RLisp: The Program

1 Introduction

RLisp is a Java packet that implements a Lisp interpreter. Lisp provides the syntax while Java provides the semantics.

RLisp can be useful to debug Java classes. It also provides a GUI interface to locate the classes and to choose the constructors and methods.

RLisp comprises the following classes:

- RPair: implements a Lisp pair. A Lisp pair has two parts: the car and the cdr. The car is the first word, and the cdr is the last. In a proper list, the cdr is another proper list, or an empty pair, nil, signalling the end.
- RFrame: implements a frame. A frame is a diccionary, that is, is a proper list of key-value pairs.
- REnvironment: implements an environment. An environment is a proper list of frames.
- RLisp: implements a minimum List evaluator.
- RLispJava: extends the former providing means to use Java objects.
- RLispInterpreter: is the Lisp Interpreter.
- RLispConsole: implements the main console, which is an output only console.
- RButton: extends class javax.swing.JButton with an object that the listener can retrieve.
- RKeyboard: to input lines from the keyboard counting parenthesis.
- RClassTree: shows the classes that are accesible from a directory or jar file, and allows to choose a field, a constructor, or a method.
- RExtFilter: extends class ExtFilter to select the files with a specified extension.
- RClassLoader: extends class URLClassLoader to make it incremental.

In addition, file RLisp.log can be read to load easily useful definitions.

There are some additional Lisp functions and special forms that are not coded in Java, but in Lisp itself. These definitions are in file RLisp.lisp that calls RLispJava.lisp, RLispArray.lisp and RLispMaths.lisp. Other useful example is file Primes.lisp.

And file RLisp2jar.bat creates the jar file with the whole packet.

2 Files

2.1 File: RPair.java

```
/**
```

Class: `RPair`

Implements a Lisp pair.

A Lisp pair has two parts: the `car` and the `cdr`. The `car` is the first word, and the `cdr` is the last.

In a proper list, the `cdr` is another proper list, or an empty pair, `nil`, signalling the end.

A Lisp sentence is a proper list. A sentence is a list of words or frases separated by spaces and enclosed by parentheses. A frase is just a sentence inside a sentence.

I use "`,`" instead of "`.`" for improper lists. Thus, `(1 , 2)` is a pair; `1` is the `car`, and `2` is the `cdr`. In proper lists, as `(1 2)`, `1` is the `car`, and `(2)` is the `cdr`. And in `(2)` the `car` is `2` and the `cdr` is `()`. So `(1 2)` is the same as `(1 , (2 , ()))`.

`@author` © Ramón Casares 2003

`@version` 2003.02.05

```
*/
package RLisp;

import java.util.StringTokenizer;
import java.util.Vector;

public class RPair {

 /**
```

Variable: `car`

```
*/
 public Object car;
 /**
```

Variable: `cdr`

```
*/
 public Object cdr;

 /**
```

Constructor: `RPair()` produces a `()` list

```
*/
 public RPair() { car = null; cdr = null; }

 /**
```

Variable: `nil` is the `()` list

```
*/
 public static final RPair nil = new RPair();

 /**
```

Method: `isNil()` checks if this `RPair` is the `nil` list

```
*/
 public boolean isNil() { return( car == null && cdr == null); }

 /**
```

Method: `isNil(Object)` checks if `o` is the `nil` list

```
*/
 public static boolean isNil(Object o) {
  if ( o instanceof RPair )
   return( ((RPair)o).car == null && ((RPair)o).cdr == null );
  else return(false);
 }


 /**
```

Constructor: `RPair(Object, Object)`

```
*/
 public RPair(Object car, Object cdr) {
  this.car = car; this.cdr = cdr;
 }

 /**
```

Constructor: `RPair(String)`

```
*/
 public RPair(String s) {
  Object o = tokenize(s);
  if ( isRPair(o) ) {
   this.car = ((RPair)o).car;
   this.cdr = ((RPair)o).cdr;
  } else {
   this.car = "ERROR!";
   this.cdr = o;
  }
 }

 /**
```

Variable: `delims`

```
*/
 static final String delims = "( ';,\n\r\t_)";

 /**
```

Method: `Tokenize(String)`

It exhausts the `String` returning an `array` with the objects.

```
*/
 public static Object[] Tokenize(String s) {
  Vector<Object> V = new Vector<Object>();
  StringTokenizer T = new StringTokenizer(s,delims,true);
  while( T.hasMoreTokens() ) V.add(tokenize(T, null));
  return(V.toArray());
 }

 /**
```

Method: `tokenize(String)`

It returns one `Object` only. It uses three `private` methods.

```
*/
public static Object tokenize(String s) {
 StringTokenizer T = new StringTokenizer(s,delims,true);
 return(tokenize(T, null));
}
private static Object tokenize(StringTokenizer T, RPair v) {
 String t;
 while ( T.hasMoreTokens() ) { t = T.nextToken();
  if ( " ".equals(t) || "\n".equals(t) || "_".equals(t) ||
      "\r".equals(t) || "\t".equals(t) ) {}
  else if ( "(".equals(t) ) {
   if (v==null) return(tokenize(T, new RPair()));
   else          v.add(tokenize(T, new RPair()));
  }
  else if ( ")".equals(t) ) { return(v); }
  else if ( ",".equals(t) ) { v.add(tokenize(T,null), false); }
  else if ( ";".equals(t) ) {
   while( T.hasMoreTokens() ) { t = T.nextToken();
    if ( "\n".equals(t) || "\r".equals(t) ) break;
   }
  }
  else if ( "\'".equals(t) ) {
   Object arg = tokenize(T,null);
   if (v==null) return(new RPair("quote",new RPair(arg,null)));
   else          v.add(new RPair("quote",new RPair(arg,null)));
  }
  else if (v==null) return(t); else v.add(t);
 }
 return(v);
}
private Object add(Object last){
 if (last == null) return(null);
 else if (isNil()) { car = last; return(last); }
 else if (cdr == null) { cdr = new RPair(last,null); return(last); }
 else if (cdr instanceof RPair) return( ((RPair)cdr).add(last) );
 else return(null);
}
private Object add(Object last, boolean properly){
 if (properly) return(add(last));
 else if (isNil(last)) return(last);
 else if (cdr == null) { cdr = last; return(last); }
 else if (cdr instanceof RPair) return( ((RPair)cdr).add(last,false) );
 else return(null);
}

/**
```

Method: `isRPair(Object)`

```
*/
 public static boolean isRPair(Object o) {
  return(o instanceof RPair);
 }

 /**
```

Method: `isAtom(Object)`

```
*/
 public static boolean isAtom(Object o) {
  return( isNil(o) || !(o instanceof RPair) );
 }

 /**
```

Method: `isList(Object)`

If the `Object o` is a proper List, it returns the length of the List. Otherwise it returns -1.

```
*/
 public static int isList(Object o) {
  if(o == null) return(-1);
  if ( o instanceof RPair ) {
   RPair p = (RPair)o;
   if ( p.car == null )
    if ( p.cdr == null ) return(0); else return(-1);
   else
    if ( p.cdr == null ) return(1); else {
     int l = isList(p.cdr);
     if (l<0) return(l); else return(1+l);
    }
  } else return(-1);
 }


 /**
```

Method: `toArray()`

```
*/
 public Object[] toArray() {
  int l = isList(this);
  if ( l >= 0 ) {
   Object[] a = new Object[l];
   for(int i=0; i<l; i++) a[i] = nth(i);
   return(a);
  } else return(null);
 }

 /**
```

Method: `toArray(Object)`

```
*/
 public static Object[] toArray(Object o) {
  int l = isList(o);
  if ( l >= 0 ) {
   Object[] a = new Object[l];
   for(int i=0; i<l; i++) a[i] = nth(o,i);
   return(a);
  } else return(null);
 }


 /**
```

Method: `cons(Object, RPair)`

```
*/
 public static RPair cons(Object car, RPair cdr) {
  if ( isNil(cdr) ) return( new RPair(car,null) );
  else              return( new RPair(car,cdr) );
 }

 /**
```

Method: `car()`

```
*/
 public Object car() { return( car ); }
 /**
```

Method: `cdr()`

```
*/
 public Object cdr() { return( cdr ); }

 /**
```

Method: `Cdr()`

```
*/
 public Object Cdr() {
  if (cdr == null) return(nil); else return(cdr);
 }

 /**
```

Method: `CDR()`

```
*/
 public RPair CDR() {
  if (cdr == null) return(nil);
  if (cdr instanceof RPair) return( ((RPair)cdr) );
  else return(null);
 }

 /**
```

Method: `nth(int)`

`@param` `n` is the position (car is 0)

`@return` the object in the `n` position

```
*/
```

```
 public Object nth(int n) {
  if (n < 0) return(null);
  else if (n == 0) return(car);
  else if (cdr instanceof RPair) return( ((RPair)cdr).nth(n-1) );
  else return(null);
 }

 /**
```

Method: nth(Object, int)

```
*/
 public static Object nth(Object o, int n) {
  if ( o == null || n < 0 ) return(null);
  if (o instanceof RPair)
   if (n == 0) return(((RPair)o).car);
   else return( nth(((RPair)o).cdr,n-1) );
  else return(null);
 }

 /**
```

Method: substitute(Object, Object)

```
*/
 public RPair substitute(Object oldo, Object newo) {
  Object newcar = null;
  if (car == null) { if (oldo == null) newcar = newo; }
  else {
   if ( car.equals(oldo) ) newcar = newo;
   else if (car instanceof RPair)
    newcar = ((RPair)car).substitute(oldo,newo);
   else newcar = car;
  }
  Object newcdr = null;
  if (cdr == null) { if (oldo == null) newcdr = newo; }
  else {
   if ( cdr.equals(oldo) ) newcdr = newo;
   else if (cdr instanceof RPair)
    newcdr = ((RPair)cdr).substitute(oldo,newo);
   else newcdr = cdr;
  }
  return(new RPair(newcar,newcdr));
 }

 private boolean loop = false;

 /**
```

Method: toString()

```
*/
 public String toString() {
  if (loop) return("RPair"+hashCode());
  else {
   loop = true;
   String s = "(" + toStringWOP() + ")";
   loop = false;
   return(s);
  }
 }
 private String toStringWOP() {
  String s;
  if ( car == null ) s = "";
  else               s = car.toString();
  if ( cdr == null ) return(s);
  else if (cdr instanceof RPair) return(s+" "+((RPair)cdr).toStringWOP());
  else return( s + " , " + cdr.toString() );
 }

 /**
```

Method: toString(boolean)

```
*/
 public String toString(boolean wp) {
  if (wp) return(toString());
  else return(toStringWOP());
 }

 /**
```

Method: equals(Object)

```
*/
 public boolean equals(Object o) {
  if ( o == null )  return( this.car == null && this.cdr == null );
  else if (o instanceof RPair) {
   RPair p = (RPair)o;
   return ( (  p.car == this.car ||
              ( p.car != null && p.car.equals(this.car)) ) &&
             (  p.cdr == this.cdr ||
              ( p.cdr != null && p.cdr.equals(this.cdr)) ) ); }
  else return(false);
 }

}
```

2.2 File: RFrame.java

```
/**
```

Class: RFrame

Implements a frame.

A frame is a diccionary, that is, is a list of key-value pairs, and a pair is a list with two items.

Following “Structure and Interpretation of Computer Programs”, page 308.

@author © Ramón Casares 2003

@version 2003.03.02

```
*/
package RLisp;
```

```

import java.util.Hashtable;
import java.util.Enumeration;

public class RFrame {

 private Hashtable<Object,Object> ht;

 /**
```

Constructor: `RFrame()`

```
*/
 public RFrame() { ht = new Hashtable<Object,Object>(); }

 /**
```

Constructor: `RFrame(Object, Object)`

```
*/
 public RFrame(Object keys, Object values) {
  ht = new Hashtable<Object,Object>(); bind(keys,values);
 }

 /**
```

Method: `bind(Object, Object)`

```
*/
 public Object bind(Object key, Object value) {
  if ( key == null ) return(null);
  else if ( RPair.isRPair(key) ) {
   if ( RPair.isRPair(value) ) {
    bind(((RPair)key).car(),((RPair)value).car());
    bind(((RPair)key).cdr(),((RPair)value).Cdr()); // note: Cdr
   } else return(null);
  } else { if ( value != null) ht.put(key,value); else ht.remove(key); }
  return(lookup(key));
 }

 /**
```

Method: `lookup(Object)`

```
*/
 public Object lookup(Object key) {
  if ( key == null ) return(null); else return(ht.get(key));
 }

 /**
```

Method: `keys()`

```
*/
 public Object[] keys() {
  Object[] ka = new Object[ ht.size() ];
  Enumeration ke = ht.keys();
  int i = 0;
  while ( ke.hasMoreElements() ) ka[i++] = ke.nextElement();
  return( ka );
 }

 private boolean loop = false;

 /**
```

Method: `toString()`

```
*/
 public String toString() {
  if ( loop ) return( "RFrame"+hashCode() );
  else {
   loop = true;
   String sr = ht.toString();
   loop = false;
   return(sr);
  }
 }

}
```

2.3 File: REnvironment.java

```
/**
```

Class: `REnvironment`

Implements an environment.

An environment is a list of frames. Each frame is a diccionary, that is, is a list of key-value pairs, and a pair is a list with two items.

Following “Structure and Interpretation of Computer Programs”, page 306.

