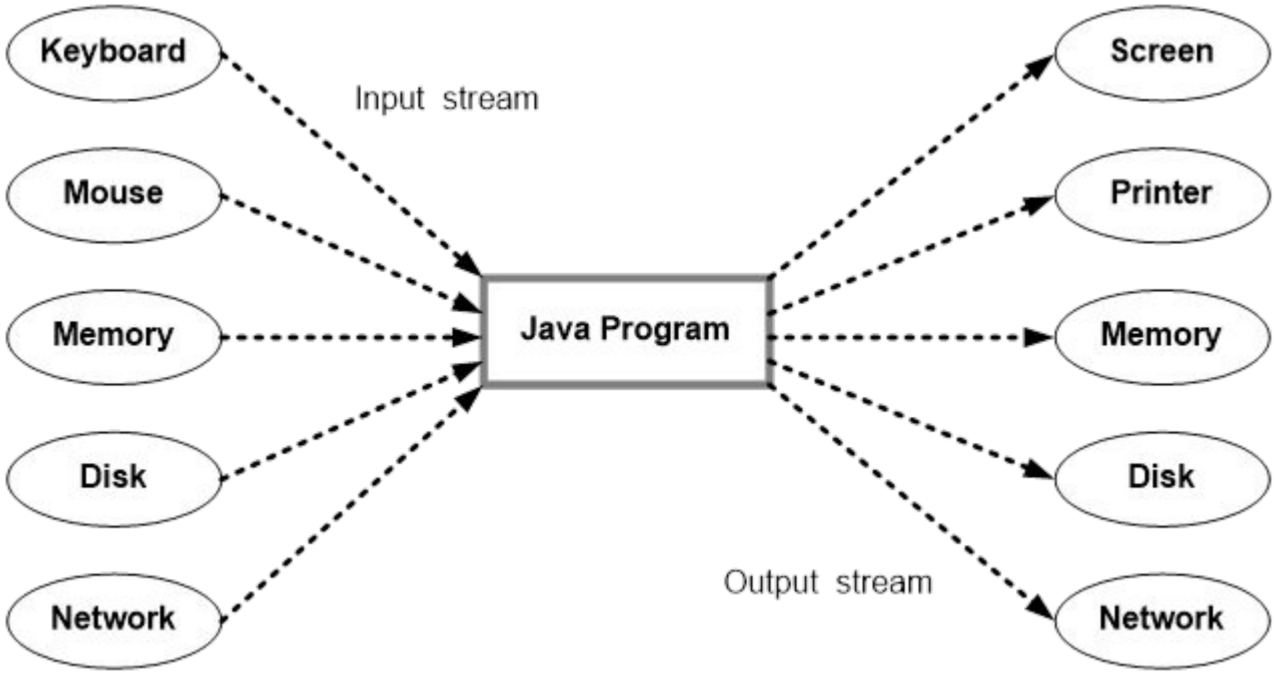
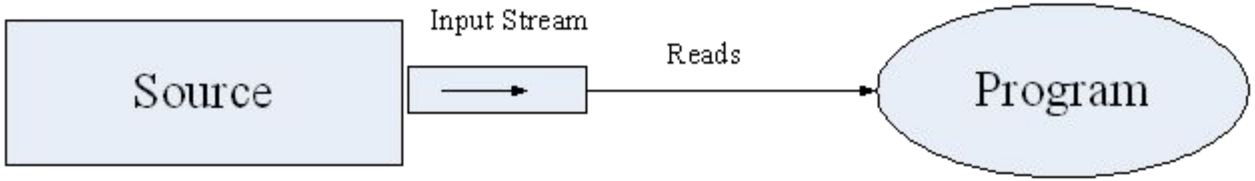
OOPs with Java

File Handling

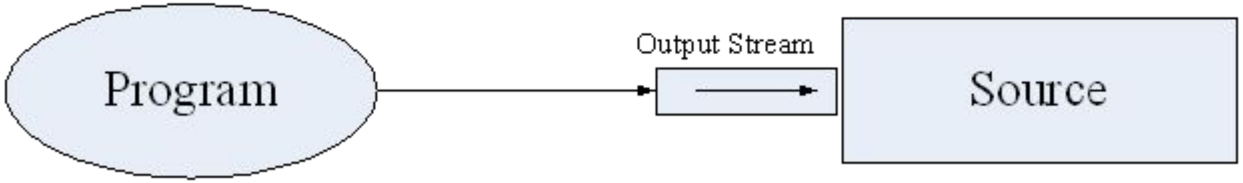
I/O in JAVA

- Java I/O (Input and Output) is used to process the input and produce the output.
- Java uses the concept of 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.


Keyboard
Mouse
Memory
Disk
Network
Input stream
Java Program
Output stream
Screen
Printer
Memory
Disk
Network



(a) Reading data into a program



(b) Writing data to a destination

Stream

- 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 with console.

