Input/Output Streams

INTRODUCTION

Input/output Streams

LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Understand concept of Streams
- Distinguish between Byte Stream and Character Stream
- Understand the IO package and its class Hierarchy
- Read and write to Files using Byte Stream
- Chaining of Streams
- Read and write to Files using Character Stream

Refer package **com.mgait.io** in the provided code base for demo programs on the topics covered in this presentation

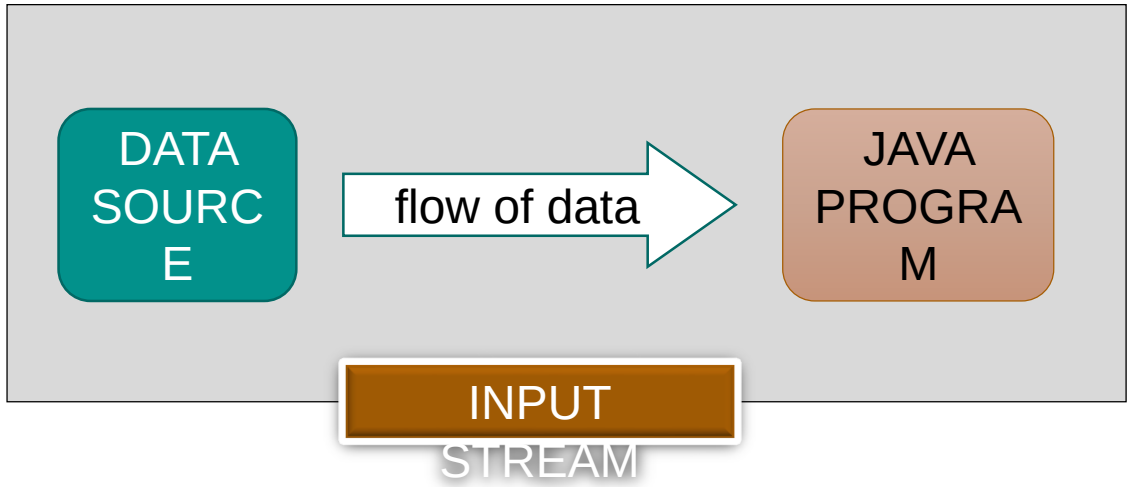
CONCEPT

Input and Output Streams

What is a Stream?

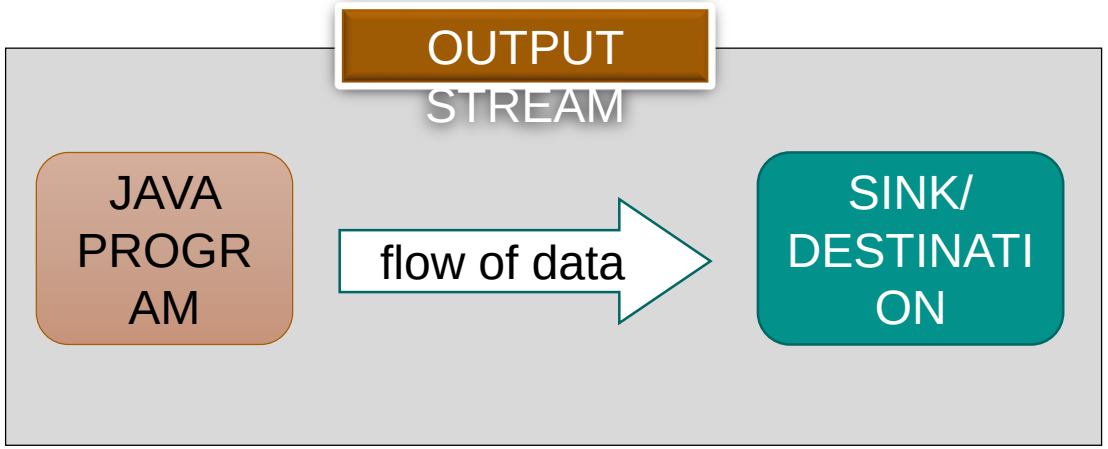
- Java programs perform I/O through streams
- A stream is
  - Flow of data from Source to Destination
  - an abstraction that either produces or consumes data
- Streams support different kinds of data – simple bytes and primitive data types.

