CS5001 Object Oriented Modelling Design and Programming Lecture 7-10

I/O and Networking

Jon Lewis & Ozgur Akgun
School of Computer Science
University of St Andrews

What we'll cover

- java.io.*
  - Java input output libraries
  - Streams, Files, Readers and Writers
- Mentions of
  - Serialisation for Java objects
  - Properties
  - Concurrency
- java.net.* - Networking packages
  - TCP, UDP focu on connection
  - Client/Server , Multi-Threaded Server
  - Synchronous/Asynchronous
  - Multicast
- HTTP refresher

I/O

File class

- A File can be either a file or a directory
  - File file = new File("test.txt");
  - File dir = new File("/cs/home/sza23/examples/");
- Does not create a new entity on filesystem automatically
  - Can be done with helper methods
- Can be used to create temporary files
- Useful methods
  - isFile (), isDirectory (), canRead (), canWrite (), createNewFile(), mkdir (), delete ()
- Doesn't allow you to read/write the file
  - must acquire a *Stream*

Streams

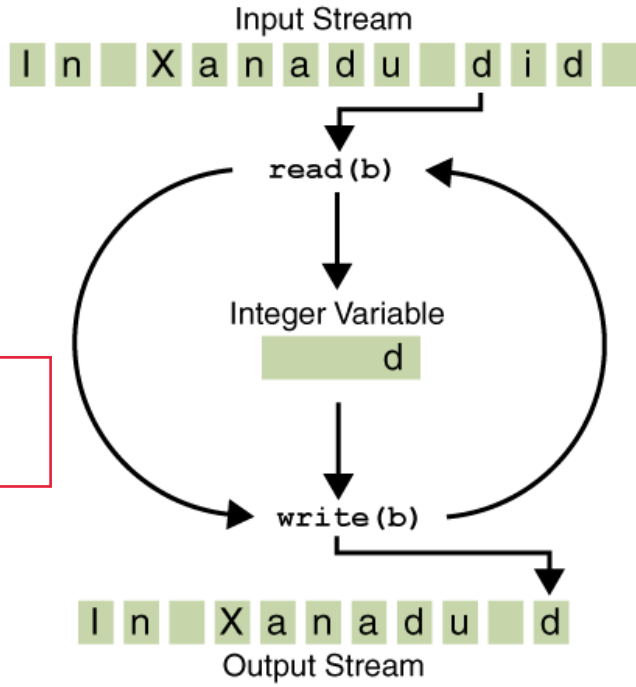
- IO in Java uses the concept of *Streams*, and subclass from
  - InputStream - Allows reading from an external source
  - OutputStream - Allows writing to an external source

- Can get hold of InputSteam/OutputStream associated with instances of class
  - File, Socket etc
  - Or can create via subclasses (FileInputStream, FileOutputSteam)
- Streams provide low level access – you read and write raw bytes
  - Useful if you want to e.g. copy file content from one stream to another regardless of whether they are text or binary (files)

Example

```
public class CopyBytes {
     public static void main(String[] args) {
          FileInputStream in = null;
          FileOutputStream out = null;
          try {
               in = new FileInputStream("xanadu.txt");
               out = new FileOutputStream("outagain.txt");
               int c; // int representing byte

               while ((c = in.read()) != -1) {
                    out.write(c);
               }
               in.close();
               out.close();
          } catch(IOException ioe){
               System.err.println(ioe.getMessage());
          }
     }
}
```



Adapted from
http://docs.oracle.com/javase/tutorial/essential/io/bytestreams.html

Some Stream Subclasses

- InputStream
  - FileInputStream
  - ObjectInputStream
  - PipedInputStream
- OutputStream
  - FileOutputStream
  - ObjectOutputStream
  - PipedOutputStream

Character streams

- Java stores text using unicode
- Character streams convert to the local character set

- Two helper classes allow mapping from Streams to CharacterStreams
  - InputStreamReader (with e.g. subclass FileReader)
  - OutputStreamWriter (with e.g. subclass FileWriter)

