Basic Java

Unit 8 – I/O Operations

Pratian Technologies (India) Pvt. Ltd.

www.pratian.com

PRATIAN TECHNOLOGIES

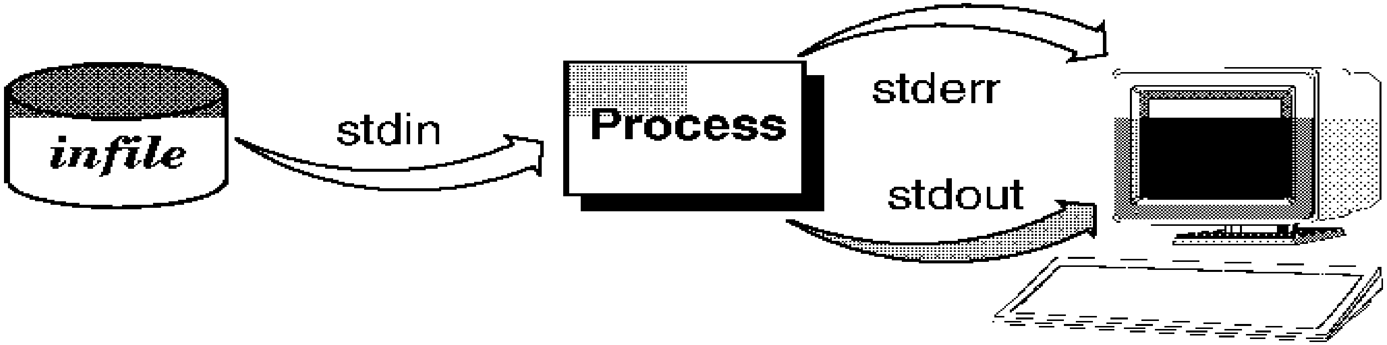
PRATIAN TECHNOLOGIES

Topics

- Overview of I/O Streams
- Data Sink and Processing Streams
- InputStream and OutputStream
- Readers and Writers
- File and File Streams
- Stream Chaining
- Filtered Streams
- DataInputStream and DataOutputStream
- PipedInputStream and PipedOutputStream
- StringReader and StringWriter
- Serialization & Object Streams

Input / Output

- Often programs need to bring in information from an external source or send out information to an external destination.



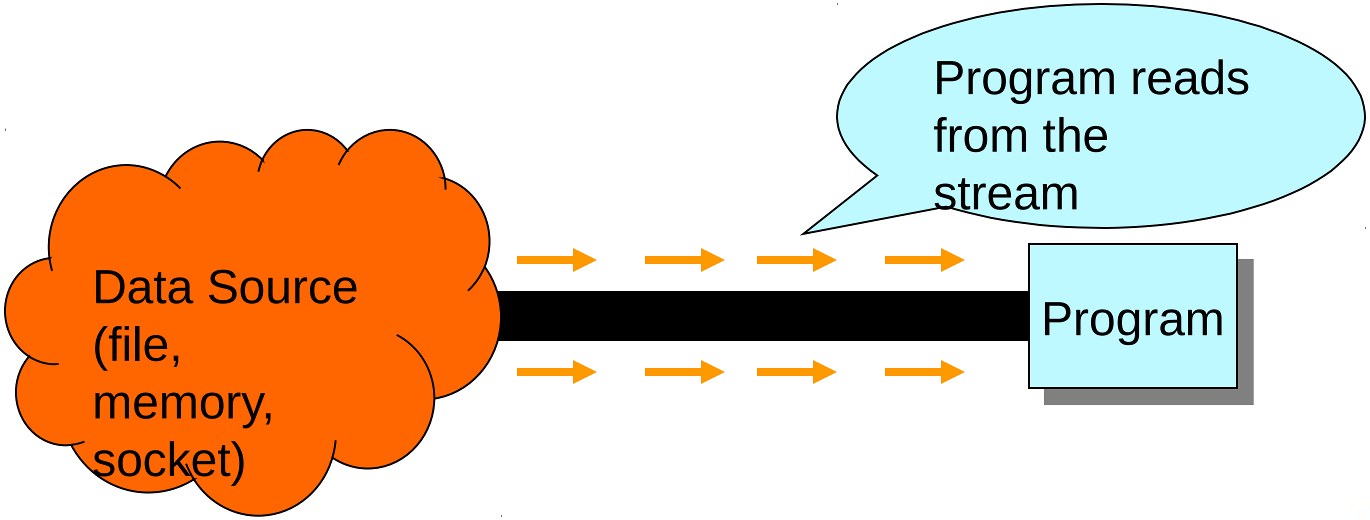
- Input and output, I/O for short, are fundamental to any computer operating system or programming language.
- The information can be anywhere: in a file, on disk, somewhere on the network, in memory, or in another program.
- **Java support for I/O is in the form of streams.**

What is a Stream?

- A stream is an ordered sequence of bytes of indeterminate length.
- Input Streams
  - Input streams move bytes of data into a Java program from some external source.
- Output Streams
  - Output streams move bytes of data from Java program to some external target.
- In some cases, streams can also move bytes from one part of Java program to another.

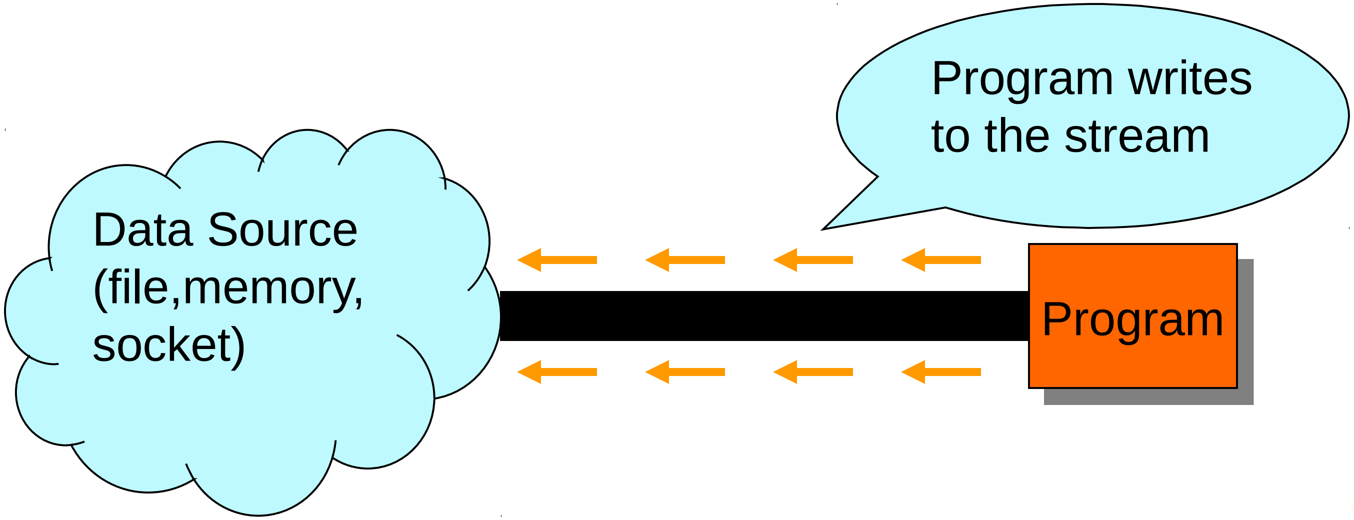
Input Stream

- To **bring in** information, a program **opens a input stream** on an information source (a file, memory, a socket) and reads the information serially, like this:


Program reads from the stream
Data Source (file, memory, socket)
Program

Output Stream

- Similarly, a program can **send information** to an external destination by **opening a output stream** to a destination and writing the information out serially, like this:


Program writes to the stream
Data Source (file,memory, socket)
Program

Java.io Package

- Java has built-in classes to support I/O and the classes are defined in the ***java.io*** package.

- The *java.io* package contains-
  - A collection of stream classes that support algorithms for reading.
  - A collection of stream classes that support algorithm for writing.

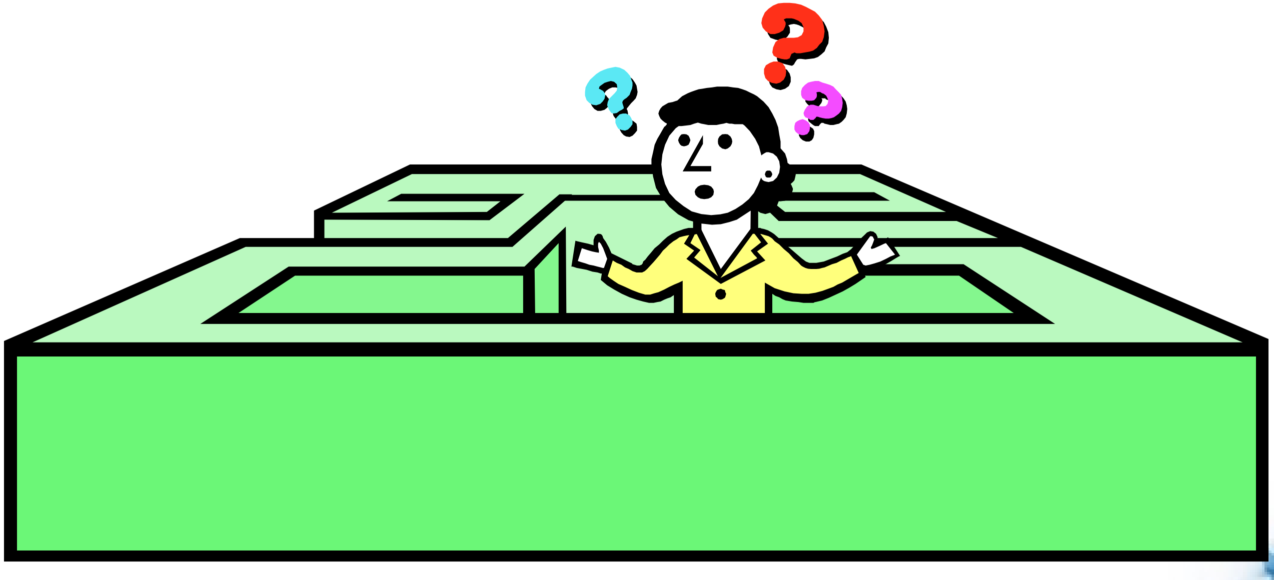
How to do I/O

```
import java.io.*;
```

1. *Open* the stream
2. *Use* the stream
3. *Close* the stream

Is Java I/O difficult?

- Java I/O is very powerful, with an overwhelming number of options
- Any given kind of I/O is not particularly difficult
- The trick is to find your way through the maze of possibilities

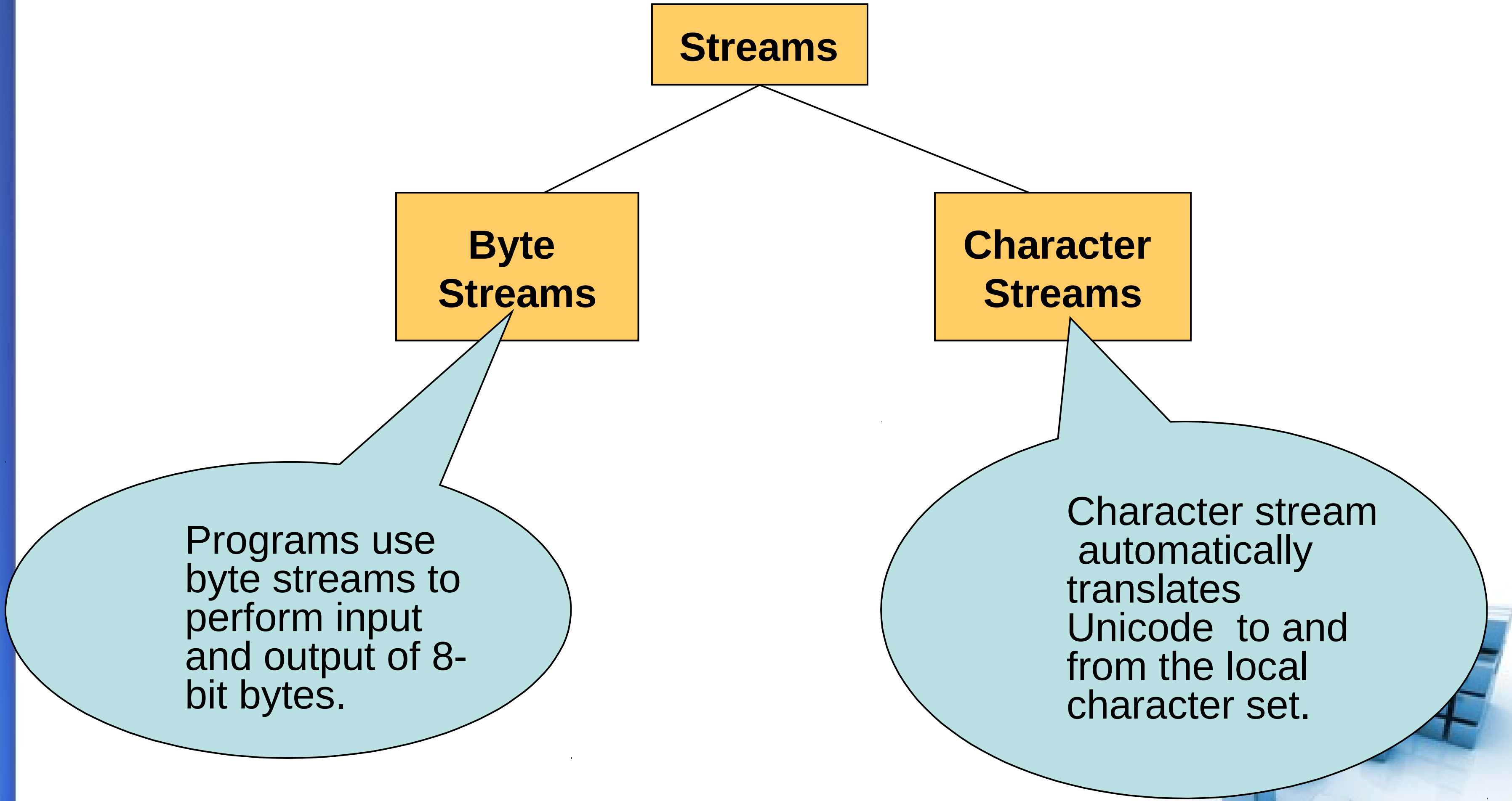


Classification of Streams Classes

- Stream Classes are classified based on
  1. The data type on which they operate.
  2. Their functionality that they provide

Types of Streams Classes

(Based on datatype)



Streams

Byte Streams

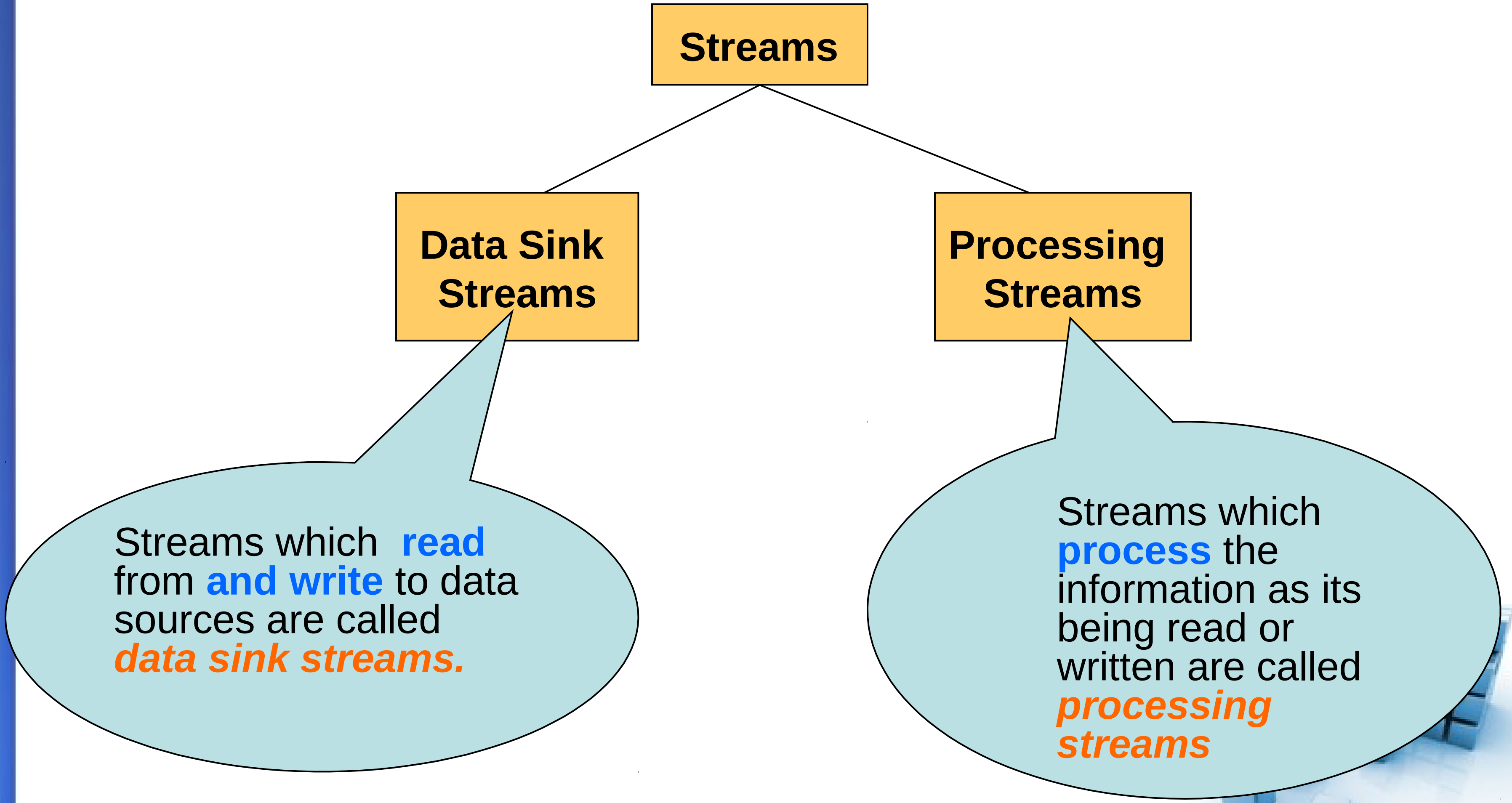
Character Streams

Programs use byte streams to perform input and output of 8-bit bytes.

Character stream automatically translates Unicode to and from the local character set.

Types of Streams Classes

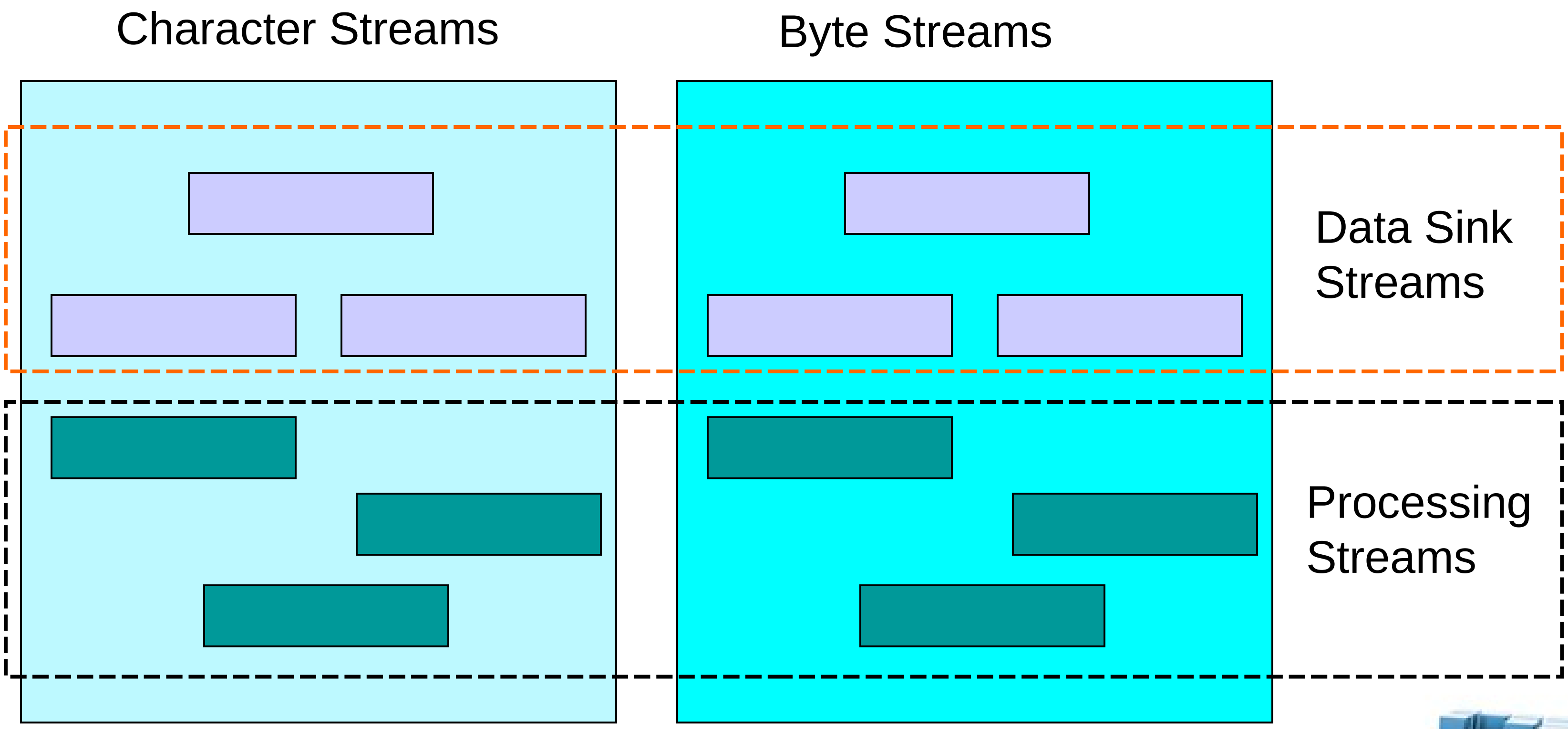
(Based on functionality)



