جامعة الأهرام الكندية
AHRAM CANADIAN UNIVERSITY

Lecture (02)
Streams & Serialization (PI)

Dr. Ahmed ElShafee

Agenda

- Introduction
- Streams
- Data Sources

Introduction

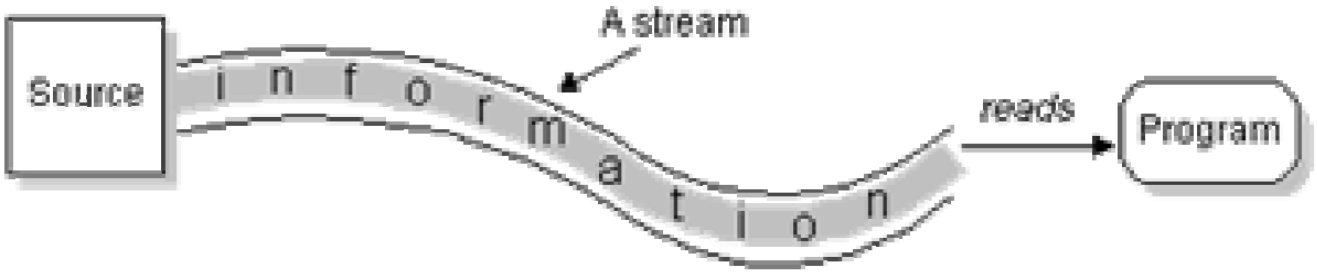
- Most programs use data in one form or another, whether as input, output, or both.
- The sources of input and output can vary between a local file, a socket on the network, a database, variables in memory, or another program.
- Even the type of data can vary between objects, characters, multimedia, and others

Introduction (2)

To bring data into a program:

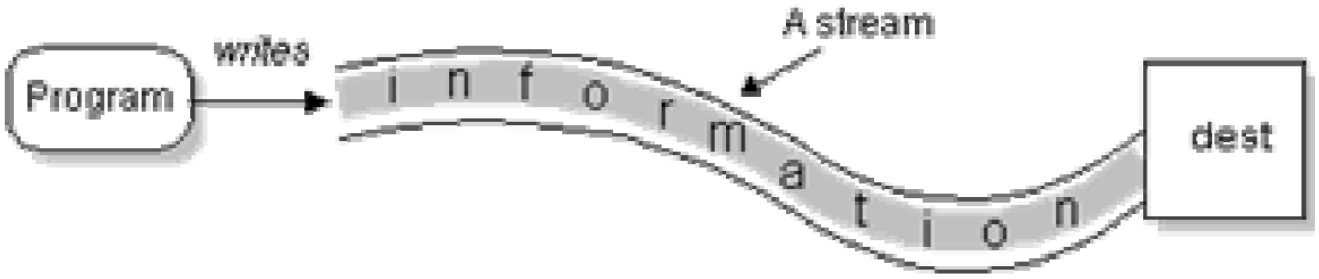
1) opens a stream to a data source, such as a file or remote socket

2) and reads the information serially



Introduction (3)

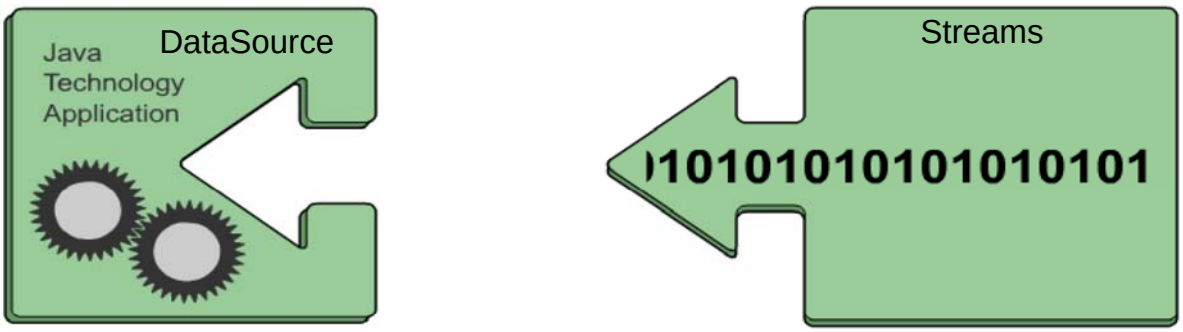
- On the flip side, a program can open a stream to a data source and write to it in a serial fashion.



