NetRexx and Prolog

René Vincent Jansen, 2012 International Rexx Language Symposium Raleigh, NC.

May 15th, 2012

Why Prolog

What's a Prolog?

- Prolog is a general purpose logic programming language associated with artificial intelligence and computational linguistics.

- Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is declarative: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations.

- The language was first conceived by a group around Alain Colmerauer in Marseille, France, in the early 1970s and the first Prolog systems were developed in 1972 by Colmerauer with Philippe Roussel, in Algol-W and FORTRAN.

What happened to Prolog?

- It lost steam
- The Japanese 5th generation computer project (remember that) was its "OS/2"
- It was hyped by recent converts in the 5th generation project and bogged down in hardware Prolog machines that never ran well
- As a rule, progress in specialized hardware is generally overtaken by progress in general computer hardware
- Other declarative languages ate its lunch - SQL in the first place
- It is coming back though - IBM's Watson is purported to run on Java and Prolog.

Rationale

- In the course of writing applications, over the years, that handle meta models, models and instance data, the perspective on the strengths and weaknesses of a purely relational and post-relational implementation led to two insights:
- a large percentage of the code consisted of a series of logic operations and set-matching and reducing code that is already and more completely implemented in the Prolog language;
- some of the relationships in the data models are better stored as rules than as instances of those relationships.

Rules

- Rules prevent combinatorial explosions in the data store
- For example, if we have a simple model of an IT infrastructure with entity types as servers, lpars, queue managers, channels, queues and applications, we can model relationships between these entity types and store instances of these. applications, queues, channels, queue managers and server machines.
- If we store a queue as *belongs to* a queue manager, and and an application as *using* a specific queue belonging to that queue manager, without rules, *we also have to store the relationship between the queue manager and the application as a separate fact*, or capture this in some ad-hoc code.
- With Prolog, we can easily define a rule that determines that if an application uses a specific queue, it also uses a specific queue manager on a specific server. We define this rule by declaring it as data in the model, and not in (procedural) code.

Some examples (from Learn Prolog Now!)

- woman(mia).
- woman(jody).
- woman(yolanda).
- loves(vincent,mia).
  loves(marcellus,mia).
  loves(pumpkin,honey_bunny).
  loves(honey_bunny,pumpkin).
- woman(X).

- state facts
- woman: predicate
- names are called atoms
- another predicate + facts
- will state all the women's names

Ask another question

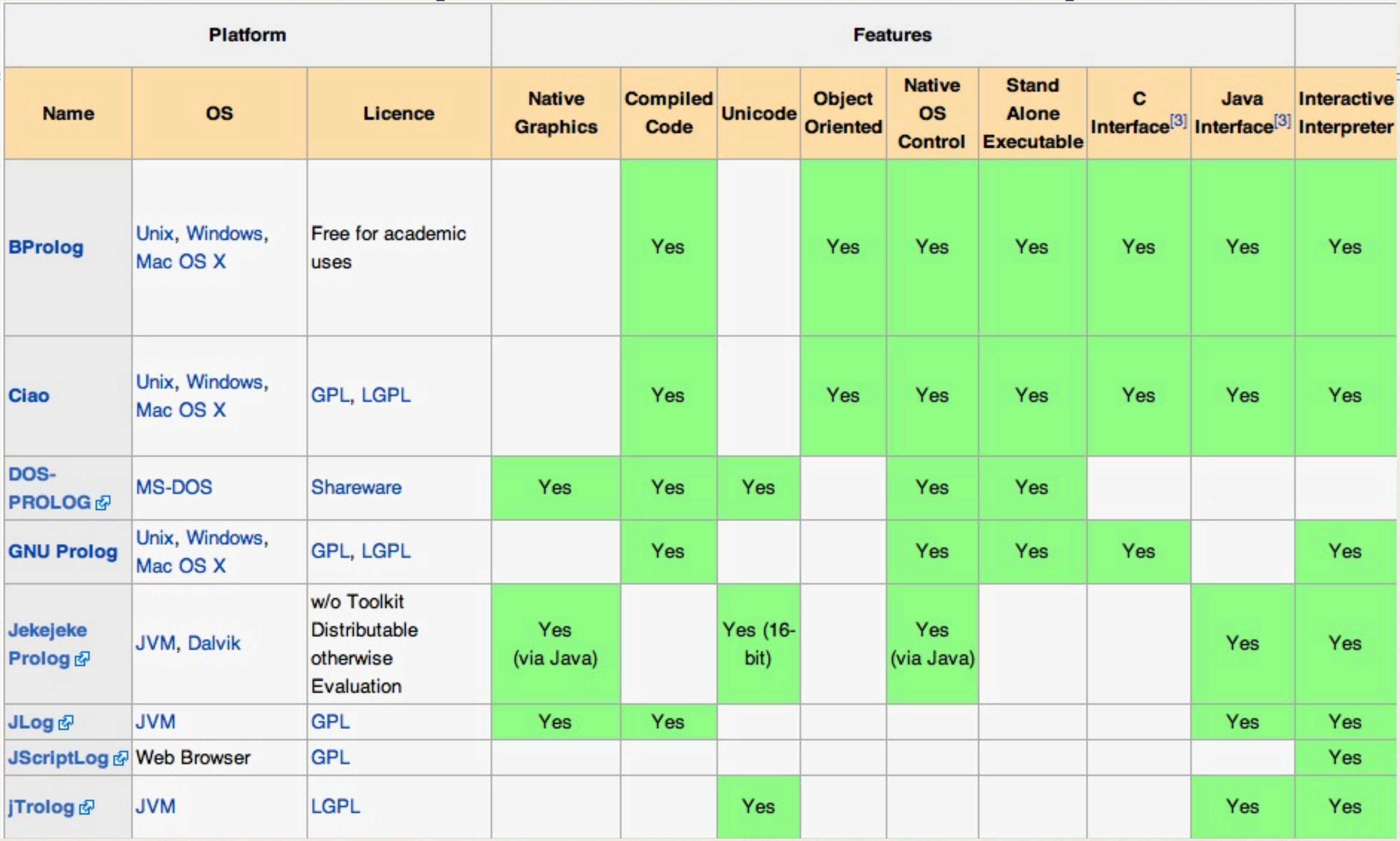
- loves(marcellus,X),woman(X).
- means:*is there any individual X such that Marcellus loves X and X is a woman?*
- the Comma (,) means *and*

A rule

- jealous(X,Y) :- loves(X,Z),loves(Y,Z).
- to be read as: *an individual X will be jealous of an individual Y if there is some individual Z that X loves, and Y loves that same individual Z too.*
- So the query:
    - jealous(marcellus,W).
- means: can you find an individual W such that Marcellus is jealous of W?
- The answer is: vincent.

