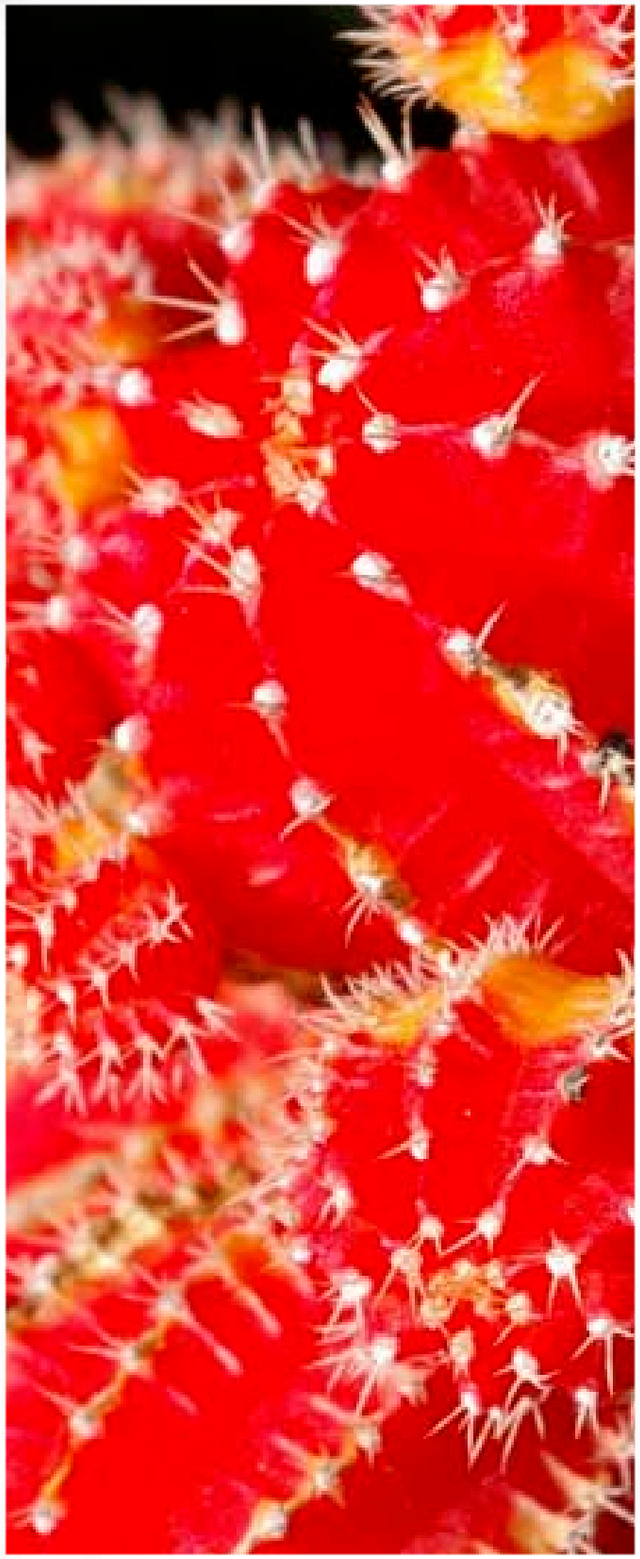
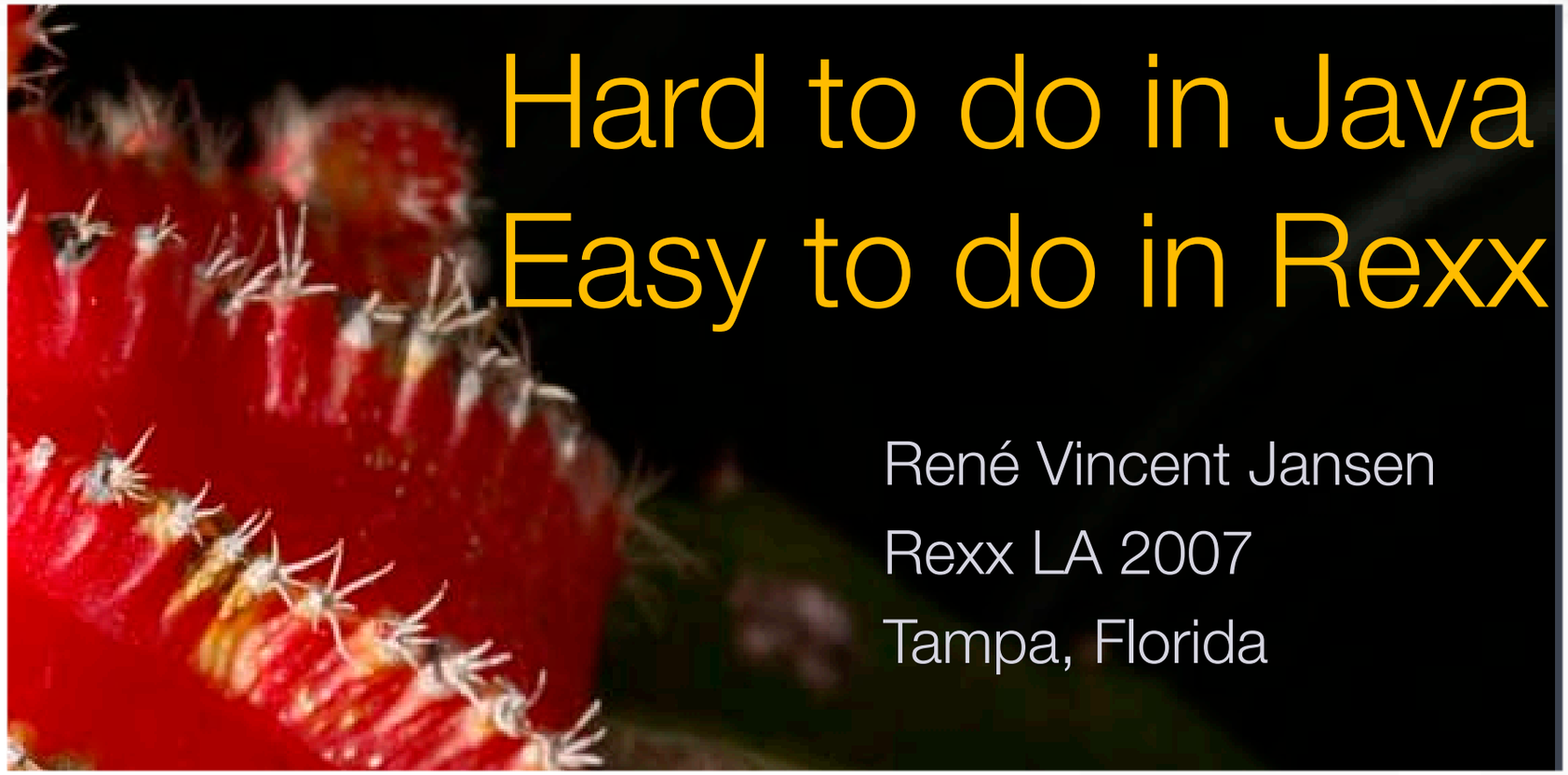


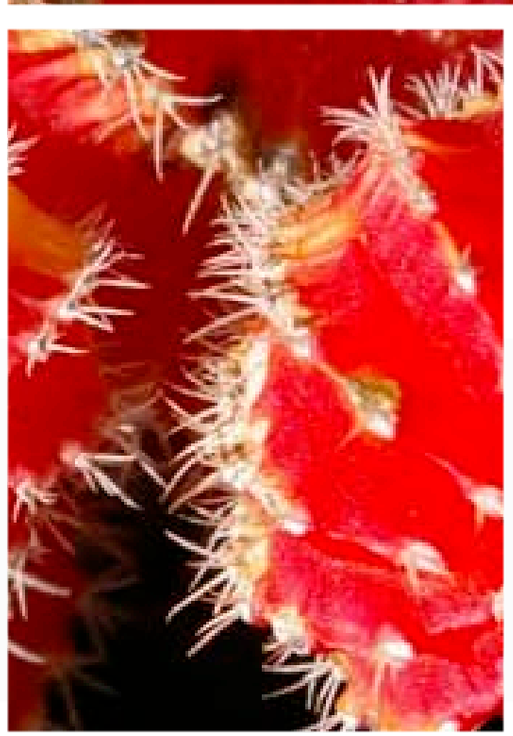


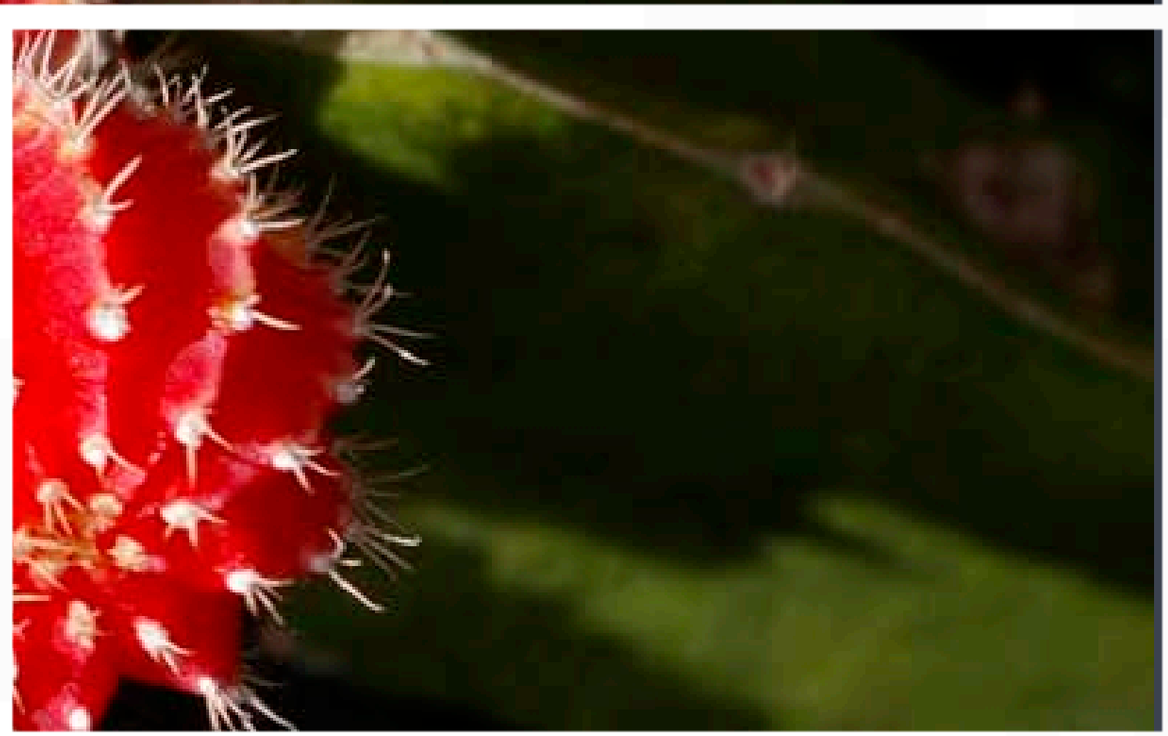
Hard to do in Java
Easy to do in Rexx

René Vincent Jansen

Rexx LA 2007

Tampa, Florida

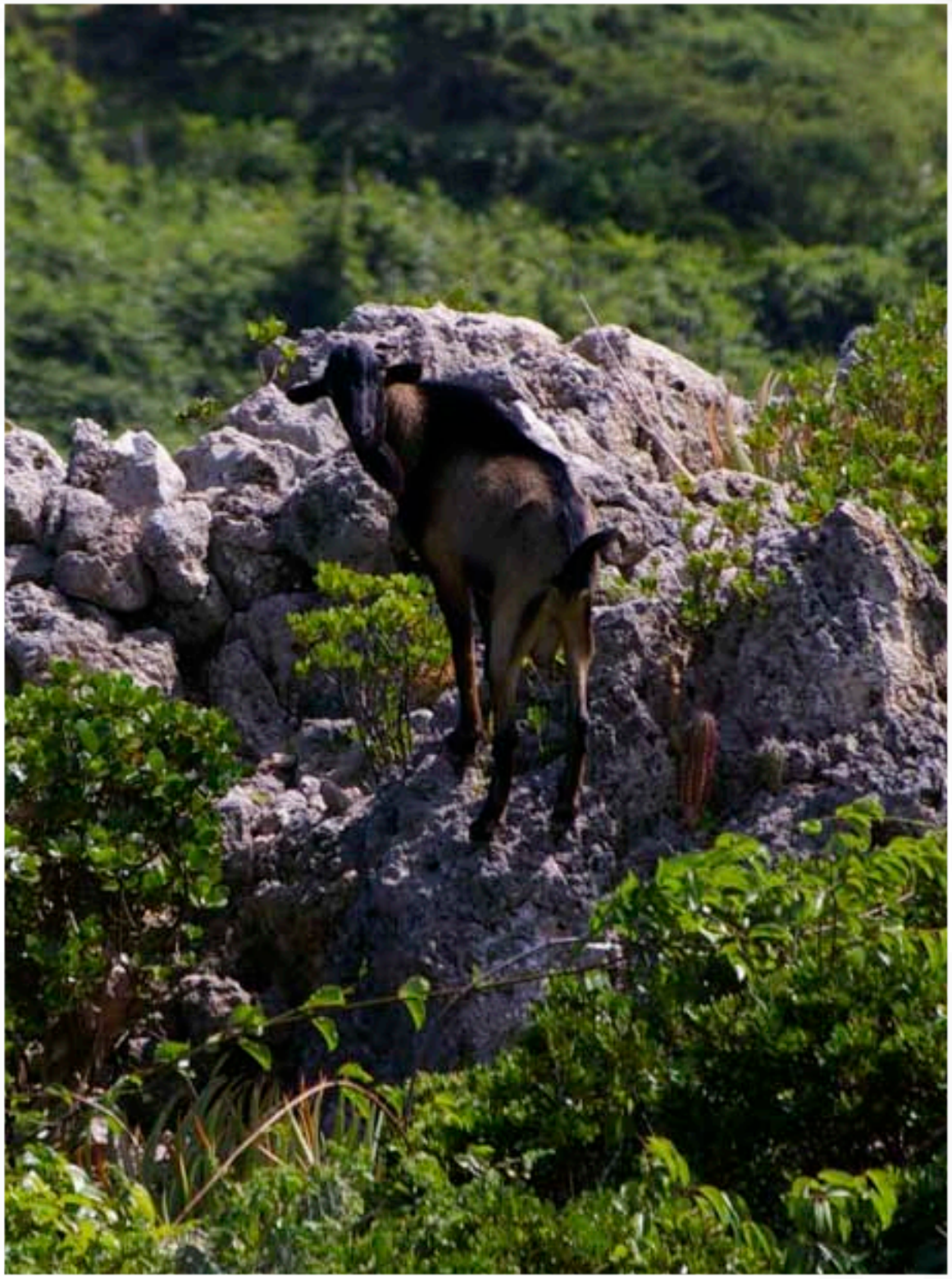




There is no doubt that Java is the COBOL of this day and age, and its influence is all pervasive in enterprise software.