`@author` © Ramón Casares 2003

`@version` 2003.03.02

```
*/
package RLisp;

public class REnvironment extends RPair {

 /**
```

Constructor: `REnvironment()`

```
*/
 public REnvironment() { super(); }

 /**
```

Constructor: `REnvironment(RFrame)`

```
*/
 public REnvironment(RFrame rf) { super(rf,null); }

 /**
```

Constructor: `REnvironment(RFrame, REnvironment)`

```
*/
 public REnvironment(RFrame rf, REnvironment re) { super(rf,re); }

 /**
```

Method: `extend(RFrame)`

```
*/
 public REnvironment extend(RFrame rf) {
  return(new REnvironment(rf,this));
 }

 /**
```

Method: `firstFrame()`

```
*/
 public RFrame firstFrame() { return( (RFrame)car() ); }
 /**
```

Method: `restFrames()`

```
*/
 public REnvironment restFrames() { return( (REnvironment)cdr() ); }

 /**
```

Method: `lookup(Object)`

```
*/
 public Object lookup(Object key) {
  if ( this.isNil() ) return(null);
  Object val = firstFrame().lookup(key);
  if ( val != null ) return(val);
  else {
   REnvironment rest = restFrames();
   if ( rest == null ) return(null);
   else return( rest.lookup(key) );
  }
 }

 /**
```

Method: `define(Object, Object)`

```
*/
 public Object define(Object key, Object value) {
  if ( this.isNil() ) return(null);
  else {
   firstFrame().bind(key,value);
   return(key);
  }
 }

 /**
```

Method: `set(Object, Object)`

```
*/
```

```
 public Object set(Object key, Object value) {
  if ( this.isNil() ) return(null);
  else {
   RFrame f = firstFrame();
   Object val = f.lookup(key);
   if ( val != null ) { f.bind(key,value); return(value); }
   else if ( restFrames() == null ) { return(null); }
   else return( restFrames().set(key,value) );
  }
 }


 private boolean loop = false;

 /**
```

Method: `toString(boolean)`

```
*/
 public String toString(boolean all) {
  if (all) return( toString() );
  else return("ENV"+hashCode()); }

 /**
```

Method: `toString()`

```
*/
 public String toString() {
  REnvironment rest = restFrames();
  if ( rest == null ) { return("GlobalENV"); }
  else if (loop) { return("ENV"+ hashCode()); }
  else {
   loop = true;
   String s = "(ENV"+ hashCode() + ": " + firstFrame().toString() +
              "->" + rest.toString() + ")";
   loop = false;
   return(s);
  }
 }

 /**
```

Method: `list()`

```
*/
 public String[] list() {
  RFrame first = firstFrame();
  if (first == null) return(null);
  Object[] ka = first.keys();
  int l = ka.length;
  String[] sa = new String[ l ];
  for(int i=0; i<l; i++)
   sa[i] = ka[i].toString() + "=" + first.lookup(ka[i]).toString();
  return(sa);
 }


 /**
```

Method: `keys()`

```
*/
 public Object[] keys() {
  RFrame first = firstFrame();
  if (first == null) return(null); else return(first.keys());
 }

 /**
```

Method: `keys(Class, boolean)`

```
*/
 public Object[] keys(Class c, boolean strict) {
  RFrame first = firstFrame();
  if (first == null || c == null) return(null);
  else if ( c == Boolean.TYPE ) c = Boolean.class;
  else if ( c == Character.TYPE ) c = Character.class;
  else if ( c == Byte.TYPE ) c = Byte.class;
  else if ( c == Short.TYPE ) c = Short.class;
  else if ( c == Integer.TYPE ) c = Integer.class;
  else if ( c == Long.TYPE ) c = Long.class;
  else if ( c == Float.TYPE ) c = Float.class;
  else if ( c == Double.TYPE ) c = Double.class;
  Object[] names = first.keys();
  java.util.Vector<Object> sel = new java.util.Vector<Object>();
  Object n; Object o;
  for(int i=0; i<names.length; i++) {
   n = names[i];
   o = first.lookup(n);
   if (strict) { if ( o.getClass() == c) sel.addElement(n); }
   else        { if ( c.isInstance(o) )  sel.addElement(n); }
  }
  return(sel.toArray());
 }


}
```

2.4 File: RLisp.java

```
/**
```

Class: `RLisp`

Implements a minimum Lisp evaluator.

`@author` © Ramón Casares 2003

`@version` 2003.02.05

```
*/
package RLisp;

public class RLisp {

 /**
```

Variable: `t` is the `true` constant

```
*/
 public static final Boolean t = new Boolean("true");
 /**
```

Method: isTrue(Object)

```
*/
 public static boolean isTrue(Object o) {
  return( o != null && o instanceof Boolean &&
          ((Boolean)o).booleanValue() );
 }

 /**
```

Variable: counter counts eval cycles

```
*/
 public int counter = 0;

 /**
```

Method: eval(Object, REnvironment)

The main evaluator.

It evaluates:

- key to its value as defined in env.
- any other no RPair object to itself.
- (quote anything) → anything.
- (eval expression [env]) evaluates expression and then evaluates the result of the evaluation.
- (def key value) binds key to value in env.
- (set! key value) changes the binding of key to value.
- (cond (b0 e01 e02 ...) (b1 e11 e12 ...) ...) → if b0 to e10 e02 ..., else if b1 to e11 e12 ..., ...
- (lambda (f0 f1 ...) e0 e1 e2 ...) → the function.
- (rho name expander) → the special form.
- (function a0 a1 ...) applies the function using arguments a0 a1 ....

```
*/
 public Object eval(Object exp, REnvironment env) { counter++;
  if ( exp == null ) return(null);
  if ( RPair.isAtom(exp) ) {
   Object v = env.lookup(exp);
   if ( v == null ) return(exp);  // self-evaluating
   else             return(v);    // identifier
  } else { RPair re = (RPair)exp;
   if ( isSpecial(re) ) return( evalSpecial(re,env) ); // hook to extend
   else if ( "quote".equals(re.car()) )  return( re.CDR().car() );
   else if ( "eval".equals(re.car()) )   return( evalEval(re,env) );
   else if ( "set!".equals(re.car()) )   return( evalSet(re,env) );
   else if ( "def".equals(re.car()) )    return( evalDef(re,env) );
   else if ( "cond".equals(re.car()) )   return( evalCond(re,env) );
   else if ( "lambda".equals(re.car()) ) return( evalLambda(re,env) );
   else if ( "rho".equals(re.car()) )    return( evalRho(re,env) );
   else return( apply(re,env) );  // function application
  }
 }

 /**
```

Method: isSpecial(RPair): Override to define new special forms (as and)

```
 */
 boolean isSpecial(RPair p) { return(false); }

 /**
```

Method: evalSpecial(RPair, REnvironment): Override to define new special forms (as and)

```
 */
 Object evalSpecial(RPair p, REnvironment env) { return(null); }

 /**
```

Method: evalEval(RPair, REnvironment)

```
 */
 Object evalEval(RPair e, REnvironment env) {
  Object exp = eval(e.nth(1),env);
  Object xenv = eval(e.nth(2),env);
  if ( exp == null ) return(null);
  else if ( xenv == null) return( eval(exp,env) );
  else if ( env.getClass().isInstance(xenv) )
   return( eval(exp,(REnvironment)xenv) );
  else return(null);
 }

 /**
```

Method: evalSet(RPair, REnvironment)

```
 */
 Object evalSet(RPair e, REnvironment env) {
  Object key = e.nth(1);
  if ( key == null ) return(null);
  else {
   Object val = eval(e.nth(2),env);
   return( env.set(key,val) );
  }
 }

 /**
```

Method: evalDef(RPair, REnvironment)

```
 */
 Object evalDef(RPair e, REnvironment env) {
  Object key = e.nth(1);
  if ( key == null ) return(null);
  else {
   Object val = eval(e.nth(2),env);
   return( env.define(key,val) );
  }
 }

 /**
```

Method: evalCond(RPair, REnvironment)

```
*/
 Object evalCond(RPair e, REnvironment env) {
  return( evalClauses(e.CDR(),env) );
 }
 private Object evalClauses(RPair cl, REnvironment env) {
  if (cl == null) return(null);
  Object car = cl.car();
  if ( car == null ) return(null);
  else {
   if ( !RPair.isRPair(car) || RPair.isNil(car) ) return(null);
   else {
    RPair first = (RPair)car;
    if ( isTrue(eval(first.car(),env)) )
     return(evalSequence(first,env));
    else return(evalClauses(cl.CDR(),env));
   }
  }
 }

 /**
```

Method: evalSequence(RPair, REnvironment)

```
*/
 Object evalSequence(RPair e, REnvironment env) {
  if (e == null) return(null);
  if ( e.cdr() == null ) return( eval(e.car(),env) );
  else {
   eval(e.car(),env); // for side-effects
   return( evalSequence(e.CDR(),env) );
  }
 }

 /**
```

Method: evalLambda(RPair, REnvironment)

```
*/
 Object evalLambda(RPair e, REnvironment env) {
  return( RPair.cons("LAMBDA", RPair.cons(env, e.CDR())) );
 }

 /**
```

Method: evalRho(RPair, REnvironment)

```
*/
 Object evalRho(RPair e, REnvironment env) {
  return( RPair.cons("RHO", e.CDR()) );
 }

 /**
```

Method: `evalRPair(RPair, REnvironment)`

```
*/
 RPair evalRPair(RPair e, REnvironment env) {
  if (e == null) return(null);
  if (e.isNil()) return(RPair.nil);
  else return( RPair.cons( eval(e.car(),env), evalRPair(e.CDR(),env) ) );
 }

 /**
```

Method: `apply(RPair, REnvironment)`

The main applicator.

It applies, where `l` is the list `(b c d)`:

- `(cons a l)` → `(a b c d)`.
- `(car l)` → b.
- `(cdr l)` → `(c d)`.
- `(atom? exp)` → true | false.
- `(eq? exp1 exp2)` → true | false.

It accepts two kinds of compound aplication:

- `((LAMBDA env formals body) actuals)` for functions.
- `((RHO name expander) expression)` for special forms.

```
*/
 Object apply(RPair e, REnvironment env) {
  Object op = eval(e.car(),env);
  if ( op == null ) {
   System.err.println("ERROR: null function!");
   return(null);
  } else {
   if ( RPair.isRPair( op ) )
    return( applyCompound( (RPair)op,e.CDR(),env) );
   else if ( isPrimitive(op) )
    return( applyPrimitive(op, evalRPair(e.CDR(),env)) );
   else {
    System.err.println("ERROR: " + op + " undefined!");
    return(null);
   }
  }
 }

 /**
```

Method: `isPrimitive(Object)`

```
*/
 boolean isPrimitive(Object op) {
  return( "cons".equals(op) || "car".equals(op) || "cdr".equals(op) ||
          "atom?".equals(op) || "eq?".equals(op) );
 }

 /**
```

Method: `applyPrimitive(Object, RPair)`

```
*/
 Object applyPrimitive(Object op, RPair args) {
  Object first = args.nth(0);
  if ( "cons".equals(op) ) { Object second = args.nth(1);
   if ( RPair.isRPair(second) ) return( RPair.cons(first, (RPair)second) );
   else return(new RPair(first, second));
  } else if ( "car".equals(op) ) {
   if ( RPair.isRPair(first) ) return( ((RPair)first).car() );
   else return(null);
  } else if ( "cdr".equals(op) ) {
   if ( RPair.isRPair(first) ) return( ((RPair)first).Cdr() );
   else return(null);
  } else if ("atom?".equals(op)) {
   if ( RPair.isAtom( args.car() ) ) return(t);
   else return(RPair.nil);
  } else if ("eq?".equals(op)) { Object second = args.nth(1);
   if ( first == null )
    if ( second == null ) return(t); else return(RPair.nil);
   else
    if ( first.equals(second) ) return(t); else return(RPair.nil);
  } else return(null);
 }

 /**
```

Method: `applyCompound(RPair, RPair, REnvironment)`

```
*/
 Object applyCompound(RPair op, RPair args, REnvironment env) {
  if ( "LAMBDA".equals(op.car()) ) return( applyLambda(op, args, env) );
  else if ( "RHO".equals(op.car()) ) return( applyRho(op, args, env) );
  else {
   System.err.println("ERROR: " + op.car() + " undefined!");
   return(null);
  }
 }

 /**
```

Method: `applyLambda(RPair, RPair, REnvironment)`

It first evaluates the arguments in the current environment. Then it extends the stored environment, that in which the function was defined, with a new `RFrame` in which each formal argument is bound to its actual value. Finally the body is evaluated in the extended environment, alse known as closure.

In applying `((lambda env (f0 f1) e0 ...) a0 a1)` then `op` = `(lambda env (f0 f1) e0 ...)`.

```
*/
 Object applyLambda(RPair op, RPair args, REnvironment currentenv) {
  Object result = null;
  try {
   RPair evargs = evalRPair(args, currentenv);
   REnvironment env = (REnvironment)(op.nth(1)); // stored env
   Object formals = op.nth(2); // (f0 f1)
   RPair body = op.CDR().CDR().CDR();  // (e0 ...)
   REnvironment ext = env.extend(new RFrame(formals,evargs));
   result = evalSequence( body, ext );
  } catch(Throwable t) { System.err.println("ERROR! in lambda: "+t); }
  return(result);
 }

 /**
```

Method: `applyRho(RPair, RPair, REnvironment)`

It first reconstructs the expression, `cons`ing the `name`. Then the reconstructed expression is `quote`d, that is, taken as data,. and, as such, is expanded by the `expander`. Finally, the expanded expression is evaluated.

