JAVA Programming

Course Instructor: **Dr. N Nandini Devi**

School of Computer Science

UPES Dehradun

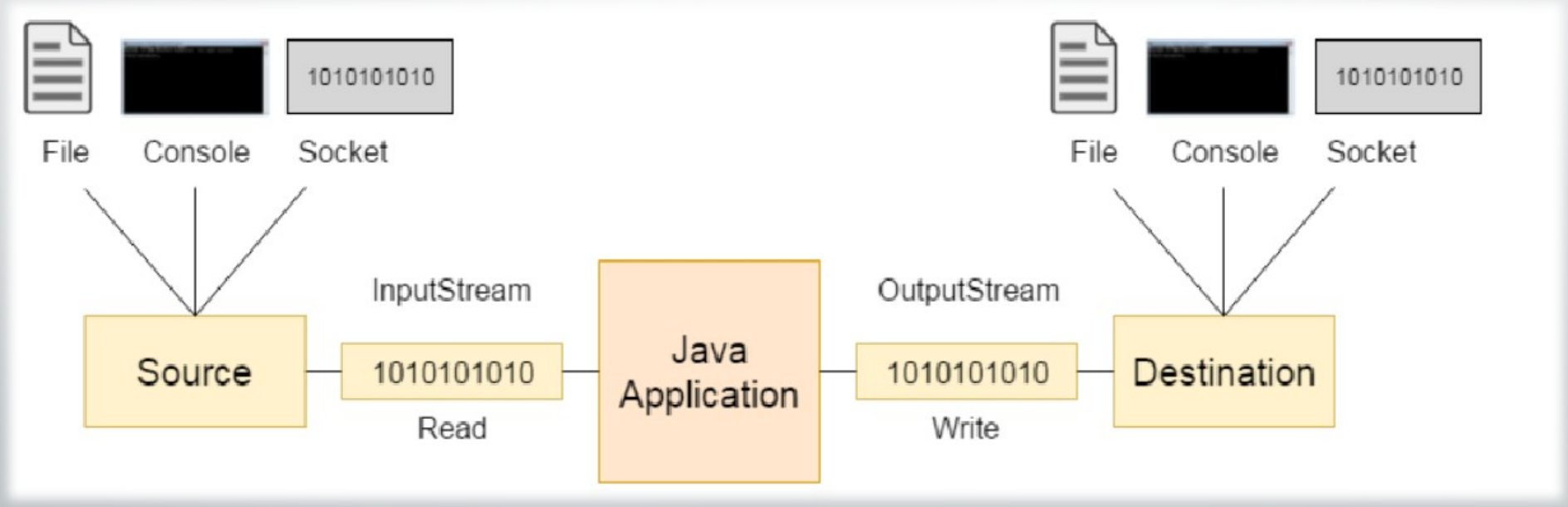
TOPICs to be discussed

- Introduction to I/O Streams in Java
- Types of Streams
- Different I/O Methods
- Byte I/O Streams
- Character I/O Streams

Let’s START ...!!!

I/O Streams in Java

- **Java I/O** (Input and Output) is used to process the input and produce the output.
- **Java** uses the concept of a stream to make **I/O operation** fast. The java.io package contains all the classes required for input and output operations.



I/O Streams in Java

Streams:

A **stream** is a sequence of data. In **Java**, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

- In **Java**, 3 streams are created for us automatically. All these streams are attached to the console.

1. **System.in:** Standard input stream that reads characters from the keyboard or any other standard input device.
2. **System.out:** Standard output stream that is used to produce the result of a program on an output device like the computer screen.
3. **System.err:** Standard error stream that is used to output all the error data that a program might throw, on a computer screen or any standard output device.

Types of Streams

- Depending on the type of operations, streams can be divided into two primary classes:
  - InputStream
  - OutputStream

