1. Introduction to the module
2. Peerwise
3. Introduction to the first section: weeks 1–4
4. Streams: basic operations and examples
5. Regular expressions
6. Serializing
7. Resource use and exceptions
SSC1 and SSC2 modules

- SSC: “Software Systems Components”
- Various topics everybody needs to do in Year 2 ⇒ compulsory
- SSC2 next semester is a different module with separate exam
- SSC1 is a 20 credit module, SSC2 another 20 credits
- One exam for all of SSC1, counting for 80%.
- Continuous assessment counts for 20% of SSC1
- Variety of continuous assessment: class test, homework with viva
Outline of SSC1 in 2011/12; who does what

- Weeks 1–4 Hayo Thielecke, also module coordinator
- Bob Hendley: Human-Computer Interaction (HCI)
- Shan He: Graphics
- Phil Smith: Teaching Assistant for all of SSC1
- Phil is pxs697@cs.bham.ac.uk
- For assessment and mitigations, contact Phil
- Welfare team decides on mitigations and then tells us
- Due diligence, attendance monitoring
10% of the continuous assessment mark will be given for participation in Peerwise, see peerwise.cs.auckland.ac.nz/docs/students/

You must write at least one multiple-choice question on a topic covered in the module so far, and answer at least three other questions. The deadline is 12 noon Friday 16 October.

Another 10% of the mark may be given as bonus points. For instance, students on the leaderboard or having achieved a certain overall score may be given bonus marks.

You need to register on the Peerwise server peerwise.cs.auckland.ac.nz/register/?bham_uk

The course ID is 5619

Your "identifier" on Peerwise is the same as your Bham student ID number.

You can choose any username and password when registering.
Making good multiple choice questions

- Distractors = wrong answers
- Distractors should be plausible
- Ideally, distractors should be based on common misconceptions
- Making and discussing questions is a more high-level skill than merely answering them
- Good distractors may generate discussion on Peerwise
1. Stream hierarchy: read input of increasing complexity
2. Mainly Java class bureaucracy; lots of documentation at oracle
3. Regular expressions: structure input according to simple patterns
4. Regular expressions in theory vs in practice
5. Parsing: structures are no longer flat, but tree-shaped
6. We need grammars for parsing
7. XML: killer app for trees that can be input and output
8. Lots of XML technologies have been and are being developed
Some terminology and its abuse

- **Scanning** means reading input according to a *regular expression*
- **Parsing** means building a tree according to a *grammar*
- Parsing is a highly developed technology with many sophisticated tools
- Terminology widely abused in “pop culture”
- Some people use “parsing” for trivial stuff that does not require a parser at all. Don’t.
- Regular expressions are not always regular in the technical sense (will be explained later).
A toy example of streams and parsing

Input string: The cat sat on the mat.
A toy example of streams and parsing

Input string: The cat sat on the mat.
We read it as a string of chars, assembled from bytes.
A toy example of streams and parsing

Input string: The cat sat on the mat.
We read it as a string of chars, assembled from bytes. A Scanner breaks it up into words: The, cat, sat, ...
A toy example of streams and parsing

Input string: The cat sat on the mat.
We read it as a string of chars, assembled from bytes.
A Scanner breaks it up into words: The, cat, sat, ...
Parsing builds a tree:
A toy example of parsing and XML

Sentence
  Subject  Verb  Object

<sentence>
  <subject>
    The cat
  </subject>
  <verb>
    sat
  </verb>
  <object>
    on the mat
  </object>
</sentence>
Streams in general

- Compare files in Unix
- Idea: “everything is a file”, including IO devices
- Stream reading is a blocking operation: may have to wait for more input
Streams in Java

- Streams can be input or output
- Read or write methods
- Class hierarchy for more and more specialized types of streams
- Two different hierarchies for byte and unicode streams
- InputStream and OutputStream are for bytes
- Reader and Writer are for unicode
- Complicated class hierarchy; APIs tend to become deprecated; move to “New IO”
- Constructors are often called to create a stream, e.g. opening a file
- Once it is constructed, we call methods on the stream
Byte vs char streams

char is 16 bits, not 8 as in C
http://download.oracle.com/javase/6/docs/api/java/io/InputStream.html
“This abstract class is the superclass of all classes representing an input stream of bytes.”
http://download.oracle.com/javase/6/docs/api/java/io/Reader.html
“Abstract class for reading character streams.”
For Reader, here is some of the class hierarchy:

Reader > BufferedReader

Reader > InputStreamReader > FileReader

Writer > BufferedWriter

Writer > OutputStreamWriter > FileWriter

Writer > PrintWriter
Constructor examples

The constructors build unicode from byte streams, e.g.

```
new InputStreamReader(System.in)
```

Files can be opened with

```
new FileWriter("filename.txt");
```
Filtering Streams and abstraction layers

We can build streams on top of other streams
The bottom level interacts with the OS
Bottom level reads raw data, e.g., bytes from a file
The higher-levels *abstract* by building types like integers, longs, objects,

*Layered architecture*: very common in Operating Systems
Abstraction is fundamental is CS in general
In Java: often pass the lower-level stream to constructor of higher-level
We call useful methods on the higher level streams, not low-level byte operations
Example: echoing a stream

```
FileReader inputStream = null;
FileWriter outputStream = null;

inputStream = new FileReader("inputfile");
outputStream = new FileWriter("characteroutput.txt");

int c;
while (true)
{
    c = inputStream.read();
    outputStream.write(c);
}
```
Echoing a stream until end of file

FileReader inputStream = null;
FileWriter outputStream = null;

inputStream = new FileReader("inputfile");
outputStream = new FileWriter("characteroutput.txt");

int c;
while ((c = inputStream.read()) != -1)
{
    outputStream.write(c);
}

Using -1 to indicate the end of the input is the usual way in C.
Network connections as Streams

- Sockets will be covered in the Networking part of SSC2
- Idea: much of the complexity of the network is hidden: just use streams
- Example: take a URL, open a network connection, and read a web page
Buffering and flushing stream

- Output gets accumulated in a buffer and sent when there is enough of it
- Closing a stream flushes the buffer
- You may need to flush an output stream
public interface Iterator<E>
boolean hasNext()
Returns true if the iteration has more elements.
E next()
Returns the next element in the iteration.
See http://download.oracle.com/javase/1,5.0/docs/api/java/util/Iterator.html
public final class Scanner
extends Object
implements Iterator<String>

Some constructors of Scanner:

Scanner(InputStream source)

Scanner(File source)

Scanner(String source)

See http://download.oracle.com/javase/1,5.0/docs/api/java/util/Scanner.html
Scanner s = null;
try {
    s = new Scanner(new BufferedReader(new FileReader("inputfile")));
    while (s.hasNext()) {
        System.out.println(s.next());
    }
} finally {
    if (s != null) { s.close(); }
}
Scanning and grouping input

- See [http://download.oracle.com/javase/1,5.0/docs/api/java/util/Scanner.html](http://download.oracle.com/javase/1,5.0/docs/api/java/util/Scanner.html)
- Quote: “A simple text scanner which can parse primitive types and strings using regular expressions.”
- But this is not really “parsing” — much more trivial.
- Not really “scanning” either, only whitespace can be configured.
- Both parsing and scanning have a precise and more powerful meaning in compiling.
String input = "1 fish 2 fish red fish blue fish";
Scanner s =
    new Scanner(input).useDelimiter("\s*fish\s*");
System.out.println(s.nextInt());
System.out.println(s.nextInt());
System.out.println(s.next());
System.out.println(s.next());
s.close();

prints the following output:
1 2 red blue
String input = "1 fish 2 fish red fish blue fish";
Scanner s =
    new Scanner(input).useDelimiter("\s*fish\s*");
System.out.println(s.nextInt());
System.out.println(s.nextInt());
System.out.println(s.next());
System.out.println(s.next());
s.close();

prints the following output:
1 2 red blue
\s*fish\s* What's that all about?
String input = "1 fish 2 fish red fish blue fish";
Scanner s =
    new Scanner(input).useDelimiter("\\s*fish\\s*");
System.out.println(s.nextInt());
System.out.println(s.nextInt());
System.out.println(s.next());
System.out.println(s.next());
s.close();

prints the following output:
1 2 red blue
\\s*fish\\s* What’s that all about? Regular Expression!
Regular Expressions

- Invented by Stephen C. Kleene (1950s)
- Used by Ken Thompson in Unix (1970s)
- Tools: grep, awk, sed, Perl, ...
- Theory: DFA, NFA will be covered in Models of Computation
- In this module: how to use reg exps
- Reg exps also relevant to parsing and XML schemas
We define what it means for a string $s$ to match a regular expression $E$:

**Constant** If $a$ is an input symbol, then $s$ matches $a$ if $s = a$.

**Composition** $s$ matches $E_1 E_2$ if $s = s_1 s_2$ where $s_1$ matches $E_1$ and $s_2$ matches $E_2$.

**Alternation** $s$ matches $E_1 \mid E_2$ if $s$ matches $E_1$ or $s$ matches $E_2$.

**Kleene star** $s$ matches $E^*$ if $s = s_1 \ldots s_n$ and each of $s_1, \ldots, s_n$ matches $E$.

**Epsilon** $s$ matches $\varepsilon$ if $s$ is the empty string.
True of false:
The definition of matching is defined in terms of matching. Such a definition is circular and therefore meaningless. It is like saying, “A table is a table is a table is a table”.
True or false?

1. aaa matches a*
Regular expressions matching examples

True or false?

1. aaa matches a*
2. The empty string matches a*

Regular expressions matching examples

True or false?

1. aaa matches a*
2. The empty string matches a*
3. aaa matches (a | b)*
True or false?

1. aaa matches a*
2. The empty string matches a*
3. aaa matches (a | b)*
4. aaa matches (a b)*
Regular expressions matching examples

True or false?

1. aaa matches a*
2. The empty string matches a*
3. aaa matches (a | b)*
4. aaa matches (a b)*
5. abab matches (a b)*
Regular expressions matching examples

True or false?

1. aaa matches $a^*$
2. The empty string matches $a^*$
3. aaa matches $(a \mid b)^*$
4. aaa matches $(a \cdot b)^*$
5. abab matches $(a \cdot b)^*$
6. ababa matches $(a \cdot b)^*$
Regular expressions matching examples

True or false?

1. aaa matches a*
2. The empty string matches a*
3. aaa matches (a | b)*
4. aaa matches (a b)*
5. abab matches (a b)*
6. ababa matches (a b)*
7. ababa matches (a b)* | a
Regular expressions matching examples

True or false?

1. aaa matches a*
2. The empty string matches a*
3. aaa matches (a | b)*
4. aaa matches (a b)*
5. abab matches (a b)*
6. ababa matches (a b)*
7. ababa matches (a b)* | a
8. ababa matches ((a b) | a)*
Regular expressions matching examples

True or false?

1. aaa matches a*  
2. The empty string matches a*  
3. aaa matches (a | b)*  
4. aaa matches (a b)*  
5. abab matches (a b)*  
6. ababa matches (a b)*  
7. ababa matches (a b)* | a  
8. ababa matches ((a b) | a)*  
9. If s matches $E^*$, then s matches $(E^*)^*$
Regular expressions matching examples

True or false?

1. aaa matches a*
2. The empty string matches a*
3. aaa matches (a | b)*
4. aaa matches (a b)*
5. abab matches (a b)*
6. ababa matches (a b)*
7. ababa matches (a b)* | a
8. ababa matches ((a b) | a)*
9. If s matches $E^*$, then s matches $(E^*)^*$
10. If s matches $E_1(E_2 | E_3)$, then s matches $(E_1E_2) | (E_1E_3)$
Regular expressions matching examples

True or false?

1. aaa matches $a^*$
2. The empty string matches $a^*$
3. aaa matches $(a | b)^*$
4. aaa matches $(a b)^*$
5. abab matches $(a b)^*$
6. ababa matches $(a b)^*$
7. ababa matches $(a b)^* | a$
8. ababa matches $((a b) | a)^*$
9. If $s$ matches $E^*$, then $s$ matches $(E^*)^*$
10. If $s$ matches $E_1(E_2 | E_3)$, then $s$ matches $(E_1 E_2) | (E_1 E_3)$
11. If $s$ matches $(E_1 | E_2)^*$, then $s$ matches $(E_1)^* | (E_2)^*$. 
More on regular expressions in Java

- Java supports a lot of contracts for reg exps
- Based on Perl syntax. Example of Perl:
  
  ```
  /\w*((%27)|(\'))((%6F)|o|(%4F))((%72)|r|(%52))/ix
  ```

"If you put a million monkeys at a million keyboards, one of them will eventually write a Java program. The rest of them will write Perl programs."
More on regular expressions in Java

- Java supports a lot of constructs for reg exps
- Based on Perl syntax. Example of Perl:
  