For some tasks however, its statically checked type system and its inflexible approach to the run time treatment of classes defeats the purpose. In this presentation I would like to show that for some tasks the more dynamic approach of Object Rexx gets the job done more easily and efficiently.

Most examples are from an existing system where the Java way of thinking has bitten us and we are pursueing an alternative approach combining NetRexx (Java) and ooRexx, linked by BSF for Rexx.



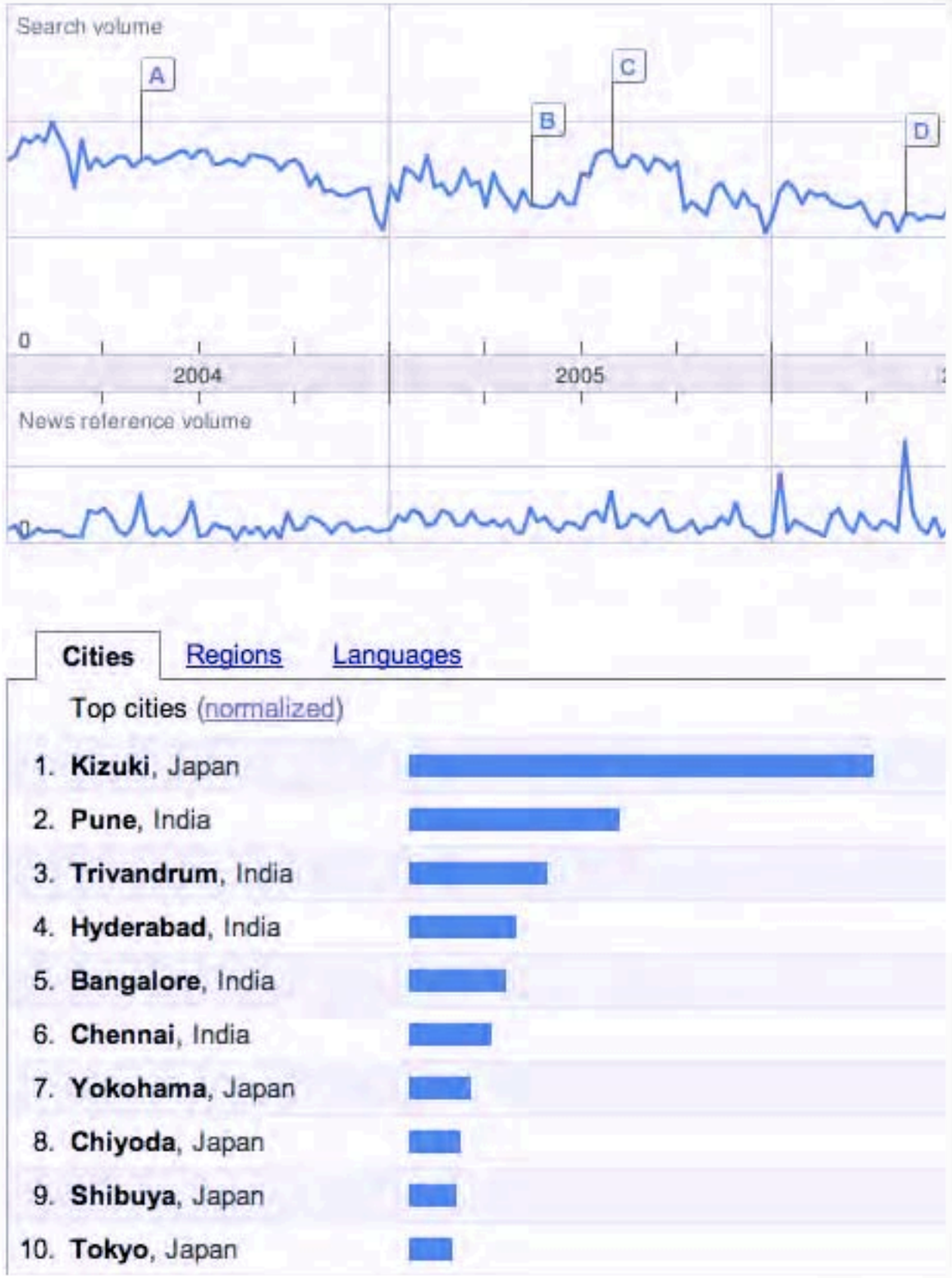
Interestingly, Rexx is at the same time an accepted piece of *proven technology* and an advanced oo-technology that is at the vanguard of more research oriented companies.

If not the proof of this, a nice illustration is what results are for queries for different technologies in *Google Trends.*

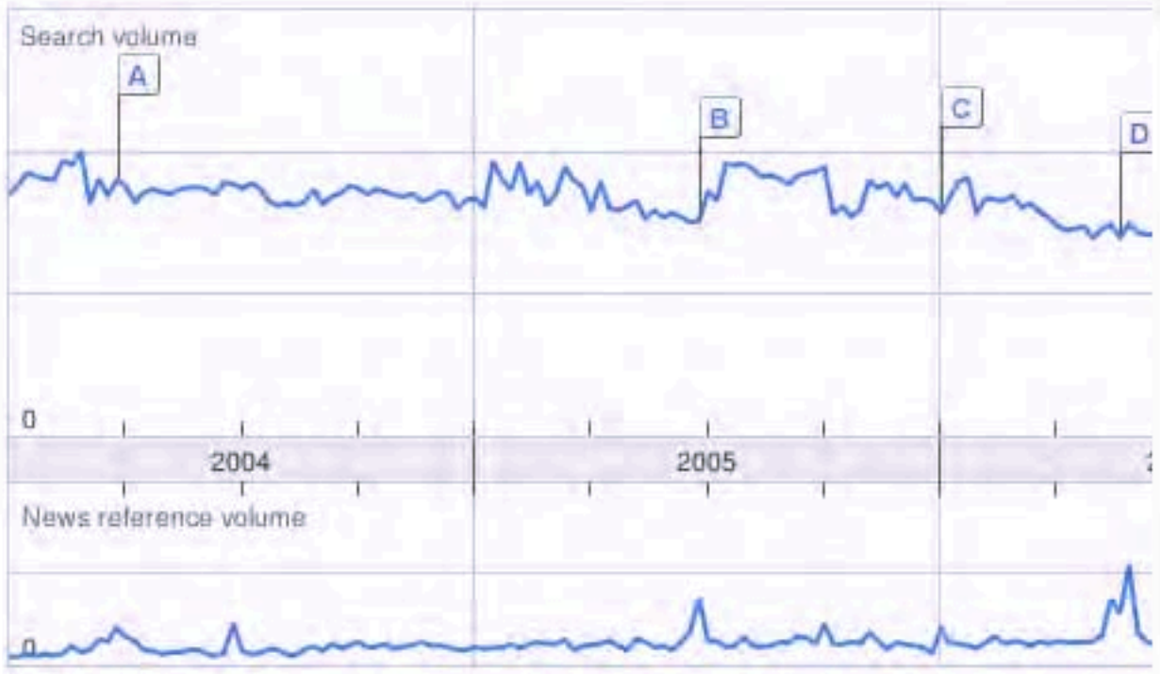
The theory that is here introduced, is that technologies that are in a very mature phase receive more hits from offshore automation centres, where state-of-the-art forward looking, research oriented technologies receive more queries from the US and Europe.

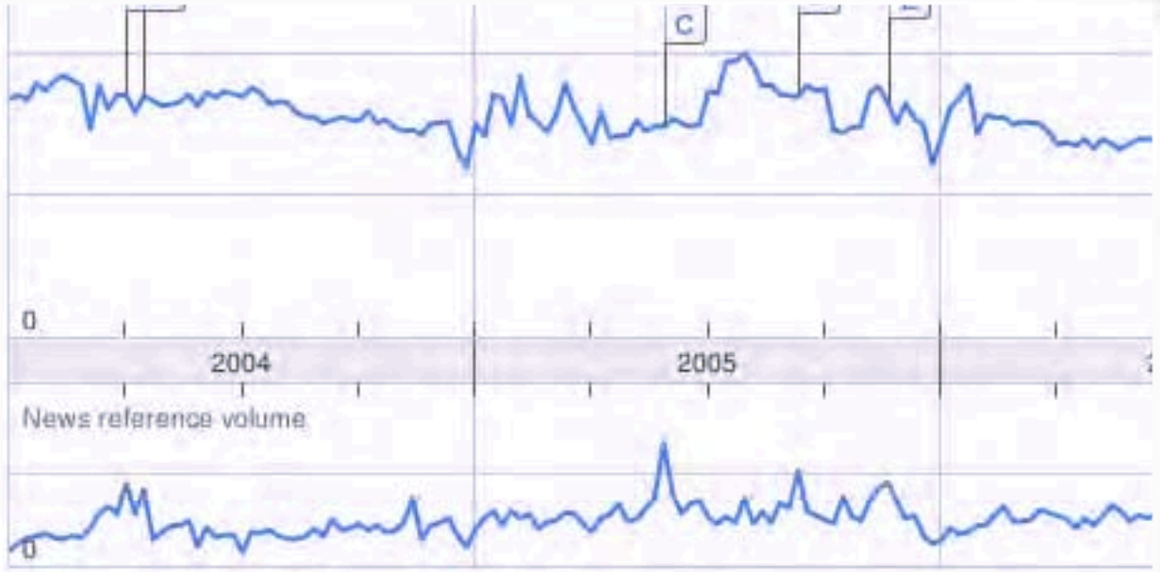
I'll make the case here for Java, XML, Ruby and Rexx.

This is the baseline of this comparison: the modern day status of COBOL in the world.



Java is queried most by outsourcing companies in India





XML is no surprise; seen as a sine qua non, not very widely loved but no real alternative after global acceptance.

Good for data, better for computers, not for human consumption. There is no better choice for data exchange. On the other hand: lots of configuration files are needlessly complex through needless usage of xml.

My guess here is that the high number for San Jose has to do with the XPATH and XQUERY efforts from IBM Research and the XML features in DB2.

OSI certified

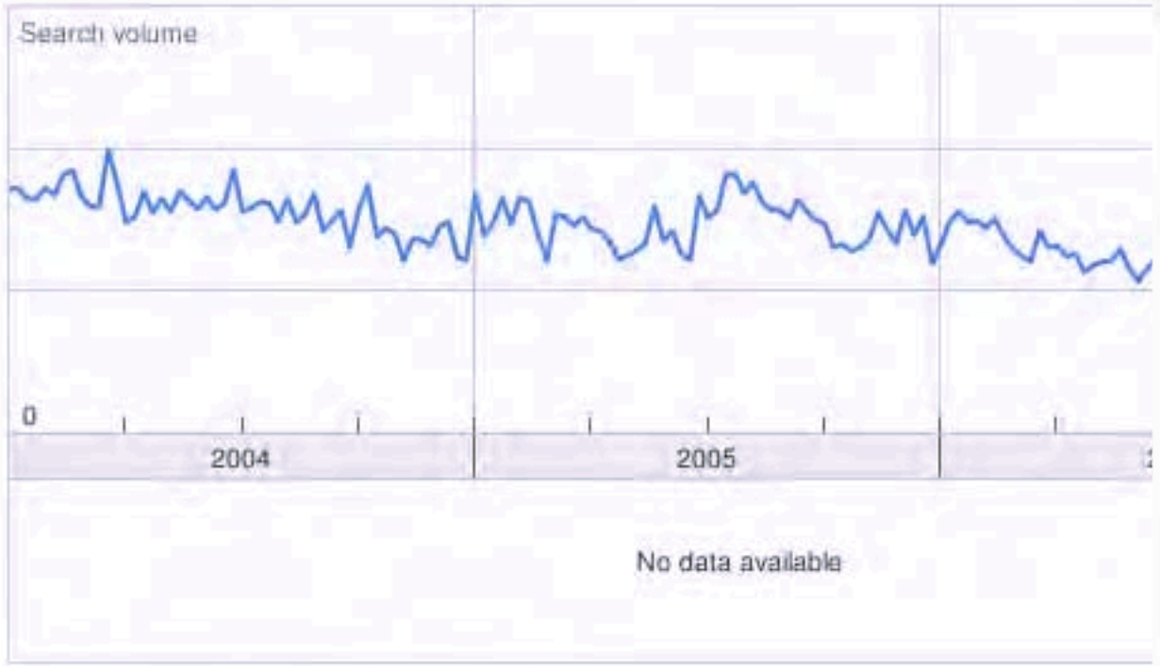
On the other hand, **JSON** is a trendy would-be replacement for XML.