    1) System.out: standard output stream

    2) System.in: standard input stream

    3) System.err: standard error stream

- The java.io package contains a large number of stream classes to support the streams
- This package provides system input and output through data streams, serialization and the file system.
    - 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.io API.

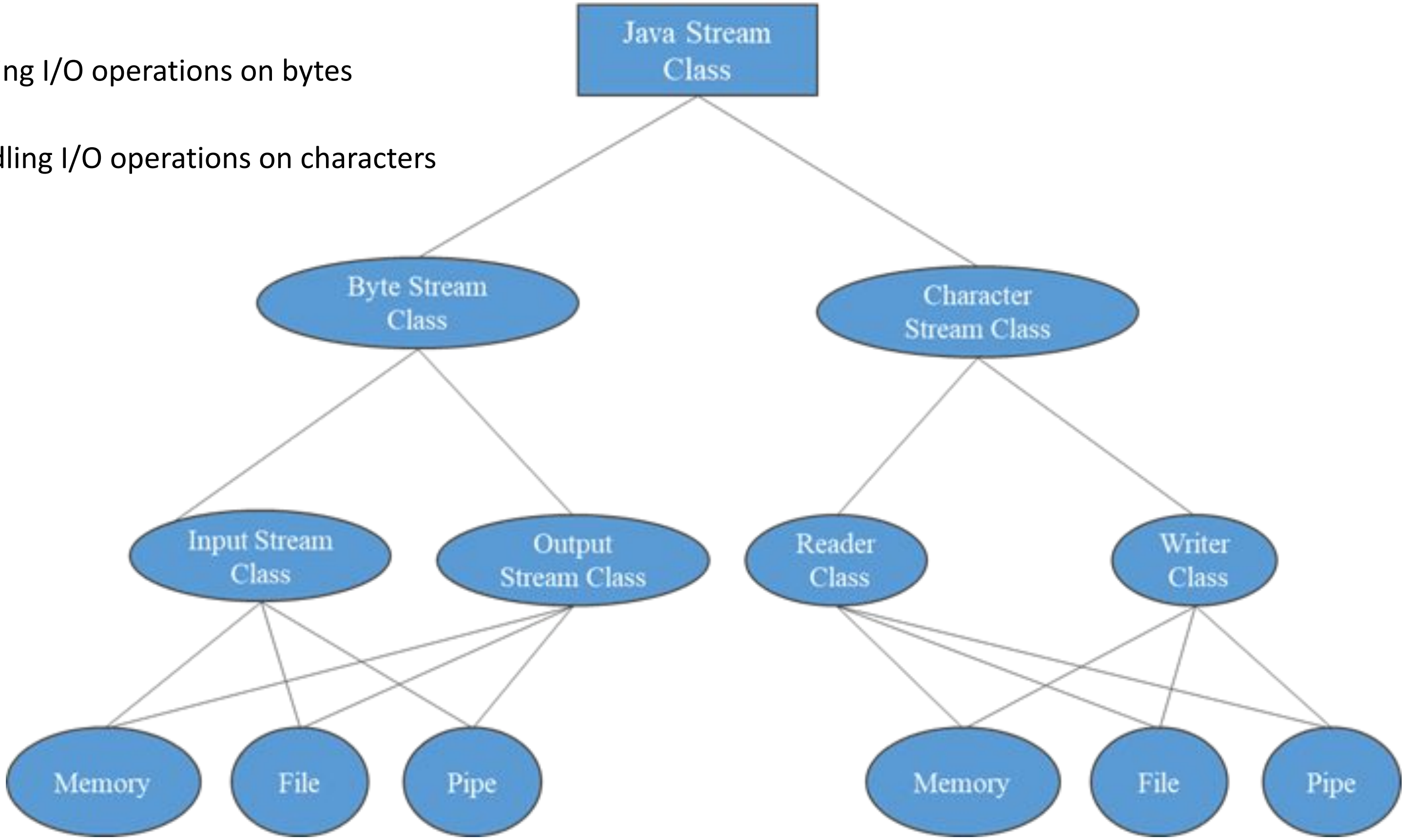
Interfaces in Package java.io

Interface	Description
Closeable	A Closeable is a source or destination of data that can be closed.
DataInput	The DataInput interface provides for reading bytes from a binary stream and reconstructing from them data in any of the Java primitive types.
DataOutput	The DataOutput interface provides for converting data from any of the Java primitive types to a series of bytes and writing these bytes to a binary stream.
Externalizable	Only the identity of the class of an Externalizable instance is written in the serialization stream and it is the responsibility of the class to save and restore the contents of its instances.
FileFilter	A filter for abstract pathnames.
FilenameFilter	Instances of classes that implement this interface are used to filter filenames.
Flushable	A Flushable is a destination of data that can be flushed.
ObjectInput	ObjectInput extends the DataInput interface to include the reading of objects.
ObjectInputValidation	Callback interface to allow validation of objects within a graph.
ObjectOutput	ObjectOutput extends the DataOutput interface to include writing of objects.
ObjectStreamConstants	Constants written into the Object Serialization Stream.
Serializable	Serializability of a class is enabled by the class implementing the java.io.Serializable interface.