Input and Output Streams

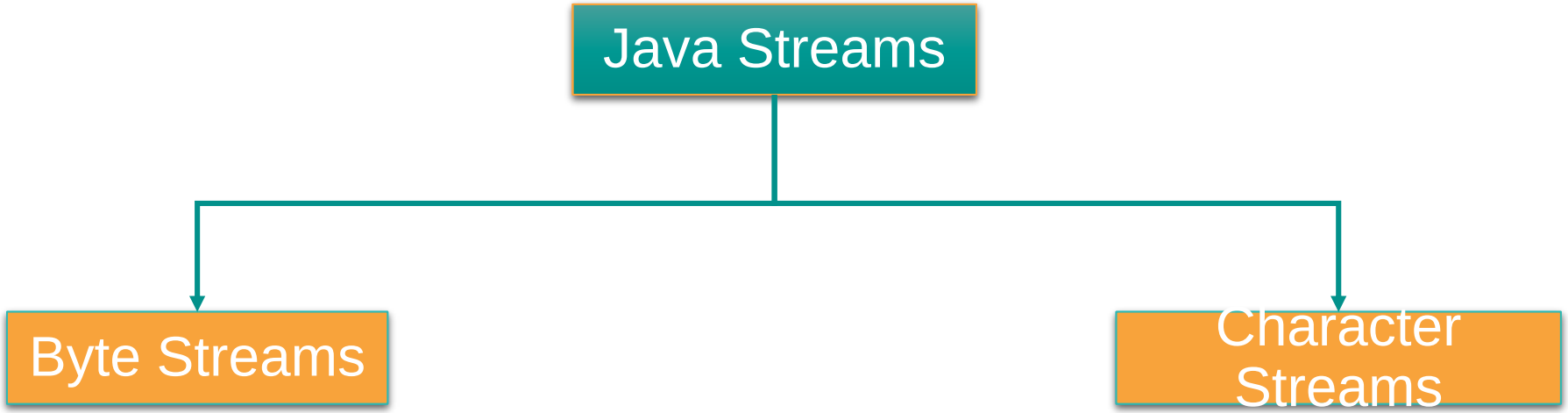


- Input Stream depicts the flow of data from data source to the programs memory
- Java Programs use an input stream to read data from a data

- Output Stream depicts the flow of data from program memory to the destination
- Java Programs uses a Output Stream to write data to a



Types of Stream Classes



- Enables input and output of data in bytes
- Used for reading or writing binary data

- Enables input and output of data in characters
- Used for reading and writing text
- Uses Unicode, and, therefore, can be internationalized

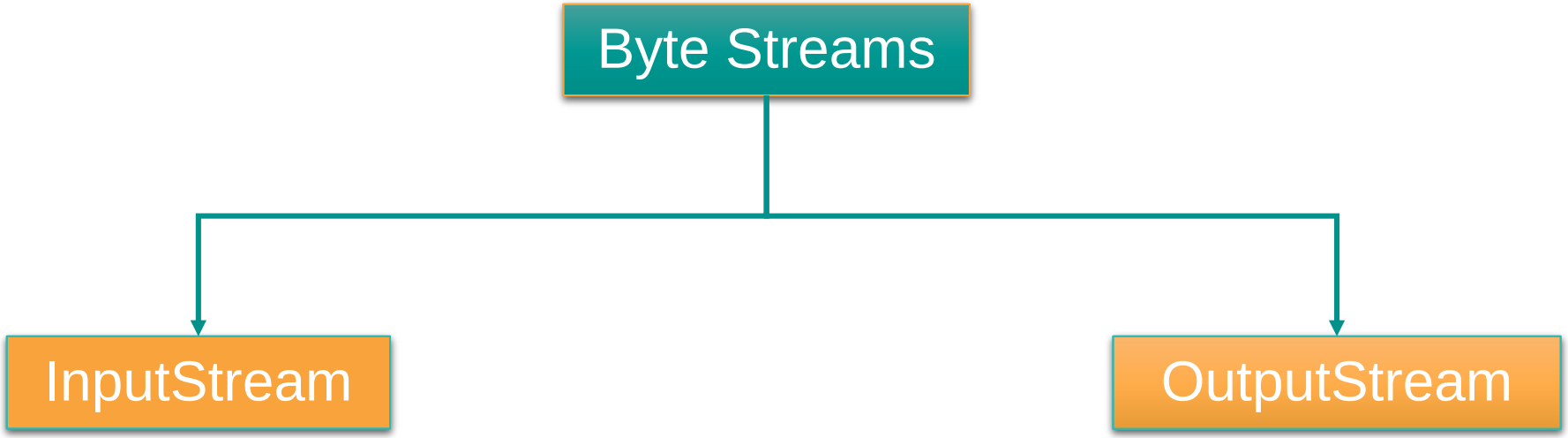
➢ **java.io** package provides extensive set of classes for handling I/O to and from various devices

