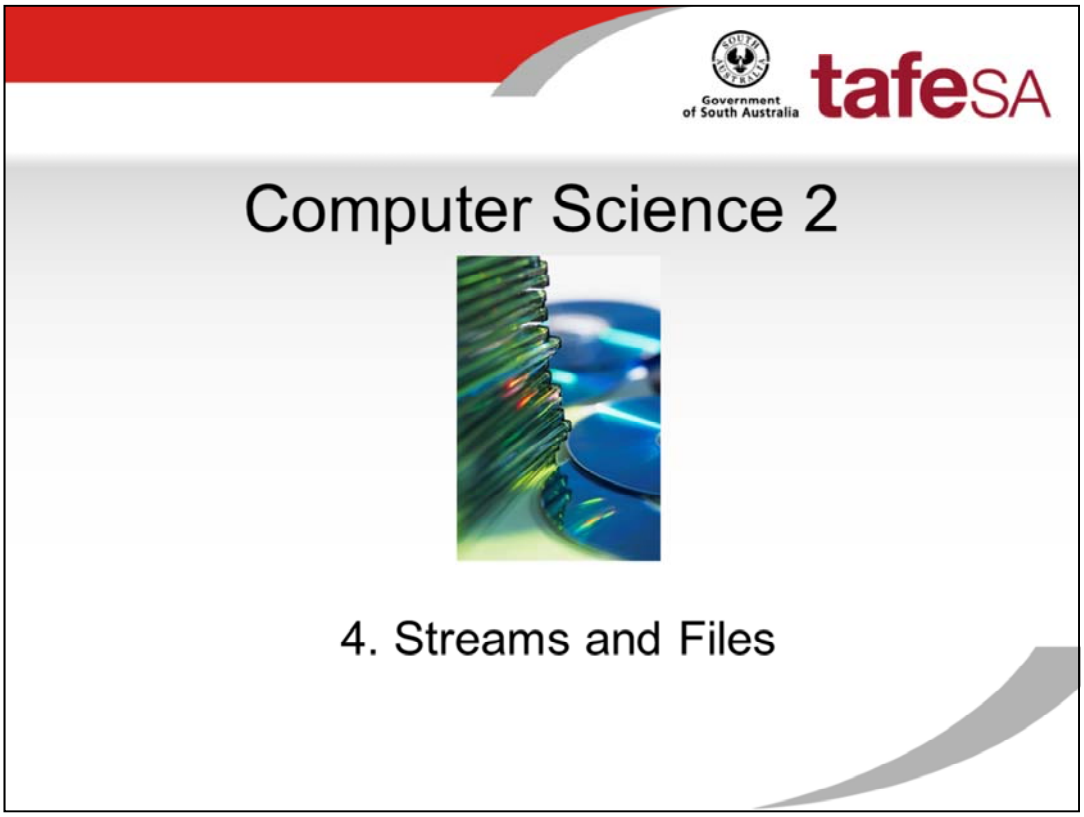
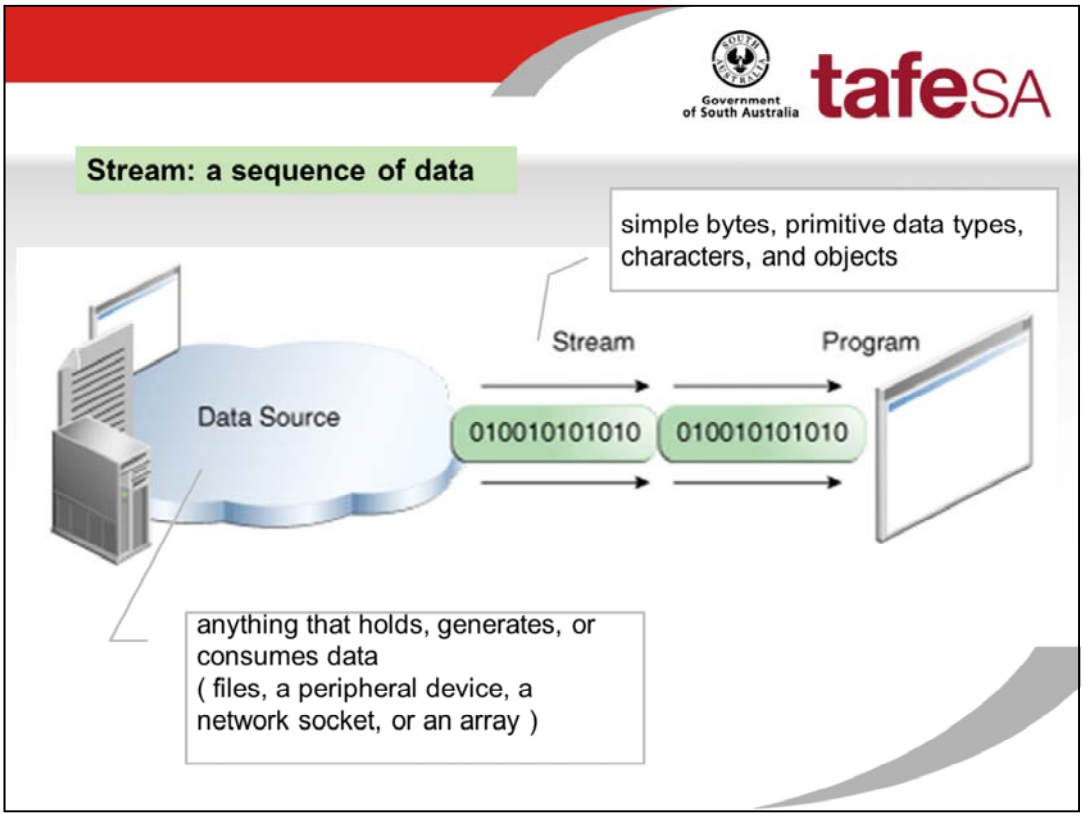