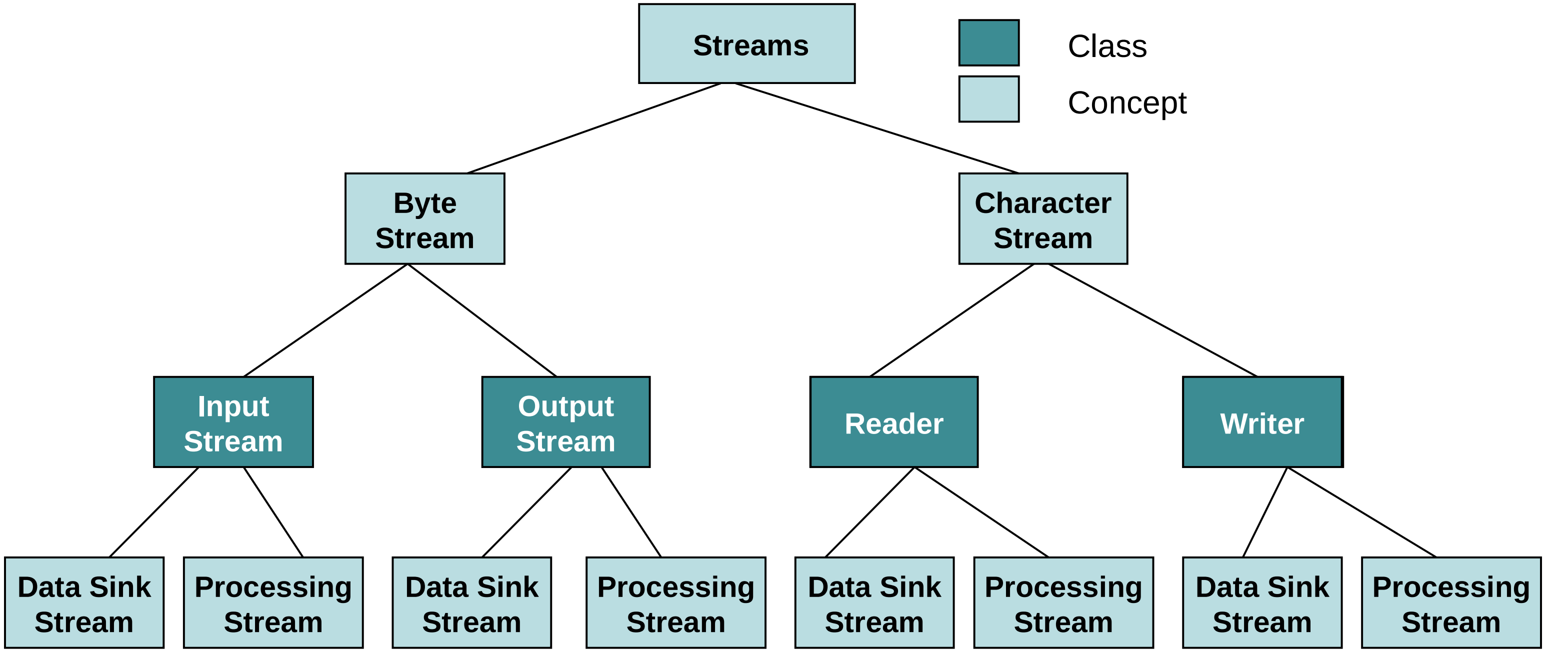
Streams which **read** from **and write** to data sources are called ***data sink streams.***

Streams which **process** the information as its being read or written are called ***processing streams***

Type of Streams Classes



Classifications Interlinked



Streams

■ Class

□ Concept

Byte Stream

Character Stream

Input Stream

Output Stream

Reader

Writer

Data Sink Stream

Processing Stream

Data Sink Stream

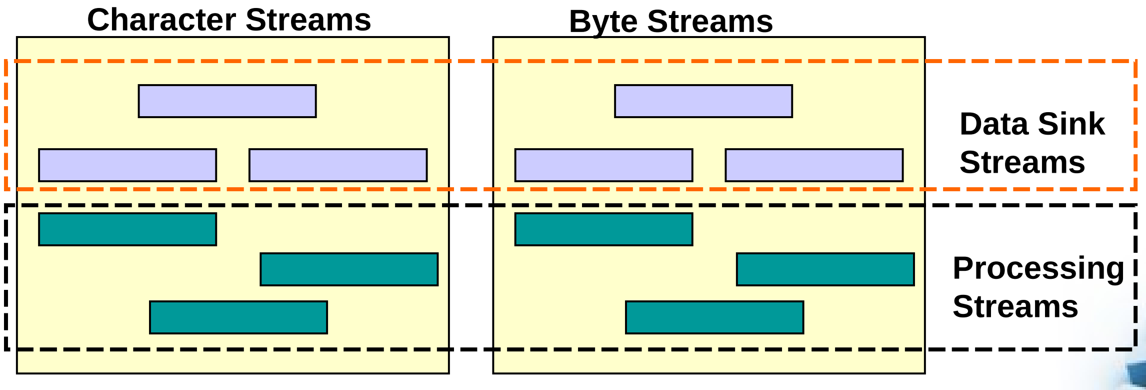
Processing Stream

Data Sink Stream

Processing Stream

Data Sink Stream

Processing Stream



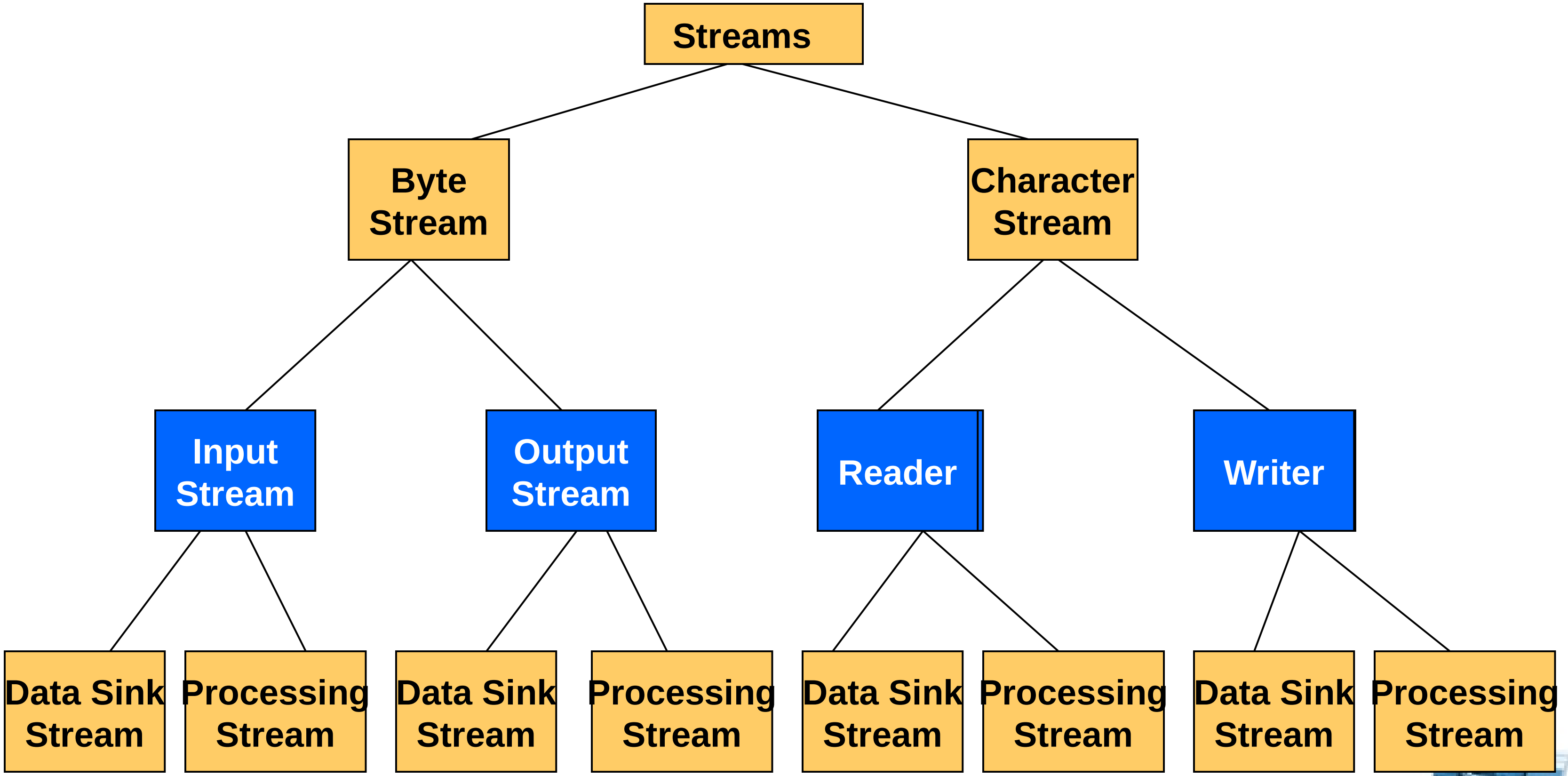
Character Streams

Byte Streams

Data Sink Streams

Processing Streams

Types of Streams Classes



Streams

Byte Stream

Character Stream

Input Stream

Output Stream

Reader

Writer

Data Sink Stream

Processing Stream

Data Sink Stream

Processing Stream

Data Sink Stream

Processing Stream

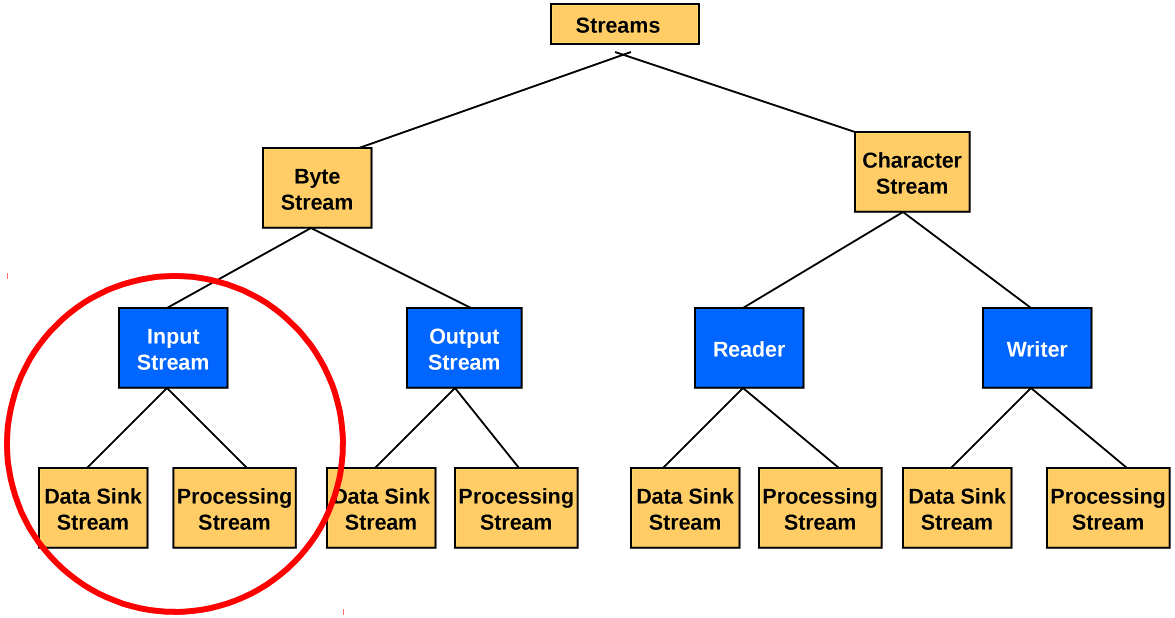
Data Sink Stream

Processing Stream

🟦 Class

🟨 Concept

Input Stream Classes

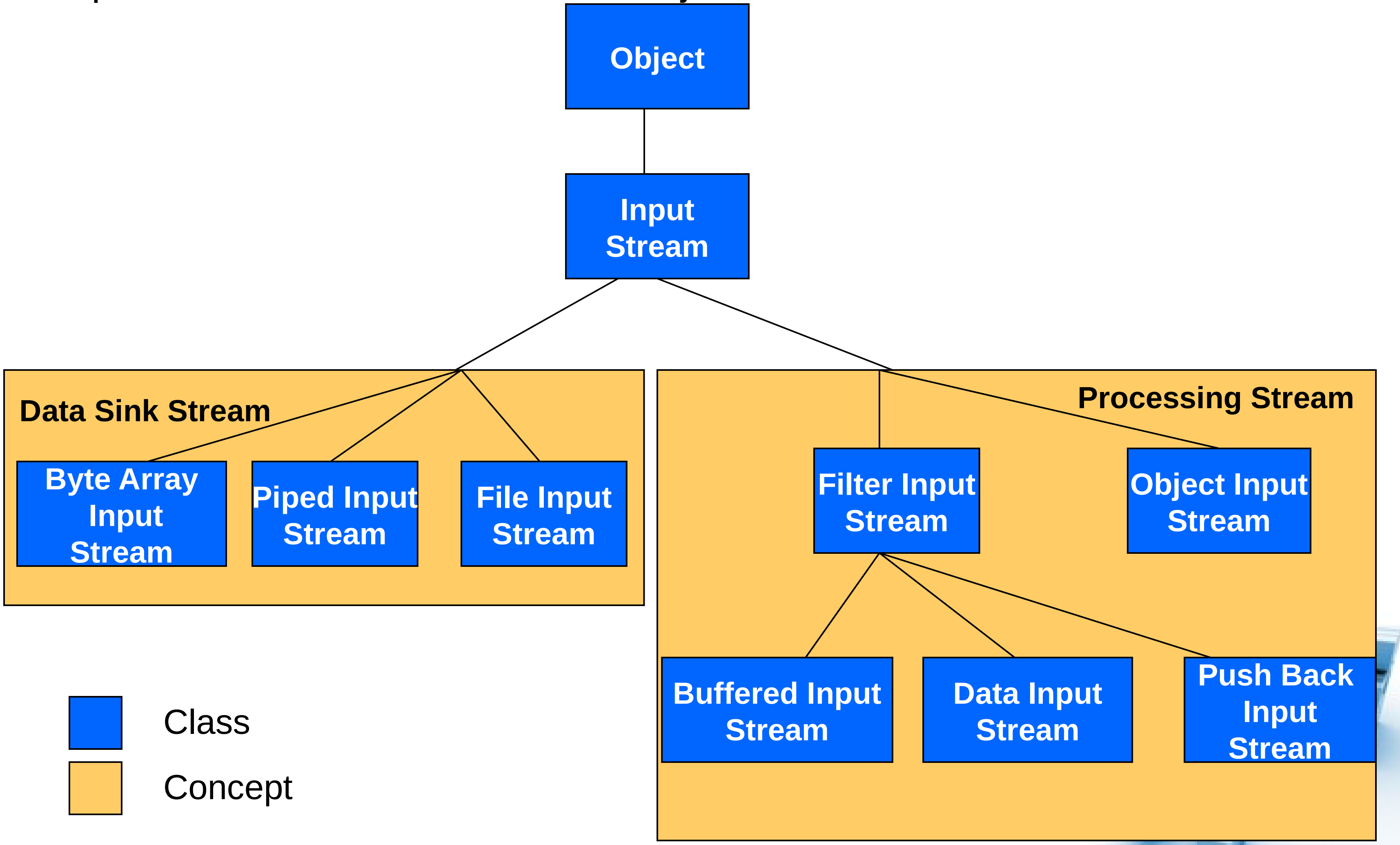


🟦 Class

🟨 Concept

Input Stream Classes

- Java's Input Stream hierarchy.
- ***InputStream*** is the abstract base class for all input stream classes
- InputStream Class extends from Object



Object

Input Stream

Data Sink Stream

Byte Array Input Stream

Piped Input Stream

File Input Stream

Processing Stream

Filter Input Stream

Object Input Stream

Buffered Input Stream

Data Input Stream

Push Back Input Stream

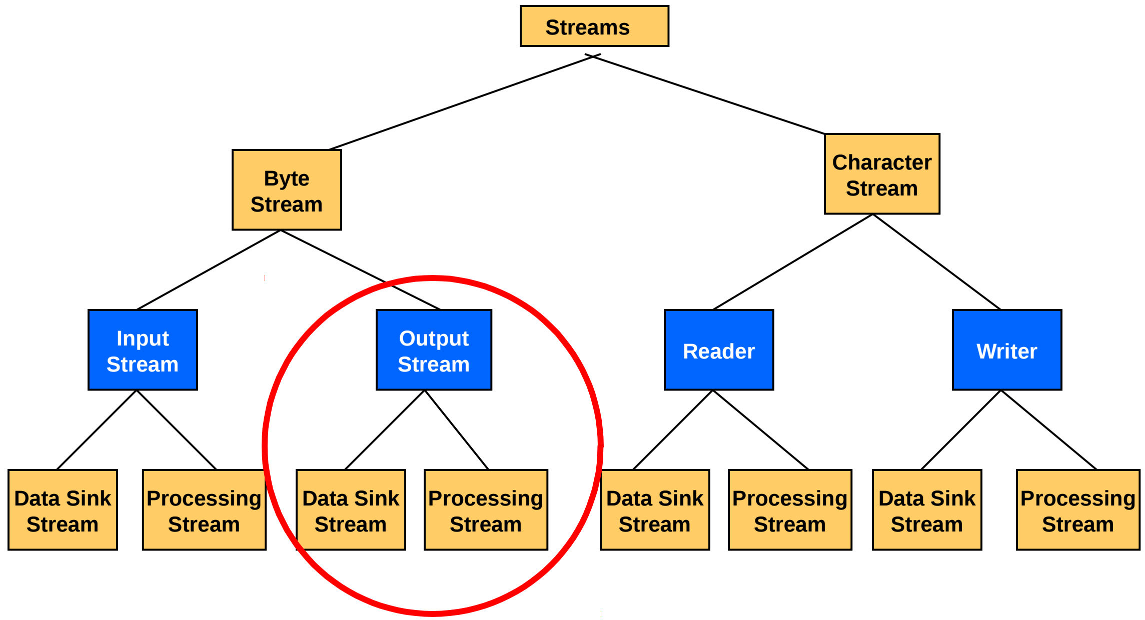
🟦 Class

🟨 Concept

Input Stream Class

- It declares three basic methods needed to read bytes of data from a stream.
  - **abstract int read()**
    - Reads the next byte of data from the input stream.
  - **int read(byte[ ] b)**
    - Reads some number of bytes from the input stream and stores them into buffer array b.
  - **int read(byte[ ] b , int off , int len)**
    - Reads up to len bytes of data from the input stream into an array of bytes.
- **Other methods**
  - **int available()**
    - Returns the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream.
  - **void close()**
    - Closes this input stream and releases any system resources associated with the stream.
  - **long skip(long n)**
    - Skips over and discards n bytes of data from this input stream.
  - **void mark(int readLimit)**
    - Marks the current position in this input stream. The readlimit arguments tells this input stream to allow that many bytes to be read before the mark position gets invalidated.
  - **void reset()**
    - Repositions this stream to the position at the time the mark method was last called on this input stream.

Output Stream Classes



Streams

Byte Stream

Character Stream

Input Stream

Output Stream

Reader

Writer

Data Sink Stream

Processing Stream

Data Sink Stream

Processing Stream

Data Sink Stream

Processing Stream

Data Sink Stream

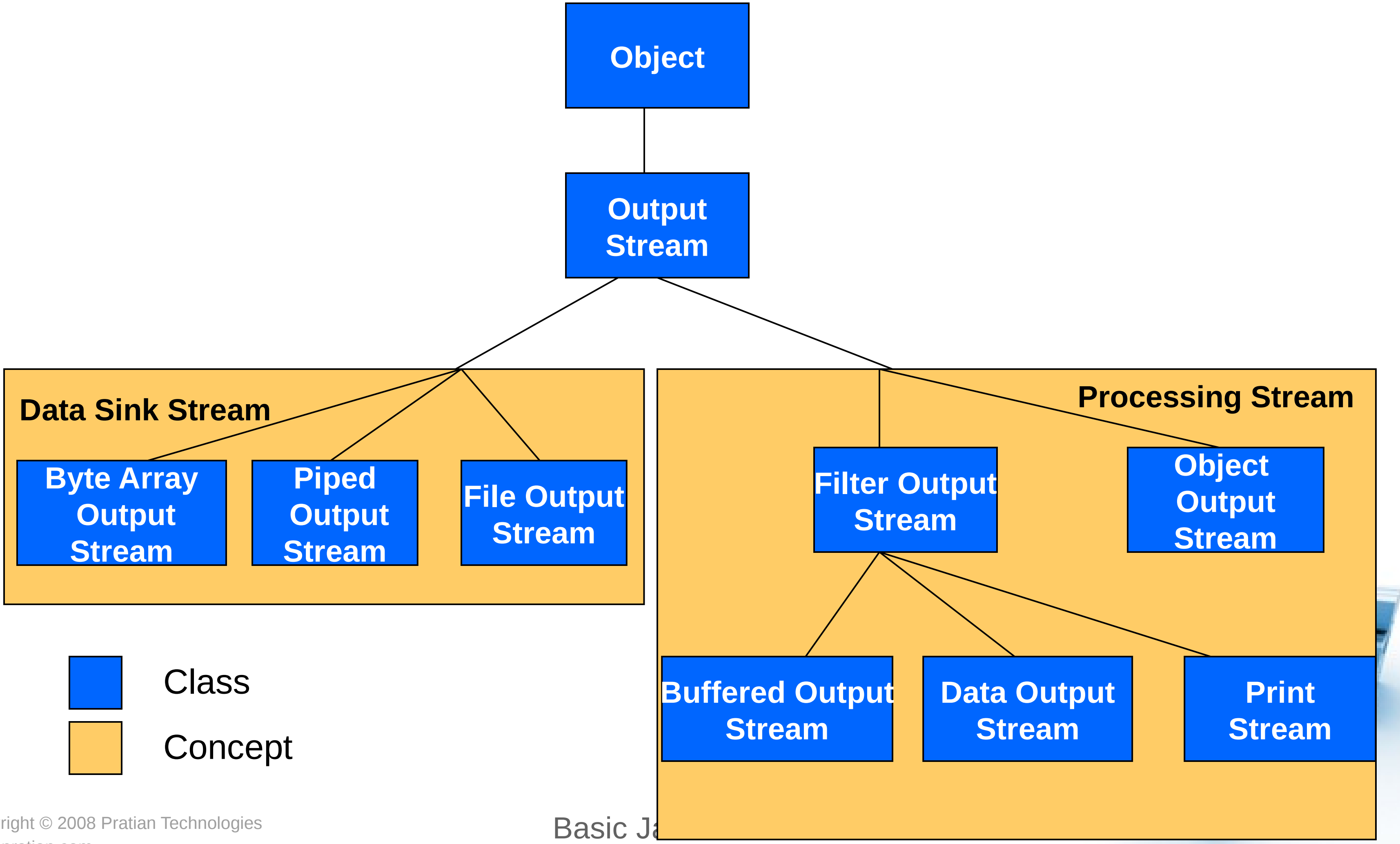
Processing Stream

🟦 Class

🟨 Concept

Output Stream Classes

- Java's Output Stream hierarchy.
- ***OutputStream*** is the abstract base class for all output stream classes
- OutputStream Class extends from Object



OutputStream Class

- It declares three basic methods needed to write bytes of data onto a stream.
  - **abstract void write(int data)**
    - Writes the specified byte to this output stream.
  - **void write(byte[ ] data)**
    - Writes b.length bytes from the specified byte array to this output stream.
  - **void write(byte[ ] data , int off , int len)**
    - Writes len bytes from the specified byte array starting at offset off to this output stream.
- **Other methods**
  - **void flush()**
    - Flushes this output stream and forces any buffered output bytes to be written out.
  - **long close()**
    - Closes this output stream and releases any system resources associated with this stream.

throws IOException Class

- Everything here can throw an **IOException**, so I'll stop mentioning it.
- If you use any of these methods, you must catch **IOException** (or **Exception**).