Streams

In Java IO streams are flows of data you can either read from, or write to.

streams are typically connected to a data source, or data destination, like a file, network connection etc.



Streams (cont),..

- A stream has no concept of an index of the read or written data, like an array does. Nor can you typically move forth and back in a stream, like you can in an array, or in a file using RandomAccessFile.
- A stream is just a continuous flow of data.
- In Java IO streams are typically byte based. This means that you can either read bytes from, or write bytes to a stream.
- If you need to read / write characters (like Latin1 or UNICODE characters), you should use a Reader or Writer.

Streams (cont),..

Types of Streams

There are two categories of streams:

1) 8-bit byte streams
   - java.io.InputStream , abstract class
   - java.io.OutputStream , abstract class

2) 16-bit Unicode character streams

Support for byte streams are provided by:

   - java.io.Reader , abstract class
   - java.io.Writer , abstract class

Streams (cont),..

InputStream

The class java.io.InputStream is the base class for all Java IO input streams.

If you are writing a component that needs to read input from a stream, try to make our component depend on an InputStream, rather than any of it's subclasses (e.g. FileInputStream).

You typically read data from an InputStream by calling the read() method.

The read() method returns a int containing the byte value of the byte read.

If there are no more data to be read, the read() method typically returns -1

Streams (cont),..

```
InputStream input = new FileInputStream("c:\\data\\input-text.txt");

int data = input.read();
while(data != -1) {
  //do something with data...
  doSomethingWithData(data);

  data = input.read();
}
input.close();
```

Streams (cont),..

OutputStream

- The class java.io.OutputStream is the base class of all Java IO output streams.
- If you are writing a component that needs to write output to a stream, try to make sure that component depends on an OutputStream and not one of its subclasses.

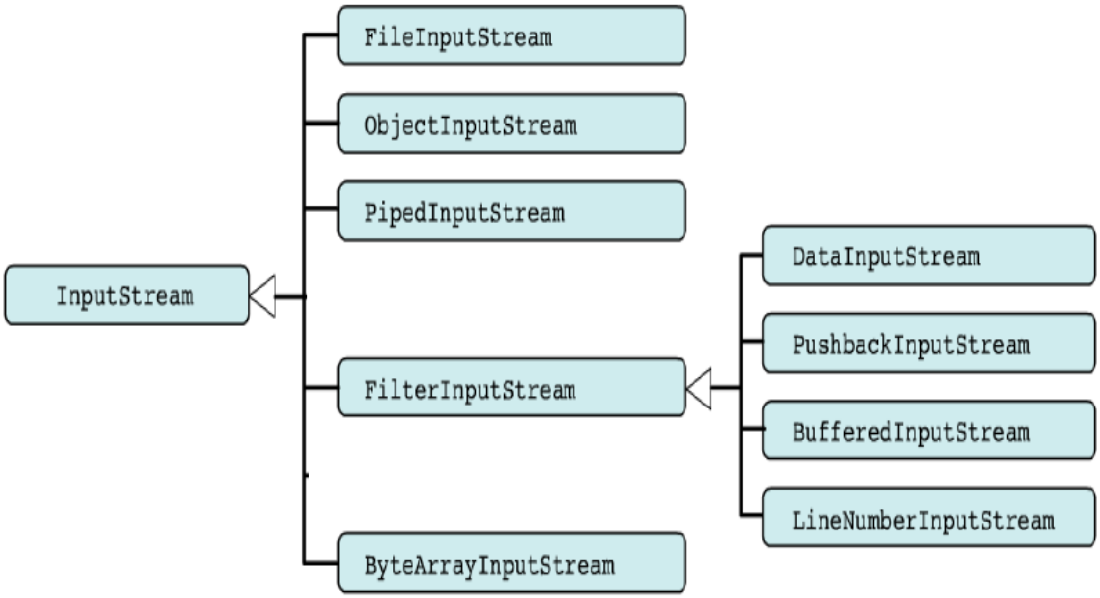
Streams (cont),..

```
OutputStream output = new FileOutputStream("c:\\data\\output-text.txt");

while(moreData) {
  int data = getMoreData();
  output.write(data);
}
output.close();
```