CONCEPT

Byte Streams

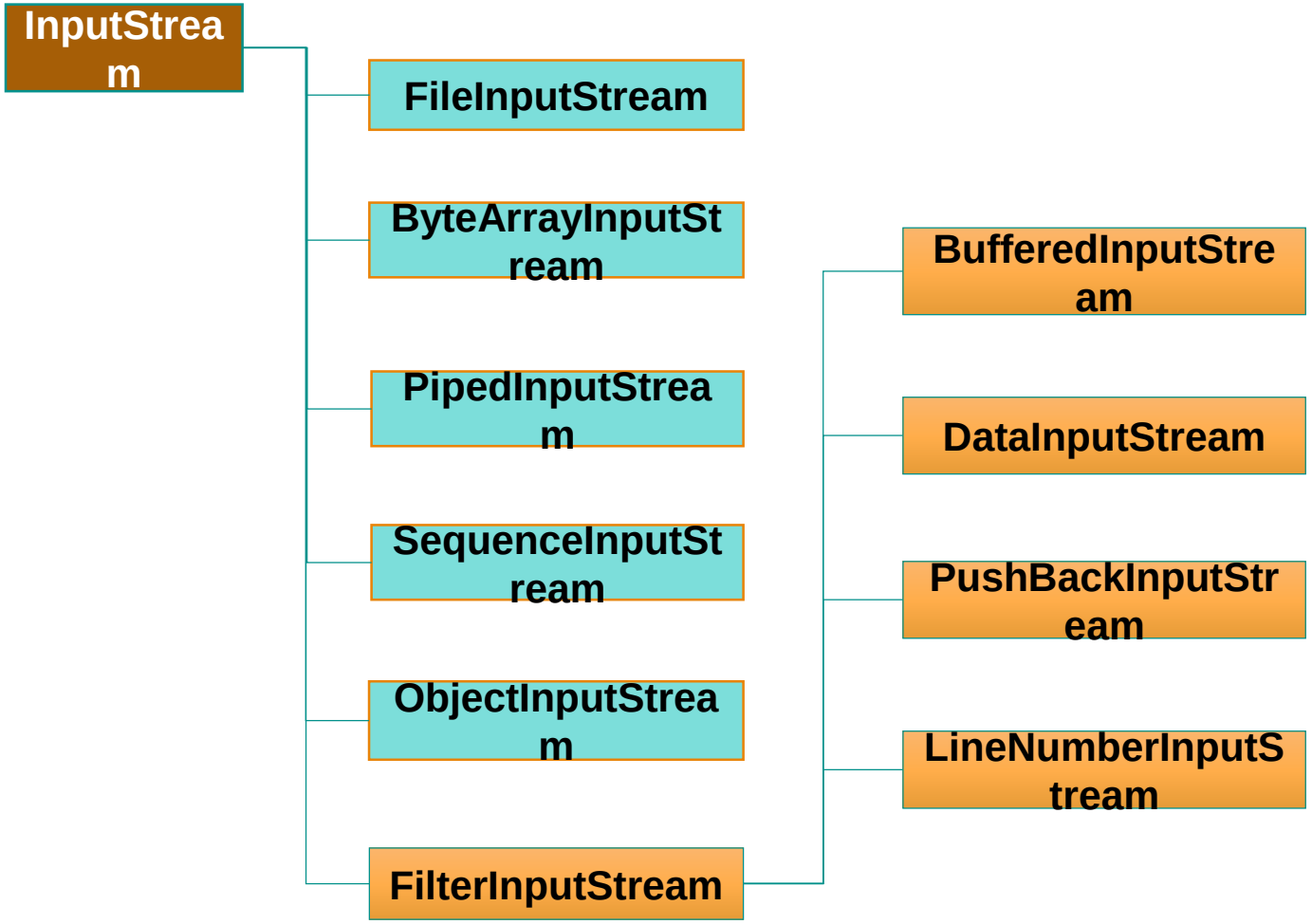
Byte Stream Classes

- Byte streams are defined by two **abstract** base classes called InputStream and OuputStream



- Contains several concrete classes for doing byte oriented reading and writing

InputStream Hierarchy



- InputStream class is an abstract class which serves as a base class for other Input stream classes
- InputStream class defines methods for reading streamed bytes of information
- Java Applications use InputStream to read data from a source which can be a file, array, device or network socket
- An InputStream is automatically opened once it is created.
- InputStream has to be closed after reading data to release any system resources held

Frequently used Methods of InputStream class

`available(): int`
Returns an estimate of the number of bytes that can be read (or skipped over) from this input stream

`read() : int`
Reads the next byte of data from InputStream. Returns -1 when the end of the stream is reached

`read(byte[] b): int`
Reads n number of bytes from the input stream and stores them into the buffer array b. Returns total number of bytes read into the buffer or -1 when the end of the stream is reached

`close(): void`
Closes the input stream and releases any system resources associated with the stream

`skip(long n): long`
Skips over and discards n bytes of data from this input stream

FileInputStream

- Used to create an InputStream to read bytes from a file in a file system
- Meant for reading streams of raw bytes such as image data
- Can be instantiated using the following constructors

```
• FileInputStream(String filepath)
• FileInputStream(File fileObj)
```

- Creating a FileInputStream for reading from a file named "input.txt"

```
FileInputStream input = new FileInputStream("c:\\input.txt");
```

- Reading from the File

```
int bytedata = input.read();
```

FileNotFoundException is thrown if the file being read is not found in the File system

BufferedInputStream

- FilterInputStream
  - Acts like a filter to transform the raw bytes of data to a desired form or to provide additional functionality
  - uses other input streams as its basic source of data

- BufferedInputStream
  - Is a FilterInputStream, which provides the ability of buffering the input, to another input stream
  - By default, the streams are not buffered. When a BufferedInputStream is created, an internal buffer array is created
  - When a read is done, BufferedInputStream reads multiple bytes in to the buffer using the original input stream

Constructor:
`BufferedInputStream(InputStream in)`

umber of reads on the original input stream are reduced

```
FileInputStream input = new FileInputStream("input.txt");
BufferedInputStream bis = new BufferedInputStream(input);
```

