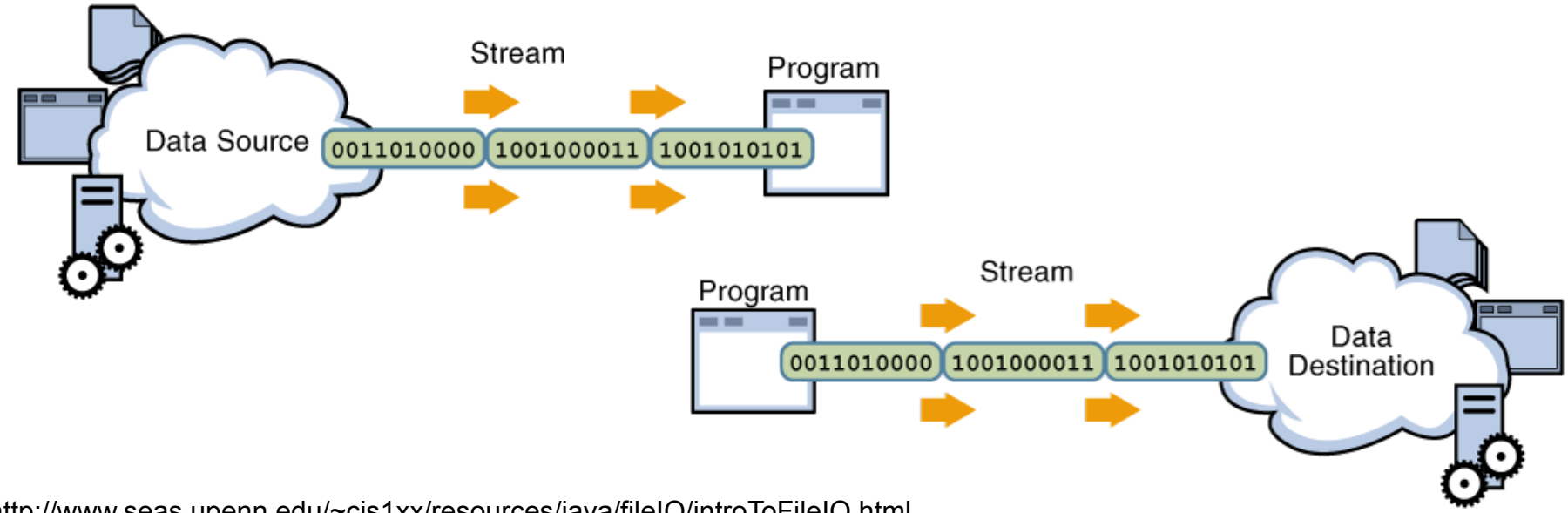
SchoolNova

IT101

File Input and Output

IO Streams

- A stream is a communication channel that a program has with the outside world. It is used to transfer data items in succession.
- An Input/Output (I/O) Stream represents an input source or an output destination. A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.
- Streams support many different kinds of data, including simple bytes, primitive data types, localized characters, and objects. Some streams simply pass on data; others manipulate and transform the data in useful ways.
- No matter how they work internally, all streams present the same simple model to programs that use them: A stream is a sequence of data.



http://www.seas.upenn.edu/~cis1xx/resources/java/fileIO/introToFileIO.html

The java.io.* Package

- The java.io package contains many classes that your programs can use to read and write data. Most of the classes implement sequential access streams. The sequential access streams can be divided into two groups:
  - those that read and write bytes
  - those that read and write Unicode characters.
- Each sequential access stream has a speciality, such as reading from or writing to a file, filtering data as its read or written, or serializing an object.
- Byte Streams
  - Byte streams perform input and output of 8-bit bytes. They read and write data one byte at a time. Using byte streams is the lowest level of I/0, so if you are reading or writing character data the best approach is to use character streams. Other stream types are built on top of byte streams.
- Character Streams
  - All character stream classes are descended from Reader and Writer. We will look at two examples of writing programs using character streams, one that reads and writes one character at a time and one that reads and writes one line at a time.

Simple IO Program

```
import java.io.*;

  public class IOTest {

    public void copy() throws IOException {
        FileReader inputStream = null;
        FileWriter outputStream = null;
        try {
            inputStream = new FileReader("/Users/serge/Desktop/test.txt");
            outputStream = new FileWriter("/Users/serge/Desktop/output.txt");
            int c;
            while ((c = inputStream.read()) != -1) {
                outputStream.write(c);
            }
            inputStream.close();
             outputStream.close();
        } catch (IOException e) {
            System.out.println("Can not perform read or write: " +
e.getMessage());
        }
    }

  public static void main (String args[]) throws IOException {
    IOTest iot = new IOTest();
    iot.copy();
  }

 }
```

- What does the code on the left do?
- Note that FileReader.read method reads one character at a time and returns an int. Why?
- Note the “try” block. Why is it needed? What is the “finally” block?

USASCII code chart

Row	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
10	LF	SUB	*	:	J	Z	j	z
11	VT	ESC	+	;	K	[	k	{
12	FF	FS	,	<	L	\	l	|
13	CR	GS	-	=	M	]	m	}
14	SO	RS	.	>	N	^	n	~
15	SI	US	/	?	O	_	o	DEL

Reading Character Files

```
import java.io.*;

 public class IOTest {

    public static void main(String[] args) {
        BufferedReader inputStream = null;
        PrintWriter outputStream = null;
        try {
            inputStream = new BufferedReader(new FileReader("/Users/serge/Desktop/test.txt"));
            outputStream = new PrintWriter(new FileWriter("/Users/serge/Desktop/output.txt"));
            String l;
            while ((l = inputStream.readLine()) != null) {
                outputStream.println(l);
            }
            inputStream.close();
            outputStream.close();
        } catch (IOException e) {
            System.out.println("Can not perform read or write: " + e.getMessage());
        }
    }

}
```

- Character I/O is usually processed in units longer than single characters. One common unit is the line: a string of characters with a line terminator at the end. A line terminator can be a carriage-return/line-feed sequence ("\r\n"), a single carriage-return ("\r"), or a single line-feed ("\n"). Supporting all possible line terminators allows programs to read text files created on any of the widely used operating systems.

Homework

- Write a program that creates a file on your computer and writes a word “Hello” into the file.
- Modify your program such that it prompts the user for one line of input and then writes the input into a file.
  - Use InputStreamReader(Standard.in) or Scanner(Standard.in) to read the user input:
    - Scanner input = new Scanner(System.in);
  - Use BufferedReader to read the entire line, instead of character by character;
  - Use PrintWriter to write the line (String) to a file.
- Optional: modify your program such that it transforms the text (e.g. toUpperCase) and then writes to a file.