I-O STREAM CLASSES IN JAVA

Java provides java.io package which contains a large number of stream classes to process all types of data:

- **Byte stream classes**
  - Support for handling I/O operations on bytes
- **Character stream classes**
  - Supports for handling I/O operations on characters



Methods in Byte Stream (input) classes

Method	Description
read( )	Read a byte from the input stream
read(byte b[ ])	Read an array of bytes into b
read(byte b[ ], int n, int m)	Reads m bytes into b starting from nth byte
available( )	Gives number of bytes available in the input
skip(n)	Skips over n bytes from the input stream
reset( )	Goes back to the beginning of the stream
close( )	Close the input steam

DataInputStream

readShort( )	**readDouble( )**
readInt( )	readLine( )
readLong( )	readChar( )
readFloat( )	readBoolean( )
readUTF( )	

Methods in Byte Stream (output) classes

Method	Description
write (int b)	Write a byte to the output stream
write (byte b[ ])	Write all bytes in the array b to the output steam
write (byte b[ ], int n, int m)	Write m bytes from array b starting from nth byte
close( )	Close the output stream
flush( )	Flushes the output stream

Methods

writeShort( )	**writeDouble( )**
writeInt( )	**writeLine( )**
writeLong( )	**writeChar( )**
writeFloat( )	**WriteBoolean( )**
writeUTF( )	

Methods in Character Stream (reader) classes

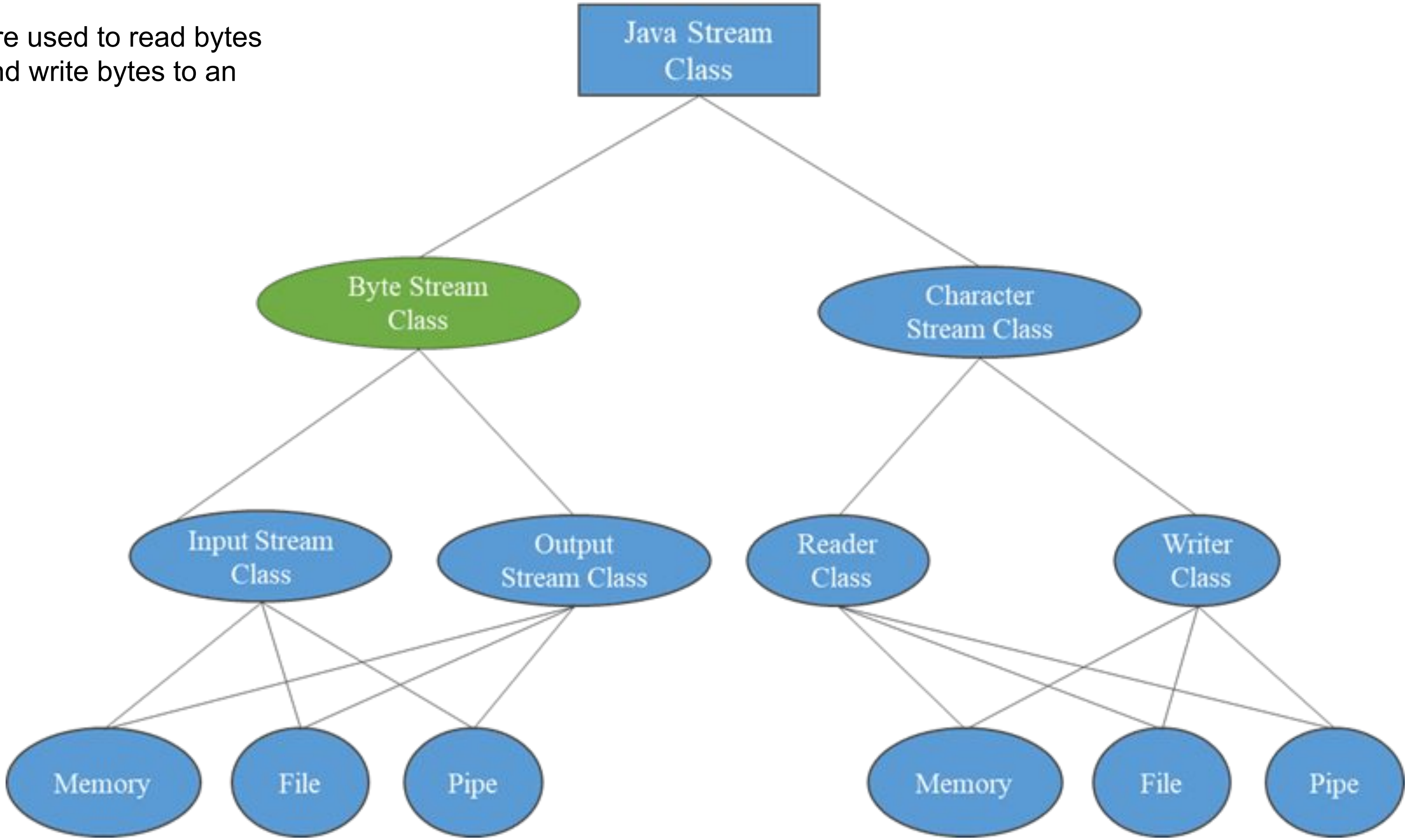
Method	Description
close()	Closes the stream and releases any system resources associated with it.
mark(int readAheadLimit)	Tells whether this stream supports the mark() operation.
markSupported()	Tells whether this stream supports the mark() operation.
read()	Reads a single character.
read(char[] cbuf)	Reads characters into an array.
read(char[] cbuf, int off, int len)	Reads characters into a portion of an array.
read(CharBuffer target)	Attempts to read characters into the specified character buffer.
ready()	Tells whether this stream is ready to be read.
reset()	Resets the stream.
skip(long n)	Skips characters.