IOException

IOException

IOException

IOException

IOException

System.out

- **System.out** is actually a kind of **OutputStream**:
  - it's a **PrintStream** object.

```
OutputStream stdout = System.out;
stdout.write(65);  // ASCII ‘A'
stdout.flush();
```

IOExceptions!

```
public static void main(String[] args)
{
  OutputStream stdout = System.out;
  try {
    stdout.write(65);  // ‘A'
    stdout.write(10);   // '\n'
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

Reading from the console -System.in

- **System.in** is a type of **InputStream**

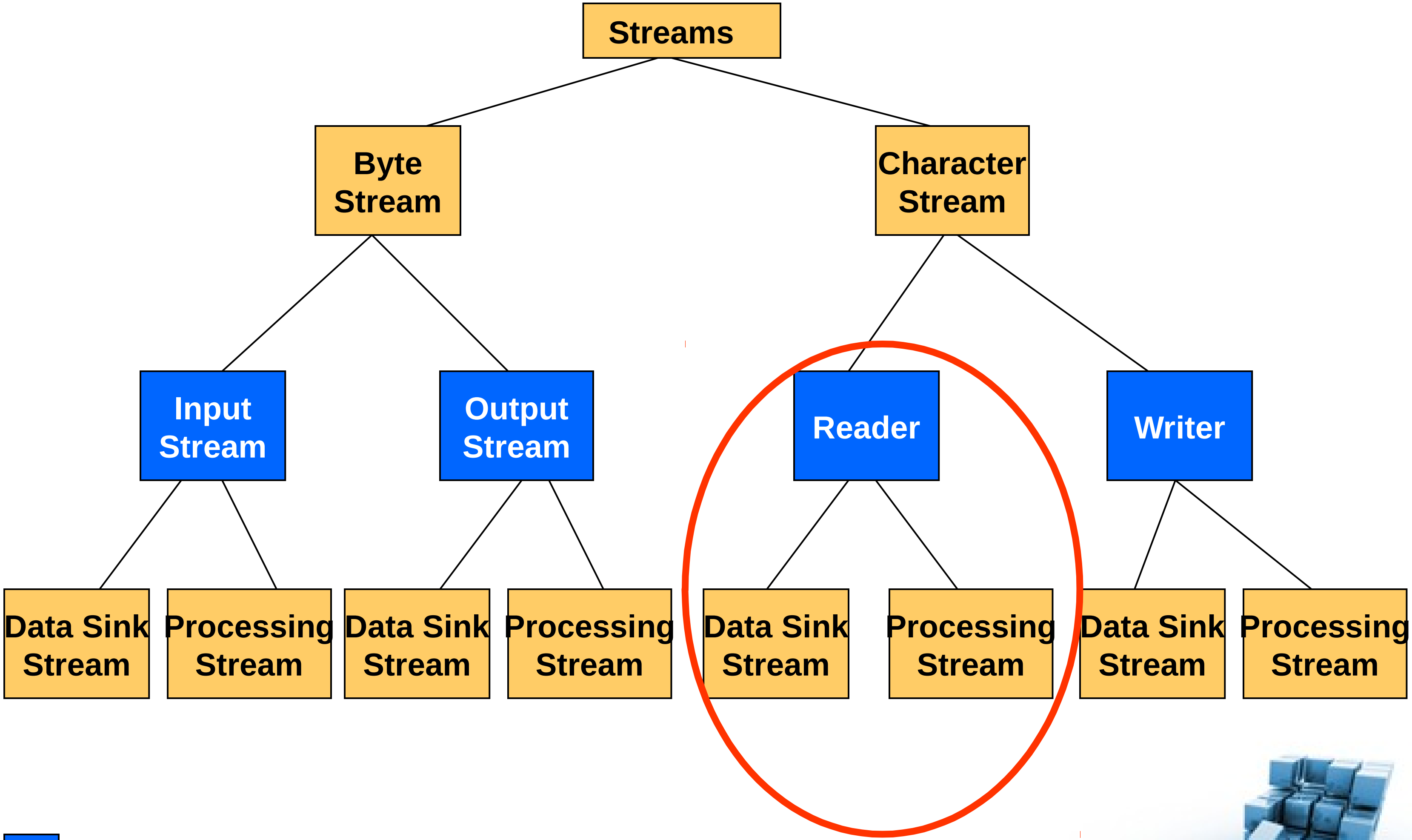
CharacterReaderDemo.java

```
public static void main(String[] args)
    {
            int data = 0;
            try
            {       System.out.print("Enter a character : ");
                    data = System.in.read();
                    char ch=(char)data;
                    System.out.print(ch);
            }

            catch(Exception e)
            {
                   System.out.println("Could not read from System.in!...");
            }
    }
```

Exercises

- Write a program to generate the ASCII table, and display the same on the console.
- Write a program to read many characters from the console and display the same using the System.out.println().

Revisiting Types of Streams Classes



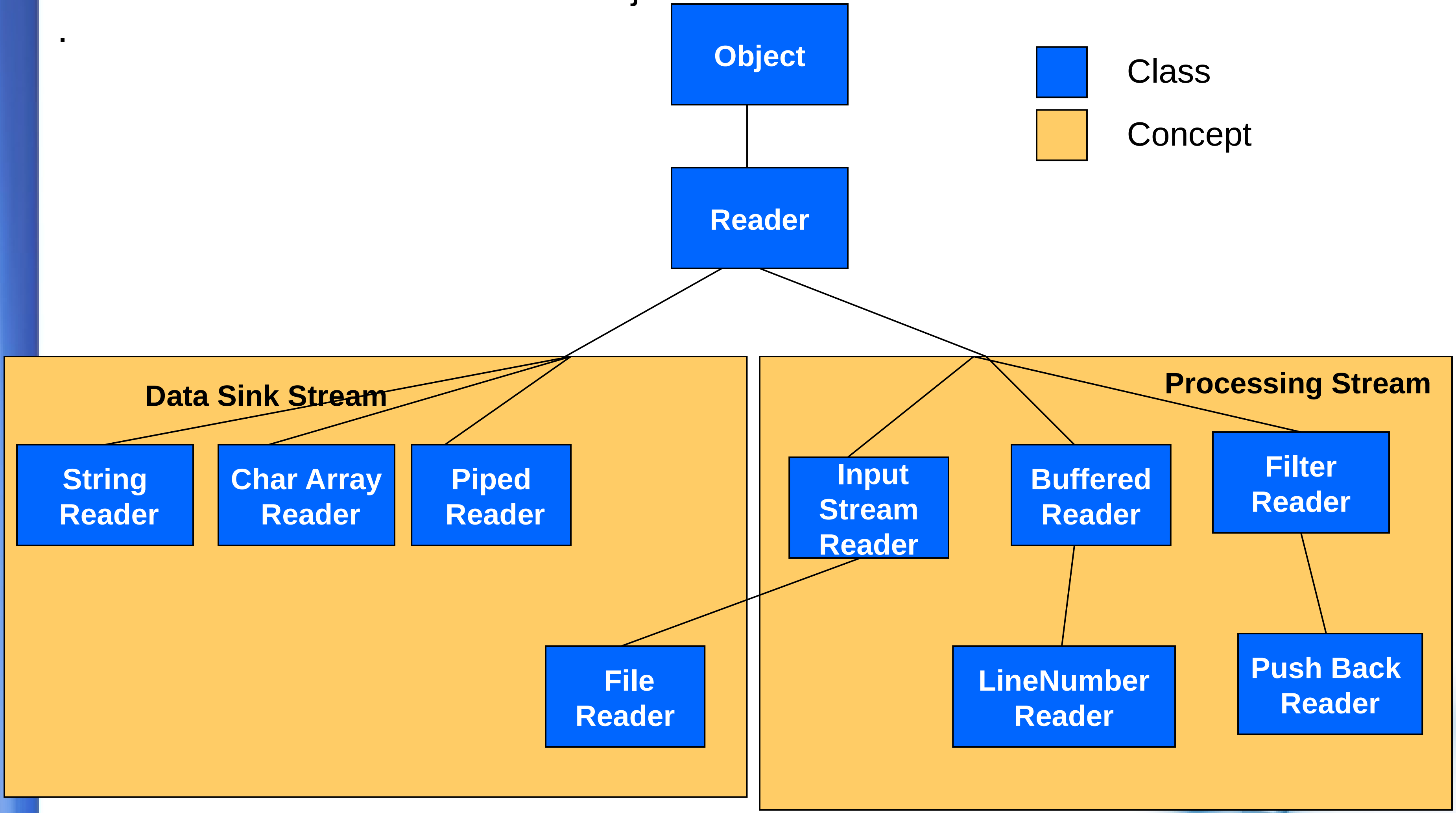
🟦 Class

🟨 Concept

Reader Classes

- Java's Reader class hierarchy.
- ***Reader*** is the abstract base class for all character stream reader classes
- Reader Class extends from Object

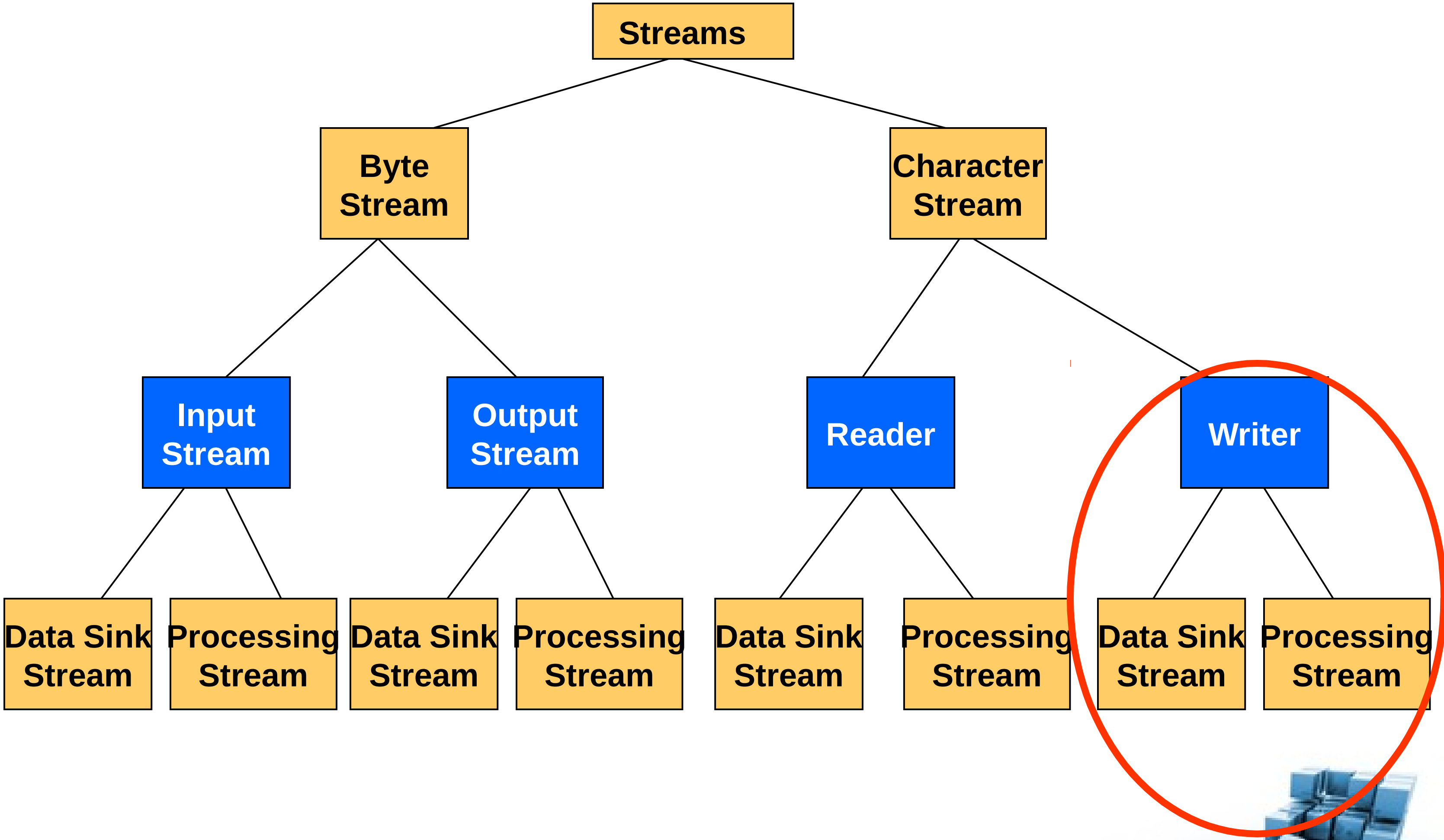
.



Reader Class

- Reader is the abstract class for reading from character streams which forms the superclass for all character stream readers.
- It declares few basic methods needed to read character of data from a stream.
  - **int read()**
  - **int read(char[ ] c)**
  - **abstract int read(char[ ] c , int off , int len)**
- It also has methods for closing streams, checking if the stream is ready to use, reset etc..
  - **void close()**
  - **long skip(long n)**
  - **boolean ready()**
  - **void reset()**

Revisiting Types of Streams Classes



Streams

Byte Stream

Character Stream

Input Stream

Output Stream

Reader

Writer

Data Sink Stream

Processing Stream

Data Sink Stream

Processing Stream

Data Sink Stream

Processing Stream

Data Sink Stream

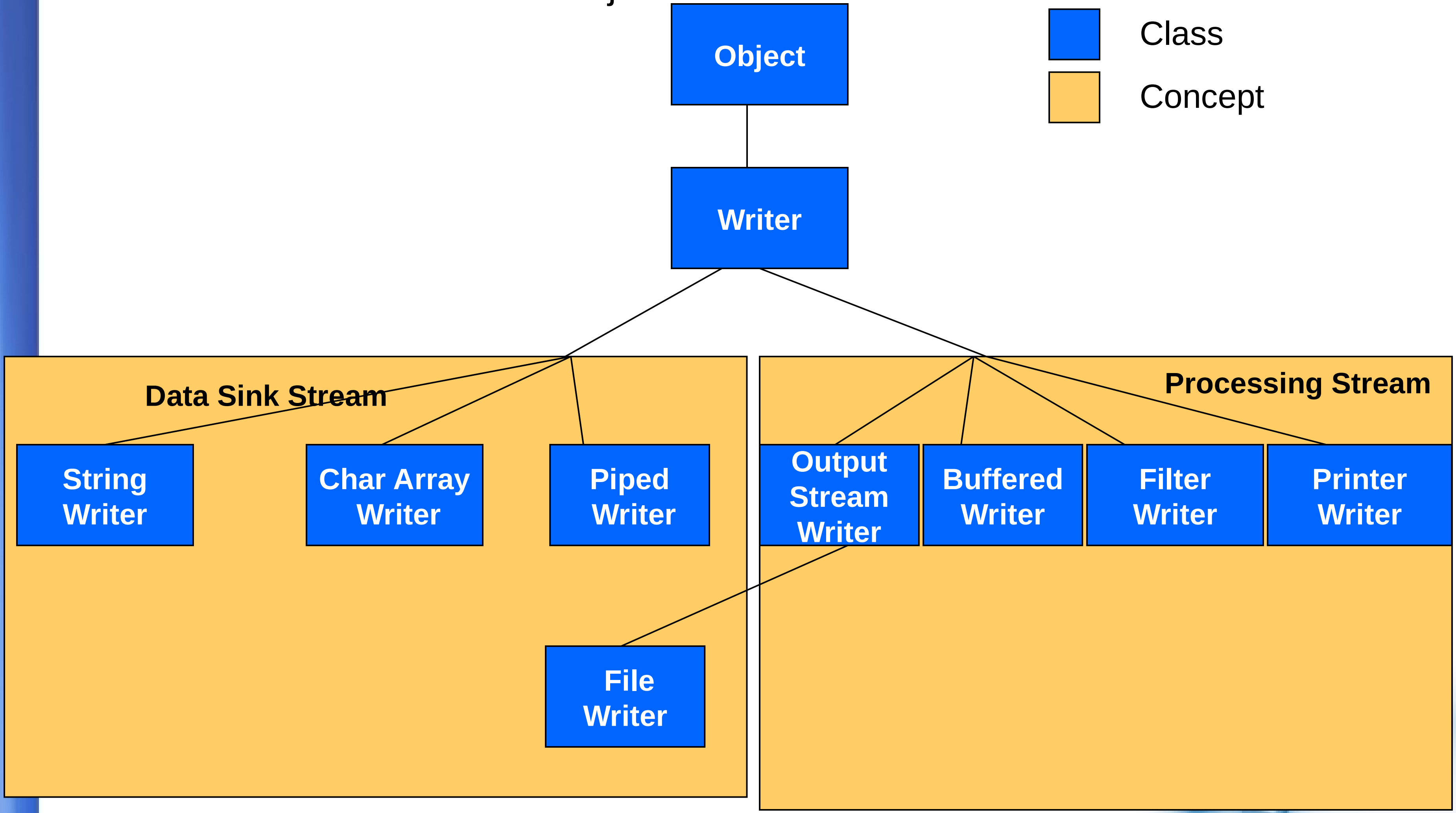
Processing Stream

🟦 Class

🟨 Concept

Writer Classes

- Java's Writer class hierarchy.
- ***Writer*** is the abstract base class for all character stream writer classes
- Writer Class extends from Object.



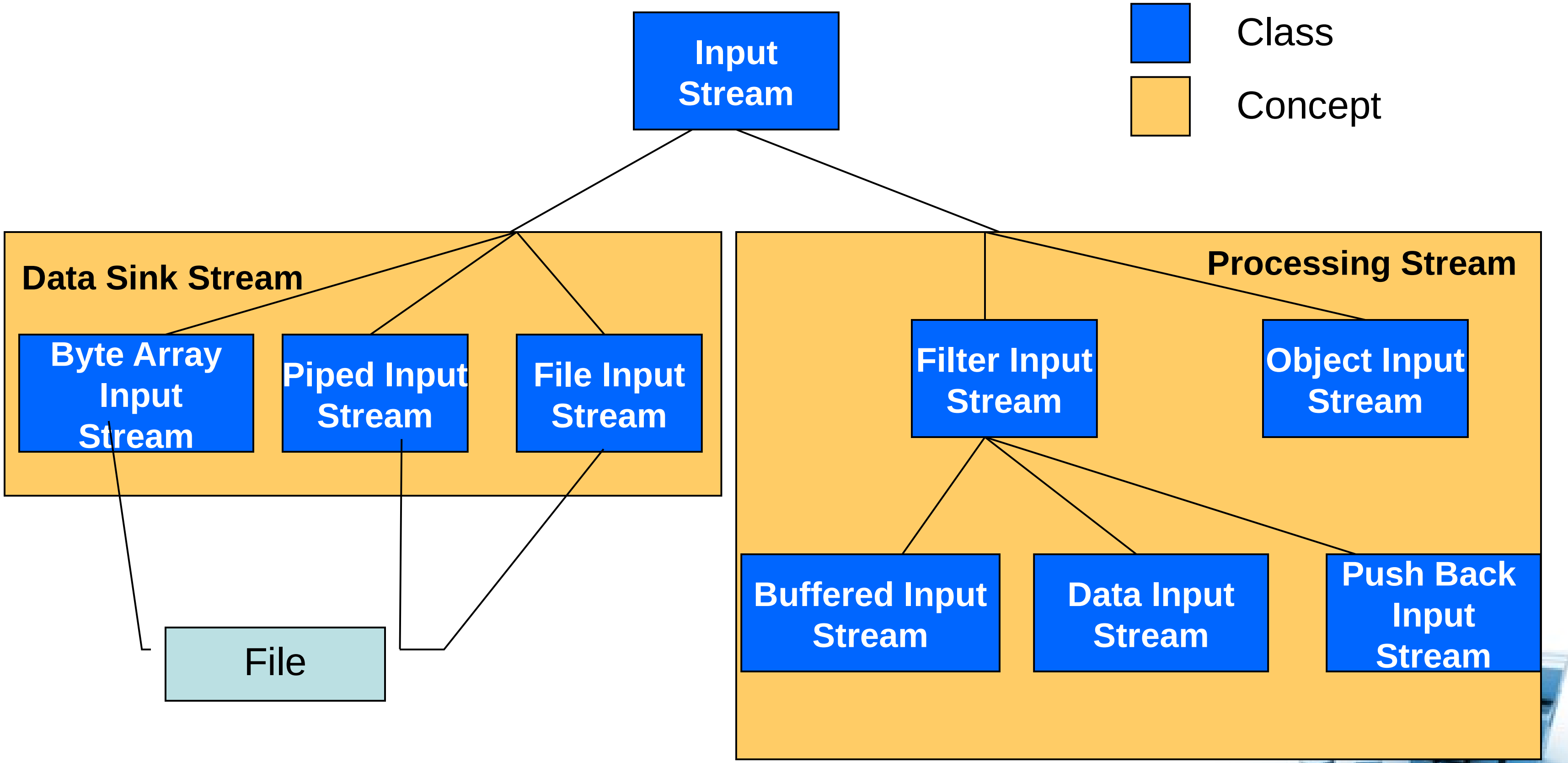
Writer Class

- Writer is the abstract class for writing to character streams which forms the superclass for all character stream writers.
- It declares few basic methods needed to write character of data to a stream.
  - **void write(int data)**
  - **void write(char[ ] data)**
  - **void write(char[ ] data , int off , int len)**
  - **void write(String str)**
  - **void write(String str , int off , int len)**

- It also has methods for closing and flushing streams.
  - **void flush()**
  - **long close()**

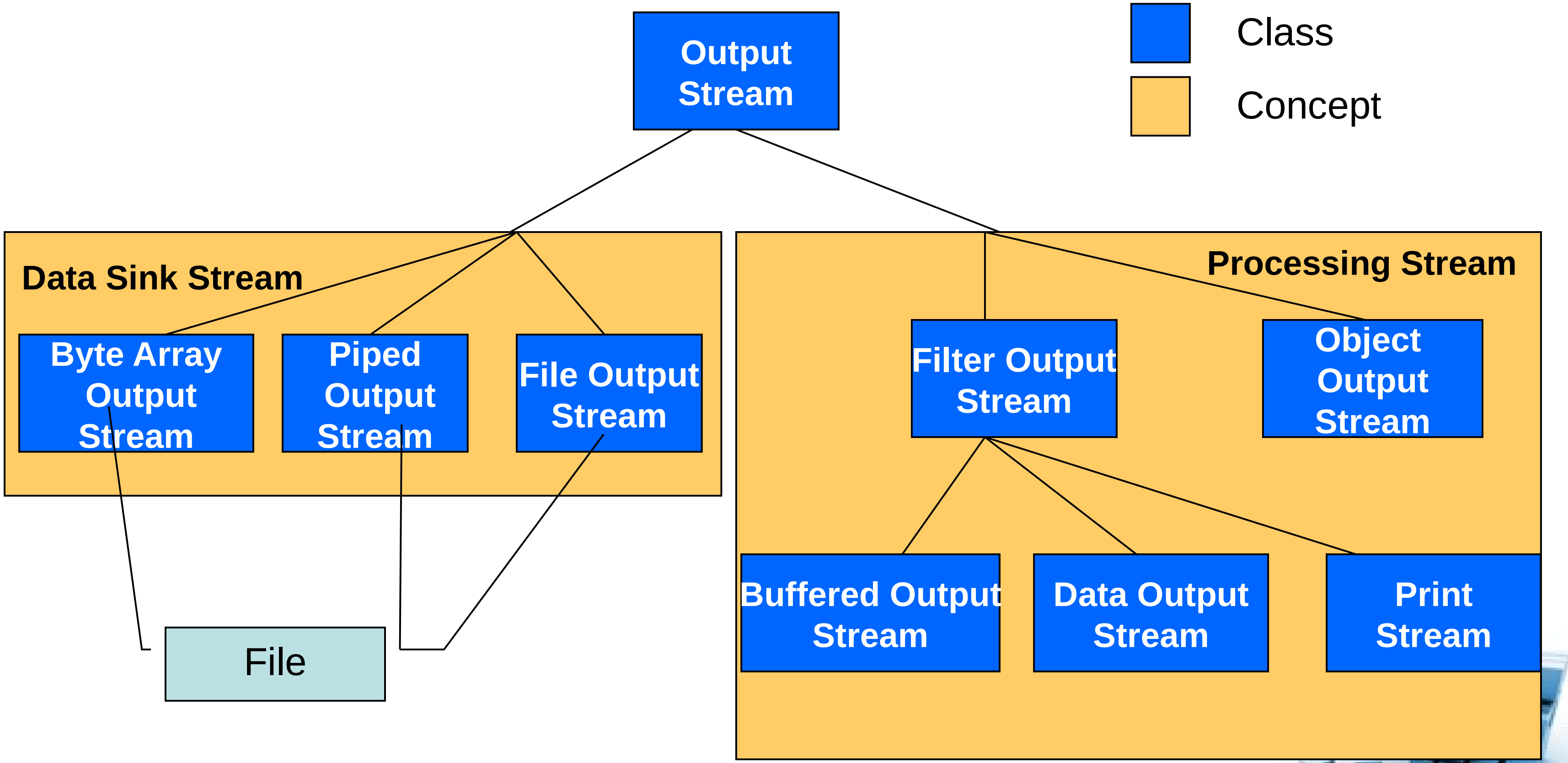
Data Sink Streams : A Closer look

- Data sink streams read from or write to specialized data sinks such as memory, files, or pipes.



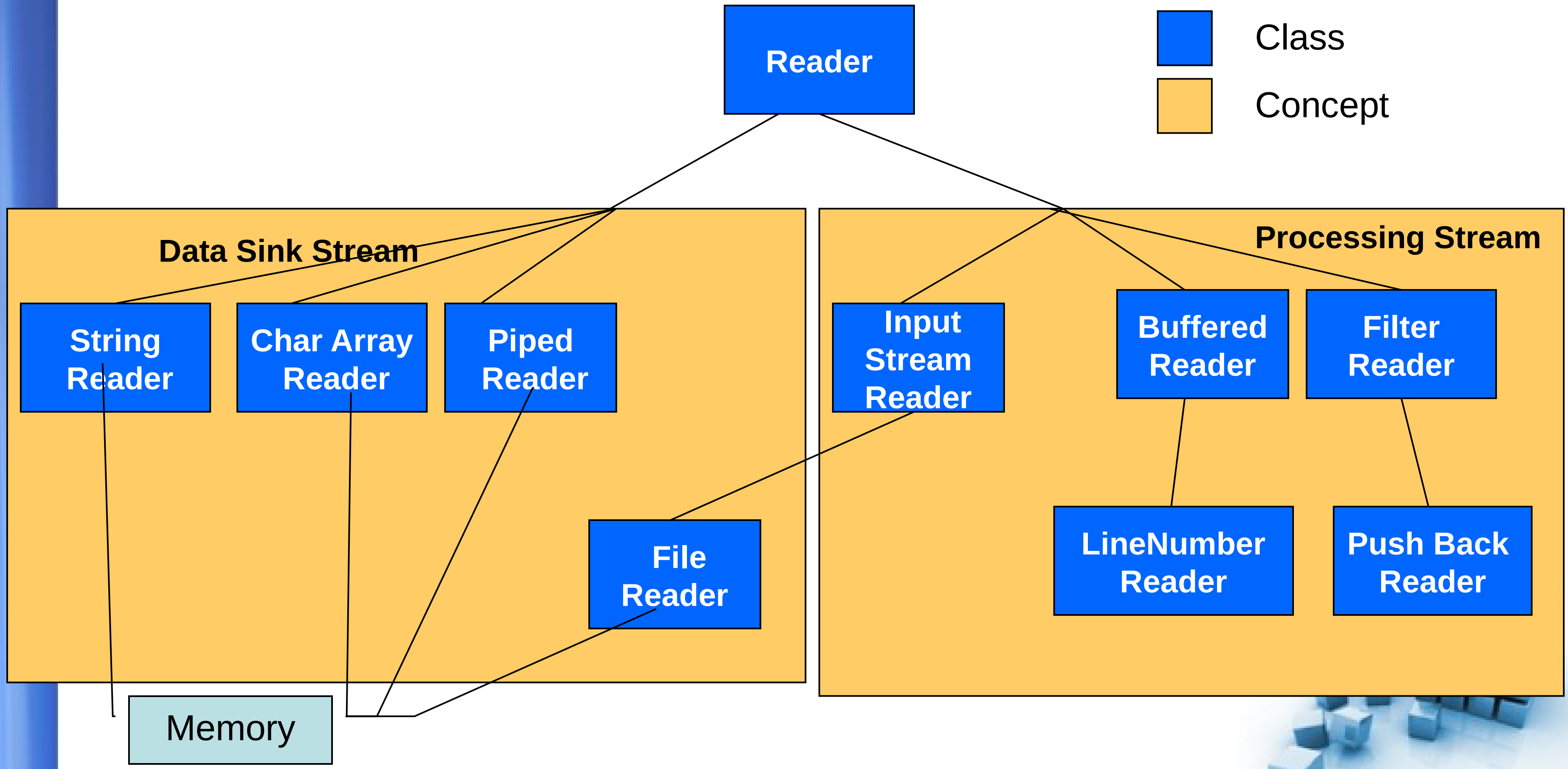
Data Sink Streams : A Closer look

- Data sink streams read from or write to specialized data sinks such as memory, files, or pipes.



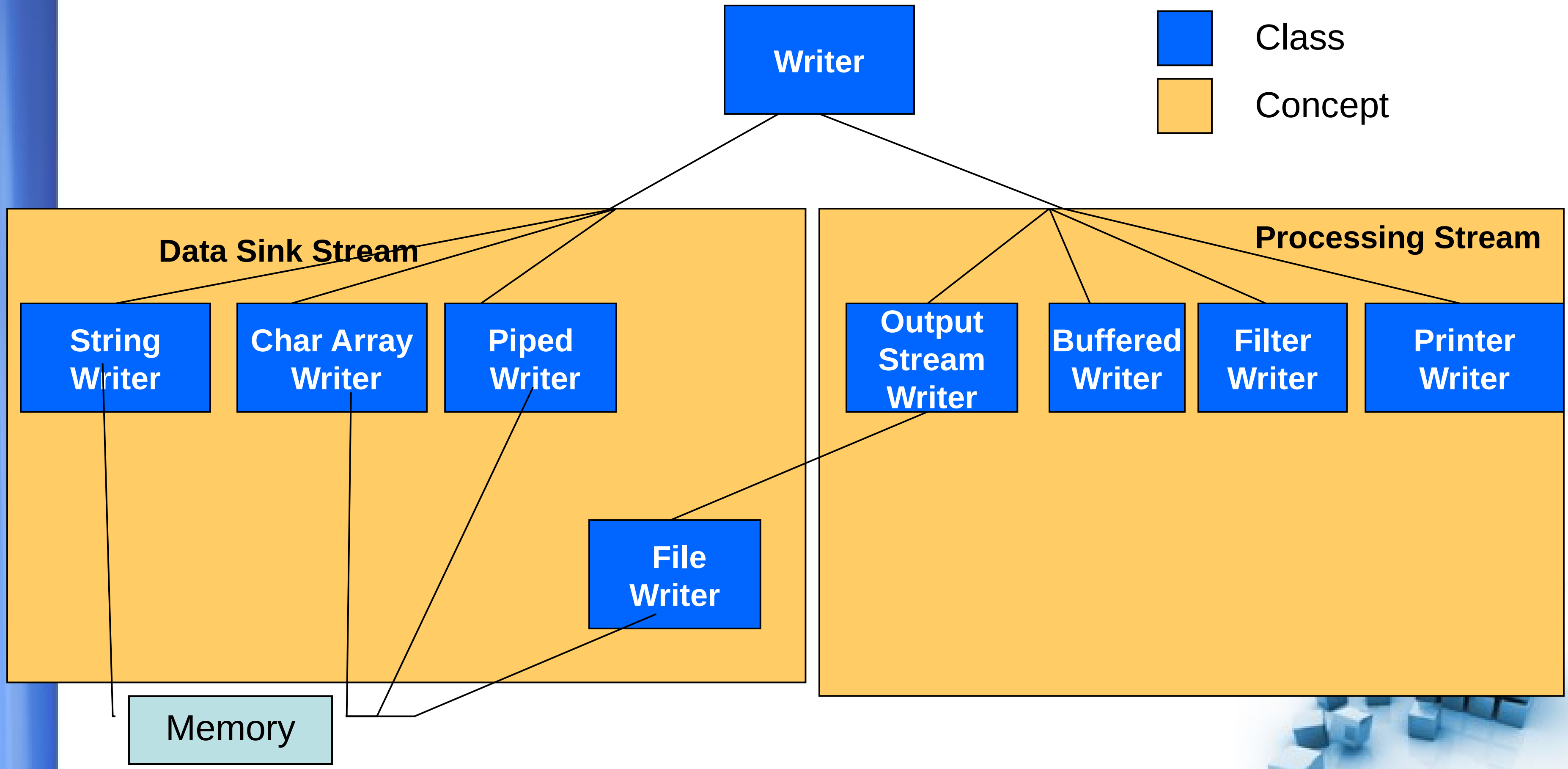
Data Sink Streams : A Closer look

- Data sink streams read from or write to specialized data sinks such as memory, files, or pipes.



Data Sink Streams : A Closer look

- Data sink streams read from or write to specialized data sinks such as memory, files, or pipes.

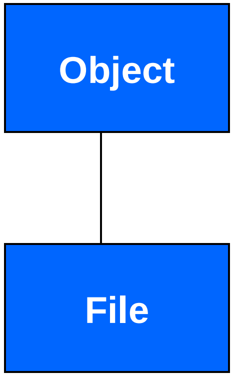


To Summarize: Data Sink Streams

Data sink streams read from or write to specialized data sinks such as memory, files, or pipes.

Sink Type	**Character Streams**	**Byte Streams**
Memory	CharArrayReader	ByteArrayInputStream,
	CharArrayWriter	ByteArrayOutputStream
	StringReader	StringBufferInputStream
	StringWriter	
Pipe	PipedReader	PipedInputStream
	PipedWriter	PipedOutputStream
File	FileReader	FileInputStream
	FileWriter	FileOutputStream

File Class



- **File** class is used to write platform independent code to examine and manipulate files.
- File class instances represent file names (conceptualize files), not the physical files. The file corresponding to the file name might not even exist.

File Class

- If a file does not exist, it can be created.
- If the file does exist, a program can
  - examine its attributes
  - perform various operations on the file, such as
    - renaming it,
    - deleting it,
    - changing its permission.
- Has methods to
  - return a File object from a pathname string
  - test whether a file exists and its permission
  - test if it is a file or directory
  - delete the file
  - get File size, last modification date, etc.

File Class

- Constructors
  - **File(String pathname)**

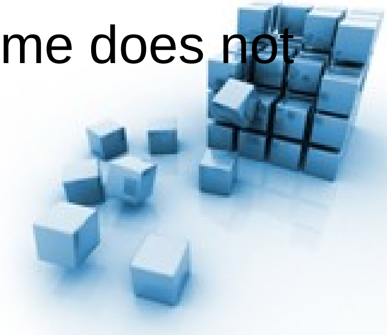
    Creates a new File instance by converting the given pathname string into an abstract pathname.
- Methods
  - **boolean exists()**

    Checks if the file or directory exists or not.
  - **boolean canRead()**

    Checks whether the application can read the file.
  - **boolean canWrite()**

    Checks whether the application can modify the file.
  - **boolean createNewFile()**

    Creates a new, empty file if a file with this name does not exist.



File Class

- Methods
  - **boolean delete()**

    Deletes the file or directory.
  - **boolean isDirectory()**

    Checks whether the file denoted is a directory.
  - **boolean isFile()**

    Checks whether the file denoted is a normal file.
  - **String[ ] list()**

    Returns an array of strings naming the files and directories in the directory.

File Example