Knowing how to use files in your programs is an essential skill. Data manipulation will quite often require you to read data from a file or write data to a file. What kind of data will you be likely to need to access via a file? It could be plain text, or records of let's say a student management system, or values read from the analog to digital converter system of a microcontroller. Considering these examples, you can clearly see that data may mean a lot of things. In the case of plain text, we're quite clear that data means ASCII codes stored in files. But what would be stored for a student record (which may include personal details, grades, enrolments, financial info, etc)? And the A/D converter values? They're different again.

In this session we look at how you can manipulate all these kinds of data in the context of files.


Government of South Australia
tafeSA
Stream: a sequence of data
simple bytes, primitive data types, characters, and objects
Stream
Program
Data Source
010010101010
010010101010
anything that holds, generates, or consumes data
(files, a peripheral device, a network socket, or an array)

Streams

The connection between a program and a data source or data destination is called a **stream**.

- An **input stream** is an object that handles data flowing into a program.
- An **output stream** is an object that handles data flowing out of a program.
- Two categories: 16-bit **character streams** and 8-bit **byte streams**

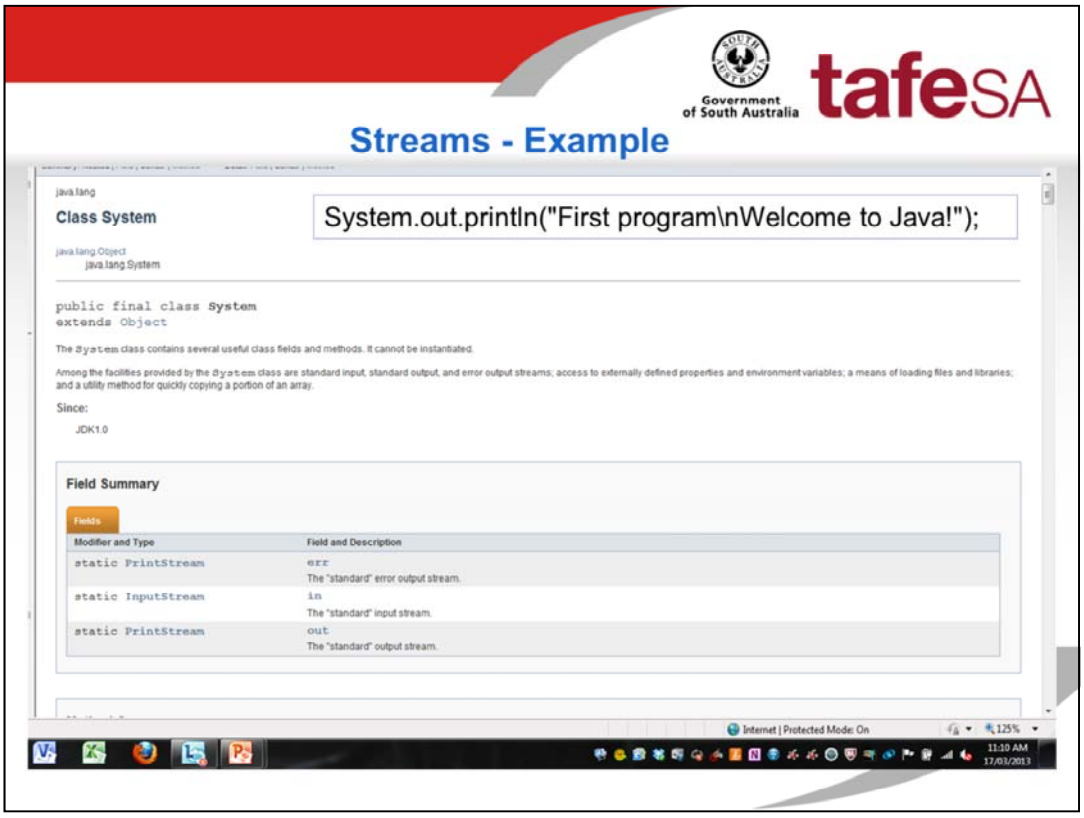
Higher level programming languages base their input/output operations on the concept of streams. You have already encountered streams – think of stdin in C or System.in in Java

A stream is nothing more than a buffer which allows the interface between your program and the input source or output destination.