Example

```
public class CopyCharacters {
      public static void main(String[] args) {
            FileReader fr = null;
            FileWriter fw = null;
            try {
                  fr = new FileReader("xanadu.txt");
                  fw = new FileWriter("characteroutput.txt");

                  int c; // int representing unicode char
                  while ((c = fr.read()) != -1) {
                        fw.write(c);
                  }
                  fr.close();
                  fw.close();
            } catch (IOException ioe) {
                  System.err.println(ioe.getMessage());
            }
      }
}
```

Adapted from
http://docs.oracle.com/javase/tutorial/essential/io/charstreams.html

Reader/Writer subclasses

- Reader
  - BufferedReader
  - CharArrayReader
  - InputStreamReader
  - PipedReader
- Writer
  - BufferedWriter
  - CharArrayWriter
  - OuputStreamWriter
  - PipedWriter
  - PrintWriter

流类关注的是文件内容，而 File 类关注的是文件在磁盘上的存储。

File 不属于文件流，只能代表一个文件或是目录的路径名而已。

提示：

如果处理文件或者目录名，就应该使用 File 对象，而不是字符串。例如， File 类的 equals 方法知道一些文件系统对大小写是敏感的，目录尾的“ / ”字符无关紧要。

FileInputStream 类或者 FileReader 类的构造函数有多个，其中典型的两个分别为：一个使用 File 对象为参数；而另一个使用表示路径的 String 对象作为参数；自己以前一直觉得直接用了 String 指定路径就可以了，一直不明白为什么很多人都先构造一个 File 对象，现在终于明白了，“ 如果处理文件或者目录名，就应该使用 File 对象，而不是字符串。 ”！

Buffered Reader/Writer & Print Writer

```
public class CopyLines {
     public static void main(String[] args) {
          BufferedReader br = null;
          BufferedWriter bw = null;
          PrintWriter pw = null;
          try {
               br = new BufferedReader(new FileReader("xanadu.txt"));
               bw = new BufferedWriter(new FileWriter("charoutput.txt"));
               pw = new PrintWriter(bw);
               String line;
               while ((line = br.readLine()) != null) {
                    pw.println(line);
               }
               br.close();
               bw.close();
          } catch (IOException ioe) {
               System.err.println(ioe.getMessage());
          }
     }
}
```

read different object

Finally closing

- Did not have enough room on previous slides to put:

```
} finally {
                if (br != null) {
                    br.close();
                }
                if (pw != null) {
                    pw.close();
                }
            }
```

无论怎样都会关

- But this is good idea

Try With Resources

```
public class CopyLinesTryWithReources {
  public static void main(String[] args) {
    try (   BufferedReader br = new BufferedReader(new FileReader(args[0]));
            BufferedWriter bw = new BufferedWriter(new FileWriter(args[1]));
            PrintWriter pw = new PrintWriter(bw);
                                                                          ){
        String line;
        while ((line = br.readLine()) != null) {
          pw.println(line);
        }
      } catch (IOException ioe) {
        System.err.println(ioe.getMessage());
      } catch (ArrayIndexOutOfBoundsException aob){
        System.err.println("You must pass 2 args to program: <infile> <outfile>");
      }
  }
}
```

只要不是null, 记得close

I/O要用好多try catch

- Since Java 7
  - Any object that implements java.lang.AutoCloseable (includes all objects which implement java.io.Closeable), can be used as a resource.