Which Prolog

Good comparison in Wikipedia



Platform			Features								
Name	OS	Licence	Native Graphics	Compiled Code	Unicode	Object Oriented	Native OS Control	Stand Alone Executable	C Interface[3]	Java Interface[3]	Interactive Interpreter
BProlog	Unix, Windows, Mac OS X	Free for academic uses		Yes		Yes	Yes	Yes	Yes	Yes	Yes
Ciao	Unix, Windows, Mac OS X	GPL, LGPL		Yes		Yes	Yes	Yes	Yes	Yes	Yes
DOS-PROLOG	MS-DOS	Shareware	Yes	Yes	Yes		Yes	Yes			
GNU Prolog	Unix, Windows, Mac OS X	GPL, LGPL		Yes			Yes	Yes	Yes		Yes
Jekejeke Prolog	JVM, Dalvik	w/o Toolkit Distributable otherwise Evaluation	Yes (via Java)		Yes (16-bit)		Yes (via Java)			Yes	Yes
JLog	JVM	GPL	Yes	Yes						Yes	Yes
JScriptLog	Web Browser	GPL									Yes
jTrolog	JVM	LGPL			Yes					Yes	Yes

Started out with SWI Prolog

- Stable and free, source available. Written in C(++). From Amsterdam.
- JPL interface to JVM
- Implemented in native library (dll) per platform
- Works very well but consistent installation trouble and complaints from associates
- Decided to simplify the architecture by choosing a Prolog implementation in pure Java

Prolog implementations in Java

Free and Open Source

- Jlog
- tuProlog
- jTrolog

JLog

- Good performance reviews
- Source available; very C++ like implementation in Java; needs separate text based configuration file for predicates - hmm
- Can also interface using BSF
- Initial Theory load has a bad (exponential) performance bug
- Did not try to fix due to ugliness and hermetical quality of source code

tuProlog

- Italian implementation from University of Turin
- Source available and nice architecture
- Good documentation of Java interface
- Actively developed
- General performance qualm: not blindingly fast

jTrolog

- Performance oriented re-write of (parts of) tuProlog by Ivar Ørstavik
- Lightning fast load of initial Theory
- No real documentation, but:
- Source code very approachable
- http://java.net/projects/jtrolog
- not very active as a project, unfortunately

I chose jTrolog

- In spite of its strange name (Trollgatan, Norway?)
- For speed and modifiability
- Rebuilt own version of it
- Added (and sometimes took out) some NetRexx specific features

What did I add to jTrolog?

- Having the source available it was tempting to add some functionality to make life easier (most of this probably would have been possible without the source, but it made the implementation much more understandable)
- I am considering to translate the whole package to NetRexx for clarity and easy access - using Marc Reme's Nrx2Java
- Added an RMI based server for instant access to larger theories
- Added a REPL that does not need semicolons entered after every query
- (Added a server based SREPL - to query over the network)

