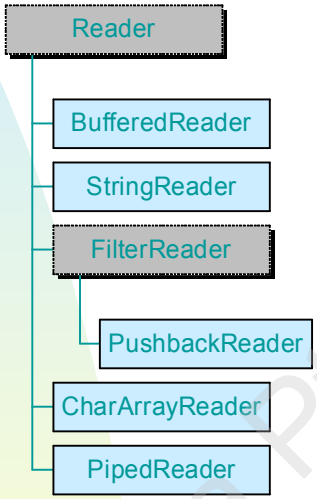
The Stream Zoo

- Conduits for sending and receiving data
  - regardless of characteristics
    - file
    - network
    - memory
    - another process
    - etc.
  - all encapsulated in the same way
    - …as far as possible
    - regularity makes life easier for us poor developers!

The Stream Zoo

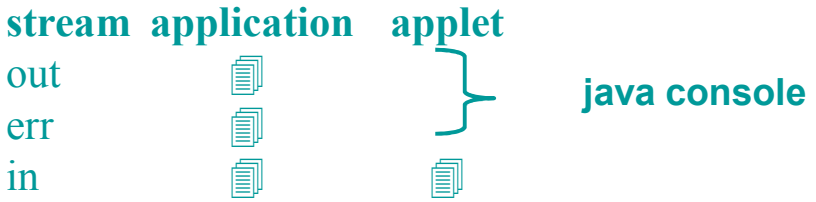
- java.io package
  - *many* classes
    - note use of abstract classes to impose order



- Reader/Writer and Input/OuputStream split
  - Reader/Writer for textual data
  - Input/OutputStream for binary data

The Stream Zoo

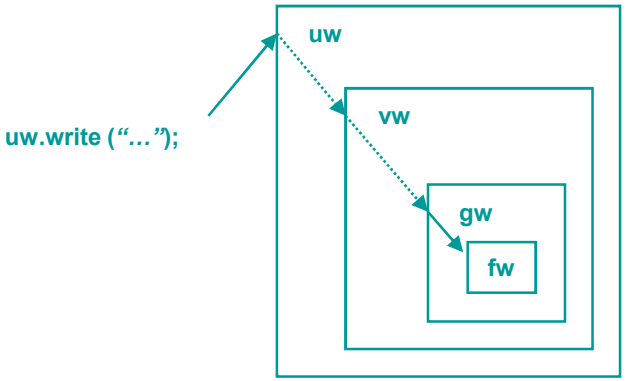
- System.{err, out} are instances of PrintStream
  - textual representation of byte stream
    - *"All characters printed by a PrintStream are converted into bytes using the platform's default character encoding. The PrintWriter class should be used in situations that require writing characters rather than bytes."*
    - print (*primitive type*)
    - println (*primitive type*)
  - *"PrintStream has been superceded in Java 1.1 with PrintWriter. The constructors of this class have been deprecated but the class itself has not, because it is still used by the System.out and System.err standard output streams."*
- Standard streams
  - System.{out, err, in}



The Stream Zoo

- "Mix and match" often necessary
  - *"The use of layered objects to dynamically and transparently add responsibilities to individual objects is referred to as the* decorator *pattern."*
  - e.g. print from a memory stream (String)

```
FileWriter fw = new FileWriter ("test.dat");
GrepWriter gw = new GrepWriter (fw, "E");
VowelWriter vw = new VowelWriter (gw);
UppercaseWriter uw = new UppercaseWriter (vw);
uw.write ("...");
```

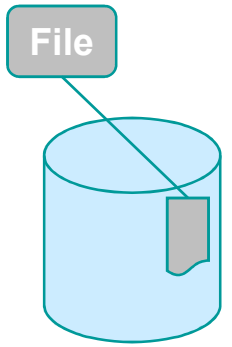


The Stream Zoo

```
import java.io.*;

public class UppercaseWriter extends FilterWriter
  {
  public UppercaseWriter (Writer w) { super (w); }

  public void write (int b) throws IOException
    { out.write ((int) Character.toUpperCase ((char) b)); }

  public void write (char [] b) throws IOException
    {
    for (int i = 0; i < b.length; i ++)
      write (b [i]);
    }

  public void write (char [] b, int off, int len) throws IOException
    {
    for (int i = 0; i < len; i ++)
      write (b [i + off]);
    }

  public void write (String s, int off, int len) throws IOException
    {
    char [] cbuf = new char [len];
    s.getChars (off, len, cbuf, 0);
    write (cbuf);
    }

  public void write (String s) throws IOException
    { write (s, 0, s.length ()); }

  public static void main (String [] args) throws IOException
    {
    UppercaseWriter uw = new UppercaseWriter (new PrintWriter (System.out));

    uw.write ("hello world"); uw.flush ();
    }
  }
```