Token access

- Up till now you've had to deal with bytes/characters/streams
- Scanner class reads input via tokens
  - default is to use whitespace delimiter - can be changed with useDelimeter method
  - Can be set to scan Files, InputStreams (including System.in), Strings, various Readers, any class implementing Readable … (depending on constructor that is used)
  - hasNextInt, hasNextDouble, hasNextByte, hasNext, return whether there is an int, double, byte, or anything left to scan respectively
  - nextInt, nextDouble, nextByte, next, return the next int, double, byte, or token respectively

See http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html

Example Scanner

只读要读的type

```
public class ScanDoubles {
      public static void main(String[] args) {
            Scanner s = null;
            double sum = 0;
            try {
                  s = new Scanner(new File("numbers.txt"));
                  while (s.hasNext()) {
                        if (s.hasNextDouble()) {
                              double d = s.nextDouble ();
                              sum = sum + d;
                        } else {
                              s.next();
                        }
                  }
                  System.out.println("The sum of all those doubles is " + sum);
                  s.close();
            } catch (FileNotFoundException fnf) {
                  System.err.println(fnf.getMessage());
            } catch (IOException ioe) {
                  System.err.println(ioe.getMessage());
            }
      }
}
```

java.util.Scanner是Java5的新特征，主要功能是简化文本扫描。这个类最实用的地方表现在获取控制台输入，其他的功能都很鸡肋，尽管Java API文档中列举了大量的API方法，但是都不怎么地。

And now

StreamIOExamples on studres

当通过new Scanner(System.in)创建一个Scanner，控制台会一直等待输入，直到敲回车键结束，把所输入的内容传给Scanner，作为扫描对象。如果要获取输入的内容，则只需要调用Scanner的nextLine()方法即可。

Regular expressions

- Concise and flexible way of identifying text of interest
- Similar (but more powerful) than wildcards from the command line (*, ?)
- A Pattern (the regex) is matched against a piece of text
- By default, regexes match any part of the string
- Basic regex techniques:
  - X? - Match one or no X
  - X+ - Match one or more X
  - X* - Match zero or more X

Regular expressions

- ?+* only match with the previous character by default.
- We build regular expressions by stringing options together one after the other.

Pattern	Matched
aX*Y*b	“ab” “aXb” “aXXb” “aYb” “aYYb” “aXXXYYYb”
aX?Y?b	“ab” “aXb” “aYb” “aXYb”
aX+Y+b	“aXYb” “aXXXYb” “aXXXYYb”

Regular expressions – Grouping Characters

- Two different ways of grouping:
- Character Class:
  - [aBz1] – matches a or B or z or 1
  - [1-5] – matches 1 to 5 (this is only for digits, not [1-10]!)
  - [a-zA-Z] – matches any letter
  - [A-Z][a-z]* - Matches a word which starts with a capital letter
- You can use . to match any character.

- Capturing Group:
  - (abc) – matches the string “abc”
  - Xabc+X - matches XabccX
  - X(abc)+X - matches XabcabcX

reuse

Pattern p= Pattern.comple("(.*)(my name is)\\s([||w]+")) ;

Regular expressions – Special Characters

- So far we cannot match a whole string
- There are special characters which we can use for matching beginnings and ends.

Boundary Construct	Description
^	The beginning of a line
$	The end of a line
\b	A word boundary
\B	A non-word boundary
\A	The beginning of the input
\G	The end of the previous match
\Z	The end of the input but for the final terminator, if any
\z	The end of the input

And now

Regular Expression Examples on studres

Some other things we may want to write/read to/from files (streams)

should be efficient to look up based on the key

Properties

- Hashtable which can be easily stored to a text file
  - name value pairs
  - XML
- Four methods for input output
  - load (InputStream)
  - store (OutputStream)
  - loadFromXML (InputStream)
  - storeToXML (OutputStream)
- Two methods for adding/retrieving data
  - setProperty (String key, String value)
  - String getProperty (String key)

Java中有个比较重要的类Properties（Java.util.Properties），主要用于读取Java的配置文件，各种语言都有自己所支持的配置文件，配置文件中很多变量是经常改变的，这样做也是为了方便用户，让用户能够脱离程序本身去修改相关的变量设置。像Python支持的配置文件是.ini文件，同样，它也有自己读取配置文件的类ConfigParse，方便程序员或用户通过该类的方法来修改.ini配置文件。在Java中，其配置文件常为.properties文件，格式为文本文件，文件的内容的格式是“键=值”的格式，文本注释信息可以用"#"来注释。

Properties Example

why we still need JSON?

