

# 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 dictionary, 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 accessible 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](#)

```
1 /**
```

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 phrases 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

```
23 */
24 package RLisp;
25
26 import java.util.StringTokenizer;
27 import java.util.Vector;
28
29 public class RPair {
30
31 /**
```

Variable: `car`

```
31 */
32 public Object car;
33 /**
```

Variable: `cdr`

```
33 */
34 public Object cdr;
35
36 /**
```

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

```
36 */
37 public RPair() { car = null; cdr = null; }
38
39 /**
```

Variable: `nil` is the () list

```
39 */
40 public static final RPair nil = new RPair();
41
42 /**
```

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

```
42 */
43 public boolean isNil() { return( car == null && cdr == null); }
44
45 /**
```

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

```
45 */
46 public static boolean isNil(Object o) {
47     if ( o instanceof RPair )
48         return( ((RPair)o).car == null && ((RPair)o).cdr == null );
49     else return(false);
50 }
51
52
53 /**
```

*Constructor:* `RPair(Object, Object)`

```
53 */
54 public RPair(Object car, Object cdr) {
55     this.car = car; this.cdr = cdr;
56 }
57
58 /**
```

*Constructor:* `RPair(String)`

```
58 */
59 public RPair(String s) {
60     Object o = tokenize(s);
61     if ( isRPair(o) ) {
62         this.car = ((RPair)o).car;
63         this.cdr = ((RPair)o).cdr;
64     } else {
65         this.car = "ERROR!";
66         this.cdr = o;
67     }
68 }
69
70 /**
```

*Variable:* `delims`

```
70 */
71 static final String delims = "( ',,\n\r\t_>";
72
73 /**
```

*Method:* `Tokenize(String)`

It exhausts the String returning an array with the objects.

```
76 */
77 public static Object[] Tokenize(String s) {
78     Vector<Object> V = new Vector<Object>();
79     StringTokenizer T = new StringTokenizer(s,delims,true);
80     while( T.hasMoreTokens() ) V.add(tokenize(T, null));
```

```

81   return(V.toArray());
82 }
83
84 /**

```

### Method: `tokenize(String)`

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

```

88 */
89 public static Object tokenize(String s) {
90   StringTokenizer T = new StringTokenizer(s,delims,true);
91   return(tokenize(T, null));
92 }
93 private static Object tokenize(StringTokenizer T, RPair v) {
94   String t;
95   while ( T.hasMoreTokens() ) { t = T.nextToken();
96     if ( " ".equals(t) || "\n".equals(t) || "_".equals(t) ||
97         "\r".equals(t) || "\t".equals(t) ) {}
98     else if ( "(" .equals(t) ) {
99       if (v==null) return(tokenize(T, new RPair()));
100      else v.add(tokenize(T, new RPair()));
101    }
102    else if ( ")" .equals(t) ) { return(v); }
103    else if ( "," .equals(t) ) { v.add(tokenize(T,null), false); }
104    else if ( ";" .equals(t) ) {
105      while( T.hasMoreTokens() ) { t = T.nextToken();
106        if ( "\n".equals(t) || "\r".equals(t) ) break;
107      }
108    }
109    else if ( "\"" .equals(t) ) {
110      Object arg = tokenize(T,null);
111      if (v==null) return(new RPair("quote",new RPair(arg,null)));
112      else v.add(new RPair("quote",new RPair(arg,null)));
113    }
114    else if (v==null) return(t); else v.add(t);
115  }
116  return(v);
117 }
118 private Object add(Object last){
119   if (last == null) return(null);
120   else if (isNil()) { car = last; return(last); }
121   else if (cdr == null) { cdr = new RPair(last,null); return(last); }
122   else if (cdr instanceof RPair) return( ((RPair)cdr).add(last) );
123   else return(null);
124 }
125 private Object add(Object last, boolean properly){
126   if (properly) return(add(last));
127   else if (isNil(last)) return(last);
128   else if (cdr == null) { cdr = last; return(last); }
129   else if (cdr instanceof RPair) return( ((RPair)cdr).add(last,false) );
130   else return(null);
131 }
132
133 /**

```

### Method: `isRPair(Object)`

```
133 */
134 public static boolean isRPair(Object o) {
135     return(o instanceof RPair);
136 }
137
138 /**
```

Method: [isAtom\(Object\)](#)

```
138 */
139 public static boolean isAtom(Object o) {
140     return( isNil(o) || !(o instanceof RPair) );
141 }
142
143 /**
```

Method: [isList\(Object\)](#)

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

```
147 */
148 public static int isList(Object o) {
149     if(o == null) return(-1);
150     if ( o instanceof RPair ) {
151         RPair p = (RPair)o;
152         if ( p.car == null )
153             if ( p.cdr == null ) return(0); else return(-1);
154         else
155             if ( p.cdr == null ) return(1); else {
156                 int l = isList(p.cdr);
157                 if (l<0) return(1); else return(1+l);
158             }
159     } else return(-1);
160 }
161
162
163 /**
```

Method: [toArray\(\)](#)

```
163 */
164 public Object[] toArray() {
165     int l = isList(this);
166     if ( l >= 0 ) {
167         Object[] a = new Object[l];
168         for(int i=0; i<l; i++) a[i] = nth(i);
169         return(a);
170     } else return(null);
171 }
172
173 /**
```

Method: [toArray\(Object\)](#)

```
173 */
174 public static Object[] toArray(Object o) {
175     int l = isList(o);
```

```
176     if ( l >= 0 ) {
177         Object[] a = new Object[l];
178         for(int i=0; i<l; i++) a[i] = nth(o,i);
179         return(a);
180     } else return(null);
181 }
182
183
184 /**
```

*Method:* `cons(Object, RPair)`

```
184 */
185 public static RPair cons(Object car, RPair cdr) {
186     if ( isNil(cdr) ) return( new RPair(car,null) );
187     else                 return( new RPair(car,cdr) );
188 }
189
190 /**
```

*Method:* `car()`

```
190 */
191 public Object car() { return( car ); }
192 /**
```

*Method:* `cdr()`

```
192 */
193 public Object cdr() { return( cdr ); }
194
195 /**
```

*Method:* `Cdr()`

```
195 */
196 public Object Cdr() {
197     if (cdr == null) return(nil); else return(cdr);
198 }
199
200 /**
```

*Method:* `CDR()`

```
200 */
201 public RPair CDR() {
202     if (cdr == null) return(nil);
203     if (cdr instanceof RPair) return( ((RPair)cdr) );
204     else return(null);
205 }
206
207 /**
```

*Method:* `nth(int)`

@param n is the position (car is 0)

@return the object in the n position

```
211 */
```

```
212 public Object nth(int n) {
213     if (n < 0) return(null);
214     else if (n == 0) return(car);
215     else if (cdr instanceof RPair) return( ((RPair)cdr).nth(n-1) );
216     else return(null);
217 }
218
219 /**
```

Method: `nth(Object, int)`

```
219 */
220 public static Object nth(Object o, int n) {
221     if ( o == null || n < 0 ) return(null);
222     if (o instanceof RPair)
223         if (n == 0) return(((RPair)o).car);
224         else return( nth(((RPair)o).cdr,n-1) );
225     else return(null);
226 }
227
228 /**
```

Method: `substitute(Object, Object)`

```
228 */
229 public RPair substitute(Object oldo, Object newo) {
230     Object newcar = null;
231     if (car == null) { if (oldo == null) newcar = newo; }
232     else {
233         if ( car.equals(oldo) ) newcar = newo;
234         else if (car instanceof RPair)
235             newcar = ((RPair)car).substitute(oldo,newo);
236         else newcar = car;
237     }
238     Object newcdr = null;
239     if (cdr == null) { if (oldo == null) newcdr = newo; }
240     else {
241         if ( cdr.equals(oldo) ) newcdr = newo;
242         else if (cdr instanceof RPair)
243             newcdr = ((RPair)cdr).substitute(oldo,newo);
244         else newcdr = cdr;
245     }
246     return(new RPair(newcar,newcdr));
247 }
248
249 private boolean loop = false;
250
251 /**
```

Method: `toString()`

```
251 */
252 public String toString() {
253     if (loop) return("RPair"+hashCode());
254     else {
255         loop = true;
256         String s = "(" + toStringWOP() + ")";
257         loop = false;
```

```

258     return(s);
259 }
260 }
261 private String toStringWOP() {
262     String s;
263     if ( car == null ) s = "";
264     else                s = car.toString();
265     if ( cdr == null ) return(s);
266     else if (cdr instanceof RPair) return(s+" "+((RPair)cdr).toStringWOP());
267     else return( s + " , " + cdr.toString() );
268 }
269
270 /**

```

Method: [toString\(boolean\)](#)

```

270 */
271 public String toString(boolean wp) {
272     if (wp) return(toString());
273     else return(toStringWOP());
274 }
275
276 /**

```

Method: [equals\(Object\)](#)

```

276 */
277 public boolean equals(Object o) {
278     if ( o == null ) return( this.car == null && this.cdr == null );
279     else if (o instanceof RPair) {
280         RPair p = (RPair)o;
281         return ( ( p.car == this.car ||
282                 ( p.car != null && p.car.equals(this.car)) ) &&
283                 ( p.cdr == this.cdr ||
284                 ( p.cdr != null && p.cdr.equals(this.cdr)) ) ); }
285     else return(false);
286 }
287
288 }

```

## 2.2 File: [RFrame.java](#)

```

1 /**

```

Class: [RFrame](#)

Implements a frame.

A frame is a dictionary, 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

```

13 */
14 package RLisp;

```



```
15
16 import java.util.Hashtable;
17 import java.util.Enumeration;
18
19 public class RFrame {
20
21     private Hashtable<Object,Object> ht;
22
23     /**
```

*Constructor: RFrame()*

```
23 */
24     public RFrame() { ht = new Hashtable<Object,Object>(); }
25
26     /**
```

*Constructor: RFrame(Object, Object)*

```
26 */
27     public RFrame(Object keys, Object values) {
28         ht = new Hashtable<Object,Object>(); bind(keys,values);
29     }
30
31     /**
```

*Method: bind(Object, Object)*

```
31 */
32     public Object bind(Object key, Object value) {
33         if ( key == null ) return(null);
34         else if ( RPair.isRPair(key) ) {
35             if ( RPair.isRPair(value) ) {
36                 bind(((RPair)key).car(),((RPair)value).car());
37                 bind(((RPair)key).cdr(),((RPair)value).Cdr()); // note: Cdr
38             } else return(null);
39         } else { if ( value != null) ht.put(key,value); else ht.remove(key); }
40         return(lookup(key));
41     }
42
43     /**
```

*Method: lookup(Object)*

```
43 */
44     public Object lookup(Object key) {
45         if ( key == null ) return(null); else return(ht.get(key));
46     }
47
48     /**
```

*Method: keys()*

```
48 */
49     public Object[] keys() {
50         Object[] ka = new Object[ ht.size() ];
51         Enumeration ke = ht.keys();
52         int i = 0;
53         while ( ke.hasMoreElements() ) ka[i++] = ke.nextElement();
```

```
54     return( ka );
55 }
56
57 private boolean loop = false;
58
59 /**
```

*Method:* `toString()`

```
59 */
60 public String toString() {
61     if ( loop ) return( "RFrame"+hashCode() );
62     else {
63         loop = true;
64         String sr = ht.toString();
65         loop = false;
66         return(sr);
67     }
68 }
69
70 }
```

### 2.3 File: [REnvironment.java](#)

```
1 /**
```

**Class:** `REnvironment`

Implements an environment.