As it is very modern, there are queries from US (West Coast) and Europe.

The numbers might indicate that Bangalore has more research oriented activity than for example Chennai or Mumbai. (The modern names for Madras and Bombai).

Ruby is the most hyped oo scripting language at the moment, and the absolute winner at the moment if we measure by book sales. Here we see the trend showing most queries from the US West Coast.

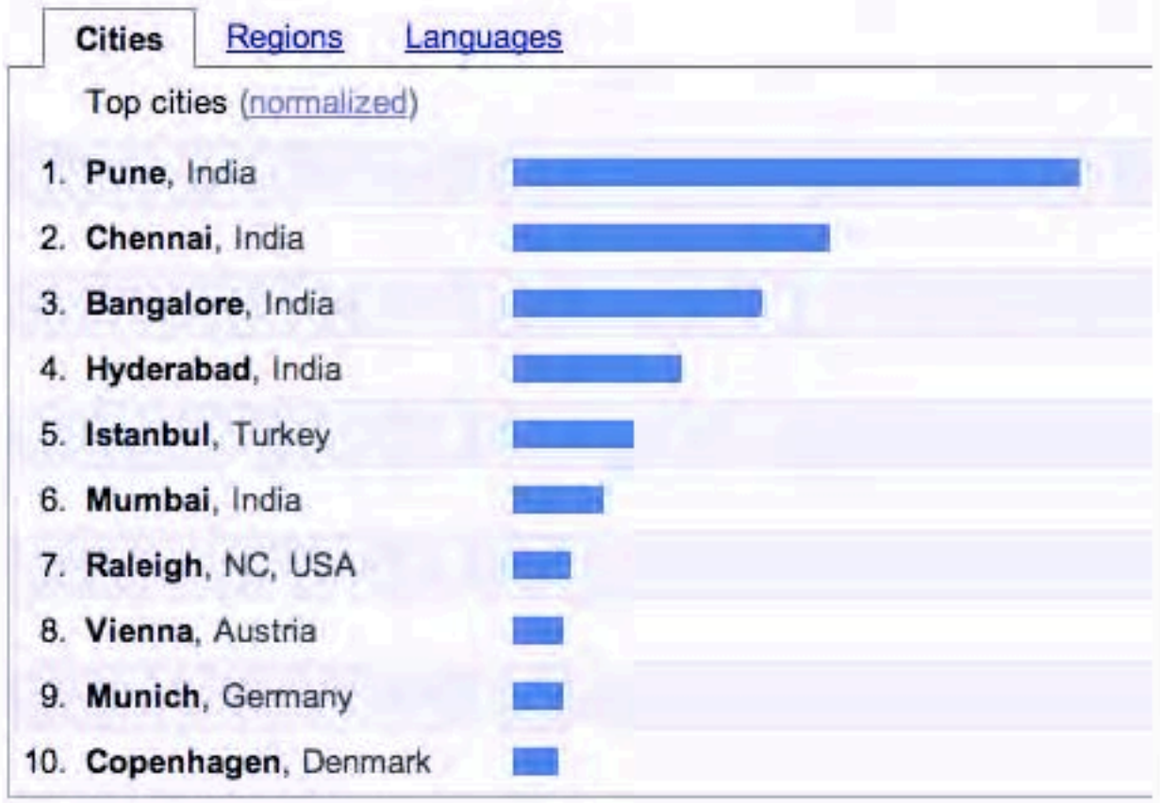
Following the here presented theory this indicates interest from research communities and the fact that is is not yet accepted as proven technology.



Rexx is at the same time an established scripting language, with lots of queries from India, doubtlessly for its use as the glue for older, traditional apps that have been offshored, but also leading edge, with the more research-oriented, agile US companies querying it in Google.

My interpretation here is that there is a distinct possibility that the US and Europe queries concern ooRexx, while the offshoring country queries concern mostly Classic Rexx - or Mainframe Rexx.

There is of course no solid proof for this.



OSI certified

Hard to do in Java, Easy to do in Rexx *Procedurally*

- Debugging and Trace
- Accessor Methods

Trace

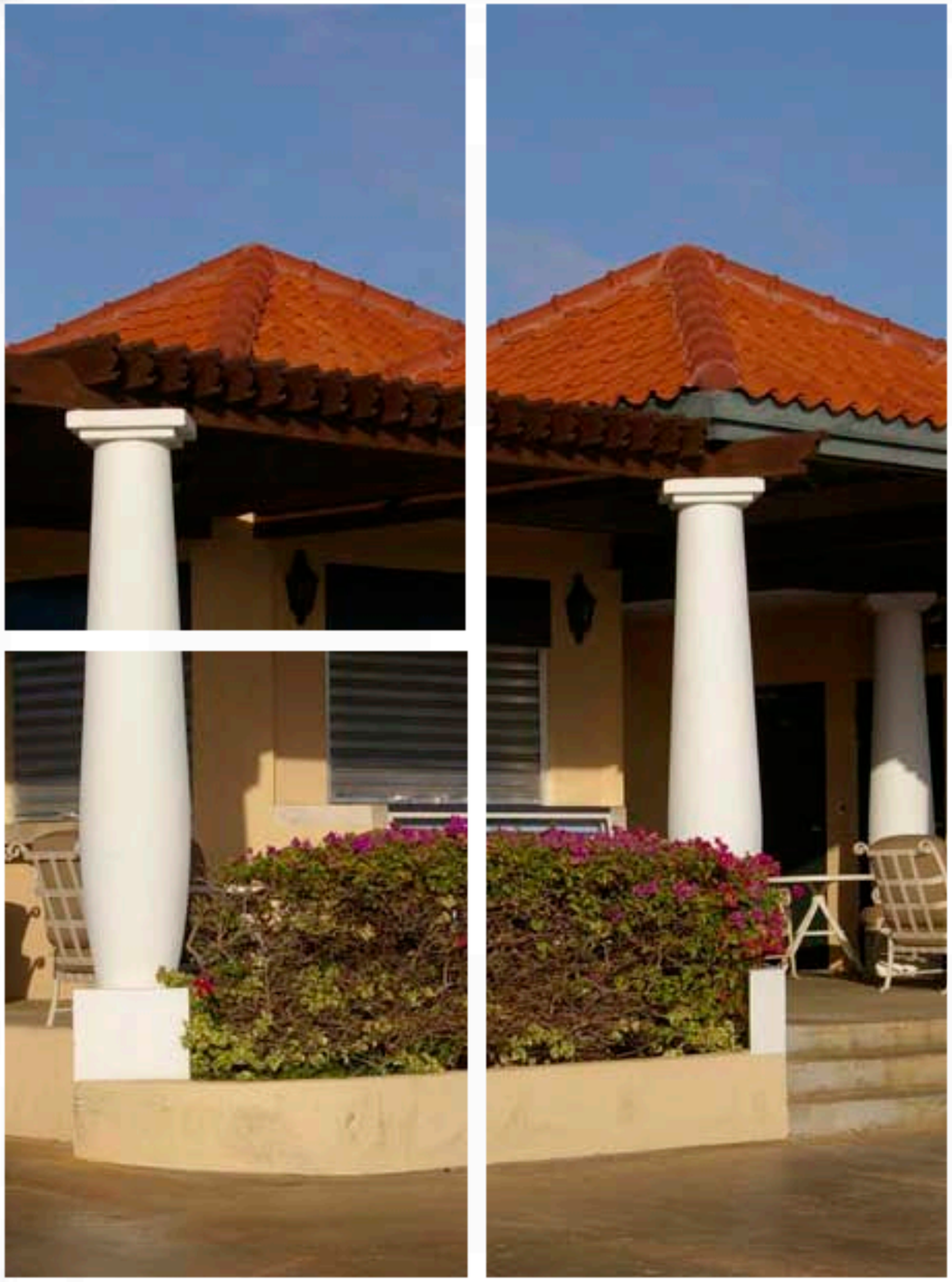
- Debugging server side code is very bothersome without trace
  - Ask anyone who needs to connect a remote debugger to some server code and suffer the performance
- Trace literally zips through the code and lets you catch the error quickly
  - It pays off to have sensible tracing criteria and limit output

Accessor Methods

- Java programmers have to count on the IDE to have this generation facility
- NetRexx has **properties indirect**
- ooRexx has accessor properties that use the = operator

Hard to do in Java, Easy to do in Rexx
Codewise

- Dynamically built Classes and Methods
- Duck Typing
- Aspect Oriented Programming
  - Metaclasses and metaprograming
  - Getting to method source and changing it
- Code blocks and Interpret
- Lispy things



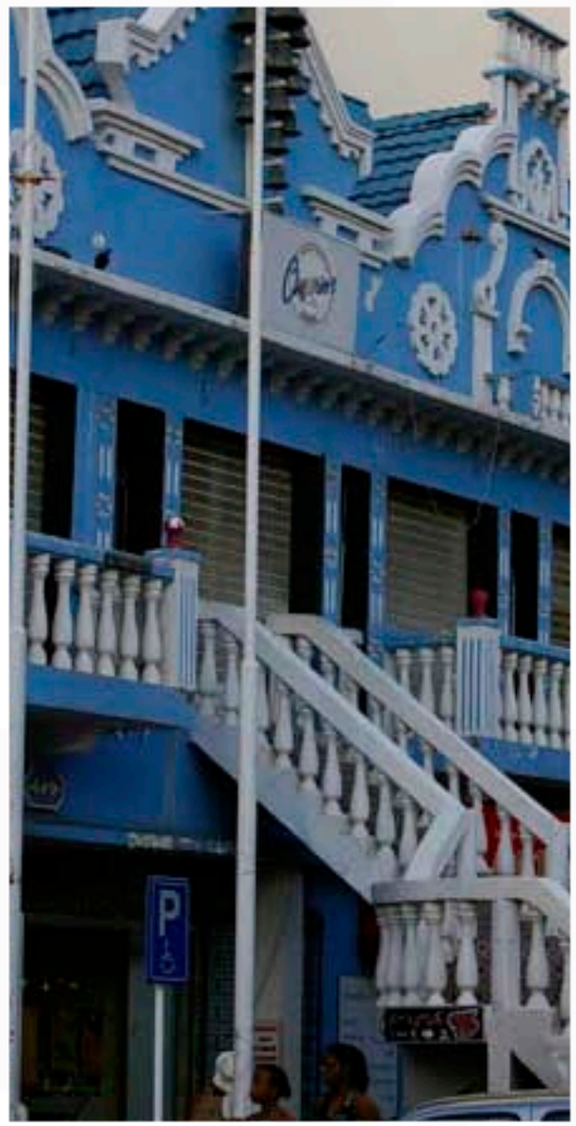
Dynamically building classes and methods

All without byte code manipulation

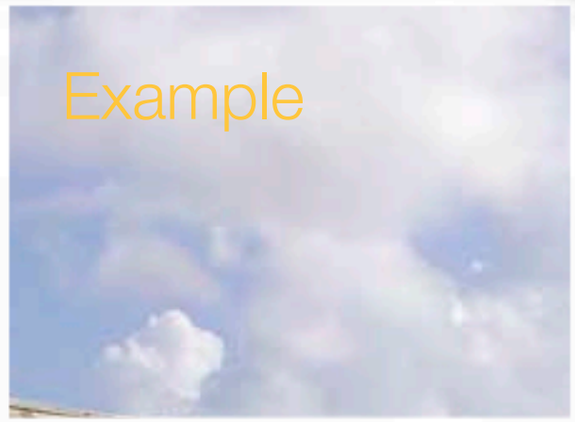
Dynamic Class Construction

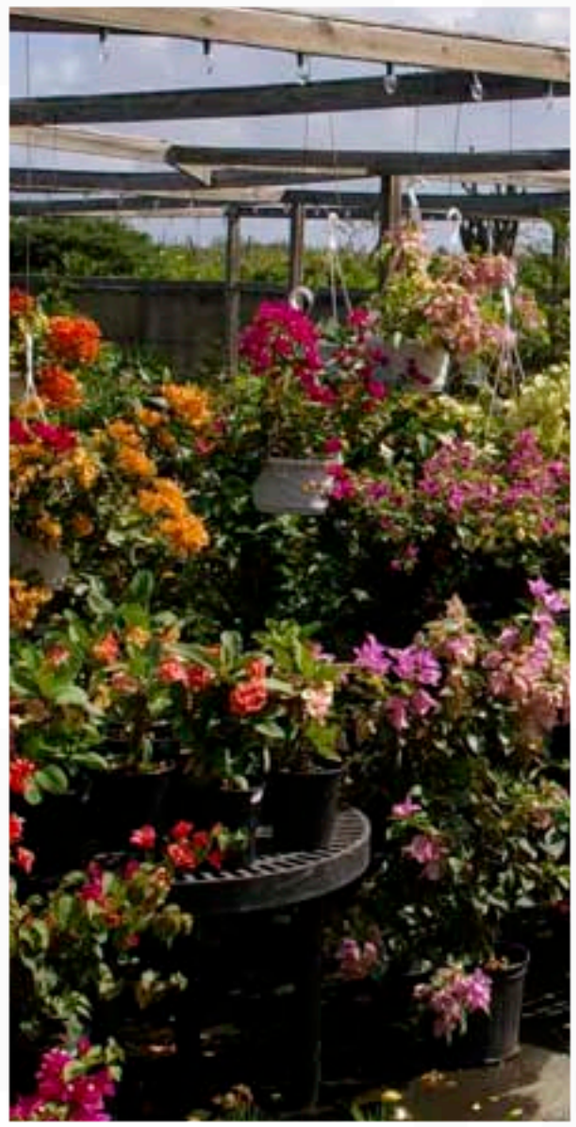
- ooRexx can build classes by just sending messages
- But why would you do that?
  - Model driven development, forward engineering of just modelled data
- Hard - to impossible - in Java
  - Have to revert to byte code engineering frameworks like BCEL, ASM
  - Then, most of the time, have to write “Java Assembler”