Methods in Character Stream (Writer) classes

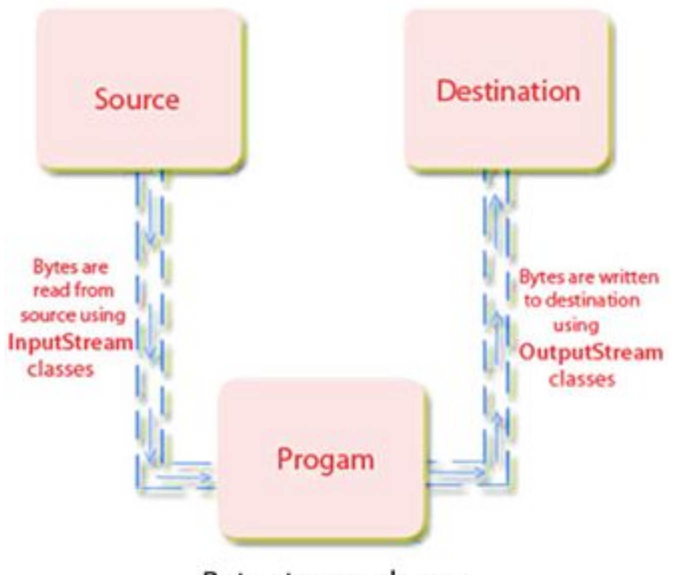
Method	Description
append(char c)	Appends the specified character to this writer.
append(CharSequence csq)	Appends the specified character sequence to this writer.
append(CharSequence csq, int start, int end)	Appends a subsequence of the specified character sequence to this writer.
close()	Closes the stream, flushing it first.
flush()	Flushes the stream.
write(char[] cbuf)	Writes an array of characters.
write(char[] cbuf, int off, int len)	Writes a portion of an array of characters.
write(int c)	Writes a single character.
write(String str)	Writes a string.
write(String str, int off, int len)	Writes a portion of a string.

IO WITH BYTE STREAMS

Byte Stream Classes are used to read bytes from an input stream and write bytes to an output stream.



Working of byte stream class


Source
Destination
Bytes are read from source using InputStream classes
Bytes are written to destination using OutputStream classes
Progam