An environment is a list of frames. Each frame is a dictionary, 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

```
14 */
15 package RLisp;
16
17 public class REEnvironment extends RPair {
18
19     /**
```

*Constructor:* `REnvironment()`

```
19 */
20 public REEnvironment() { super(); }
21
22 /**
```

*Constructor:* `REnvironment(RFrame)`

```
22 */
23 public REEnvironment(RFrame rf) { super(rf,null); }
24
25 /**
```

*Constructor:* `REnvironment(RFrame, REEnvironment)`

```
25 */
26 public REEnvironment(RFrame rf, REEnvironment re) { super(rf,re); }
27
28 /**
```

*Method:* `extend(RFrame)`

```
28 */
29 public REEnvironment extend(RFrame rf) {
30     return(new REEnvironment(rf,this));
31 }
32
33 /**
```

*Method:* `firstFrame()`

```
33 */
34 public RFrame firstFrame() { return( (RFrame)car() ); }
35 /**
```

*Method:* `restFrames()`

```
35 */
36 public REEnvironment restFrames() { return( (REEnvironment)cdr() ); }
37
38 /**
```

*Method:* `lookup(Object)`

```
38 */
39 public Object lookup(Object key) {
40     if ( this.isNil() ) return(null);
41     Object val = firstFrame().lookup(key);
42     if ( val != null ) return(val);
43     else {
44         REEnvironment rest = restFrames();
45         if ( rest == null ) return(null);
46         else return( rest.lookup(key) );
47     }
48 }
49
50 /**
```

*Method:* `define(Object, Object)`

```
50 */
51 public Object define(Object key, Object value) {
52     if ( this.isNil() ) return(null);
53     else {
54         firstFrame().bind(key,value);
55         return(key);
56     }
57 }
58
59 /**
```

*Method:* `set(Object, Object)`

```
59 */
```

```
60 public Object set(Object key, Object value) {
61     if ( this.isNil() ) return(null);
62     else {
63         RFrame f = firstFrame();
64         Object val = f.lookup(key);
65         if ( val != null ) { f.bind(key,value); return(value); }
66         else if ( restFrames() == null ) { return(null); }
67         else return( restFrames().set(key,value) );
68     }
69 }
70
71
72 private boolean loop = false;
73
74 /**
```

Method: toString(boolean)

```
74 */
75 public String toString(boolean all) {
76     if (all) return( toString() );
77     else return("ENV"+hashCode()); }
78
79 /**
```

Method: toString()

```
79 */
80 public String toString() {
81     REEnvironment rest = restFrames();
82     if ( rest == null ) { return("GlobalENV"); }
83     else if (loop) { return("ENV"+ hashCode()); }
84     else {
85         loop = true;
86         String s = "(ENV"+ hashCode() + ": " + firstFrame().toString() +
87                 "->" + rest.toString() + ")";
88         loop = false;
89         return(s);
90     }
91 }
92
93 /**
```

Method: list()

```
93 */
94 public String[] list() {
95     RFrame first = firstFrame();
96     if (first == null) return(null);
97     Object[] ka = first.keys();
98     int l = ka.length;
99     String[] sa = new String[ l ];
100    for(int i=0; i<l; i++)
101        sa[i] = ka[i].toString() + "=" + first.lookup(ka[i]).toString();
102    return(sa);
103 }
104
105
106 /**
```

Method: `keys()`

```

106 */
107 public Object[] keys() {
108     RFrame first = firstFrame();
109     if (first == null) return(null); else return(first.keys());
110 }
111
112 /**

```

Method: `keys(Class, boolean)`

```

112 */
113 public Object[] keys(Class c, boolean strict) {
114     RFrame first = firstFrame();
115     if (first == null || c == null) return(null);
116     else if ( c == Boolean.TYPE ) c = Boolean.class;
117     else if ( c == Character.TYPE ) c = Character.class;
118     else if ( c == Byte.TYPE ) c = Byte.class;
119     else if ( c == Short.TYPE ) c = Short.class;
120     else if ( c == Integer.TYPE ) c = Integer.class;
121     else if ( c == Long.TYPE ) c = Long.class;
122     else if ( c == Float.TYPE ) c = Float.class;
123     else if ( c == Double.TYPE ) c = Double.class;
124     Object[] names = first.keys();
125     java.util.Vector<Object> sel = new java.util.Vector<Object>();
126     Object n; Object o;
127     for(int i=0; i<names.length; i++) {
128         n = names[i];
129         o = first.lookup(n);
130         if (strict) { if ( o.getClass() == c) sel.addElement(n); }
131         else          { if ( c.isInstance(o) ) sel.addElement(n); }
132     }
133     return(sel.toArray());
134 }
135
136
137 }

```

## 2.4 File: RLisp.java

```

1 /**

```

Class: `RLisp`

Implements a minimum Lisp evaluator.

@author © Ramón Casares 2003

@version 2003.02.05

```

7 */
8 package RLisp;
9
10 public class RLisp {
11
12     /**

```

Variable: `t` is the true constant

```

12 */
13 public static final Boolean t = new Boolean("true");
14 /**

```

Method: `isTrue(Object)`

```

14 */
15 public static boolean isTrue(Object o) {
16     return( o != null && o instanceof Boolean &&
17            ((Boolean)o).booleanValue() );
18 }
19
20 /**

```

Variable: `counter` counts eval cycles

```

20 */
21 public int counter = 0;
22
23 /**

```

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 ....`

```

43 */
44 public Object eval(Object exp, REnvironment env) { counter++;
45     if ( exp == null ) return(null);
46     if ( RPair.isAtom(exp) ) {
47         Object v = env.lookup(exp);
48         if ( v == null ) return(exp); // self-evaluating
49         else return(v); // identifier
50     } else { RPair re = (RPair)exp;
51         if ( isSpecial(re) ) return( evalSpecial(re,env) ); // hook to extend
52         else if ( "quote".equals(re.car()) ) return( re.CDR().car() );
53         else if ( "eval".equals(re.car()) ) return( evalEval(re,env) );

```

```

54     else if ( "set!".equals(re.car()) ) return( evalSet(re,env) );
55     else if ( "def".equals(re.car()) ) return( evalDef(re,env) );
56     else if ( "cond".equals(re.car()) ) return( evalCond(re,env) );
57     else if ( "lambda".equals(re.car()) ) return( evalLambda(re,env) );
58     else if ( "rho".equals(re.car()) ) return( evalRho(re,env) );
59     else return( apply(re,env) ); // function application
60   }
61 }
62
63 /**

```

*Method: [isSpecial\(RPair\)](#): Override to define new special forms (as and)*

```

65 */
66 boolean isSpecial(RPair p) { return(false); }
67
68 /**

```

*Method: [evalSpecial\(RPair, REnvironment\)](#): Override to define new special forms (as and)*

```

70 */
71 Object evalSpecial(RPair p, REnvironment env) { return(null); }
72
73 /**

```

*Method: [evalEval\(RPair, REnvironment\)](#)*

```

73 */
74 Object evalEval(RPair e, REnvironment env) {
75   Object exp = eval(e.nth(1),env);
76   Object xenv = eval(e.nth(2),env);
77   if ( exp == null ) return(null);
78   else if ( xenv == null ) return( eval(exp,env) );
79   else if ( env.getClass().isInstance(xenv) )
80     return( eval(exp,(REnvironment)xenv) );
81   else return(null);
82 }
83
84 /**

```

*Method: [evalSet\(RPair, REnvironment\)](#)*

```

84 */
85 Object evalSet(RPair e, REnvironment env) {
86   Object key = e.nth(1);
87   if ( key == null ) return(null);
88   else {
89     Object val = eval(e.nth(2),env);
90     return( env.set(key,val) );
91   }
92 }
93
94 /**

```

*Method: [evalDef\(RPair, REnvironment\)](#)*

```

94 */
95 Object evalDef(RPair e, REnvironment env) {

```

```

96   Object key = e.nth(1);
97   if ( key == null ) return(null);
98   else {
99     Object val = eval(e.nth(2),env);
100    return( env.define(key,val) );
101  }
102 }
103
104 /**

```

*Method:* `evalCond(RPair, REnvironment)`

```

104 */
105 Object evalCond(RPair e, REnvironment env) {
106   return( evalClauses(e.CDR(),env) );
107 }
108 private Object evalClauses(RPair cl, REnvironment env) {
109   if (cl == null) return(null);
110   Object car = cl.car();
111   if ( car == null ) return(null);
112   else {
113     if ( !RPair.isRPair(car) || RPair.isNil(car) ) return(null);
114     else {
115       RPair first = (RPair)car;
116       if ( isTrue(eval(first.car(),env)) )
117         return(evalSequence(first,env));
118       else return(evalClauses(cl.CDR(),env));
119     }
120   }
121 }
122
123 /**

```

*Method:* `evalSequence(RPair, REnvironment)`

```

123 */
124 Object evalSequence(RPair e, REnvironment env) {
125   if (e == null) return(null);
126   if ( e.cdr() == null ) return( eval(e.car(),env) );
127   else {
128     eval(e.car(),env); // for side-effects
129     return( evalSequence(e.CDR(),env) );
130   }
131 }
132
133 /**

```

*Method:* `evalLambda(RPair, REnvironment)`

```

133 */
134 Object evalLambda(RPair e, REnvironment env) {
135   return( RPair.cons("LAMBDA", RPair.cons(env, e.CDR())) );
136 }
137
138 /**

```

*Method:* `evalRho(RPair, REnvironment)`



```

138 */
139 Object evalRho(RPair e, REnvironment env) {
140     return( RPair.cons("RHO", e.CDR()) );
141 }
142
143 /**

```

*Method:* `evalRPair(RPair, REnvironment)`

```

143 */
144 RPair evalRPair(RPair e, REnvironment env) {
145     if (e == null) return(null);
146     if (e.isNil()) return(RPair.nil);
147     else return( RPair.cons( eval(e.car(),env), evalRPair(e.CDR(),env) ) );
148 }
149
150 /**

```

*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 application:

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

```

168 */
169 Object apply(RPair e, REnvironment env) {
170     Object op = eval(e.car(),env);
171     if ( op == null ) {
172         System.err.println("ERROR: null function!");
173         return(null);
174     } else {
175         if ( RPair.isRPair( op ) )
176             return( applyCompound( (RPair)op,e.CDR(),env) );
177         else if ( isPrimitive(op) )
178             return( applyPrimitive(op, evalRPair(e.CDR(),env)) );
179         else {
180             System.err.println("ERROR: " + op + " undefined!");
181             return(null);
182         }
183     }
184 }
185
186 /**

```

Method: `isPrimitive(Object)`

```

186 */
187 boolean isPrimitive(Object op) {
188     return( "cons".equals(op) || "car".equals(op) || "cdr".equals(op) ||
189           "atom?".equals(op) || "eq?".equals(op) );
190 }
191
192 /**

```

Method: `applyPrimitive(Object, RPair)`

```

192 */
193 Object applyPrimitive(Object op, RPair args) {
194     Object first = args.nth(0);
195     if ( "cons".equals(op) ) { Object second = args.nth(1);
196         if ( RPair.isRPair(second) ) return( RPair.cons(first, (RPair)second) );
197         else return(new RPair(first, second));
198     } else if ( "car".equals(op) ) {
199         if ( RPair.isRPair(first) ) return( ((RPair)first).car() );
200         else return(null);
201     } else if ( "cdr".equals(op) ) {
202         if ( RPair.isRPair(first) ) return( ((RPair)first).Cdr() );
203         else return(null);
204     } else if ("atom?".equals(op)) {
205         if ( RPair.isAtom( args.car() ) ) return(t);
206         else return(RPair.nil);
207     } else if ("eq?".equals(op)) { Object second = args.nth(1);
208         if ( first == null )
209             if ( second == null ) return(t); else return(RPair.nil);
210         else
211             if ( first.equals(second) ) return(t); else return(RPair.nil);
212     } else return(null);
213 }
214
215 /**

```

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

```

215 */
216 Object applyCompound(RPair op, RPair args, REnvironment env) {
217     if ( "LAMBDA".equals(op.car()) ) return( applyLambda(op, args, env) );
218     else if ( "RHO".equals(op.car()) ) return( applyRho(op, args, env) );
219     else {
220         System.err.println("ERROR: " + op.car() + " undefined!");
221         return(null);
222     }
223 }
224
225 /**

```

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, also known as closure.