```
import java.io.*;

class FileDemo
{
    public static void main(String[] args) throws Exception
    {
        File file = new File("FileDemo.java");

       if (file.exists())
         System.out.println("The file exists...");
       else
        {
      System.out.println("The file does not exist, creating a new file...");
      file.createNewFile();
        }
        System.out.println("Absolute path of the file “ +
         file.getAbsolutePath());
    }
}
```

File Class

- File class has some useful methods for working with directories.
  - **boolean mkdir()**

    Creates the directory named by this abstract pathname.
  - **boolean mkdirs()**

    Creates the directory named by this abstract pathname,

    including any necessary but nonexistent parent.
  - **String[ ] list()**

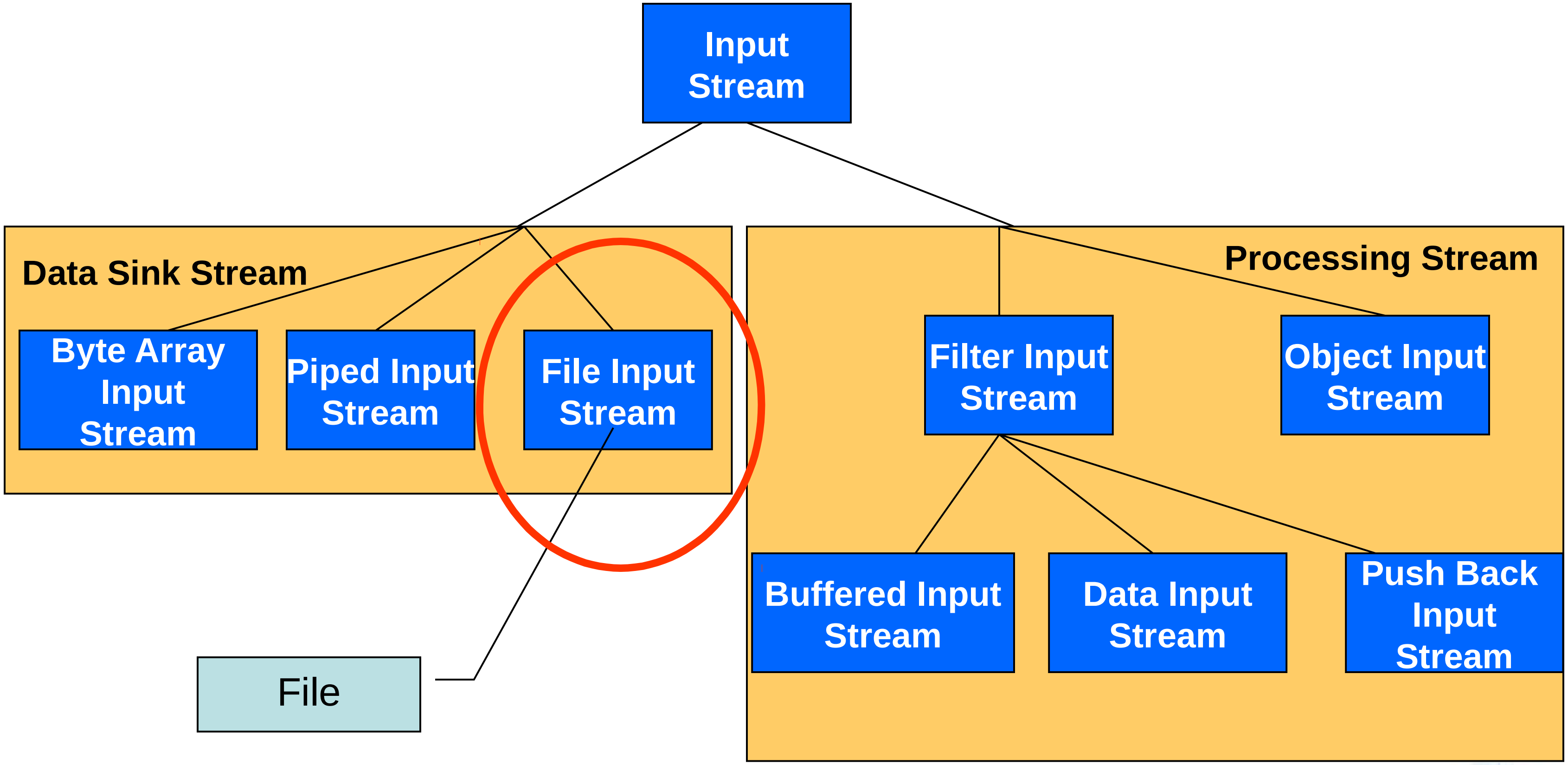
    Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname.
  - **File[ ] listFiles()**

    Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.

Exercise

- Write a program which creates a directory and add a text file MyFile.txt to it and make the file read only.
- Write a program which lists all the directories and files under the C:\ drive.

FileInputStream



🟦 Class

🟨 Concept

FileInputStream

- public class **FileInputStream** extends InputStream
  - A FileInputStream class is used by a program to read information from a file in bytes.
  - FileInputStream is meant for reading streams of raw bytes such as image data.
- Constructors
  - **FileInputStream(File file)**

    Creates a FileInputStream by opening a connection to

    actual file named in the File object.
  - **FileInputStream(String name)**

    Creates a FileInputStream by opening a connection to

    actual file named by the pathname.

FileInputStream

- Methods
  - **int read()**

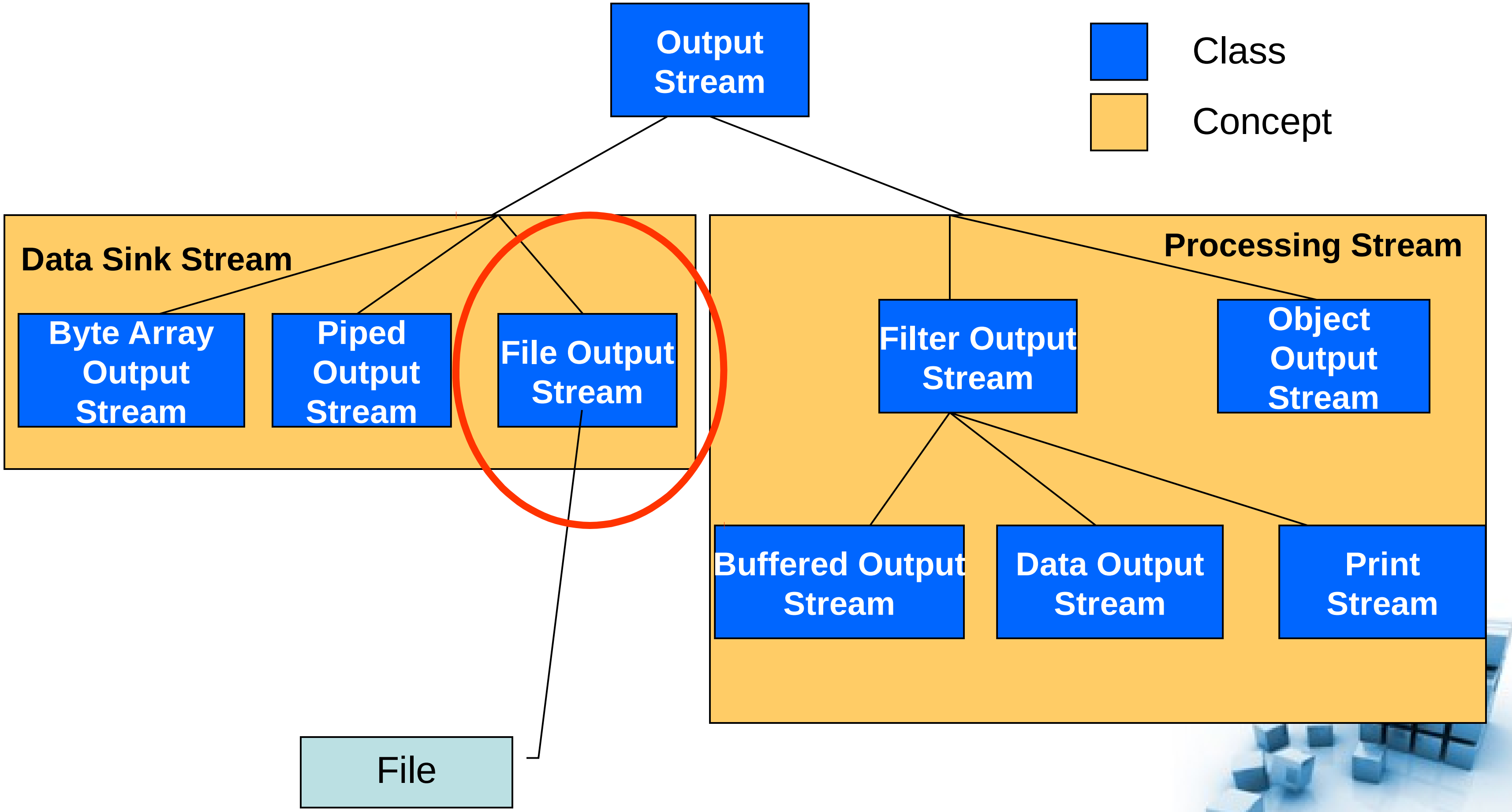
    Reads a byte of data from this input stream.
  - **int read(byte[ ] b)**

    Reads up to b.length bytes of data from this input stream into an array of bytes.
  - **void close()**

    Closes this file input stream and releases any system resources associated with the stream.
  - **protected void finalize()**

    Ensures that the close method of this file input stream is called when there are no more references to it.

FileOutputStream



FileOutputStream

- public class **FileOutputStream** extends OutputStream
  - A FileOutputStream is an output stream for writing data to a File.
  - A FileOutPutStream is meant for writing streams of raw bytes such as image data.
- Constructors
  - FileOutputStream(File file)
  - FileOutputStream(File f, boolean append)
  - FileOutputStream(String name)
  - FileOutputStream(String name, boolean append)

FileOutputStream

- Methods
  - **void write(int i)**

    Writes specified bytes to this FileOutputStream
  - **void write(byte[ ] b)**

    Writes b.length bytes from the specified byte array to this FileOutputStream.
  - **void write(byte[ ] b, int off, int len)**

    Writes len bytes from the specified byte array starting at offset off to this FileOutputStream.
  - **void close()**

    Closes this FileOutputStream
  - **protected void finalize()**

    Ensures that the close method of this FileOutputStream is called when there are no more references to it.

Example

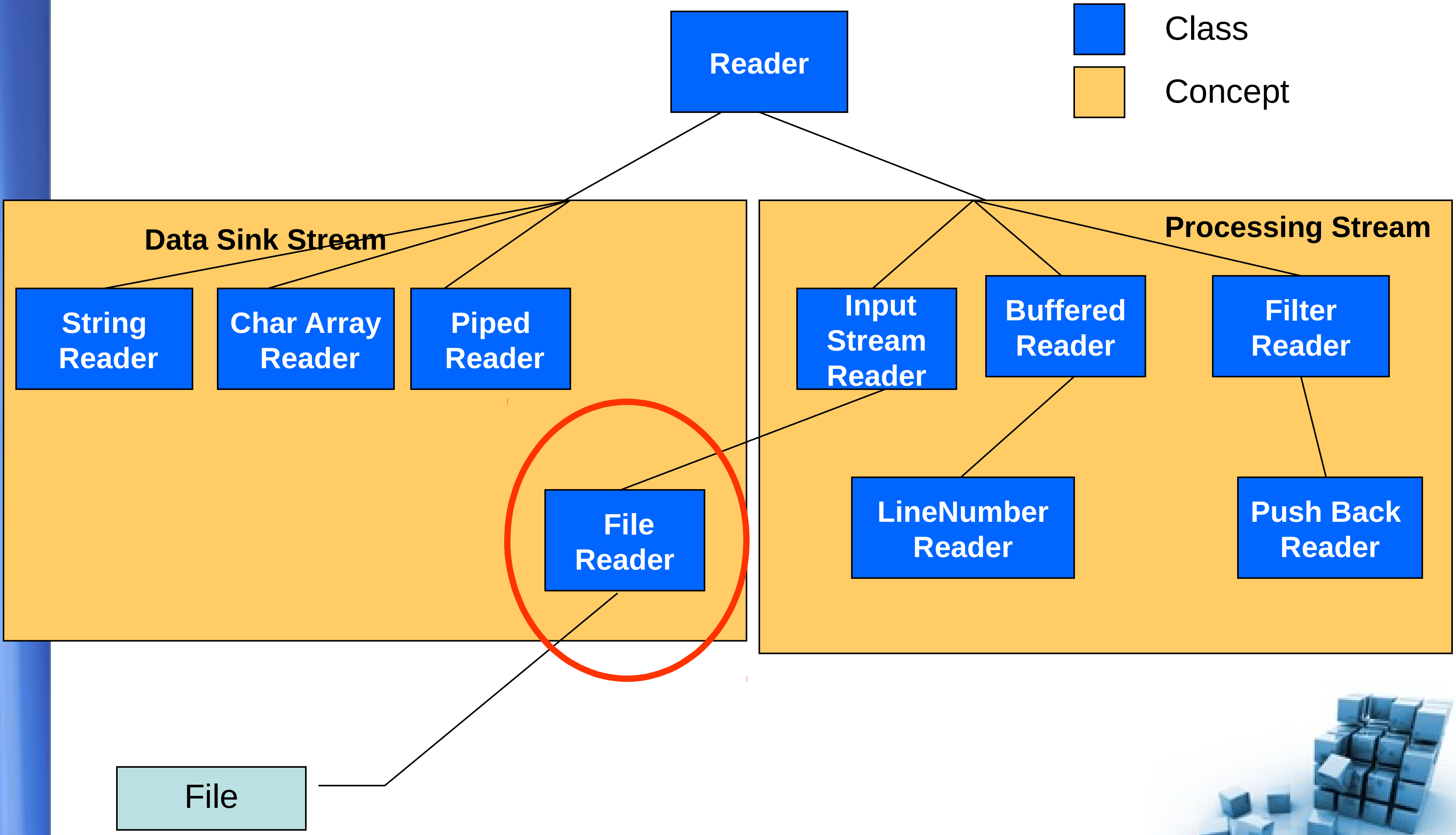
```
public static void main(String[] args) throws Exception
{
        File sourceFile = new File(“MyText.txt");
        FileInputStream in = new FileInputStream(sourceFile);
        File targetFile = new File(“NewText.txt");
        FileOutputStream out =new FileOutputStream(targetFile);
        for(int c = in.read() ; c != -1 ; c = in.read())
        {
                out.write(c);
        }
        in.close();
        out.close();
}
```

See Listing :FileInputStreamDemo.java

Exercises

- Write a program to create a new file and write the ASCII table to the file and save it.
- Write a program to make a copy of a image file.

FileReader


Reader
Class
Concept
Data Sink Stream
Processing Stream
String Reader
Char Array Reader
Piped Reader
Input Stream Reader
Buffered Reader
Filter Reader
File Reader
LineNumber Reader
Push Back Reader
File

FileReader

- public class **FileReader** extends InputStreamReader
  - Convenience class for reading character files.
  - FileReader is meant for reading streams of characters.

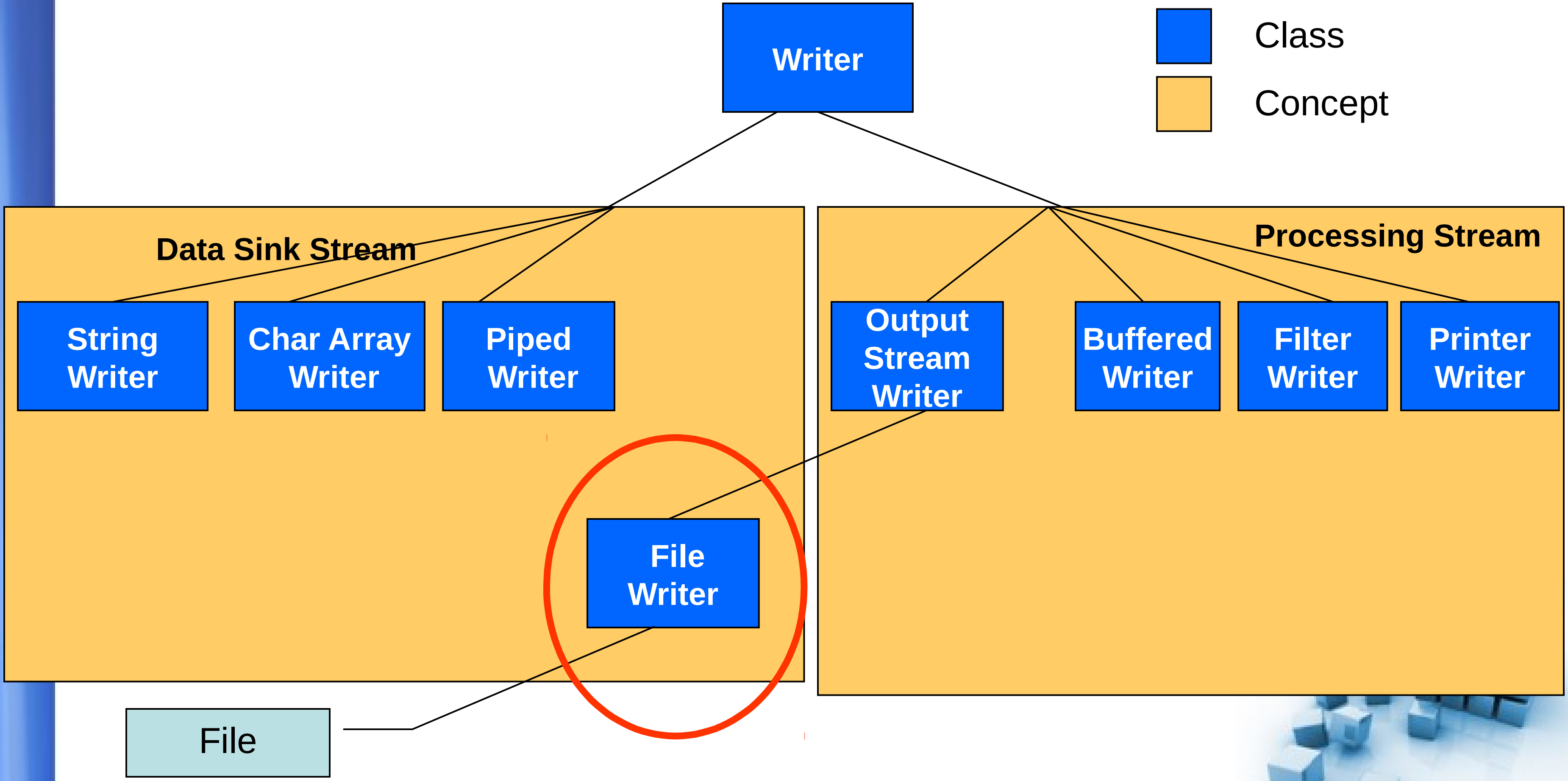
- Constructors
  - **FileReader(File file)**

    Creates a new FileReader for a given File to read from.
  - **FileReader(String fileName)**

    Creates a new FileReader for a given file name to read from.

- Methods
  - FileReader does not define any methods of its own, but inherits methods from InputStreamReader and Reader classes.

FileWriter


Writer
🟦 Class
🟨 Concept
Data Sink Stream
String Writer
Char Array Writer
Piped Writer
File Writer
File
Processing Stream
Output Stream Writer
Buffered Writer
Filter Writer
Printer Writer

FileWriter

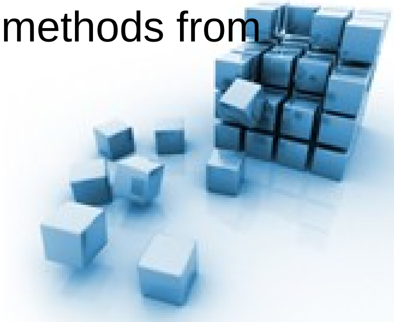
- public class **FileWriter** extends OutputStreamWriter
  - Convenience class for writing character files.
  - FileWriter is meant for writing streams of characters.

- Constructors
  - **FileWriter(File file)**

    Creates a new FileWriter for a given File to read from.
  - **FileWriter(String fileName)**

    Creates a new FileReader for a given file name to read from.
  - **FileWriter(File file , boolean append)**
  - **FileWriter(String fileName , boolean append)**

- Methods
  - FileWriter does not define any methods of its own, but inherits methods from OutputStreamReader and Writer classes.



Example

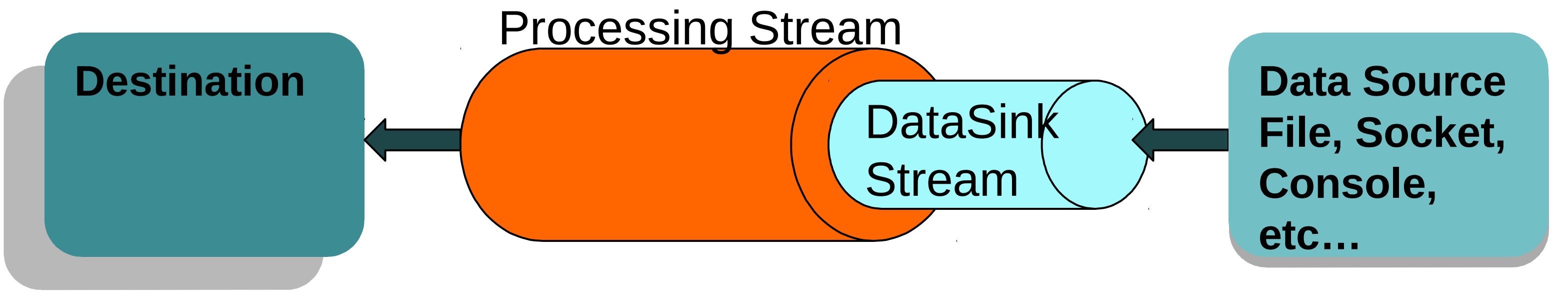
```
import java.io.*;
public class Copy {
    public static void main(String[] args) throws IOException
 {
     File inputFile = new File("farrago.txt");      //existing file
     File outputFile = new File("outagain.txt");
    FileReader in = new FileReader(inputFile);
    FileWriter out = new FileWriter(outputFile);
           int c;
           while ((c = in.read()) != -1)
                 out.write(c);