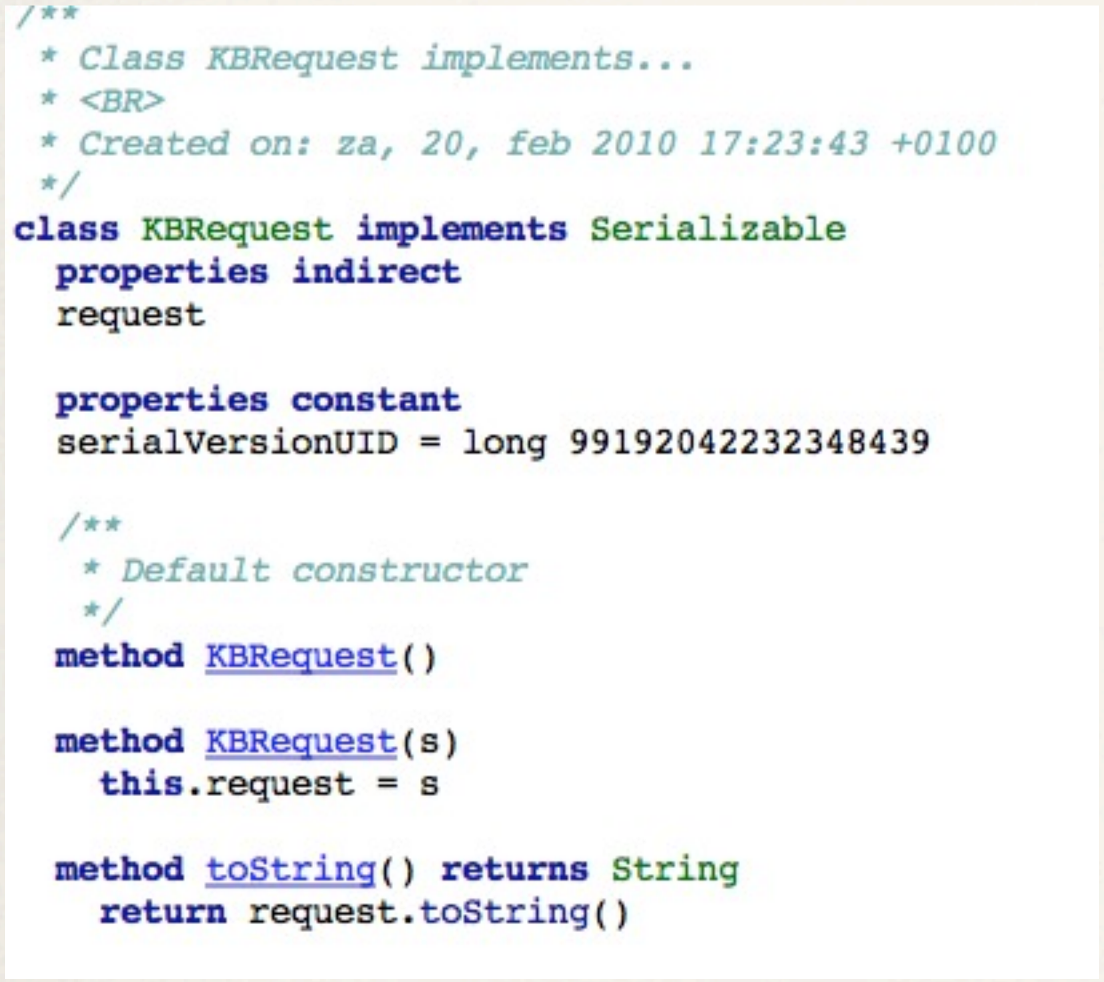
JLine to the rescue

- JVM programs under Unix shells lack one facility that Windows cmd.exe does have: up-arrow for command history
- This is absolutely critical when doing interactive, accumulative development
- I employed the excellent JLine library to accomplish this - just add a batch file to the java main class

RMI Server

- Loads the Prolog engine, reads the initial theory, waits for requests
- Optionally use ssl and user authentication
- Tried and true code
- Implemented Request and Response classes to marshal queries and output

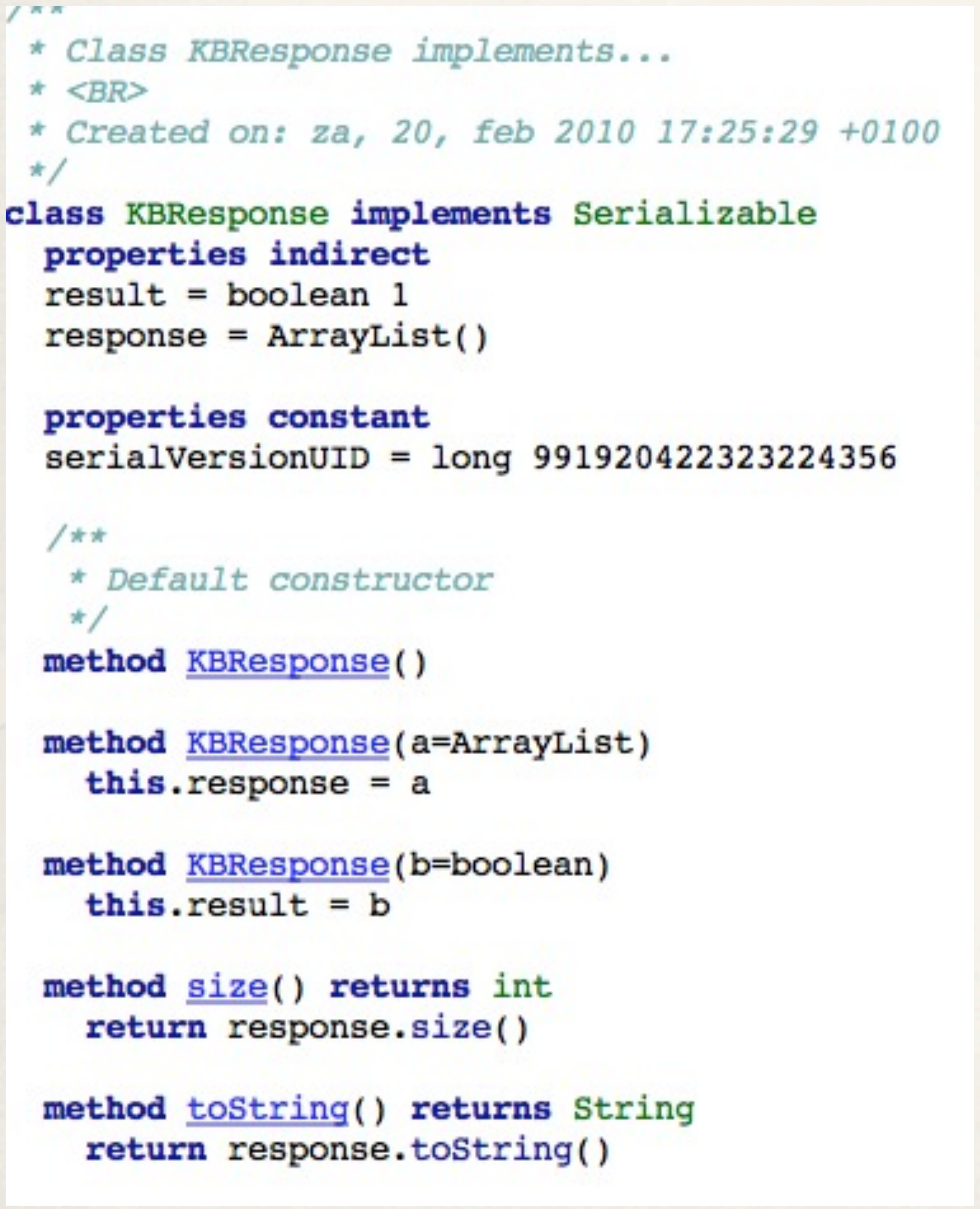
KBRequest



```
/**
 * Class KBRequest implements...
 * <BR>
 * Created on: za, 20, feb 2010 17:23:43 +0100
 */
class KBRequest implements Serializable
  properties indirect
  request

  properties constant
  serialVersionUID = long 99192042232348439

  /**
   * Default constructor
   */
  method KBRequest()

  method KBRequest(s)
    this.request = s

  method toString() returns String
    return request.toString()
```