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

```

236 */
237 Object applyLambda(RPair op, RPair args, REnvironment currentenv) {
238     Object result = null;
239     try {
240         RPair evargs = evalRPair(args, currentenv);
241         REnvironment env = (REnvironment)(op.nth(1)); // stored env
242         Object formals = op.nth(2); // (f0 f1)
243         RPair body = op.CDR().CDR().CDR(); // (e0 ...)
244         REnvironment ext = env.extend(new RFrame(formals, evargs));
245         result = evalSequence( body, ext );
246     } catch(Throwable t) { System.err.println("ERROR! in lambda: "+t); }
247     return(result);
248 }
249
250 /**

```

Method: [applyRho\(RPair, RPair, REnvironment\)](#)

It first reconstructs the expression, consing the `name`. Then the reconstructed expression is quoted, 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.

```

261 */
262 Object applyRho(RPair op, RPair args, REnvironment env) {
263     Object result = null;
264     try{
265         Object name = op.CDR().car();
266         Object expander = op.CDR().CDR().car();
267         RPair expression = RPair.cons(name, args);
268         RPair qexp = RPair.cons("quote", RPair.cons(expression, null));
269         Object expansion = eval(RPair.cons(expander, RPair.cons(qexp, null)), env);
270         result = eval(expansion, env);
271     } catch(Throwable t) { System.err.println("ERROR! in rho: "+t); }
272     return( result );
273 }
274
275 /**

```

Method: [toString\(\)](#)

```

275 */
276 public String toString() { return("RLisp"); }
277
278 }

```

## 2.5 File: [RLispJava.java](#)

```

1 /**

```

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

```

16 */
17 package RLisp;
18
19 import java.lang.reflect.Constructor;
20 import java.lang.reflect.Method;
21 import java.lang.reflect.Array;
22 import java.lang.reflect.Field;
23 import java.lang.reflect.InvocationTargetException;
24 import java.net.URL;
25 import java.io.InputStreamReader;
26 import java.io.File;
27 import java.io.FileReader;
28 import java.io.BufferedReader;
29
30 public class RLispJava extends RLisp {
31
32 /**

```

Variable: `rcl` is the incremental class loader used

```

32 */
33 private RClassLoader rcl;
34
35 /**

```

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

```

35 */
36 private URL baseURL;
37
38 /**

```

Constructor: `RLispJava(RClassLoader)`

```

38 */
39 public RLispJava(RClassLoader rcl) {
40     super();
41     this.rcl = rcl;
42     baseURL = null;
43     try {
44         baseURL = new URL("file:"+System.getProperty("user.dir")+File.separator);
45     } catch(Throwable t) { System.err.println(t); }
46 }
47
48 /**

```

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

```

50 */
51 boolean isSpecial(RPair p) { return ( super.isSpecial(p) ||
52     "string".equals(p.car()) ||
53     "new".equals(p.car())    || "method".equals(p.car()) ||
54     "field".equals(p.car())  || "array".equals(p.car())  ||
55     "path".equals(p.car())   || "load".equals(p.car())   );
56 }
57
58 /**

```

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

```

60 */
61 Object evalSpecial(RPair p, REnvironment env) {
62     if ( super.isSpecial(p) ) return( super.evalSpecial(p,env) );
63     else return( evalJava(p,env) );
64 }
65
66 /**

```

Method: `evalJava(RPair, REnvironment)`

```

66 */
67 Object evalJava(RPair je, REnvironment env) {
68     //String op = je.nth(0).toString();    // operation
69     Object op = eval(je.nth(0),env);
70     if ( "new".equals(op) )    return( evalJnew(je.CDR(),env) );
71     else if ( "array".equals(op) ) return( evalJarray(je.CDR(),env) );
72     else if ( "method".equals(op) ) return( evalJrun(je.CDR(),env) );
73     else if ( "field".equals(op) ) return( evalJset(je.CDR(),env) );
74     else if ( "path".equals(op) ) return( evalJpath(je.CDR(),env) );
75     else if ( "load".equals(op) ) return( evalJload(je.CDR(),env) );
76     else if ( "string".equals(op) ) return( je.CDR().toString(false) );
77     else {
78         System.err.println("ERROR: (" + op + " ...) undefined");
79         return(null);
80     }
81 }
82
83
84 /**

```

Method: `evalJnew(RPair, REnvironment)`

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

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

```

90 */
91 Object evalJnew(RPair je, REnvironment env) {
92     if ( je == null || je.isNil() ) {
93         System.err.println("ERROR: (new) found!");
94         return(null);
95     }
96     Object result = null;
97     try {

```

```

98   Object co = eval(je.car(),env);
99   if ( co == null ) return(null);
100  Class cc = StoC(co.toString());
101  RPair args = null; Object arg = null;
102  int l = RPair.isList( je.CDR() );
103  if ( l < 0 ) return(null); else args = (RPair)(je.CDR());
104  Class[] argc = new Class[l];
105  Object[] arga = new Object[l];
106  for(int i=0; i<l; i++) {
107    arg = args.nth(i);
108    arga[i] = eval(arg,env);
109    if ( arga[i] == null ) argc[i] = Void.TYPE;
110    else {
111      argc[i] = CtoC(arga[i].getClass());
112      if ( RPair.isRPair(arga[i]) && ((RPair)arga[i]).car() != null ) {
113        Class ca = StoC( ((RPair)arga[i]).car().toString() );
114        if ( ca != null ) { argc[i] = ca;
115          Object oa = ((RPair)arga[i]).cdr();
116          if ( oa == null && ca.isInstance(RPair.nil) ) arga[i] = RPair.nil;
117          else if ( oa == null || ca.isInstance(oa) ) arga[i] = oa;
118          else arga[i] = StoO(ca,oa.toString());
119        }
120      }
121    }
122  }
123  if ( cc.isArray() ) {
124    Class cp = cc; while (cp.isArray()) cp = cp.getComponentType();
125    int[] argi = new int[arga.length];
126    for(int i=0; i<arga.length; i++)
127      argi[i] = Integer.parseInt(arga[i].toString());
128    result = Array.newInstance(cp, argi);
129  } else result = cc.getConstructor(argc).newInstance(arga);
130 } catch(Throwable te) {
131   System.err.println("ERROR: (new "+je.toString(false)+" ["+"te+"]");
132   result = null;
133 }
134 return(result);
135 }
136
137 /**

```

*Method:* evalJarray(RPair, REnvironment)

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

```

140 */
141 Object evalJarray(RPair je, REnvironment env) {
142   Object result = null;
143   if ( je == null || je.isNil() ) {
144     System.err.println("ERROR: (array) found!");
145     return(null);
146   }
147   try{
148     Object co = eval(je.car(),env);
149     if ( co == null ) return(null);
150     Class cc = StoC(co.toString());
151     RPair args = null;

```

```

152     int l = RPair.isList( je.CDR() );
153     if ( l < 0 ) return(null); else args = (RPair)(je.CDR());
154     result = Array.newInstance(cc,l);
155     Object arg = null;
156     for(int i=0; i<l; i++) {
157         arg = eval(args.nth(i),env);
158         if ( "String".getClass().equals(arg.getClass() )
159             arg = StoO(cc, (String)arg);
160         Array.set(result, i, arg);
161     }
162 } catch(Throwable t) { System.err.println("ERROR: "+t); }
163 return(result);
164 }
165
166 /**

```

*Method:* `evalJrun(RPair, REnvironment)`

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

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

```

172 */
173 Object evalJrun(RPair je, REnvironment env) {
174     if (je == null || je.isNil() ) {
175         System.err.println("ERROR: (method) found!");
176         return(null);
177     }
178     Object result = null;
179     Object o = eval(je.car(),env);
180     Class cc = null;
181     try { cc = Class.forName( o.toString(), true, rcl ); }
182     catch(Throwable t) { cc = o.getClass(); } // o is not a Class name
183     try {
184         Object mo = eval(je.CDR().car(),env);
185         if ( mo == null ) {
186             System.err.println("ERROR: method not found!");
187             return(null);
188         }
189         String mn = mo.toString();
190         RPair args = null; Object arg = null;
191         int l = RPair.isList( je.CDR().CDR() );
192         if ( l < 0 ) return(null); else args = (RPair)(je.CDR().CDR());
193         Class[] argc = new Class[l];
194         Object[] arga = new Object[l];
195         for(int i=0; i<l; i++) {
196             arg = args.nth(i);
197             arga[i] = eval(arg,env);
198             if ( arga[i] == null ) argc[i] = Void.TYPE;
199             else {
200                 argc[i] = CtoC(arga[i].getClass());
201                 if ( RPair.isRPair(arga[i]) && ((RPair)arga[i]).car() != null) {
202                     Class ca = StoC( ((RPair)arga[i]).car().toString() );
203                     if ( ca != null ) { argc[i] = ca;
204                         Object oa = ((RPair)arga[i]).cdr();
205                         if ( oa == null && ca.isInstance(RPair.nil) ) arga[i] = RPair.nil;
206                         else if ( oa == null || ca.isInstance(oa) ) arga[i] = oa;

```

```

207         else arga[i] = StoO(ca,oa.toString());
208     }
209 }
210 }
211 }
212 //Method met = cc.getDeclaredMethod(mn,argc);
213 Method met = cc.getMethod(mn,argc);
214 result = met.invoke(o,arga);
215 } catch(Throwable te) {
216     System.err.println("ERROR: (method "+je.toString(false)+" ["+te+"]");
217     result = null;
218 }
219 return(result);
220 }
221
222
223 /**

```

Method: `evalJset(RPair, REnvironment)`

```

Note that je = ([Class|obj] Field [val| ])
226 */
227 Object evalJset(RPair je, REnvironment env) {
228     if (je == null || je.isNil() ) {
229         System.err.println("ERROR: (field) found!");
230         return(null);
231     }
232     Object result = null;
233     Object o = eval(je.car(),env);
234     Class cc = null;
235     try { cc = Class.forName( o.toString(), true, rcl ); }
236     catch(Throwable t) { cc = o.getClass(); } // o is not a Class name
237     try {
238         Object fo = eval(je.CDR().car(),env);
239         if ( fo == null ) {
240             System.err.println("ERROR: field not found!");
241             return(null);
242         }
243         String fn = fo.toString();
244         //Field f = cc.getDeclaredField( fn );
245         Field f = cc.getField( fn );
246         if ( !RPair.isNil(je.CDR().CDR()) ) {
247             Object v = eval(je.nth(2),env);
248             if ( "String".getClass().equals( v.getClass() ) )
249                 v = StoO( f.getType(), (String)v );
250             f.set(o, v);
251         }
252         result = f.get(o);
253     }
254     catch(Throwable te) {
255         System.err.println("ERROR: (field "+je.toString(false)+" ["+te+"]");
256         result = null;
257     }
258     return(result);
259 }
260