            in.close();
           out.close();
    }
}
```

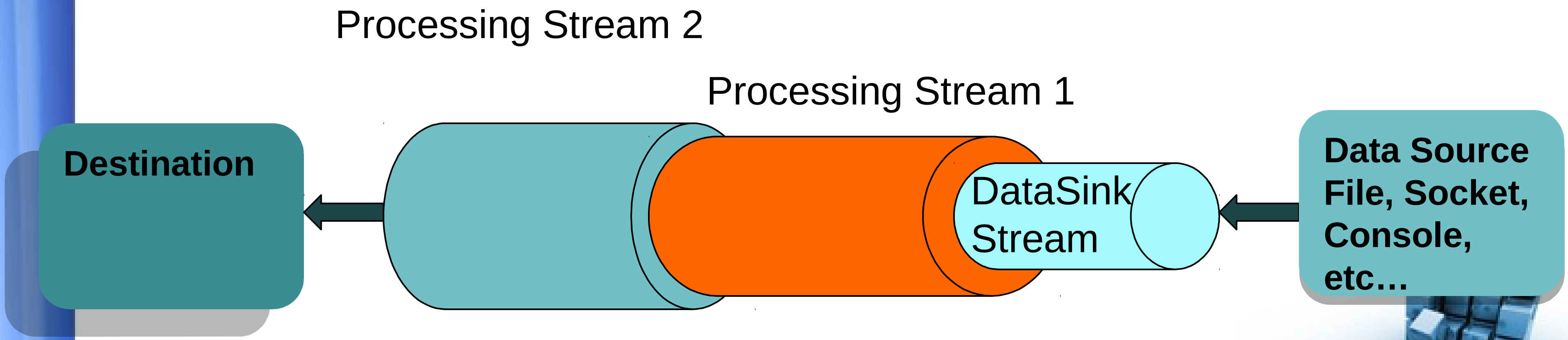
See Listing :FileReaderWriterDemo1.java

Stream Chaining

- Processing streams do not have the ability to directly read or write from any data source.
- They depend on an underlying stream to supply it with bytes/characters.
- They are always used by attaching them on top of another stream.
- This process of attaching one stream over another is called **Stream Chaining**.

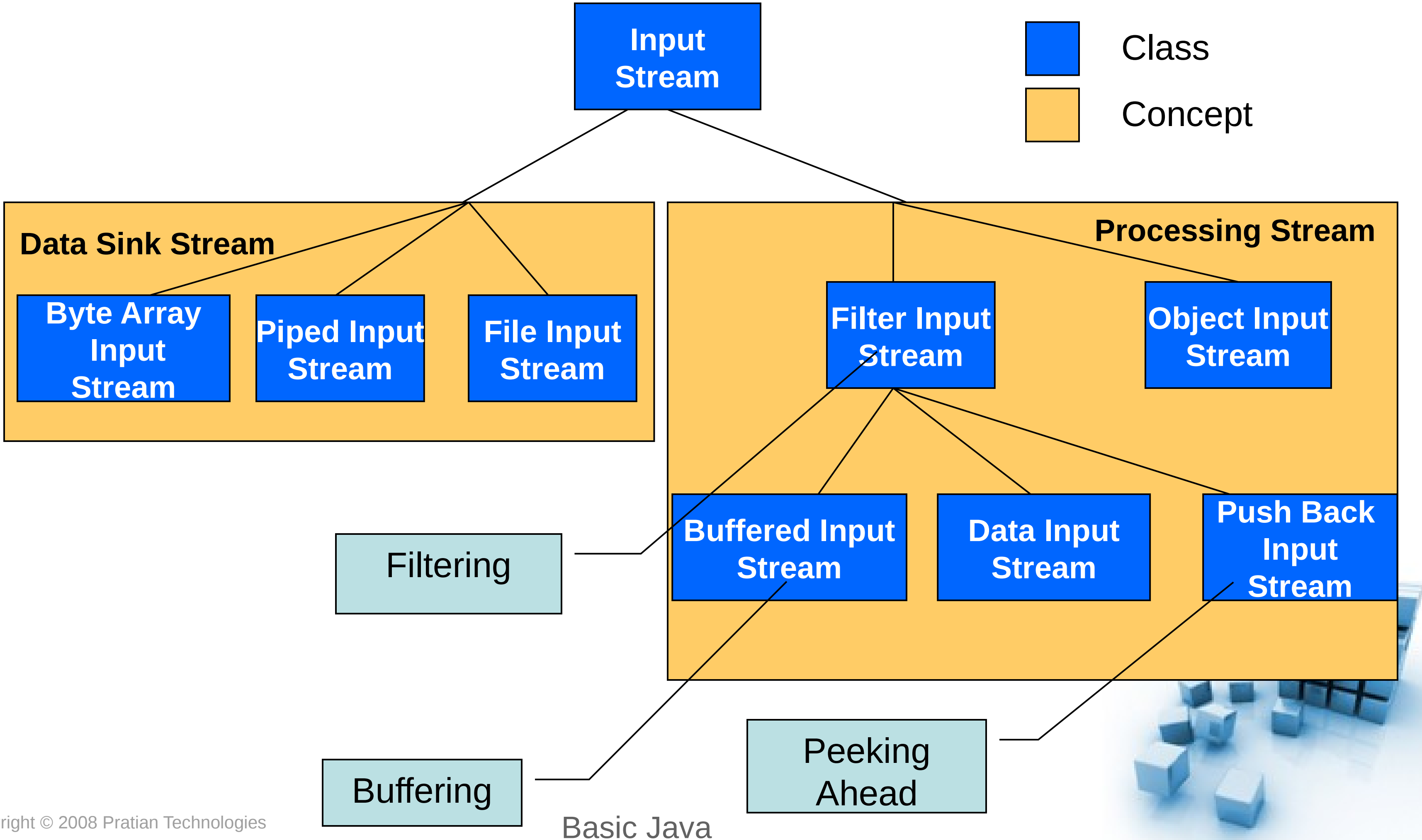
Stream Chaining





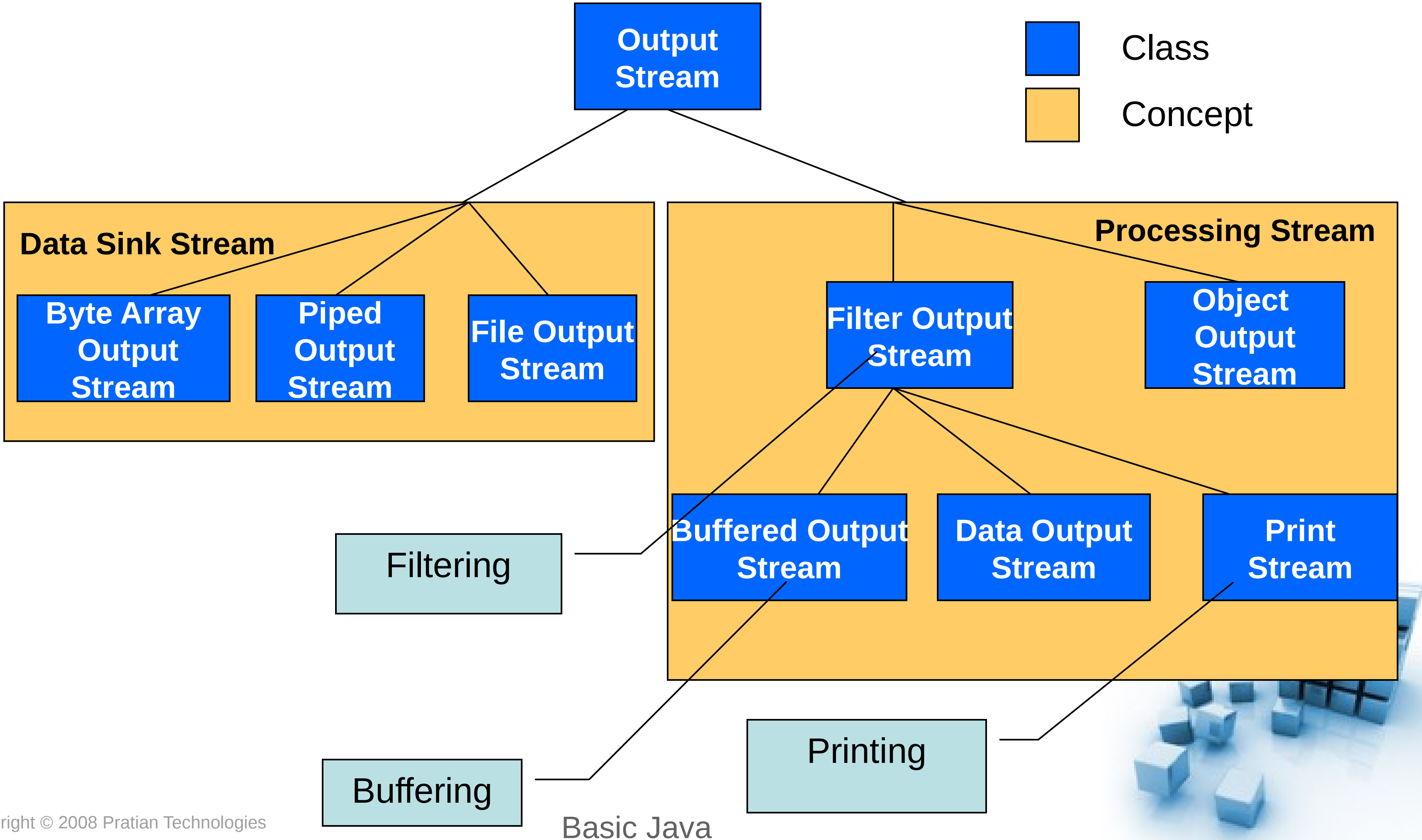
Processing Streams : A Closer look

- Processing streams perform some sort of operation, such as buffering or data conversion, as they read and write.



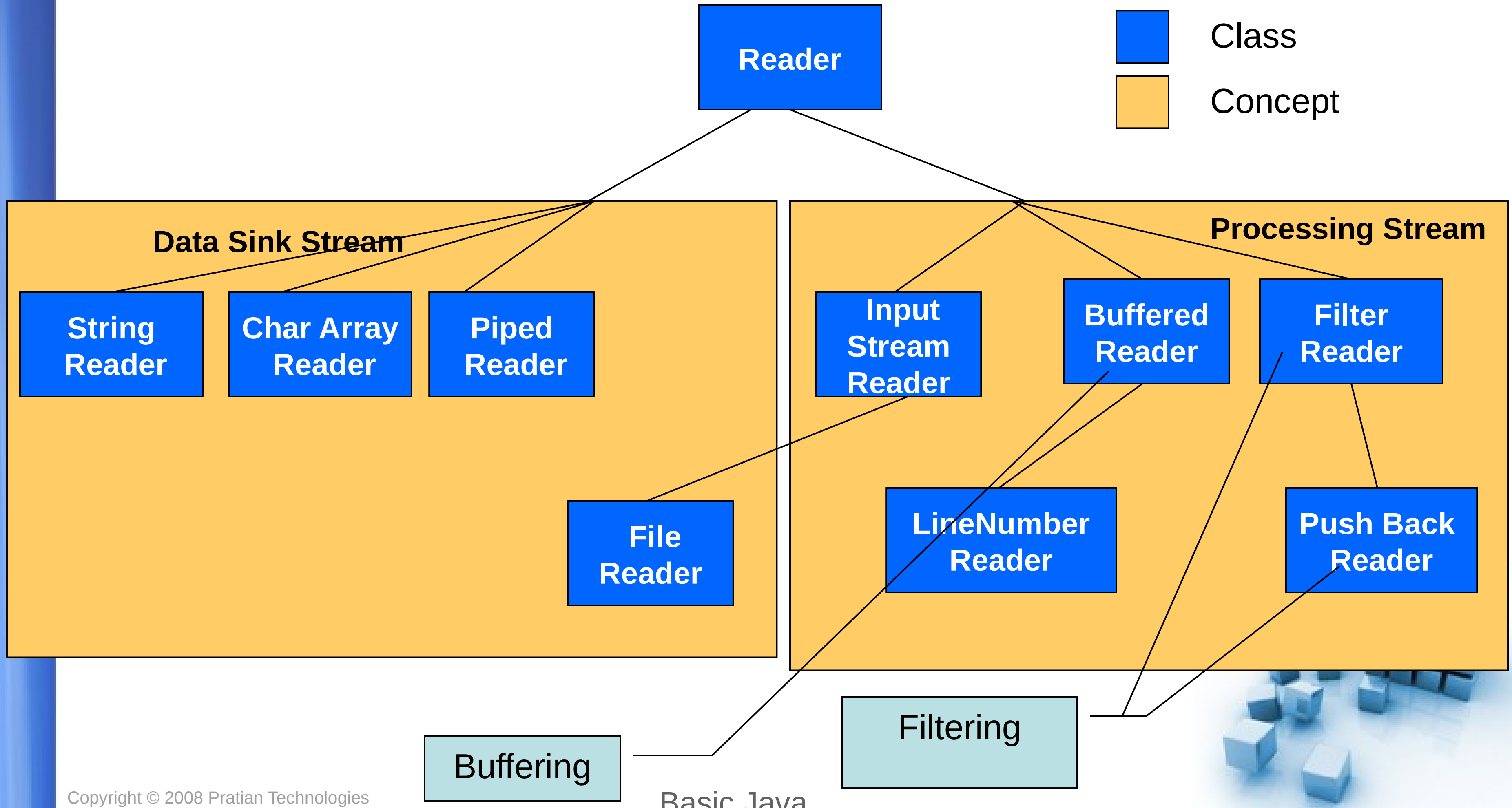
Processing Streams : A Closer look

- Processing streams perform some sort of operation, such as buffering or data conversion, as they read and write.



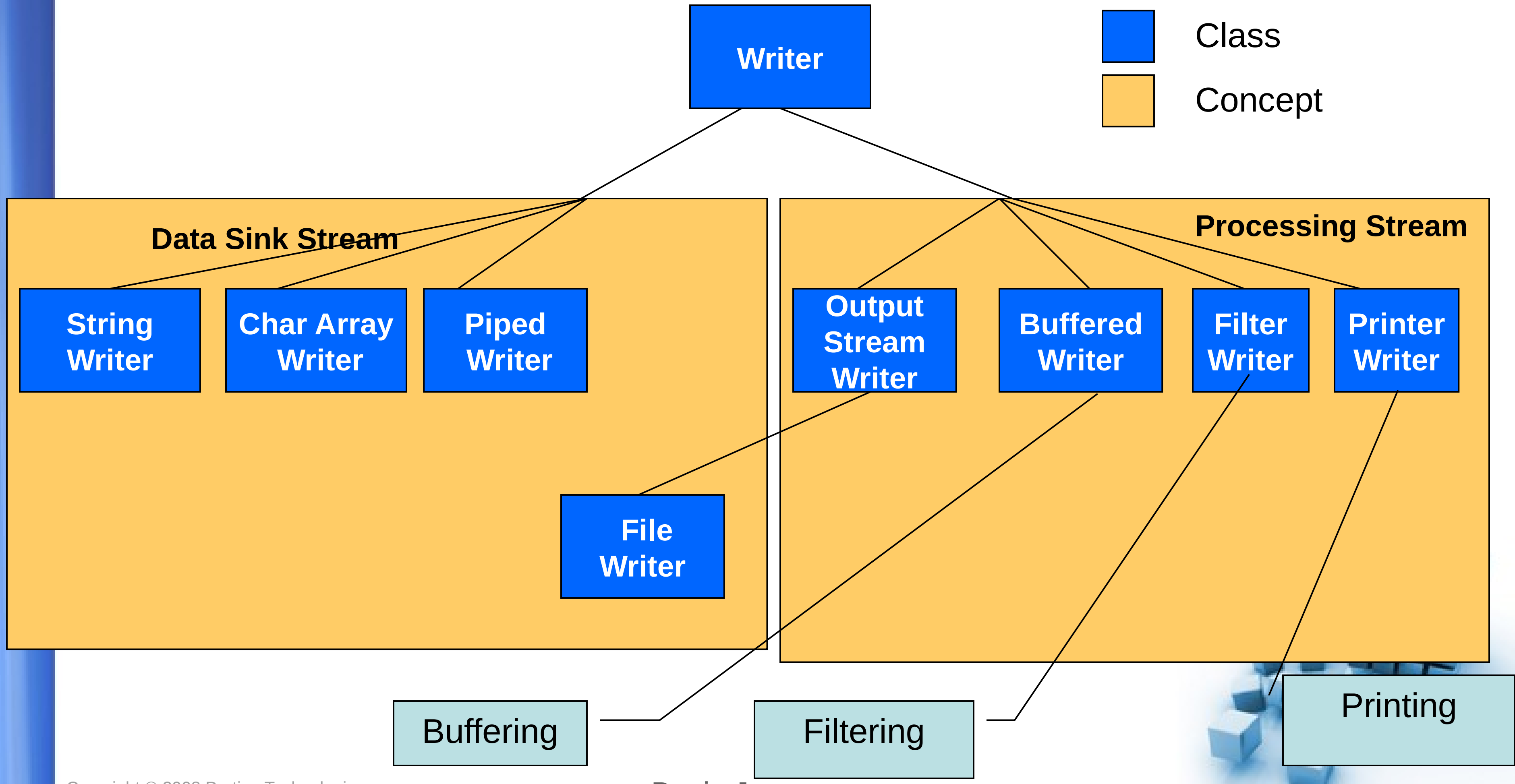
Processing Streams : A Closer look

- Processing streams perform some sort of operation, such as buffering or data conversion, as they read and write.



Processing Streams : A Closer look

- Processing streams perform some sort of operation, such as buffering or data conversion, as they read and write.

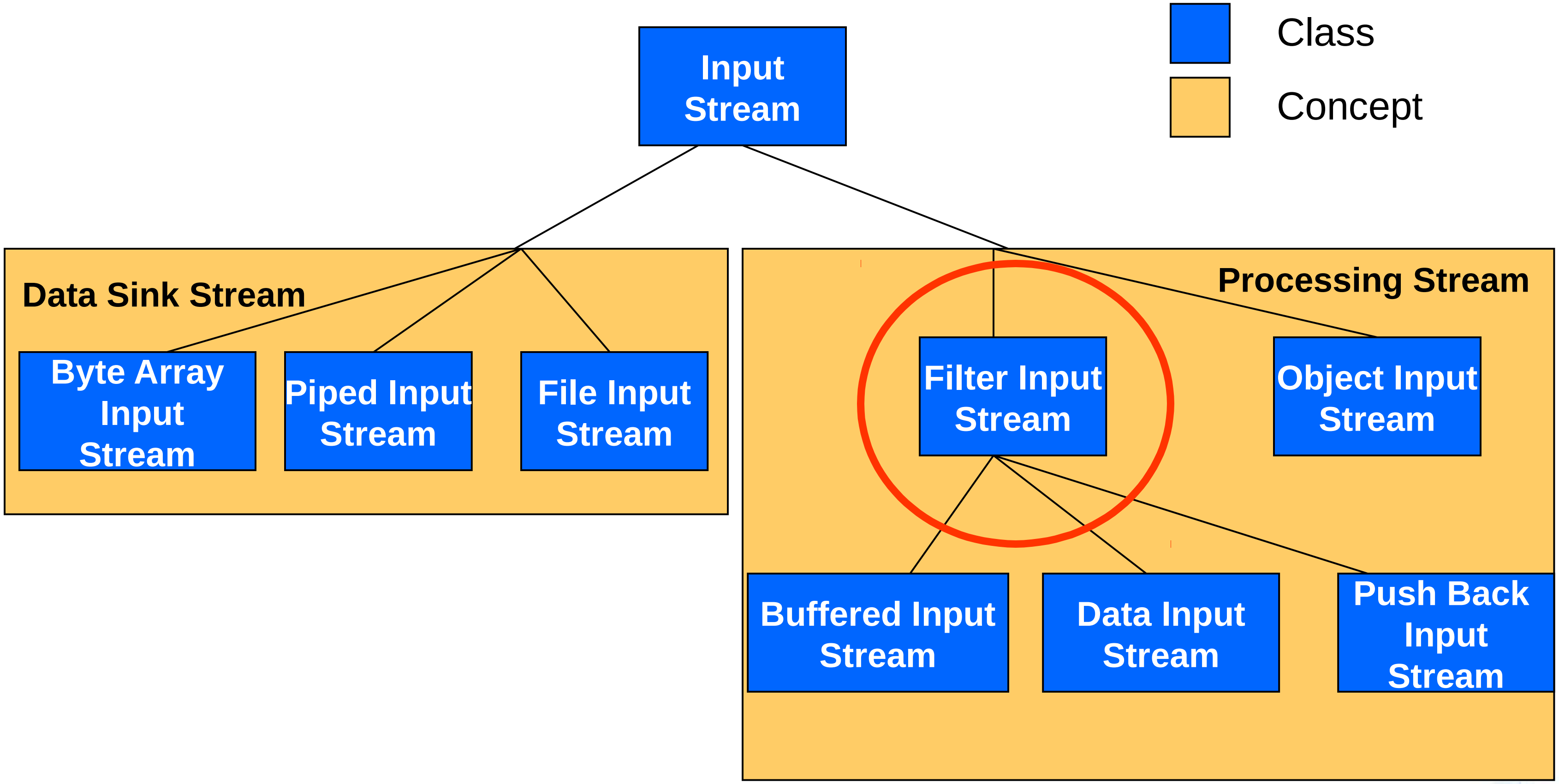


To Summarize: Processing Streams

- Processing streams perform some sort of operation, such as buffering or data conversion, as they read and write.

Process	**CharacterStreams**	**Byte Streams**
Buffering	BufferedReader	BufferedInputStream
	BufferedWriter	BufferedOutputStream
Filtering	FilterReader	FilterInputStream
	FilterWriter	FilterOutputStream
Peeking Ahead	PushbackReader	PushbackInputStream
Printing	PrintWriter	PrintStream

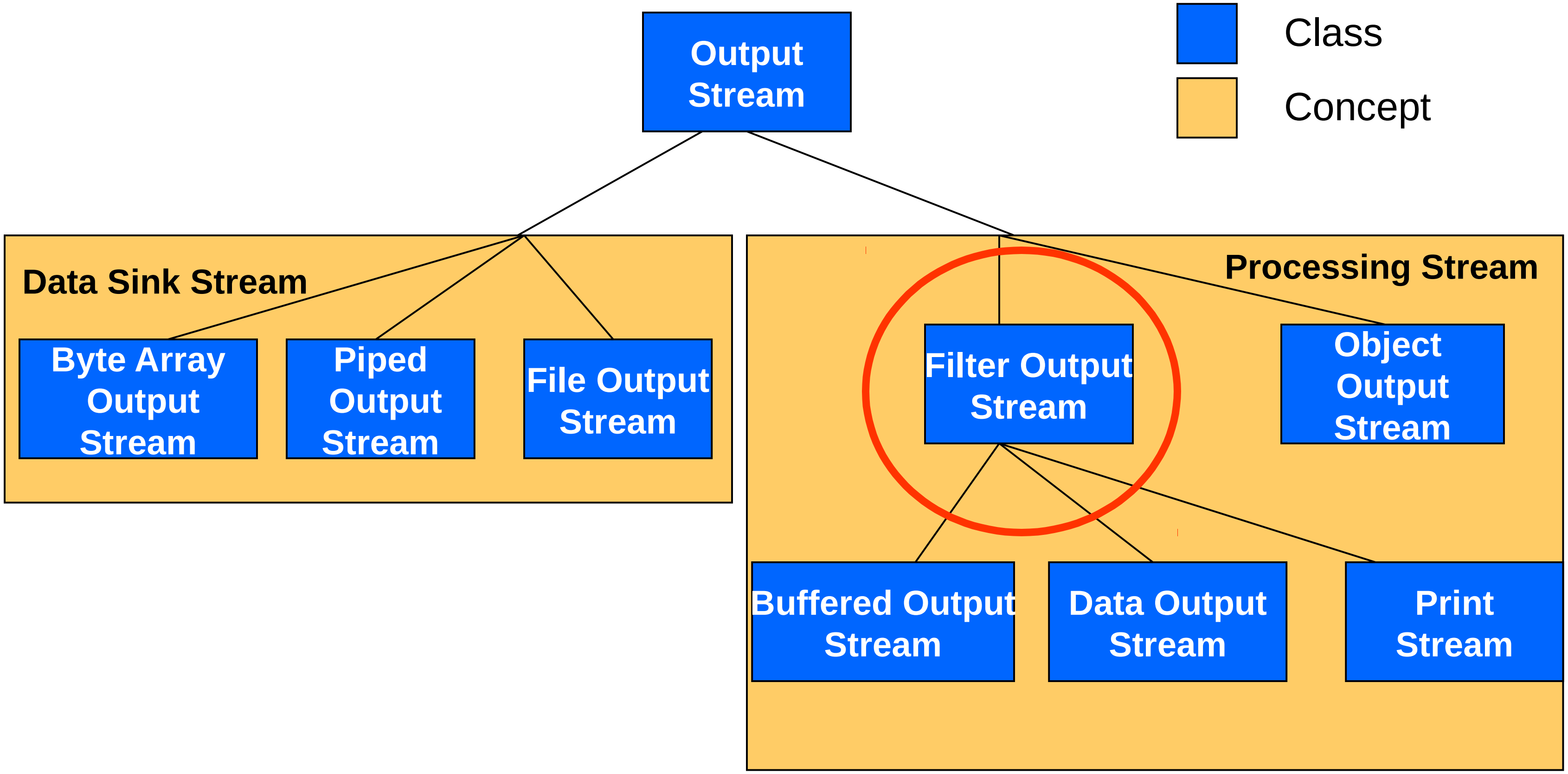
FilterInputStream



FilterInputStream

- public class **FilterInputStream** extends InputStream
  - FilterInputStream is a concrete superclass for all input stream subclasses which somehow modify or manipulate data of an underlying stream.
  - A FilterInputStream contains some other input stream, which it uses to as its basic source of data, possibly to transform the data or for providing additional functionality.
  - The class FilterInputStream itself overrides all methods of InputStream class.
  - Some of the subclasses are,
    - **BufferedInputStream**
    - **DataInputStream**
    - **PushbackInputStream**

FilterOutputStream



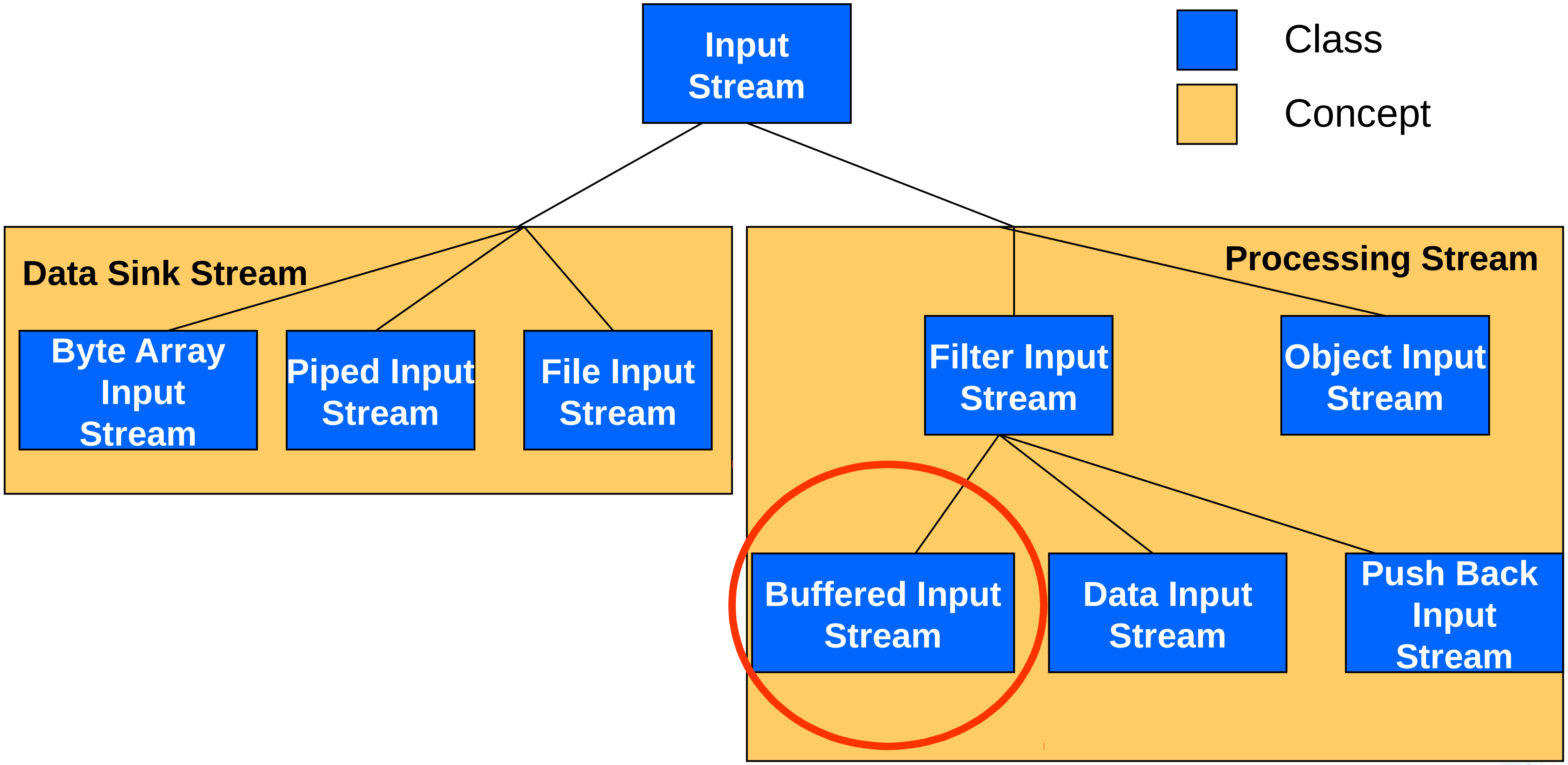
FilterOutputStream

- public class **FilterOutputStream** extends OutputStream
  - FilterOutputStream is a concrete superclass for all output stream subclasses which somehow modify or manipulate data of an underlying stream.
  - A FilterOutputStream sits on top of an already existing output stream, which it uses to as its basic source of data, possibly to transform the data or for providing additional functionality.
  - The class FilterOutputStream itself overrides all methods of InputStream class.
  - Some of the subclasses are,
    - **BufferedOutputStream**
    - **DataOutputStream**
    - **PushbackOutputStream**

Using Buffered Streams

- In unbuffered I/O, each read or write request is handled by the underlying OS. This operation is relatively expensive.
- To reduce the overhead, Java platform implements buffered I/O streams.
- Buffered input streams read data from a memory area known as buffer, the native input API is called only when the buffer is empty.
- Buffered output streams write data to buffer, and the native API is called when the buffer is full.

BufferedInputStream



BufferedInputStream

- public class **BufferedInputStream** extends FilterInputStream
  - A BufferedInputStream adds functionality to another input stream-namely, the ability to buffer the input and to support the *mark* and *reset* methods.
  - When the BufferedInputStream is created, an internal buffer array is created, as bytes from the stream are read or skipped, the internal buffer is refilled as necessary from the contained input stream, many bytes at a time.
  - The *mark* operation remembers a point in the input stream and the *reset* operation causes all the bytes read since the most recent mark operation to be reread before new bytes are taken from the contained input stream.

BufferedInputStream

- Constructors
  - **BufferedInputStream(InputStream in)**
  - **BufferedInputStream(InputStream in , int size)**

    Creates a BufferedInputStream with specified buffer size.

- Some important methods
  - **int read()**

    Reads the next byte of data from the input stream.
  - **int read(byte[ ] b , int off , int len)**

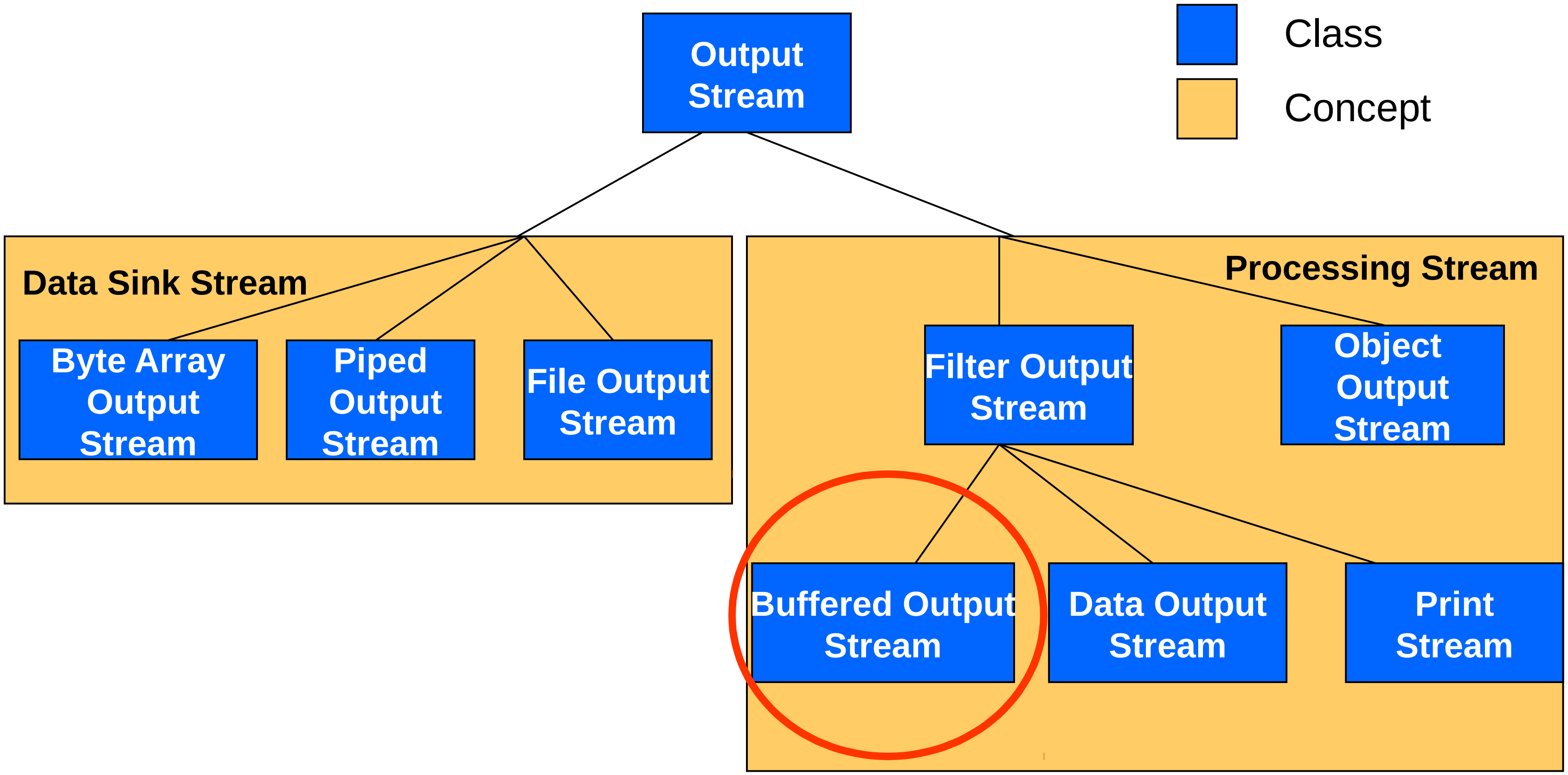
    Reads bytes from this byte-input stream into the specified byte array, starting at the given offset.
  - **void reset()**

    Repositions this stream to the position at the time the mark method was last called on this input stream.
  - **void mark(int readLimit)**

    Marks the current position in this input stream.
  - **void close()**

    Closes this input stream and releases any system resources associated with the stream.

BufferedOutputStream



BufferedOutputStream

- public class **BufferedOutputStream** extends FilterOutputStream
  - When the BufferedInputStream is created, an internal buffer array is created, buffered output streams store data in an internal byte array until the buffer is full or the stream is flushed; then the data is written out to the underlying output stream in one swoop.

- Constructors
  - **BufferedOutputStream(OutputStream out)**
  - **BufferedOutputStream(OutputStream out , int size)**

    Creates a BufferedOutputStream with specified buffer size.

BufferedOutputStream

- Some important methods
  - **void write(int b)**

    Writes the specified byte to this buffered output stream.
  - **void write(byte[] b , int off , int len)**

    Writes len bytes from the specified byte array starting at offset off to this buffered output stream.
  - **void flush()**

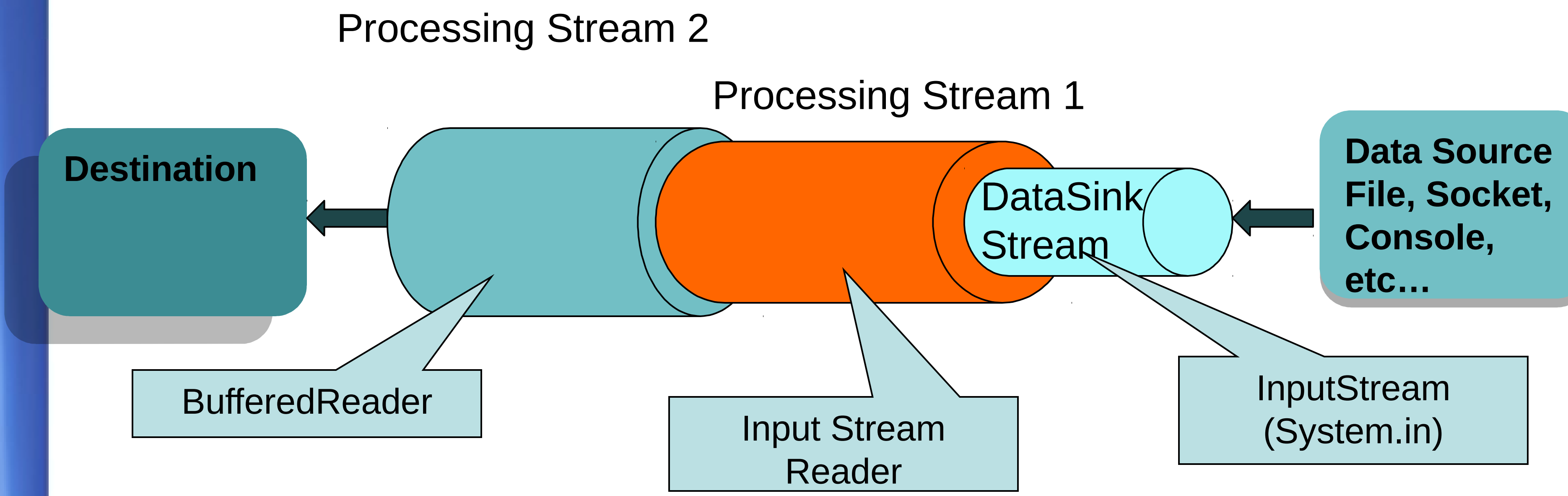
    Flushes this buffered output stream.

Example

