M T W T F S S

Page No.:

Date:

YOUVA

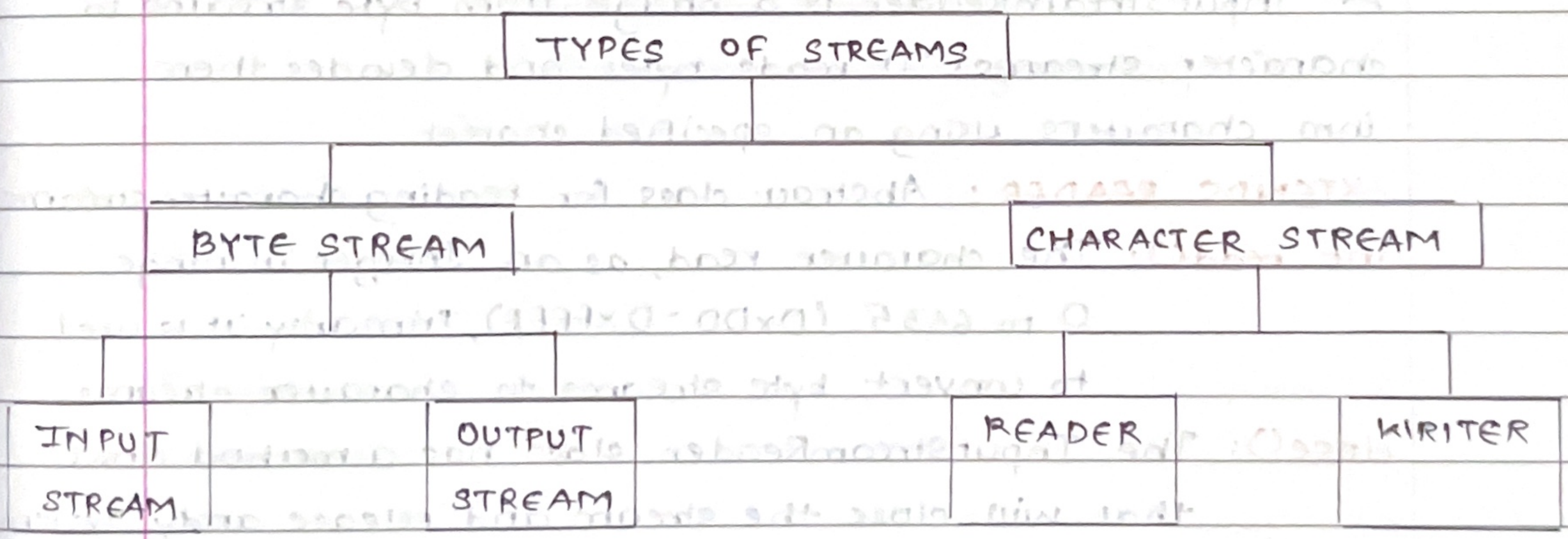
# FILE HANDLING

## IO Exceptions:

An IO Exception is any unexpected problem the JVM encounters while attempting to run a program.
When a IO Exception is thrown, it means whatever is throwing the exception can throw a IO Exception, for example if a file is not found, corrupted etc.

## Streams:

A stream simply represents a sequence of data (bytes or unicode) in some sort of sequential queue.
Java program perform I/O through streams. A stream is an abstraction that either produces or consumes information.
A stream is linked to physical device by Java I/O system.




**Byte streams:** Byte streams provide a convinent means for handling input and output of bytes.
Byte stream contains binary data which is useful for things such as reading or writting to file

**Character streams:** Character streams provide convinent means for handling input and output of characters.

They use unicode and therefore can be internationalized, good candidate for things like keyboard input and console output.
Also in some cases character streams are much more efficient than byte streams.

Predefined streams:
System.out refers to the standard output stream. By default this is console.
System.in refers to standard input which is the keyboard by default.
System.err refers to the standard error stream, which also is the console by default.