```



```
261  /**
```

*Method:* `evalJpath(RPair, REnvironment)`

```
261  */
262  Object evalJpath(RPair je, REnvironment env) {
263      String p = je.toString(false);
264      try { rcl.addURL(new URL(baseURL, p)); }
265      catch(Throwable t) { System.err.println(t); p = null; }
266      return(p);
267  }
268
269  /**
```

*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`).

```
283  */
284  Object evalJload(RPair je, REnvironment env) {
285      Object res = null;
286      URL oldbaseURL = baseURL;
287      try {
288          String urln = je.toString(false);
289          URL url = new URL(baseURL, urln);
290          // System.err.println("baseURL = "+baseURL);
291          // System.err.println("    url = "+url);
292          baseURL = url;
293          InputStreamReader jisr = new InputStreamReader(url.openStream());
294          BufferedReader in = new BufferedReader(jisr);
295          String fc = "";
296          String newline = in.readLine();
297          while (newline != null) {
298              fc = fc + newline + "\n";
299              newline = in.readLine();
300          }
301          Object[] exp = RPair.Tokenize(fc);
302          if (exp == null) return(null);
303          for(int i=0; i<exp.length; i++) res = eval(exp[i],env);
304      } catch(Throwable t) {
305          System.err.println(t);
306          res = null;
307      }
308      baseURL = oldbaseURL;
309      return(res);
310  }
311
312
313  /**
```

*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`.

```

318 */
319 private Class CtoC(Class oc) {
320     Class c = oc;
321     if (oc==null) return(null);
322     else if (oc.equals(Boolean.class)) c = Boolean.TYPE;
323     else if (oc.equals(Character.class)) c = Character.TYPE;
324     else if (oc.equals(Integer.class)) c = Integer.TYPE;
325     else if (oc.equals(Byte.class)) c = Byte.TYPE;
326     else if (oc.equals(Short.class)) c = Short.TYPE;
327     else if (oc.equals(Long.class)) c = Long.TYPE;
328     else if (oc.equals(Float.class)) c = Float.TYPE;
329     else if (oc.equals(Double.class)) c = Double.TYPE;
330     return(c);
331 }
332
333 /**

```

*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

```

340 */
341 private Class StoC(String cn) {
342     if ( cn == null ) return(null);
343     Class c;
344     int dims = 0; int l = cn.length();
345     while ( cn.lastIndexOf("[]") == l-2 ) { dims++;
346         cn = cn.substring(0,l-2); l = cn.length();
347     }
348     if (cn.equals("java.lang.String")) c = "String".getClass();
349     else if (cn.equals("String")) c = "String".getClass();
350     else if (cn.equals("boolean")) c = Boolean.TYPE;
351     else if (cn.equals("char")) c = Character.TYPE;
352     else if (cn.equals("int")) c = Integer.TYPE;
353     else if (cn.equals("byte")) c = Byte.TYPE;
354     else if (cn.equals("short")) c = Short.TYPE;
355     else if (cn.equals("long")) c = Long.TYPE;
356     else if (cn.equals("float")) c = Float.TYPE;
357     else if (cn.equals("double")) c = Double.TYPE;
358     else if (cn.equals("void")) c = Void.TYPE;
359     else try { c = Class.forName(cn,true,rcl); }
360     catch(ClassNotFoundException e) {
361         System.err.println("Class not found: " + cn);
362         c = null;
363     }
364     if (c != null && dims > 0) c = arrayClass(c,dims);
365     return(c);
366 }

```

```
367
368 /**
```

Method: `arrayClass(Class, int)`

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

```
373 */
374 public static Class arrayClass(Class c, int dims) {
375     if ( c == null || dims < 0 ) return(null);
376     if ( dims == 0 ) return(c);
377     int[] d = new int[dims]; for(int i=0; i<dims; i++) d[i] = 0;
378     Class ac = null;
379     try{ ac = Array.newInstance(c,d).getClass(); }
380     catch(Throwable t) { System.err.println("ERROR: " + t); }
381     return(ac);
382 }
383
384 /**
```

Method: `Sto0(Class, String)`

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

```
389 */
390 public static Object Sto0(Class c, String on) {
391     if ( c == null || on == null ) return(null);
392     Object o = null;
393     if ( c.isInstance(on) ) o = on;
394     else if ( c.equals("String".getClass()) ) o = on;
395     else if ( c.equals(Boolean.TYPE) ) o = new Boolean(on);
396     else if ( c.equals(Character.TYPE) ) o = new Character(on.charAt(0));
397     else if ( c.equals(Integer.TYPE) ) o = new Integer(on);
398     else if ( c.equals(Byte.TYPE) ) o = new Byte(on);
399     else if ( c.equals(Short.TYPE) ) o = new Short(on);
400     else if ( c.equals(Long.TYPE) ) o = new Long(on);
401     else if ( c.equals(Float.TYPE) ) o = new Float(on);
402     else if ( c.equals(Double.TYPE) ) o = new Double(on);
403     else if ( c.equals(Void.TYPE) ) o = null;
404     else {
405         String[] arg = new String[1]; arg[0] = on;
406         Class[] carg = new Class[1]; carg[0] = "String".getClass();
407         //try { o = c.getDeclaredConstructor(carg).newInstance((Object[])arg); }
408         try { o = c.getConstructor(carg).newInstance((Object[])arg); }
409         catch(Throwable t) { System.err.println(t); o = null; }
410     }
411     return(o);
412 }
413
414 /**
```

Method: `toString()`

```
414 */
415 public String toString() { return("RLispJava"); }
416
417 }
```

## 2.6 File: [RLispInterpreter.java](#)

```
1 /**
```

### Class: [RLispInterpreter](#)

A Lisp interpreter.

It uses [ENV](#) as global environment, and an [RLisp](#) evaluator [lisp](#).

@author © Ramón Casares 2003

@version 2003.03.03

```
10 */
11 package RLisp;
12
13 public class RLispInterpreter {
14
15 /**
```

### Variable: [ENV](#)

```
15 */
16 public REnvironment ENV;
17
18 /**
```

### Variable: [lisp](#)

```
18 */
19 private RLisp lisp;
20
21 /**
```

### Method: [counter\(int\)](#)

```
21 */
22 public int counter(int val) {
23     int i = lisp.counter;
24     lisp.counter = val;
25     return(i);
26 }
27 public int counter() { return(lisp.counter); }
28
29 /**
```

### Constructor: [RLispInterpreter\(RLisp\)](#)

```
29 */
30 public RLispInterpreter(RLisp lisp) {
31     RFrame FR = new RFrame();
32     ENV = new REnvironment(FR);
33     this.lisp = lisp;
34 }
35
36 /**
```

### Method: [Eval\(String\)](#)

```
36 */
```

```

37 public Object Eval(String input) {
38     Object[] exp = RPair.Tokenize(input);
39     if (exp == null) return(null);
40     Object res = null;
41     for(int i=0; i<exp.length; i++) res = lisp.eval(exp[i],ENV);
42     return(res);
43 }
44
45 /**

```

Method: `eval(String)`

```

45 */
46 public Object eval(String input) {
47     return( lisp.eval(RPair.tokenize(input), ENV) );
48 }
49
50 /**

```

Method: `toString()`

```

50 */
51 public String toString() {
52     return(lisp.toString() + " on " + ENV.toString(false));
53 }
54
55 /**

```

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

```

67 */
68 public static void main(String[] args) {
69     if ( args.length > 0 ) {
70         String s = "";
71         for(int i=0; i<args.length; i++) s = s + " " + args[i];
72         s = s.substring(1);
73         java.net.URL[] urls = new java.net.URL[1];
74         java.io.File ud = new java.io.File(System.getProperty("user.dir"));
75         try { urls[0] = ud.toURL(); }
76         catch (java.net.MalformedURLException mue) {} // always right
77         RClassLoader rcl = new RClassLoader(urls);
78         RLispInterpreter rli = new RLispInterpreter( new RLispJava(rcl) );
79         System.out.println( rli.Eval(s) );
80     } else System.out.println( "[null]" );
81 }
82
83 }

```

## 2.7 File: [RLisp.lisp](#)

```
1 ; RLisp.lisp
2
3 (def nil (cons))
4 (def t (eq? (cons) (cons)))
5 (def null? (lambda (x) (eq? x nil)))
6 (def not (lambda (b) (cond (b nil) (t t))))
7 (def list (lambda l l))
8 (def cadr (lambda (l) (car (cdr l))))
9
10 (def macro
11   (rho macro
12     (lambda ((macro name expander))
13       (list 'def name (list 'rho name expander))
14     )))
15
16 (def syntax
17   (rho syntax
18     (lambda ((syntax template expansion))
19       (list 'def (car template)
20         (list 'rho (car template)
21           (list 'lambda (list template) expansion)
22         )))))
23
24 (syntax (define name definition)
25   (cond
26     ((atom? name) (list 'def name definition))
27     (t (list 'define (car name) (list 'lambda (cdr name) definition))))
28 ))
29
30 (syntax (if test t-clause f-clause)
31   (list 'cond (list test t-clause) (list 't f-clause)))
32
33 (syntax (sequence , expressions)
34   (list 'cond (cons 't expressions)))
35
36 ; (or b1 b2 ...) => (cond (b1 t) (t (or b2 ...)))
37 (syntax (or , terms)
38   (cond ((eq? terms nil) 'nil)
39     (t (list 'cond (list (car terms) 't) (list 't (cons 'or (cdr terms))))))
40 ))
41
42 ; (and b1 b2 ...) => (cond (b1 (and b2 ...) (t nil))) =>
43 (syntax (and , terms)
44   (cond ((eq? terms nil) 't)
45     (t (list 'cond (list (car terms) (cons 'and (cdr terms))) (list 't 'nil))))
46 ))
47
48 (define (mapcar f l)
49   (cond ((eq? l nil) nil)
50     (t (cons (f (car l)) (mapcar f (cdr l)))))
51 ))
52
53 ; (let ((f1 v1) (f2 v2)) body) => ((lambda (f1 f2) body) v1 v2)
54 (syntax (let arglist , body)
```

```
55 (cons
56   (cons 'lambda (cons (mapcar car arglist) body))
57   (mapcar cadr arglist)
58 ))
59
60 (def GENV (car (cdr (lambda))))
61 (syntax (Gdefine name definition)
62   (list 'eval (list 'def name definition) 'GENV))
63
64 (load RLispJava.lisp)
65 (load RLispArray.lisp)
66 (load RLispMaths.lisp)
```

## 2.8 File: [RLispJava.lisp](#)

```
1 ; RLispJava.lisp (RMCG20040131)
2
3 (cond ((eq? 'define define) (load RLisp.lisp)) (t))
4
5 ; Java null cannot be in the dictionary. Write (car (cons)) to get it.
6
7 (define (boolean b) (new 'java.lang.Boolean (method b 'toString)))
8 (define (char c) (new 'java.lang.Character
9   (method (method c 'toString) 'charAt (int 0))))
10
11 (define (byte n) (new 'java.lang.Byte (method n 'toString)))
12 (define (short n) (new 'java.lang.Short (method n 'toString)))
13 (define (int n) (new 'java.lang.Integer (method n 'toString)))
14 (define (long n) (new 'java.lang.Long (method n 'toString)))
15
16 (define (float n) (new 'java.lang.Float (method n 'toString)))
17 (define (double n) (new 'java.lang.Double (method n 'toString)))
```