In applying `((rho name expander) expression)` then `op` = `(rho name expander)`, `(car (cdr op))` is the `name`, and `(car (cdr (cdr op)))` is the `expander`.

```
*/
 Object applyRho(RPair op, RPair args, REnvironment env) {
  Object result = null;
  try{
   Object name = op.CDR().car();
   Object expander = op.CDR().CDR().car();
   RPair expression = RPair.cons(name,args);
   RPair qexp = RPair.cons("quote", RPair.cons(expression,null));
   Object expansion = eval(RPair.cons(expander,RPair.cons(qexp,null)), env);
   result = eval(expansion, env);
  } catch(Throwable t) { System.err.println("ERROR! in rho: "+t); }
  return( result );
 }

 /**
```

Method: `toString()`

```
*/
 public String toString() { return("RLisp"); }

}
```

2.5 File: RLispJava.java

```
/**
```

Class: `RLispJava`

It extends `RLisp` with the following special forms:

- `(string esto es todo)`
- `(new Class (arg0 arg1))` with arg = ob `| (cons 'Class ob)`
- `(method [ Class | ob ] Method (arg0 arg1))`
- `(array Class (ob0 ob1 ob2))`
- `(field [ Class | ob ] Field [ val | ])`
- `(path URL)`
- `(load URL)`

`@author` © Ramón Casares 2003

`@version` 2003.02.08

```
*/
package RLisp;

import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.lang.reflect.Array;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.net.URL;
import java.io.InputStreamReader;
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;

public class RLispJava extends RLisp {

 /**
```

Variable: `rcl` is the incremental class loader used

```
*/
 private RClassLoader rcl;

 /**
```

Variable: `baseURL` is the base directory used when loading files

```
*/
 private URL baseURL;

 /**
```

Constructor: `RLispJava(RClassLoader)`

```
*/
 public RLispJava(RClassLoader rcl) {
  super();
  this.rcl = rcl;
  baseURL = null;
  try {
   baseURL = new URL("file:"+System.getProperty("user.dir")+File.separator);
  } catch(Throwable t) { System.err.println(t); }
 }

 /**
```

Method: isSpecial(RPair): Overrides parent method isSpecial(RPair).

```
*/
 boolean isSpecial(RPair p) { return ( super.isSpecial(p) ||
   "string".equals(p.car()) ||
   "new".equals(p.car())    || "method".equals(p.car()) ||
   "field".equals(p.car())  || "array".equals(p.car())  ||
   "path".equals(p.car())   || "load".equals(p.car())   );
 }

 /**
```

Method: evalSpecial(RPair, REnvironment): Overrides parent method evalSpecial(RPair, REnvironment).

```
*/
 Object evalSpecial(RPair p, REnvironment env) {
  if ( super.isSpecial(p) ) return( super.evalSpecial(p,env) );
  else return( evalJava(p,env) );
 }

 /**
```

Method: evalJava(RPair, REnvironment)

```
*/
 Object evalJava(RPair je, REnvironment env) {
  //String op = je.nth(0).toString();    // operation
  Object op = eval(je.nth(0),env);
  if      ( "new".equals(op) )    return( evalJnew(je.CDR(),env) );
  else if ( "array".equals(op) )  return( evalJarray(je.CDR(),env) );
  else if ( "method".equals(op) ) return( evalJrun(je.CDR(),env) );
  else if ( "field".equals(op) )  return( evalJset(je.CDR(),env) );
  else if ( "path".equals(op) )   return( evalJpath(je.CDR(),env) );
  else if ( "load".equals(op) )   return( evalJload(je.CDR(),env) );
  else if ( "string".equals(op) ) return( je.CDR().toString(false) );
  else {
   System.err.println("ERROR: (" + op +  " ...) undefined");
   return(null);
  }
 }


 /**
```

Method: evalJnew(RPair, REnvironment)

Note that je = (Class arg0 arg1 ...)

We treat specially the case argi = (cons 'Class object).

```
*/
 Object evalJnew(RPair je, REnvironment env) {
  if (je == null || je.isNil() ) {
   System.err.println("ERROR: (new) found!");
   return(null);
  }
  Object result = null;
  try {
   Object co = eval(je.car(),env);
   if ( co == null ) return(null);
   Class cc = StoC(co.toString());
   RPair args = null; Object arg = null;
   int l = RPair.isList( je.CDR() );
   if ( l < 0 ) return(null); else args = (RPair)(je.CDR());
   Class[] argc = new Class[l];
   Object[] arga = new Object[l];
   for(int i=0; i<l; i++) {
    arg = args.nth(i);
    arga[i] = eval(arg,env);
    if ( arga[i] == null ) argc[i] = Void.TYPE;
    else {
     argc[i] = CtoC(arga[i].getClass());
     if ( RPair.isRPair(arga[i]) && ((RPair)arga[i]).car() != null ) {
      Class ca = StoC( ((RPair)arga[i]).car().toString() );
      if ( ca != null ) { argc[i] = ca;
       Object oa = ((RPair)arga[i]).cdr();
       if ( oa == null && ca.isInstance(RPair.nil) ) arga[i] = RPair.nil;
       else if ( oa == null || ca.isInstance(oa) ) arga[i] = oa;
       else arga[i] = StoO(ca,oa.toString());
      }
     }
    }
   }
   if ( cc.isArray() ) {
    Class cp = cc; while (cp.isArray()) cp = cp.getComponentType();
    int[] argi = new int[arga.length];
    for(int i=0; i<arga.length; i++)
     argi[i] = Integer.parseInt(arga[i].toString());
    result = Array.newInstance(cp, argi);
   } else result = cc.getConstructor(argc).newInstance(arga);
  } catch(Throwable te) {
   System.err.println("ERROR: (new "+je.toString(false)+") ["+te+"]");
   result = null;
  }
  return(result);
 }

 /**
```

Method: `evalJarray(RPair, REnvironment)`

Note that `je = (Class ob0 ob1 ...)`

```
 */
 Object evalJarray(RPair je, REnvironment env) {
  Object result = null;
  if (je == null || je.isNil() ) {
   System.err.println("ERROR: (array) found!");
   return(null);
  }
  try{
   Object co = eval(je.car(),env);
   if ( co == null ) return(null);
   Class cc = StoC(co.toString());
   RPair args = null;
   int l = RPair.isList( je.CDR() );
   if ( l < 0 ) return(null); else args = (RPair)(je.CDR());
   result = Array.newInstance(cc,l);
   Object arg = null;
   for(int i=0; i<l; i++) {
    arg = eval(args.nth(i),env);
    if ( "String".getClass().equals(arg.getClass()) )
     arg = StoO(cc, (String)arg);
    Array.set(result, i, arg);
   }
  } catch(Throwable t) { System.err.println("ERROR: "+t); }
  return(result);
 }

 /**
```

Method: evalJrun(RPair, REnvironment)

Note that je = ([Class | ob] Method arg0 arg1 ...))

We treat specially the case argi = (cons 'Class object).

```
 */
 Object evalJrun(RPair je, REnvironment env) {
  if (je == null || je.isNil() ) {
   System.err.println("ERROR: (method) found!");
   return(null);
  }
  Object result = null;
  Object o = eval(je.car(),env);
  Class cc = null;
  try { cc = Class.forName( o.toString(), true, rcl ); }
  catch(Throwable t) { cc = o.getClass(); } // o is not a Class name
  try {
   Object mo = eval(je.CDR().car(),env);
   if ( mo == null ) {
    System.err.println("ERROR: method not found!");
    return(null);
   }
   String mn = mo.toString();
   RPair args = null; Object arg = null;
   int l = RPair.isList( je.CDR().CDR() );
   if ( l < 0 ) return(null); else args = (RPair)(je.CDR().CDR());
   Class[] argc = new Class[l];
   Object[] arga = new Object[l];
   for(int i=0; i<l; i++) {
    arg = args.nth(i);
    arga[i] = eval(arg,env);
    if ( arga[i] == null ) argc[i] = Void.TYPE;
    else {
     argc[i] = CtoC(arga[i].getClass());
     if ( RPair.isRPair(arga[i]) && ((RPair)arga[i]).car() != null) {
      Class ca = StoC( ((RPair)arga[i]).car().toString() );
      if ( ca != null ) { argc[i] = ca;
       Object oa = ((RPair)arga[i]).cdr();
       if ( oa == null && ca.isInstance(RPair.nil) ) arga[i] = RPair.nil;
       else if ( oa == null || ca.isInstance(oa) ) arga[i] = oa;
```

```
        else arga[i] = StoO(ca,oa.toString());
       }
      }
     }
    }
    //Method met = cc.getDeclaredMethod(mn,argc);
    Method met = cc.getMethod(mn,argc);
    result = met.invoke(o,arga);
   } catch(Throwable te) {
    System.err.println("ERROR: (method "+je.toString(false)+") ["+te+"]");
    result = null;
   }
   return(result);
  }


  /**
```

Method: `evalJset(RPair, REnvironment)`

Note that je = `([Class|ob] Field [val| ])`

```
 */
  Object evalJset(RPair je, REnvironment env) {
   if (je == null || je.isNil() ) {
    System.err.println("ERROR: (field) found!");
    return(null);
   }
   Object result = null;
   Object o = eval(je.car(),env);
   Class cc = null;
   try { cc = Class.forName( o.toString(), true, rcl ); }
   catch(Throwable t) { cc = o.getClass(); } // o is not a Class name
   try {
    Object fo = eval(je.CDR().car(),env);
    if ( fo == null ) {
     System.err.println("ERROR: field not found!");
     return(null);
    }
    String fn = fo.toString();
    //Field f = cc.getDeclaredField( fn );
    Field f = cc.getField( fn );
    if ( !RPair.isNil(je.CDR().CDR()) ) {
     Object v = eval(je.nth(2),env);
     if ( "String".getClass().equals( v.getClass() ) )
      v = StoO( f.getType(), (String)v );
     f.set(o, v);
    }
    result = f.get(o);
   }
   catch(Throwable te) {
    System.err.println("ERROR: (field "+je.toString(false)+") ["+te+"]");
    result = null;
   }
   return(result);
  }

 /**
```

Method: `evalJpath(RPair, REnvironment)`

```
*/
 Object evalJpath(RPair je, REnvironment env) {
  String p = je.toString(false);
  try { rcl.addURL(new URL(baseURL, p)); }
  catch(Throwable t) { System.err.println(t); p = null; }
  return(p);
 }

 /**
```

Method: `evalJload(RPair, REnvironment)`

A `(load URL)` calculates the location from a context. The base context is the user directory, `System.getProperty("user.dir")`. But each `(load URL)` sets the context to this `URL`, so from file `Primes.lisp` to load a file `Maths.lisp` in the same directory just write `(load Maths.lisp)`.

For file addresses use: `(load file:path/filename.ext)`.

For files inside `jar` files use:
`(load jar:file:path/file.jar!/inpath/filename.ext)`.

```
*/
 Object evalJload(RPair je, REnvironment env) {
  Object res = null;
  URL oldbaseURL = baseURL;
  try {
   String urln = je.toString(false);
   URL url = new URL(baseURL, urln);
   // System.err.println("baseURL = "+baseURL);
   // System.err.println("    url = "+url);
   baseURL = url;
   InputStreamReader jisr = new InputStreamReader(url.openStream());
   BufferedReader in = new BufferedReader(jisr);
   String fc = "";
   String newline = in.readLine();
   while (newline != null) {
    fc = fc + newline + "\n";
    newline = in.readLine();
   }
   Object[] exp = RPair.Tokenize(fc);
   if (exp == null) return(null);
   for(int i=0; i<exp.length; i++) res = eval(exp[i],env);
  } catch(Throwable t) {
   System.err.println(t);
   res = null;
  }
  baseURL = oldbaseURL;
  return(res);
 }


 /**
```

Method: `CtoC(Class)`

If the `Class oc` is a primitive type enclosing class, then returns the `Class` object representing the primitive type. Otherwise it returns `oc`.

```
*/
 private Class CtoC(Class oc) {
  Class c = oc;
  if (oc==null) return(null);
  else if (oc.equals(Boolean.class)) c = Boolean.TYPE;
  else if (oc.equals(Character.class)) c = Character.TYPE;
  else if (oc.equals(Integer.class)) c = Integer.TYPE;
  else if (oc.equals(Byte.class)) c = Byte.TYPE;
  else if (oc.equals(Short.class)) c = Short.TYPE;
  else if (oc.equals(Long.class)) c = Long.TYPE;
  else if (oc.equals(Float.class)) c = Float.TYPE;
  else if (oc.equals(Double.class)) c = Double.TYPE;
  return(c);
 }

 /**
```

Method: `StoC(String)`

Given a name, it returns the Class using the incremental Class loader. A bidimensional array of base class Class is noted Class[][].

`@param cn` is the Class name

`@return` the Class object

```
*/
 private Class StoC(String cn) {
  if ( cn == null ) return(null);
  Class c;
  int dims = 0; int l = cn.length();
  while ( cn.lastIndexOf("[]") == l-2 ) { dims++;
   cn = cn.substring(0,l-2); l = cn.length();
  }
  if (cn.equals("java.lang.String")) c = "String".getClass();
  else if (cn.equals("String")) c = "String".getClass();
  else if (cn.equals("boolean")) c = Boolean.TYPE;
  else if (cn.equals("char")) c = Character.TYPE;
  else if (cn.equals("int")) c = Integer.TYPE;
  else if (cn.equals("byte")) c = Byte.TYPE;
  else if (cn.equals("short")) c = Short.TYPE;
  else if (cn.equals("long")) c = Long.TYPE;
  else if (cn.equals("float")) c = Float.TYPE;
  else if (cn.equals("double")) c = Double.TYPE;
  else if (cn.equals("void")) c = Void.TYPE;
  else try { c = Class.forName(cn,true,rcl); }
  catch(ClassNotFoundException e) {
   System.err.println("Class not found: " + cn);
   c = null;
  }
  if (c != null && dims > 0) c = arrayClass(c,dims);
  return(c);
 }

 /**
```

Method: `arrayClass(Class, int)`

Given a base type and a number of dimensions, it returns the corresponding array class.

```
*/
 public static Class arrayClass(Class c, int dims) {
  if ( c == null || dims < 0 ) return(null);
  if ( dims == 0 ) return(c);
  int[] d = new int[dims]; for(int i=0; i<dims; i++) d[i] = 0;
  Class ac = null;
  try{ ac = Array.newInstance(c,d).getClass(); }
  catch(Throwable t) { System.err.println("ERROR: " + t); }
  return(ac);
 }

 /**
```

Method: `StoO(Class, String)`

Given a Class and the name of one value, it returns the corresponding object.

```
*/
 public static Object StoO(Class c, String on) {
   if ( c == null || on == null ) return(null);
   Object o = null;
   if ( c.isInstance(on) ) o = on;
   else if ( c.equals("String".getClass()) ) o = on;
   else if (c.equals(Boolean.TYPE)) o = new Boolean(on);
   else if (c.equals(Character.TYPE)) o = new Character(on.charAt(0));
   else if (c.equals(Integer.TYPE)) o = new Integer(on);
   else if (c.equals(Byte.TYPE)) o = new Byte(on);
   else if (c.equals(Short.TYPE)) o = new Short(on);
   else if (c.equals(Long.TYPE)) o = new Long(on);
   else if (c.equals(Float.TYPE)) o = new Float(on);
   else if (c.equals(Double.TYPE)) o = new Double(on);
   else if (c.equals(Void.TYPE)) o = null;
   else {
    String[] arg = new String[1]; arg[0] = on;
    Class[] carg = new Class[1]; carg[0] = "String".getClass();
    //try { o = c.getDeclaredConstructor(carg).newInstance((Object[])arg); }
    try { o = c.getConstructor(carg).newInstance((Object[])arg); }
    catch(Throwable t) { System.err.println(t); o = null; }
   }
   return(o);
  }

 /**
```

Method: `toString()`

```
*/
 public String toString() { return("RLispJava"); }

}
```

2.6 File: RLispInterpreter.java

```
/**
```

Class: RLispInterpreter

A Lisp interpreter.