  ```
  /\w*((\%27)|('))(\%6F)|o|(\%4F))(\%72)|r|(\%52))/ix
  
  "If you put a million monkeys at a million keyboards, one of them will eventually write a Java program."
More on regular expressions in Java

- Java supports a lot of constructs for reg exps
- http://download.oracle.com/javase/tutorial/essential/regex/
- Based on Perl syntax. Example of Perl:
  
  `\w*\((\%27)|('')((\%6F)|o|((\%4F))((\%72)|r|((\%52)))/ix`

- “If you put a million monkeys at a million keyboards, one of them will eventually write a Java program. The rest of them will write Perl programs.”
Pattern syntax in Java: http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html
[abc] a, b, or c (simple class)
[a-zA-Z] a through z or A through Z, inclusive (range)
Backslashes are used to escape characters or to stand for classes of characters.
\[ or \( and so on stand for brackets
\\ The backslash character
\d a digit: [0-9]
\s a whitespace character
In Java code, a backslash must be escaped as a double backslash \\

What then is the meaning of `\s*fish\s*`?
Reading regular expression syntax

What then is the meaning of `\s*fish\s*`?
What then is the meaning of `\s*fish\s*`?

1. The word “fish” between two possibly empty sequences of whitespaces.

2. The word “fish” between two possibly empty sequences of whitespaces.
What then is the meaning of `\s*fish\s*`?

1. The word “fish” between two possibly empty sequences of whitespaces.

2. You need to understand basic constructs like `*` and perhaps extra notational cruft.
Regular expression theory vs regex in practice

- Regular expressions are clean and concise
- correspond to DFA and NFA— more on that in Models of Computation
- regex in practice: Java, based on Perl; lots of additional syntax
- Some extensions that are not regular, e.g. backreferences
- For more on matching and efficiency, read http://swtch.com/~rsc/regexp/regexp1.html
True or false:

1. Suppose we have an integer in memory. In principle, it is easy to write the integer to a stream. We just have to write the bytes that make up the integer. When we read back the bytes, we reconstruct the integer.
Writing objects to a stream and reading them back

True or false:

1. Suppose we have an integer in memory. In principle, it is easy to write the integer to a stream. We just have to write the bytes that make up the integer. When we read back the bytes, we reconstruct the integer.

2. Suppose we have two objects in memory, each with a reference to the other. Writing these objects is no different from writing an integer. We just write bytes from memory.
If you want to read and write objects of some class C, you need to make sure that C implements interface `Serializable`. The Java compiler and runtime do the work of serialization. Problem: An object reference is meaningless outside its context (inside the JVM). It assigns *serial numbers* to objects to keep track of references. Writing and reading objects gives an isomorphic copy. Memory addresses are different, but the structure of references is the same.
r.open();
    r.use();
...
    r.use();
r.close();

The resource could be a stream. (Or memory management in C.)
Pattern: open a resource, use it, close it

\begin{verbatim}
  r.open();
    r.use();
    ...
    r.use();
  r.close();
\end{verbatim}

The resource could be a stream. (Or memory management in C.)
Problem: exceptions could be raised and close never reached.
Pattern: close resource in finally

Idiom: finally block always reached, even with exception

```java
try {
    r.open();
    r.use();
    ...
    r.use();
}
finally {
    r.close();
}
```

May require further conditionals (if resource could be null).
Try with resources: new Java construct

New feature in Java 7, release July 2011
See http://download.oracle.com/javase/7/docs/technotes/guides/language/try-with-resources.html

```java
static String readFirstLine(String path) throws IOException {
    try (BufferedReader br =
         new BufferedReader(new FileReader(path)))
    {
        return br.readLine();
    }
}
```

The class BufferedReader, in Java SE 7 and later, implements the interface java.lang.Closeable.
Each player supplies an URL of a web page they choose. The program then connects to the URL and scans the web page for the keywords from the buzzword file. Every time the program finds an occurrence of one of those buzzwords, a score is updated and at the end, the player with the highest score wins.
BuzzBingo exercise

Each player supplies an URL of a web page they choose. The program then connects to the URL and scans the web page for the keywords from the buzzword file. Every time the program finds an occurrence of one of those buzzwords, a score is updated and at the end, the player with the highest score wins.

Sample buzzwords:
out of the box social media critically reflective proactive going forward no brainer
Each player supplies an URL of a web page they choose. The program then connects to the URL and scans the web page for the keywords from the buzzword file. Every time the program finds an occurrence of one of those buzzwords, a score is updated and at the end, the player with the highest score wins.

Sample buzzwords:
* out of the box*
* social media*
* critically reflective*
* proactive*
* going forward*
* no brainer*

Attempt it now; model answer next week.