Make sure a request is of type Rexx so it easily can be massaged

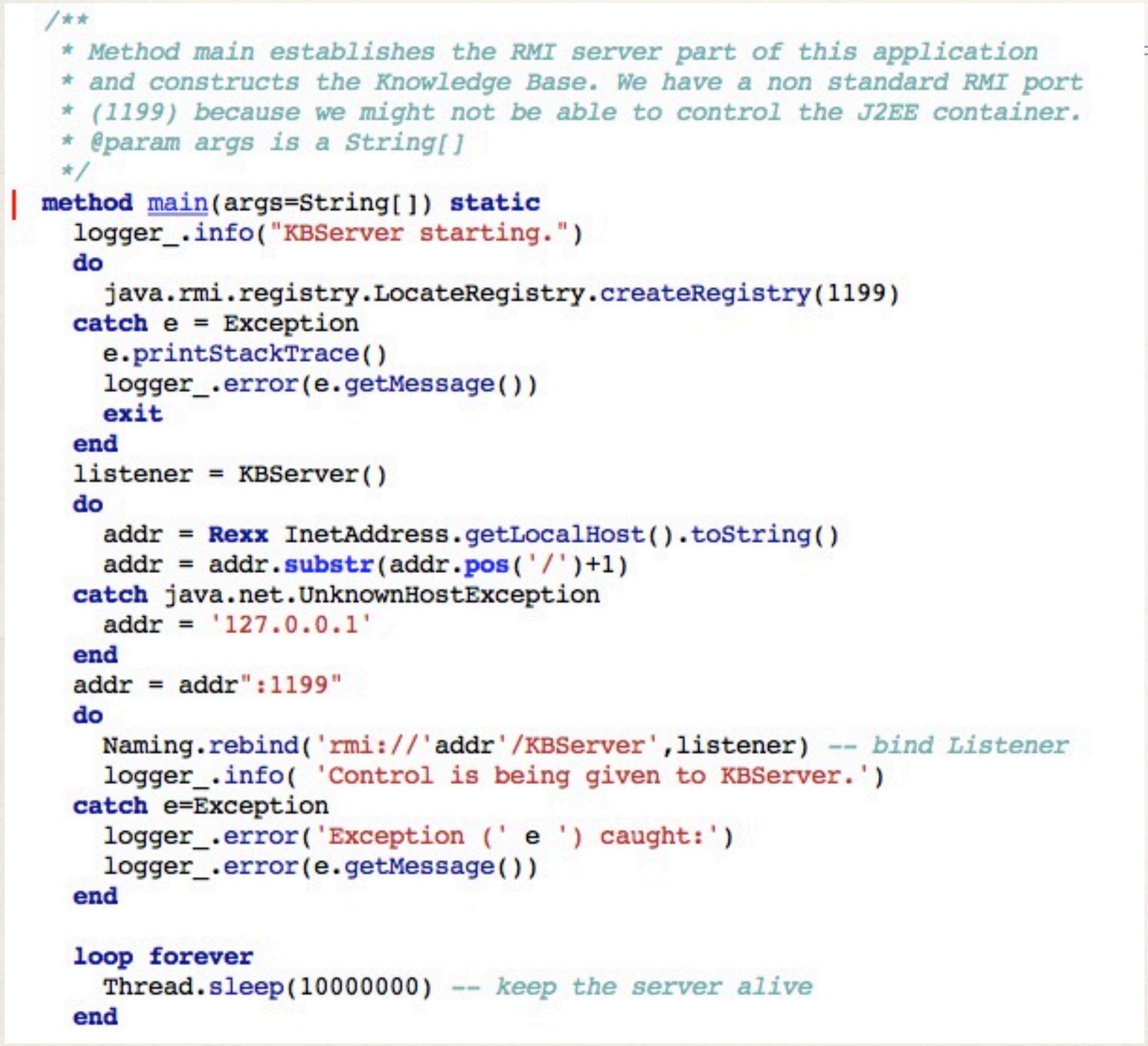
KBResponse



```
/**
 * Class KBResponse implements...
 * <BR>
 * Created on: za, 20, feb 2010 17:25:29 +0100
 */
class KBResponse implements Serializable
  properties indirect
  result = boolean 1
  response = ArrayList()

  properties constant
  serialVersionUID = long 991920422323224356

  /**
   * Default constructor
   */
  method KBResponse()

  method KBResponse(a=ArrayList)
    this.response = a

  method KBResponse(b=boolean)
    this.result = b

  method size() returns int
    return response.size()

  method toString() returns String
    return response.toString()
```