The Stream Zoo

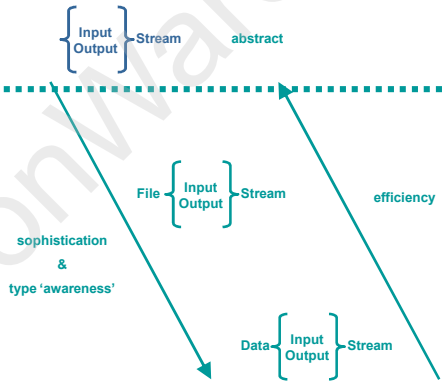
- File
  - doesn't refer to an actual file...
  - represents either name of file or the set of files within a directory
  - pathSeparator attribute
  - create, rename, delete, etc.
  - length, modified, isFile, etc.
  - URL style pathnames usually used:
    - /C|/Bob/stuff/other%20stuff/thing
      - the only "100% pure" way of doing things
    - can accept platform specific versions as well:
      - Portitia HD:Bob:Java:Stuff
      - C:\Bob\Java\Stuff



The Stream Zoo

```
import java.io.*;

public class DirList
  {
  public static void main (String args [])
    {
    File path = new File (".");
    String [] list =
      (args.length == 0) ? path.list () : path.list (new DirFilter (args [0]));
    for(int i = 0; i < list.length; i ++)
      System.out.println (list [i]);
     }
  }

class DirFilter implements FilenameFilter
  {
  private String afn;

  public DirFilter (String afn)
    { this.afn = afn; }

  public boolean accept (File dir, String name)
    {
    String f = new File (name).getName ();
    return (f.indexOf (afn) != -1);
    }
  }
```

The Stream Zoo



- {Input, Output}Stream
  - base abstract 'pattern' classes
- {File, Data}{Input, Output}Stream
  - File*
    - byte stream only—no structuring
    - *"structure is in the eye of the beholder"*
  - Data*
    - understands primitive data structures
      - {read, write}Char, {read, write}Double, etc.
      - {read, write}UTF for strings
        - UTF-8 variable-width encoding for transmission
    - big-endian format defined for types
      - *a-la* SPARC (surprise!)

The Stream Zoo

```
import java.io.*;

// copy a stream of character-based data from one file to another
// usage: java TextFileCopy input output
public class TextFileCopy
  {
  private static final int BUFSIZE = 1024; // gratuitous optimization

  public static void main (String [] args)
    {
    BufferedReader r = null;
    BufferedWriter w = null;
    try
      {
      r = new BufferedReader (new FileReader (args [0]), BUFSIZE);
      w = new BufferedWriter (new FileWriter (args [1]), BUFSIZE);
      char [] buffer = new char [BUFSIZE];
      for ( ; ; )
        {
        int bytes_read = r.read (buffer);
        if (bytes_read == -1)
          break;
        w.write (buffer, 0, bytes_read);
        }
      }
    catch (IOException e)
      {
      e.printStackTrace (System.err);
      }
    finally
      {
      try { r.close (); } catch (Exception e) { e.printStackTrace (System.err); }
      try { w.close (); } catch (Exception e) { e.printStackTrace (System.err); }
      }
    }
  }
```

```
while ((bytes_read = r.read (buffer)) != -1)
  w.write (buffer, 0, bytes_read);
```

The Stream Zoo

- RandomAccessFile
  - no ‘append’ mode for streams; use this instead:

```
RandomAccessFile st = new RandomAccessFile ("myfile", "rw");
st.seek (st.length ());
```

- StreamTokenizer (*java.io*)
  - break a stream into component tokens
    - ☞ good for tokenising Java source
  - c.f. StringTokenizer (*java.util*)

```
StringTokeniser st = new StringTokeniser (str, " \t\n\r");
```

- ByteArrayInputStream
  - read from an array of bytes
- Pipe Streams for communication between threads

The Stream Zoo

- (applications) communication between processes
  - note: loss of portability

```
import java.io.*;

public class SysIO
  {
  public static void main (String [] args)
    {
    PrintWriter out = new PrintWriter (System.out, true),
                err = new PrintWriter (System.err, true);
    try
      {
      String [] cmdArray = { "/bin/ls", "-la" };

      Process dir = Runtime.getRuntime ().exec (cmdArray);
      String s;
      BufferedReader r =
        new BufferedReader (new InputStreamReader (dir.getInputStream ()));
      while ((s = r.readLine ()) != null)
        out.println (s);

      int ev;
      if ((ev = dir.exitValue ()) != 0)
                err.println ("NON-Zero exit value: " + ev);
      }
    catch (Exception e)
      {
      e.printStackTrace (System.err);
      }
    }
  }
```

The Stream Zoo

- Serialization

  "…imagine a GUI interface builder tool that allows a programmer to build a GUI using a point-and-click interface. Such a tool could create a tree of AWT components within an Applet panel, and then serialize the applet, including all of the GUI components that it contains. When deserialized, the applet would have a complete GUI, despite the fact that the applet's class file does not contain any code to create the GUI."