It uses ENV as global environment, and an RLisp evaluator lisp.

@author © Ramón Casares 2003

@version 2003.03.03

```
*/
package RLisp;

public class RLispInterpreter {

 /**
```

Variable: ENV

```
*/
 public REnvironment ENV;

 /**
```

Variable: lisp

```
*/
 private RLisp lisp;

 /**
```

Method: counter(int)

```
*/
 public int counter(int val) {
  int i = lisp.counter;
  lisp.counter = val;
  return(i);
 }
 public int counter() { return(lisp.counter); }

 /**
```

Constructor: RLispInterpreter(RLisp)

```
*/
 public RLispInterpreter(RLisp lisp) {
  RFrame FR = new RFrame();
  ENV = new REnvironment(FR);
  this.lisp = lisp;
 }

 /**
```

Method: Eval(String)

```
*/
```

```
 public Object Eval(String input) {
  Object[] exp = RPair.Tokenize(input);
  if (exp == null) return(null);
  Object res = null;
  for(int i=0; i<exp.length; i++) res = lisp.eval(exp[i],ENV);
  return(res);
 }

 /**
```

Method: `eval(String)`

```
*/
 public Object eval(String input) {
  return( lisp.eval(RPair.tokenize(input), ENV) );
 }

 /**
```

Method: `toString()`

```
*/
 public String toString() {
  return(lisp.toString() + " on " + ENV.toString(false));
 }

 /**
```

Method: `main(String[])`

Interpretes the arguments as a list sequence.

Example:
```
<< java RLisp/RLispInterpreter (load RLisp/Primes.lisp) (divisors 1222)
>> (2 13 47)
```

`@param args` the command line arguments

```
*/
 public static void main(String[] args) {
  if ( args.length > 0 ) {
   String s = "";
   for(int i=0; i<args.length; i++) s = s + " " + args[i];
   s = s.substring(1);
   java.net.URL[] urls = new java.net.URL[1];
   java.io.File ud = new java.io.File(System.getProperty("user.dir"));
   try { urls[0] = ud.toURL(); }
   catch (java.net.MalformedURLException mue) {} // always right
   RClassLoader rcl = new RClassLoader(urls);
   RLispInterpreter rli = new RLispInterpreter( new RLispJava(rcl) );
   System.out.println( rli.Eval(s) );
  } else System.out.println( "[null]" );
 }

}
```

2.7 File: RLisp.lisp

```
; RLisp.lisp

(def nil (cons))
(def t (eq? (cons) (cons)))
(def null? (lambda (x) (eq? x nil)))
(def not (lambda (b) (cond (b nil) (t t))))
(def list (lambda l l))
(def cadr (lambda (l) (car (cdr l))))

(def macro
 (rho macro
  (lambda ((macro name expander))
   (list 'def name (list 'rho name expander))
)))

(def syntax
 (rho syntax
  (lambda ((syntax template expansion))
   (list 'def (car template)
    (list 'rho (car template)
     (list 'lambda (list template) expansion)
)))))

(syntax (define name definition)
 (cond
  ((atom? name) (list 'def name definition))
  (t (list 'define (car name) (list 'lambda (cdr name) definition)))
))

(syntax (if test t-clause f-clause)
 (list 'cond (list test t-clause) (list 't f-clause)))

(syntax (sequence , expressions)
 (list 'cond (cons 't expressions)))

; (or b1 b2 ...) =>  (cond (b1 t) (t (or b2 ...)))
(syntax (or , terms)
 (cond ((eq? terms nil) 'nil)
  (t (list 'cond (list (car terms) 't) (list 't (cons 'or (cdr terms)))))
))

; (and b1 b2 ...) => (cond (b1 (and b2 ...) (t nil))) =>
(syntax (and , terms)
 (cond ((eq? terms nil) 't)
  (t (list 'cond (list (car terms) (cons 'and (cdr terms))) (list 't 'nil)))
))

(define (mapcar f l)
 (cond ((eq? l nil) nil)
  (t (cons (f (car l)) (mapcar f (cdr l))))
))

; (let ((f1 v1) (f2 v2)) body) => ((lambda (f1 f2) boby) v1 v2)
(syntax (let arglist , body)
 (cons
  (cons 'lambda (cons (mapcar car arglist) body))
  (mapcar cadr arglist)
))

(def GENV (car (cdr (lambda))))
(syntax (Gdefine name definition)
 (list 'eval (list 'def name definition) 'GENV))

(load RLispJava.lisp)
(load RLispArray.lisp)
(load RLispMaths.lisp)
```

2.8 File: RLispJava.lisp

```
; RLispJava.lisp (RMCG20040131)

(cond ((eq? 'define define) (load RLisp.lisp)) (t))

; Java null cannot be in the dictionary. Write (car (cons)) to get it.

(define (boolean b) (new 'java.lang.Boolean (method b 'toString)))
(define (char c) (new 'java.lang.Character
  (method (method c 'toString) 'charAt (int 0))))

(define (byte n) (new 'java.lang.Byte (method n 'toString)))
(define (short n) (new 'java.lang.Short (method n 'toString)))
(define (int n) (new 'java.lang.Integer (method n 'toString)))
(define (long n) (new 'java.lang.Long (method n 'toString)))

(define (float n) (new 'java.lang.Float (method n 'toString)))
(define (double n) (new 'java.lang.Double (method n 'toString)))
```

2.9 File: RLispArray.lisp

```
; RLispArray.lisp (RMCG20040131)

(cond ((eq? 'define define) (load RLisp.lisp)) (t))
(cond ((eq? 'int int) (load RLispJava.lisp)) (t))

(define (isArray? o) (method (method o 'getClass) 'isArray))

; (new Class[] dim1 dim2) creates a bidimensional array sized dim1 x dim2
; (array Class ob1 ob2) creates an array of length 2 initialized

(define (array-length a)
 (method 'java.lang.reflect.Array 'getLength (cons 'java.lang.Object a)))

(define (array-get a i)
 (method 'java.lang.reflect.Array 'get
  (cons 'java.lang.Object a) (cons 'int i)
))

(define (array-set! a i v)
 (method 'java.lang.reflect.Array 'set
  (cons 'java.lang.Object a) (cons 'int i) (cons 'java.lang.Object v)
))

(define (l2v l) ; creates a vector and adds objects in list l to it
 (l2vv l (new 'java.util.Vector)))

(define (l2vv l v) ; adds objects in list l to Vector v
 (cond
  ((eq? l nil) v)
  (t
   (method v 'add (cons 'java.lang.Object (car l)))
   (l2vv (cdr l) v)
)))
```

2.10 File: RLispMaths.lisp

```
; RLispMaths.lisp (RMCG20030716)

(cond ((eq? 'define define) (load RLisp.lisp)) (t))
(cond ((eq? 'int int) (load RLispJava.lisp)) (t))

(define (# x)
 (new 'java.math.BigInteger (method x toString)))

(define (++ x y) (method (# x) add (# y)))
(define (- x y) (method (# x) subtract (# y)))
(define (** x y) (method (# x) multiply (# y)))
(define (/ x y) (method (# x) divide (# y)))
(define (% x y) (method (# x) remainder (# y)))
(define (> x y)
 (eq? (int 1) (method (# x) compareTo (# y))))
(define (= x y) (eq? (# x) (# y)))

(define (sigma l)
 (cond ((eq? l nil) (# 0))
  (t (++ (car l) (sigma (cdr l))))))
(define + (lambda l (sigma l)))

(define (pi l)
 (cond ((eq? l nil) (# 1))
  (t (** (car l) (pi (cdr l))))))
(define (* , l) (pi l))
```

2.11 File: Primes.lisp

```
; Primes.lisp (RMCG20030716)

(cond ((eq? 'define define) (load RLisp.lisp)) (t))
(cond ((eq? '* *) (load RLispMaths.lisp)) (t))

(define (divides? a b) (= (% b a) 0))
(define (square x) (* x x))
(define (find-divisor n test-divisor)
 (cond
  ((> (square test-divisor) n) n)
  ((divides? test-divisor n) test-divisor)
  (t (find-divisor n (+ test-divisor 1)))))
(define (smallest-divisor n) (find-divisor n 2))
(define (prime? n) (= n (smallest-divisor n)))
(define (divisors x)
 (cond
  ((prime? x) (cons x nil))
  (t (cons (smallest-divisor x) (divisors (/ x (smallest-divisor x)))))))
```

2.12 File: RLispConsole.java

```
/**
```

Class: `RLispConsole`

An `RLispConsole` is a Graphical User Interface (GUI) that implements a Java interpreter. This interpreter uses a complete Lisp interpreter, with its own environment to store the named objects, so the Java interpreter syntax is Lispian (or Schemian).

It allows: `new`) to define a Java object, `array`) or a matrix, `path`) from a directory (also known as folder), `name`) and give it a name, `unname`) or took it away; `run`) to run a method, `set`) and to see or set a field value.

`@author` © Ramón Casares 2003

`@version` 2003.03.19

```
*/
package RLisp;

import java.net.URL;

import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.lang.reflect.Field;
import java.lang.reflect.Array;
import java.lang.reflect.Modifier;
import java.lang.reflect.InvocationTargetException;

import java.util.Vector;
import java.util.BitSet;
import java.util.Date;

import java.io.PrintWriter;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.PrintStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.InputStream;
import java.io.FileReader;

import java.io.IOException;
import java.io.FileNotFoundException;
import javax.swing.text.BadLocationException;

import java.util.*;
import java.awt.*;
import java.io.File;
import javax.swing.*;
import java.awt.event.*;
import java.io.PipedWriter;
import java.io.PipedReader;
import java.io.BufferedReader;
import java.io.IOException;

import javax.swing.JOptionPane;

public class RLispConsole implements WindowListener, ActionListener {

 public RLispInterpreter lisp;

 private Object result; // referred to as @
 private String expression;
 private String logfilename = "RLisp.log"; // default name
 private boolean logging = false;
 private PrintWriter outfile = null;
 private String version = "20040115";

 /**
```

Variable: `rcl`

It is an incremental `ClassLoader` that is used for loading all of the classes. For the Java Virtual Machine, the same `.class` file loaded twice from the same directory by two different `ClassLoader`s, are two completelly different classes. Because of this, the same `ClassLoader` should load every class.

```
*/
 private RClassLoader rcl;

 JFrame frame;
 JTextArea textArea;
 JLabel statuslabel;
 Container contentPane;

 /**
```