## 2.9 File: [RLispArray.lisp](#)

```
1 ; RLispArray.lisp (RMCG20040131)
2
3 (cond ((eq? 'define define) (load RLisp.lisp)) (t))
4 (cond ((eq? 'int int) (load RLispJava.lisp)) (t))
5
6 (define (isArray? o) (method (method o 'getClass) 'isArray))
7
8 ; (new Class[] dim1 dim2) creates a bidimensional array sized dim1 x dim2
9 ; (array Class ob1 ob2) creates an array of length 2 initialized
10
11 (define (array-length a)
12   (method 'java.lang.reflect.Array 'getLength (cons 'java.lang.Object a)))
13
14 (define (array-get a i)
15   (method 'java.lang.reflect.Array 'get
16     (cons 'java.lang.Object a) (cons 'int i)
17 ))
18
19 (define (array-set! a i v)
20   (method 'java.lang.reflect.Array 'set
```

```

21 (cons 'java.lang.Object a) (cons 'int i) (cons 'java.lang.Object v)
22 ))
23
24 (define (l2v l) ; creates a vector and adds objects in list l to it
25 (l2vv l (new 'java.util.Vector)))
26
27 (define (l2vv l v) ; adds objects in list l to Vector v
28 (cond
29 ((eq? l nil) v)
30 (t
31 (method v 'add (cons 'java.lang.Object (car l)))
32 (l2vv (cdr l) v)
33 )))

```

## 2.10 File: [RLispMaths.lisp](#)

```

1 ; RLispMaths.lisp (RMCG20030716)
2
3 (cond ((eq? 'define define) (load RLisp.lisp)) (t))
4 (cond ((eq? 'int int) (load RLispJava.lisp)) (t))
5
6 (define (# x)
7 (new 'java.math.BigInteger (method x toString)))
8
9 (define (++ x y) (method (# x) add (# y)))
10 (define (- x y) (method (# x) subtract (# y)))
11 (define (** x y) (method (# x) multiply (# y)))
12 (define (/ x y) (method (# x) divide (# y)))
13 (define (% x y) (method (# x) remainder (# y)))
14 (define (> x y)
15 (eq? (int 1) (method (# x) compareTo (# y))))
16 (define (= x y) (eq? (# x) (# y)))
17
18 (define (sigma l)
19 (cond ((eq? l nil) (# 0))
20 (t (++ (car l) (sigma (cdr l))))))
21 (define + (lambda l (sigma l)))
22
23 (define (pi l)
24 (cond ((eq? l nil) (# 1))
25 (t (** (car l) (pi (cdr l))))))
26 (define (* , l) (pi l))

```

## 2.11 File: [Primes.lisp](#)

```

1 ; Primes.lisp (RMCG20030716)
2
3 (cond ((eq? 'define define) (load RLisp.lisp)) (t))
4 (cond ((eq? '* *) (load RLispMaths.lisp)) (t))
5
6 (define (divides? a b) (= (% b a) 0))
7 (define (square x) (* x x))
8 (define (find-divisor n test-divisor)
9 (cond
10 ((> (square test-divisor) n) n)
11 ((divides? test-divisor n) test-divisor)

```



```
12 (t (find-divisor n (+ test-divisor 1))))
13 (define (smallest-divisor n) (find-divisor n 2))
14 (define (prime? n) (= n (smallest-divisor n)))
15 (define (divisors x)
16 (cond
17 ((prime? x) (cons x nil))
18 (t (cons (smallest-divisor x) (divisors (/ x (smallest-divisor x)))))))
```

## 2.12 File: [RLispConsole.java](#)

```
1 /**
```

### 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

```
19 */
20 package RLisp;
21
22 import java.net.URL;
23
24 import java.lang.reflect.Constructor;
25 import java.lang.reflect.Method;
26 import java.lang.reflect.Field;
27 import java.lang.reflect.Array;
28 import java.lang.reflect.Modifier;
29 import java.lang.reflect.InvocationTargetException;
30
31 import java.util.Vector;
32 import java.util.BitSet;
33 import java.util.Date;
34
35 import java.io.PrintWriter;
36 import java.io.BufferedWriter;
37 import java.io.FileWriter;
38 import java.io.PrintStream;
39 import java.io.BufferedReader;
40 import java.io.InputStreamReader;
41 import java.io.InputStream;
42 import java.io.FileReader;
43
44 import java.io.IOException;
45 import java.io.FileNotFoundException;
46 import javax.swing.text.BadLocationException;
47
48 import java.util.*;
49 import java.awt.*;
```

```
50 import java.io.File;
51 import javax.swing.*;
52 import java.awt.event.*;
53 import java.io.PipedWriter;
54 import java.io.PipedReader;
55 import java.io.BufferedReader;
56 import java.io.IOException;
57
58 import javax.swing.JOptionPane;
59
60 public class RLispConsole implements WindowListener, ActionListener {
61
62     public RLispInterpreter lisp;
63
64     private Object result; // referred to as @
65     private String expression;
66     private String logfilename = "RLisp.log"; // default name
67     private boolean logging = false;
68     private PrintWriter outfile = null;
69     private String version = "20040115";
70
71     /**
```

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 `ClassLoaders`, are two completely different classes. Because of this, the same `ClassLoader` should load every class.

```
78 */
79     private RClassLoader rcl;
80
81     JFrame frame;
82     JTextArea textArea;
83     JLabel statuslabel;
84     Container contentPane;
85
86     /**
```

Constructor: `RLispConsole(String)`

```
86 */
87     public RLispConsole(String title) {
88
89         URL[] urls = new URL[1];
90         File ud = new File(System.getProperty("user.dir"));
91         try { urls[0] = ud.toURL(); }
92         catch (java.net.MalformedURLException mue) {} // always right
93         rcl = new RClassLoader(urls);
94
95         lisp = new RLispInterpreter( new RLispJava(rcl) );
96
97         frame = new JFrame(title);
98
99         frame.addNotify();
100        frame.addWindowListener(this);
```

```
101
102     frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
103
104     //setBounds(200, 200, 500, 200);
105
106     contentPane = frame.getContentPane();
107     //contentPane.setLayout(new FlowLayout());
108     //contentPane.setLayout(new GridLayout(3,1));
109     //contentPane.setLayout(new BorderLayout(contentPane,BorderLayout.PAGE_AXIS));
110     contentPane.setLayout(new BorderLayout(contentPane,BorderLayout.Y_AXIS));
111
112     JMenuBar Bar = new JMenuBar();
113     JMenu menuFile = new JMenu("File"); //////////////////////////////////
114     JMenuItem miLoad = new JMenuItem("Load");
115     miLoad.addActionListener(this);
116     menuFile.add(miLoad);
117     JMenuItem miLoadFrom = new JMenuItem("Load from");
118     miLoadFrom.addActionListener(this);
119     menuFile.add(miLoadFrom);
120     JMenuItem miSave = new JMenuItem("Save");
121     miSave.addActionListener(this);
122     menuFile.add(miSave);
123     JMenuItem miSaveTo = new JMenuItem("Save to");
124     miSaveTo.addActionListener(this);
125     menuFile.add(miSaveTo);
126     menuFile.addSeparator();
127     JMenuItem miTree = new JMenuItem("Tree");
128     miTree.addActionListener(this);
129     menuFile.add(miTree);
130     JMenuItem miKeyboard = new JMenuItem("Keyboard");
131     miKeyboard.addActionListener(this);
132     menuFile.add(miKeyboard);
133     JMenuItem miSession = new JMenuItem("Session");
134     miSession.addActionListener(this);
135     menuFile.add(miSession);
136     menuFile.addSeparator();
137     JMenuItem miClose = new JMenuItem("Close");
138     miClose.addActionListener(this);
139     menuFile.add(miClose);
140     JMenu menuEdit = new JMenu("Edit"); //////////////////////////////////
141     JMenuItem miCut = new JMenuItem("Cut");
142     miCut.setEnabled(false);
143     miCut.addActionListener(this);
144     menuEdit.add(miCut);
145     JMenuItem miCopy = new JMenuItem("Copy");
146     miCopy.addActionListener(this);
147     menuEdit.add(miCopy);
148     JMenuItem miPaste = new JMenuItem("Paste");
149     miPaste.setEnabled(false);
150     miPaste.addActionListener(this);
151     menuEdit.add(miPaste);
152     JMenu menuAction = new JMenu("Action"); //////////////////////////////////
153     JMenuItem miPath = new JMenuItem("Path");
154     miPath.addActionListener(this);
155     menuAction.add(miPath);
156     JMenuItem miLisp = new JMenuItem("Lisp code");
```

```

157     miLisp.addActionListener(this);
158     menuAction.add(miLisp);
159     JMenuItem miName = new JMenuItem("Name");
160     miName.addActionListener(this);
161     menuAction.add(miName);
162     JMenuItem miUnname = new JMenuItem("Unname");
163     miUnname.addActionListener(this);
164     menuAction.add(miUnname);
165     JMenuItem miList = new JMenuItem("List");
166     miList.addActionListener(this);
167     menuAction.add(miList);
168     JMenu menuHelp = new JMenu("Help"); ////////////////
169     JMenuItem miManual = new JMenuItem("Manual");
170     miManual.addActionListener(this);
171     menuHelp.add(miManual);
172     JMenuItem smiManual = new JMenuItem("Spanish Manual");
173     smiManual.addActionListener(this);
174     menuHelp.add(smiManual);
175     JMenuItem miCode = new JMenuItem("Code");
176     miCode.addActionListener(this);
177     menuHelp.add(miCode);
178     JMenuItem miAbout = new JMenuItem("About RLisp");
179     miAbout.addActionListener(this);
180     menuHelp.add(miAbout);
181
182     Bar.add(menuFile);
183     Bar.add(menuEdit);
184     Bar.add(menuAction);
185     Bar.add(menuHelp);
186     frame.setJMenuBar(Bar);
187
188     textArea = new JTextArea(15,50);
189     textArea.setEditable(false);
190     textArea.setLineWrap(false);
191     textArea.setBackground(new Color(1.0F,1.0F,0.5F)); // yellow
192     textArea.setFont(new Font("Monospaced",Font.PLAIN,12));
193
194     contentPane.add(new JScrollPane(textArea,
195     JScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
196     JScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED));
197
198     Box statusbox = Box.createHorizontalBox();
199     statuslabel = new JLabel("Status: ");
200     statusbox.add(statuslabel);
201     statusbox.add(Box.createHorizontalGlue());
202     contentPane.add(statusbox);
203
204     frame.setVisible(true);
205     frame.pack();
206 }
207
208 /**

```

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.

```

214 */
215 public void initLisp() {
216     try{
217         URL lispURL =
218             this.getClass().getClassLoader().getResource("RLisp/RLisp.lisp");
219         String sURL = lispURL.toString();
220         writeln("<< (load " + sURL + ")");
221         lisp.eval("(load " + sURL + ")");
222         System.out.println("Init Lisp [" + lisp.counter(0) + "]");
223     } catch(Throwable t) { System.err.println(t); }
224 }
225
226 /**

```

Method: `setLogFile(String)`

```

226 */
227 public void setLogFile(String filename) { this.logfilename = filename; }
228
229 /**