Demo
Class :

Reading a file using Byte Streams

```
File inFile = new File("inputFile.txt");
FileInputStream fis = null; BufferedInputStream bis = null;
int data;
StringBuilder content = new StringBuilder();
// Reading from a File Using BufferedInputStream
try {
    fis = new FileInputStream(inFile);
    bis = new BufferedInputStream(fis);
    while ((data = bis.read()) != -1) {
        content.append((char) data);
    }
    System.out.println(content);
} catch (FileNotFoundException e) {
    System.out.println("File Not Found");
} catch (IOException e) {
    e.printStackTrace();
} finally {
    if (bis != null) {
        try {
            bis.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Creating a BufferedInputStream for reading from inputFile.txt

-1 indicates end of File

Casting the byte read in to character

DataInputStream

- DataInputStream is a FilterInputstream that used to read primitive data types from an input stream in a machine-independent way
- It aggregates group of bytes in to primitive datatypes

```
Constructor:
DataInputStream(InputStream in)
```

```
FileInputStream input = new FileInputStream("input.txt");
DataInputStream dis = new DataInputStream(input);
```

- Methods

```
readInt()   : int (reads an input stream and returns an int)
readByte()  : byte
readFloat()  : float
readDouble()  : double
readChar()  : char
readBoolean()  : boolean
```

ByteArrayInputStream

- Allows to read data from byte arrays as streams
- Closing a ByteArrayInputStream has no effect

Constructors:

- `ByteArrayInputStream(byte[] buf)`
- `ByteArrayInputStream(byte[] buf, int offset, int length)`

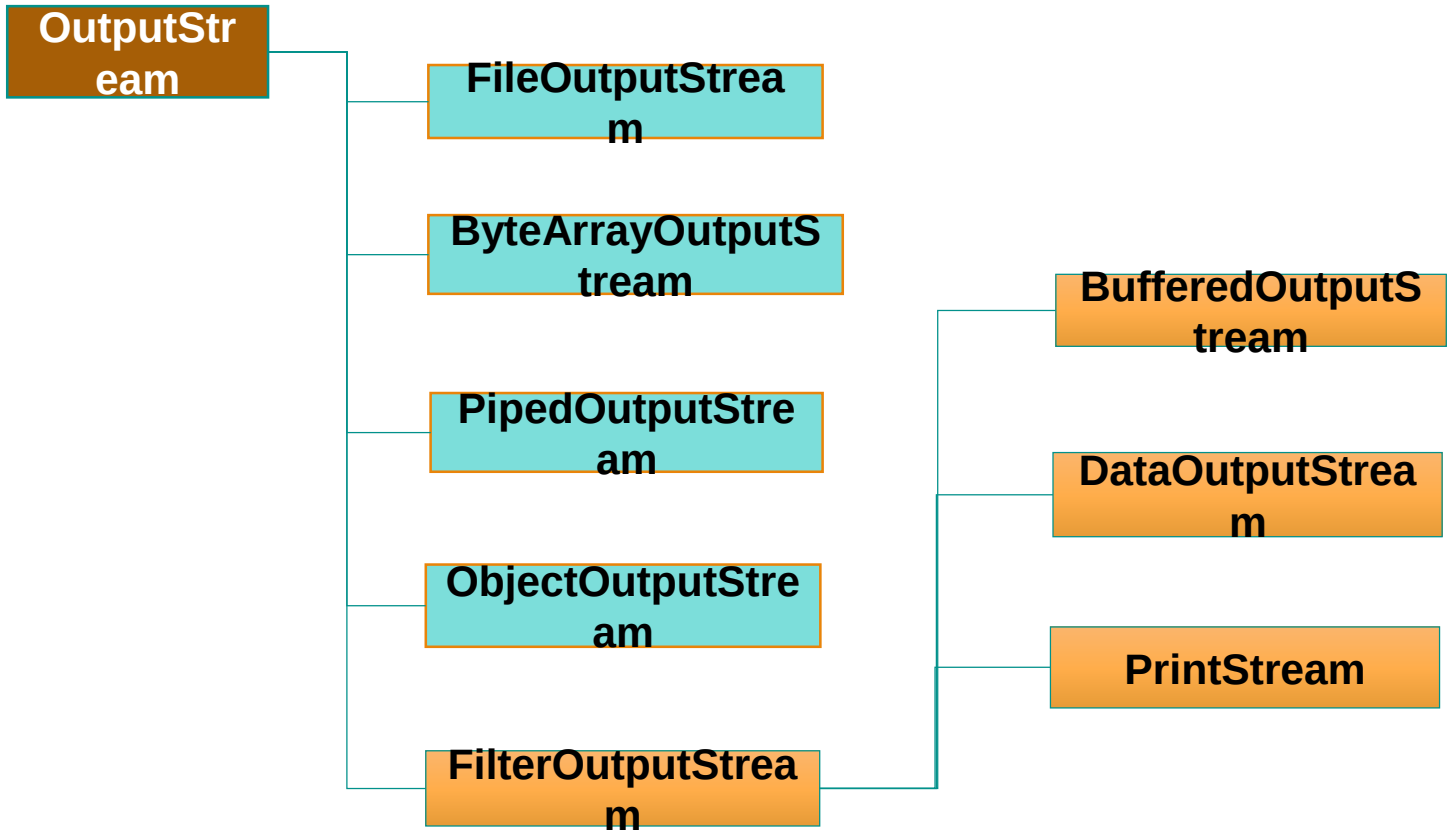
```
String inp = "test";
byte[] bytes = inp.getbytes();
InputStream input = new ByteArrayInputStream(bytes);
int data = input.read();
while(data != -1) {
        data = input.read();
}
```

Chaining of Streams

- Java allows multiple streams to be chained to obtain a desired functionality
- For ex.
  - To read primitive values stored in a file in filesystem using a buffer, we would need a the following streams
    - FileInputStream for reading bytes from file
    - BufferedInputStream for buffering the bytes
    - DataInputStream for transforming the bytes in to primitive data types