Constructor: `RLispConsole(String)`

```
*/
 public RLispConsole(String title) {

  URL[] urls = new URL[1];
  File ud = new File(System.getProperty("user.dir"));
  try { urls[0] = ud.toURL(); }
  catch (java.net.MalformedURLException mue) {} // always right
  rcl = new RClassLoader(urls);

  lisp = new RLispInterpreter( new RLispJava(rcl) );

  frame = new JFrame(title);

  frame.addNotify();
  frame.addWindowListener(this);

  frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

  //setBounds(200, 200, 500, 200);

  contentPane = frame.getContentPane();
  //contentPane.setLayout(new FlowLayout());
  //contentPane.setLayout(new GridLayout(3,1));
  //contentPane.setLayout(new BoxLayout(contentPane,BoxLayout.PAGE_AXIS));
  contentPane.setLayout(new BoxLayout(contentPane,BoxLayout.Y_AXIS));

  JMenuBar Bar = new JMenuBar();
   JMenu menuFile = new JMenu("File"); ////////////////////
    JMenuItem miLoad = new JMenuItem("Load");
     miLoad.addActionListener(this);
     menuFile.add(miLoad);
    JMenuItem miLoadFrom = new JMenuItem("Load from");
     miLoadFrom.addActionListener(this);
     menuFile.add(miLoadFrom);
    JMenuItem miSave = new JMenuItem("Save");
     miSave.addActionListener(this);
     menuFile.add(miSave);
    JMenuItem miSaveTo = new JMenuItem("Save to");
     miSaveTo.addActionListener(this);
     menuFile.add(miSaveTo);
     menuFile.addSeparator();
    JMenuItem miTree = new JMenuItem("Tree");
     miTree.addActionListener(this);
     menuFile.add(miTree);
    JMenuItem miKeyboard = new JMenuItem("Keyboard");
     miKeyboard.addActionListener(this);
     menuFile.add(miKeyboard);
    JMenuItem miSession = new JMenuItem("Session");
     miSession.addActionListener(this);
     menuFile.add(miSession);
    menuFile.addSeparator();
    JMenuItem miClose = new JMenuItem("Close");
     miClose.addActionListener(this);
     menuFile.add(miClose);
   JMenu menuEdit = new JMenu("Edit"); ////////////////////
    JMenuItem miCut = new JMenuItem("Cut");
     miCut.setEnabled(false);
     miCut.addActionListener(this);
     menuEdit.add(miCut);
    JMenuItem miCopy = new JMenuItem("Copy");
     miCopy.addActionListener(this);
     menuEdit.add(miCopy);
    JMenuItem miPaste = new JMenuItem("Paste");
     miPaste.setEnabled(false);
     miPaste.addActionListener(this);
     menuEdit.add(miPaste);
   JMenu menuAction = new JMenu("Action"); ////////////////
    JMenuItem miPath = new JMenuItem("Path");
     miPath.addActionListener(this);
     menuAction.add(miPath);
    JMenuItem miLisp = new JMenuItem("Lisp code");
```

```
      miLisp.addActionListener(this);
      menuAction.add(miLisp);
     JMenuItem miName = new JMenuItem("Name");
      miName.addActionListener(this);
      menuAction.add(miName);
     JMenuItem miUnname = new JMenuItem("Unname");
      miUnname.addActionListener(this);
      menuAction.add(miUnname);
     JMenuItem miList = new JMenuItem("List");
      miList.addActionListener(this);
      menuAction.add(miList);
    JMenu menuHelp = new JMenu("Help"); ////////////////////
     JMenuItem miManual = new JMenuItem("Manual");
      miManual.addActionListener(this);
      menuHelp.add(miManual);
     JMenuItem smiManual = new JMenuItem("Spanish Manual");
      smiManual.addActionListener(this);
      menuHelp.add(smiManual);
     JMenuItem miCode = new JMenuItem("Code");
      miCode.addActionListener(this);
      menuHelp.add(miCode);
     JMenuItem miAbout = new JMenuItem("About RLisp");
      miAbout.addActionListener(this);
      menuHelp.add(miAbout);

   Bar.add(menuFile);
   Bar.add(menuEdit);
   Bar.add(menuAction);
   Bar.add(menuHelp);
   frame.setJMenuBar(Bar);

   textArea = new JTextArea(15,50);
   textArea.setEditable(false);
   textArea.setLineWrap(false);
   textArea.setBackground(new Color(1.0F,1.0F,0.5F)); // yellow
   textArea.setFont(new Font("Monospaced",Font.PLAIN,12));

   contentPane.add(new JScrollPane(textArea,
    ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
    ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED));

   Box statusbox = Box.createHorizontalBox();
    statuslabel = new JLabel("Status: ");
    statusbox.add(statuslabel);
    statusbox.add(Box.createHorizontalGlue());
    contentPane.add(statusbox);

   frame.setVisible(true);
   frame.pack();
  }

  /**
```

Method: `initLisp()`

If the `RLisp` is running from `RLisp.jar`, then it loads the file `RLisp.lisp` in the `jar` file. Otherwise it loads the file RLisp.lisp which is in the same directory that the RLispConsole.class that is running.

```
*/
 public void initLisp() {
  try{
   URL lispURL =
    this.getClass().getClassLoader().getResource("RLisp/RLisp.lisp");
   String sURL = lispURL.toString();
   writeln("<< (load " + sURL + ")");
   lisp.eval("(load " + sURL + ")");
   System.out.println("Init Lisp [" + lisp.counter(0) + "]");
  } catch(Throwable t) { System.err.println(t); }
 }

 /**
```

Method: setLogFile(String)

```
*/
 public void setLogFile(String filename) { this.logfilename = filename; }

 /**
```

Method: readFile(String)

```
*/
 public void readFile(String filename) throws IOException {
  BufferedReader infile = new BufferedReader(new FileReader(filename));
  String line = infile.readLine();
  while ( (line != null) && !("<< quit".equals(line)) ) {
   textArea.append(line + "\n");
   if ( line.startsWith("<< ") ) {
    expression = line.substring(3);
    result = lisp.eval(expression);
    textArea.append(">< " + resultToString() + "\n" );
    lisp.ENV.define("@",result);
   }
   line = infile.readLine();
  }
  infile.close();
  if ("<< quit".equals(line)) closeAction();
 }

 /**
```

Method: session()

It runs a session in the system console. If there were not a system console, because Java was call as javaw, then control would be lost and should be recovered manually by pressing Ctr-Alt-Del and then aborting task javaw.

```
*/
 public void session() throws IOException {
  frame.setVisible(false);
  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
  String outputLine = "New session: " + new Date();
  System.out.println(">> "+ outputLine); writeln(">> " + outputLine);
  System.out.println("<>  To end the session, enter \"quit\"");
  String oldLine = "quit";
  System.out.print("<< "); expression = in.readLine();
  if (expression.equals("")) expression = oldLine;
  while ( !("quit".equals(expression)) ) {
   writeln("<< " + expression);
   result = lisp.eval(expression);
   outputLine = resultToString();
   System.out.println(">> " + outputLine);
   writeln(">> " + outputLine);
   lisp.ENV.define("@",result);
   oldLine = expression;
   System.out.print("<< "); expression = in.readLine();
   if (expression.equals("")) expression = oldLine;
  }
  writeln("<> Session finished: " + new Date());
  frame.setVisible(true);
 }

 /**
```

Method: `write(String)`

```
*/
 public void write(String s) {
  textArea.append(s);
  if ( outfile != null ) outfile.print(s);
 }

 /**
```

Method: `writeln(String)`

```
*/
 public void writeln(String s) {
  textArea.append(s); textArea.append("\n");
  if ( logging ) outfile.println(s);
 }

 /**
```

Method: `writeln()`

```
*/
 public void writeln() { writeln(""); }


 /**
```

Method: `executeObject(Object)`

```
*/
 public void executeObject(Object rro) {
  if ( rro == null ) return;
  Object[] values = null;
  Object value = null;
  String keyselected = null;

```

```
try {

 if ( rro.getClass() == Class.forName("java.lang.reflect.Constructor") ) {
  Constructor c = (Constructor)rro;
  expression = "(new \'" + c.getName();
  values = getValues( c.getParameterTypes() );
  expression = expression + ")";
  writeln("<< " + expression);
  if ( values != null ) {
   result = c.newInstance(values);
   writeln(">> " + resultToString());
   lisp.ENV.define("@",result);
  } else { System.err.println("new " + rro + " ERROR!"); }

 } else if ( rro.getClass() == Class.forName("java.lang.reflect.Method") ) {
  Method c = (Method)rro;
  Class cl = c.getDeclaringClass();
  Object obj = null;
  if ( Modifier.isStatic(c.getModifiers()) ) {
   expression = "(method \'" + cl.getName() + " \'" + c.getName();
   values = getValues( c.getParameterTypes() );
   expression = expression + ")";
   writeln("<< " + expression);
  } else {
   Object[] objs = lisp.ENV.keys(cl,false);
   if (objs != null && objs.length > 0 ) {
    keyselected = (String)JOptionPane.showInputDialog(null,
                   getName(cl), "Choose an object", // toString
                   JOptionPane.QUESTION_MESSAGE, null,
                   objs, objs[0] );
    if ( keyselected == null ) values = null;
    else {
     obj = lisp.ENV.lookup( keyselected );
     expression = "(method " + keyselected + " \'" + c.getName();
     values = getValues( c.getParameterTypes() );
     expression = expression + ")";
     writeln("<< " + expression);
    }
   }
  }
  if ( values != null ) {
   result = c.invoke(obj,values);
   if ( result == null ) writeln(">> [null]");
   else writeln(">> " + resultToString());
   lisp.ENV.define("@",result);
  } else { System.err.println("run " + rro + " ERROR!"); }

 } else if ( rro.getClass().isArray() ) {
  Class c = rro.getClass().getComponentType();
  expression = "(array \'" + c.getName();
  Vector<Object> vals = new Vector<Object>();
  Object val = getValue(c);
  while ( val != null ) { vals.add(val); val = getValue(c); }
  expression = expression + ")";
  writeln("<< " + expression);
  Object[] valsa = vals.toArray();
```

```
    int l = valsa.length;
    result = Array.newInstance(c,l);
    for(int i=0; i<l; i++) Array.set(result, i, valsa[i]);
    writeln(">> " + resultToString());
    lisp.ENV.define("@",result);

   } else if ( rro.getClass() == Class.forName("java.lang.reflect.Field") ) {
    Field f = (Field)rro;
    Class cl = f.getDeclaringClass();
    Object obj = null;
    String owner = null;
    if ( Modifier.isStatic( f.getModifiers() ) ) {
     owner = "\'" + cl.getName();
    } else {
     Object[] objs = lisp.ENV.keys(cl,false);
     if (objs != null && objs.length > 0 ) {
      keyselected = (String)JOptionPane.showInputDialog(null,
                     getName(cl), "Choose an object", // toString
                     JOptionPane.QUESTION_MESSAGE, null,
                     objs, objs[0] );
      if ( keyselected != null ) {
       obj = lisp.ENV.lookup( keyselected );
       owner = keyselected;
      }
     }
    }
    Object currentValue = f.get(obj);
    Class fc = f.getType();
    if ( Modifier.isFinal( f.getModifiers() ) ) {
     JOptionPane.showMessageDialog(null,
      "Final value: " + prettyPrint(currentValue),
      "Field " + f.toString(),
      JOptionPane.INFORMATION_MESSAGE);
     expression = "(field " + owner + " \'" + f.getName() + ")";
     writeln("<< " + expression);
     result = currentValue;
     writeln(">> " + resultToString());
     lisp.ENV.define("@",result);
    } else if (
     JOptionPane.showConfirmDialog(null,
      "Current value: " + prettyPrint(currentValue) + "\n" + // toString
      "Do you want to change it?",
      "Field " + f.toString(),
      JOptionPane.YES_NO_OPTION,
      JOptionPane.INFORMATION_MESSAGE) == JOptionPane.YES_OPTION ) {
     expression = "(field " + owner + " \'" + f.getName();
     currentValue = getValue(fc);
     f.set(obj, currentValue);
     expression = expression + ")";
     writeln("<< " + expression);
     result = f.get(obj);
     writeln(">> " + resultToString());
     lisp.ENV.define("@",result);
    } else { // NO OPTION
     expression = "(field " + owner + " \'" + f.getName() + ")";
     writeln("<< " + expression);
     result = currentValue;
     writeln(">> " + resultToString());
     lisp.ENV.define("@",result);
    }

   } else if ( rro.getClass() == Class.forName("java.io.File") ) {
    RClassTree rct = new RClassTree(this,(File)rro,rcl);
    writeln("<> Tree from  " + rct.cd.toURI().toURL() );
    writeln("<< (path " + rct.cd.toURI().toURL() + ")");

   } else System.err.println("ERROR: No action for class "+rro.getClass());
  }
  catch (ClassNotFoundException cnfe) { System.err.println(cnfe); }
  catch (IllegalAccessException iae) { System.err.println(iae); }
  catch (InvocationTargetException ite) { System.err.println(ite); }
  catch (InstantiationException ie) { System.err.println(ie); }
  catch (IllegalArgumentException iae) { System.err.println(iae); }
  catch (java.net.MalformedURLException mue) { System.err.println(mue); }
  }


  /**
```

Method: `getValue(Class)`

It gets from the user an object of the given `Class`.

`@param c` is the given `Class`

`@return` the chosen object

```
*/
 Object getValue(Class c) {
  if ( c == null ) return(null);
  Object value = null;
  String keyselected = null;
  String sin = getName(c);
  String sout = null;
  Object[] names = lisp.ENV.keys(c,false);
  if ( names == null || names.length == 0 ) {
   sout = JOptionPane.showInputDialog(sin + " expression");
   if ( sout == null ) return(null);
   value = lisp.eval(sout);
   expression = expression + " " + sout;
  } else {
   Object[] namess = new Object[names.length+1];
   namess[0] = sin + " expression";
   for(int j=0; j<names.length; j++) namess[j+1] = names[j];
   keyselected = (String)JOptionPane.showInputDialog(null,
                    sin, "Input value",
                    JOptionPane.QUESTION_MESSAGE, null,
                    namess, namess[0] );
   if ( keyselected == null ) return(null);
   else if ( keyselected.equals(sin + " expression") ) {
    sout = JOptionPane.showInputDialog(sin + " expression");
    if ( sout == null ) return(null);
    value = lisp.eval(sout);
```

```
     expression = expression + " " + sout;
    } else {
     value = lisp.ENV.lookup( keyselected );
     expression = expression + " " + keyselected;
    }
   }
   if ( value == null ) return(null);
   if ( "String".getClass().equals(value.getClass()) )
    value = RLispJava.StoO(c,(String)value);
   return(value);
  }

  /**
```

Method: getValues(Class[])

It gets from the user an array of Objects of the given Classes.