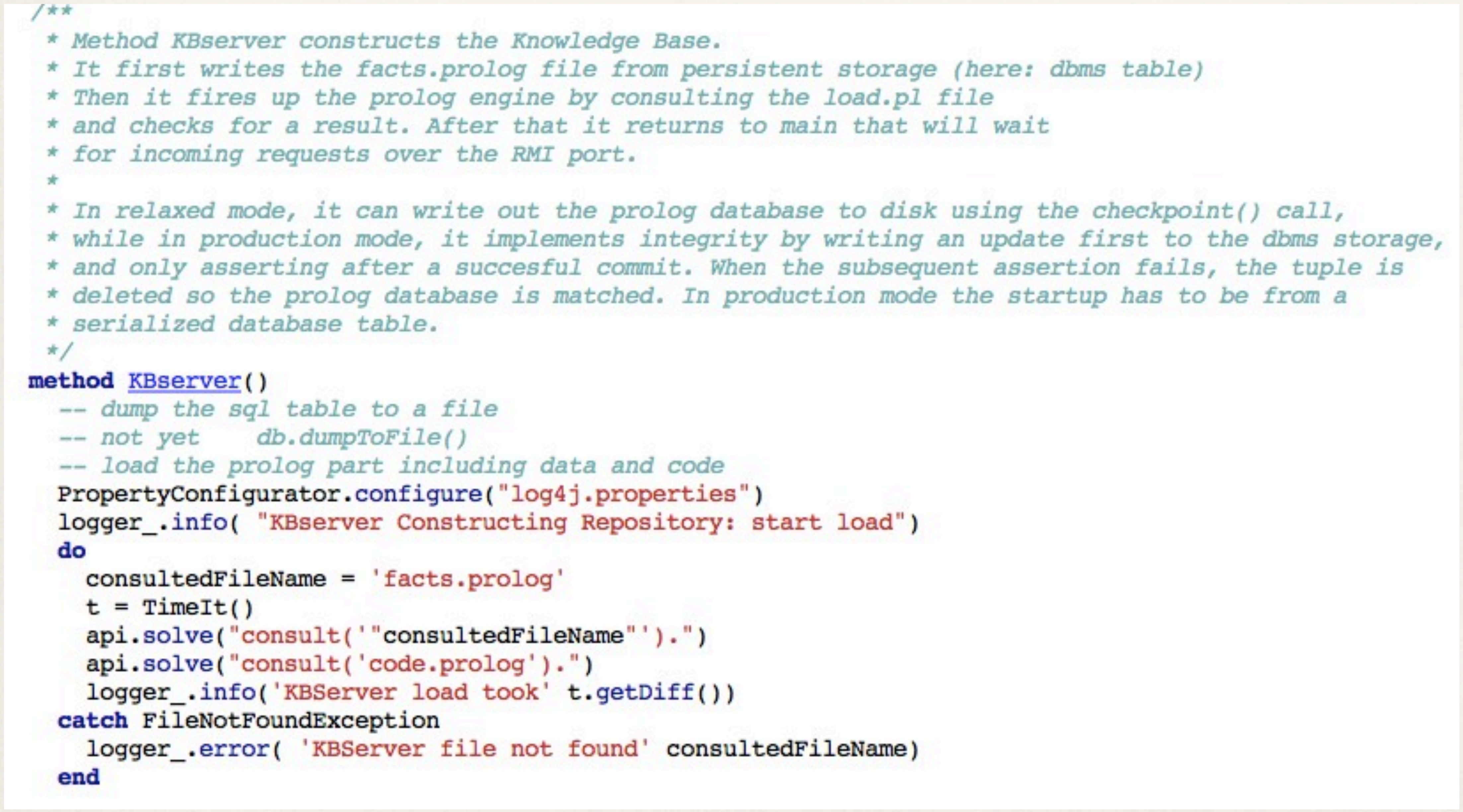
A response needs to be a list of Map

KBServer



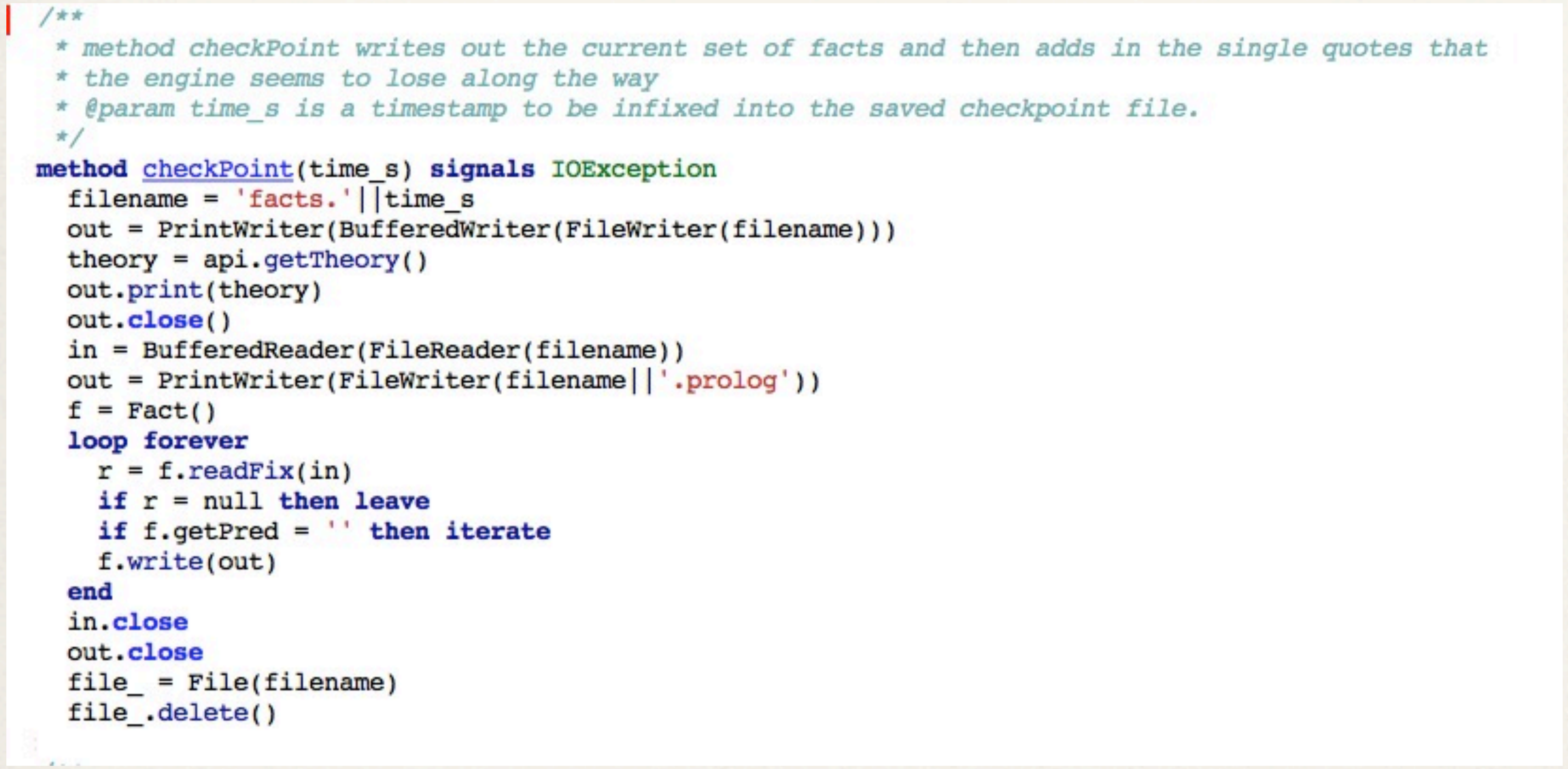
```
  /**
   * Method main establishes the RMI server part of this application
   * and constructs the Knowledge Base. We have a non standard RMI port
   * (1199) because we might not be able to control the J2EE container.
   * @param args is a String[]
   */
method main(args=String[]) static
  logger_.info("KBServer starting.")
  do
    java.rmi.registry.LocateRegistry.createRegistry(1199)
  catch e = Exception
    e.printStackTrace()
    logger_.error(e.getMessage())
    exit
  end
  listener = KBServer()
  do
    addr = Rexx InetAddress.getLocalHost().toString()
    addr = addr.substr(addr.pos('/')+1)
  catch java.net.UnknownHostException
    addr = '127.0.0.1'
  end
  addr = addr":1199"
  do
    Naming.rebind('rmi://'addr'/KBServer',listener) -- bind Listener
    logger_.info( 'Control is being given to KBServer.')
  catch e=Exception
    logger_.error('Exception (' e ') caught:')
    logger_.error(e.getMessage())
  end

  loop forever
    Thread.sleep(10000000) -- keep the server alive
  end
```