Byte stream classes

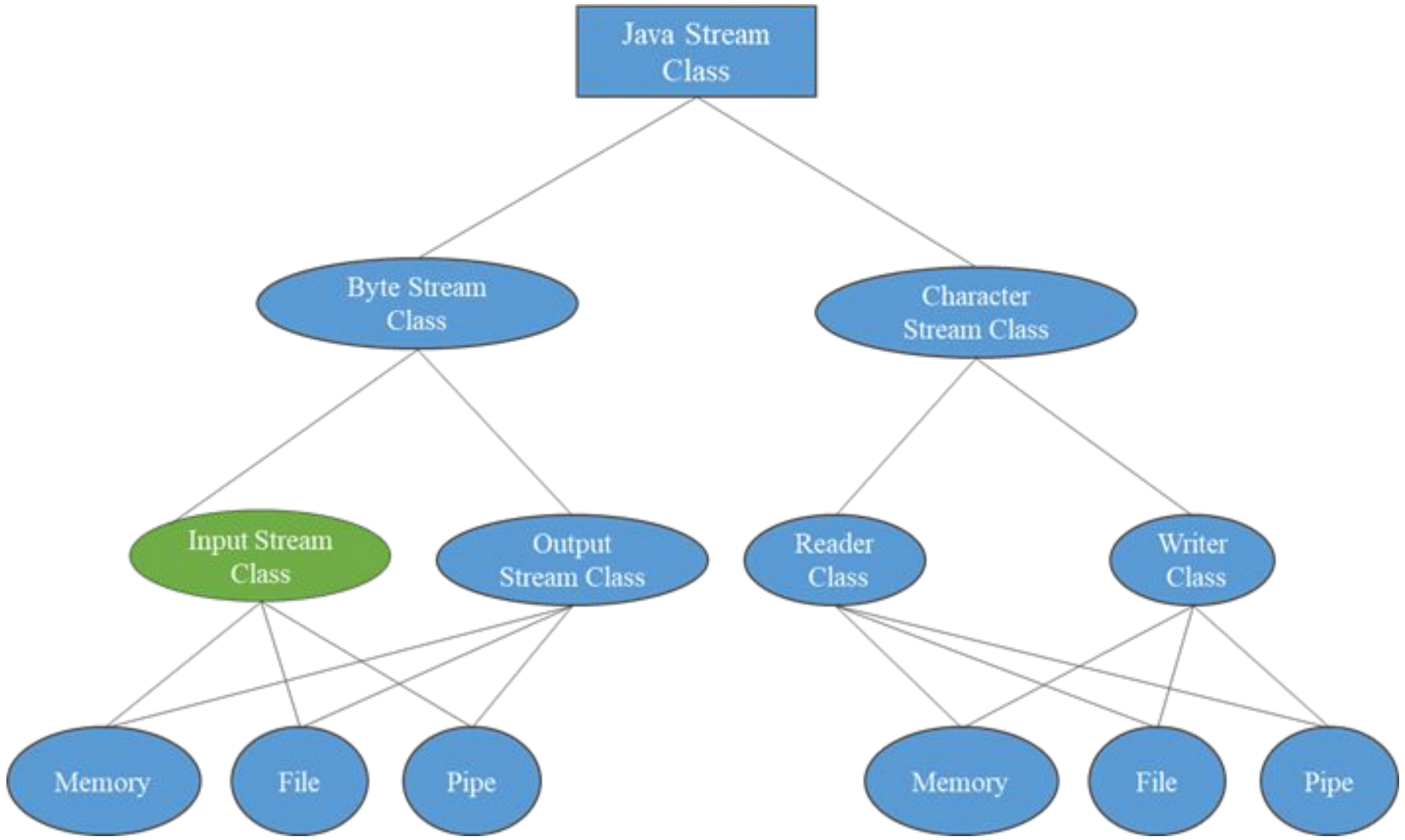
Byte Stream classes are divided in two groups:

- **InputStream** classes: These classes are subclasses of an abstract class InputStream and they are used to read bytes from a source (file, memory, or console).
- **OutputStream** classes: These classes are subclasses of an abstract class OutputStream and they are used to write bytes to a destination (file, memory or console).

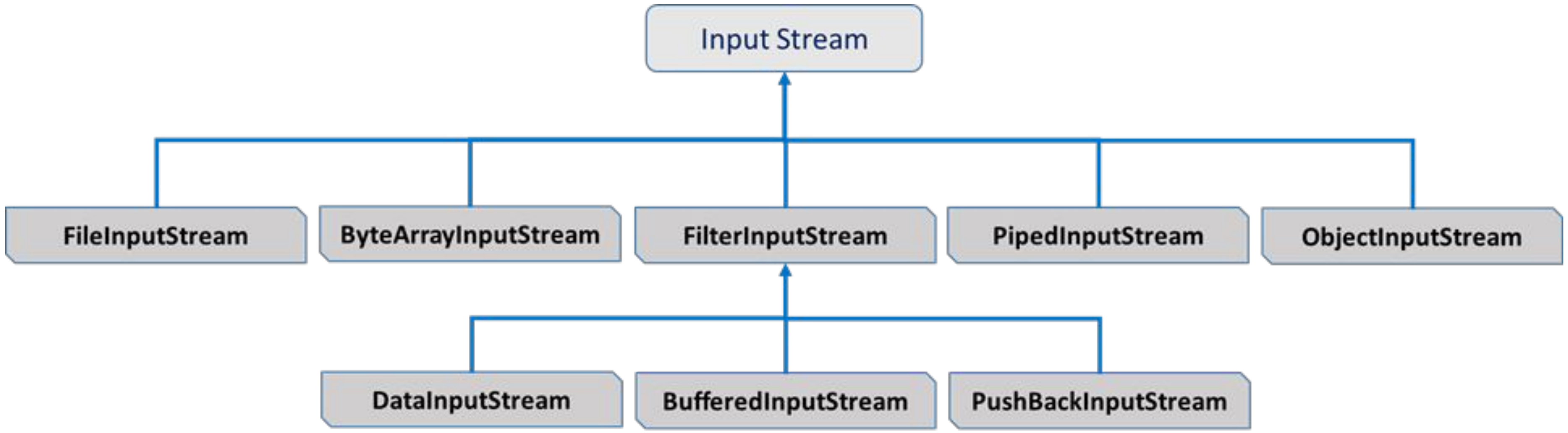
Notes:

- Programs use byte streams to perform input and output **of 8-bit bytes**.
- All byte stream classes are descended from InputStream and OutputStream.
- There are many byte stream classes. To demonstrate how byte streams work, we'll focus on the file I/O byte streams, FileInputStream and FileOutputStream.
- Other kinds of byte streams are used in much the same way; they differ mainly in the way they are constructed.