- **InputStream:**

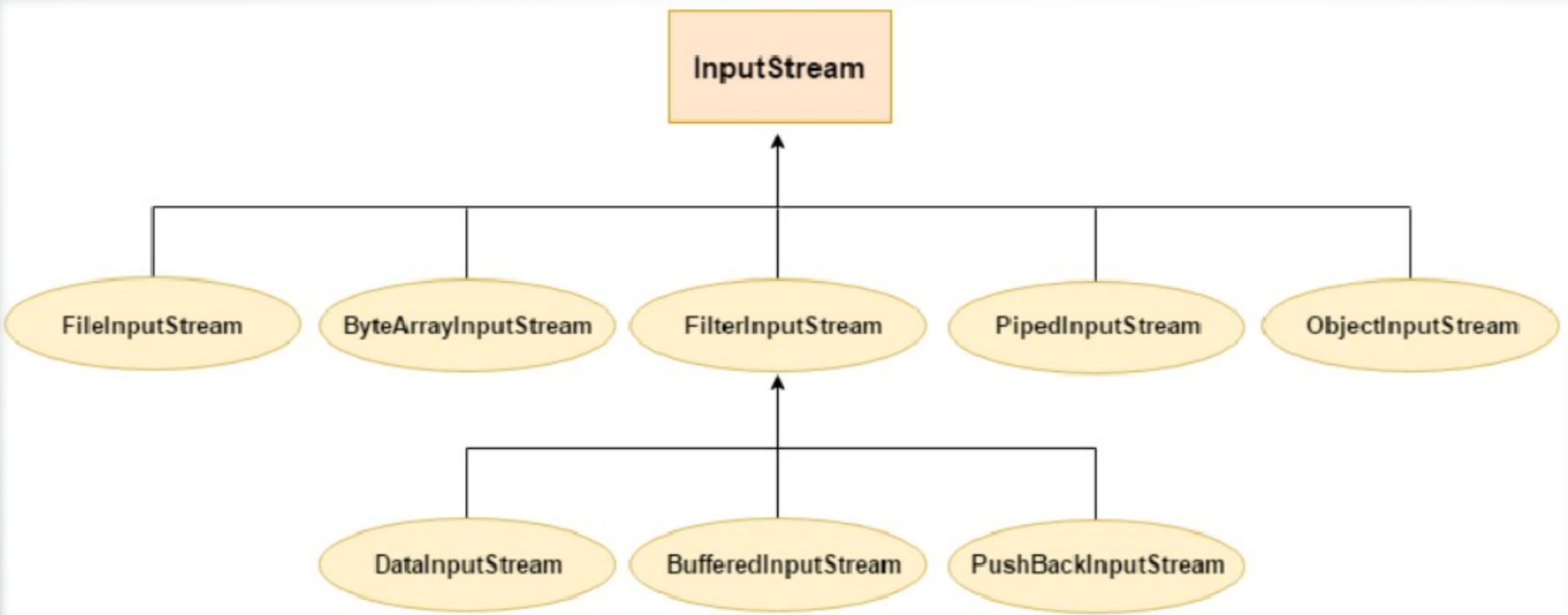
Java application uses an **input stream** to read data from a source; it may be a file, an array, a peripheral device, or a socket.

- **OutputStream:**

Java application uses an **output stream** to write data to a destination; it may be a file, an array, a peripheral device, or a socket.

InputStream Class

- InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.



Useful Methods of InputStream

- Some of the most used methods in the InputStream class are:

1) `public abstract int read() throws IOException:`

Reads the next byte of data from the input stream. It returns -1 at the end of the file.

2) `public int available()throws IOException:`

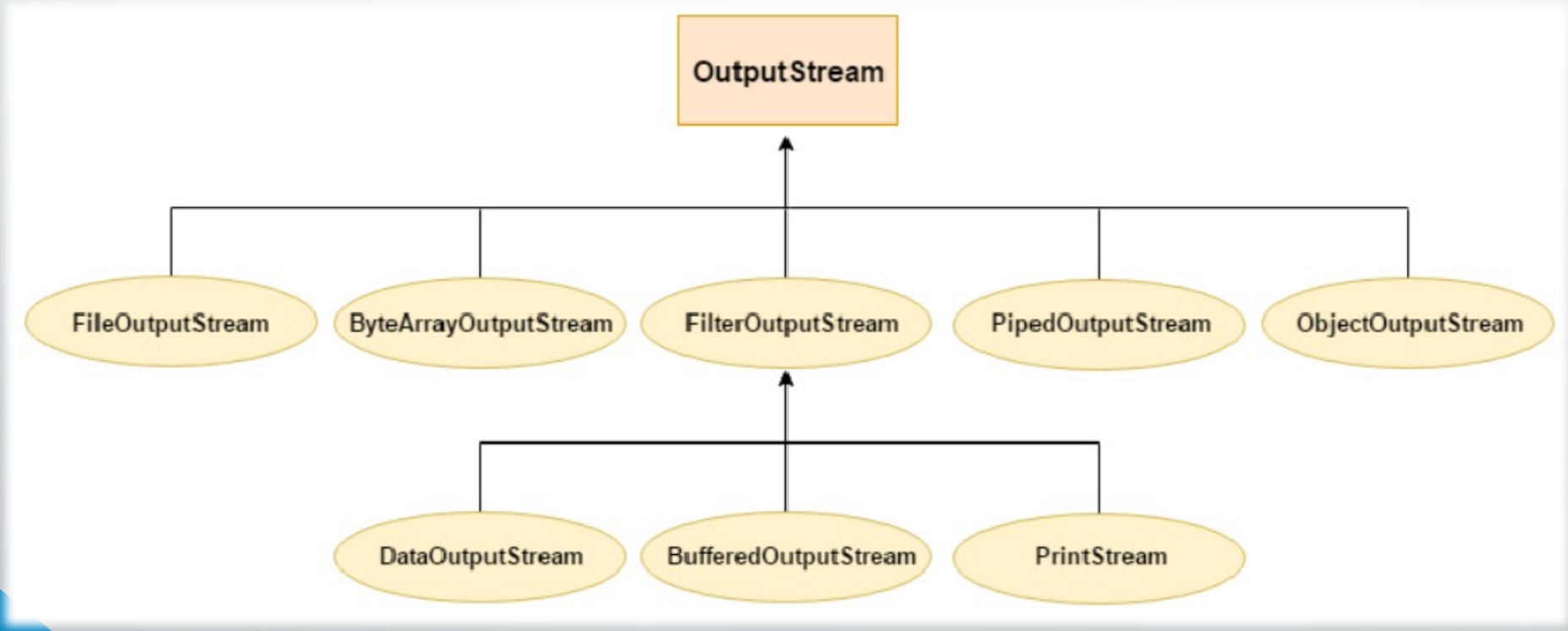
Returns an estimate of the number of bytes that can be read from the current input stream.