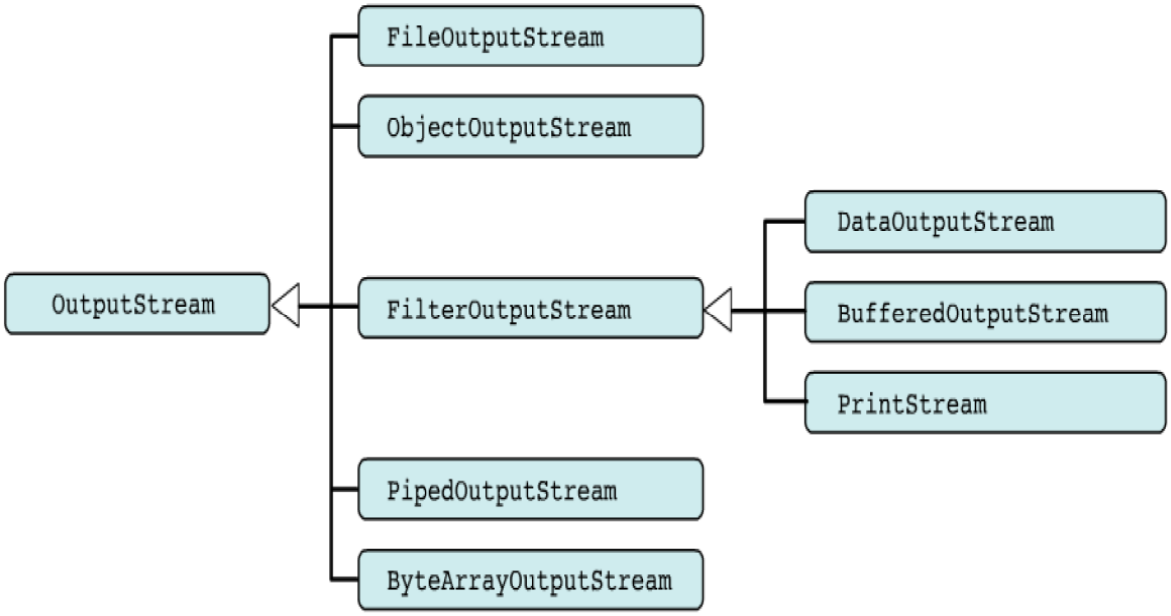
Streams (cont),..

InputStream Hierarchy



Streams (cont),..

OutputStream Hierarchy



Streams (cont),..

Reader and Writers

- Java IO's Reader and Writer work much like the InputStream and OutputStream with the exception that Reader and Writer are character based.
- They are intended for reading and writing text.

Streams (cont),..

Reader

- The Reader is the base class of all Reader's in the Java IO API.
- Subclasses include a BufferedReader, FileReader,…

```
Reader reader = new FileReader();

int data = reader.read();
while(data != -1){
    char dataChar = (char) data;
    data = reader.read();
}
```

Streams (cont),..

- Notice, that while an InputStream returns one byte at a time, meaning a value between -128 and 127 (0 ~ 255), the Reader returns a char at a time, meaning a value between 0 and 65535.
- This does not necessarily mean that the Reader reads two bytes at a time from the source it is connected to.
- It may read one or more bytes at a time, depending on the encoding of the text being read.

Streams (cont),..

