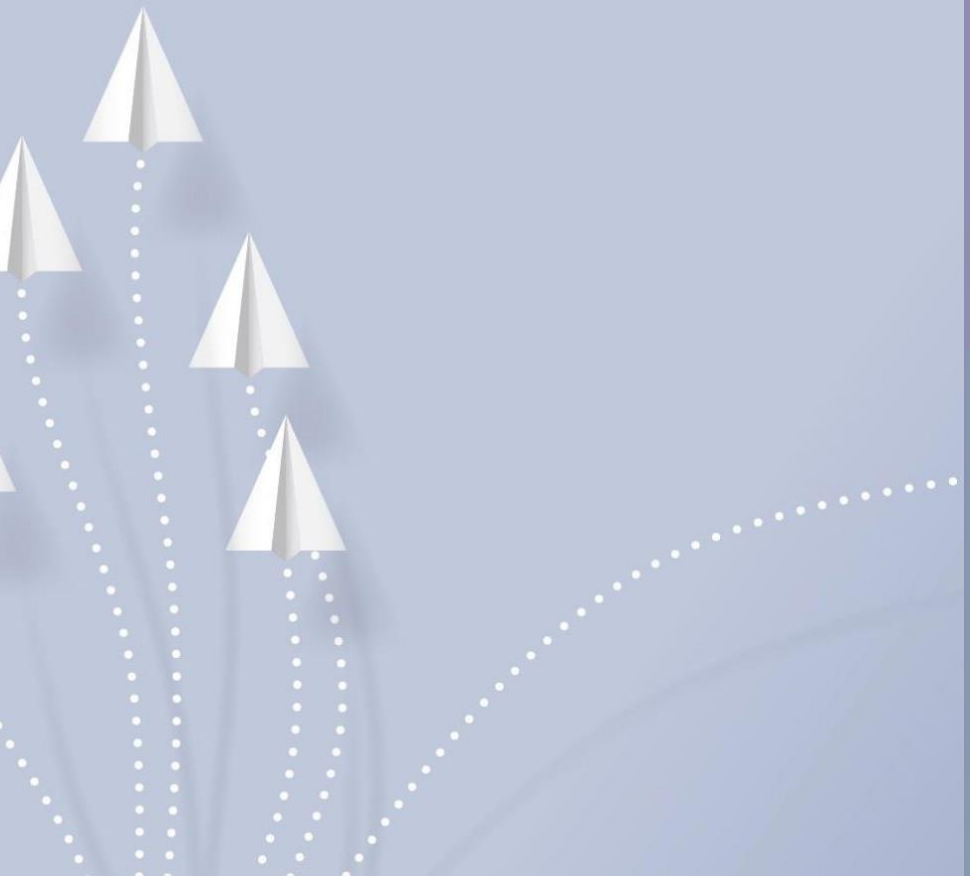
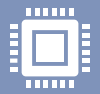


FILE HANDLING IN JAVA

JAVA I/O AND STREAM

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.

We can perform **file handling in Java** by Java I/O API.

STREAM

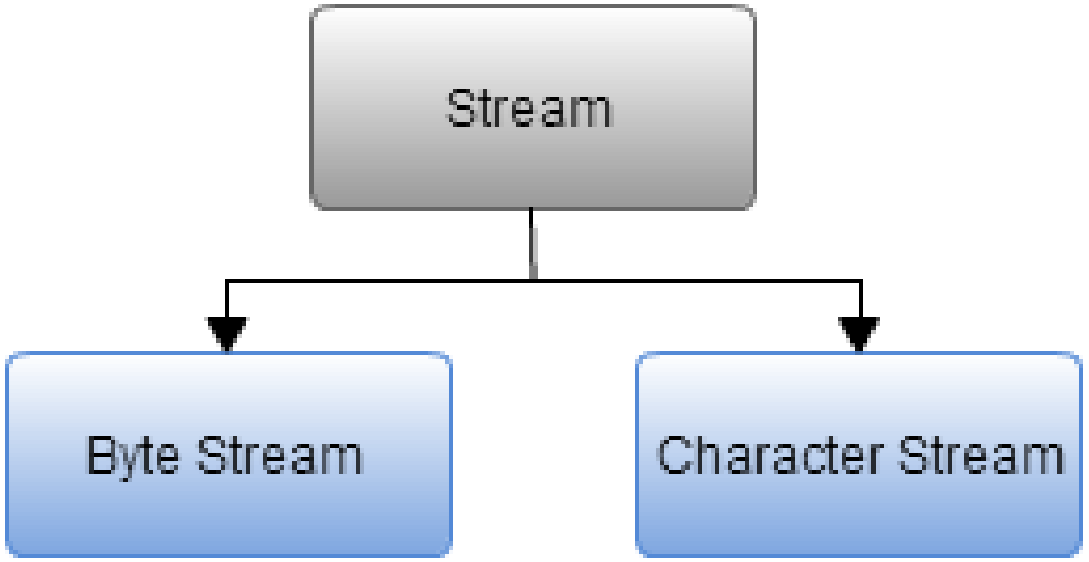
It is an ordered sequence of data that has a Source and Destination.