KBServer: load Theory in ctor



```
/**
 * Method KBserver constructs the Knowledge Base.
 * It first writes the facts.prolog file from persistent storage (here: dbms table)
 * Then it fires up the prolog engine by consulting the load.pl file
 * and checks for a result. After that it returns to main that will wait
 * for incoming requests over the RMI port.
 *
 * In relaxed mode, it can write out the prolog database to disk using the checkpoint() call,
 * while in production mode, it implements integrity by writing an update first to the dbms storage,
 * and only asserting after a succesful commit. When the subsequent assertion fails, the tuple is
 * deleted so the prolog database is matched. In production mode the startup has to be from a
 * serialized database table.
 */
method KBserver()
  -- dump the sql table to a file
  -- not yet    db.dumpToFile()
  -- load the prolog part including data and code
  PropertyConfigurator.configure("log4j.properties")
  logger_.info( "KBserver Constructing Repository: start load")
  do
    consultedFileName = 'facts.prolog'
    t = TimeIt()
    api.solve("consult('"consultedFileName"').")
    api.solve("consult('code.prolog').")
    logger_.info('KBServer load took' t.getDiff())
  catch FileNotFoundException
    logger_.error( 'KBServer file not found' consultedFileName)
  end
```

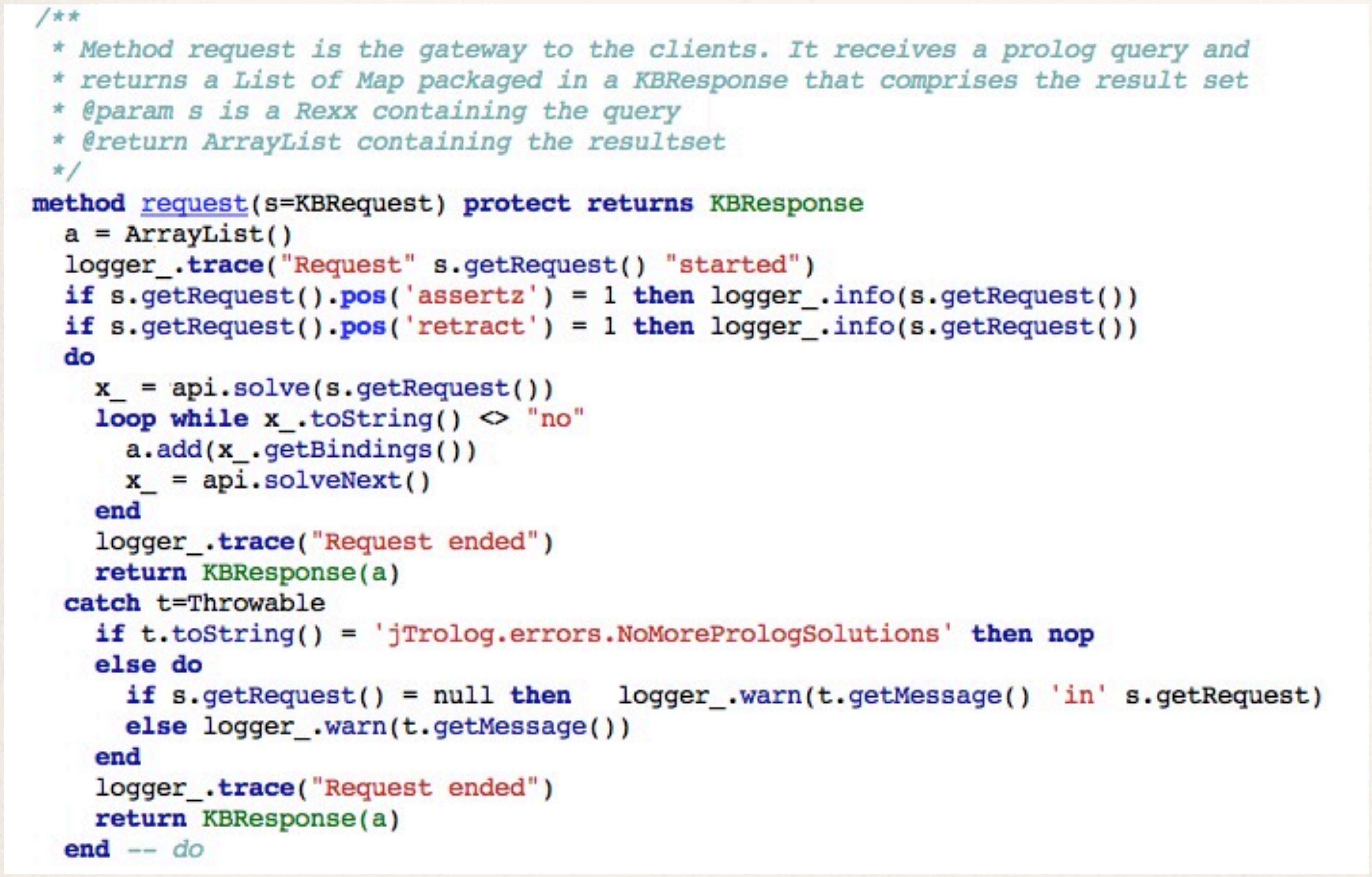
Modified the way jTrolog writes



```
/**
 * method checkPoint writes out the current set of facts and then adds in the single quotes that
 * the engine seems to lose along the way
 * @param time_s is a timestamp to be infixed into the saved checkpoint file.
 */
method checkPoint(time_s) signals IOException
  filename = 'facts.'||time_s
  out = PrintWriter(BufferedWriter(FileWriter(filename)))
  theory = api.getTheory()
  out.print(theory)
  out.close()
  in = BufferedReader(FileReader(filename))
  out = PrintWriter(FileWriter(filename||'.prolog'))
  f = Fact()
  loop forever
    r = f.readFix(in)
    if r = null then leave
    if f.getPred = '' then iterate
    f.write(out)
  end
  in.close
  out.close
  file_ = File(filename)
  file_.delete()
```