```
FileInputStream fis = new FileInputStream("input.txt");

BufferedInputStream bis = new BufferedInputStream(fis);

DataInputStream dis = new DataInputStream(bis);

int i = dis.readInt();
```

OutputStream Class Hierarchy



- OutputStream is an abstract super class of all classes representing an output stream of bytes
- Java Applications use OutputStream to write data to a destination which can be a file, array, device or network socket
- OutputStream class contains methods for writing bytes to the destination

Frequently used Methods of OutputStream class

`write(int b) : void`
Writes the specified byte to this output stream

`write(byte[] b): void`
Writes b.length bytes from the specified byte array to this output stream

`flush(): void`
Flushes the output stream and forces any buffered output bytes to be written out

`close(): void`
Closes the output stream and releases any system resources associated with the stream

FileOutputStream

- is an OutputStream used to write data to a file in bytes

Constructors

- `FileOutputStream(String filepath)`
- `FileOutputStream(File fileObj)`
- `FileOutputStream(String filepath, boolean append)`
  - if boolean arg is true, file is opened in append mode
- `FileOutputStream(File fileObj, boolean append)`

- Writing to a File using FileOutputStream

```
File outFile = new File("outFile.txt");
FileOutputStream fos = new FileOutputStream(outFile,true);
String text = "Hello";
byte[] textBytes = text.getBytes();
fos.write(textBytes);
```

Note: If the file denoted by the path does not exist, OutputStream's create a new File in the file system

FilterOutputStream

- BufferedOutputStream
  - OutputStream which uses an internal buffer where the bytes are written, thus reducing the number of writes to the destination device

Constructors

- `BufferedOutputStream(OutputStream out)`
- `BufferedOutputStream(OutputStream out,int bufferSize)`

```
FileoutputStream fos = new FileOutputStream("output.txt");
BufferedOutputStream bis = new BufferedOutputStream(fos);
```

- DataOutputStream
  - lets an application write primitive Java data types to an output stream in a portable

```
FileoutputStream fos = new FileOutputStream("output");
DataOutputStream dos = new DataOutputStream(fos);
dos.writeFloat(12.0f);
```

Writing to a File using Byte Stream

```
public class OutputStreamDemo {
    public static void main(String args[]) {
        File outFile = new File("OutFile.txt");
        FileOutputStream fos = null;
        BufferedOutputStream bos = null;
        String data = "Hello World";
        try {
            fos = new FileOutputStream(outFile);
            bos = new BufferedOutputStream(fos);
            bos.write(data.getBytes());
            bos.flush();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (bos != null) {
                try {
                    bos.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

If OutFile.txt does not exist on file system, a new file is created

Getting a byte array from a string

Flushes the data in the buffer and writes in to the destination

PrintStream

- Adds functionality to another output stream to print representations of various data values
- Enables other output streams to write **formatted data** to the destination
- Does not throw IOException in case of failure
- Need to use checkError() method to check failure

Constructors

- `PrintStream(File file)`
- `PrintStream(String fileName)`
- `PrintStream(OutputStream out)`

Methods

```
print(boolean b)
print(char c)
print(char[] s)
print(String s)
print(double d)
print(int i)
println(…)
…
format(String format, Object... args)
```

PrintStream

```
PrintStream output=new PrintStream("C:\\test.txt");
output.print("Employee ID : ");
output.println(101);
output.format("%1$10s : %2$,5.2f " ,"Salary",2000.0f);
output.flush();
```

Employee ID : 101
Salary : 2,000.00

Predefined streams

- System class of the java.lang package contains three predefined stream variables
  - in
  - out
  - err

- These variables are declared as public and static within System:
- System.out is a PrintStream object, whose destination is the console.
- System.in is a InputStream object, which reads from the the keyboard.
- System.err is a PrintStream object, whose destination is the console.