The Stream Zoo

```
// Serialize today's date to a file
FileOutputStream f = new FileOutputStream ("tmp");
ObjectOutputStream s = new ObjectOutputStream (f);
s.writeObject ("Today");
s.writeObject (new Date());
s.flush();
s.close ();

// Deserialize a string and date from a file
FileInputStream in = new FileInputStream ("tmp");
ObjectInputStream s = new ObjectInputStream (in);
String today = (String)s.readObject ();
Date date = (Date)s.readObject ()
```

The Stream Zoo

- save the state of an object for later restoration
  - c.f. database ‘blobs’
- tedious/error-prone by hand
- key support feature for Java RMI and Java Beans
- object should implement the *Serializable* marker interface

```
package java.io;
public interface Serializable {}
```

  - also exists Externalizable interface (extends Serializable)
    - if more control over processing needed
- fields marked with the ‘transient’ modifier keyword are not serialized
  - transient Date now = new Date ();

The Stream Zoo

- *"when an object is stored, all of the objects that are reachable from that object are stored as well"*
  - references a problem
    - references replaced with ID
    - ID checked on write
- mechanism provides for evolution
  - rules
    - establish compatibility between classes in system and class in received stream
    - fairly straightforward "Receiver Makes Right" system
  - mechanisms like Stream Unique Identifier
    - versions of a class must declare the SUID that they are compatible with

The Stream Zoo

- writeObject:
  - *"writes the class of the object, the class signature, and the values of all non-transient and non-static fields."*
- for readObject:
  - *"The class of the object, the signature of the class, and the values of the non-transient and non-static fields of the class and all of its supertypes are read"*
- readObject reads *against* an Object
  - created 'internally' to readObject
  - so class needs to be accessible to reader
    - java.io.InvalidClassException

The Stream Zoo

```
// build a tree
ThreeNode root = new TreeNode ("Root");
root.addLeftChild (new TreeNode ("Left Child"));
  TreeNode rightChild = new TreeNode ("Right Child");
  rightChild. addLeftChild (new TreeNode ("Right Left Child"));
  rightChild. addRightChild (new TreeNode ("Right Right Child"));
root.addRightChild (rightChild);

// serialize the complete tree
FileOutputStream fOut = new FileOutputStream ("SerializedTree");
ObjectOutputStream oOut = new ObjectOutputStream (fOut);
oOut.writeObject (root);
oOut.flush ();
oOut.close ();

… (in another class, perhaps)

// deserialise the tree from the file
FileInputStream fIn = new FileInputStream ("SerializedTree");
ObjectInputStream oIn = new ObjectInputStream (fIn);
TreeNode newRoot = (TreeNode) oIn.readObject ();
```

The Stream Zoo

- Version 1.1 of Java introduced:
  - new IOExceptions
    - finer granularity of reporting
  - use of {input, Output}Stream classes deprecated
    - replaced with Writers
    - more efficient; locale-savvy

The Stream Zoo

Character-stream class	Description	Byte-stream class
Reader	Abstract class for character-input streams	InputStream
BufferedReader	Buffers input, parses lines	BufferedInputStream
LineNumberReader	Keeps track of line numbers	LineNumberInputStream
CharArrayReader	Reads from a character array	ByteArrayInputStream
InputStreamReader	Translates a byte stream into a character stream	(none)
FileReader	Translates bytes from a file into a character stream	FileInputStream
FilterReader	Abstract class for filtered character input	FilterInputStream
PushbackReader	Allows characters to be pushed back	PushbackInputStream
PipedReader	Reads from a PipedWriter	PipedInputStream
StringReader	Reads from a String	StringBufferInputStream
Writer	Abstract class for character-output streams	OutputStream
BufferedWriter	Buffers output, uses platform's line separator	BufferedOutputStream
CharArrayWriter	Writes to a character array	ByteArrayOutputStream
FilterWriter	Abstract class for filtered character output	FilterOutputStream
OutputStreamWriter	Translates a character stream into a byte stream	(none)
FileWriter	Translates a character stream into a byte file	FileOutputStream
PrintWriter	Prints values and objects to a Writer	PrintStream
PipedWriter	Writes to a PipedReader	PipedOutputStream
StringWriter	Writes to a String	(none)