- InputStreamReader class
An InputStreamReader is a bridge from byte streams to character streams. It reads bytes and decodes them into characters using an specified charset
EXTENDS READER: Abstract class for reading character streams
int read(): The character read, as an Integer in range 0 to 6535 (0x00 - 0xffff), primarily it is used to convert byte streams to character streams.
close(): The InputStreamReader class has a method close() that will close the stream and release any system resources associated with it.
The close() method should be called once you are done with input stream.
try-with-resources: The try-with-resources statement is a try statement that declares one or more resources. A resource is an object that must be closed after a program is finished with it. The try-with-resource statement ensures that statement is closed in end.

★ How do we know if a class is resource? Simple If it implements java.lang.Autoclosable the class can be considered as a resource.

Example:

```
public class Main {
    public static void main(String[] args) {
        try(InputStreamReader isr = new InputStreamReader
            (System.in)) {
            System.out.println("Enter letters:");
            int letters = isr.read();
            while(isr.ready()){
                System.out.println((char) letters);
                letters = isr.read();
            }
        } catch(IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Output:

```
Enter letters:
abc
a
b
c
```

- FileReader

  The purpose of the FileReader class is to read character files. FileReader class does not define or override any public methods, so it inherits all of its methods from its superclass InputStreamReader.

class hierarchy:

OBJECT → READER → INPUTSTREAMREADER → FILEREADER

Example:

note.txt

abc

Main.java:

```
public class Main {
    public static void main(String[] args) {
        try(FileReader fr = new FileReader("note.txt")){
            int letters;
            while (letters = fr.read() != -1) {
                System.out.print((char) letters);
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Output:

abc

- BufferedReader

It extends Reader. Hence it has own's read() which is same as InputStreamReader.

The BufferedReader is used to read text from character stream

The BufferedReader class introduces a method named ReadLine() which will read an entire line of text.

The BufferedReader class implements AutoClosable.
To obtain a character based stream that is attached to the console, wrap System.in in a BufferedReader object.

Example:

```
public class Main {
    public static void main(String[] args) {
        try(BufferedReader br = new BufferedReader(
            new InputStreamReader(System.in))) {
            System.out.println("You typed: " + br.readLine());
        } catch(IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Output:
hello
You typed: hello

- OutputStreamWriter

OutputStreamWriter is a class that bridges between character streams and byte streams
There are only four public methods in OutputStreamWriter class:

String getEncoding(): Returns the name of the character encoding being used by this stream.
void flush(): Flushes the stream.
void close(): Closes the stream, flushing it first. Once the stream has been closed, further write() or flush() invocation will cause IOException to be thrown

write() : Method has three overloaded versions:

- void write(int c) : writes a single character
- void write(char[]cbuf) : writes a portion of array of characters.
- void write(String str) : writes a portion of string.

Example:

```
public class Main {
    public static void main(String[] args) {
        try(OutputStreamWriter osw = new OutputStream
            Writer(System.out)) {
            osw.write("Hello World");
            osw.write(97); // 'a'
            osw.write(10); // New line
            osw.write('A');
            osw.write('\n');
            char[] arr = new "hello world".toCharArray();
            osw.write(arr);
        } catch(IOException e){
            System.out.println(e.getMessage());
        }
    }
}
```

Output:

```
Hello World a
A
hello world
```

- FileWriter

The purpose of the FileWriter class is to simply write character based files.

FileWriter class does not define or override any public methods, so it inherits all of its methods from its

superclass OutputStreamWriter.

Example:

note.txt

abc

Main.java

```
public class Main {
    public static void main (String [] args) {
        try (FileWriter fw = new FileWriter("note.txt")) {
            fw.write("hello world");
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

note.txt

hello world ⟶ overwrites the file.

If we dont want to overwrite and just append. change in constructor.

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

- BufferedWriter

The BufferedWriter class is used to write text to a character stream.

The BufferedWriter class method named newLine() which means no need to hardcode "\r\n" into output stream.

Example:

note.txt

a

Main.java

```
public class Main {
    public static void main(String[] args) {
        try (BufferedWriter bw = new BufferedWriter(
            new FileWriter("note.txt"))) {
            bw.write("xyz");
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

note.txt

xyz

- File.

The file class provides us with several tools to dynamically create our directory and file structure.

Example:

```
class Main {
    public static void main(String[] args) {
        try {
            File fo = new File("File.txt");
            fo.createNewFile();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```