Lecture (03)
Steam & Serialization (PII)

Dr. Ahmed ElShafee

Agenda

- Bridging Streams
- Stream Chaining
- Serialization

Bridging Streams

To bridge the gap between the byte and character stream classes. Use,…

java.io.InputStreamReader and ***java.io.OutputStreamWriter*** classes.

Usage:

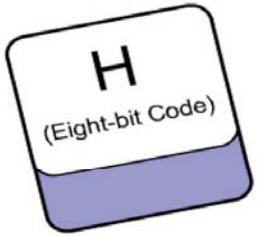
The only purpose of these classes is to convert byte data into character based data according to a specified (or the platform default) encoding.

Bridging Streams (cont,..)

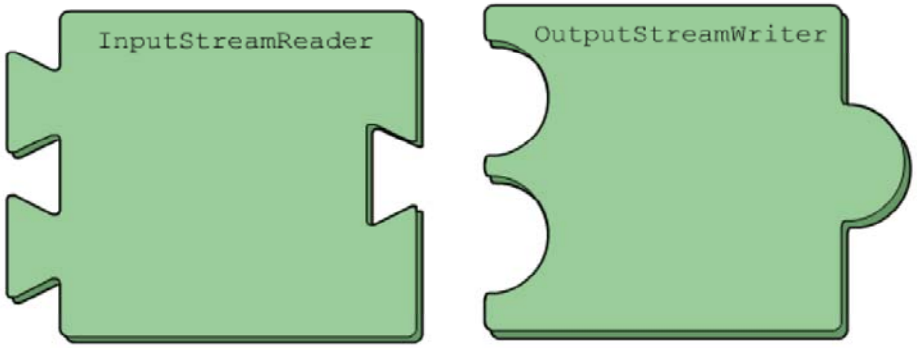
The InputStreamReader class is intended to wrap an InputStream, thereby turning the byte based input stream into a character based Reader.

The OutputStreamWriter class is intended to wrap an OutputStream, thereby turning the byte based output stream into a character based Writer.

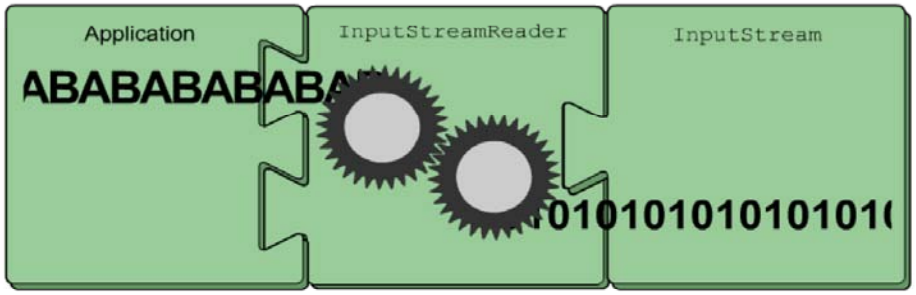
Bridging Streams (cont,..)



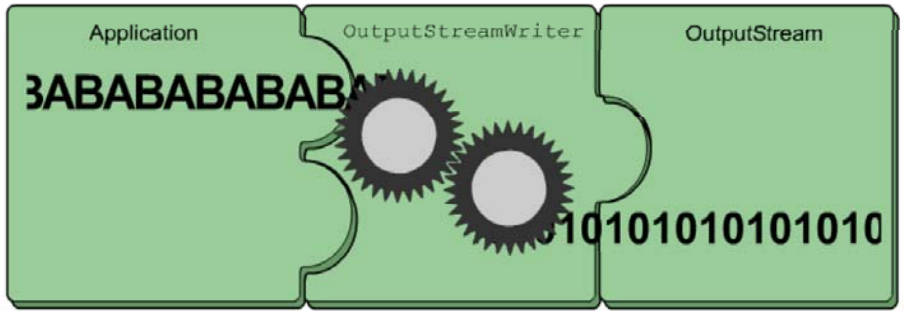
Bridging Streams (cont,..)