JSON is human-readable,
serialisable is not human-readable

They do different things

```
Properties p = new Properties ();
p.setProperty ("name", "Jon Lewis");
p.setProperty ("eyes", "blue");

FileOutputStream fos =
new FileOutputStream ("aboutme.properties");

p.store (fos, "My Properties");
fos.close ();
```

```
FileInputStream fis =
new FileInputStream ("aboutme.properties");

Properties p2 = new Properties();
p2.load (fis);
String name = p2.getProperty("name");
fis.close();
```

read the whole in

Object Persistence

- Saving/loading data structures
  - Serialise/Deserialise a data structure to/from InputStream/OutputStream

- Java provides object serialisation
  - Any Java object can be (de)serialised
  - Must implement the Serializable interface
    - no methods to implement

- If we serialize an object, any object it contains will also be serialised as long as they implement the *Serializable* interface
- Some of the standard classes already implement *Serialiazable*

序列化之后就可以作为表单发送出去了

Implementing Serializable

```
public class Person implements Serializable {

	public String name;
	public int age;

	public Person(String name, int age) {
		this.name = name;
		this.age = age;
	}

	public String toString() {
		return "Person [name=" + name + ", age=" + age + "]";
	}
}
```

Object Streams

- ObjectInputStream and ObjectOutputStream can be used to serialise Java objects

- Interesting methods
  - writeObject (), readObject ()
  - writeXXX (), readXXX ()
  - Where XXX is the name of a primitive type (int, float etc)

- Objects are written to the stream using a custom protocol
  - you can replace the protocol with another one, XML based for example

Object Stream – Writer

```
int i = 1234567;
String s = "Hello World";
Person p = new Person("Jon Lewis", 32);
try{
      FileOutputStream fos = new FileOutputStream ("object.ser");
      ObjectOutputStream oos = new ObjectOutputStream (fos);
      oos.writeInt(i);
      oos.writeObject(s);
      oos.writeObject(p);
      oos.close();
} catch(IOException e){
      System.err.println(e.getMessage());
}
```

can be anything you want

Object Stream – Reader

```
try{
      FileInputStream fis = new FileInputStream ("object.ser");
      ObjectInputStream ois = new ObjectInputStream (fis);
      int i = ois.readInt();
      String s = (String) ois.readObject();
      Person p = (Person) ois.readObject();
      System.out.println("i="+i+", s="+s+", p="+p);
      ois.close();
} catch(IOException e){
      System.err.println(e.getMessage());
} catch(ClassNotFoundException cne){
      System.err.println(cne.getMessage());
}
```

Object Streams

- ObjectInputStream.readObject() returns an Object
- Must be cast into the correct type
- What happens if the input object is ***not*** the type you expect?
  - ClassCastException, so may need to catch this as well

- Can use instanceof
  - allows you to check if object is an instance of a specific class

```
Object o = ois.readObject ();
if (o instanceof String) {
  String s = (String)o;
} else if (o instanceof Person) {
  Person p = (Person) o;
}
```

And now

ObjectStreamExample on studres

Networking

Network Access

- Java supports Internet Domain Sockets
  - TCP and UDP
  - No Raw IP
- Socket classes
  - java.net.ServerSocket
  - java.net.Socket
- Once connected both ServerSocket and Socket allow you to get an InputStream and an OutputStream
- InetAddress
  - This class represents an Internet Protocol (IP) address
  - Allows resolving of hostname to addresses

The Server

```
ServerSocket ss = new ServerSocket (8888);  1
Socket conn = ss.accept ();  2

InputStreamReader isr =                                  3
  new InputStreamReader (conn.getInputStream ());
BufferedReader in = new BufferedReader (isr);  4
PrintWriter out =
  new PrintWriter (conn.getOutputStream (), true);

String line = in.readLine ();  5
out.println (line);  6

conn.close ();   7
```