@param c is the array of given Classes

@return the array of chosen Objects

```
 */
  Object[] getValues(Class[] ca) {
   if ( ca == null ) return(null);
   Object[] values = new Object[ca.length];
   String keyselected; Object value;
   String sin; String sout;
   for(int i=0; i<ca.length; i++) {
    keyselected = null; value = null;
    sin = getName(ca[i]); sout = null;
    Object[] names = lisp.ENV.keys(ca[i],false);
    if ( names == null || names.length == 0 ) {
     sout = JOptionPane.showInputDialog(sin + " expression");
     if ( sout == null ) return(null);
     value = lisp.eval(sout);
     expression = expression + " " +
                      "(cons \'" + sin + " " + sout + ")";
    } else {
     Object[] namess = new Object[names.length+1];
     namess[0] = sin + " expression";
     for(int j=0; j<names.length; j++) namess[j+1] = names[j];
     keyselected = (String)JOptionPane.showInputDialog(null,
                     sin, "Input value["+i+"]",
                     JOptionPane.QUESTION_MESSAGE, null,
                     namess, namess[0] );
     if ( keyselected == null ) return(null);
     else if ( keyselected.equals(sin + " expression") ) {
      sout = JOptionPane.showInputDialog(sin + " expression");
      if ( sout == null ) return(null);
      value = lisp.eval(sout);
      expression = expression + " " +
                       "(cons \'" + sin + " " + sout + ")";
     } else {
      value = lisp.ENV.lookup( keyselected );
      expression = expression + " " +
                  "(cons \'" + sin + " " + keyselected + ")";
     }
    }
    if ( value == null ) return(null);
    else {
     if ( "String".getClass().equals(value.getClass()) )
      values[i] = RLispJava.StoO(ca[i],(String)value);
     else
      values[i] = value;
    }
   }
   return values;
  }

  /**
```

Method: getName(Class)

Works as Class.getName(), except with arrays.

@param c is the Class

@return its name

```
 */
 public static String getName(Class c) {
  if ( c == null ) return(null);
  String cis = c.getName();
  if ( c.isArray() ) {
   String atend = "";
   Class pc = c;
   while ( cis.charAt(0) == '[' ) {
    atend = atend + "[]";
    pc = pc.getComponentType();
    cis = cis.substring(1);
   }
   cis = pc.getName() + atend;
  }
  return(cis);
 }

 /**
```

Method: windowClosing(WindowEvent)

Implements interface WindowListener. The only no void method is windowClosing(WindowEvent)

```
 */
 public void windowOpened(WindowEvent e) {}
 public void windowClosing(WindowEvent e) { closeAction(); }
 public void windowClosed(WindowEvent e) {}
 public void windowIconified(WindowEvent e) {}
 public void windowDeiconified(WindowEvent e) {}
 public void windowActivated(WindowEvent e) {}
 public void windowDeactivated(WindowEvent e) {}


 /**
```

Method: `actionPerformed(ActionEvent)`

Implements the `ActionListener` interface.

```
*/
public void actionPerformed(ActionEvent e) {
 String texto = e.getActionCommand();
 statuslabel.setText(texto);
 if      ("Cut".equals(texto))       textArea.cut();
 else if ("Copy".equals(texto))      textArea.copy();
 else if ("Paste".equals(texto))     textArea.paste();
 else if ("Load".equals(texto))      loadAction();
 else if ("Load from".equals(texto)) loadFromAction();
 else if ("Save".equals(texto))      saveAction();
 else if ("Save to".equals(texto))   saveToAction();
 else if ("Close".equals(texto))     closeAction();
 else if ("Tree".equals(texto))      treeAction();
 else if ("Name".equals(texto))      nameAction();
 else if ("Unname".equals(texto))    unnameAction();
 else if ("List".equals(texto))      listAction();
 else if ("Path".equals(texto))      pathAction();
 else if ("Lisp code".equals(texto)) loadLispAction();
 else if ("Keyboard".equals(texto))  keyboardAction();
 else if ("Line".equals(texto))      lineAction(e.getSource());
 else if ("Session".equals(texto))   sessionAction();
 else if ("OK".equals(texto))        okTreeAction(e.getSource());
 else if ("Manual".equals(texto))    pdfAction("RLispManE.pdf");
 else if ("Spanish Manual".equals(texto)) pdfAction("RLispManS.pdf");
 else if ("Code".equals(texto))      pdfAction("RLispCode.pdf");
 else if ("About RLisp".equals(texto)) aboutAction();
 else System.err.println("ERROR: Action " + texto + " undefined!");
 System.out.println(texto + " [" + lisp.counter(0) + "]");
}

/**
```

Method: `closeAction()`

```
*/
private void closeAction() {
 writeln("<> Closing: " + new Date());
 if (outfile != null) outfile.close();
 System.exit(0);
}

/**
```

Method: `lineAction(Object)`

```
*/
private void lineAction(Object so) {
 RButton rb = (RButton)so;
 expression = (String)rb.getObject();
 writeln("<< " + expression);
 if ( "quit".equals(expression) ) closeAction();
 else {
  result = lisp.eval(expression);
  writeln(">> " + resultToString());
```

```
   lisp.ENV.define("@",result);
  }
 }

 /**
```

Method: okTreeAction(Object)

```
*/
 private void okTreeAction(Object so) {
  if (so == null) return;
  RButton rb = (RButton)so;
  RClassTree rct = (RClassTree)rb.getObject();
  executeObject(rct.getSelectedObject());
 }

 /**
```

Method: sessionAction()

```
*/
 private void sessionAction() {
  try { session(); } catch (Throwable t) {System.err.println(t);}
 }

 /**
```

Method: keyboardAction()

```
*/
 private void keyboardAction() {
  new RKeyboard(this,"(quote Keyboard)");
 }

 /**
```

Method: listAction()

```
*/
 private void listAction() {
  Object[] names = lisp.ENV.keys();
  Object value; String vn;
  writeln("<< names in " + lisp.ENV);
  for (int i=0; i<names.length; i++) {
   value = lisp.ENV.lookup(names[i]);
   writeln( ">> >> " + getName(value.getClass()) + " " +
   names[i] + " = " + prettyPrint(value));
  }
 }

 /**
```

Method: unnameAction()

```
*/
 private void unnameAction() {
  Object[] names = lisp.ENV.keys();
  if (names != null && names.length > 0) {
   Object keyselected = (String)JOptionPane.showInputDialog(null,
                      "Unname", "Select key to delete",
                      JOptionPane.QUESTION_MESSAGE, null,
                      names, names[0] );
   if ( keyselected != null ) {
    expression = "(set! " + keyselected + ")";
    writeln("<< " + expression);
    result = lisp.ENV.set( keyselected, null );
    writeln(">> " + resultToString());
    lisp.ENV.define("@",result);
   }
  }
 }

 /**
```

Method: nameAction()

```
 */
 private void nameAction() {
  if ( result == null) {
   JOptionPane.showMessageDialog(null,"null can not be named!");
  } else {
   String title = cutString("Name for " + resultToString(),32);
   String name = JOptionPane.showInputDialog(title);
   if ( name != null) {
    expression = "(def " + name + " @)";
    writeln("<< " + expression);
    result = lisp.ENV.define(name,result);
    writeln(">> " + resultToString());
    lisp.ENV.define("@",result);
   }
  }
 }

 /**
```

Method: pathAction()

```
 */
 private void pathAction() {
  JFileChooser chooser = new JFileChooser(System.getProperty("user.dir"));
  chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
  if(chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
   File cd = chooser.getSelectedFile();
   try {
    rcl.addURL( cd.toURI().toURL() );
    writeln("<< (path " + cd.toURI().toURL() + ")" );
   } catch(java.net.MalformedURLException mue) { System.err.println(mue); }
  }
 }

 /**
```

Method: treeAction()

```
 */
 private void treeAction() {
  try {
   RClassTree rct = new RClassTree(this,null,rcl);
   writeln("<> Tree from  " + rct.cd.toURI().toURL() );
   writeln("<< (path " + rct.cd.toURI().toURL() + ")");
  } catch(Throwable t) { System.err.println(t); }
 }

 /**
```

Method: `loadAction()`

```
*/
 private void loadAction() {
  writeln("<> Loading from " + logfilename);
  try { readFile(logfilename); } catch (Throwable t) {System.err.println(t);}
  writeln("<> Loaded " + logfilename);
 }

 /**
```

Method: `loadFromAction()`

```
*/
 private void loadFromAction() {
  JFileChooser chooser = new JFileChooser(System.getProperty("user.dir"));
  //chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
  chooser.setFileFilter(new RExtFilter(".log"));
  if(chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
   File cd = chooser.getSelectedFile();
   if ( cd.canRead() ) {
    try {
     String filename = cd.getAbsolutePath();
     writeln("<> Loading from " + filename);
     rcl.addURL( cd.getParentFile().toURL() );
     readFile(filename);
     writeln("<> Loaded " + filename);
    } catch (Throwable t) {System.err.println(t);}
   }
  }
 }

 /**
```

Method: `loadLispAction()`

```
*/
 private void loadLispAction() {
  JFileChooser chooser = new JFileChooser(System.getProperty("user.dir"));
  //chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
  chooser.setFileFilter(new RExtFilter(".lisp"));
  if(chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
   File cd = chooser.getSelectedFile();
   if ( cd.canRead() ) {
    try {
     URL filename = cd.toURI().toURL();
     URL path = cd.getParentFile().toURI().toURL();
     writeln("<< (path " + path.toString() + ")");
     rcl.addURL(path);
     writeln("<< (load " + filename + ")");
     lisp.eval("(load " + filename + ")");
    } catch (Throwable t) {System.err.println(t);}
   }
  }
 }

 /**
```

Method: `saveAction()`

```
*/
 private void saveAction() {
  try {
   writeln("<> Saving to " + logfilename);
   logging = true;
   outfile = new PrintWriter(new BufferedWriter
    (new FileWriter(logfilename,true)));
   writeln("<> Date: " + new Date());
  } catch (Throwable t) {System.err.println(t);} //t.printStackTrace();}
 }

 /**
```

Method: `saveToAction()`

```
*/
 private void saveToAction() {
  JFileChooser chooser = new JFileChooser(System.getProperty("user.dir"));
  if(chooser.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
   File cd = chooser.getSelectedFile();
   String filename = cd.getAbsolutePath();
   if ( filename != null && filename.length() >0 ) {
    logfilename = filename;
    saveAction();
   }
  }
 }

 /**
```

Method: `pdfAction(String)`

The user manual is `RLispMan.pdf`. The code is in `RLispCode.pdf`.

```
*/
 private void pdfAction(String fn) {
  URL ju = this.getClass().getClassLoader().getResource("RLisp.jar");
  if (ju==null)
   ju = this.getClass().getClassLoader().getResource("RLisp/RLispConsole.class");
  try{
   URL fu = new URL(ju,fn);
   String name = fu.getFile().substring(1);
   // name = name.replaceAll("%20"," "); // is Java 1.4
    int i = name.indexOf("%20");
    while (i >= 0) {
     name = (new StringBuffer(name)).replace(i,i+3," ").toString();
     i = name.indexOf("%20");
    }
   Runtime.getRuntime().exec("Start \"/MAX\" \"" + name + "\""); // Windows
   writeln("<> Start \"/MAX\" \"" + name + "\"");
  } catch (Throwable t) {
   writeln("<> File " + fn + " not found!");
   System.err.println(t);
  }
 }

 /**
```

Method: aboutAction()

```
*/
 private void aboutAction() {
  try {
   String[] message = new String[3];
   message[0] = "RLisp " + version;
   message[1] = "© 2004 Ramón Casares";
   message[2] = "r.casares@ieee.org";
   JOptionPane.showMessageDialog(null, message, "About RLisp",
    JOptionPane.INFORMATION_MESSAGE);
   writeln("<> RLisp "+version+ " ("+lisp+")");
  } catch (Throwable t) {System.err.println(t);}
 }

 /**
```

Method: toArray(String)

```
*/
 public static String[] toArray(String sentence) {
  StringTokenizer st = new StringTokenizer(sentence," ");
  int l = st.countTokens();
  String[] word = new String[l];
  for(int i=0; i<l; i++) word[i] = st.nextToken();
  return(word);
 }

 /**
```

Method: cutString(String, int)

```
*/
 public static String cutString(String s, int i) {
  if( s.length() > i ) return(s.substring(0,i)+"...");
  else return(s);
 }

 /**
```

Method: arrayToString(Object[])

```
*/
 public static String arrayToString(Object[] array) {
  String s = "{";
  for(int i=0; i<array.length; i++) {
   if (i>0) s = s + ", ";
   if ( array[i] == null ) s = s + "[null]";
   else s = s + prettyPrint(array[i]);
  }
  s = s + "}";
  return(s);
 }

 /**
```

Method: `prettyPrint(Object)`

```
*/
 public static String prettyPrint(Object o){
  String s = "ERROR!";
  if ( o == null ) s = "[null]";
  else if ( o.getClass().isArray() ) {
   Object[] oa = new Object[Array.getLength(o)];
   for(int i=0; i<oa.length; i++) oa[i] = Array.get(o,i);
   s = arrayToString(oa);
  } else s = o.toString();
  return(s);
 }

 /**
```

Method: `resultToString()`

```
*/
 private String resultToString() { return(prettyPrint(result)); }

 /**
```

Method: `main(String[])`

It creates a console to play the Java `Object`s accessible from the curren directory.

`@param args` are the command line arguments

```
*/
 public static void main(String[] args) {
  RLispConsole rlc = new RLispConsole("RLisp");
  rlc.initLisp();
  if ( args.length > 0 ) {
   String s = "";
   for(int i=0; i<args.length; i++) s = s + " " + args[i];
   s = s.substring(1);
   try {
    rlc.writeln("<> Loading from " + s);
    rlc.readFile(s);
    rlc.writeln("<> Loaded " + s);
    System.out.println("Load from [" + rlc.lisp.counter(0) + "]");
   } catch(IOException ioe) {
    System.err.println("ERROR: file " + s + " not found!");
   }
  }
 }

}
```

2.13 File: RButton.java

```
/**
```

Class: `RButton`

Extends class `javax.swing.JButton` adding an object which can be retrieved by using method `getObject()` or changed with `setObject(Object)`.

`@author` © Ramón Casares 2002

`@version` 2002.08.07

```
*/
package RLisp;

public class RButton extends javax.swing.JButton {

 /**
```

Variable: `rbo` is the object attached to the button.

```
 */
 private Object rbo;


 /**
```

Constructor: `RButton(String, Object)`

Extends `JButton` attaching it an object.