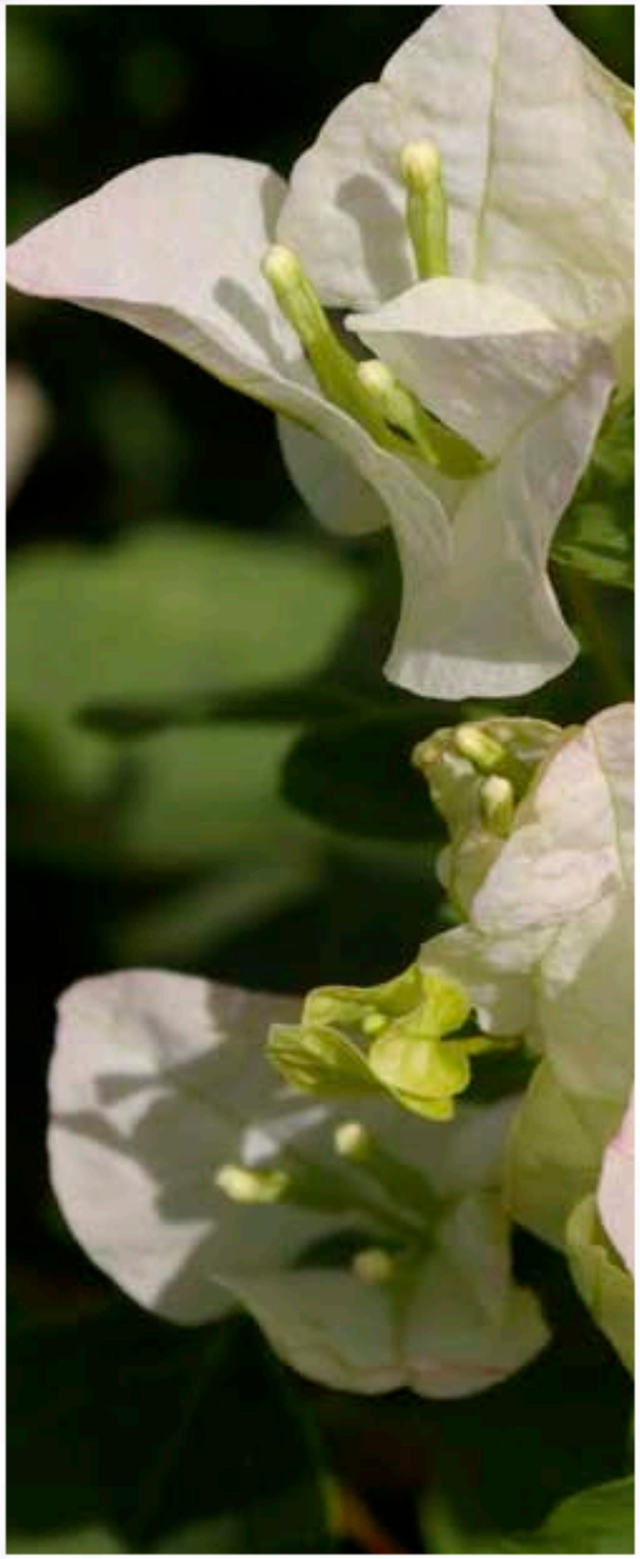
```
ClassWriter cw = new ClassWriter(0);
cw.visit(V1_1, ACC_PUBLIC, "Example", null, "java/lang/Object", null);

// creates a MethodWriter for the (implicit) constructor
MethodVisitor mw = cw.visitMethod(ACC_PUBLIC,
        "<init>",
        "()V",
        null,
        null);
// pushes the 'this' variable
mw.visitVarInsn(ALOAD, 0);
// invokes the super class constructor
mw.visitMethodInsn(INVOKESPECIAL, "java/lang/Object", "<init>", "()V");
mw.visitInsn(RETURN);
// this code uses a maximum of one stack element and one local variable
mw.visitMaxs(1, 1);
mw.visitEnd();

// creates a MethodWriter for the 'main' method
mw = cw.visitMethod(ACC_PUBLIC + ACC_STATIC,
        "main",
        "([Ljava/lang/String;)V",
        null,
        null);
// pushes the 'out' field (of type PrintStream) of the System class
mw.visitFieldInsn(GETSTATIC,
        "java/lang/System",
        "out",
        "Ljava/io/PrintStream;");
// pushes the "Hello World!" String constant
mw.visitLdcInsn("Hello world!");
// invokes the 'println' method (defined in the PrintStream class)
mw.visitMethodInsn(INVOKEVIRTUAL,
        "java/io/PrintStream",
        "println",
        "(Ljava/lang/String;)V");
mw.visitInsn(RETURN);
// this code uses a maximum of two stack elements and two local
// variables
mw.visitMaxs(2, 2);
mw.visitEnd();

// gets the bytecode of the Example class, and loads it dynamically
byte[] code = cw.toByteArray();
```

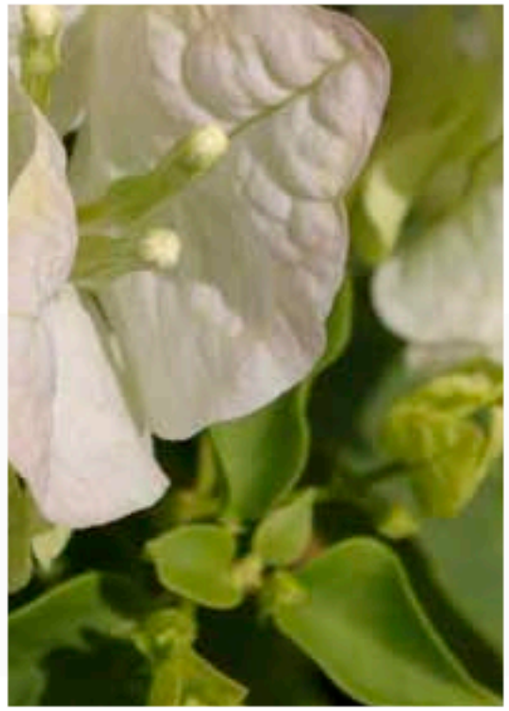


```
t = .test1~new

t~testMethod('aap','noot','mies')
test2 = .object~subclass('test1')
test2~define("testje","say 'thats it!'")
test2~define("unknown","say 'doh'")
s = test2~new
s~testje
s~dilbert
say 'now is the time'~word(3)
exit

::class test1

::method init
say "init of class test1"

::method testMethod
use arg a, b, c

say a
say b
say c

return 0
```









Duck Typing

... Walks like a duck, talks like a duck ... must be of type Duck

[inspired by our Ruby friends]

Just looks at available methods

- As opposed to Java, an object's type is determined by what it can do, not by its class
- In Java, to successfully call a method, it must belong to the class, or an implemented interface, of an object

The UNKNOWN method

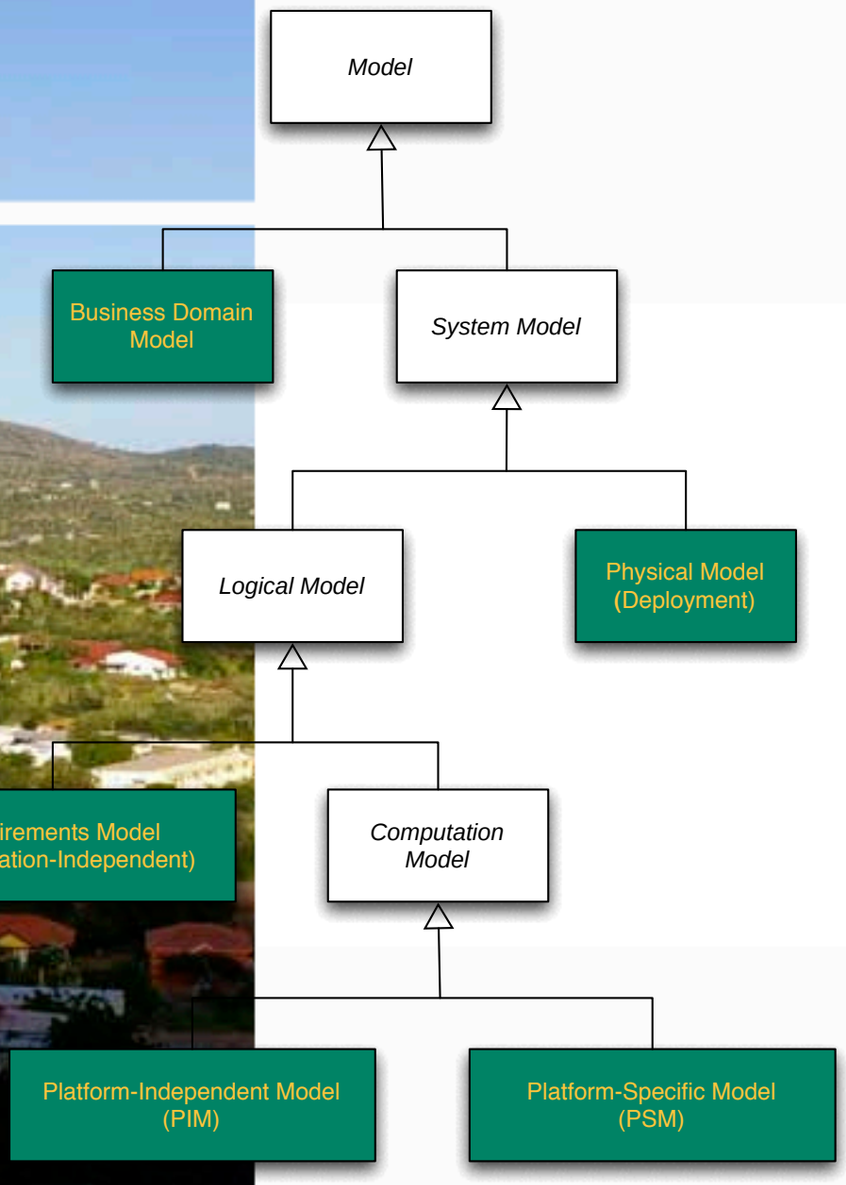
- Duck Typing benefits a catch-all method for messages that are not understood by the receiver
- ooRexx has this built in
- Java will issue you a ‘method not found’ at compiler time

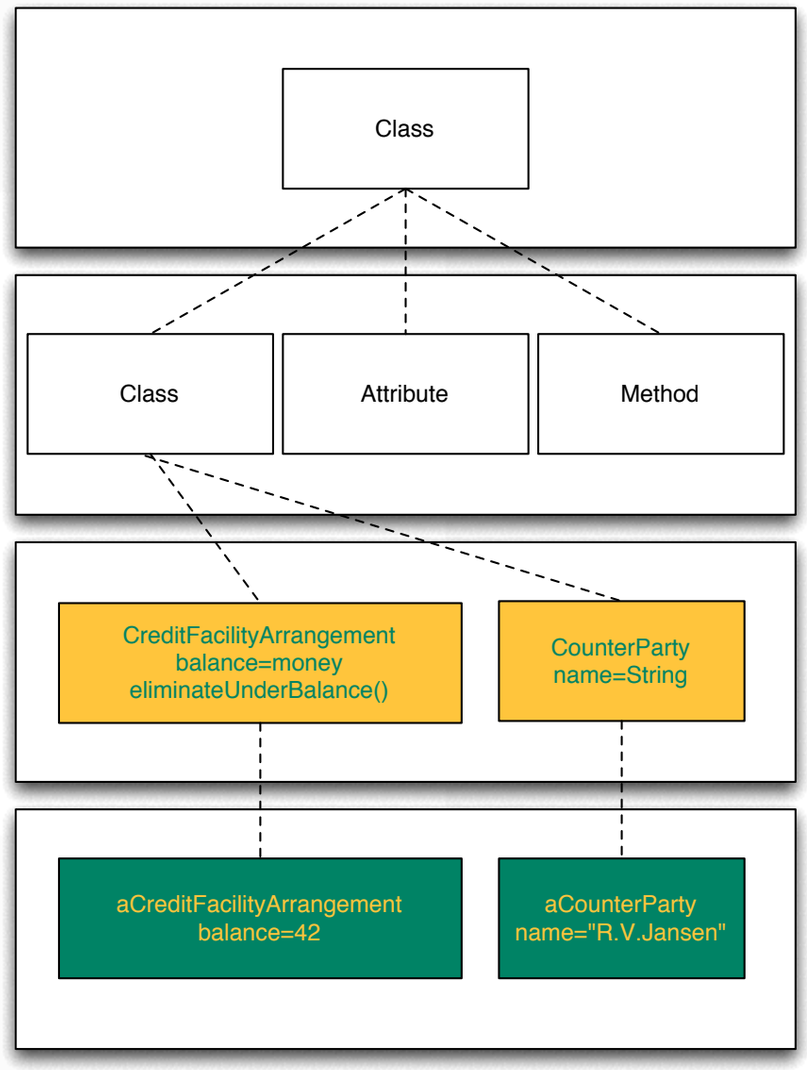
“There's no doubt that you can prototype more quickly in an environment that lets you get away with murder at compile time, but I do think the resulting programs are less robust. I think that to get the most robust programs, you want to do as much static type checking as possible.”