Server Explanation

- Create a TCP server socket which listens on port 8888 (1)
- Wait for an incoming connection, when one is made *conn* socket serves as the endpoint (2)
- Create a Reader (3) and a Writer (4) from the streams
- Read in a line, terminated by \n from the connected client (5)
- Send that line back to the client (6)
- Close the connection (7)

The Client

Could be any valid hostname

```
Socket socket = new Socket ("localhost", 8888);  1

InputStreamReader isr =                          2
  new InputStreamReader (socket.getInputStream ());
BufferedReader in = new BufferedReader (isr);    3
PrintWriter out =
  new PrintWriter (socket.getOutputStream (), true);

out.println ("Ping");                            4
String rec = in.readLine ();                     5
socket.close ();                                 6
```

只有在有下一行的时候才能成功，ping不打一行就deadlock了

Client explanation

- Create a socket to connect to a Server on *localhost* port 8888 1
- Create a Reader 2 and a Writer 3 from the InputStream and OutputStream
- Send the text “Ping” to the server 4
- Read the response from the server 5
- Close the socket connection 6

Common Network Problems

- Only one server can be attached to each port at a time.
- Deadlock
  - When two programs are waiting for the other to send more data.
  - Make sure if you want to read a full line you send one, along with '\n'.
  - Flush streams!
- Learn to use telnet (in linux) or e.g. putty.exe (in raw telnet mode on windows) to debug server issues.
  - Do not be afraid to use debugger, or lots of print statements!

telnet能帮助debug

UDP

- UDP sockets provided by
  - DatagramSocket

- Create a UDP packet
  - DatagramPacket

Example

```
DatagramSocket s = new DatagramSocket(8888);1

byte[] buf = new byte[1500];
byte[] out = "Hello World".getBytes();

InetAddress lh = InetAddress.getLocalHost();2
DatagramPacket dpOut =
  new DatagramPacket(out, out.length, lh, 8888);3
DatagramPacket dpIn =
  new DatagramPacket(buf, buf.length);

s.send(dpOut);4
s.receive(dpIn);5
```

DatagramSocket Explanation

1. Create a DatagramSocket on port 8888, this can be used to send and receive
2. Create a DatagramPacket which is used to send data to a specific location (localhost port 8888)
3. Create a DatagramPacket which is used to receive data from a remote system
4. Send a packet using our DatagramSocket, the location to send the packet to is specified in the packet
5. Receive a packet sent to our DatagramSocket

Asynchronous Networking

- So far we have only covered synchronous communication
  - Server waits until a client wants a connection
  - Server then blocks until data can be read
- What if you want server to do useful work while nothing can be read from some stream
  - Need asynchrony

- Can check return of a socket's getInputStream.available() method before trying to read (not proper asynchrony)

- Better to use a socket's setSoTimeout(…) method to set a timeout and do useful work when a **java.net.SocketTimeoutException** is raised due to the timer expiring (can do timeout on socket.accept as well as on read and write for client connection)

Asynchronous Examples

- Examples at Examples/AsynchTCP
- TcpServer1a.java – blocking server using synchronous accept and read
- TcpServer1b.java
  - Non-blocking TCP server permitting asynchronous read using .available() method in ByteReader.readBytes method
- TcpServer1c.java
  - Non-blocking TCP server permitting asynchronous accept() and asynchronous Socket connection by using Socket.setSoTimeout(…) method on ServerSocket and connection Socket
- The corresponding TcpClient1[abc] classes are in the same directory

Java NIO useful for data-intensive application

- Adds Channel interface as abstraction over connections supporting synchronous and asynchronous communication for intensive I/O applications
- Implementing classes include:
  - FileChannel & AsynchronousFileChannel
  - SocketChannel & AsynchronousSocketChannel
  - ServerSocketChannel & AsynchronousServerSocketChannel
- Typically you read/write from/to ByteBuffer objects
- NIO2 Released with Java 7 includes java.nio.file filesystem API with support for
  - Paths
  - Change notifications
- Feel free to explore in more detail