Bridging Streams (cont,..)



Bridging Streams (cont,..)



Bridging Streams (cont,..)

```
InputStream inputStream = new FileInputStream("c:\\data\\input.txt");
Reader      reader      = new InputStreamReader(inputStream);

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

reader.close();
```


Bridging Streams (cont,..)

```
OutputStream outputStream = new FileOutputStream("c:\\data\\output.txt");
Writer       writer       = new OutputStreamWriter(outputStream);

writer.write("Hello World");

writer.close();
```


Stream Chaining

Stream chaining is a way of connecting several stream classes together to get the data in the form required.

Each class performs a specific task on the data and forwards it to the next class in the chain.

The output produced by one component becomes the input to the next component in the chain.

Stream Chaining (cont,..)

- A Reader can be combined with an InputStream.
- If you have an InputStream and want to read characters from it, you can wrap it in an InputStreamReader.
- Pass the InputStream to the constructor of the InputStreamReader like this:

Reader reader = new InputStreamReader(inputStream);

In the constructor you can also specify what character set to use to decode the text etc.

More on that in the text on InputStreamReader.

Stream Chaining (cont,..)

- A Writer can be combined with an OutputStream just like Readers and InputStream's.
- Wrap the OutputStream in an OutputStreamWriter and all characters written to the Writer are passed on to the OutputStream.
- Here is how that looks:

```
Writer writer = new OutputStreamWriter(outputStream);
```


Stream Chaining (cont,..)

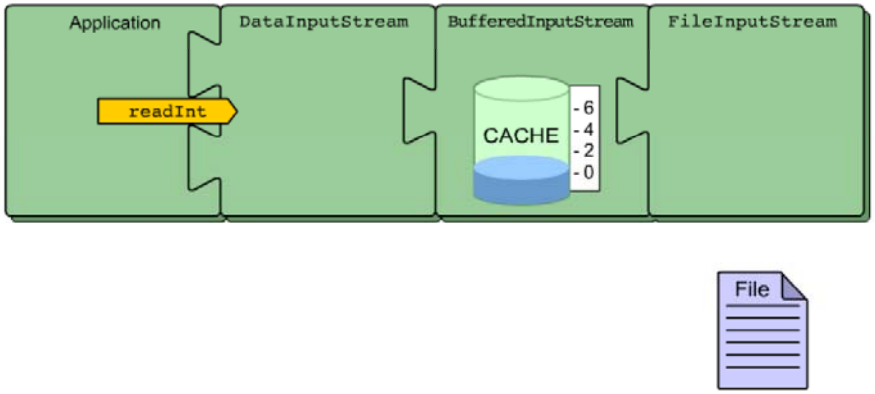
Combining Readers and Writers

- Just like with streams, Reader's and Writer's can be combined into chains to achieve more interesting IO.
- It works just like combining the Reader with InputStream's or the Writer with OutputStream's.
- For instance, you can achieve buffering by wrapping a Reader in a BufferedReader, or a Writer in a BufferedWriter.
- Here are two such examples:

```
Reader reader = new BufferedReader(new FileReader(...));
Writer writer = new BufferedWriter(new FileWriter(...));
```

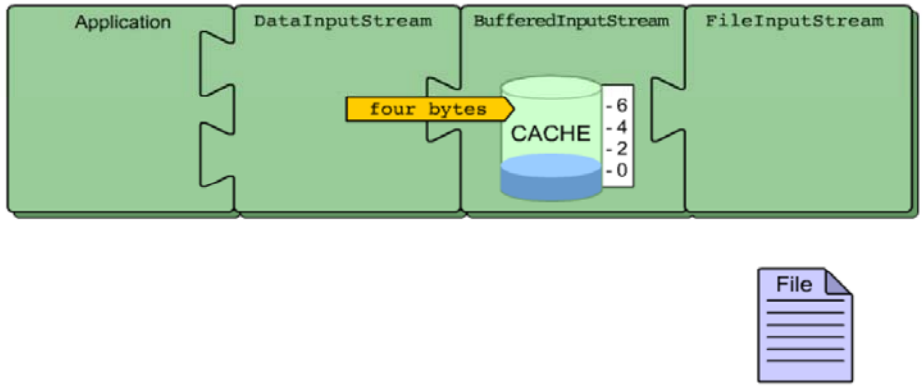

Stream Chaining (cont,..)