Government of South Australia tafeSA

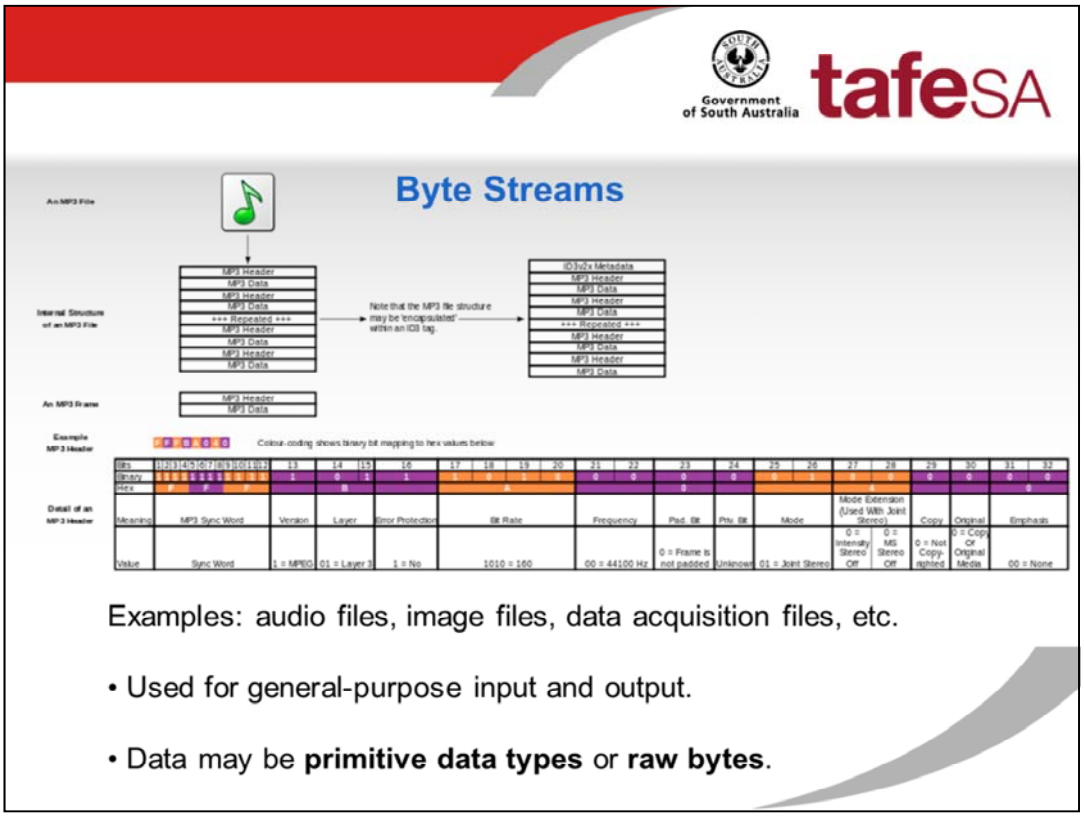
Streams

- There is a whole set of IO stream classes defined in java.io
- Any input/output of data needs to be done via a stream, hence, you need to declare objects to allow data to be 'streamed' in/out
- Java opens 3 stream objects when a program begins executing: System.in, System.out, System.err
- The 3 standard streams can be redirected if needed