In Java Stream is composed of Bytes.

In Java three types of streams are automatically created for us.

- **System.out [Standard OutputStream]**
  - This is used to output the data produced by the user's program.
  - usually a computer screen is used for standard output stream and represented as **System.out.**
- **System.in [Standard InputStream]**
  - This is used to feed the data to user's program.
  - Usually a keyboard is used as standard input stream and represented as **System.in.**
- **System.error [Standard ErrorStream]**
  - This is used to output the error data produced by the user's program.
  - Usually a computer screen is used for standard error stream and represented as **System.err.**

TYPES OF STREAM



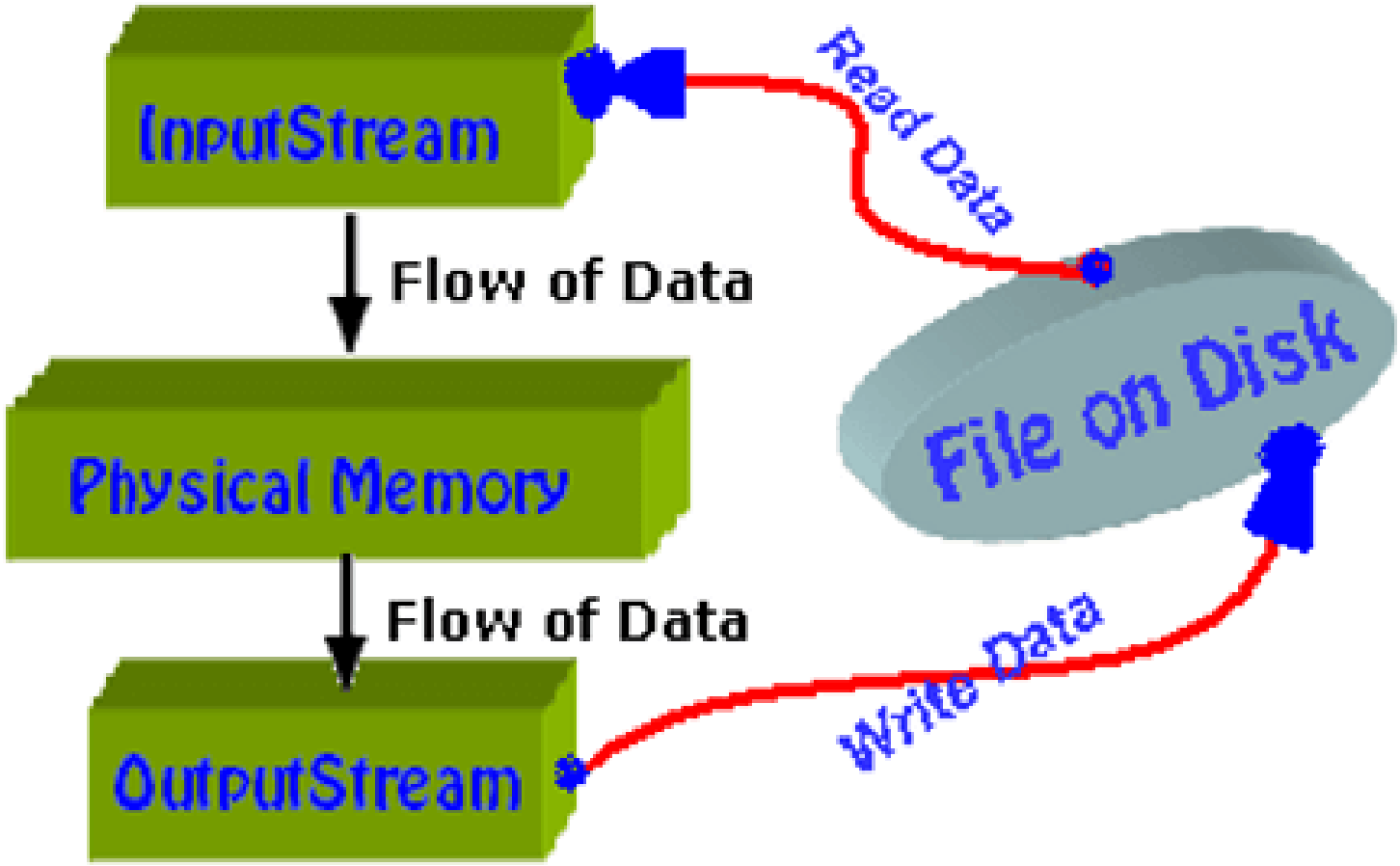
CHARACTER STREAM

- In Java, characters are stored using Unicode conventions.
- Character stream automatically allows us to read/write data character by character.
- For example FileReader and FileWriter are character streams used to read from source and write to destination.