Josh Bloch, Author of *Effective Java*

Static type checking for robustness

Level Issues



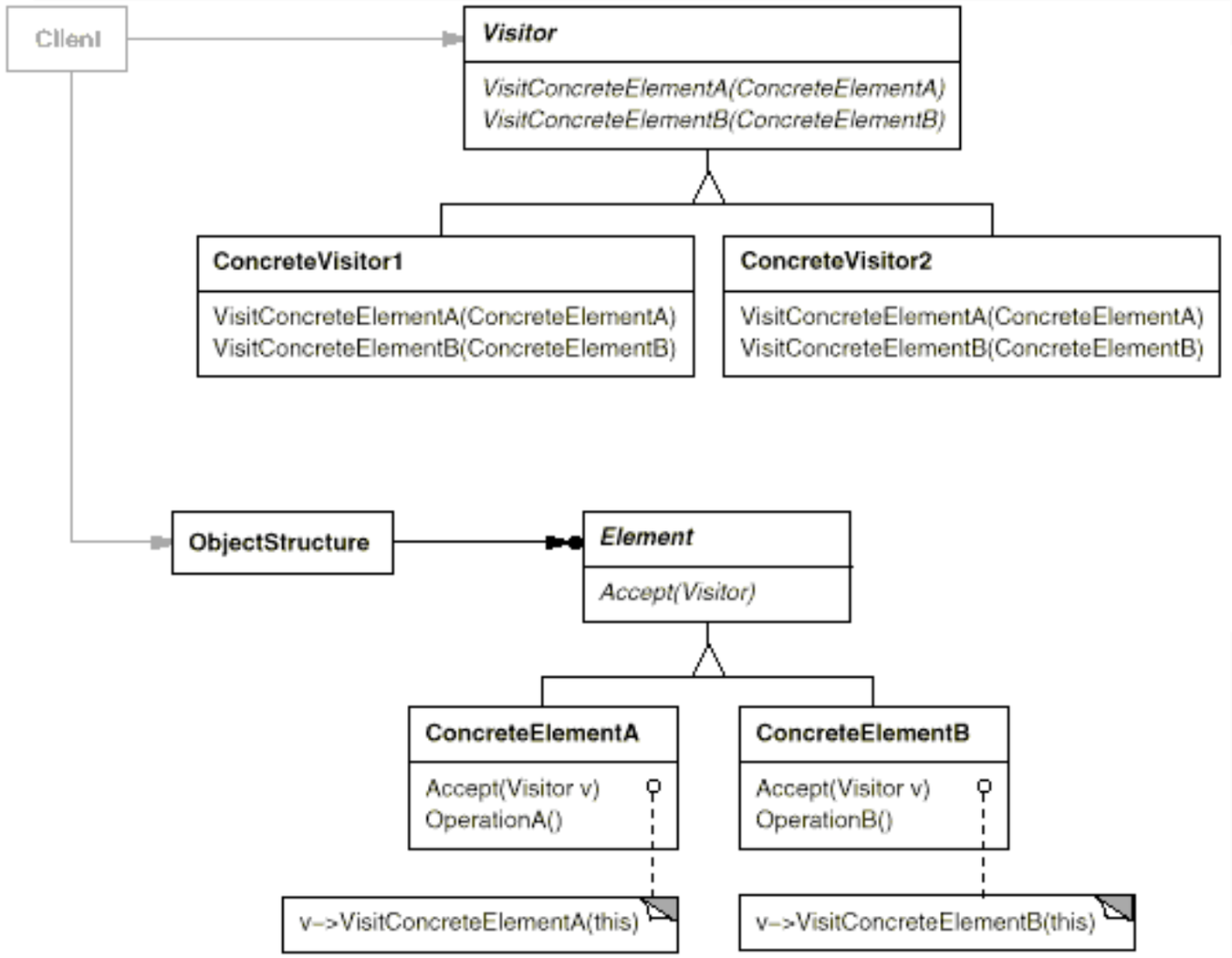


Level errors are the reason why optional type checking is preferable

Sometimes static typing outlook is grim

For example, implementing a visitor pattern for all subclasses of a particular class

The Visitor Pattern



Used in compilers, but also in our applications **Universal Editor**.

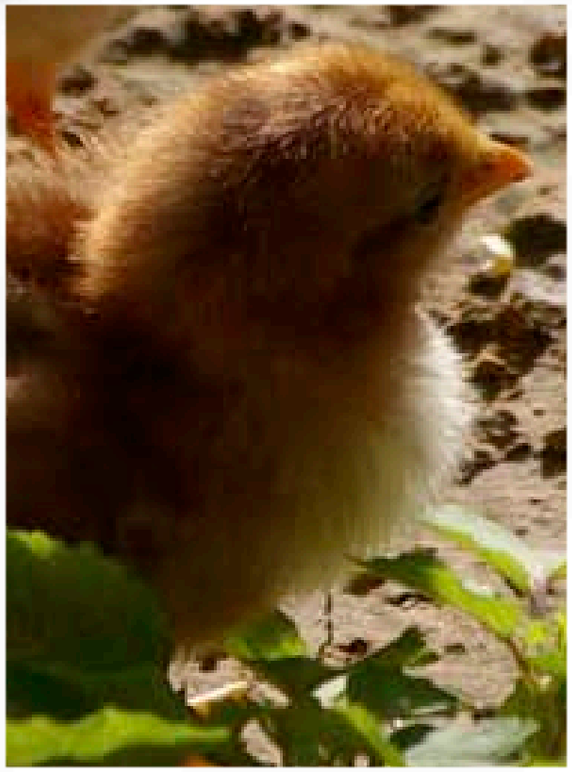
Have to adapt the visitor class every time a new subtype is generated.

That's fairly inefficient.

Visitor Pattern

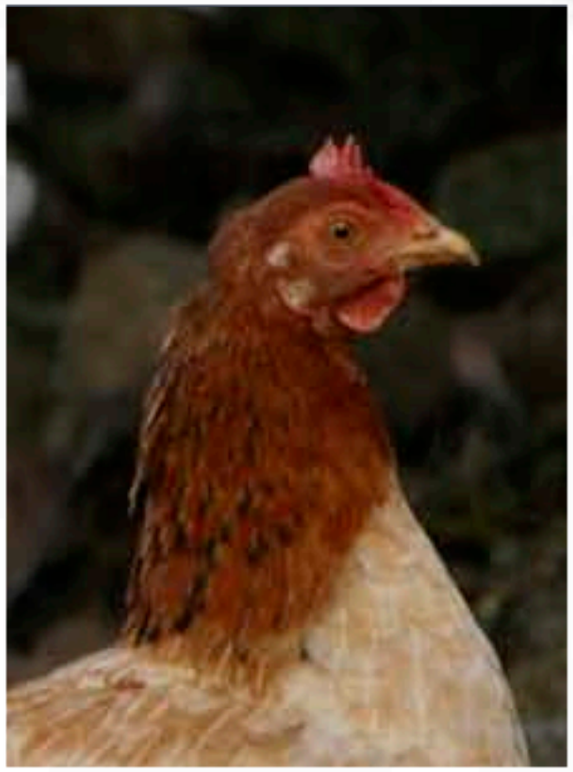
The Visitor pattern, when employed in Java (or C++, for that matter), requires the programmer to repeat the same method with a signature that matches every subtype argument.

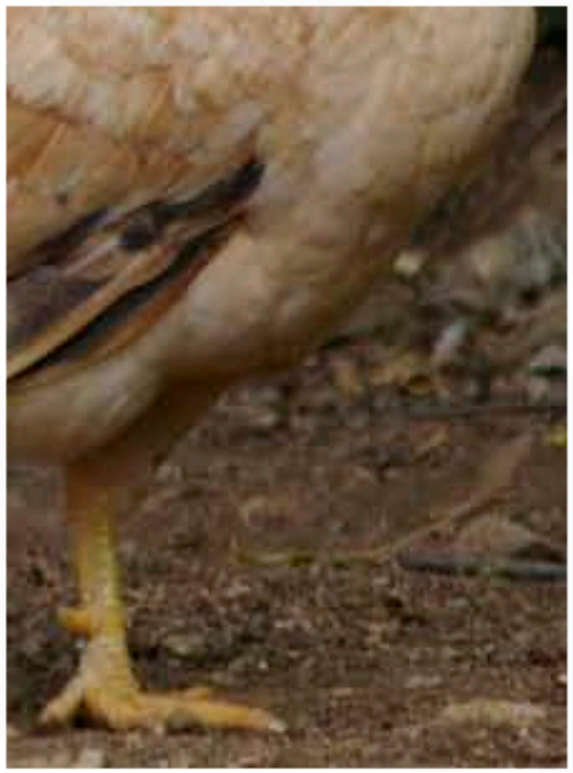
```
class Visitor abstract
  method visit(a  = Abbrev
  method visit(c  = Classi
  method visit(c  = Classi
  method visit(c  = TypeSe
  method visit(s  = Scheme
  method visit(s  = SubSch
  method visit(d  = Descri
  method visit(d  = Domain
  method visit(e  = com.ab
  method visit(f  = Fundam
  method visit(i  = Identi
  method visit(i  = Instan
  method visit(i  = Instan
  method visit(i  = Involv
  method visit(c  = Commun
  method visit(i  = Indivi
  method visit(e  = Employ
  method visit(o  = Organi
  method visit(o  = Organi
  method visit(s  = StaffM
  method visit(is = Indivi
  method visit(es = Employ
  method visit(is = Involv
  method visit(os = Organi
  method visit(u  = Useric
  method visit(r  = Recomm
  method visit(e  = Passwo
  method visit(i2 = ISOCou
  method visit(i3 = ISOCou
  method visit(m  = Measur
  method visit(m  = ModelC
  method visit(n  = Name)
  method visit(n  = com.ab
  method visit(o  = Oid) a
  method visit(o  = com.ab
```



Metaclasses

And Aspect Oriented programming - the two go together quite nicely





```
::class metatest public subclass class

  ::method unknown
    use arg msg, args
    if msg = 'TRACE' then
      do
    s = self~methods
    do while s~available
      mname = s~index
      m = s~item
      if m~source <> "" then
        do
          methodText = self~method(mname~string)~source
          methodText[1] = 'trace results;' methodText[1]
          tracedMethod = .method~new(' ', methodText)
          self~define(mname~string, tracedMethod)
        end
    s~next
  end -- do while
  return self~new(args)
  end
```

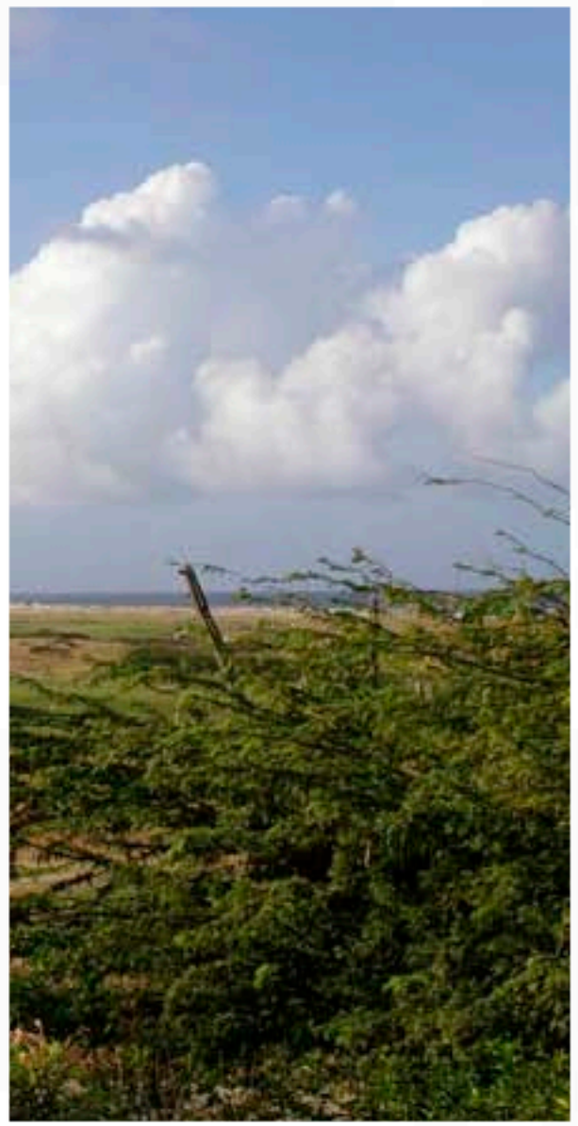
The idea: have a metaclass that inserts TRACE statements

```
::class metatest public subclass class

  ::method unknown
    use arg msg, args
    if msg = 'TRACE' then
      do
     s = self~methods
     do while s~available
       mname = s~index
       m = s~item
       if m~source <> "" then
         do
         methodText = self~method(mname~string)~source
         i=1;
         do while methodText[i]~string~wordpos('expose') > 0
            i = i + 1
         end
         methodText[i] = 'trace results;' methodText[i]
         tracedMethod = .method~new(' ', methodText)
         self~define(mname~string, tracedMethod)
       end
    s~next
  end -- do while
  return self~new(args)
   end
```