Example



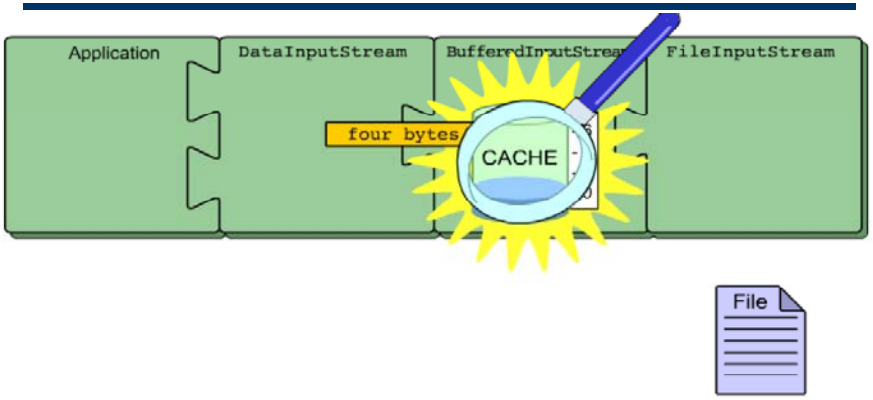
When an application invokes a method like `readInt` on a `DataInputStream` object, what happens beneath the surface?

Stream Chaining (cont,..)



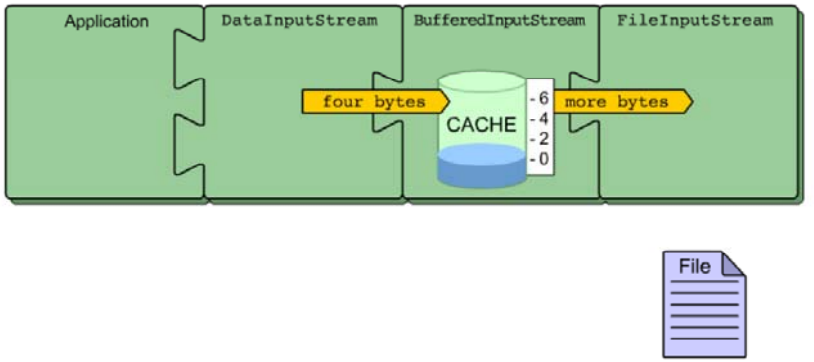
The `DataInputStream` object requests four bytes (the representation for an `int` type) from the `BufferedInputStream` object (a reference to which is contained within the `DataInputStream` object).

Stream Chaining (cont,..)



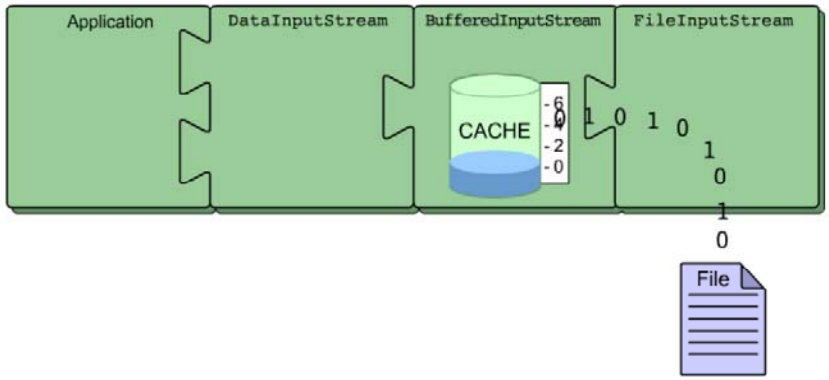
The `BufferedInputStream` object inspects its internal cache. If the cache does not contain four bytes of leftover data from a previous read, the `BufferedInputStream` object will request additional bytes from the `FileInputStream` object (a reference to which is contained within the `BufferedInputStream` object).