```

Method: `readFile(String)`

```

229 */
230 public void readFile(String filename) throws IOException {
231     BufferedReader infile = new BufferedReader(new FileReader(filename));
232     String line = infile.readLine();
233     while ( (line != null) && !("<< quit".equals(line)) ) {
234         textArea.append(line + "\n");
235         if ( line.startsWith("<< ") ) {
236             expression = line.substring(3);
237             result = lisp.eval(expression);
238             textArea.append(">< " + resultToString() + "\n" );
239             lisp.ENV.define("@",result);
240         }
241         line = infile.readLine();
242     }
243     infile.close();
244     if ("<< quit".equals(line)) closeAction();
245 }
246
247 /**

```

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`.

```

253 */
254 public void session() throws IOException {
255     frame.setVisible(false);
256     BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
257     String outputLine = "New session: " + new Date();
258     System.out.println(">> " + outputLine); writeln(">> " + outputLine);

```

```
259 System.out.println("<> To end the session, enter \"quit\\\"");
260 String oldLine = "quit";
261 System.out.print("<< "); expression = in.readLine();
262 if (expression.equals("")) expression = oldLine;
263 while ( !("quit".equals(expression)) ) {
264     writeln("<< " + expression);
265     result = lisp.eval(expression);
266     outputLine = resultToString();
267     System.out.println(">> " + outputLine);
268     writeln(">> " + outputLine);
269     lisp.ENV.define("@",result);
270     oldLine = expression;
271     System.out.print("<< "); expression = in.readLine();
272     if (expression.equals("")) expression = oldLine;
273 }
274 writeln("<> Session finished: " + new Date());
275 frame.setVisible(true);
276 }
277
278 /**
```

Method: [write\(String\)](#)

```
278 */
279 public void write(String s) {
280     textArea.append(s);
281     if ( outfile != null ) outfile.print(s);
282 }
283
284 /**
```

Method: [writeln\(String\)](#)

```
284 */
285 public void writeln(String s) {
286     textArea.append(s); textArea.append("\n");
287     if ( logging ) outfile.println(s);
288 }
289
290 /**
```

Method: [writeln\(\)](#)

```
290 */
291 public void writeln() { writeln(""); }
292
293
294 /**
```

Method: [executeObject\(Object\)](#)

```
294 */
295 public void executeObject(Object rro) {
296     if ( rro == null ) return;
297     Object[] values = null;
298     Object value = null;
299     String keyselected = null;
300
```

```
301 try {
302
303     if ( rro.getClass() == Class.forName("java.lang.reflect.Constructor") ) {
304         Constructor c = (Constructor)rro;
305         expression = "(new \'' + c.getName());
306         values = getValues( c.getParameterTypes() );
307         expression = expression + ")";
308         writeln("<< " + expression);
309         if ( values != null ) {
310             result = c.newInstance(values);
311             writeln(">> " + resultToString());
312             lisp.ENV.define("@",result);
313         } else { System.err.println("new " + rro + " ERROR!"); }
314
315     } else if ( rro.getClass() == Class.forName("java.lang.reflect.Method") ) {
316         Method c = (Method)rro;
317         Class cl = c.getDeclaringClass();
318         Object obj = null;
319         if ( Modifier.isStatic(c.getModifiers()) ) {
320             expression = "(method \'' + cl.getName() + " \'' + c.getName());
321             values = getValues( c.getParameterTypes() );
322             expression = expression + ")";
323             writeln("<< " + expression);
324         } else {
325             Object[] objs = lisp.ENV.keys(cl,false);
326             if (objs != null && objs.length > 0 ) {
327                 keyselected = (String)JOptionPane.showInputDialog(null,
328                     getName(cl), "Choose an object", // toString
329                     JOptionPane.QUESTION_MESSAGE, null,
330                     objs, objs[0] );
331                 if ( keyselected == null ) values = null;
332                 else {
333                     obj = lisp.ENV.lookup( keyselected );
334                     expression = "(method " + keyselected + " \'' + c.getName());
335                     values = getValues( c.getParameterTypes() );
336                     expression = expression + ")";
337                     writeln("<< " + expression);
338                 }
339             }
340         }
341         if ( values != null ) {
342             result = c.invoke(obj,values);
343             if ( result == null ) writeln(">> [null]");
344             else writeln(">> " + resultToString());
345             lisp.ENV.define("@",result);
346         } else { System.err.println("run " + rro + " ERROR!"); }
347
348     } else if ( rro.getClass().isArray() ) {
349         Class c = rro.getClass().getComponentType();
350         expression = "(array \'' + c.getName());
351         Vector<Object> vals = new Vector<Object>();
352         Object val = getValue(c);
353         while ( val != null ) { vals.add(val); val = getValue(c); }
354         expression = expression + ")";
355         writeln("<< " + expression);
356         Object[] valsa = vals.toArray();
```

```
357     int l = valsa.length;
358     result = Array.newInstance(c,l);
359     for(int i=0; i<l; i++) Array.set(result, i, valsa[i]);
360     writeln(">> " + resultToString());
361     lisp.ENV.define("@",result);
362
363 } else if ( rro.getClass() == Class.forName("java.lang.reflect.Field") ) {
364     Field f = (Field)rro;
365     Class cl = f.getDeclaringClass();
366     Object obj = null;
367     String owner = null;
368     if ( Modifier.isStatic( f.getModifiers() ) ) {
369         owner = "\'" + cl.getName();
370     } else {
371         Object[] objs = lisp.ENV.keys(cl,false);
372         if (objs != null && objs.length > 0 ) {
373             keyselected = (String)JOptionPane.showInputDialog(null,
374                 getName(cl), "Choose an object", // toString
375                 JOptionPane.QUESTION_MESSAGE, null,
376                 objs, objs[0] );
377             if ( keyselected != null ) {
378                 obj = lisp.ENV.lookup( keyselected );
379                 owner = keyselected;
380             }
381         }
382     }
383     Object currentValue = f.get(obj);
384     Class fc = f.getType();
385     if ( Modifier.isFinal( f.getModifiers() ) ) {
386         JOptionPane.showMessageDialog(null,
387             "Final value: " + prettyPrint(currentValue),
388             "Field " + f.toString(),
389             JOptionPane.INFORMATION_MESSAGE);
390         expression = "(field " + owner + " \'' + f.getName() + ")";
391         writeln("<< " + expression);
392         result = currentValue;
393         writeln(">> " + resultToString());
394         lisp.ENV.define("@",result);
395     } else if (
396         JOptionPane.showConfirmDialog(null,
397             "Current value: " + prettyPrint(currentValue) + "\n" + // toString
398             "Do you want to change it?",
399             "Field " + f.toString(),
400             JOptionPane.YES_NO_OPTION,
401             JOptionPane.INFORMATION_MESSAGE) == JOptionPane.YES_OPTION ) {
402         expression = "(field " + owner + " \'' + f.getName();
403         currentValue = getValue(fc);
404         f.set(obj, currentValue);
405         expression = expression + ")";
406         writeln("<< " + expression);
407         result = f.get(obj);
408         writeln(">> " + resultToString());
409         lisp.ENV.define("@",result);
410     } else { // NO OPTION
411         expression = "(field " + owner + " \'' + f.getName() + ")";
412         writeln("<< " + expression);
```



```

413     result = currentValue;
414     writeln(">> " + resultToString());
415     lisp.ENV.define("@",result);
416 }
417
418 } else if ( rro.getClass() == Class.forName("java.io.File") ) {
419     RClassTree rct = new RClassTree(this,(File)rro,rcl);
420     writeln("<> Tree from " + rct.cd.toURI().toURL() );
421     writeln("<< (path " + rct.cd.toURI().toURL() + ")");
422
423 } else System.err.println("ERROR: No action for class "+rro.getClass());
424 }
425 catch (ClassNotFoundException cnfe) { System.err.println(cnfe); }
426 catch (IllegalAccessException iae) { System.err.println(iae); }
427 catch (InvocationTargetException ite) { System.err.println(ite); }
428 catch (InstantiationException ie) { System.err.println(ie); }
429 catch (IllegalArgumentException iae) { System.err.println(iae); }
430 catch (java.net.MalformedURLException mue) { System.err.println(mue); }
431 }
432
433
434 /**

```

### Method: [getValue\(Class\)](#)

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

@param c is the given Class

@return the chosen object

```

440 */
441 Object getValue(Class c) {
442     if ( c == null ) return(null);
443     Object value = null;
444     String keyselected = null;
445     String sin = getName(c);
446     String sout = null;
447     Object[] names = lisp.ENV.keys(c,false);
448     if ( names == null || names.length == 0 ) {
449         sout = JOptionPane.showInputDialog(sin + " expression");
450         if ( sout == null ) return(null);
451         value = lisp.eval(sout);
452         expression = expression + " " + sout;
453     } else {
454         Object[] names = new Object[names.length+1];
455         names[0] = sin + " expression";
456         for(int j=0; j<names.length; j++) names[j+1] = names[j];
457         keyselected = (String)JOptionPane.showInputDialog(null,
458             sin, "Input value",
459             JOptionPane.QUESTION_MESSAGE, null,
460             names, names[0] );
461         if ( keyselected == null ) return(null);
462         else if ( keyselected.equals(sin + " expression") ) {
463             sout = JOptionPane.showInputDialog(sin + " expression");
464             if ( sout == null ) return(null);
465             value = lisp.eval(sout);

```

```

466     expression = expression + " " + sout;
467 } else {
468     value = lisp.ENV.lookup( keyselected );
469     expression = expression + " " + keyselected;
470 }
471 }
472 if ( value == null ) return(null);
473 if ( "String".getClass().equals(value.getClass()) )
474     value = RLispJava.Sto0(c,(String)value);
475 return(value);
476 }
477
478 /**

```

### 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

```

484 */
485 Object[] getValues(Class[] ca) {
486     if ( ca == null ) return(null);
487     Object[] values = new Object[ca.length];
488     String keyselected; Object value;
489     String sin; String sout;
490     for(int i=0; i<ca.length; i++) {
491         keyselected = null; value = null;
492         sin = getName(ca[i]); sout = null;
493         Object[] names = lisp.ENV.keys(ca[i],false);
494         if ( names == null || names.length == 0 ) {
495             sout = JOptionPane.showInputDialog(sin + " expression");
496             if ( sout == null ) return(null);
497             value = lisp.eval(sout);
498             expression = expression + " " +
499                 "(cons \'' + sin + " " + sout + ")";
500         } else {
501             Object[] names = new Object[names.length+1];
502             names[0] = sin + " expression";
503             for(int j=0; j<names.length; j++) names[j+1] = names[j];
504             keyselected = (String)JOptionPane.showInputDialog(null,
505                 sin, "Input value["+i+"]",
506                 JOptionPane.QUESTION_MESSAGE, null,
507                 names, names[0] );
508             if ( keyselected == null ) return(null);
509             else if ( keyselected.equals(sin + " expression") ) {
510                 sout = JOptionPane.showInputDialog(sin + " expression");
511                 if ( sout == null ) return(null);
512                 value = lisp.eval(sout);
513                 expression = expression + " " +
514                     "(cons \'' + sin + " " + sout + ")";
515             } else {
516                 value = lisp.ENV.lookup( keyselected );
517                 expression = expression + " " +
518                     "(cons \'' + sin + " " + keyselected + ")";

```

```

519     }
520   }
521   if ( value == null ) return(null);
522   else {
523     if ( "String".getClass().equals(value.getClass()) )
524       values[i] = RLispJava.Sto0(ca[i],(String)value);
525     else
526       values[i] = value;
527   }
528 }
529 return values;
530 }
531
532 /**