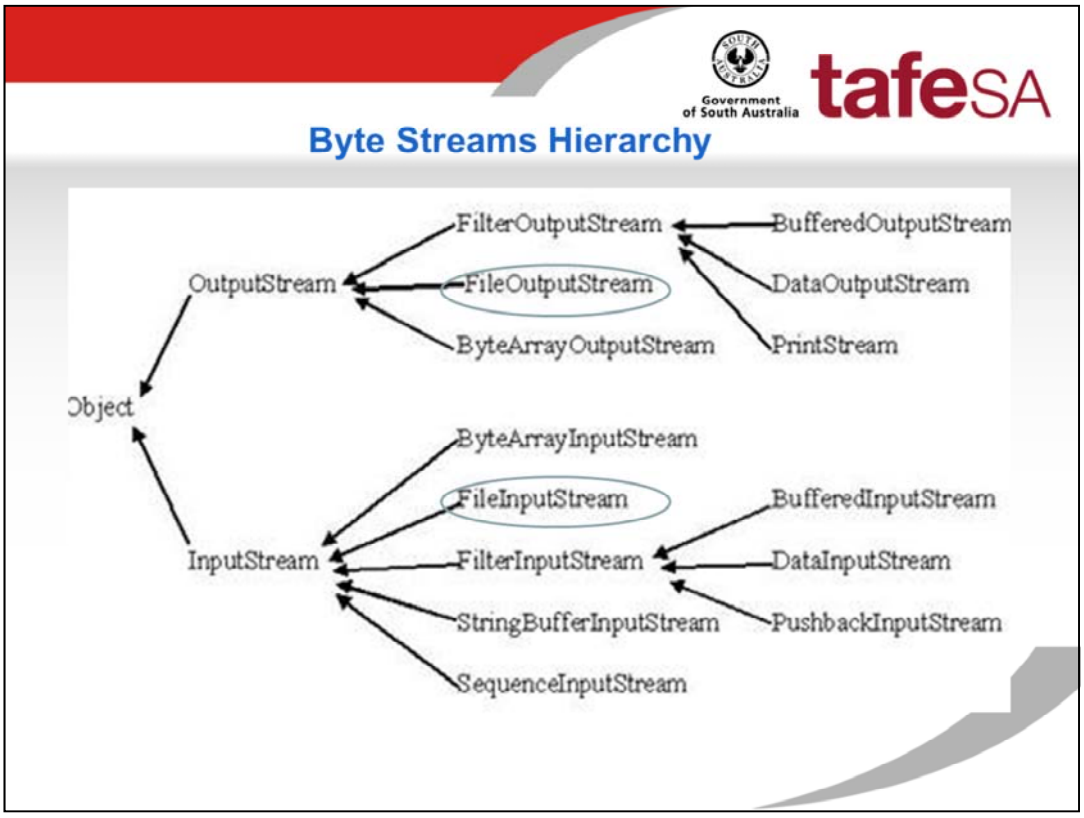
So out is a data member of class System, and it is static – I can call it from the class directly, I do not need an object.

But what type is out? It is of type PrintStream – go and check its methods, and you'll find a println, a printf, etc.



The length of the whole mp3 field is 32 bits.

You may also encounter files written by a data acquisition system whose byte length is 8 bit.