```
import java.io.*;
class BufferedStreamDemo
{
        public static void main(String[] args) throws Exception
        {
                int c = 0;
         File file1 = new File(“Dream.jpg”);
         File file2 = new File(“MyDream.jpg”);
         BufferedInputStream in =
                              new BufferedInputStream( new FileInputStream(file1));
         BufferedOutputStream out = new BufferedOutputStream(
                                                  new FileOutputStream(file2));
         while(c != -1)
         {
                 c = in.read();
                 out.write();
                 }
        in.close();
        out.close();
        }
}
```

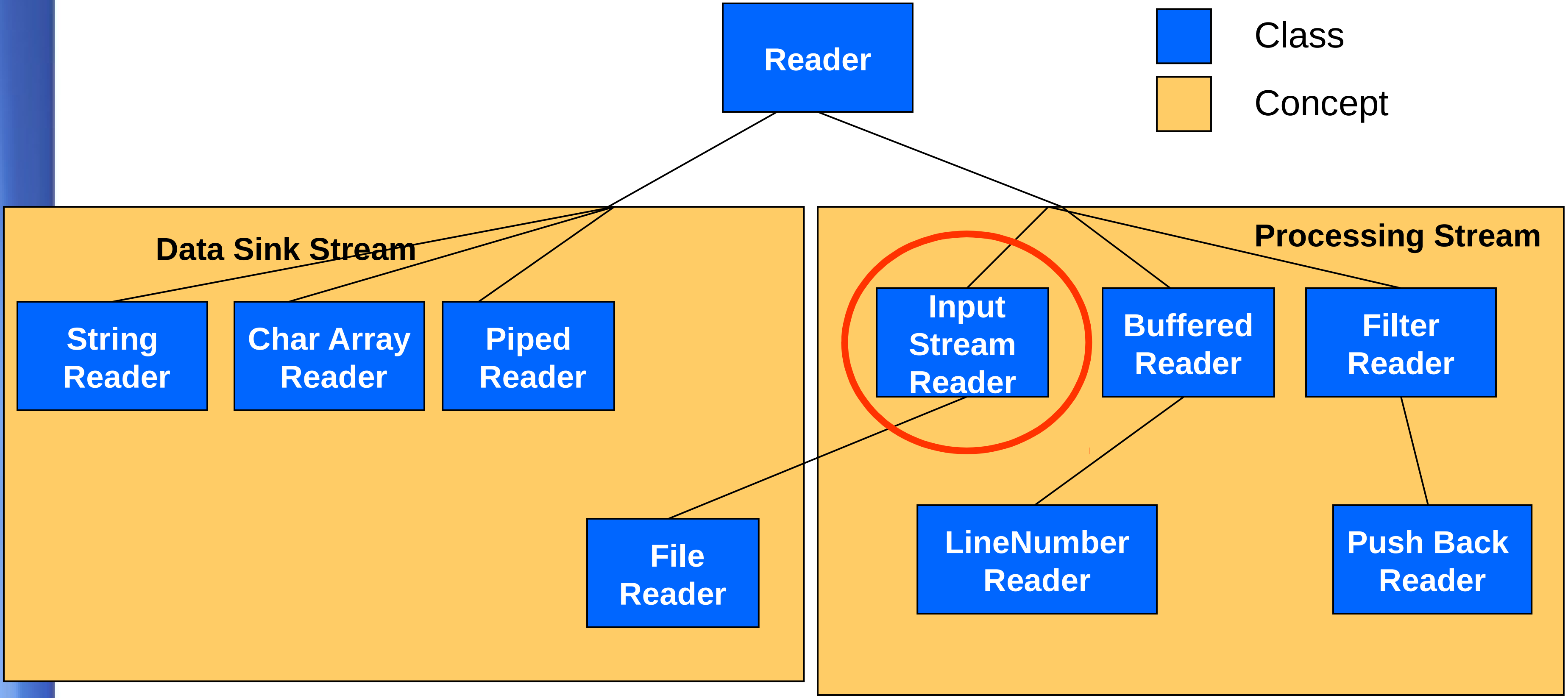
See Listing :BufferedStreamDemo.java

Converting Streams

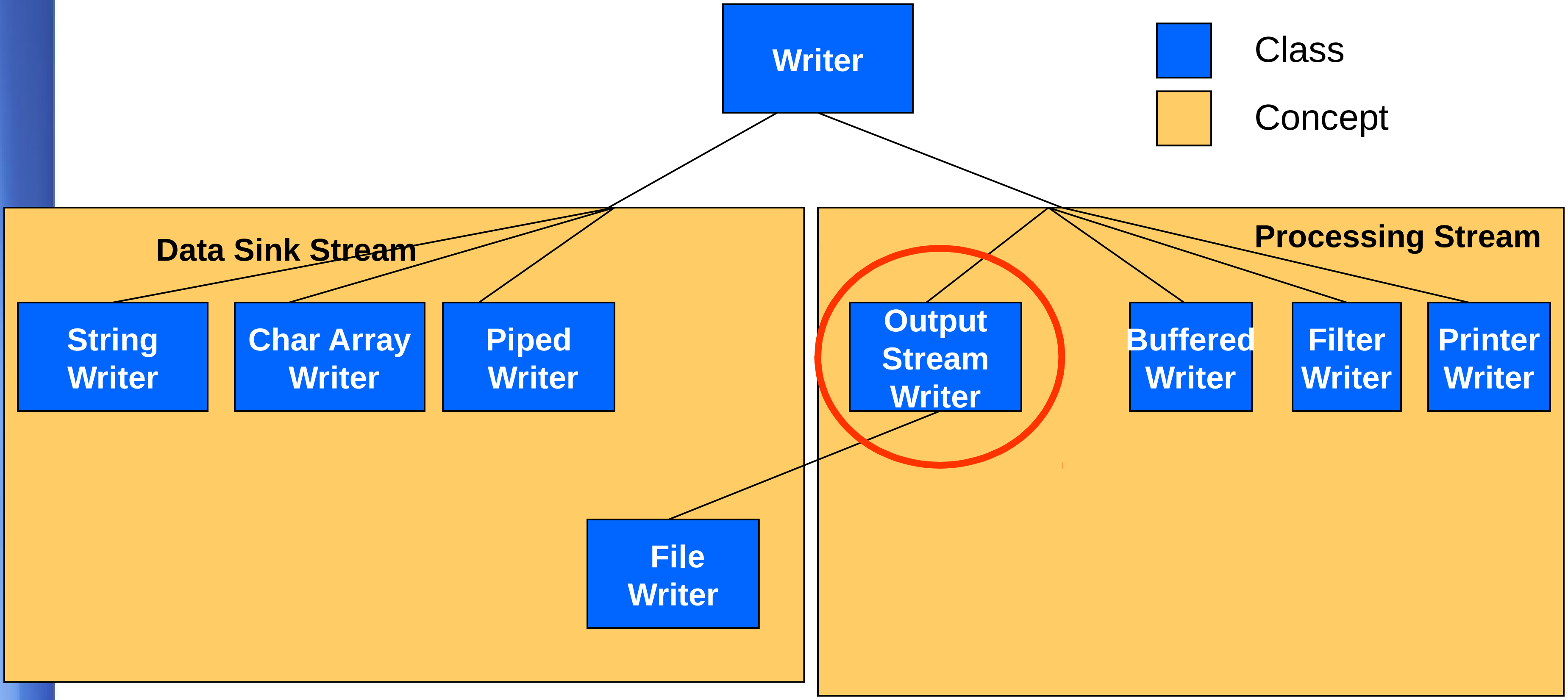


- InputStreamReader or OutputStreamWriter act as bridge between byte streams and character streams.

Converting Streams



Converting Streams



Example

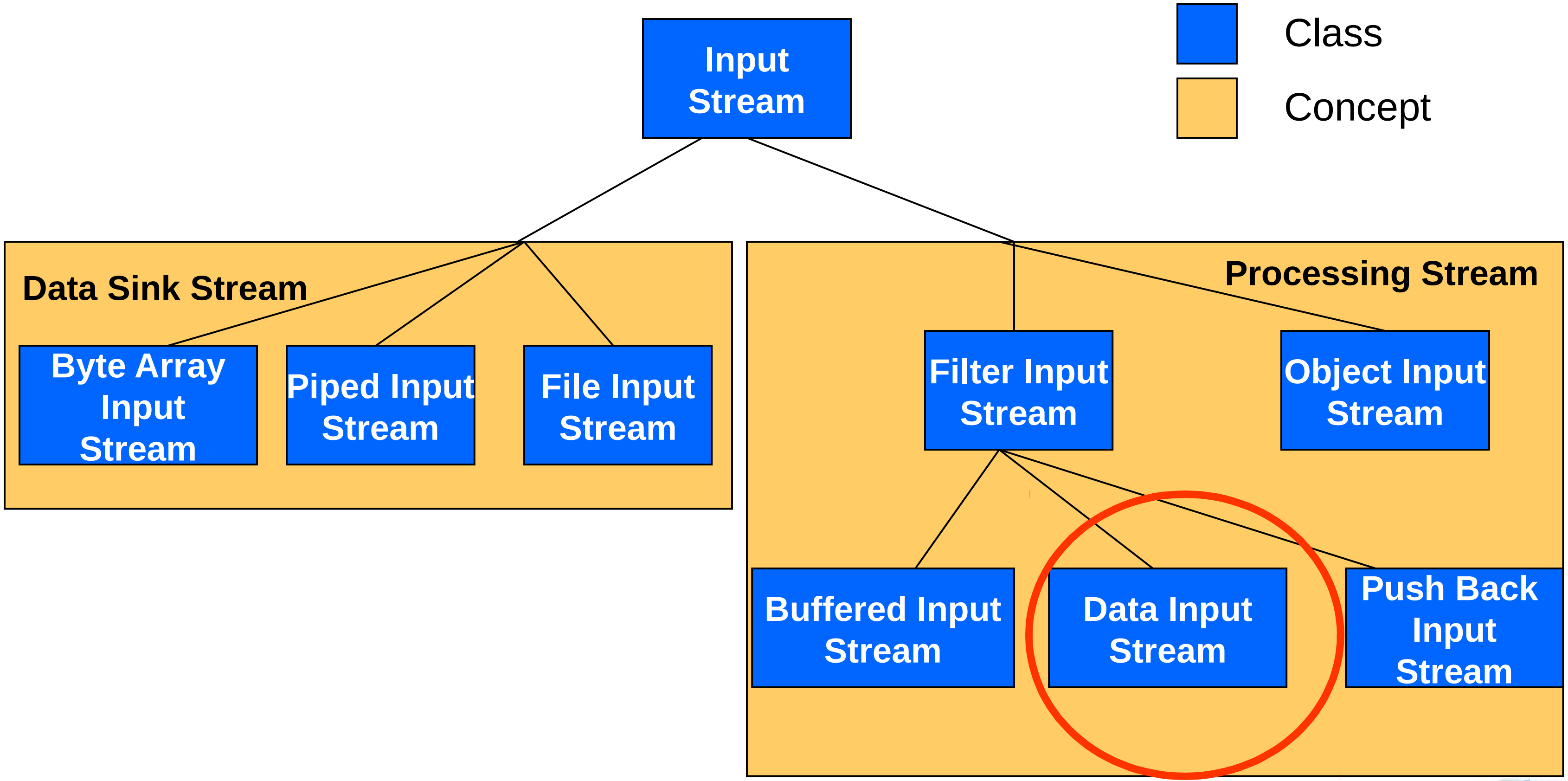
```
import java.io.*;