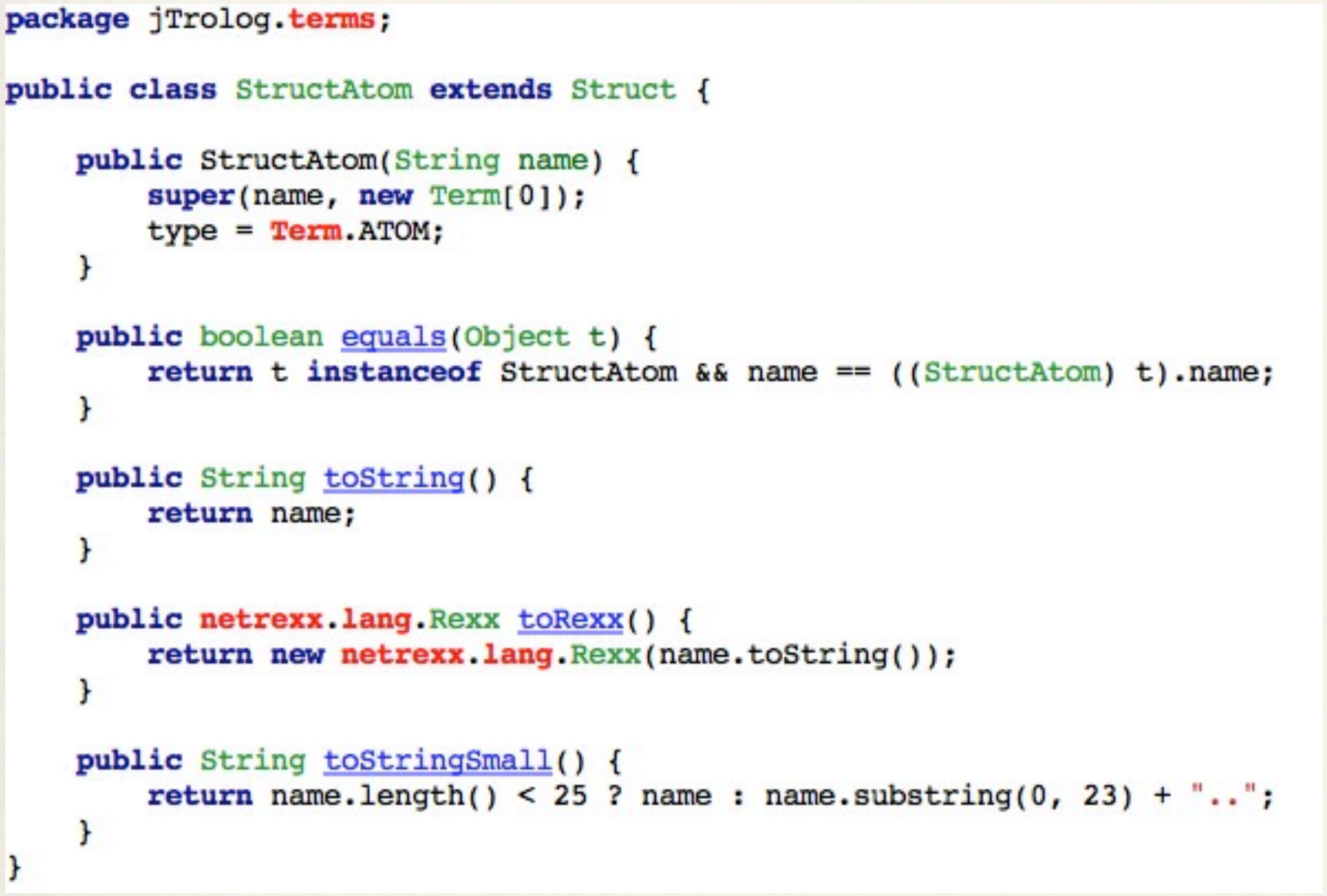
How to interface

Solving a query



```
/**
 * Method request is the gateway to the clients. It receives a prolog query and
 * returns a List of Map packaged in a KBResponse that comprises the result set
 * @param s is a Rexx containing the query
 * @return ArrayList containing the resultset
 */
method request(s=KBRequest) protect returns KBResponse
  a = ArrayList()
  logger_.trace("Request" s.getRequest() "started")
  if s.getRequest().pos('assertz') = 1 then logger_.info(s.getRequest())
  if s.getRequest().pos('retract') = 1 then logger_.info(s.getRequest())
  do
    x_ = api.solve(s.getRequest())
    loop while x_.toString() <> "no"
      a.add(x_.getBindings())
      x_ = api.solveNext()
    end
    logger_.trace("Request ended")
    return KBResponse(a)
  catch t=Throwable
    if t.toString() = 'jTrolog.errors.NoMorePrologSolutions' then nop
    else do
      if s.getRequest() = null then   logger_.warn(t.getMessage() 'in' s.getRequest)
      else logger_.warn(t.getMessage())
    end
    logger_.trace("Request ended")
    return KBResponse(a)
  end -- do
```