1. What is the output of the below code?

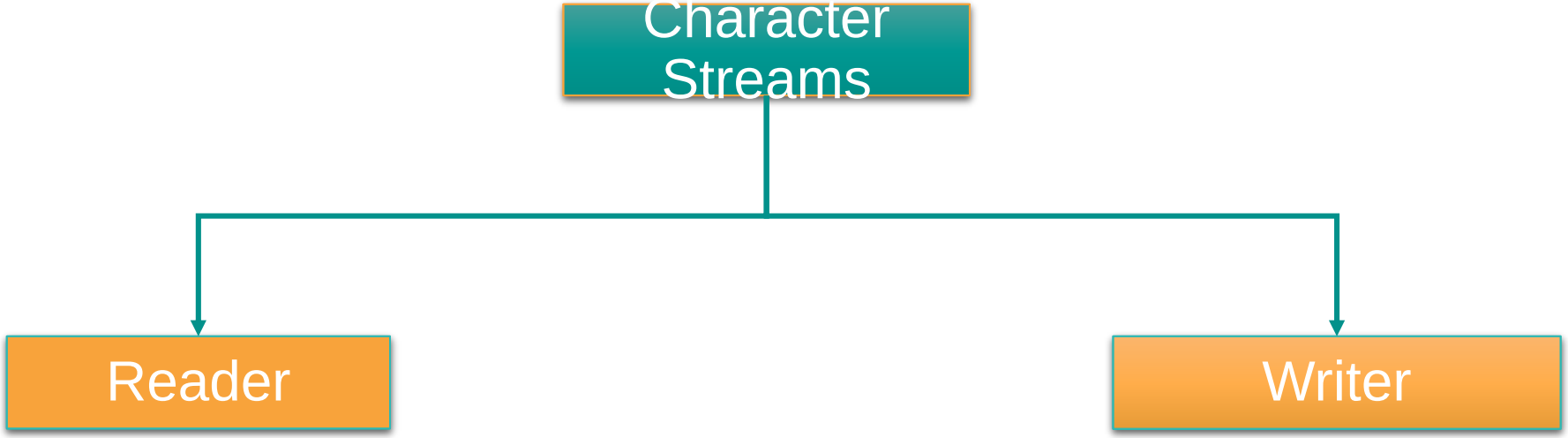
```
int b;
String test = "hello";
byte[] bArr = test.getBytes();

ByteArrayInputStream inp = new ByteArrayInputStream(bArr);
while((b =inp.read()) != -1){
    System.out.println((char)b);
}

ByteArrayInputStream inp1 = new ByteArrayInputStream(bArr,3,2);
while((b =inp1.read()) != -1){
    System.out.println((char)b);
}
```

CONCEPT

Character Streams

Character Stream Classes

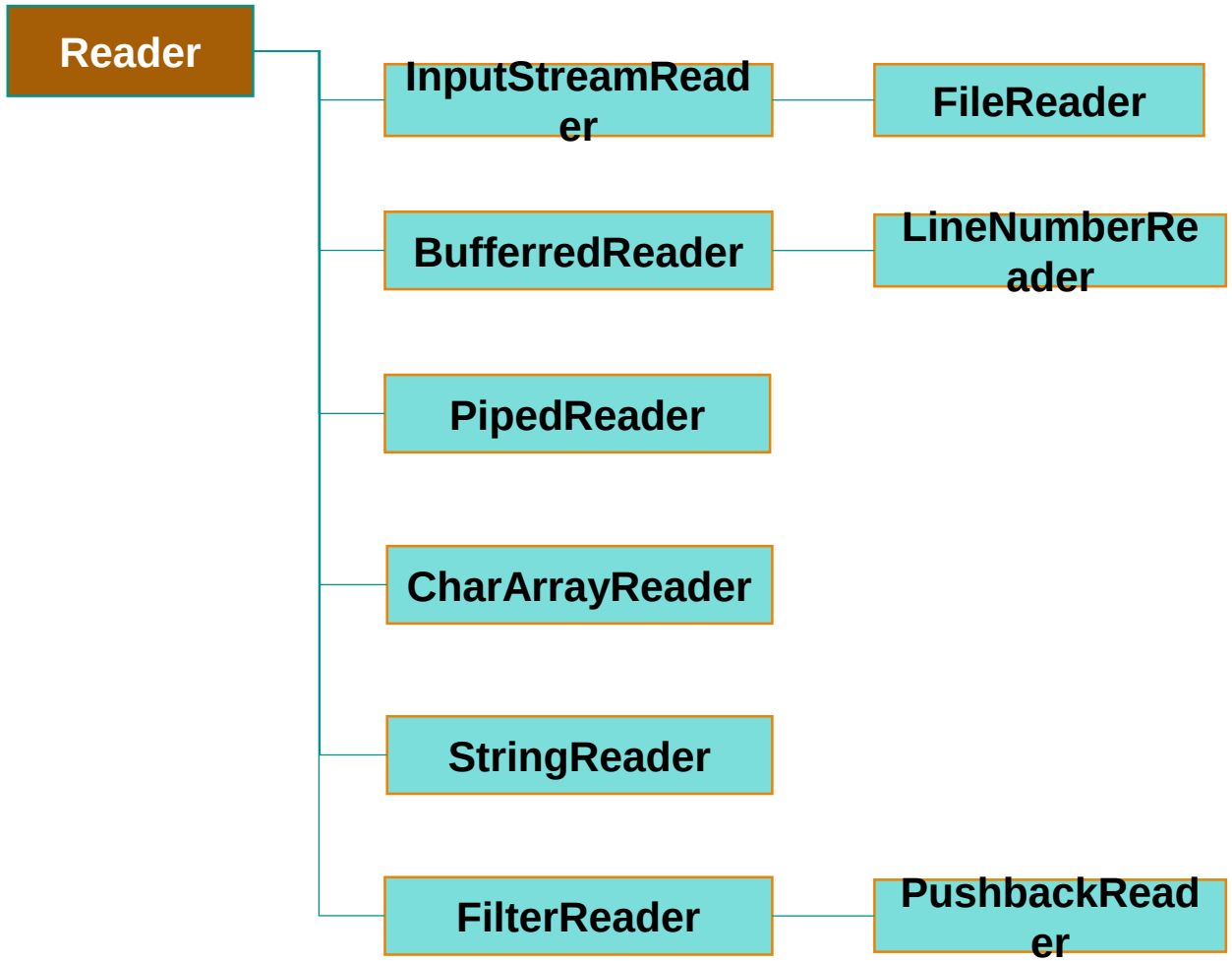
- Character streams are defined by two **abstract** base classes called Reader and Writer