These classes are best to handle 8-bit byte streams.

Using InputStream

```
- See the example: FileLength.java
//program to read data from either the keyboard or from a file
and find out the length of data read in bytes
//to read from the keyboard, run it with java FileLength
//to read from a file run it with java FileLength nameOfFile
```

- note the usage of the command-line arguments
- inStream could be connected to a file-input or to a keyboard-input
- to close a text input, you need to use CTRL Z
- the read() method returns *int*

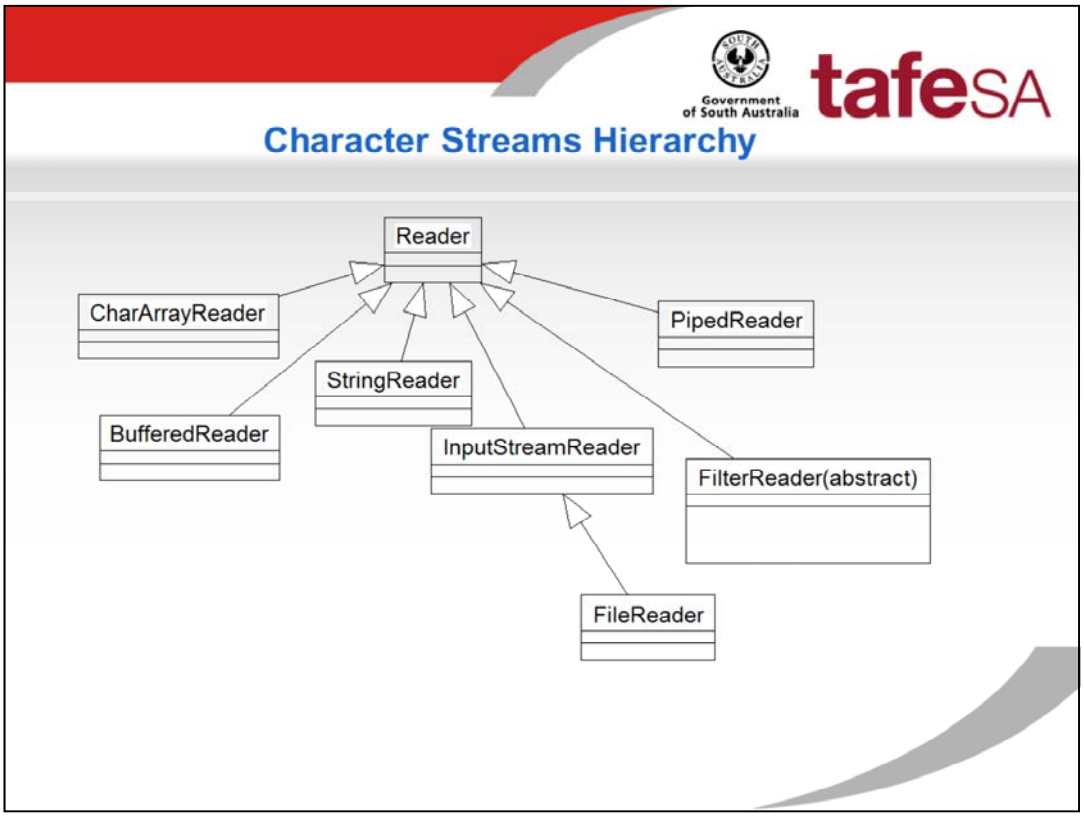
- Note the declaration of the object inStream – it's an abstract class, so you can not instantiate the class (no new InputStream() allowed)
- String[] args = array of strings
- See InputStreams methods, like read() and close()
- Run the application with keyboard entry (end entry with CTRL Z) and then run it for the file music.mp3
- Try running the application with an invalid file name – what happens?

Character Streams

- Specialised to handle character data
- Based around the abstract classes: Reader and Writer.
- The Java platform stores character values using Unicode conventions.

The FileLength.java program reads characters (letters) from the keyboard, but if it reads from a file, it can also read raw data.

If you only have character data to read, the Character stream is the type of stream to use.



Reader is an abstract class.

Classes of interest: FileReader

Writer is similar. It has OutputStreamWriter and FileWriter.