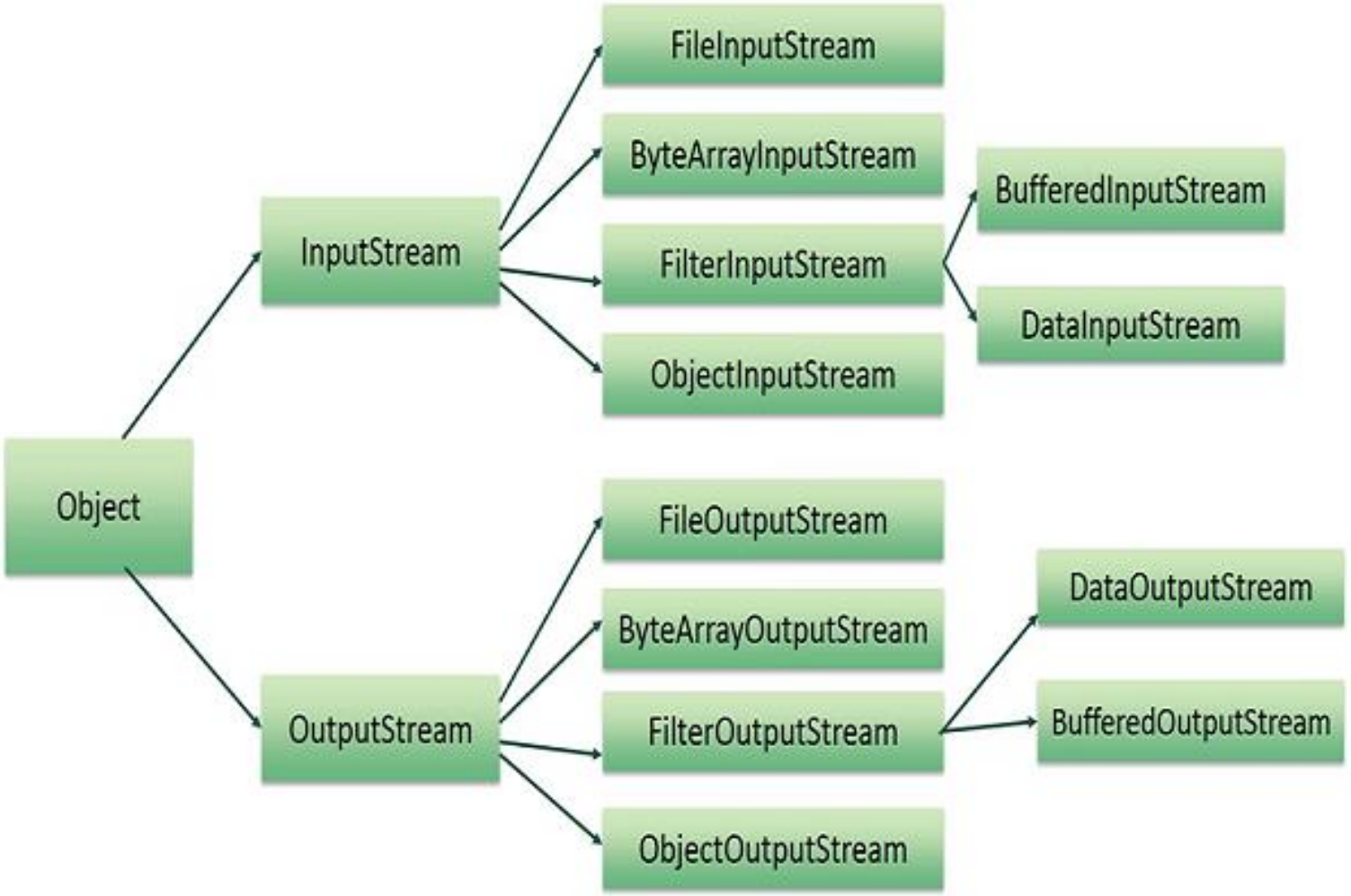
BYTE STREAM

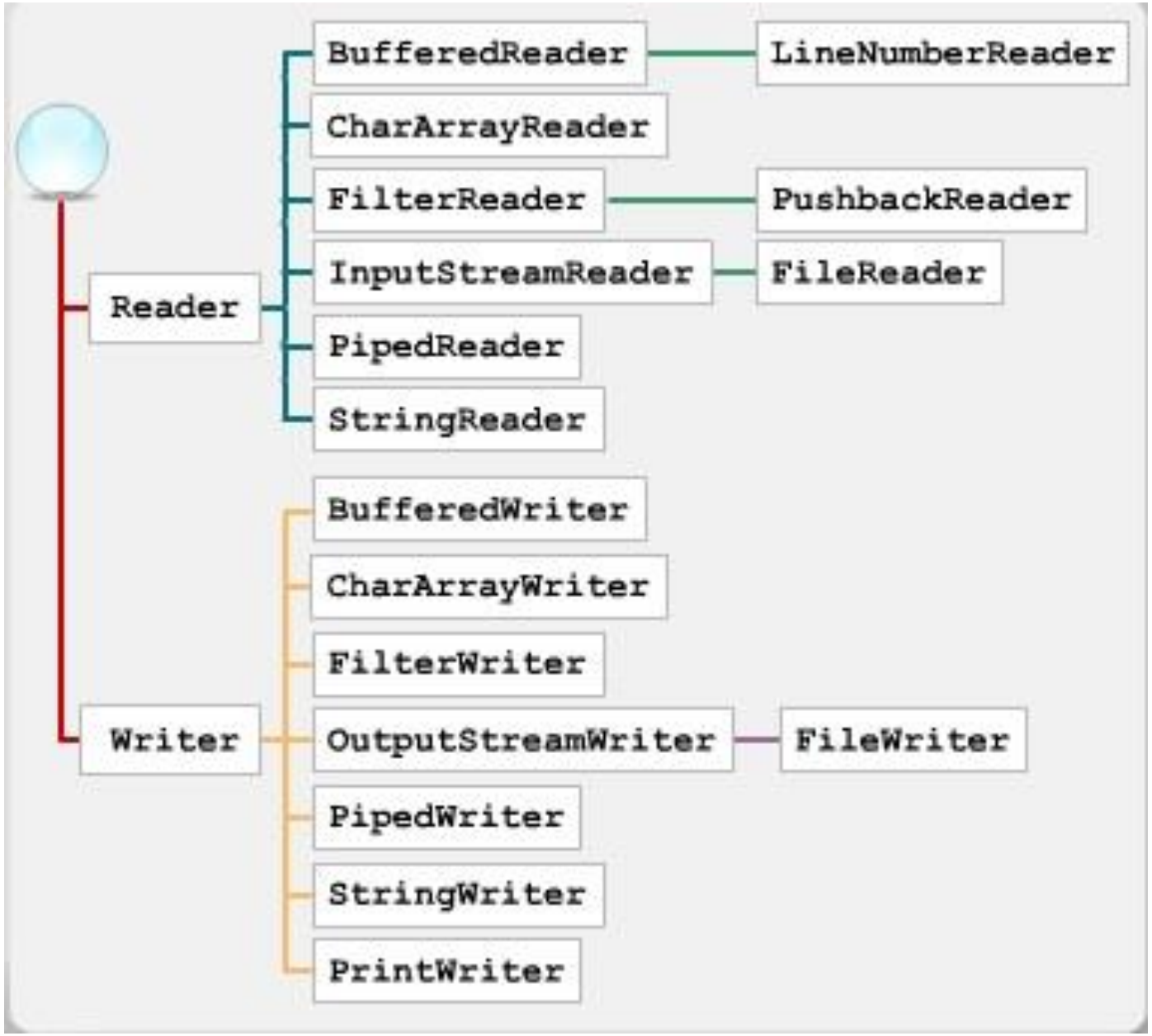
- Byte streams process data byte by byte (8 bits).
- For example FileInputStream is used to read from source and FileOutputStream to write to the destination.

HOW I/O STREAM WORKS?