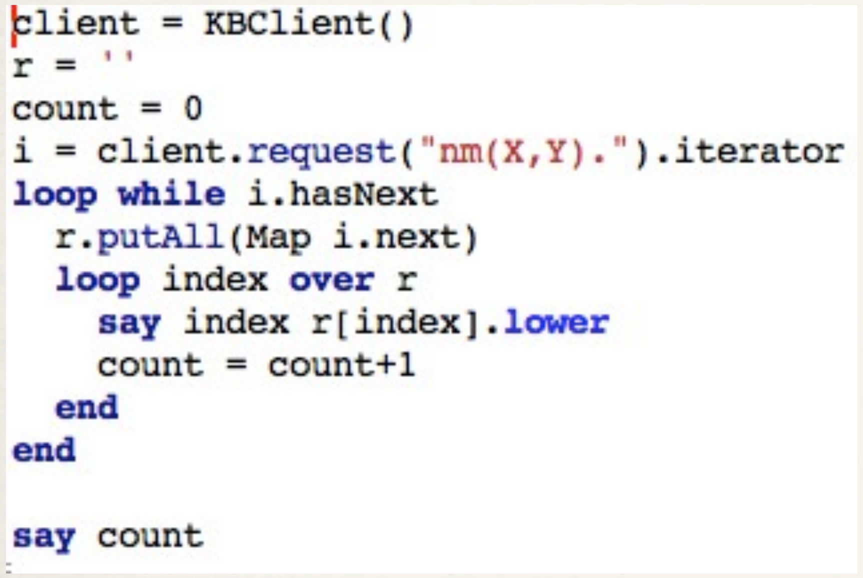
First added a toRexx on StructAtom



```
package jTrolog.terms;

public class StructAtom extends Struct {

    public StructAtom(String name) {
        super(name, new Term[0]);
        type = Term.ATOM;
    }

    public boolean equals(Object t) {
        return t instanceof StructAtom && name == ((StructAtom) t).name;
    }

    public String toString() {
        return name;
    }

    public netrexx.lang.Rexx toRexx() {
        return new netrexx.lang.Rexx(name.toString());
    }

    public String toStringSmall() {
        return name.length() < 25 ? name : name.substring(0, 23) + "..";
    }
}
```

But later decided that full collection class support in NetRexx with automatic conversion to Rexx would be more desirable

Using ∝-level NetRexx collection class support



```
client = KBClient()
r = ''
count = 0
i = client.request("nm(X,Y).").iterator
loop while i.hasNext
  r.putAll(Map i.next)
  loop index over r
    say index r[index].lower
    count = count+1
  end
end

say count
```

put all the values of a Map to an indexed Rexx string and loop over its contents

this saves you from the drudge work of repeating all the map keys

The can be even shorter when all collection classes are supported

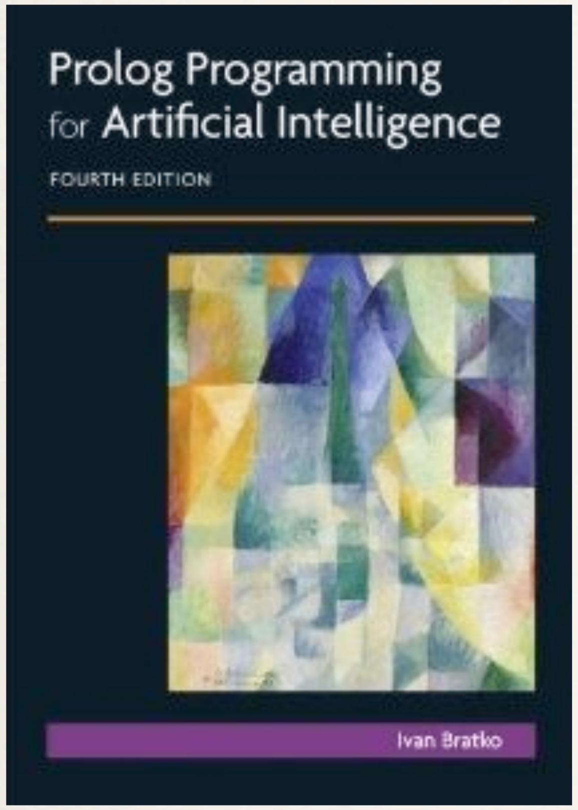
Added OO meta model constructs to the Prolog theory

- Added predicates for object, classifier, schema, subtypes, domains so post-relational modeling is supported
- Honey, I shrunk the code base!
- A huge negative KLOC score achieved over the last decade - must have lost 50 over a number of years. Partly because source is not generated for concrete NetRexx classes any more.
- Still looking to generate these on-the-fly using the interpreter

Short Demonstration

- Start the KBServer
- treeTest: list the model hierarchy
- connect with srepl
    - list names: nm(X,Y).
    - list Queue Managers and Lpars

Literature



Ivan Bratko, *Prolog Programming for Artificial Intelligence*

Patrick Blackburn et al, *Learn Prolog Now!*

Thank you for your attention

Q? rvjansen@xs4all.nl