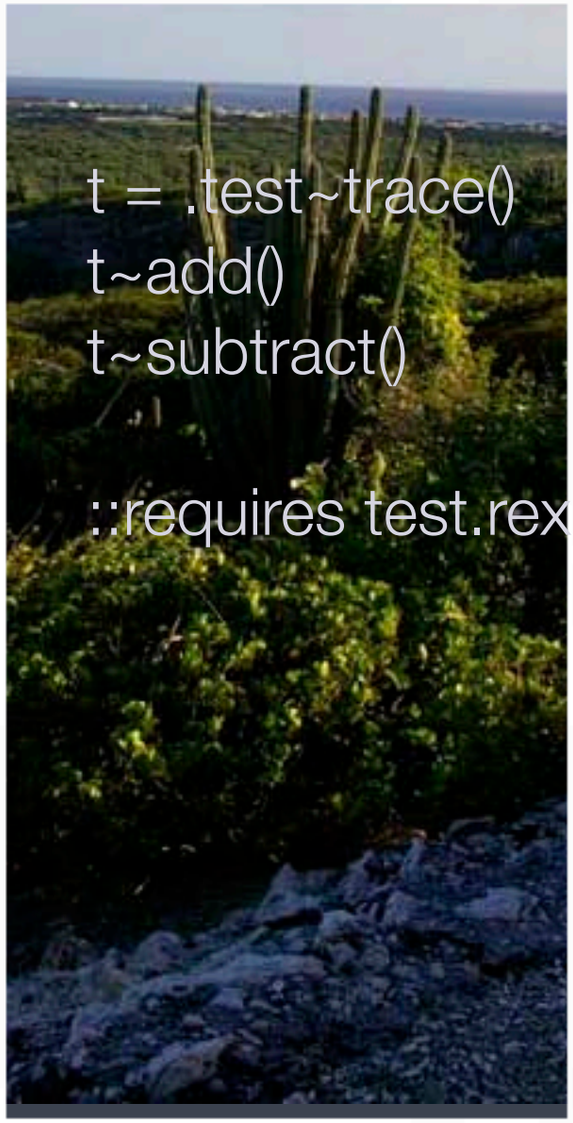
Add a check, ‘expose’ must be first keyword in method if used

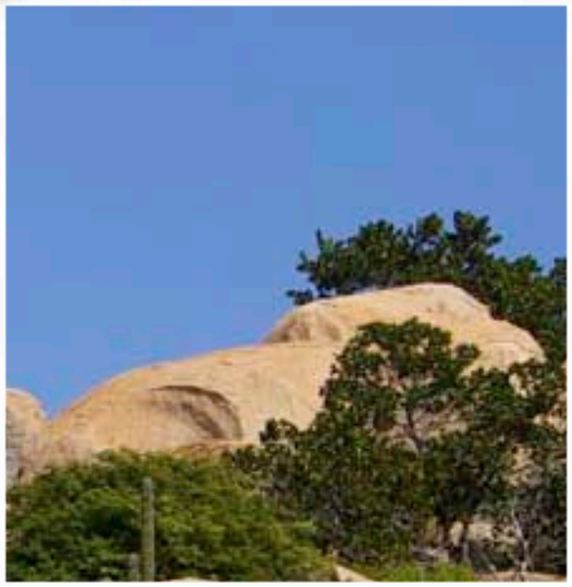
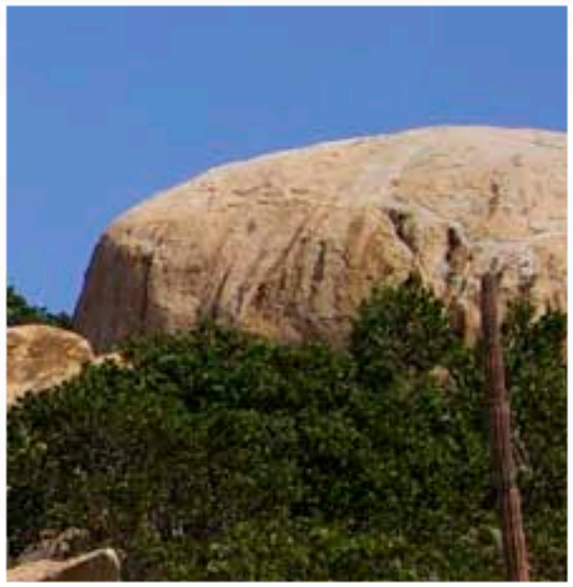
It might be an idea to let go of this constraint.

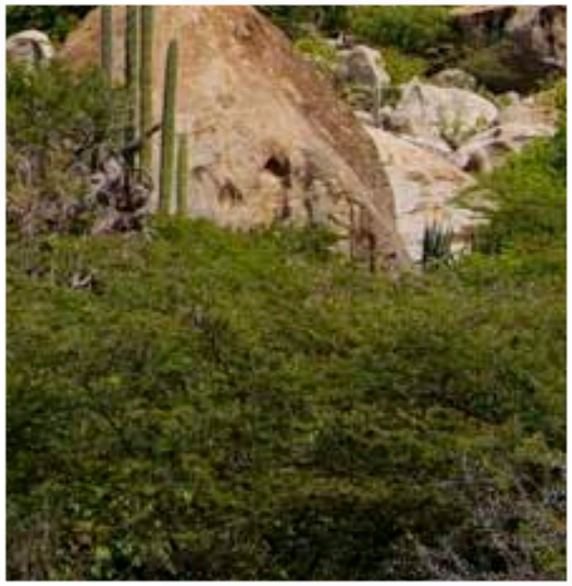
```
::requires metatest.rex
::class test public metaclass metatest

  ::method init
    say 'we instigated a new instance of class test'

  ::method add
    x = 40
    y = 2
    say 'x + y =' x+y
    self~multiply

::method subtract
    x = 44
    y = 2
    say 'x - y =' x-y
    self~multiply

::method multiply
    x = 22
    y = 2
    say 'x * y =' x*y
```



```
[liberty:~/rxtrace] rvjansen% rexx testtest.rex
     1 *-* trace results;
     1 *-* say 'we instigated a new instance of class test'
       >>>   "we instigated a new instance of class test"
we instigated a new instance of class test
     1 *-* x = 40
       >>>   "40"
     2 *-* y = 2
       >>>   "2"
     3 *-* say 'x + y =' x+y
       >>>   "x + y = 42"
x + y = 42
     4 *-* self~multiply
     1 *-* x = 22
       >>>   "22"
     2 *-* y = 2
       >>>   "2"
     3 *-* say 'x * y =' x*y
       >>>   "x * y = 44"
x * y = 44
     1 *-* x = 44
       >>>   "44"
     2 *-* y = 2
       >>>   "2"
     3 *-* say 'x - y =' x-y
       >>>   "x - y = 42"
x - y = 42
     4 *-* self~multiply
     1 *-* x = 22
       >>>   "22"
     2 *-* y = 2
       >>>   "2"
     3 *-* say 'x * y =' x*y
       >>>   "x * y = 44"
x * y = 44
```






Monkey Patching a Java Object

http://kofno.wordpress.com/2007/02/24/monkey-patch-java-objects-from-jruby/

What is a monkey patch?

- It is adding a method to an existing Java object using a proxy in another language
- ooRexx can pull this off using BSF4Rexx. The proxying ooRexx class can have methods dynamically added and pass them on to the Java object - a monkey patch.

(That method is only available from the proxy class)

Monke

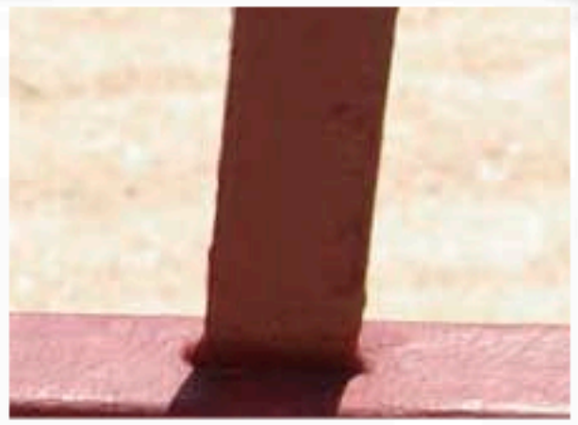




Closures

Rexx had it before it was called a closure

```
#! /opt/ooRexx/bin/rexx

s = .symposium~new

::class symposium
  ::method init
    l = .array~of('chip','gil','lee','mark','mike','rick','rony')
    x = .xeq~new
    x~map(l, "say hello")
    x~map(l, "say goodbye")

::class xeq
  ::method do
    use arg b, c
    interpret c b

  ::method map
    use arg a, v
    do k over a
      self~do(k,v)
    end
```

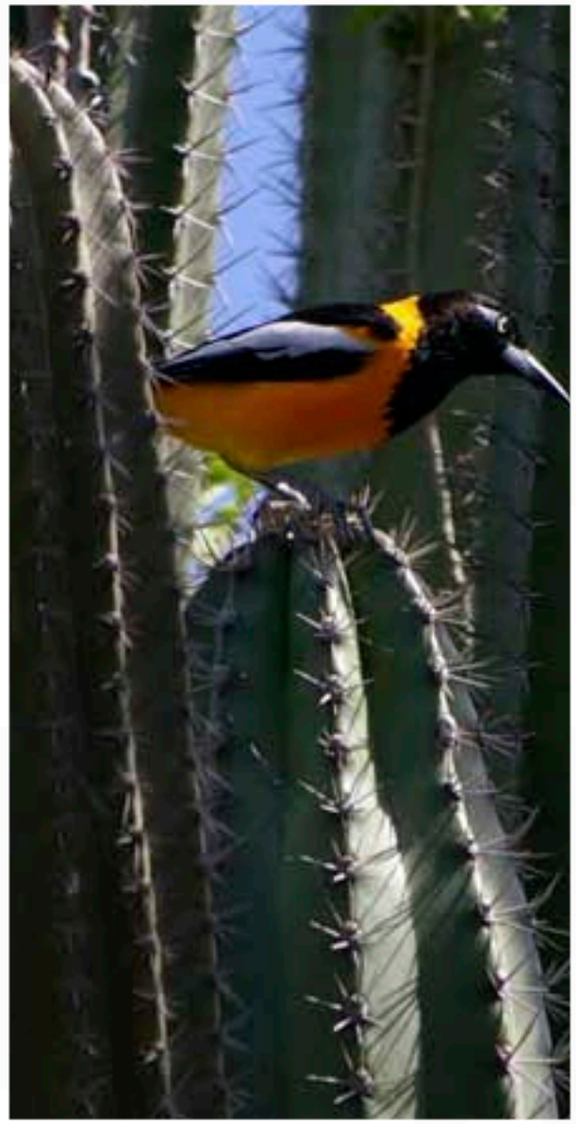


Code blocks are a popular modern idiom to, for example, compactly express actions on a collection

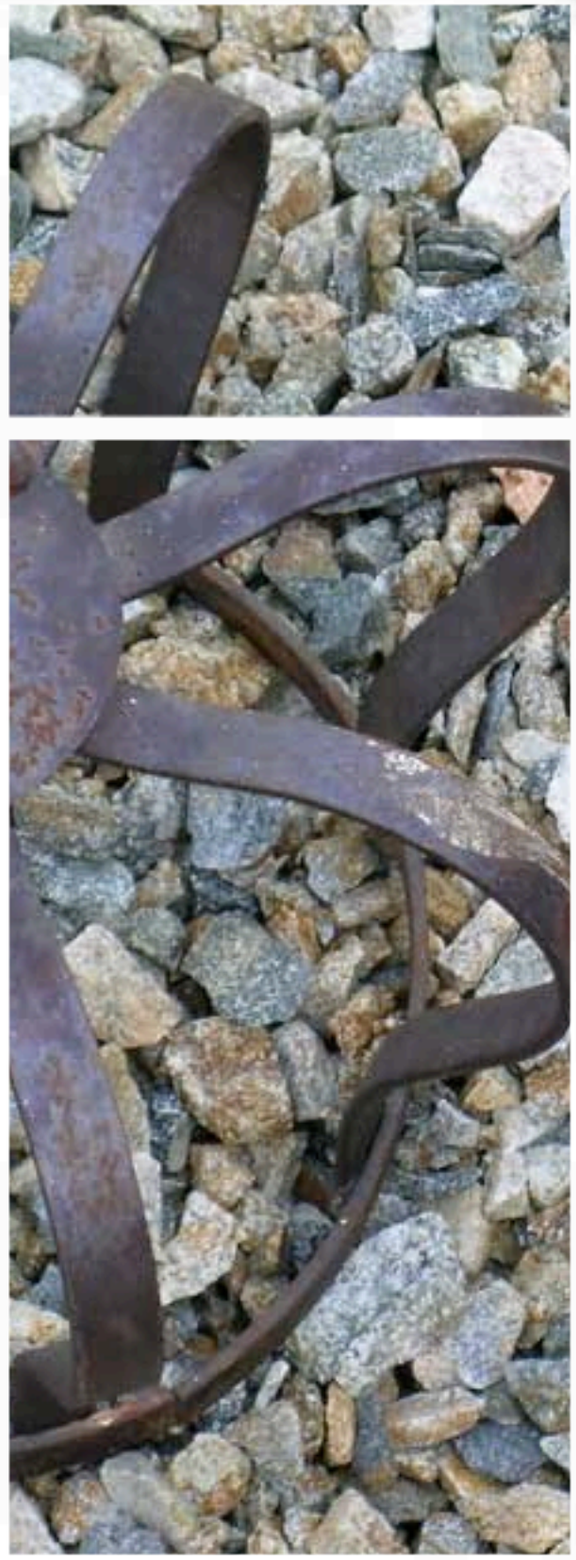
They can be found in modern languages like Ruby, Groovy and Python. Java needs inner class syntax to approximate it, and then still it does not give the same ease of use.