Most programs should use readers and writers for text. (they can handle Unicode). The byte streams use 8-bit bytes representation.

Government of South Australia tafeSA

Using Readers

Fill in the gaps in the file *CharStreams.java*, to use *InputStreamReader* and *FileReader* to read text from keyboard or a text file.

Writing to a Text File - FileWriter

java.io

Class FileWriter

```
java.lang.Object
  └ java.io.Writer
      └ java.io.OutputStreamWriter
          └ java.io.FileWriter
```

All Implemented Interfaces:
Closeable, Flushable, Appendable

```
public class FileWriter
extends OutputStreamWriter
```

Convenience class for writing character files. The constructors of this class assume that the default character encoding and the default byte-buffer size are acceptable. To specify these values yourself, construct an Out

Whether or not a file is available or may be created depends upon the underlying platform. Some platforms, in particular, allow a file to be opened for writing by only one `FileWriter` (or other file-writing object) at a tim open.

`FileWriter` is meant for writing streams of characters. For writing streams of raw bytes, consider using a `FileOutputStream`.

Since:
JDK1.1

See Also:
`OutputStreamWriter`, `FileOutputStream`

Field Summary

Fields inherited from class java.io.Writer

`lock`

Constructor Summary

`FileWriter(File file)`
Constructs a FileWriter object given a File object.

`FileWriter(File file, boolean append)`
Constructs a FileWriter object given a File object.

`FileWriter(FileDescriptor fd)`
Constructs a FileWriter object associated with a file descriptor.

`FileWriter(String fileName)`
Constructs a FileWriter object given a file name.

`FileWriter(String fileName, boolean append)`
Constructs a FileWriter object given a file name with a boolean indicating whether or not to append the data written.

Method Summary

- You can use it either via a File object or via the name of the file.

Writing to a Text File - FileWriter

Methods

public void **close()** throws IOException

public void **flush()** throws IOException

public void **write(char c)** throws IOException

public void **write(String str)** throws IOException

public void **write(String str, int off, int len)** throws IOException

- Check out FileReader

Reading & Writing to a Text File

```
See the application CopyCharactersErr.java
```

Run CopyCharactersErr.java. Fix the errors – why do we get errors??

Government of South Australia

tafeSA

Byte Streams vs Character Streams

- **Byte Streams** process data as stream of bytes (no interpretation of the data is made).
  Based around the abstract classes: InputStream and OutputStream
  int read() – returns a 8-bit char value (a byte)

- **Character Streams** read and write 16 bit *Unicode* characters
  Based around the abstract classes: Reader and Writer.
  int read() – returns a 16-bit char value (Unicode)

File Processing

Typical file processing is:

- OPEN A FILE
- CHECK FILE OPENED
- READ/WRITE FROM/TO FILE
- CLOSE FILE

But, before you do all this, are you sure that it is a file? Does it actually exist?!

Government of South Australia tafeSA

File class

Constructors

public **File**(String name)

Example: File f = new File("readme.txt");

Notes:

1. it makes no assumption about whether the file exists or not
2. it does not open a file for reading or writing – you need to 'attach it' to a stream

```
FileInputSteam inStream = null;
if (f.exists())
        inStream = new FileInputStream(f);
```

public **File**(String pathToName, String name)
public **File**(File directory, String name)

Some of its Methods

boolean **exists**( )

Returns: true if the file/directory exists, false otherwise.

File class

Government of South Australia

tafeSA

boolean **canRead**( )
Returns: true if the file is readable, false otherwise.

boolean **canWrite**( )
Returns: true if the file is writeable, false otherwise.

boolean **isFile**( )
Returns: true if the *name* specified to the constructor is a file.

boolean **isDirectory**( )
Returns: true if the *name* specified to the constructor is a directory.

String **getAbsolutePath**( )
Returns a String with the absolute path.

String **getName**( )
Returns a String with the name.

String **getPath**( )
Returns a String with the path.

String **getParent**( )
Returns a String with the parent directory.

long **length**( )
Returns the length of the file in bytes.

long **lastModified**( )
Returns platform-dependent representation of the last modified time.

String[] **list**( )
Returns an array of strings representing the content of a directory.