- Contains several concrete classes for doing character oriented reading and writing
- Readers and Writers support same operations as InputStreams and OutputStreams

- Handles Unicode characters and hence easy to internationalize

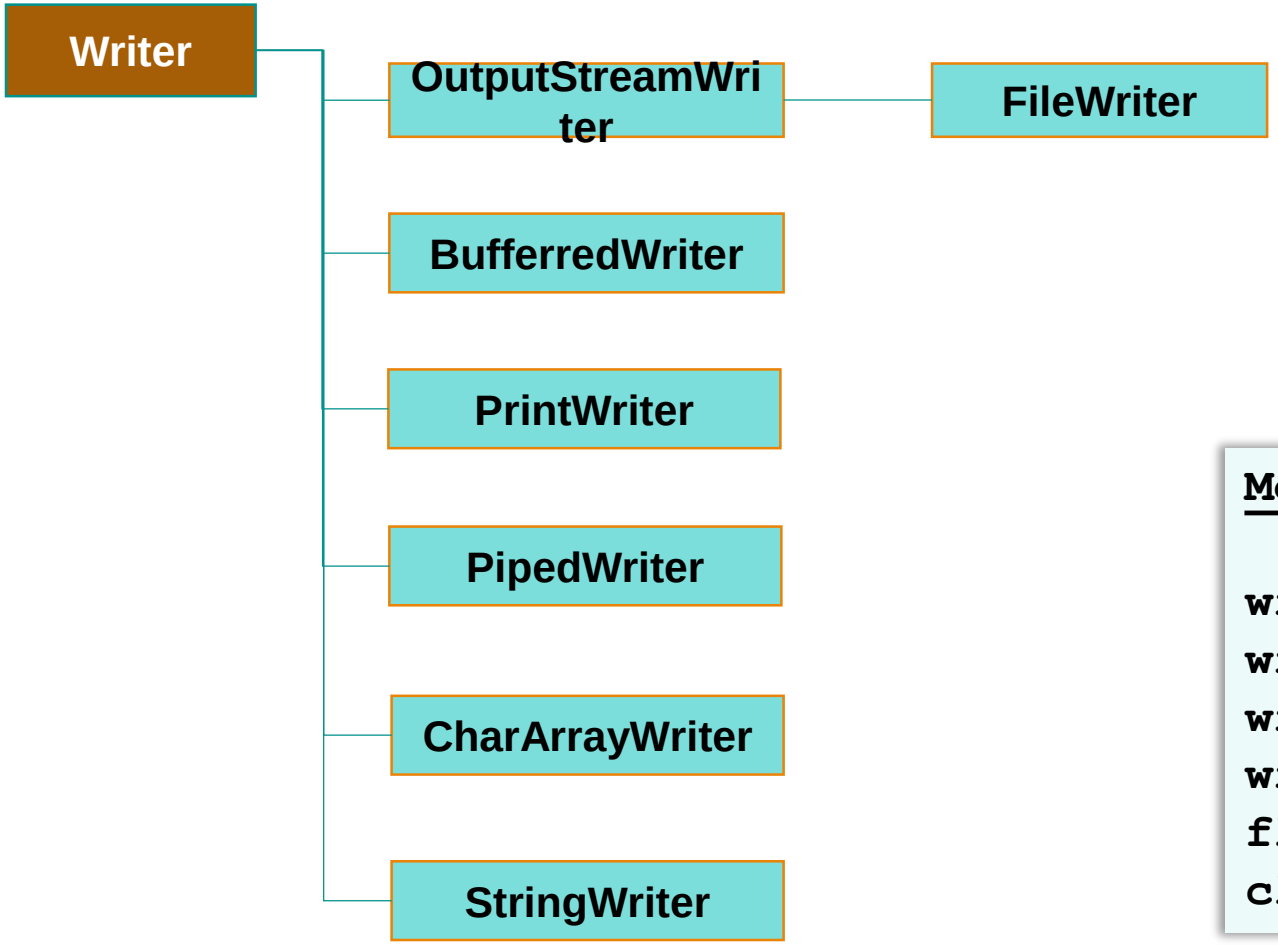
Reader Hierarchy



Methods of Reader class

```
read()                  : int
read(char[] cbuf)       : int
ready()                 : boolean
close()                 : void
skip(long n)            : long
```