Stream Chaining (cont,..)



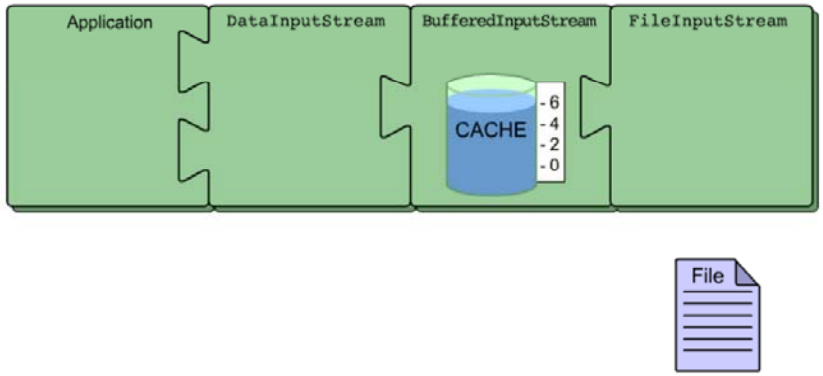
The `BufferedInputStream` object inspects its internal cache. If the cache does not contain four bytes of leftover data from a previous read, the `BufferedInputStream` object will request additional bytes from the `FileInputStream` object (a reference to which is contained within the `BufferedInputStream` object).

Stream Chaining (cont,..)



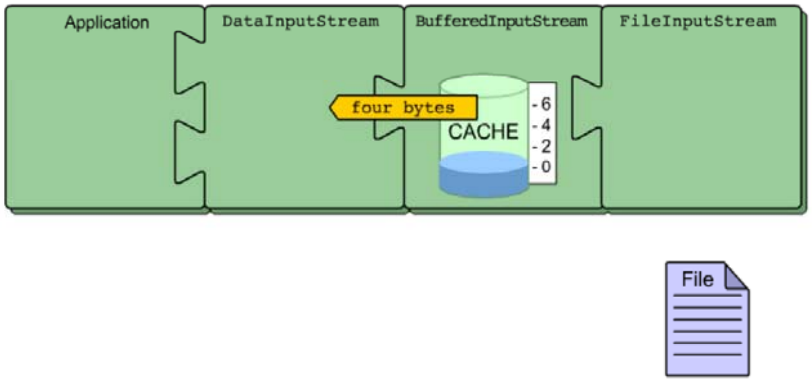
The `FileInputStream` object reads the requested numbers of bytes from the file and returns them to the `BufferedInputStream` object, which then caches the data in its internal storage buffer.

Stream Chaining (cont,..)



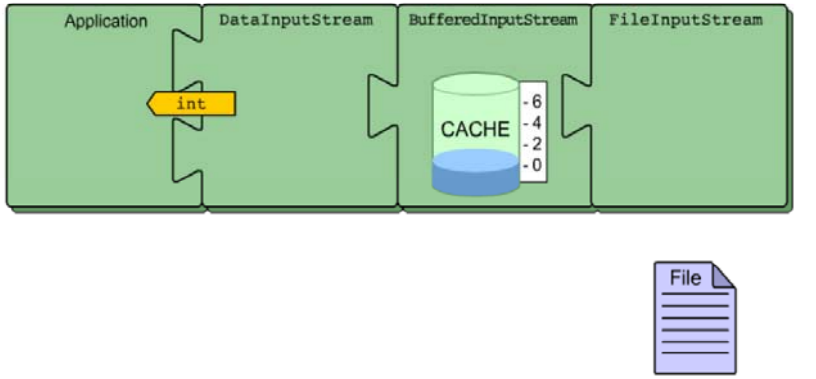
The `FileInputStream` object reads the requested numbers of bytes from the file and returns them to the `BufferedInputStream` object, which then caches the data in its internal storage buffer.