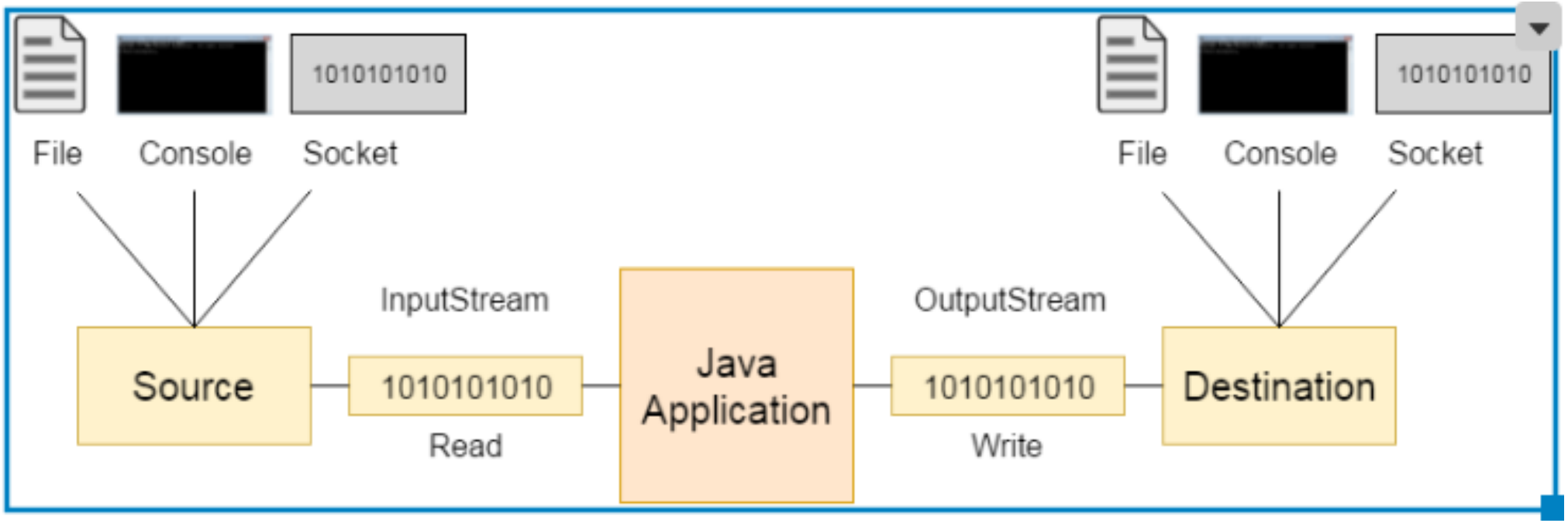
JAVA I/O STREAM CLASS HIERARCHY




Reader
BufferedReader
LineNumberReader
CharArrayReader
FilterReader
PushbackReader
InputStreamReader
FileReader
PipedReader
StringReader
Writer
BufferedWriter
CharArrayWriter
FilterWriter
OutputStreamWriter
FileWriter
PipedWriter
StringWriter
PrintWriter

INPUTSTREAM CLASS

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



METHODS

- **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.
- **public int available()throws IOException**
    - returns an estimate of the number of bytes that can be read from the current input stream.
- **public void close()throws IOException**
    - is used to close the current input stream.

OUTPUTSTREAM CLASS

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

METHODS

- **public void write(int)throws IOException**
  - is used to write a byte to the current output stream.
- **public void write(byte[])throws IOException**
  - is used to write an array of byte to the current output stream.
- **public void flush()throws IOException**
  - flushes the current output stream.
- **public void close()throws IOException**
  - is used to close the current output stream.

WRITE CONTENT INTO THE FILE

```java
import java.io.*;
import java.util.*;
class WriteFile
{
    public static void main(String[] args)
    {

        Scanner sc  = new Scanner(System.in);
        String line =sc.nextLine();
        byte[] b = line.getBytes();
        try
        {
            FileOutputStream fos = new FileOutputStream("D:/JAVA_NEW/demo.txt");
            fos.write(b);
            fos.flush();
            fos.close();
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Caught="+ e);
        }
        catch(IOException e)
        {
            System.out.println("Caught="+ e);
        }
    }
}
```

READ CONTENT INTO THE FILE

```java
import java.io.*;
class ReadFile
{
    public static void main(String []args)
    {
        FileInputStream fis=null;
        int i;
        try
        {
            fis = new FileInputStream("D:/JAVA_NEW/demo.txt");
            while((i=fis.read())!=-1)
            {
                System.out.print((char)i);
            }

            fis.close();
        }
        catch(FileNotFoundException e)
        {
            System.out.println(e.getMessage());
        }
        catch(IOException e)
        {
            System.out.println(e.getMessage());
        }


    }
}
```