3) `public void close()throws IOException:`

Is used to close the current input stream.

OutputStream Class

- OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes.
- An output stream accepts output bytes and sends them to some sink.



Useful Methods of OutputStream

- Some of the most used methods in the OutputStream class are:

1) `public void write(int)throws IOException:`

   Is used to write a byte to the current output stream.

2) `public void write(byte[])throws IOException:`

   Is used to write an array of bytes to the current output stream.

3) `public void flush()throws IOException:`

   Flushes the current output stream.

4) `public void close()throws IOException:`

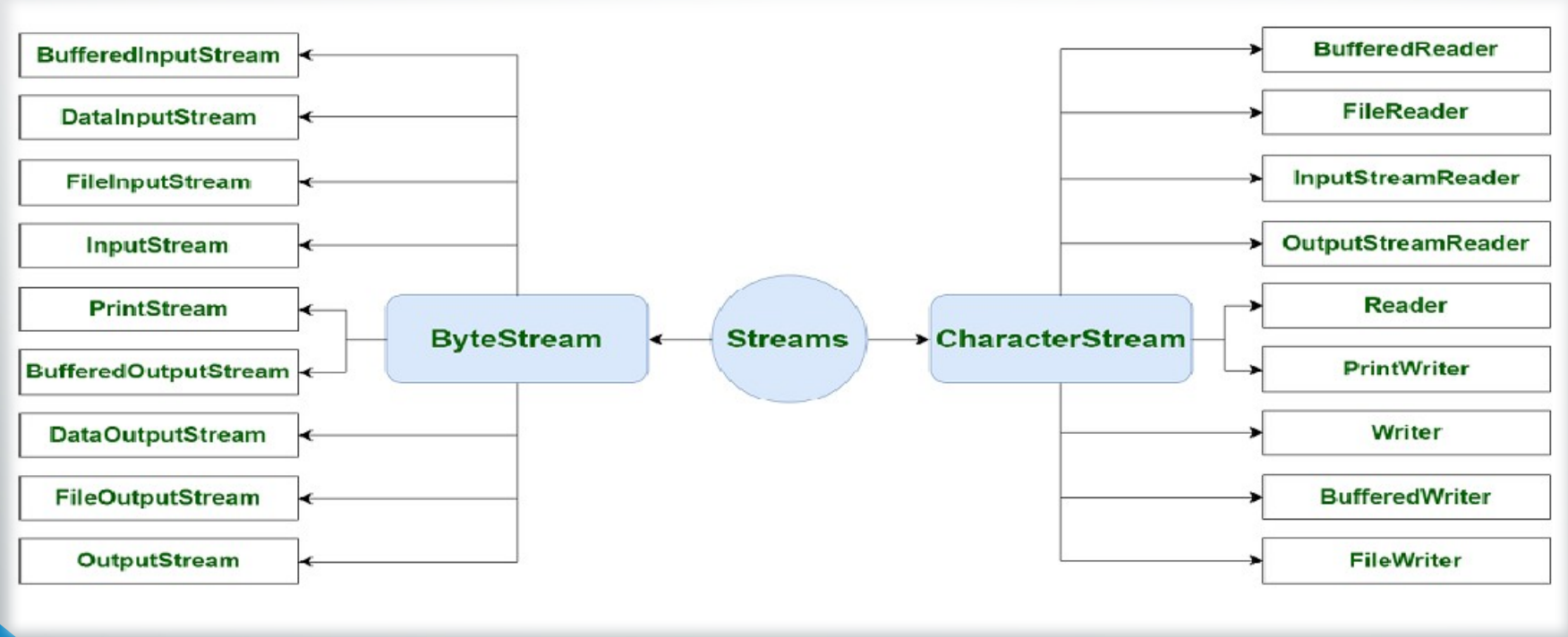
   Is used to close the current output stream.