```

*Method:* [getName\(Class\)](#)

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

@param c is the Class

@return its name

```

538 */
539 public static String getName(Class c) {
540   if ( c == null ) return(null);
541   String cis = c.getName();
542   if ( c.isArray() ) {
543     String atend = "";
544     Class pc = c;
545     while ( cis.charAt(0) == '[' ) {
546       atend = atend + "[]";
547       pc = pc.getComponentType();
548       cis = cis.substring(1);
549     }
550     cis = pc.getName() + atend;
551   }
552   return(cis);
553 }
554
555 /**

```

*Method:* [windowClosing\(WindowEvent\)](#)

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

```

559 */
560 public void windowOpened(WindowEvent e) {}
561 public void windowClosing(WindowEvent e) { closeAction(); }
562 public void windowClosed(WindowEvent e) {}
563 public void windowIconified(WindowEvent e) {}
564 public void windowDeiconified(WindowEvent e) {}
565 public void windowActivated(WindowEvent e) {}
566 public void windowDeactivated(WindowEvent e) {}
567
568
569 /**

```

*Method:* `actionPerformed(ActionEvent)`

Implements the ActionListener interface.

```

572 */
573 public void actionPerformed(ActionEvent e) {
574     String texto = e.getActionCommand();
575     statuslabel.setText(texto);
576     if ("Cut".equals(texto))        textArea.cut();
577     else if ("Copy".equals(texto))  textArea.copy();
578     else if ("Paste".equals(texto)) textArea.paste();
579     else if ("Load".equals(texto))  loadAction();
580     else if ("Load from".equals(texto)) loadFromAction();
581     else if ("Save".equals(texto))  saveAction();
582     else if ("Save to".equals(texto)) saveToAction();
583     else if ("Close".equals(texto)) closeAction();
584     else if ("Tree".equals(texto))  treeAction();
585     else if ("Name".equals(texto))  nameAction();
586     else if ("Unname".equals(texto)) unnameAction();
587     else if ("List".equals(texto))  listAction();
588     else if ("Path".equals(texto))  pathAction();
589     else if ("Lisp code".equals(texto)) loadLispAction();
590     else if ("Keyboard".equals(texto)) keyboardAction();
591     else if ("Line".equals(texto))  lineAction(e.getSource());
592     else if ("Session".equals(texto)) sessionAction();
593     else if ("OK".equals(texto))    okTreeAction(e.getSource());
594     else if ("Manual".equals(texto)) pdfAction("RLispManE.pdf");
595     else if ("Spanish Manual".equals(texto)) pdfAction("RLispManS.pdf");
596     else if ("Code".equals(texto))    pdfAction("RLispCode.pdf");
597     else if ("About RLisp".equals(texto)) aboutAction();
598     else System.err.println("ERROR: Action " + texto + " undefined!");
599     System.out.println(texto + " [" + lisp.counter(0) + "]");
600 }
601
602 /**

```

*Method:* `closeAction()`

```

602 */
603 private void closeAction() {
604     writeln("<> Closing: " + new Date());
605     if (outfile != null) outfile.close();
606     System.exit(0);
607 }
608
609 /**

```

*Method:* `lineAction(Object)`

```

609 */
610 private void lineAction(Object so) {
611     RButton rb = (RButton)so;
612     expression = (String)rb.getObject();
613     writeln("<< " + expression);
614     if ("quit".equals(expression)) closeAction();
615     else {
616         result = lisp.eval(expression);
617         writeln(">> " + resultToString());

```

```
618     lisp.ENV.define("@",result);
619   }
620 }
621
622 /**
```

Method: [okTreeAction\(Object\)](#)

```
622 */
623 private void okTreeAction(Object so) {
624     if (so == null) return;
625     RButton rb = (RButton)so;
626     RClassTree rct = (RClassTree)rb.getObject();
627     executeObject(rct.getSelectedObject());
628 }
629
630 /**
```

Method: [sessionAction\(\)](#)

```
630 */
631 private void sessionAction() {
632     try { session(); } catch (Throwable t) {System.err.println(t);}
633 }
634
635 /**
```

Method: [keyboardAction\(\)](#)

```
635 */
636 private void keyboardAction() {
637     new RKeyboard(this,"(quote Keyboard)");
638 }
639
640 /**
```

Method: [listAction\(\)](#)

```
640 */
641 private void listAction() {
642     Object[] names = lisp.ENV.keys();
643     Object value; String vn;
644     writeln("<< names in " + lisp.ENV);
645     for (int i=0; i<names.length; i++) {
646         value = lisp.ENV.lookup(names[i]);
647         writeln( ">> >> " + getName(value.getClass()) + " " +
648             names[i] + " = " + prettyPrint(value));
649     }
650 }
651
652 /**
```

Method: [unnameAction\(\)](#)

```
652 */
653 private void unnameAction() {
654     Object[] names = lisp.ENV.keys();
655     if (names != null && names.length > 0) {
656         Object keyselected = (String)JOptionPane.showInputDialog(null,
```

```

657         "Unname", "Select key to delete",
658         JOptionPane.QUESTION_MESSAGE, null,
659         names, names[0] );
660     if ( keyselected != null ) {
661         expression = "(set! " + keyselected + ")";
662         writeln("<< " + expression);
663         result = lisp.ENV.set( keyselected, null );
664         writeln(">> " + resultToString());
665         lisp.ENV.define("@",result);
666     }
667 }
668 }
669
670 /**

```

Method: [nameAction\(\)](#)

```

670 */
671 private void nameAction() {
672     if ( result == null) {
673         JOptionPane.showMessageDialog(null,"null can not be named!");
674     } else {
675         String title = cutString("Name for " + resultToString(),32);
676         String name = JOptionPane.showInputDialog(title);
677         if ( name != null) {
678             expression = "(def " + name + " @)";
679             writeln("<< " + expression);
680             result = lisp.ENV.define(name,result);
681             writeln(">> " + resultToString());
682             lisp.ENV.define("@",result);
683         }
684     }
685 }
686
687 /**

```

Method: [pathAction\(\)](#)

```

687 */
688 private void pathAction() {
689     JFileChooser chooser = new JFileChooser(System.getProperty("user.dir"));
690     chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
691     if(chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
692         File cd = chooser.getSelectedFile();
693         try {
694             rcl.addURL( cd.toURI().toURL() );
695             writeln("<< (path " + cd.toURI().toURL() + ")" );
696         } catch(java.net.MalformedURLException mue) { System.err.println(mue); }
697     }
698 }
699
700 /**

```

Method: [treeAction\(\)](#)

```

700 */
701 private void treeAction() {
702     try {

```

```
703     RClassTree rct = new RClassTree(this,null,rcl);
704     writeln("<> Tree from " + rct.cd.toURI().toURL() );
705     writeln("<< (path " + rct.cd.toURI().toURL() + ")");
706 } catch(Throwable t) { System.err.println(t); }
707 }
708
709 /**
```

Method: `loadAction()`

```
709 */
710 private void loadAction() {
711     writeln("<> Loading from " + logfilename);
712     try { readFile(logfilename); } catch (Throwable t) {System.err.println(t);}
713     writeln("<> Loaded " + logfilename);
714 }
715
716 /**
```

Method: `loadFromAction()`

```
716 */
717 private void loadFromAction() {
718     JFileChooser chooser = new JFileChooser(System.getProperty("user.dir"));
719     //chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
720     chooser.setFileFilter(new RExtFilter(".log"));
721     if(chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
722         File cd = chooser.getSelectedFile();
723         if ( cd.canRead() ) {
724             try {
725                 String filename = cd.getAbsolutePath();
726                 writeln("<> Loading from " + filename);
727                 rcl.addURL( cd.getParentFile().toURL() );
728                 readFile(filename);
729                 writeln("<> Loaded " + filename);
730             } catch (Throwable t) {System.err.println(t);}
731         }
732     }
733 }
734
735 /**
```

Method: `loadLispAction()`

```
735 */
736 private void loadLispAction() {
737     JFileChooser chooser = new JFileChooser(System.getProperty("user.dir"));
738     //chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
739     chooser.setFileFilter(new RExtFilter(".lisp"));
740     if(chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
741         File cd = chooser.getSelectedFile();
742         if ( cd.canRead() ) {
743             try {
744                 URL filename = cd.toURI().toURL();
745                 URL path = cd.getParentFile().toURI().toURL();
746                 writeln("<< (path " + path.toString() + ")");
747                 rcl.addURL(path);
748                 writeln("<< (load " + filename + ")");

```

```

749     lisp.eval("(load " + filename + ")");
750   } catch (Throwable t) {System.err.println(t);}
751   }
752 }
753 }
754
755 /**

```

Method: [saveAction\(\)](#)

```

755 */
756 private void saveAction() {
757   try {
758     writeln("<> Saving to " + logfilename);
759     logging = true;
760     outfile = new PrintWriter(new BufferedWriter
761       (new FileWriter(logfilename,true)));
762     writeln("<> Date: " + new Date());
763   } catch (Throwable t) {System.err.println(t);} //t.printStackTrace();}
764 }
765
766 /**

```

Method: [saveToAction\(\)](#)

```

766 */
767 private void saveToAction() {
768   JFileChooser chooser = new JFileChooser(System.getProperty("user.dir"));
769   if(chooser.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
770     File cd = chooser.getSelectedFile();
771     String filename = cd.getAbsolutePath();
772     if ( filename != null && filename.length() >0 ) {
773       logfilename = filename;
774       saveAction();
775     }
776   }
777 }
778
779 /**

```

Method: [pdfAction\(String\)](#)

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

```

783 */
784 private void pdfAction(String fn) {
785   URL ju = this.getClass().getClassLoader().getResource("RLisp.jar");
786   if (ju==null)
787     ju = this.getClass().getClassLoader().getResource("RLisp/RLispConsole.class");
788   try{
789     URL fu = new URL(ju,fn);
790     String name = fu.getFile().substring(1);
791     // name = name.replaceAll("%20"," "); // is Java 1.4
792     int i = name.indexOf("%20");
793     while (i >= 0) {
794       name = (new StringBuffer(name)).replace(i,i+3," ").toString();
795       i = name.indexOf("%20");
796     }

```



```
797     Runtime.getRuntime().exec("Start \"/MAX\" \"\" + name + "\""); // Windows
798     writeln("<> Start \"/MAX\" \"\" + name + "\"");
799 } catch (Throwable t) {
800     writeln("<> File " + fn + " not found!");
801     System.err.println(t);
802 }
803 }
804
805 /**
```

*Method:* [aboutAction\(\)](#)

```
805 */
806 private void aboutAction() {
807     try {
808         String[] message = new String[3];
809         message[0] = "RLisp " + version;
810         message[1] = "© 2004 Ramón Casares";
811         message[2] = "r.casares@ieee.org";
812         JOptionPane.showMessageDialog(null, message, "About RLisp",
813             JOptionPane.INFORMATION_MESSAGE);
814         writeln("<> RLisp "+version+ " (" +lisp+")");
815     } catch (Throwable t) {System.err.println(t);}
816 }
817
818 /**
```

*Method:* [toArray\(String\)](#)

```
818 */
819 public static String[] toArray(String sentence) {
820     StringTokenizer st = new StringTokenizer(sentence, " ");
821     int l = st.countTokens();
822     String[] word = new String[l];
823     for(int i=0; i<l; i++) word[i] = st.nextToken();
824     return(word);
825 }
826
827 /**
```

*Method:* [cutString\(String, int\)](#)

```
827 */
828 public static String cutString(String s, int i) {
829     if( s.length() > i ) return(s.substring(0,i)+"...");
830     else return(s);
831 }
832
833 /**
```

*Method:* [arrayToString\(Object\[\]\)](#)

```
833 */
834 public static String arrayToString(Object[] array) {
835     String s = "{";
836     for(int i=0; i<array.length; i++) {
837         if (i>0) s = s + ", ";
838         if ( array[i] == null ) s = s + "[null]";
```

```
839     else s = s + prettyPrint(array[i]);
840   }
841   s = s + "}";
842   return(s);
843 }
844
845 /**
```

Method: [prettyPrint\(Object\)](#)

```
845 */
846 public static String prettyPrint(Object o){
847   String s = "ERROR!";
848   if ( o == null ) s = "[null]";
849   else if ( o.getClass().isArray() ) {
850     Object[] oa = new Object[Array.getLength(o)];
851     for(int i=0; i<oa.length; i++) oa[i] = Array.get(o,i);
852     s = arrayToString(oa);
853   } else s = o.toString();
854   return(s);
855 }
856
857 /**
```

Method: [resultToString\(\)](#)

```
857 */
858 private String resultToString() { return(prettyPrint(result)); }
859
860 /**
```

Method: [main\(String\[\]\)](#)

It creates a console to play the Java Objects accessible from the current directory.

