s);
3. write(char[] c);
4. flush();
5. close();
6. Print(char ch, int I, boolean b, double d, String s);
7. Println(char ch, int I, boolean b, double d, String s);

-> pw. Write(); pw.println();

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

THE MOST ENHANCED WRITER TO WRITE CHARACTER DATA TO THE FILE IS PRINTWRITER, WHERE AS THE MOST ENHANCED READER TO READ CHARACTER DATA FROM THE FILE IS BUFFEREDREADER.

Character data : reader concept, writer concept

Binary data(pdf, images, video, audio): OutputStream is for write , InputStream is for read binary data from file.

IN GENERAL WE CAN USE READERS AND WRITERS TO HANDLE CHARCATER DATA(TEXT DATA). WHERE AS WE CAN USE STREAM TO HANDLE BINARY DATA(IMAGES, PDF, AUDIO, VIDEO, ETC). WE CAN USE OUTPUTSTREAM TO WRITE BINARY DATA TO THE FILE, INPUTSTREAM TO READ BINARY DATA FROM THE FILE.

Q. Write a java program to merge data from two files into a third file?

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
public class test
{
	public static void main(String args[])
	{
		String inputfile1 = "file1.txt";
		String inputfile2 = "file2.txt";
		String inputfile3 = "file3.txt";

		mergefile(inputfile1, inputfile2, inputfile3);
	}
	public static void mergefile(String f1, String f2, String f3)
	{
		try(BufferedReader r1 = new BufferedReader(new FileReader(f1));
			BufferedReader r2 = new BufferedReader(new FileReader(f2));
			BufferedWriter w = new BufferedWriter(new FileWriter(f3)))
		{
			String line;

			//merge data from the first line
			while((line = r1.readLine()) != null)
			{
				w.write(line);
				w.newLine();
			}

			//merge data from the second file
			while((line = r2.readLine()) != null)
			{
				w.write(line);
				w.newLine();
			}
			System.out.println("Files Merged Successfully!");
		}catch(IOException e){
			e.printStackTrace();
		}
	}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
Files Merged Successfully!