Writer

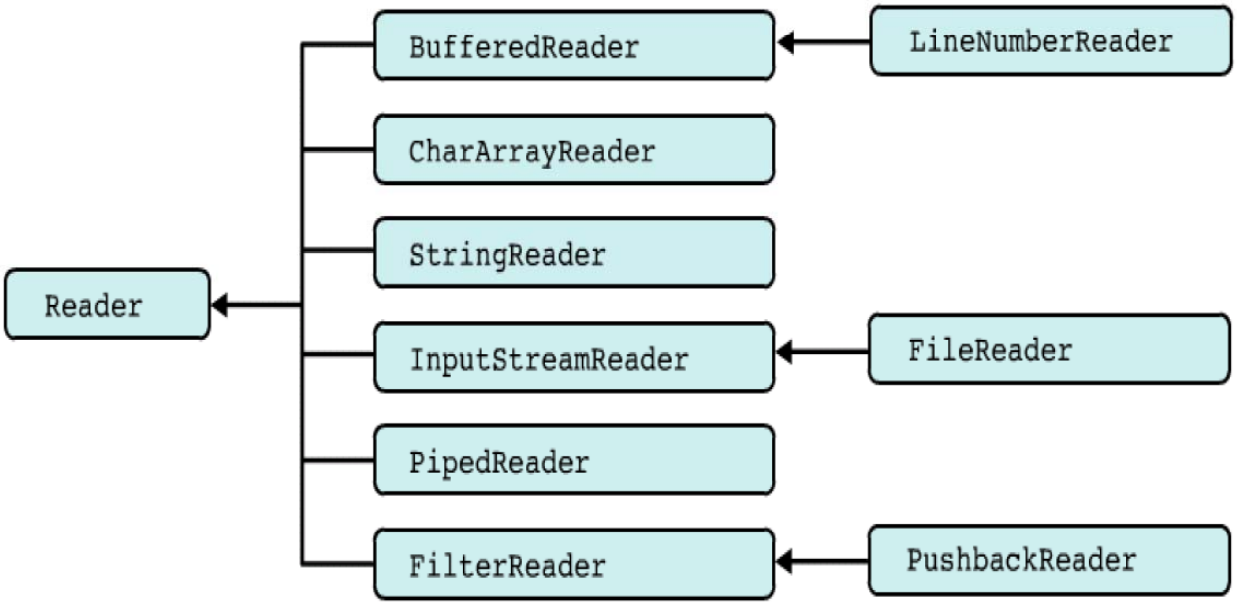
- The Writer class is the baseclass of all Writer's in the Java IO API.
- Subclasses include BufferedWriter and PrintWriter among others.

```
Writer writer = new FileWriter("c:\\data\\file-output.txt");

writer.write("Hello World Writer");
writer.close();
```

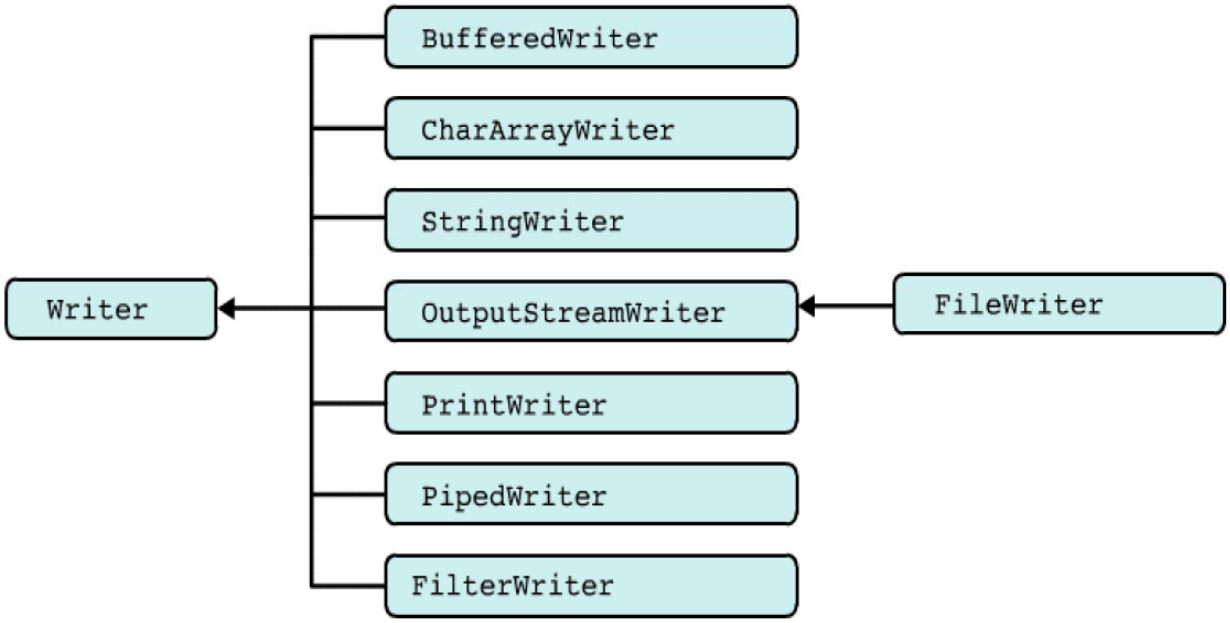
Streams (cont),..