And now

BasicClientServerExample on studres

Threading + More Networking + HTTP

Threading

Concurrent Execution

- Concurrent execution is provided by the Thread class
- Either write a class that extends (i.e. is a subclass of) the Thread class,
- Or write class that implements the Runnable interface and pass an instance of your object a ***new Thread(…)***
- ***start*** the new thread running

```
public class ThreadExample extends Thread {
  public void run () {
    /* code here to e.g. call methods ... */
  }

  pubic static void main (String argv[]) {
    ThreadExample t = new ThreadExample ();
    t.start ();
  }
}
```

- java.util.concurrent.* contains e.g. Executor class to e.g. create thread-pools

Concurrent Execution

- Concurrent execution is provided by the Thread class
- Either write a class that extends (i.e. is a subclass of) the Thread class,
- Or write class that implements the *Runnable* interface and pass an instance of your object a ***new Thread(…)***
- ***start*** the new thread running

- java.util.concurrent.* contains e.g. Executor class to e.g. create thread-pools

Concurrent Execution

- Concurrent execution is provided by the Thread class
- Either write a class that extends (i.e. is a subclass of) the Thread class,
- Or write class that implements the Runnable interface and pass an instance of your object a ***new Thread(…)***
  - The class is like any other – it can have constructor arguments and members.
- ***start*** the new thread running.

```
public class HelloRunnable implements Runnable {
    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }
}
```

More on Threads

- Pausing a thread
  - Thread.sleep (time_in_ms); // static method
- Wait for completion
  - thread.join (); // non-static method
- Interrupt
  - Indicate a thread should stop
  - thread.interrupt (); // non-static method
  - Can be handled in code by catching an InterruptedException resulting from certain methods (wait, join, sleep) + others (blocking io)

Concurrency Control

- Java has a number of ways of controlling concurrency
- Synchronized keyword
  - synchronized methods
  - synchronized statements
- A number of concurrency objects
  - java.util.concurrent.*
    - Thread pools, many other objects

Synchronized methods

```
public class SynchronizedCounter {
    private int count = 0;   // Make sure no-one touches it!
    public synchronized void increment() {
        count = count + 1;
    }

    public synchronized void decrement() {
        count = count - 1;
    }
}
```

- It is not possible for two method invocations on the same object to interleave.
  - Synchronized applies to all methods, you cannot increment and decrement at the same time!
- When a synchronized method exits it forms a happen-before relationship with any subsequent invocation

Synchronized Statements

```
public void addName(String name) {
    synchronized(this) {
        lastName = name;
        nameCount++;
    }
}
```

- Synchronizes a block of code
- Finer grain control on what is lock
- Lock is made using an object (*this* in this case)

- java.util.concurrent.* also contains a Semaphore class (low level concurrency control primitve)

Threads and Networking

Simple Concurrent Server

```
ServerSocket server = new ServerSocket (8888);
while(true){
   Socket connection = server.accept ();
   ClientHandler ch = new ClientHandler(connection);
   ch.start();
}
```

- All you have to do is
  - write a *ClientHandler* class that extends the *Thread* class
  - Write the *run*() method in the *ClientHandler* class to do the required work
  - Every client gets its own *ClientHandler* thread which handles communication with that client

And now

ClientServerExample on studres

More Sophisticated Server

- Instead of creating a new thread for each client
  - use *Executor.newFixedThreadPool*(…) to create pool of threads
  - Use *ExecutorService* class and its *execute* method to run *ClientHandler* instances in the thread pool
- Better resource utilisation through re-use of threads
- Prevents too many threads from being created

Multicast

Multicast

- So far you have seen Unicast communication
  - 1:1 communication
  - Data sent on a source socket ends up at 1 destination socket
- What if we want to send data to programs running on many different computers
  - Multicast

Multicast (2)