InputStream/OutputStream (Example)

```
import java.io.*;
class StreamExample {
   public static void main(String[] args) throws IOException {
      String inputFile = "input.txt";   //Source file path
      String outputFile = "output.txt"; //Destination file path
      InputStream inputStream = new FileInputStream(inputFile);
      OutputStream outputStream = new FileOutputStream(outputFile);
      byte[] buffer = new byte[1024];   //Buffer to hold data chunks during reading
      int bytesRead;
      //Reading from input and writing to output
      while((bytesRead = inputStream.read(buffer)) != -1){
          outputStream.write(buffer, 0, bytesRead); //Write only the bytes read
      }
      System.out.println("Data copied successfully from " + inputFile + " to " + outputFile);
      outputStream.flush(); //Using flush to ensure all data is written
      //Closing streams
      inputStream.close();
      outputStream.close();
   }
}
```

ByteStream and CharacterStream

- Depending on the types of file, streams can be divided into two primary classes which can be further divided into other classes
  - ByteStream
  - CharacterStream



ByteStream

- **ByteStream** is used to process data byte by byte (8 bits).

Classes	Description	Classes	Description
BufferedInputStream	It is used for Buffered Input Stream.	**BufferedOutputStream**	It is used for Buffered Output Stream.
DataInputStream	It contains method for reading java standard datatypes.	**DataOutputStream**	It contains method for writing java standard data types.
FileInputStream	It is used to reads from a file	**FileOutputStream**	This is used to write to a file.
InputStream	It is an abstract class that describes stream input.	**OutputStream**	This is an abstract class that describes stream output.
PrintStream	It contains the most used print() and println() method		

- Though it has many classes, the **FileInputStream** and the **FileOutputStream** are the most popular ones. The **FileInputStream** is used to read from the source and **FileOutputStream** is used to write to the destination.

ByteStream (Example)

```
import java.io.*;
class ByteStreamDemo {
    public static void main(String[] args) throws IOException {
        FileInputStream sourceStream = null;
        FileOutputStream targetStream = null;
        try{
            sourceStream = new FileInputStream("sorcefile.txt");
            targetStream = new FileOutputStream("targetfile.txt");
            //Reading source file and writing content to target file byte by byte
            int temp;
            while((temp = sourceStream.read())!= -1)
                targetStream.write((byte)temp);
        }finally{
            if(sourceStream != null)
                sourceStream.close();
            if(targetStream != null)
                targetStream.close();
        }
    }
}
```

CharacterStream

- **CharacterStream** automatically allows us to read/write data character by character.

Classes	Description	Classes	Description
BufferedReader	It is used to handle buffered input stream.	**PrintWriter**	This contains the most used print() and println() method
FileReader	This is an input stream that reads from file.	**Writer**	This is an abstract class that define character stream output.
InputStreamReader	This input stream is used to translate byte to character.	**BufferedWriter**	This is used to handle buffered output streams.
OutputStreamReader	This output stream is used to translate character to byte.	**FileWriter**	This is used to output the stream that writes to the file.
Reader	This is an abstract class that defines character stream input.		

- Though it has many classes, the **FileReader** and the **FileWriter** are the most popular ones. **FileReader** and **FileWriter** are character streams used to read from the source and write to the destination, respectively.

CharacterStream (Example)

```
import java.io.*;
class CharacterStreamDemo {
    public static void main(String[] args) throws IOException {
        FileReader sourceStream = null;
        FileWriter targetStream = null;
        try{
            sourceStream = new FileReader("sorcefile.txt");
            targetStream = new FileWriter("targetfile.txt");
            //Reading source file and writing content to target file character by character
            int temp;
            while((temp = sourceStream.read())!= -1)
                targetStream.write((char)temp);
        }finally{
            if(sourceStream != null)
                sourceStream.close();
            if(targetStream != null)
                targetStream.close();
        }
    }
}
```

ByteStream vs CharacterStream

- **ByteStreams** handle data in raw bytes (8-bit data), whereas **CharacterStreams** handle data in 16-bit Unicode characters.
- **ByteStreams** are used when dealing with binary files like images, audio, and videos, whereas **CharacterStreams** are used when dealing with text files, as they handle character encoding and decoding.
- **ByteStream** classes extend InputStream and OutputStream classes, whereas **CharacterStream** classes extend Reader and Writer classes.
- **ByteStreams** are Faster for raw binary data since no encoding/decoding is involved, whereas **CharacterStreams** are slightly slower due to character encoding and decoding processes, but this is necessary for handling text data correctly.

Summary

Today, we learned about

- Java Input/Output Streams
- Useful methods in InputStream and OutputStream classes
- ByteStream classes with Example
- CharacterStream classes with Example
- Difference of ByteStream and CharacterStream classes


Thank You!