Government of South Australia tafeSA

File Processing

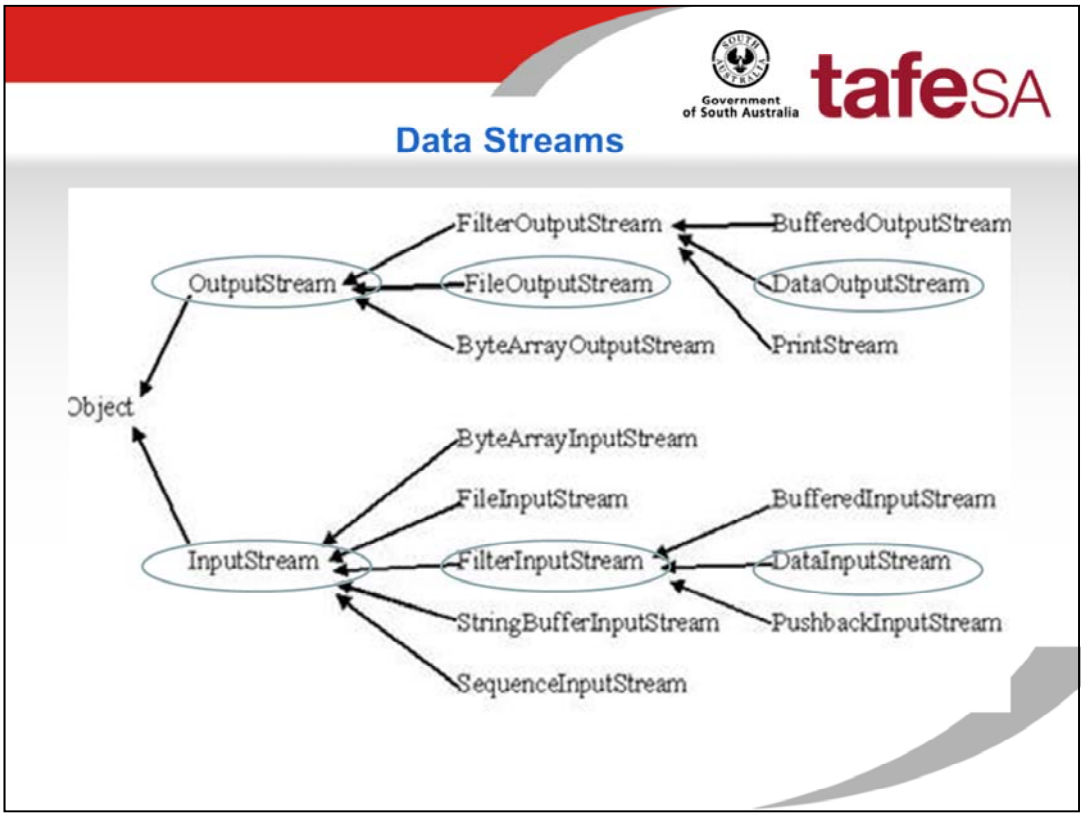
- OPEN A FILE

```
File fileOrDir = new File( name );
```

- CHECK FILE OPENED

```
if ( fileOrDir.exists() )
```

- READ/WRITE FROM/TO FILE
  - use stream objects
    (decide which one based on the type of data stored in the file)
  - When using stream objects you may not need to create a new File object. If the file does not exist, an exception will occur (for reading) or the file will be created (for writing).
- CLOSE FILE



A data output stream lets an application write primitive Java data types to an output stream in a portable way. An application can then use a data input stream to read the data back in.

Government of South Australia tafeSA

Data Streams

- allows direct input and output of the primitive data types
- based on DataInputStream and DataOutputStream
- these classes can only be created as wrappers for an existing byte stream

```
out = new DataOutputStream(new FileOutputStream(dataFile));
```

See the example DataStreams.java

Wrapper – class based on another class in the hierarchy

Government of South Australia tafeSA

Object Streams

See the example AccountRecordSerializable in Deitel

- allows you to save and retrieve whole objects to/from a file
- based on ObjectInputStream and ObjectOutputStream
- converting an in-memory object into a stream of bytes is called ***serialisation***
- the class whose objects are to be saved needs to implement the Serializable interface

Page 754