Writer Hierarchy



Methods of Writer class

```
write(int c)              : void
write(char[] cbuf)        : void
write(String str)         : void
write(String str, int off, int len):void
flush()                   : void
close()                   : void
```

FileReader and FileWriter

FileReader

- used to read character data from a file
- Can read an array of Characters

```
Constructors
FileReader(String path)
FileReader(File  f)
```

FileWriter

- Used to write character data to a File
- Has method to write a String to a

```
Constructors
FileWriter(String path)
FileWriter(File f)

To Append data in a File
FileWriter(String path, boolean apnd)
FileWriter(File f, boolean apnd)
```

Writing to File using FileWriter

Demo
Class :

```
FileWriter fw=new FileWriter("out.txt",true);
fw.write(109);
fw.write("anipal");
fw.write("\n");
char[] chArr={'g','l','o','b','a','l'};
fw.write(chArr);
fw.flush();
```

Output

```
manipal
global
```

BufferedReader and BufferedWriter

- BufferedReader
  - makes the reading of characters, arrays, Strings efficient by providing the functionality of buffering the characters
  - provides a method **readLine()** using which a line of text can be read

```
BufferedReader in = new BufferedReader(new FileReader("input.txt"));
String line = in.readLine();
```

- BufferedWriter
  - Buffers characters to provide efficient writing of single characters, arrays, and strings.
  - Provide a method newline() for writing a line separator based on the platform

```
BufferedWriter out = new BufferedWriter(new FileWriter("out.txt"));
out.write("Hello");
out.newline();
```

Copying a Text file

Demo
Class :

```
reader = new BufferedReader(new FileReader("myFile.txt"));
writer = new BufferedWriter(new FileWriter("myFile_Copy.txt"));
while((line = reader.readLine())!= null){
        writer.write(line);
        writer.newLine();
}
writer.flush();
```

null indicates the end of file

Reading from Console using InputStreamReader

- InputStreamReader
  - Bridge from byte streams to character streams
  - Reads bytes and decodes them into characters using a specified charset

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader in = new BufferedReader(isr);
System.out.print("Enter name: ");
name = in.readLine();
System.out.print("Enter city: ");
city = in.readLine();
```

The Scanner Class

- Used to read strings and primitives from Files and Console
- part of the java.util package

- Reading a File Using Scanner

```
File file = new File("test.txt");
Scanner scan = new Scanner(file);
```

Using Scanner to read from the file

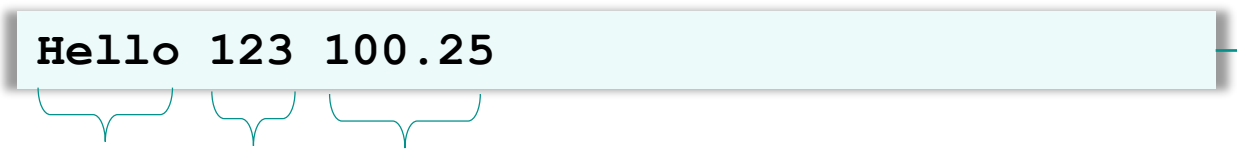
- Reading from console using Scanner

```
Scanner scan = new Scanner(System.in);
```

Using Scanner to Read from the console

Reading a File using Scanner

- Uses a delimiter to break the text in to multiple tokens
- Default delimiter is white space character
- Delimiter can be changed by *useDelimiter()* method
- Has methods which can translate individual tokens to their data type




```
File file = new File("test.txt");
Scanner scan = new Scanner(file);
String test = scan.next();
int i = scan.nextInt();
float f = scan.nextFloat();
```

Methods of Scanner class

```
next() : String
Returns the token as a String
nextInt() : int
Returns the next token as an int
nextFloat() : float
nextXXX()     : XXX
```

References

- Refer following demo videos on EduNxt
  - M6l3l2 Writing Data To A Text File – Demo
  - M6l3l4 Reading Data From A Text File – Demo
  - M6l4l1 Writing Numbers To A File – Demo
  - M6l4l2 Reading Numbers From A File – Demo
  - M6l4l7 Demonstration Of Reading From Keyboard – Demo
  - M8l4l2 Using Scanner To Read Data From Keyboard - Demo

SUMMARY

Input/output Streams

SUMMARY

In this lesson, you've learned to:

- Distinguish between Byte Stream and Character Stream
- Read and write to Files using Byte Streams
- Chain Streams
- Read and write to Files using Character Stream
- Read from Console