Reader Hierarchy



Streams (cont),..

Writer Hierarchy



Data sources

Input and Output - Source and Destination

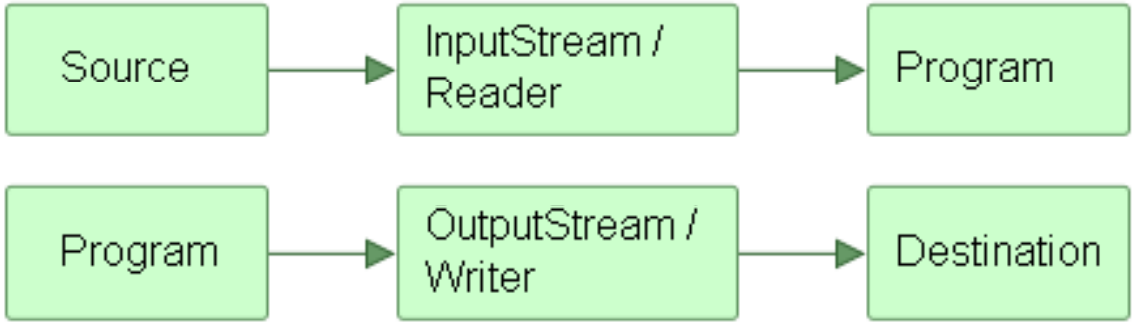
The most typical sources and destinations of data are these:

- System.in, System.out, System.error
- Files
- Pipes
- In-memory Buffers (e.g. arrays)
- Network Connections

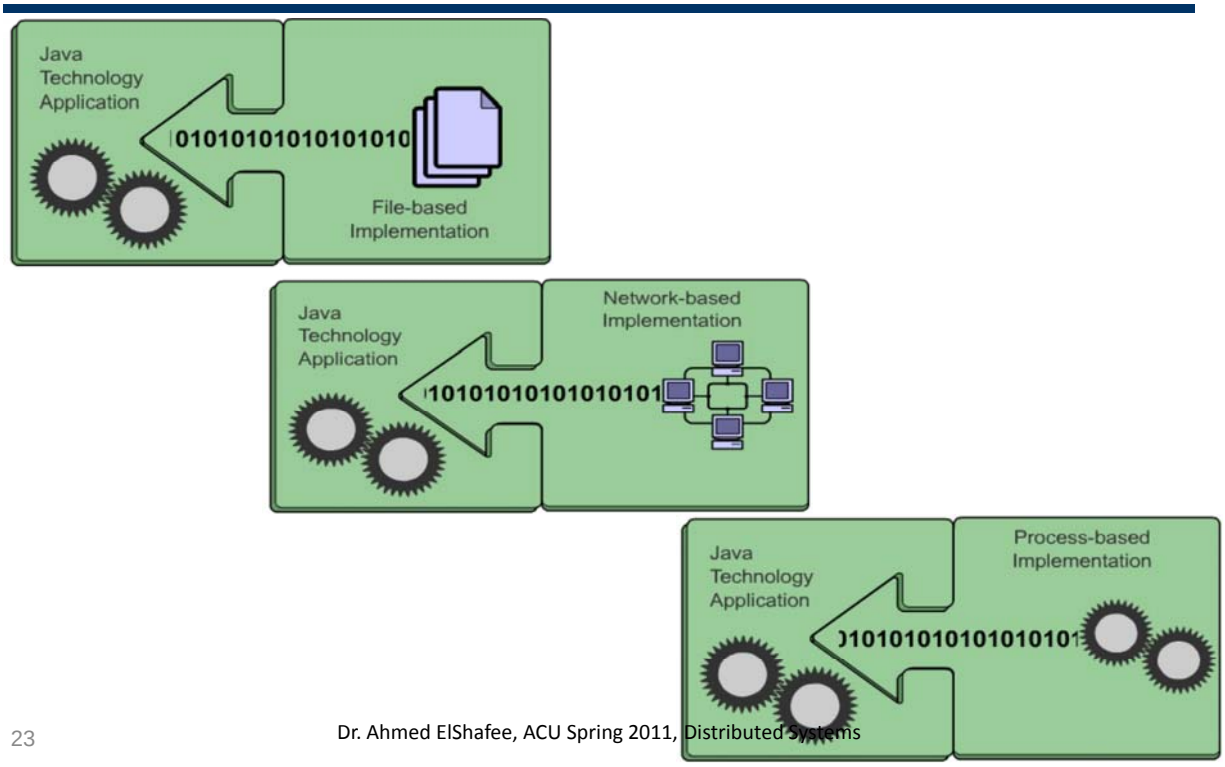
The diagram below illustrates the principle of a program reading data from a source and writing it to some destination:

Data sources (cont,..)

- A program that needs to read data from some source needs an input stream or Reader.
- A program that needs to write data to some destination needs an output stream or writer.
- This is also illustrated in the diagram below:



Data sources (cont,..)



Data sources (cont,..)

Java IO Purposes and Features

- The Java IO classes, which mostly consists of streams and readers / writers, are addressing various purposes.
- That is why there are so many different classes.

Data sources (cont,..)

- The purposes addressed are summarized below:
  - File Access
  - Network Access
  - Internal Memory Buffer Access
  - Inter-Thread Communication (Pipes)
  - Buffering
  - Filtering
  - Parsing
  - Reading and Writing Text (Readers / Writers)
  - Reading and Writing Primitive Data (long, int etc.)
  - Reading and Writing Objects

Java IO Class Overview Table

	Byte Based		Character Based	
	Input	Output	Input	Output
Basic	InputStream	OutputStream	Reader InputStreamReader	Writer OutputStreamWriter
Arrays	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
Files	FileInputStream RandomAccessFile	FileOutputStream RandomAccessFile	FileReader	FileWriter
Pipes	PipedInputStream	PipedOutputStream	PipedReader	PipedWriter
Buffering	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
Filtering	FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
Parsing	PushbackInputStream StreamTokenizer		PushbackReader LineNumberReader	
Strings			StringReader	StringWriter
Data	DataInputStream	DataOutputStream		
Data - Formatted		PrintStream		PrintWriter
Objects	ObjectInputStream	ObjectOutputStream		
Utilities	SequenceInputStream			

Data sources (cont,..)

1. Java IO: System.in, System.out, and System.error

The 3 streams System.in, System.out, and System.err are also common sources or destinations of data.

Most commonly used is probably System.out for writing output to the console from console programs.

- These 3 streams are initialized by the Java runtime when a JVM starts up, so you don't have to instantiate any streams yourself (although you can exchange them at runtime).

System.in

- System.in is an InputStream which is typically connected to keyboard input of console programs.

Data sources (cont,..)

System.out

- System.out is a PrintStream. System.out normally outputs the data you write to it to the console.

System.err

- System.err is a PrintStream. System.err works like System.out except it is normally only used to output error texts.

```
Int byte;
byte=System.in.read();
System.out.write(byte);
```

Data sources (cont,..)

```
try
{
// do somthing
}
catch (IOException e)
{
System.err.println("File opening failed:.");
System.err.println("Check File Path\"c:\\data\\IOExample.txt\"");
e.printStackTrace();
}
```


Data sources (cont,..)

2. Java IO: Files

Working with files via Java IO can be done in a few different ways:

- Reading files
  - FileInputStream
  - FileReader
- Writing files
  - FileOutputStream
  - FileWriter

Data sources (cont,..)

- Random access to files

  Random doesn't mean that you read or write from truly random places.

  It means that you can read from or write to it at the same time. This makes it possible to write only parts of an existing file, to append to it, or delete from it.

  – RandomAccessFile
- File and Directory Info Access

  access to information about a file,

  file size or the file attributes of a file or its directory.