s = .symposium~new

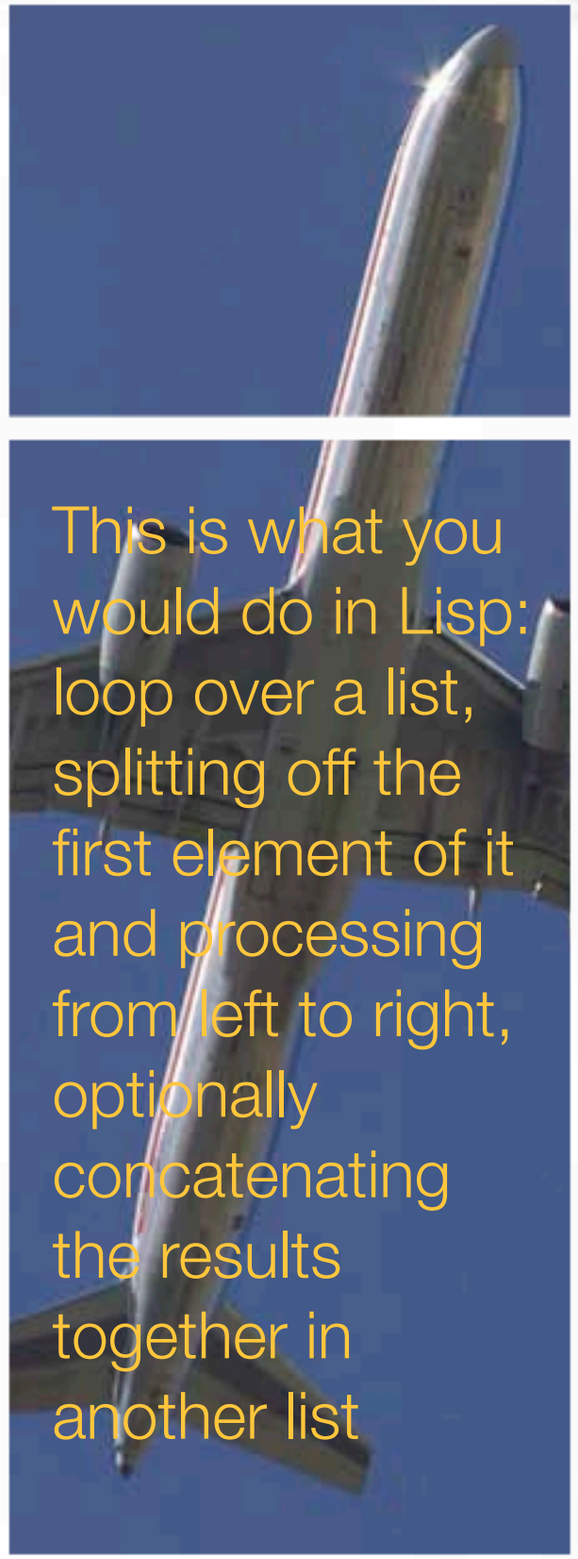
HELLO CHIP
HELLO GIL
HELLO LEE
HELLO MARK
HELLO MIKE
HELLO RICK
HELLO RONY
GOODBYE CHIP
GOODBYE GIL
GOODBYE LEE
GOODBYE MARK
GOODBYE MIKE
GOODBYE RICK
GOODBYE RONY



Using **Parse** as **LISP**

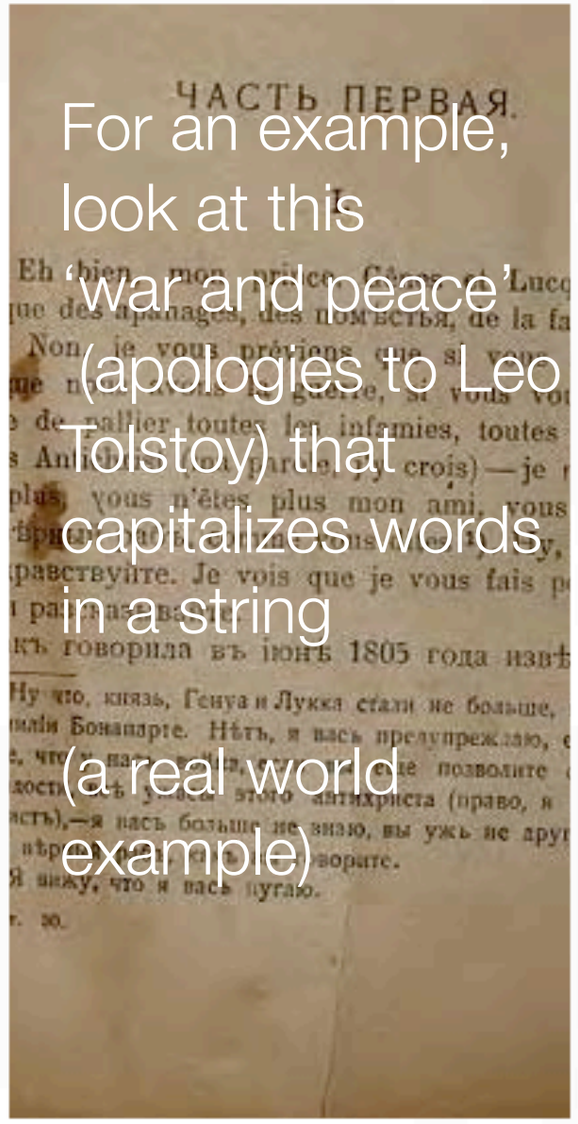


```
cdr = “foo bar baz”
loop while cdr <> ‘’
    parse var cdr car ‘ ‘ cdr
      -- do something to car
end
```



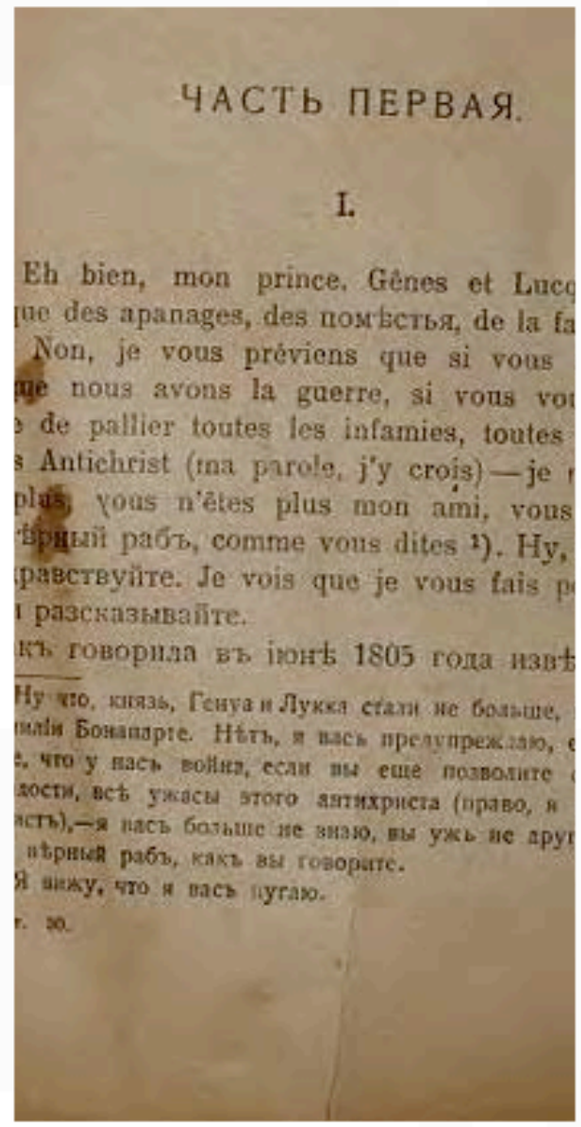
```
import java.util.regex.*;

/**
 * Static methods to parse out words from a String
 *
 */
public class Words {

  /**
   * Count the number of words in the String
   *
   * @param str a string containing words that match the regular expression \w+
   * separated by \s+
   *
   * @return int the number of words in the string
   */
  public static int countWords( String str ) {
    String[] words = getWords(str);Image:WarAndPeace.jpg
    int numWords = words.length;
    return numWords;
  }

  /**
   * Gives a String array containing the parsed out words from the string. The
   * method uses the regular expression \s+ to split the string into words.
   *
   * @param str a string containing words that match the regular expression \w+
   * separated by \s+
   *
   * @return String[] containing the words
   */
  public static String[] getWords( String str ) {
    String[] words = java.util.regex.Pattern.compile("\\s+").split(str.trim());
    return words;
  }
```

Java can be very ‘wordy’

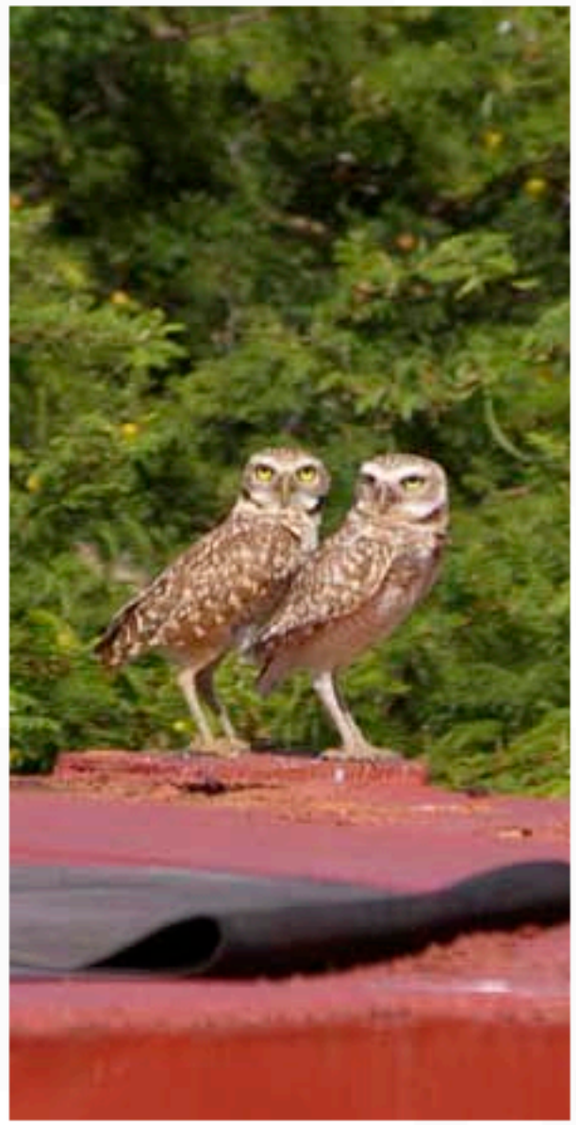


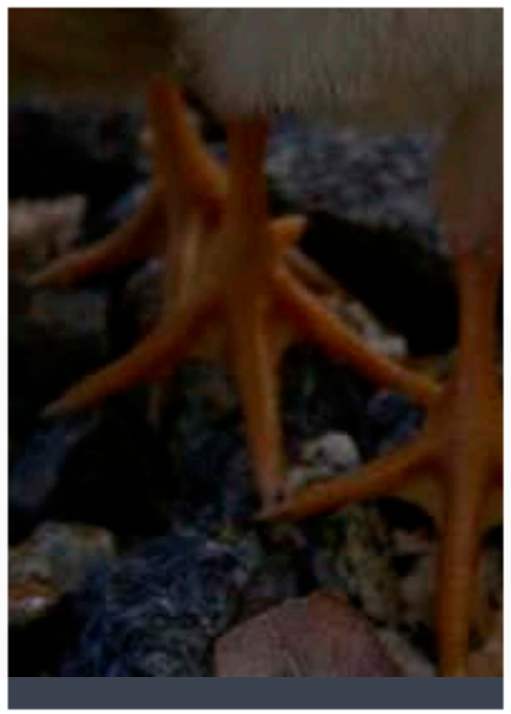
For an example, look at this ‘war and peace’ (apologies to Leo Tolstoy) that capitalizes words in a string (a real world example)