JAVA INPUT STREAM CLASSES



Stream



Note:

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

public abstract class InputStream **extends** Object **implements** Closeable

Classes under InputStream class

Class	Description
BufferedInputStream	A BufferedInputStream adds functionality to another input stream-namely, the ability to buffer the input and to support the mark and reset methods.
ByteArrayInputStream	A ByteArrayInputStream contains an internal buffer that contains bytes that may be read from the stream.
DataInputStream	A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way.
FileInputStream	A FileInputStream obtains input bytes from a file in a file system.
PipedInputStream	A piped input stream should be connected to a piped output stream; the piped input stream then provides whatever data bytes are written to the piped output stream.
ObjectInputStream	An ObjectInputStream deserializes primitive data and objects previously written using an ObjectOutputStream.

Methods

available()	Returns an estimate of the number of bytes that can be read (or skipped over) from this input stream without blocking by the next invocation of a method for this input stream.
close()	Closes this input stream and releases any system resources associated with the stream.
mark(int readlimit)	Marks the current position in this input stream.
markSupported()	Tests if this input stream supports the mark and reset methods.
read()	Reads the next byte of data from the input stream.
read(byte[] b)	Reads some number of bytes from the input stream and stores them into the buffer array b.
read(byte[] b, int off, int len)	Reads up to len bytes of data from the input stream into an array of bytes.
reset()	Repositions this stream to the position at the time the mark method was last called on this input stream.
skip(long n)	Skips over and discards n bytes of data from this input stream.

InputStream classes is used to read 8-bit bytes and supports a number of input-related methods

- Reading bytes
- Closing streams
- Marking positions in streams
- Skipping ahead in streams
- Finding the number of bytes in stream
- and many more

Some input stream methods

DataInputStream

readShort( )	**readDouble( )**
readInt( )	**readLine( )**
readLong( )	**readChar( )**
readFloat( )	**readBoolean( )**
readUTF( )	

Reading from Keyboard

```
DataInputStream r=new
DataInputStream(System.in);
```

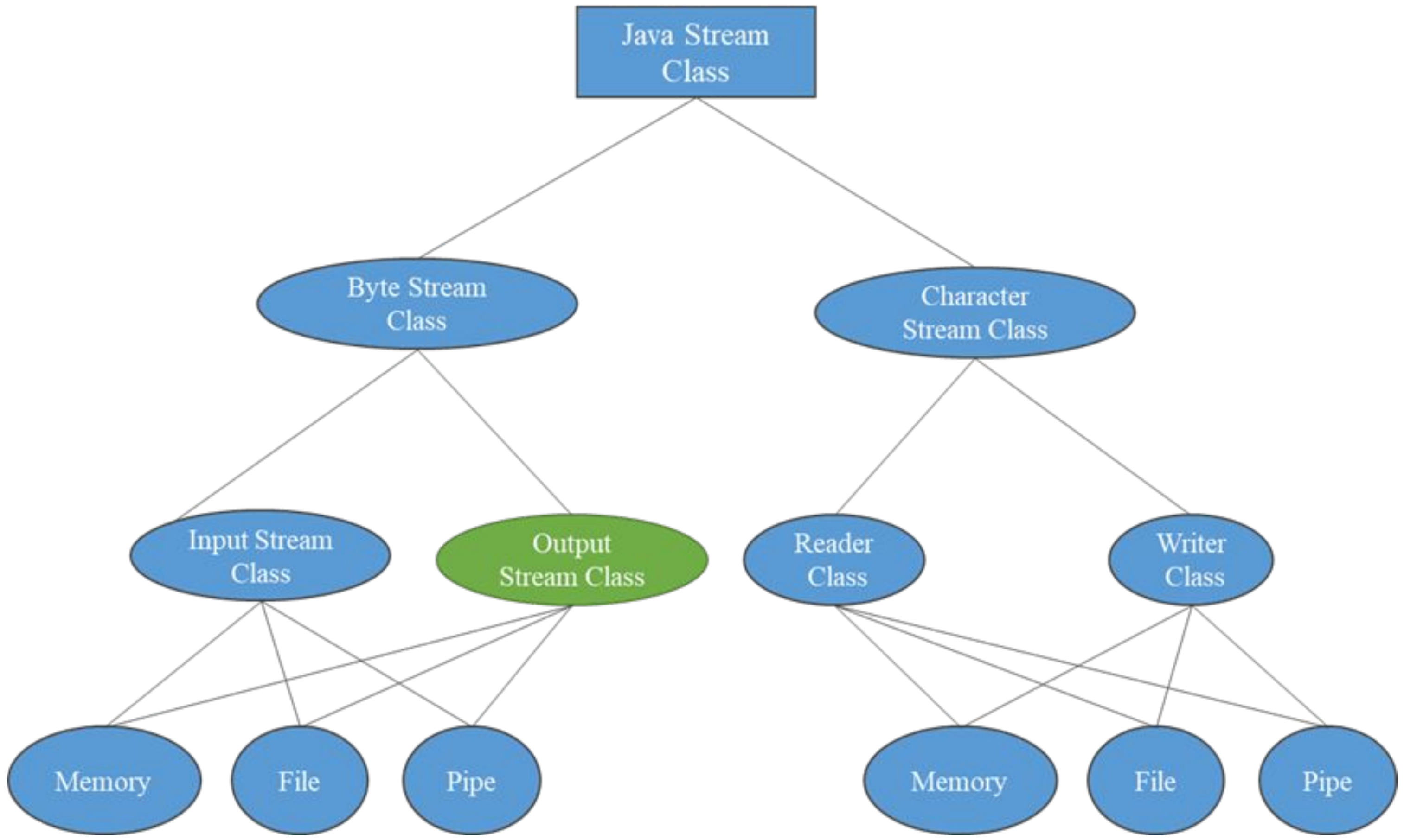
```
import java.io.*;
class G5{
public static void main(String args[])throws Exception{

InputStreamReader r=new InputStreamReader(System.in);
BufferedReader br=new BufferedReader(r);

System.out.println("Enter your name");
String name=br.readLine();
System.out.println("Welcome "+name);
 }
}
```

```
Output:Enter your name
       Amit
       Welcome Amit
```