  – File class

Data sources (cont,..)

```
InputStream input = new FileInputStream("c:\\data\\input-text.txt");

int data = input.read();
while(data != -1) {
  //do something with data...
  doSomethingWithData(data);

  data = input.read();
}
input.close();
```

Data sources (cont,..)

```
Reader reader = new FileReader("c:\\data\\input-text.txt");

int data = reader.read();
while(data != -1) {
  //do something with data...
  doSomethingWithData(data);

  data = reader.read();
}
reader.close();
```


Data sources (cont,..)

```
OutputStream output = new FileOutputStream("c:\\data\\output-text.txt");

while(moreData) {
  int data = getMoreData();
  output.write(data);
}
output.close();
```

```
Writer writer = new FileWriter("c:\\data\\output.txt");

while(moreData) {
  String data = getMoreData();
  write.write(data);
}
writer.close();
```

Data sources (cont,..)

```
RandomAccessFile file = new RandomAccessFile("c:\\data\\file.txt", "rw");

int aByte = file.read();

file.close();
```

```
RandomAccessFile file = new RandomAccessFile("c:\\data\\file.txt", "rw");

file.write("Hello World".getBytes());

file.close();
```


Data sources (cont,..)

```
File file = new File("c:\\data\\input-file.txt");
boolean fileExists = file.exists();
long length = file.length();

boolean success = file.renameTo(new File("c:\\data\\new-file.txt"));
boolean success = file.delete();
boolean isDirectory = file.isDirectory();
String[] fileNames = file.list();

File[]   files = file.listFiles();
```

Data sources (cont,..)

3. Java IO: Pipes

- Pipes in Java IO provides the ability for two threads running in the same JVM to communicate.
- As such pipes are a common source or destination of data.

Creating Pipes via Java IO

- Creating a pipe using Java IO is done via the PipedOutputStream and PipedInputStream classes.
- A PipedInputStream should be connected to a PipedOutputStream.
- The data written to the PipedOutputStream by one thread, can thus be read from the connected PipedOutputStream by another thread.

Data sources (cont,..)

Pipe Example using Java IO

- Here is a simple example of how to connect a PipedInputStream to a PipedOutputStream:

```
PipedOutputStream output = new PipedOutputStream();
PipedInputStream input = new PipedOutputStream(output);
```

You can also connect the two pipe streams using their connect() methods.

Both PipedInputStream and PipedOutputStream has a connect() method that can connect one to the other.

Data sources (cont,..)

Pipes and Threads

- Remember, when using the two connected pipe streams, pass one stream to one thread, and the other stream to another thread.
- he read() and write() calls on the streams are blocking, meaning if you try to use the same thread to both read and write, this may result in the thread deadlocking itself.

Data sources (cont,..)

```
InputStream input = new PipedInputStream(pipedOutputStream);

int data = input.read();
while(data != -1) {
  //do something with data...
  doSomethingWithData(data);

  data = input.read();
}
input.close();
```

Data sources (cont,..)

```
Reader reader = new PipedReader(pipedWriter);

int data = reader.read();
while(data != -1) {
  //do something with data...
  doSomethingWithData(data);

  data = reader.read();
}
reader.close();
```


Data sources (cont,..)

```
OutputStream output = new PipedOutputStream(pipedInputStream);

while(moreData) {
  int data = getMoreData();
  output.write(data);
}
output.close();
```

```
PipedWriter writer = new PipedWriter(pipedReader);

while(moreData()) {
  int data = getMoreData();
  writer.write(data);
}
writer.close();
```

Data sources (cont,..)

4. Java IO: Byte and Char Arrays

- Byte and char arrays are often used in Java to temporarily store data internally in an application. As such arrays are also a common source or destination of data.
- You may also prefer to load a file into an array, if you need to access the contents of that file a lot while the program is running.
- Of course you can access these arrays directly by indexing into them. But what if you have a component that is designed to read some specific data from an InputStream or Reader and not an array?

Data sources (cont,..)

Reading Arrays via InputStream or Reader

- To make such a component read from the data from an array, you will have to wrap the byte or char array in an ByteArrayInputStream or CharArrayReader.
- This way the bytes or chars available in the array can be read through the wrapping stream or reader.

```
byte[] bytes = ... //get byte array from somewhere.

InputStream input = new ByteArrayInputStream(bytes);

int data = input.read();
while(data != -1) {
  //do something with data

  data = input.read();
}
input.close();
```

Data sources (cont,..)

```
char[] chars = ... //get char array from somewhere.

Reader reader = new CharArrayReader(chars);

int data = reader.read();
while(data != -1) {
  //do something with data

  data = reader.read();
}

reader.close();
```


Data sources (cont,..)