`@param text` is the label of the button

`@param rbo` is the object attached to the button

```
*/
 public RButton(String text, Object rbo) {
  super(text);
  this.rbo = rbo;
 }

 /**
```

Method: `getObject()`

`@return` the object attached to this button

```
*/
 public Object getObject() { return(rbo); }


 /**
```

Method: `setObject(Object)`

Changes the object attached to this button

```
*/
 public void setObject(Object rbo){
  this.rbo = rbo;
 }

}
```

2.14 File: RKeyboard.java

```
/**
```

Class: RKeyboard

An RKeyboard is a window to enter lines of text.

@author © Ramón Casares 2003

@version 2003.01.13

```
*/
package RLisp;

import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

import javax.swing.JTextArea;
import javax.swing.JFrame;
import javax.swing.WindowConstants;
import javax.swing.text.BadLocationException;
import javax.swing.ScrollPaneConstants;
import javax.swing.JScrollPane;
import javax.swing.JButton;
import javax.swing.JToolBar;
import javax.swing.JLabel;
import javax.swing.Box;
import java.awt.Color;
import java.awt.Font;
import javax.swing.BoxLayout;
import java.awt.Container;

public class RKeyboard implements KeyListener, ActionListener {

 private JTextArea ta;
 private JLabel statuslabel;

 /**
```

Variable: callingObject

```
*/
 private ActionListener callingObject;

 /**
```

Variable: endLine is an invisible button that is clicked every time the carriage return is keyed

```
*/
 private RButton endLine;

 /**
```

Variable: inputline saves the last written line

```
*/
```

```
private String inputline;

/**
```

Constructor: RKeyboard(ActionListener, String)

```
*/
RKeyboard(ActionListener callingObject, String firstline) {
 endLine = new RButton("Line", firstline);
 JFrame keyframe = new JFrame("Lisp from Keyboard");
 inputline = firstline;
 this.callingObject = callingObject;
 if (callingObject == null) {
  endLine.addActionListener(this);
  keyframe.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
 } else {
  endLine.addActionListener(callingObject);
  keyframe.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
 }
 keyframe.setBounds(200, 200, 500, 200);
 Container cp = keyframe.getContentPane();
 cp.setLayout(new BoxLayout(cp,BoxLayout.Y_AXIS));

 JToolBar toolBar = new JToolBar();
  JButton jbNesting = new JButton("Nesting");
   jbNesting.addActionListener(this);
   toolBar.add(jbNesting);
  JButton jbWord = new JButton("Word");
   jbWord.addActionListener(this);
   toolBar.add(jbWord);
  JButton jbMax = new JButton("Maximum");
   jbMax.addActionListener(this);
   toolBar.add(jbMax);
  JButton jbMin = new JButton("Minimum");
   jbMin.addActionListener(this);
   toolBar.add(jbMin);
  JButton jbNext = new JButton("Next");
   jbNext.addActionListener(this);
   toolBar.add(jbNext);
  JButton jbPre = new JButton("Previous");
   jbPre.addActionListener(this);
   toolBar.add(jbPre);
  cp.add(toolBar);

 ta = new JTextArea(15,40);
 ta.setEditable(true);
 ta.setLineWrap(false);
 ta.setBackground(new Color(0.5F,1.0F,0.5F));
 ta.setFont(new Font("Monospaced",Font.PLAIN,12));
 cp.add(new JScrollPane(ta,
  ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
  ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED));

 Box statusbox = Box.createHorizontalBox();
 statuslabel = new JLabel();
 statusbox.add(statuslabel);
 statusbox.add(Box.createHorizontalGlue());
 cp.add(statusbox);

 keyframe.setVisible(true);
 keyframe.pack();
 //keyframe.show(); // deprecated
 ta.addKeyListener(this);
 ta.requestFocus();
}

public static int nesting(String text, int p) {
 if ( p > text.length() ) p = text.length();
 int l = 0;
 char c;
 for(int i=0; i<p; i++) { c = text.charAt(i);
  if (c == '(') l++; else if (c == ')') l--;
  if (l < 0) l = 0;
 }
 return(l);
}

public static int prePar(String text, int p, int q) {
 int pp = p;
 int qq = q;
 char c;
 while (qq>0 && pp>0) {
  c = text.charAt(--pp);
  if (c == ')') qq++; else if (c == '(') qq--;
 }
 while (pp>0 && (text.charAt(pp-1) == '\'')) pp--;
 return(pp);
}

public static int nextPar(String text, int p, int q) {
 int pp = p;
 int qq = q;
 char c;
 while (qq>0 && pp<text.length()) {
  c = text.charAt(pp++);
  if (c == ')') qq--; else if (c == '(') qq++;
 }
 return(pp);
}

private static char[] separator = " \t\n\r".toCharArray();

public static String oneLine(String text) {
 while ( text.indexOf(';') != -1 ) {
  int l = text.length();
  int ini = text.indexOf(';');
  int fn = text.indexOf('\n',ini); if (fn==-1) fn = l;
  int fr = text.indexOf('\r',ini); if (fr==-1) fr = l;
  int fin = fn; if (fr<fn) fin = fr;
  text = text.substring(0,ini).concat(text.substring(fin,l));
 }
 for(int i=0; i<separator.length; i++)
```

```
   text = text.replace(separator[i], ' ');
  // text = text.replaceAll(" +"," "); // is Java 1.4
   int i = text.indexOf("  ");
   while (i >= 0) {
    text = (new StringBuffer(text)).replace(i,i+2," ").toString();
    i = text.indexOf("  ");
   }
  return(text);
 }

 public static boolean isInSet(char c, String set) {
  char[] cset = set.toCharArray();
  for(int i=0; i < cset.length; i++) if ( c == cset[i] ) return(true);
  return(false);
 }

 public static String thisWord(String text, int p) {
  if ( p < 0 ) p = 0;
  if ( p >= text.length() ) p = text.length() - 1;
  int ini = p;
  int fin = p;
  while ( ini > 0 &&
          !isInSet( text.charAt(ini-1)," \t\n\r\'()") ) ini--;
  while ( fin < text.length() &&
          !isInSet( text.charAt(fin)," \t\n\r\'()") ) fin++;
  return(text.substring(ini,fin));
 }

 String maxExpression(String text, int p){
  int nl = nesting(text,p);
  if (nl == 0) return(thisWord(text,p));
  else {
   int ini = prePar(text,p,nl);
   int fin = nextPar(text,p,nl);
   if ( ini < 0 || fin < 0 ) return("");
   else return(text.substring(ini,fin));
  }
 }

 public static String minExpression(String text, int p){
  int nl = nesting(text,p);
  if (nl == 0) return(thisWord(text,p));
  else {
   int ini = prePar(text,p,1);
   int fin = nextPar(text,p,1);
   if ( ini < 0 || fin < 0 ) return("");
   else return(text.substring(ini,fin));
  }
 }

 String preExpression(String text, int p){
  return(maxExpression(text,p-1));
 }

 String nextExpression(String text, int p){
  return(maxExpression(text,p+1)); }


 /**
```

Method: `keyTyped(KeyEvent e)`

Implements the `KeyListener` interface. The only method that it is not empty is `keyTyped`.

```
*/
 public void keyTyped(KeyEvent e) {
  char c = e.getKeyChar();
  if ( c == '\n' ) { // new line
   int cp = ta.getCaretPosition();
   int nl = nesting( ta.getText(), cp-1 );
   if ( nl == 0 ) {
    inputline = preExpression( ta.getText() , cp-1 );
    statuslabel.setText(inputline);
    endLine.setObject(oneLine(inputline));
    endLine.doClick();
   } else {
    statuslabel.setText("Nesting = " + nl);
    String ss = "";
    for(int i=0; i<nl; i++) ss = ss + " ";
    ta.insert(ss,cp);
   }
  } else if ( c == ')' ) {
   int cp = ta.getCaretPosition();
   String sta = ta.getText() + ")";
   int nl = nesting(sta, cp+1);
   statuslabel.setText("Nesting = " + nl);
  }
 }

 public void keyPressed(KeyEvent e) {}
 public void keyReleased(KeyEvent e) {}


 /**
```

Method: `actionPerformed(ActionEvent)`

Implements the `ActionListener` interface.

`@param` `e` the action event

```
*/
 public void actionPerformed(ActionEvent e) {
  boolean react = false;
  String texto = e.getActionCommand();
  if ("Line".equals(texto)) { react = false;
   inputline = ((RButton)(e.getSource())).getObject().toString();
  } else if ("Nesting".equals(texto)) { react = false;
   inputline = "Nesting = " + nesting(ta.getText(),ta.getCaretPosition());
  } else if ("Word".equals(texto)) { react = true;
   inputline = thisWord(ta.getText(),ta.getCaretPosition());
  } else if ("Maximum".equals(texto)) { react = true;
```

```
    inputline = maxExpression(ta.getText(),ta.getCaretPosition());
   } else if ("Minimum".equals(texto)) { react = true;
    inputline = minExpression(ta.getText(),ta.getCaretPosition());
   } else if ("Next".equals(texto)) { react = true;
    inputline = nextExpression(ta.getText(),ta.getCaretPosition());
   } else if ("Previous".equals(texto)) { react = true;
    inputline = preExpression(ta.getText(),ta.getCaretPosition());
   } else { react = false;
    System.err.println("ERROR: Action " + texto + " no implemented!");
   }
   statuslabel.setText(inputline);
   if ( callingObject == null || !react ) {
    System.out.println(oneLine(inputline));
   } else {
    endLine.setObject(oneLine(inputline));
    endLine.doClick();
   }
  }


  /**
```

Method: `main(String[])` to test the class

`@param args` are the command line arguments

```
 */
  public static void main(String[] args) {
   RKeyboard k = new RKeyboard(null, "RConsole");
  }

 }
```

2.15 File: RClassTree.java

```
 /**
```

Class: `RClassTree`

Class to select a constructor or a method from a tree. The leaves of the tree are the constructors and methods defined in the classes that are accesible from the selected directory or `jar` file.

`@author` © Ramón Casares 2002

`@version` 2002.08.05

```
 */
 package RLisp;

 import java.net.URL;

 import java.lang.reflect.Array;
 import java.lang.reflect.Field;
 import java.lang.reflect.Constructor;
 import java.lang.reflect.Method;

 import java.io.File;
```

```
import java.util.Vector;
import java.util.Enumeration;

import java.util.jar.JarFile;
import java.util.jar.JarEntry;

import java.awt.Container;
import java.awt.Color;
import javax.swing.BoxLayout;
import javax.swing.JScrollPane;
import javax.swing.JFrame;
import javax.swing.WindowConstants;
import javax.swing.JButton;
import javax.swing.JToolBar;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;

import javax.swing.JFileChooser;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.WindowListener;
import java.awt.event.WindowEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.event.TreeSelectionEvent;

public class RClassTree extends JFrame
 implements ActionListener, TreeSelectionListener {

 /**
```

Variable: `tree`

```
*/
 private JTree tree;

 /**
```

Variable: `callingObject` is the `ActionListener` that receives the action events.

```
*/
 private ActionListener callingObject;

 /**
```

Variable: `ob` is the object selected so far.

```
*/
 private Object ob;

 /**
```

Method: `getSelectedObject()`

```
*/
 public Object getSelectedObject() { return(ob); }

 /**
```

Variable: loader to load classes from any directory

```
*/
 private RClassLoader loader;

 /**
```

Variable: cd is the selected directory or jar file

```
*/
 File cd;
 /**
```

Variable: cds is cd path

```
*/
 String cds;
 /**
```

Variable: cdsl is cds length

```
*/
 int cdsl;
 /**
```

Variable: fileSeparator

```
*/
 char fileSeparator = System.getProperty("file.separator").charAt(0);


 /**
```

Method: setRoot(File)

```
*/
 public int setRoot(File cd) {
  this.cd = cd;
  cds = cd.getAbsolutePath();
  cdsl = cds.length();
  return(cdsl);
 }

 /**
```

Constructor: RClassTree(boolean, RClassLoader)

```
*/
 public RClassTree(File dir, RClassLoader loader) {
  this(null,dir,loader);
 }

 /**
```

Constructor: RClassTree(ActionListener, RClassLoader)

```
*/
 public RClassTree(ActionListener callingObject, RClassLoader loader) {
  this(callingObject,null,loader);
 }

 /**
```

Constructor: `RClassTree(RClassLoader)`

```
*/
 public RClassTree(RClassLoader loader) { this(null,null,loader); }

 /**
```

Constructor: `RClassTree(ActionListener, File, RClassLoader)`

Builds a `JFrame` console with a tree containing a branch for each class in the current directory. For each of these there are two branches, one for the class constructors and the other for the methods.

