Problem decomposition

How to solve a complex problem

Problem decomposition

- You have been given an exercise and your first thought is
- Oh my God! How on earth am I to do that?

Problem decomposition

- Don't worry this is a common enough first reaction.
- But to be come competent at programming you have to get over the shock and start breaking the problem down into simple steps.

Your example problem

Break it down from the top into component problems

- How do I build a dictionary
- How do I encode a file
- How do I decode a file

Break up your software

- Break your software up into smaller tasks corresponding to the problems you see.
- These smaller tasks can be handled either by
  - separate programs you write
  - methods you write
  - separate classes you write
  - build in classes of java
  - operating system commands

Example decompostion

How to build a dictionary

- Decide on your input data – a training file of words
- Decide on your output data – we will discuss this next slide
- Invent a mapping between input and output

Occams razor

KISS principle – Keep It Simple Stupid!

- More classically
  - entia non sunt multiplicanda praeter necessitatem,
  - entities should not be multiplied without cause
- So what is the simplest file format for a dictionary?

A simple dictionary

- Here is the start of a dictionary (trained using The Origin of the Species, and the Communist Manifesto)
- It is just a list of entries one to a line
- Most frequent first

\n
,
the
of
and
in
to
a
that
;
is
as
have
be
by
which
on
or
are
species
with

Transformation

We know what our output is to be

We now need to break the transformation of input to output into further steps

1) Split the input into tokens
2) Sort tokens by frequency of occurrence
3) Print out in order most frequent first

Split the input into tokens

- Again we apply the same rules
  - determine the inputs,
  - determine the outputs,
  - devise a transformation
- Input is a text file
- What should the output be?
- By the KISS principle lets make it a file of tokens one per line

Transform : split in tokens

- How do we do the transform
- We read tokens in one at a time and print them to the output stream one per line

How do we read a token?

- Input <- a stream
- Output <- a string

Do we write this or use a library?

We could do either but I will assume we use a library

What library/class to use

String tokenizer?

- Requires us first to read in the lines

Stream tokenizer?

- Directly uses an input stream
- This is simpler,
  - apply KISS principle

Program outline

```
class Tokens{
public static void main(String[] args)
{
  if (args.length>=1)try{
        – run the program
 else
  System.err.println("No filename provided");
}
}
```

- Program outline simply checks if we have a filename supplied
- Now lets look at the guts of the program

Initialisation

- Set up the stream tokenizer

```
FileInputStream f= new FileInputStream(new File(args[0]));
StreamTokenizer s=new StreamTokenizer(f);
s.ordinaryChar('\n');
```

We tell it not to ignore new lines since we do not want to loose them

Fields of a Stream Tokenzer

- **double nval**
  If the current token is a number, this field contains the value of that number.
- **String sval**
  If the current token is a word token, this field contains a string giving the characters of the word token.
- **static int TT_EOF**
  A constant indicating that the end of the stream has been read.
- **static int TT_EOL**
  A constant indicating that the end of the line has been read.
- **static int TT_NUMBER**
  A constant indicating that a number token has been read.
- **static int TT_WORD**
  A constant indicating that a word token has been read.
- **int ttype**
  After a call to the nextToken method, this field contains the type of the token just read.

Inner loop

- This has to loop through all tokens till we reach the end of file

```
while((i=s.nextToken())!=s.TT_EOF){
String t;
```

  - get one token as a string in t
    - *work this out for yourselves*

```
System.out.println(t);
}
```

Use makefile to patch programs together

```
#makefile line for getting tokens
tokens: Tokens.class  $(DATA)
    java Tokens $(DATA) >tokens
```

Next problem

- Finding the frequency of unique words
- Input <- stream of words one per line
- Output <- ???
  - see right panel

```
00000004temporarily
00000010reactionary
00000001progressively
00000001flax
00000001thicknesses
00000013flat
00000003mimicked
00000002meeting
00000001persevering
00000004birthplace
•
```

Freq Output format

Why have I chosen this

It lists

1. the number of times a word occurs
2. the word itself

The frequency is in fixed format

Watch this.....

Use of OS utilities

- I can use the sort command to help me here

Frequency counting loop

- Let br be a buffered reader
- let h be a hash table

```
while((s=br.readLine())!=null){
 if (h.get(s)==null){int [] count={1};h.put(s,count);}
 else{(h.get(s))[0]++;}
}
```

This is similar to what I showed you for a visitor class in the last lecture

Formatted printing

- Let p be a printstream
- Let freq be the frequency of occurrence of a word
- Let w be the word
  - then

  p.format("%1$08d%2$s\n",freq,w);

  will print the frequency to 8 digits followed by the word

  Look up the PrintStream class for this but I think you have already had formatted io