- Java supports multicast through the multicast datagram socket class MulticastSocket
  - UDP datagram socket with functionality for joining multicast groups
- Multicast group is defined by
  - IP address in range 224.0.0.0 - 239.255.255.255
  - Port number
- When one sends a message to a multicast group, **all** subscribing recipients to that host and port receive the message (within the time-to-live range of the packet)

Simple Multicast Example

```
// join a Multicast group and send the group salutations
msg = "Hello";
InetAddress group = InetAddress.getByName("228.5.6.7");
MulticastSocket s = new MulticastSocket(6789);
s.joinGroup(group);
DatagramPacket hi = new DatagramPacket(msg.getBytes(),
                        msg.length(), group, 6789);
s.send(hi);

// get any responses!
byte[] buf = new byte[1000];
DatagramPacket recv = new DatagramPacket(buf,
buf.length);
s.receive(recv);

// OK, I'm done talking - leave the group
s.leaveGroup(group);
```

HTTP Refresher

Hypertext Transfer Protocol

- Simple request reply protocol
- Request types
  - GET - Request a resource from the server
  - HEAD - Request meta information about a resource from a server - just the header information
  - + others
- Response
  - Status code (e.g. HTTP/1.1 200 OK)
  - Some metadata (e.g. My Java Web Server …)
  - Content length in bytes (Content-Length: 128)
  - Content type (e.g. Content-Type: text/html)
  - The content (the html page or image that was requested)

Request

- <requestType> <resource name> <protocol version><cr><lf>
- <cr><lf>

- where
  - <requestType> is GET, HEAD
  - <resource name> is the name of the resource to be required
    - filename, CGI script to execute etc
  - <protocol version> is the version of HTTP that the client is using
    - normally HTTP/1.1 from a real browser

Response

- <header>
- <cr><lf>
- <content>

- where <cr><lf> denotes <carriage return><line feed>, in Java “\r\n”

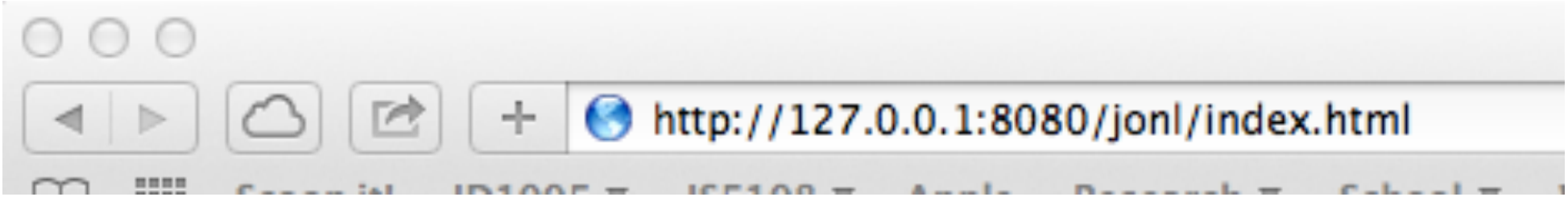
- where <header> is
  - <protocol> <responseCode> <cr><lf>
  - <responseText>

- where <response_text> is
  - Server: MySimpleServer written in Java 6 <cr><lf>
  - Content-Length: <length of content> <cr><lf>
  - Content-Type: <mime type> <cr><lf>

Response codes

- On everything working okay
  - HTTP/1.1 200 OK
- On resource not found
  - HTTP/1.1 404 Not Found
- On request type not implemented
  - HTTP/1.1 501 Not Implemented

Example *Request*



GET /jonl/index.html HTTP/1.1
Host: localhost:8080
User-Agent: Safari …
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cache-Control: max-age=0

Example *OK* Response

- Header

  HTTP/1.1 200 OK

  Server: Simple Java Http Server

  Content-Type: text/html

  Content-Length: 1279

- Body
  - The html page from the file in this case containing 1279 bytes

Example *Not Found* Response

- Header

  HTTP/1.1 404 Not Found

  Server: Simple Java Http Server

  Content-Type: text/html

  Content-Length: 128

- Body
  - response message in this case containing 128 bytes of error message as an html page