Stream Chaining (cont,..)



The `BufferedInputStream` object extracts the four bytes requested by the `DataInputStream` object from the cache and returns them to the calling method.

Stream Chaining (cont,..)

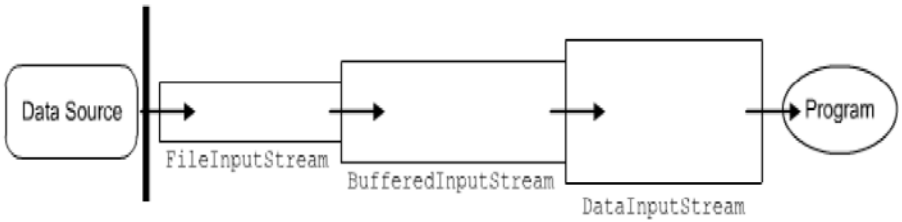


The `DataInputStream` object returns the four bytes to the application in the form of an `int` type.

Stream Chaining (cont,..)

IntputStream Chain

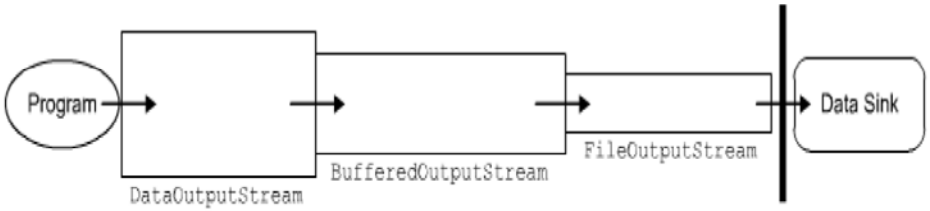
File InputStream theF i le = new FileInputStream (input.dat) ;
BufferedInputStream theBuffer = new BufferedInputStream (theFile) ;
DataInputStream theData = new DataInputStream(theBuffer) ;



Stream Chaining (cont,..)

OutputStream Chain

FileOutputStrearm theFile = new FileOutputStrearm ("output. dat") ;
BufferedOutputStrearm theBuffer = new BufferedOutputStrearm theFile) ;
DataOutputStrearm theData = new DataOutputStrearm(theBuffer) ;



Serialization

Serialization is a process of writing an object to a byte stream.

Providing a way for objects to be written as a stream of bytes and then later recreated from that stream of bytes.

Serialization (cont,..)

ObjectInputStream

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

You wrap an InputStream in a ObjectInputStream and then you can read objects from it.

```
ObjectInputStream input = new ObjectInputStream(
                            new FileInputStream("object.data"));

MyClass object = (MyClass) input.readObject();
//etc.

input.close();
```


Serialization (cont,..)

For this example to work the object you read must be an instance of MyClass, and must have been serialized into the file "object.data" via an ObjectOutputStream.

Serialization (cont,..)

ObjectOutputStream

The ObjectOutputStream class enables you to write Java objects to OutputStream's instead of only bytes.

You wrap an OutputStream in a ObjectOutputStream and then you can write objects to it.

```
ObjectOutputStream output = new ObjectOutputStream(
                            new FileOutputStream("object.data"));

MyClass object = new MyClass();

output.writeObject(object);
//etc.

output.close();
```

Serialization (cont,..)

This serialized object can now be read via an ObjectInputStream.

Serialization (cont,..)

The Serializable interface is a marker interface your classes must implement if they are to be serialized / deserialized, like shown the examples with ObjectInputStream and ObjectOutputStream.

The **Serializable** interface eliminates the drawbacks of sending objects across streams.

Each object to be sent has to implement this interface

Thanks,

See you next Week, isA