class InputStreamReaderDemo
{
    public static void main(String[ ] args) throws Exception
        {
                String str = null;
                BufferedReader reader  = new BufferedReader(
                    new InputStreamReader(System.in));
                str = reader.readLine();
                System.out.println(str);
        }
}
```

See Listing :InputStreamReaderDemo.java

Exercise

- Write a program to copy a image file using File streams and Buffered streams , calculate the time taken in both the cases for the copy and display the same.
- Write a program to read a line from the console and send it to an output file.

DataInputStream



DataInputStream

- public class **DataInputStream** extends FilterInputStream implements DataInput
  - The real purpose of DataInputStream is not the read raw bytes using the standard input stream methods, but to read and interpret multibyte data like ints, floats, doubles, and chars written using DataOutputStream.

- Constructors

  DataInputStream(InputStream in)

  Creates a DataInputStream that uses the specified underlying InputStream.

DataInputStream

- Some important methods
  - **int read(byte[ ] b)**

    Reads some number of bytes from the contained input stream and stores them into the buffer array b.
  - **boolean readBoolean()**

    Reads one input byte and returns true if that byte is nonzero, false if that byte is zero.
  - **byte readByte()**

    Reads and returns one input byte.
  - **char readChar()**

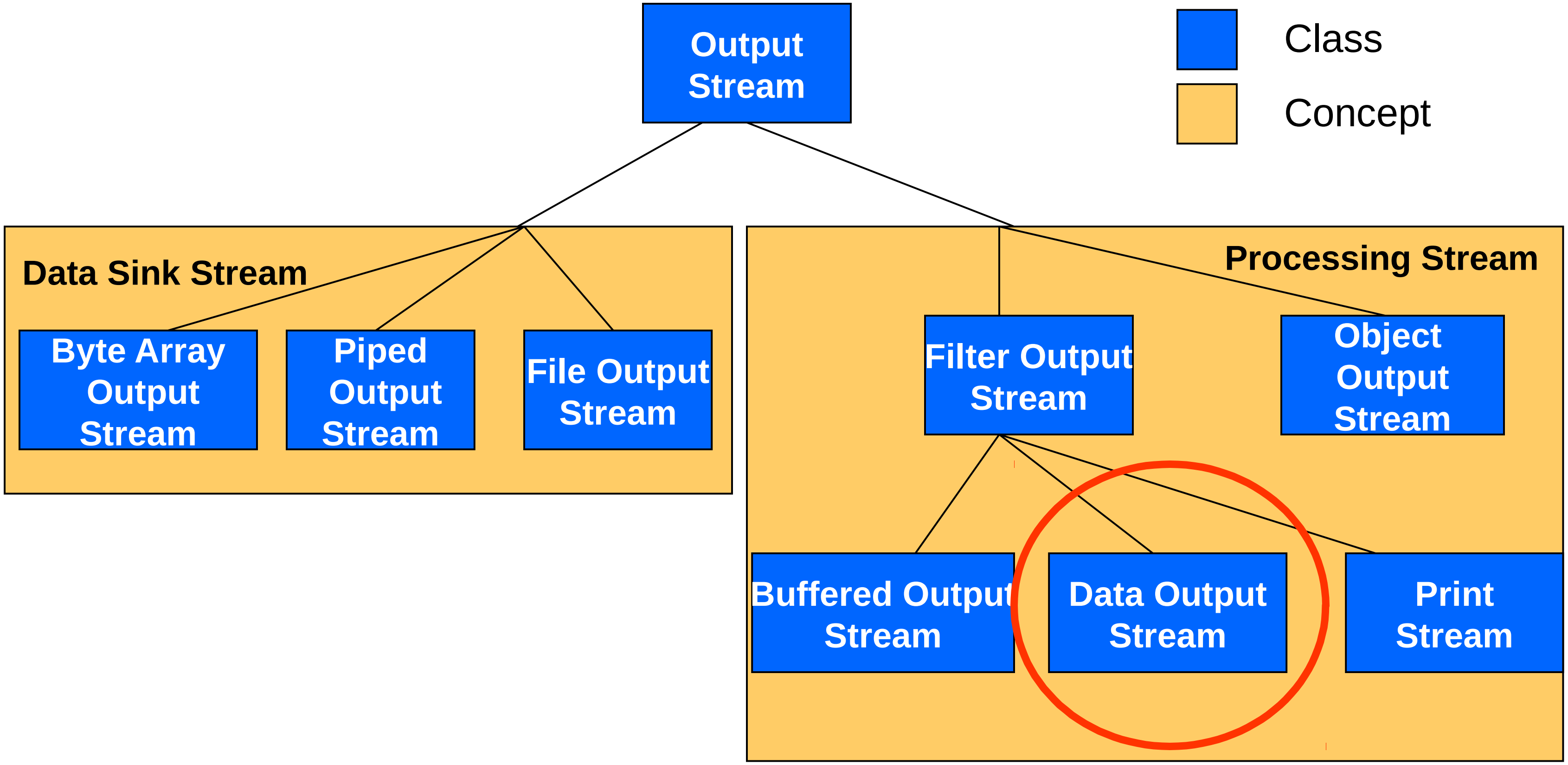
    Reads an input char and returns the char value.
  - **double readDouble()**

    Reads eight input bytes and returns a double value.
  - **int readInt()**

    Reads four input bytes and returns an int value.
  - **String readLine()**

    Reads the next line of text from the input stream.

DataOutputStream



DataOutputStream

- public class **DataOutputStream** extends FilterOutputStream implements DataOutput
  - The real purpose of DataOutputStream is not to write raw bytes using the standard output stream methods, but a data output stream lets an application write primitive Java data types to an output stream in a portable way.
- Constructors

  DataInputStream(InputStream in)

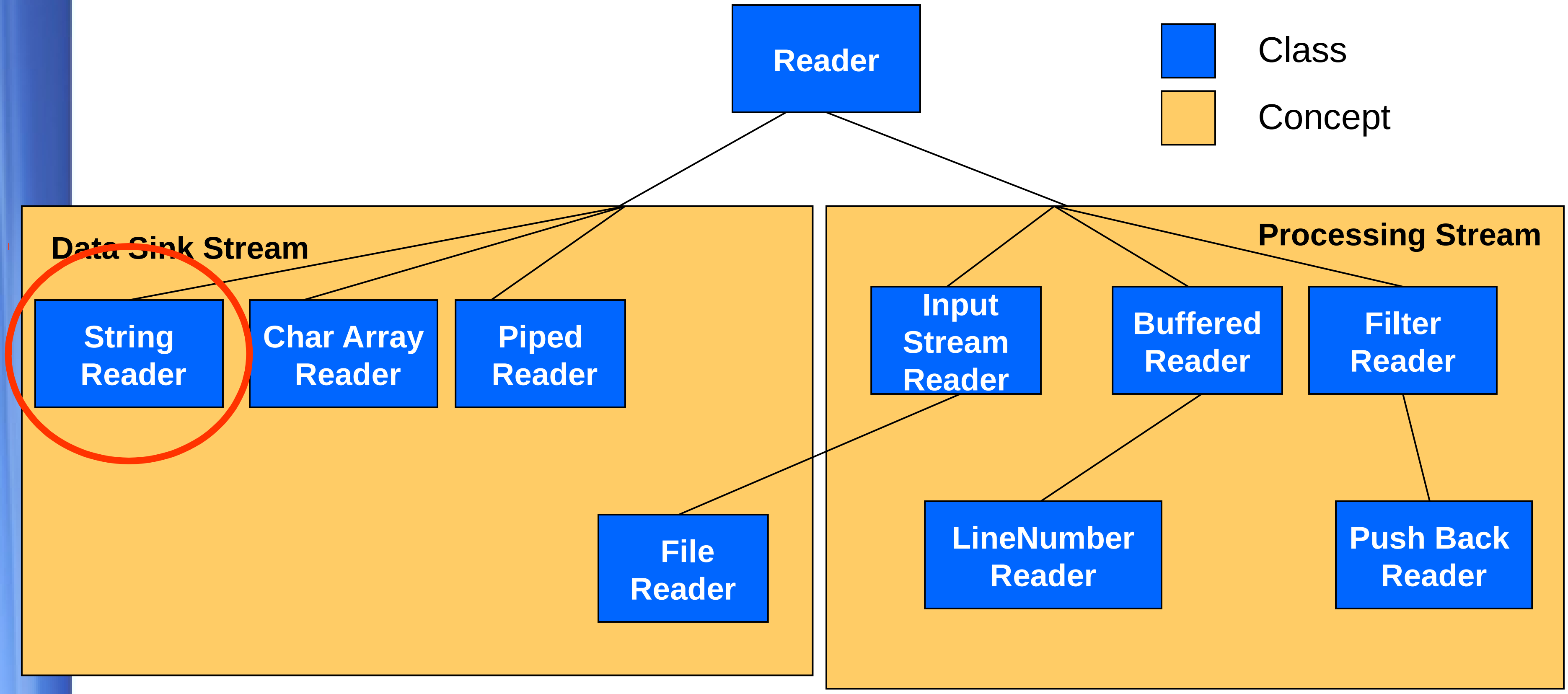
  Creates a DataInputStream that uses the specified underlying InputStream.

DataOutputStream

- Some important methods
  - **void write(int b)**
    Writes the specified byte to the underlying output stream.
  - **void writeBoolean(boolean v)**
    Writes a boolean to the underlying output stream as a 1-byte value.
  - **void writeByte(int v)**
    Writes out a byte to the underlying output stream as a 1-byte value.
  - **void writeChar(int v)**
    Writes a char to the underlying output stream as a 2-byte value,
    high byte first.
  - **void writeInt(int v)**
    Writes an int to the underlying output stream as four bytes, high byte first.
  - **void writeChars(String s)**
    Writes a string to the underlying output stream as a sequence of
    characters.

Sample listing : **DataStreamsDemo.java**

StringReader



StringReader

- public class **StringReader** extends Reader
  - It is a character stream whose source is a string.
  - A StringReader uses the methods of the Reader class to get characters from a string.
  - Since String objects are immutable, the data in the string may not be changed after the StringReader is constructed.
- Constructor
  - **StringReader(String s)**

StringReader

- Some important methods
  - **void close()**

    Close the stream.
  - **void mark(int readAheadLimit)**

    Mark the present position in the stream.
  - **int read()**

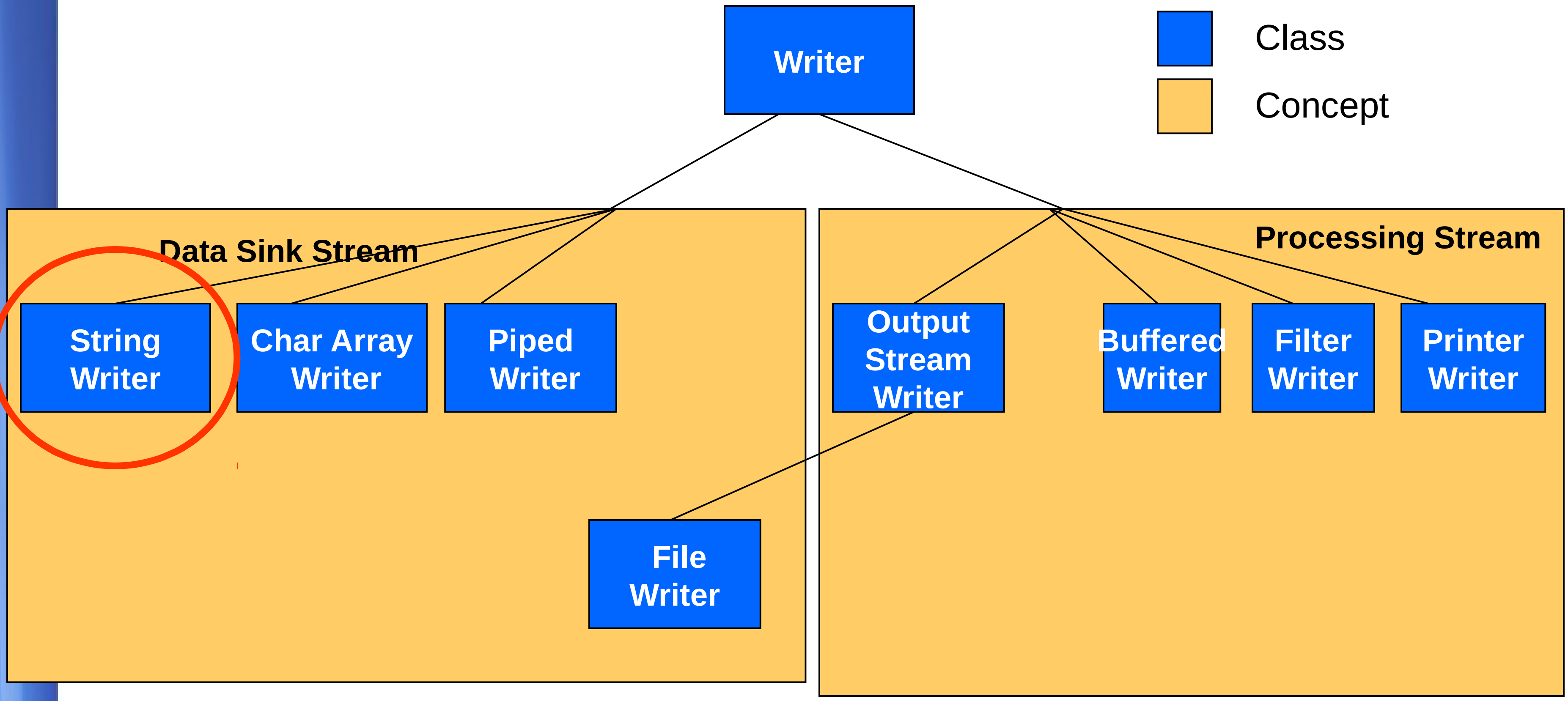
    Read a single character.
  - **int read(char[ ] cbuf , int off , int len)**

    Read characters into a portion of an array.
  - **void reset()**

    Reset the stream to the most recent mark, or to the beginning of the string if it has never been marked.
  - **long skip(long ns)**

    Skips the specified number of characters in the stream.

String Writer



String Writer

- public class **StringWriter** extends Writer
  - StringWriter extends from the Writer class.
  - A StringWriter maintains an internal StringBuffer to which it appends characters.
  - This buffer can easily be converted to a string as necessary.
- Constructors
  - **StringWriter()**
  - **StringWriter(int intialSize)**

    Create a new string writer, using the specified initial string-buffer size.

String Writer

- Some important methods
  - StringWriter append(char ch)

    Appends the specified character to this writer.
  - StringBuffer getBuffer()

    Returns the internally used string buffer.
  - String toString()

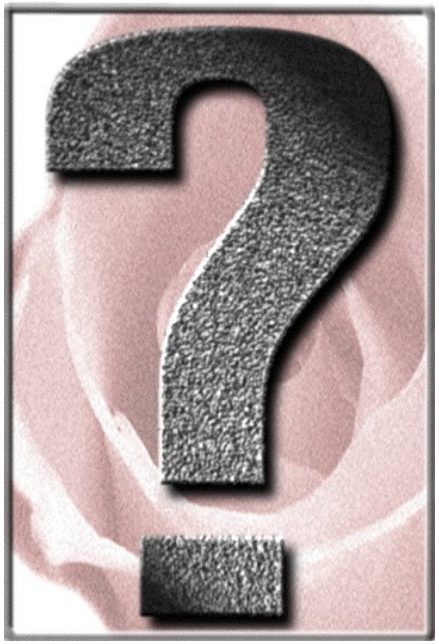
    Return the buffer's current value as a string.
  - void write(int c)

    Write a single character.
  - void write(String str)

    Write a string.

Sample listing : StringWriterDemo.java

Question time



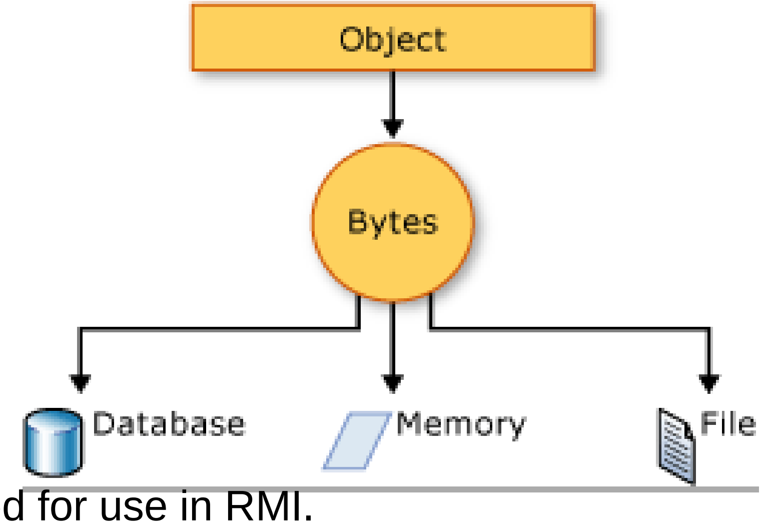
Please try to limit the questions to the topics discussed during the session.

Participants are encouraged to discuss other issues during the breaks.

Thank you.

Object Serialization

Serialization of an object just means writing the values of all its data fields into a stream of bytes.



- **Need for Serialization**
  - Serialization in Java was first developed for use in RMI.
  - RMI allows an object in one virtual machine to invoke methods in an object in another virtual machine, which may be another computer in the network.
  - This requires a way to convert those arguments and return values to and from byte streams. It's a trivial task for primitive data types, but this should be achieved for objects as well.
  - It may be necessary to persist an object, wherein the object is written to disk.
  - This is achieved using object serialization.

Serializable interface

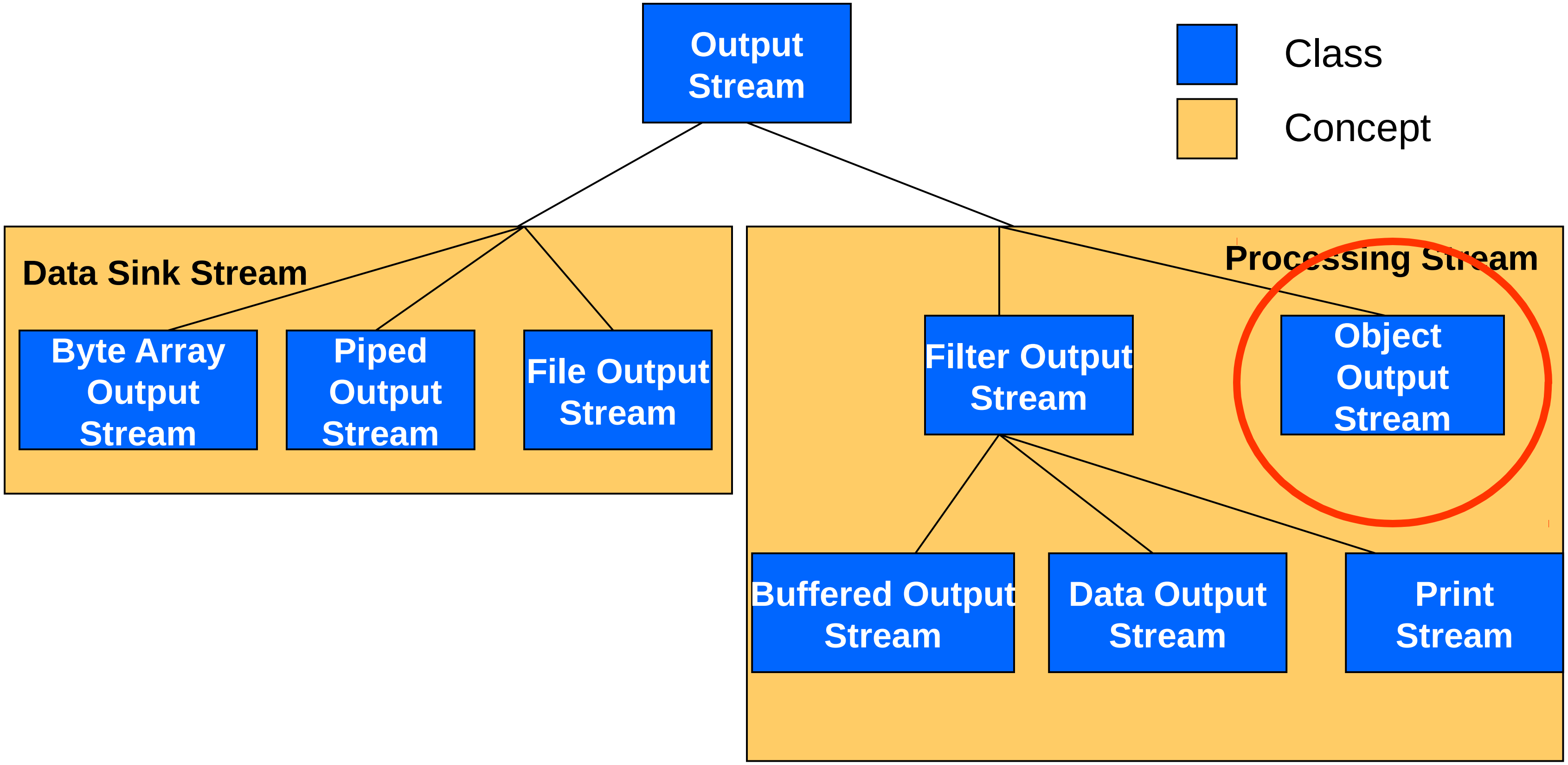
- The serializability of a class is enabled by the class **implementing** java.io.**Serializable** interface
- Classes that do not implement this interface will not have any of their state serialized or deserialized.
- Serializable is a **marker interface**, that is, it has no methods or fields and serves only to indicate that a class may be serialized.
- All subtypes of a serializable class are themselves serializable.
- A class can be serialized by using the **ObjectOutputStream** class.
- A class can be deserialized by using the **ObjectInputStream.**