Writing to Arrays via OutputStream or Writer

- It is also possible to write data to an ByteArrayOutputStream or CharArrayWriter.

```
ByteArrayOutputStream output = new ByteArrayOutputStream();

//write data to output stream

byte[] bytes = output.toByteArray();
```

Data sources (cont,..)

```
CharArrayWriter writer = new CharArrayWriter();

//write characters to writer.

char[] chars = writer.toCharArray();
```


Data sources (cont,..)

5. Java IO: Networking

- Java Networking will described in more detail in lecture 4.
- since network connections are a common source or destination of data, and because you use the Java IO API to communicate over a network connection
- Once a network connection is established between two processes, the processes communicates via the network connection just like they would with a file: Using an InputStream to read data, and an OutputStream to write data. In other words, Java IO is being used to pass the data to send to the Java networking API.

Data sources (cont,..)

- Basically this means that if you have code that is capable of writing something to a file, that same something could easily be written to a network connection.
- All that is required is that your component doing the writing depends on an InputStream instead of a FileInputStream.
- Since FileInputStream is a subclass of InputStream this should be no problem.

Data sources (cont,..)

Java IO: Buffered

The BufferedInputStream class provides buffering to your input streams.

Buffering can speed up IO quite a bit.

Rather than read one byte at a time from the network or disk, you read a larger block at a time.

This is typically much faster, especially for disk access and larger data amounts.

The main difference between BufferedReader and BufferedInputStream is that Reader's work on characters (text), wheres InputStream's works on raw bytes.

Data sources (cont,..)

```
InputStream input = new BufferedInputStream(
                    new FileInputStream("c:\\data\\input-file.txt"));
```

```
InputStream input = new BufferedInputStream(
                    new FileInputStream("c:\\data\\input-file.txt"),
                    8 * 1024
    );
```

```
Reader input = new BufferedReader(
                    new FileReader("c:\\data\\input-file.txt"));
```

```
    Reader input = new BufferedReader(
                        new FileReader("c:\\data\\input-file.txt"),
                    8 * 1024
    );
```


Data sources (cont,..)

Java IO: Data

The DataInputStream class enables you to read Java primitives from InputStream's instead of only bytes.

You wrap an InputStream in a DataInputStream and then you can read primitives from it.

Data sources (cont,..)

```
DataInputStream input = new DataInputStream(
                            new FileInputStream("binary.data"));

int    aByte   = input.read();
int    anInt   = input.readInt();
float  aFloat  = input.readFloat();
double aDouble = input.readDouble();
//etc.

input.close();
```


Data sources (cont,..)

- The DataOutputStream class enables you to write Java primitives to OutputStream's instead of only bytes.
- You wrap an OutputStream in a DataOutputStream and then you can write primitives

Data sources (cont,..)

```
DataOutputStream output = new DataOutputStream(
                            new FileOutputStream("binary.data"));

output.write(45);            //byte data
output.writeInt(4545);       //int data
output.writeDouble(109.123); //double data

output.close();
```


Data sources (cont,..)

Java IO: Print

The PrintStream class enables you to write formatted data to an underlying OutputStream.

For instance, writing int, long and other primtive data formatted the PrintWriter class enables you to write formatted data to an underlying Writer.

For instance, writing int, long and other primtive data formatted as text, rather than as their byte values. s text, rather than as their byte values.

Data sources (cont,..)

```
PrintStream output = new PrintStream(outputStream);

output.print(true);
output.print((int) 123);
output.print((float) 123.456);

output.printf(Locale.UK, "Text + data: %1$", 123);

output.close();
```


Data sources (cont,..)

```
PrintWriter writer = new PrintWriter(writer);

writer.print(true);
writer.print((int) 123);
writer.print((float) 123.456);

writer.printf(Locale.UK, "Text + data: %1$", 123);

writer.close();
```

Thanks,

See you next Week, isA