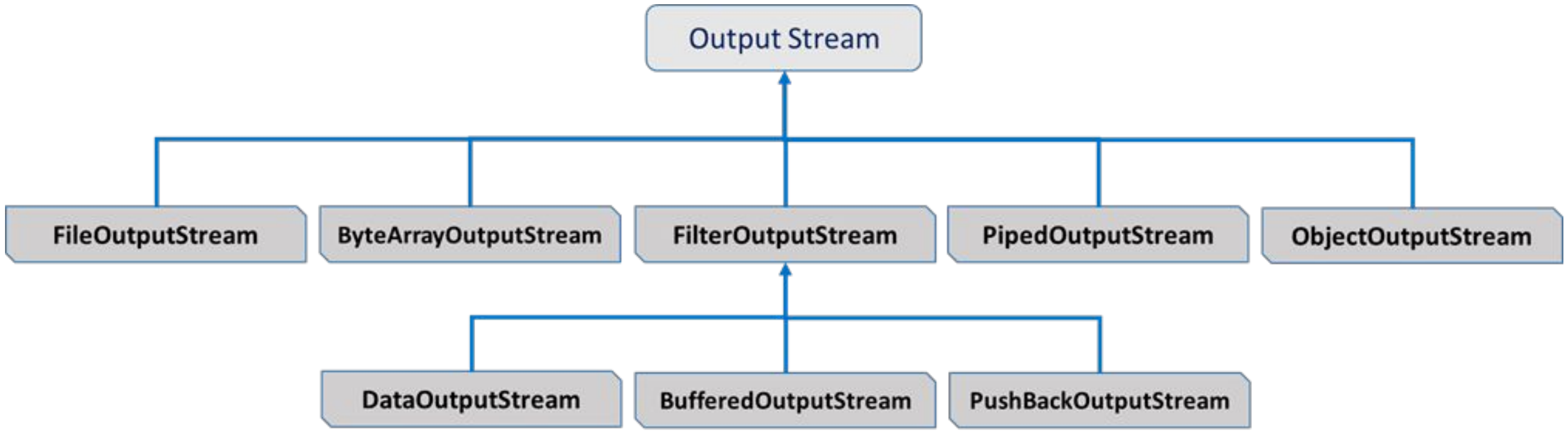
Reading bytes from a file and display data

```
import java.io.*;
public class FileInputExample {
    public static void main(String args[])throws Exception{
         FileInputStream fr=new FileInputStream("D:\\testout.txt");
         int i;
         while((i=fr.read())!=-1)
         System.out.print((char)i);
         fr.close();
    }
}
```