```
class MyClass
implements
Serializable
{

}
```

Nonserializable references

- A common problem that prevents a serializable class from being serialized is that its ***graph*** contains objects that do not implement Serializable.
- The ***graph*** of an object is the collection of all objects that the object holds references to, and all the objects those objects hold references to, and all the objects those objects hold references to, and so on, until there are no more connected objects that haven't appeared in the collection.
- For an object to be serialized, all the objects it holds references to must also be serializable, and all the objects they hold references to must be serializable, and so on.

ObjectOutputStream



PRATIAN TECHNOLOGIES

ObjectOutputStream

- public class **ObjectOutputStream** extends OutputStream implements ObjectOutput , ObjectStreamConstants
- Objects are serialized by using the ObjectOutputStream.
- An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream.
- The objects can be reconstituted using a ObjectInputStream.
- Persistent storage of objects can be accomplished by using a file for the stream. If the stream is a network socket stream, the objects can be reconstituted on another host or in another process.
- The class of each serializable object is encoded including the class name and signature of the class, the values of the object's fields and arrays, and the closure of any other objects referenced from the initial objects.

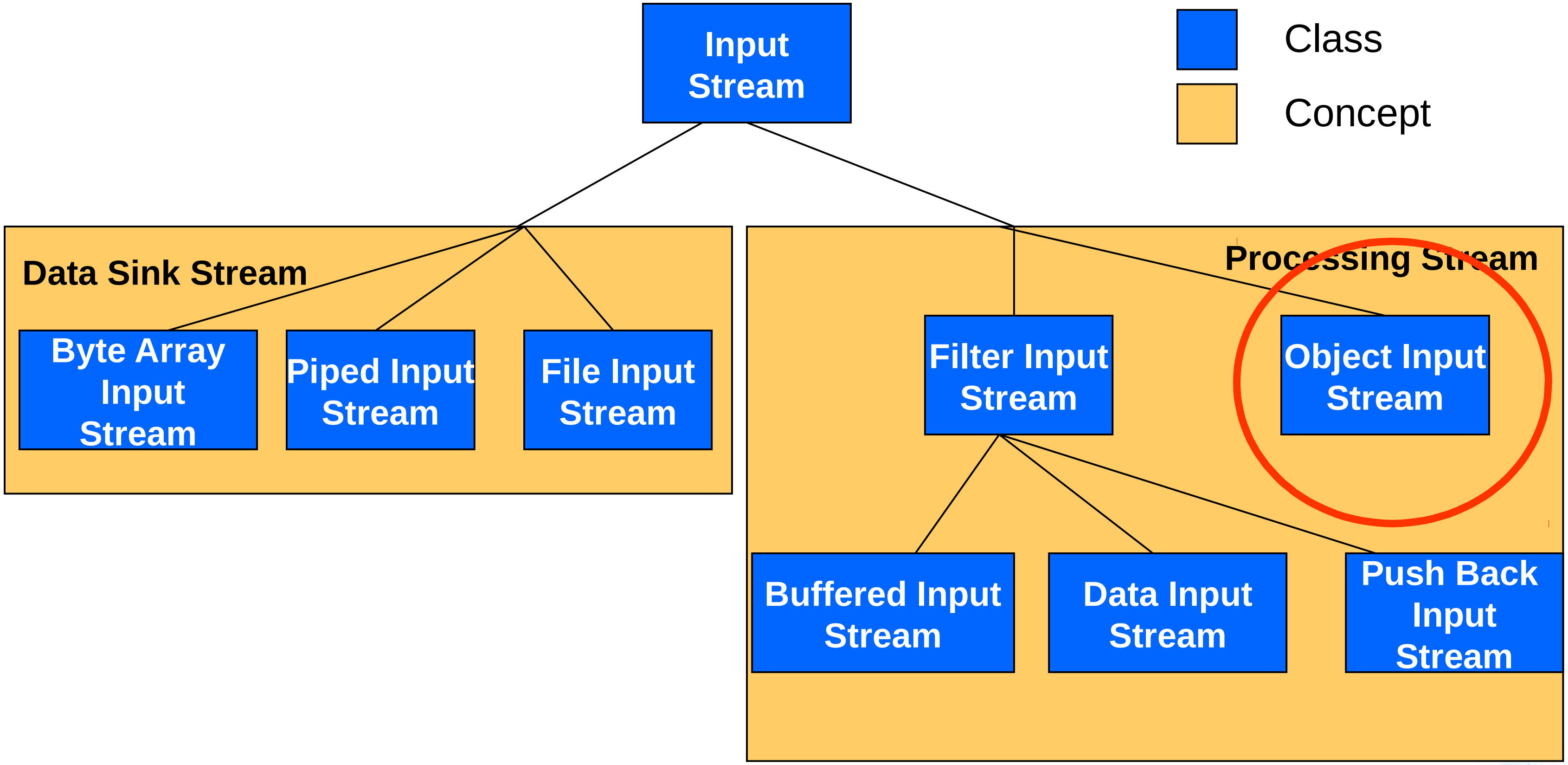
ObjectOutputStream

- Constructors
  - **protected ObjectOutputStream()**
  - **ObjectOutputStream(OutputStream out)**

    Creates an ObjectOutputStream that writes to the specified OutputStream.
- Some important methods
  - **void writeObject(Object obj)**

    The writeObject method is responsible for writing the state of the object for its particular class so that the corresponding readObject method can restore it.

ObjectInputStream



ObjectInputStream

- public class ObjectInputStream extends InputStream implements ObjectInput , ObjectStreamsConstants
- Objects are deserialized by using the ObjectInputStream.
- An ObjectInputStream deserializes primitive data and objects previously written using an ObjectOutputStream.
- ObjectInputStream is used to recover those objects previously serialized.
- Other uses include passing objects between hosts using a socket stream or for marshaling and unmarshaling arguments and parameters in a remote communication system.

ObjectInputStream

- Constructors
  - **protected ObjectInputStream()**
  - **ObjectInputStream(InputStream in)**

    Creates an ObjectInputStream that reads from the specified InputStream.
- Some important methods
  - **Object readObject()**

    The readObject method is responsible for reading and restoring the state of the object for its particular class using data written to the stream by the corresponding writeObject method.

Sample listing : **ObjectStreamDemo.java**

Exercise

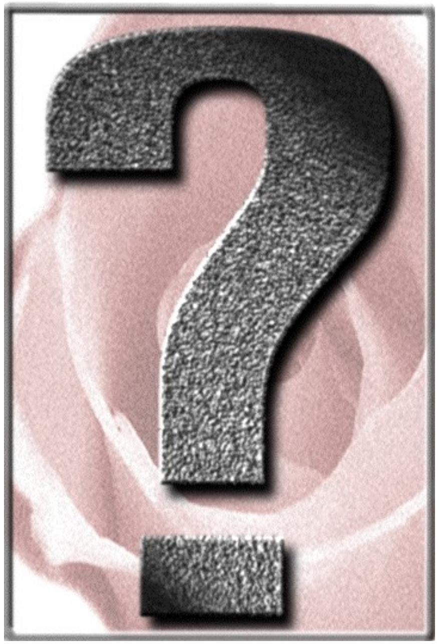
- Write a class GameInfo with data members, int points and double time.
- Write a class Game with an instance of GameInfo and methods

  Play() , set some arbitrary value to points and time of GameInfo object.

  saveGame(), which saves the GameInfo object state to a file.

  restoreGame() which reads object state from a file and displays the same.

Question time



Please try to limit the questions to the topics discussed during the session.

Participants are encouraged to discuss other issues during the breaks.

Thank you.

Opening a stream

- Open a stream
  - Make a connection to an external place
- External place indicates the data external to your program that you want to get from or put into
- Once the connection is made, you forget about the external place and just use the stream

```
    FileInputStream in = new FileInputStream(sourceFileName);
```

Using a stream

- Some streams can be used only for input, others only for output, still others for both
- *Using* a stream means doing input from it or output to it
- But it’s not usually that simple--you need to manipulate the data in some way as it comes in or goes out

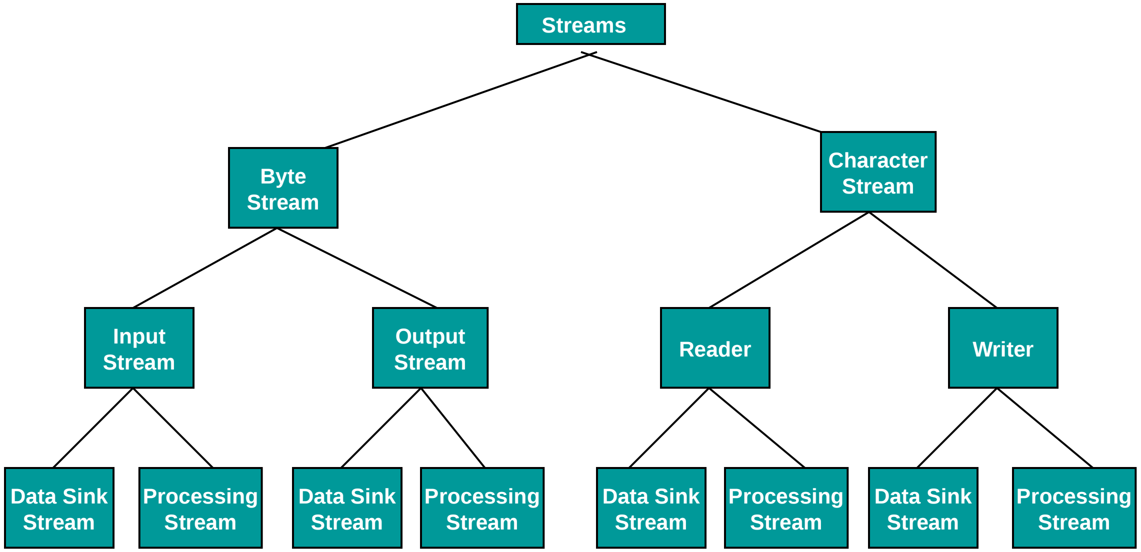
```
int val = in.read();

while(val != -1) {
    out.write(val);
    val = in.read();
}
```

Closing the Stream

- A stream is an expensive resource
- There is a limit on the number of streams that you can have open at one time
- You should not have more than one stream open on the same file
- You must close a stream before you can open it again
- *Always close your streams!*

```
in.close();
out.close();
```

Input Stream Classes



Example ?

```
import java.io.*;
public class StreamPrinter {
    public static void main(String[ ] args) {
         try {
              while (true)  {
             int data = System.in.read( );
             if (data == -1) break;
             System.out.write(data);
              }
         }
         catch (IOException ex) {
         System.out.println("Couldn't read from System.i
         }
     }
}
```

Using the InputStream to read

Using the OutputStream to write