C:\Users\Atish kumar sahu\desktop>
```

file3.txt

File Edit View

This Is File 01
Atish Kumar Sahu
This Is File 02
Lipun Kumar Sahu

file2.txt

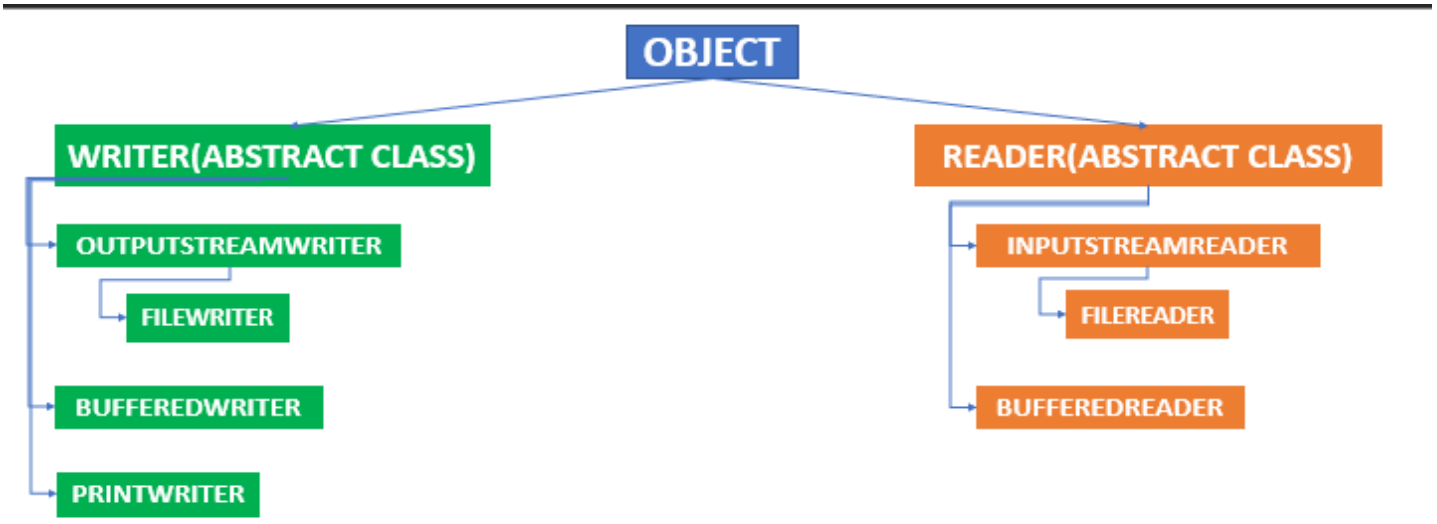
File Edit View

This Is File 02
Lipun Kumar Sahu

file1.txt

File Edit View

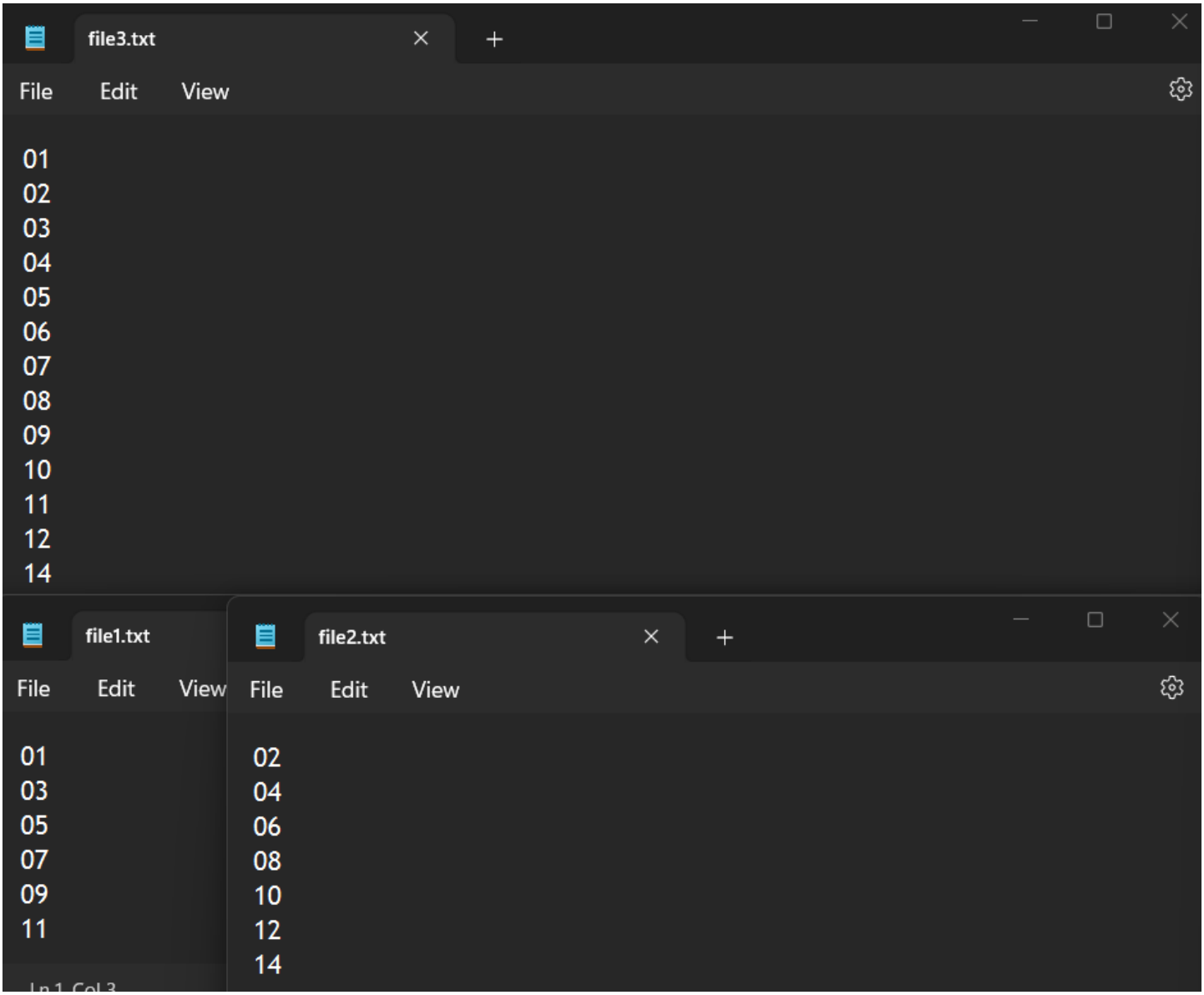
This Is File 01
Atish Kumar Sahu



Q. Write a java program to perform file merge operation where merging should be done line by line alternatively?

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
public class test
{
	public static void main(String args[])
	{
		String inputfile1 = "file1.txt";
		String inputfile2 = "file2.txt";
		String inputfile3 = "file3.txt";

		mergefile(inputfile1, inputfile2, inputfile3);
	}
	public static void mergefile(String f1, String f2, String f3)
	{
		try(BufferedReader r1 = new BufferedReader(new FileReader(f1));
			BufferedReader r2 = new BufferedReader(new FileReader(f2));
			BufferedWriter w = new BufferedWriter(new FileWriter(f3)))
		{
			String line1, line2;

			//alternative data add
			while((line1 = r1.readLine()) != null && (line2 = r2.readLine()) != null)
			{
				w.write(line1);
				w.newLine();
				w.write(line2);
				w.newLine();
			}
			while((line1 = r1.readLine()) != null)
			{
				w.write(line1);
				w.newLine();
			}
			while((line2 = r2.readLine()) != null)
			{
				w.write(line2);
				w.newLine();
			}
			System.out.println("Files Merged Successfully!");
		}catch(IOException e){
			e.printStackTrace();
		}
	}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
Files Merged Successfully!

C:\Users\Atish kumar sahu\desktop>
```



Q. Write a java program to extract a zip file into a folder?

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;

public class test {
    public static void main(String[] args) {
        String zipFilePath = "file1.zip";
        String outputFolder = "extractedContents";

        extractZipFile(zipFilePath, outputFolder);
    }
```

```
    private static void extractZipFile(String zipFilePath, String outputFolder) {
        byte[] buffer = new byte[1024];

        try (ZipInputStream zipInputStream = new ZipInputStream(new FileInputStream(zipFilePath))) {

            File folder = new File(outputFolder);
            if (!folder.exists()) {
                folder.mkdir();
            }

            ZipEntry zipEntry;
            while ((zipEntry = zipInputStream.getNextEntry()) != null) {
                String entryName = zipEntry.getName();
                File entryFile = new File(outputFolder + File.separator + entryName);

                // Create parent directories if they don't exist
                new File(entryFile.getParent()).mkdirs();

                try (FileOutputStream outputStream = new FileOutputStream(entryFile)) {
                    int length;
                    while ((length = zipInputStream.read(buffer)) > 0) {
                        outputStream.write(buffer, 0, length);
                    }
                }
                zipInputStream.closeEntry();
            }

            System.out.println("Contents extracted successfully to: " + outputFolder);

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

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
Contents extracted successfully to: extractedContents

C:\Users\Atish kumar sahu\desktop>
```