`@param callingObject` is the `actionListener` object that will receive the action events

`@param dir` is the root directory; `null` means current dir

`@param loader` is the incremental `ClassLoader`

```
*/
 public RClassTree(ActionListener callingObject, File dir, RClassLoader
    loader) {
  super("Class Tree");
  this.loader = loader;
  Container contentPane = this.getContentPane();
  contentPane.setLayout(new BoxLayout(contentPane,BoxLayout.Y_AXIS));

  this.callingObject = callingObject;
  if ( callingObject == null )
   this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
  else
   this.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

  JToolBar toolBar = new JToolBar();
   RButton jbOK = new RButton("OK",this);
    if (callingObject == null) jbOK.addActionListener(this);
    else jbOK.addActionListener(callingObject);
    toolBar.add(jbOK);
    this.rootPane.setDefaultButton(jbOK);
   JButton jbName = new JButton("Name");
    if (callingObject == null) jbName.addActionListener(this);
    else jbName.addActionListener(callingObject);
    toolBar.add(jbName);
   //contentPane.add(toolBar,BorderLayout.NORTH);
   contentPane.add(toolBar);

  DefaultMutableTreeNode top = new DefaultMutableTreeNode("Classes");
  this.tree = new JTree(top);
  tree.addTreeSelectionListener(this);
  //tree.setRootVisible(false);

  JScrollPane treeView = new JScrollPane(tree);
  treeView.setPreferredSize(new java.awt.Dimension(200, 200));

  try{
   if (dir == null) setRoot( chooseRoot() );
   else if ( dir.isDirectory() ) setRoot(dir);
   if (cd == null) setRoot( new File(System.getProperty("user.dir")) );

   loader.addURL( cd.toURL() );
   populateTree(top, cd);

  } catch (Throwable t) {
   System.err.println(t);
   if ( callingObject == null ) System.exit(0);
   else { this.dispose(); return; }
  }
  contentPane.add(treeView);
  //tree.setBackground(new Color(1.0F,1.0F,0F)); // yellow
  this.setVisible(true);
  this.pack();
 }


 /**
```

Method: `chooseRoot()`

```
 */
 File chooseRoot() {
  JFileChooser chooser = new JFileChooser();
  chooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
  chooser.setFileFilter(new RExtFilter(".jar"));
  chooser.setSelectedFile(new File(System.getProperty("user.dir")));
  int returnVal = chooser.showOpenDialog(null);
  if( returnVal == JFileChooser.APPROVE_OPTION) {
   return( chooser.getSelectedFile() );
  }
  return(null);
 }

 /**
```

Method: `populateTree(DefaultMutableTreeNode, File)`

```
 */
 private void populateTree(DefaultMutableTreeNode top, File cd) throws
    Throwable {
  if ( cd.isFile() ) {
   String fn = cd.getAbsolutePath();
   int lp = fn.lastIndexOf('.');
   if ( lp >= 0 ) {
    String ext = fn.substring(lp);
    int pr = cdsl; if ( fn.charAt(cdsl) == fileSeparator ) pr++;
    String name = fn.substring(pr,lp).replace(fileSeparator,'.');
    if ( ".class".equals(ext) ) addClassToTree(top, name);
    if ( ".jar".equals(ext) )   addJarToTree(top, cd, name);
   }
  } else if ( cd.isDirectory() ) {
   System.err.println( cd.toURL().toString() );
   DefaultMutableTreeNode dir = new DefaultMutableTreeNode(cd);
   top.add(dir);
   File[] ls = cd.listFiles();
   for(int i=0; i<ls.length; i++) populateTree(dir, ls[i]);
  }
 }

  /**
```

Method: addClassToTree(DefaultMutableTreeNode, String)

```
*/
  void addClassToTree(DefaultMutableTreeNode top, String classname) {
   DefaultMutableTreeNode mClass;
   DefaultMutableTreeNode mArray;
   DefaultMutableTreeNode mFields = null;
    Field[] fields = null;
   DefaultMutableTreeNode mNew = null;
    Constructor[] constructors = null;
   DefaultMutableTreeNode mMethods = null;
    Method[] methods = null;
   Class c = null;
   try {
    c = loader.loadClass(classname);
    mClass = new DefaultMutableTreeNode(classname);
    top.add(mClass);
    mArray = new DefaultMutableTreeNode("Array");
    mClass.add(mArray);
    mArray.add( new DefaultMutableTreeNode( Array.newInstance(c,1) ) );
    fields = c.getFields();
    if ( fields.length > 0 ) {
     mFields = new DefaultMutableTreeNode("Fields");
     mClass.add(mFields);
     for (int j=0; j<fields.length; j++) {
      mFields.add( new DefaultMutableTreeNode( fields[j] ) );
     }
    }
    constructors = c.getConstructors();
    if ( constructors.length > 0 ) {
     mNew = new DefaultMutableTreeNode("Constructors");
     mClass.add(mNew);
     for (int j=0; j<constructors.length; j++) {
      mNew.add( new DefaultMutableTreeNode( constructors[j] ) );
     }
    }
    methods = c.getMethods();
    if ( methods.length > 0 ) {
     mMethods = new DefaultMutableTreeNode("Methods");
     mClass.add(mMethods);
     for (int j=0; j<methods.length; j++) {
      mMethods.add( new DefaultMutableTreeNode( methods[j] ) );
     }
    }
   } catch (java.lang.NoClassDefFoundError ncdfe) {
    System.err.println("Class "+ ncdfe.getMessage() + " not loadable!");
   } catch (Throwable t) { System.err.println(t); }
  }

  /**
```

Method: addJarToTree(DefaultMutableTreeNode, File, String)

```
*/
  void addJarToTree(DefaultMutableTreeNode top, File cd, String classname)
    throws Throwable {
   DefaultMutableTreeNode jd = new DefaultMutableTreeNode(cd);
   top.add(jd);
   JarFile jar = new JarFile(cd);
   Enumeration<JarEntry> jee = jar.entries();
   JarEntry je = null;
   String jen = null;
   while ( jee.hasMoreElements() ) {
    je = jee.nextElement();
    jen = je.getName(); // System.out.println(jen);
    if ( !je.isDirectory() && jen.lastIndexOf('.') >= 0 &&
        ".class".equals(jen.substring(jen.lastIndexOf('.'))) )
     addClassToTree(jd, jen.substring(0,jen.lastIndexOf('.')).replace('/','.')
    );
   }
  }


 /**
```

Method: actionPerformed(ActionEvent)

Implements the `ActionListener` interface. Usually the `callingObject` gets the action events and this implementation is only for testing purposes.

`@param` `e` the action event

```
*/
 public void actionPerformed(ActionEvent e) {
  String texto = e.getActionCommand();
  if ("OK".equals(texto)) System.out.println(ob.toString());
  else if ("Name".equals(texto)) System.out.println("Name");
  else System.err.println("ERROR: Action " + texto + " no implemented!");
 }


 /**
```

Method: valueChanged(TreeSelectionEvent)

Implements the `TreeSelectionListener` interface. Updates the `ob` object.

`@param` `e` the tree selection event

```
*/
 public void valueChanged(TreeSelectionEvent e){
  DefaultMutableTreeNode node = (DefaultMutableTreeNode)
                                tree.getLastSelectedPathComponent();
  if (node == null) return;
  //Object nodeInfo = node.getUserObject();
  //if (node.isLeaf()) { ob = nodeInfo; }
  ob = node.getUserObject();
 }


 /**
```

Method: `main(String[])`

Just for testing this class.

`@param args` command line options are ignored

```
*/
 public static void main(String[] args) {
  boolean choosing = true;
  if ( args.length > 0 && args[0].equals("false") ) choosing = false;
  URL[] urls = new URL[1];
  File ud = new File(System.getProperty("user.dir"));
  try { urls[0] = ud.toURL(); }
  catch (java.net.MalformedURLException mue) {} // always right
  RClassLoader rcl = new RClassLoader(urls);
  RClassTree frame = new RClassTree(rcl);
 }

}
```

2.16 File: RExtFilter.java

```
/**
```

Class: `RExtFilter`

Filters files having the extension defined when building the object.

It can be used as a `javax.swing.filechooser.FileFilter` for a `JFileChooser`, and also as a `FilenameFilter` or `java.io.FileFilter` for `File.list(filter)` method.

`@author` © Ramón Casares 2001

`@version` 2001.08.29

```
*/
package RLisp;

import java.io.File;
import java.io.FilenameFilter;

public class RExtFilter extends javax.swing.filechooser.FileFilter
 implements FilenameFilter, java.io.FileFilter {

 /**
```

Variable: `ext` contains the extension

```
*/
 String ext;

 /**
```

Constructor: `RExtFilter(String)`

`@param ext` is the extension, starting with a dot

```
*/
 public RExtFilter(String ext) { this.ext = ext; }


 /**
```

Method: `accept(File, String)`

Tests if the specified file should be included in a file list. Implements the `FilenameFilter` interface.

`@param dir` the directory in which the file was found

`@param name` the name of the file

`@return` true if and only if the name should be included in the file list; false otherwise

```
*/
 public boolean accept(File dir, String name) {
  String ext = "";
  if ( name.lastIndexOf('.') > 0 )
   ext = name.substring( name.lastIndexOf('.') );
  return ( this.ext.equals(ext) );
 }

 /**
```

Method: `accept(File)`

Tests if the specified file should shown by a file chooser. Implements the `FileFilter` interface. Overrides the `FileFilter` class method `accept(File)`.

`@param file` is the file to show, or not to show

`@return` true if and only if the file should be shown, false otherwise

```
*/
 public boolean accept(File file) {
  if ( file.isFile() ) {
   String name = file.getPath();
   String ext = "";
   if ( name.lastIndexOf('.') > 0 )
    ext = name.substring( name.lastIndexOf('.') );
   return ( this.ext.equals(ext) );
  } else return(true);
 }

 /**
```

Method: `getDescrition()`

Overrides the `FileFilter` class method `getDescription()`.

`@return` a readable description of the filter

```
*/
 public String getDescription() {
  if ( ext.length() > 1 )
   return( ext.substring(1) + " files" );
  else return("ERROR: Extension not yet defined!");
 }

}
```

2.17 File: RClassLoader.java

```
/**
```

Class: `RClassLoader`

Class `RClassLoader` extends `URLClassLoader` making method `addURL(URL)` `public`, and so `RClassLoader` is an incremental `ClassLoader`.

`@author` © Ramón Casares 2003

`@version` 2003.03.22

```
*/
package RLisp;

import java.net.URL;
import java.net.URLClassLoader;

public class RClassLoader extends URLClassLoader {

 /**
```

Constructor: `RClassLoader(URL[])`

```
*/
 public RClassLoader(URL[] urls) { super(urls); }

 /**
```

Method: `addURL(URL)`

Makes `public` the `protected` `super.addURL(URL)` method.

It first checks if the `url` has been already loaded, because the `super` method doesn't.

`@param` `url` is the URL incremented

```
*/
 public void addURL(URL url) {
  if ( url == null ) return;
  URL[] urls = this.getURLs();
  boolean isNew = true;
  for (int i=0; i<urls.length; i++) isNew = isNew && !url.equals(urls[i]);
  if ( isNew ) super.addURL(url);
 }

}
```

2.18 File: RLisp.log

```
<< (new RLisp.RPair (string (1 2 3))
>> (1 2 3)
<< (def lista123 @)
>> lista123
<< (method lista123 getClass)
>> class RLisp.RPair
```

2.19 File: RLisp2jar.bat

```
cd ..
javac RLisp/RPair.java
javac RLisp/RFrame.java
javac RLisp/REnvironment.java
javac RLisp/RLisp.java
javac RLisp/RLispJava.java
javac RLisp/RLispInterpreter.java
javac RLisp/RLispConsole.java
javac RLisp/RButton.java
javac RLisp/RKeyboard.java
javac RLisp/RClassTree.java
javac RLisp/RExtFilter.java
javac RLisp/RClassLoader.java
jar cf RLisp/RLisp.jar RLisp/RPair.class RLisp/RPair.java
jar uf RLisp/RLisp.jar RLisp/RFrame.class RLisp/RFrame.java
jar uf RLisp/RLisp.jar RLisp/REnvironment.class RLisp/REnvironment.java
jar uf RLisp/RLisp.jar RLisp/RLisp.class RLisp/RLisp.java
jar uf RLisp/RLisp.jar RLisp/RLispJava.class RLisp/RLispJava.java
jar uf RLisp/RLisp.jar RLisp/RLispInterpreter.class RLisp/RLispInterpreter.java
jar uf RLisp/RLisp.jar RLisp/RLispConsole.class RLisp/RLispConsole.java
jar uf RLisp/RLisp.jar RLisp/RButton.class RLisp/RButton.java
jar uf RLisp/RLisp.jar RLisp/RKeyboard.class RLisp/RKeyboard.java
jar uf RLisp/RLisp.jar RLisp/RClassTree.class RLisp/RClassTree.java
jar uf RLisp/RLisp.jar RLisp/RExtFilter.class RLisp/RExtFilter.java
jar uf RLisp/RLisp.jar RLisp/RClassLoader.class RLisp/RClassLoader.java
echo jar uf RLisp/RLisp.jar RLisp/RLisp.tex
echo jar uf RLisp/RLisp.jar RLisp/RLisp.pdf
jar uf RLisp/RLisp.jar RLisp/RLisp.log
jar uf RLisp/RLisp.jar RLisp/RLisp.lisp
jar uf RLisp/RLisp.jar RLisp/RLispJava.lisp
jar uf RLisp/RLisp.jar RLisp/RLispArray.lisp
jar uf RLisp/RLisp.jar RLisp/RLispMaths.lisp
jar uf RLisp/RLisp.jar RLisp/Primes.lisp
jar uf RLisp/RLisp.jar RLisp/RLisp2jar.bat
echo Main-Class: RLisp/RLispConsole> RLisp.MF
echo Class-Path: .\ RLisp.jar>> RLisp.MF
jar umf RLisp.MF RLisp/RLisp.jar
del RLisp.MF
cd RLisp
```

3 To-Do List

3.1 To enhance the list command

It should be possible to choose an object from the list of named (with `name`) objects and then, by doing it, the methods that we can apply to it were shown, so one of the methods could be selected. This is to do with the named objects the same thing already done with `RClassTree`.

3.2 To enhance the session command

It is dangerous to run a `session()` when there is no console, which is the case when the Java Virtual Machine was call by `javaw`. Because of this, it would be better that this option were not activated in these circumstances. But I don't know how a Java class can learn in run-time whether there is a system console or not.

3.3 To enhance the input of arrays

If, for example, function `main(String[])` is chosen in `Tree`, it is not possible to input the argument, except when there is already a named object which is a `String[]`. One solution is `(array String This is not a String)`, which is a `String array`. Note that `(array String)` evaluates to the `null String array`.

3.4 To add edit buttons to the keybord

Although, at least in Windows, you can use the Ctrl-X, Ctrl-C and Ctrl-V key combinations, it would be nice to have also the Cut, Copy and Paste buttons in the keyboard window toolbar.

4 Legalities

Java Index

Table of Contents