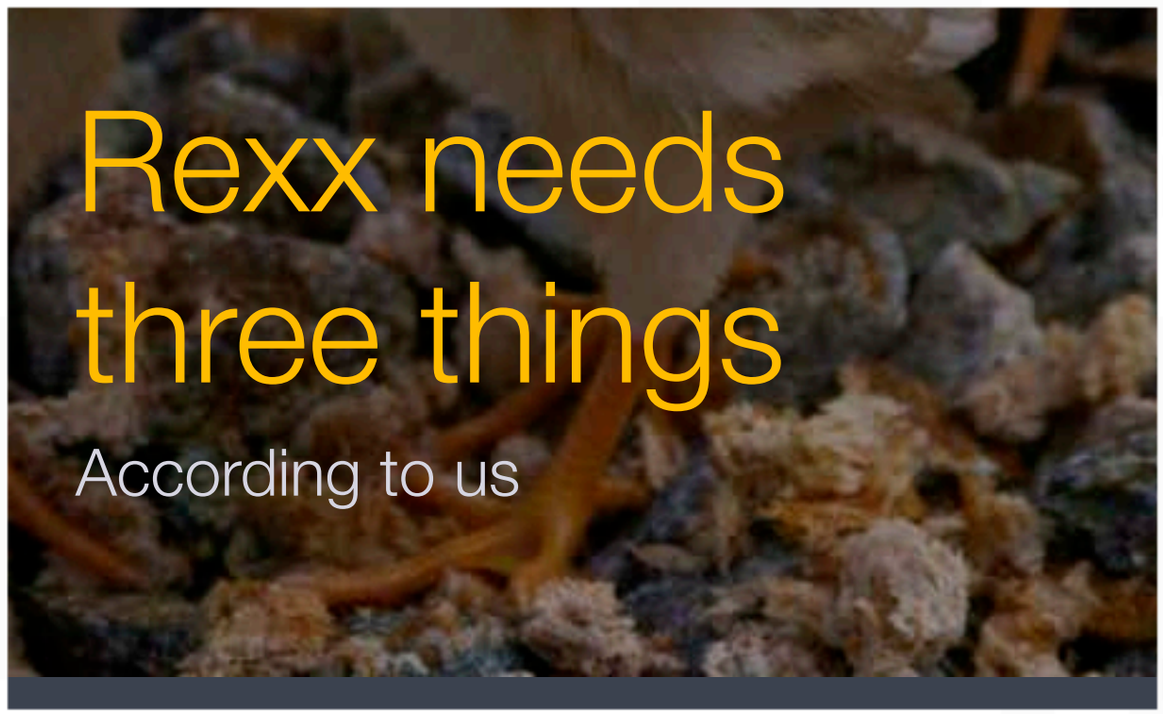
```
/**
 * Capitalise the first letter of each word in the string. The method uses
 * countWords(String) and getWords(String).
 *
 * @param str a string containing words that match the regular expression \w+
 * separated by \s+
 *
 * @return String containing the original string with the first letter of each
 * word in uppercase.
 */
public static String capitalise( String str ) {
  String capitalised = null;
  int numWords = countWords(str);
  String[] words = getWords(str);
  for (int i = 0; i < numWords; i++) {
    StartingBuffer sb = new StringBuffer(words[i]);
    Character c = sb.charAt(0);
    sb.setCharAt(0, Character.toUpperCase(c));
    words[i] = sb.toString();
    if (capitalised == null) {
      capitalised = words[i];
    } else {
      capitalised = capitalised + " " + words[i];
    }
  }
  return capitalised;
 }

}
```



```
cdr = “foo bar baz”; line = ‘’
loop while cdr <> ‘’
    parse cdr car ‘ ‘ cdr
      line = line car.upper(1,1)
end
```

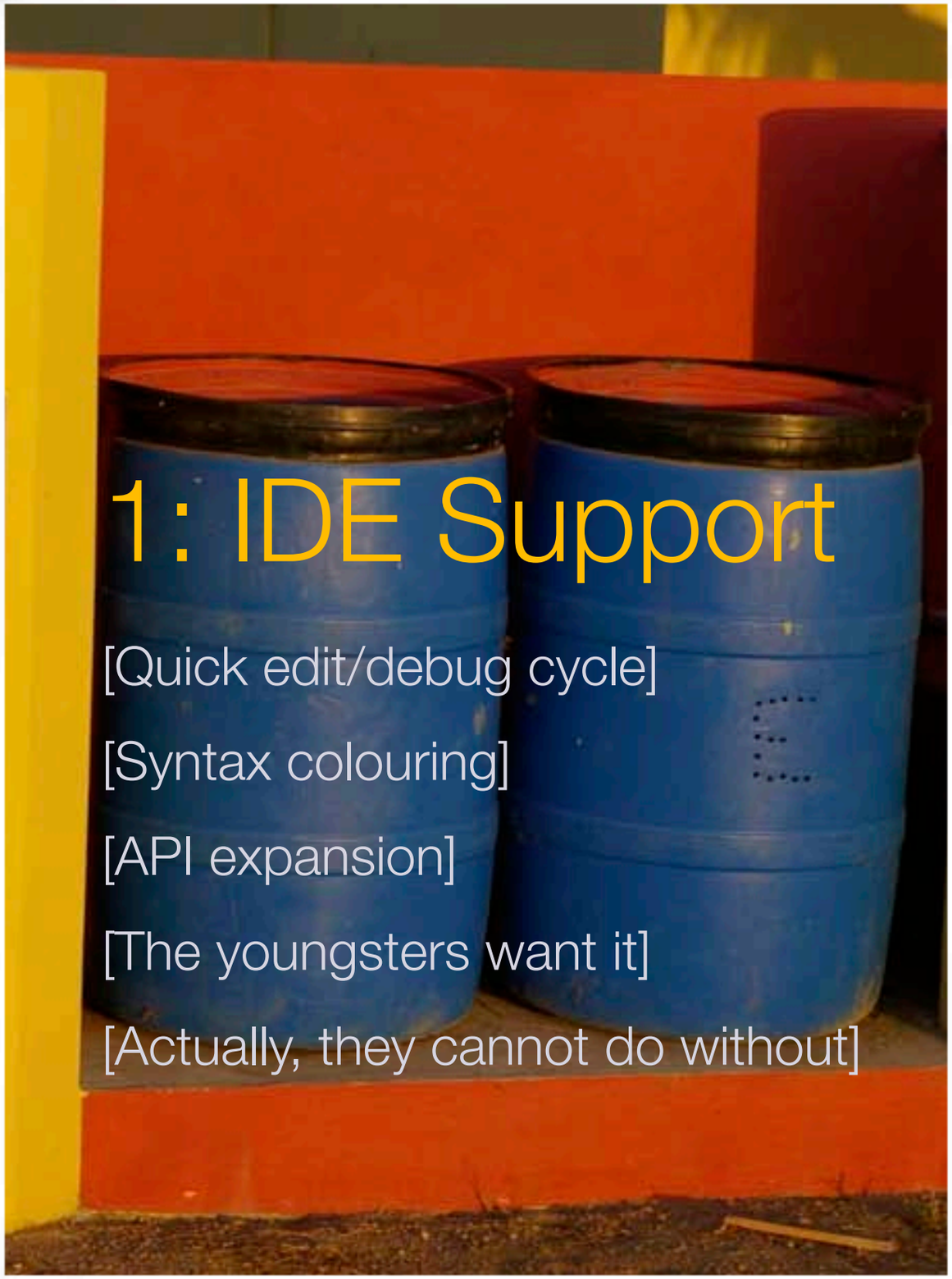






Rexx needs three things

According to us



1: IDE Support

[Quick edit/debug cycle]

[Syntax colouring]

[API expansion]

[The youngsters want it]

[Actually, they cannot do without]

2: Web Framework & ORM layer

[To quickly put together a web app]

[Think Rexx on Rails]

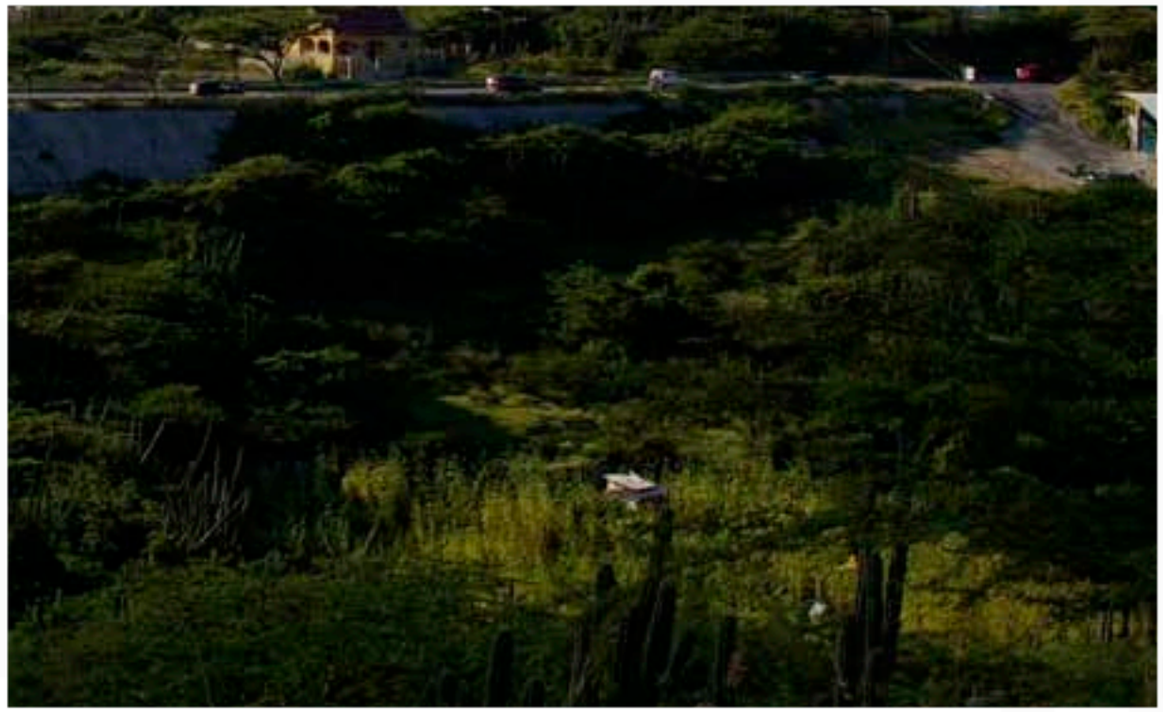
3: Dialect Unification

[The story is too difficult to sell]

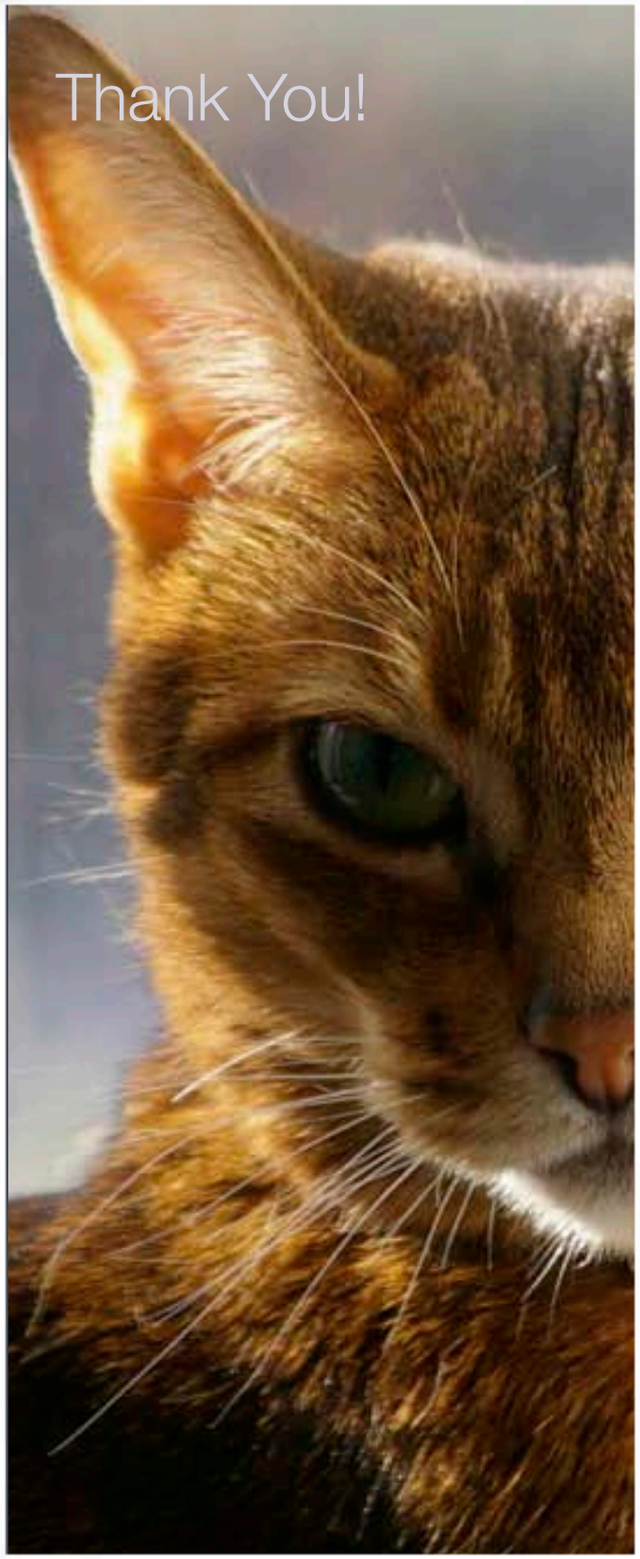
[ooRexx, NetRexx should enjoy a rapprochement]

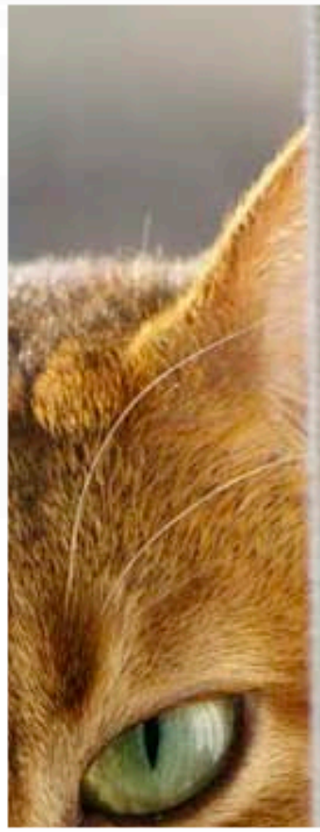
[Very happy with recent enhancements already - like **loop**]

[ooRexx should also run on Java VM]



Thank You!





Get In Touch

rvjansen@xs4all.nl

rene.vincent.jansen@gmail.com