BYTE OUTPUT STREAM CLASSES



Hierarchy of OutputStream



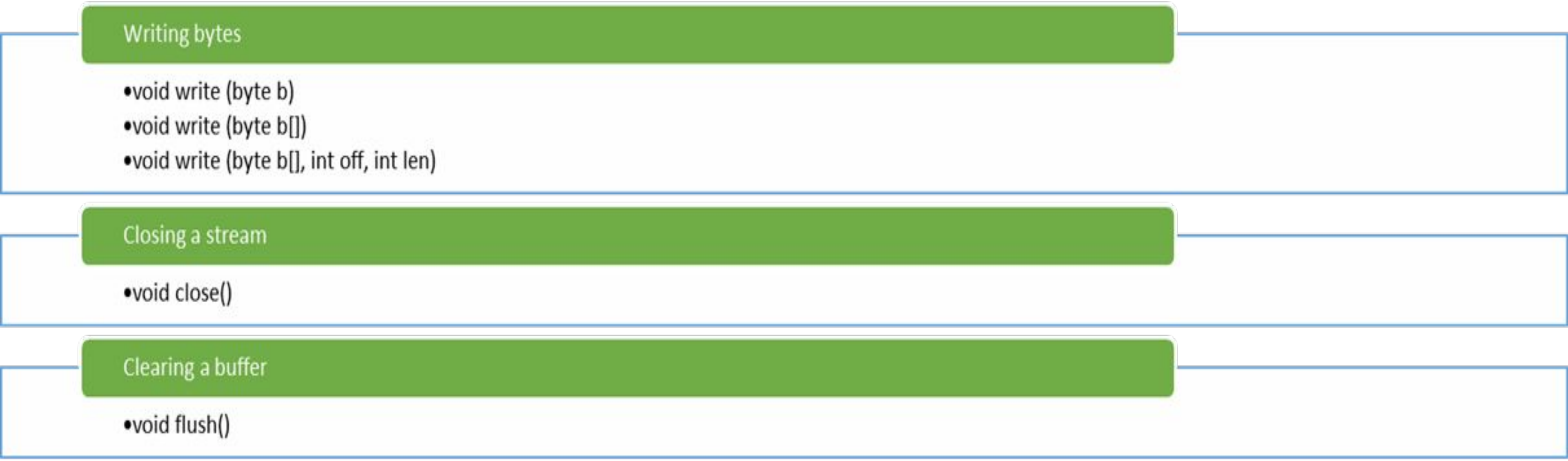
OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

```
public abstract class OutputStream extends Object
implements Closeable, Flushable
```

OutputStream classes

Class	Description
BufferedOutputStream	The class implements a buffered output stream.
ByteArrayOutputStream	This class implements an output stream in which the data is written into a byte array.
DataOutputStream	A data output stream lets an application write primitive Java data types to an output stream in a portable way.
FileOutputStream	A file output stream is an output stream for writing data to a File or to a FileDescriptor.
ObjectOutputStream	An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream.

Use of class OutputStream



Writing bytes

- void write (byte b)
- void write (byte b[])
- void write (byte b[], int off, int len)

Closing a stream

- void close()

Clearing a buffer

- void flush()

Methods of OutputStream class

Method	Description
close()	Closes this output stream and releases any system resources associated with this stream.
flush()	Flushes this output stream and forces any buffered output bytes to be written out.
write(byte[] b)	Writes b.length bytes from the specified byte array to this output stream.
write(byte[] b, int off, int len)	Writes len bytes from the specified byte array starting at offset off to this output stream.
write(int b)	Writes the specified byte to this output stream.

OutputStream classes is used to write 8-bit bytes and supports a number of input-related methods

- Writing bytes
- Closing streams
- Flushing streams
- etc.

DataOutputStream

writeShort( )	**writeDouble( )**
writeInt( )	**writeLine( )**
writeLong( )	**writeChar( )**
writeFloat( )	**WriteBoolean( )**
writeUTF( )	

Storing data into a File

```java
import java.io.FileOutputStream;
public class FileOutputStreamExample {
    public static void main(String args[]){
            try{
             FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
             fout.write(65);
             fout.close();
             System.out.println("success...");
            }catch(Exception e){System.out.println(e);}
      }
}
```

Copying bytes from one file to another

```
import java.io.*;
class IOTest{
          public void readFile(){
             try {
                    //Creating FileInputStream object.
                    FileInputStream fis =  new FileInputStream("F:\\New folder\\data1.txt");
                    //Creating FileOutputStream object.
                    FileOutputStream fos = new FileOutputStream("F:\\New folder\\data7.txt");

                    int i;
                    //read file.
                    while((i=fis.read())!=-1){
                            fos.write(i);
                    }
                    System.out.println("Content writen successfully.");
                  } catch (Exception e) {
                            e.printStackTrace();
                  }
          }
}
public class ReadWriteExample {
          public static void main(String args[]){
                  //creating IOTest object.
                  IOTest obj = new IOTest();
                  //method call.
                  obj.readFile();
          }
 }
```

Note:

- **Closing a stream** when it's no longer needed is very important.
- Always make sure that each stream variable contains an object reference before invoking close.
- **Using Bytes** is a kind of low-level I/O that should be avoided. Since for character data, the best approach is to use character streams. There are also streams for more complicated data types.
- Byte streams should only be used for the most primitive I/O.
- All other stream types are built on byte streams.