@param args are the command line arguments

```
866 */
867 public static void main(String[] args) {
868   RLispConsole rlc = new RLispConsole("RLisp");
869   rlc.initLisp();
870   if ( args.length > 0 ) {
871     String s = "";
872     for(int i=0; i<args.length; i++) s = s + " " + args[i];
873     s = s.substring(1);
874     try {
875       rlc.writeln("<> Loading from " + s);
876       rlc.readFile(s);
877       rlc.writeln("<> Loaded " + s);
878       System.out.println("Load from [" + rlc.lisp.counter(0) + "]");
879     } catch(IOException ioe) {
880       System.err.println("ERROR: file " + s + " not found!");
881     }
882   }
883 }
884
885 }
```

## 2.13 File: [RButton.java](#)

```
1 /**
```

### 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

```
9 */
10 package RLisp;
11
12 public class RButton extends javax.swing.JButton {
13
14     /**
```

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

```
14 */
15     private Object rbo;
16
17
18     /**
```

### 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

```
24 */
25     public RButton(String text, Object rbo) {
26         super(text);
27         this.rbo = rbo;
28     }
29
30     /**
```

### Method: [getObject\(\)](#)

@return the object attached to this button

```
33 */
34     public Object getObject() { return(rbo); }
35
36
37     /**
```

### Method: [setObject\(Object\)](#)

Changes the object attached to this button

```
40 */
41     public void setObject(Object rbo){
42         this.rbo = rbo;
43     }
44
45 }
```

## 2.14 File: [RKeyboard.java](#)

```
1 /**
```

### Class: [RKeyboard](#)

An RKeyboard is a window to enter lines of text.

@author © Ramón Casares 2003

@version 2003.01.13

```

7 */
8 package RLisp;
9
10 import java.awt.event.KeyListener;
11 import java.awt.event.KeyEvent;
12 import java.awt.event.ActionListener;
13 import java.awt.event.ActionEvent;
14
15 import javax.swing.JTextArea;
16 import javax.swing.JFrame;
17 import javax.swing.WindowConstants;
18 import javax.swing.text.BadLocationException;
19 import javax.swing.ScrollPaneConstants;
20 import javax.swing.JScrollPane;
21 import javax.swing.JButton;
22 import javax.swing.JToolBar;
23 import javax.swing.JLabel;
24 import javax.swing.Box;
25 import java.awt.Color;
26 import java.awt.Font;
27 import javax.swing.BoxLayout;
28 import java.awt.Container;
29
30 public class RKeyboard implements KeyListener, ActionListener {
31
32     private JTextArea ta;
33     private JLabel statuslabel;
34
35     /**
```

### Variable: [callingObject](#)

```

35 */
36     private ActionListener callingObject;
37
38     /**
```

Variable: [endLine](#) is an invisible button that is clicked every time the carriage return is keyed

```

39 */
40     private RButton endLine;
41
42     /**
```

Variable: [inputline](#) saves the last written line

```

42 */
```

```
43 private String inputline;
44
45 /**
```

*Constructor: RKeyboard(ActionListener, String)*

```
45 */
46 RKeyboard(ActionListener callingObject, String firstline) {
47     endLine = new RButton("Line", firstline);
48     JFrame keyframe = new JFrame("Lisp from Keyboard");
49     inputline = firstline;
50     this.callingObject = callingObject;
51     if (callingObject == null) {
52         endLine.addActionListener(this);
53         keyframe.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
54     } else {
55         endLine.addActionListener(callingObject);
56         keyframe.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
57     }
58     keyframe.setBounds(200, 200, 500, 200);
59     Container cp = keyframe.getContentPane();
60     cp.setLayout(new BorderLayout(cp, BorderLayout.Y_AXIS));
61
62     JToolBar toolBar = new JToolBar();
63     JButton jbNesting = new JButton("Nesting");
64     jbNesting.addActionListener(this);
65     toolBar.add(jbNesting);
66     JButton jbWord = new JButton("Word");
67     jbWord.addActionListener(this);
68     toolBar.add(jbWord);
69     JButton jbMax = new JButton("Maximum");
70     jbMax.addActionListener(this);
71     toolBar.add(jbMax);
72     JButton jbMin = new JButton("Minimum");
73     jbMin.addActionListener(this);
74     toolBar.add(jbMin);
75     JButton jbNext = new JButton("Next");
76     jbNext.addActionListener(this);
77     toolBar.add(jbNext);
78     JButton jbPre = new JButton("Previous");
79     jbPre.addActionListener(this);
80     toolBar.add(jbPre);
81     cp.add(toolBar);
82
83     ta = new JTextArea(15,40);
84     ta.setEditable(true);
85     ta.setLineWrap(false);
86     ta.setBackground(new Color(0.5F,1.0F,0.5F));
87     ta.setFont(new Font("Monospaced",Font.PLAIN,12));
88     cp.add(new JScrollPane(ta,
89         JScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
90         JScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED));
91
92     Box statusbox = Box.createHorizontalBox();
93     statuslabel = new JLabel();
94     statusbox.add(statuslabel);
```

```
95     statusbox.add(Box.createHorizontalGlue());
96     cp.add(statusbox);
97
98     keyframe.setVisible(true);
99     keyframe.pack();
100    //keyframe.show(); // deprecated
101    ta.addKeyListener(this);
102    ta.requestFocus();
103 }
104
105 public static int nesting(String text, int p) {
106     if ( p > text.length() ) p = text.length();
107     int l = 0;
108     char c;
109     for(int i=0; i<p; i++) { c = text.charAt(i);
110         if (c == '(') l++; else if (c == ')') l--;
111         if (l < 0) l = 0;
112     }
113     return(l);
114 }
115
116 public static int prePar(String text, int p, int q) {
117     int pp = p;
118     int qq = q;
119     char c;
120     while (qq>0 && pp>0) {
121         c = text.charAt(--pp);
122         if (c == ')') qq++; else if (c == '(') qq--;
123     }
124     while (pp>0 && (text.charAt(pp-1) == '\')) pp--;
125     return(pp);
126 }
127
128 public static int nextPar(String text, int p, int q) {
129     int pp = p;
130     int qq = q;
131     char c;
132     while (qq>0 && pp<text.length()) {
133         c = text.charAt(pp++);
134         if (c == ')') qq--; else if (c == '(') qq++;
135     }
136     return(pp);
137 }
138
139 private static char[] separator = " \t\n\r".toCharArray();
140
141 public static String oneLine(String text) {
142     while ( text.indexOf(';') != -1 ) {
143         int l = text.length();
144         int ini = text.indexOf(';');
145         int fn = text.indexOf('\n',ini); if (fn==-1) fn = l;
146         int fr = text.indexOf('\r',ini); if (fr==-1) fr = l;
147         int fin = fn; if (fr<fn) fin = fr;
148         text = text.substring(0,ini).concat(text.substring(fin,l));
149     }
150     for(int i=0; i<separator.length; i++)
```

```
151     text = text.replace(separator[i], ' ');
152     // text = text.replaceAll(" +"," "); // is Java 1.4
153     int i = text.indexOf(" ");
154     while (i >= 0) {
155         text = (new StringBuffer(text)).replace(i,i+2," ").toString();
156         i = text.indexOf(" ");
157     }
158     return(text);
159 }
160
161 public static boolean isInSet(char c, String set) {
162     char[] cset = set.toCharArray();
163     for(int i=0; i < cset.length; i++) if ( c == cset[i] ) return(true);
164     return(false);
165 }
166
167 public static String thisWord(String text, int p) {
168     if ( p < 0 ) p = 0;
169     if ( p >= text.length() ) p = text.length() - 1;
170     int ini = p;
171     int fin = p;
172     while ( ini > 0 &&
173           !isInSet( text.charAt(ini-1)," \t\n\r\'\"") ) ini--;
174     while ( fin < text.length() &&
175           !isInSet( text.charAt(fin)," \t\n\r\'\"") ) fin++;
176     return(text.substring(ini,fin));
177 }
178
179 String maxExpression(String text, int p){
180     int nl = nesting(text,p);
181     if (nl == 0) return(thisWord(text,p));
182     else {
183         int ini = prePar(text,p,nl);
184         int fin = nextPar(text,p,nl);
185         if ( ini < 0 || fin < 0 ) return("");
186         else return(text.substring(ini,fin));
187     }
188 }
189
190 public static String minExpression(String text, int p){
191     int nl = nesting(text,p);
192     if (nl == 0) return(thisWord(text,p));
193     else {
194         int ini = prePar(text,p,1);
195         int fin = nextPar(text,p,1);
196         if ( ini < 0 || fin < 0 ) return("");
197         else return(text.substring(ini,fin));
198     }
199 }
200
201 String preExpression(String text, int p){
202     return(maxExpression(text,p-1));
203 }
204
205 String nextExpression(String text, int p){
206     return(maxExpression(text,p+1)); }
```

```

207
208
209 /**

```

*Method:* [keyTyped\(KeyEvent e\)](#)

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

```

213 */
214 public void keyTyped(KeyEvent e) {
215     char c = e.getKeyChar();
216     if ( c == '\n' ) { // new line
217         int cp = ta.getCaretPosition();
218         int nl = nesting( ta.getText(), cp-1 );
219         if ( nl == 0 ) {
220             inputline = preExpression( ta.getText() , cp-1 );
221             statuslabel.setText(inputline);
222             endLine.setObject(oneLine(inputline));
223             endLine.doClick();
224         } else {
225             statuslabel.setText("Nesting = " + nl);
226             String ss = "";
227             for(int i=0; i<nl; i++) ss = ss + " ";
228             ta.insert(ss,cp);
229         }
230     } else if ( c == ')' ) {
231         int cp = ta.getCaretPosition();
232         String sta = ta.getText() + ")";
233         int nl = nesting(sta, cp+1);
234         statuslabel.setText("Nesting = " + nl);
235     }
236 }
237
238 public void keyPressed(KeyEvent e) {}
239 public void keyReleased(KeyEvent e) {}
240
241
242 /**

```

*Method:* [actionPerformed\(ActionEvent\)](#)

Implements the `ActionListener` interface.

@param e the action event

```

247 */
248 public void actionPerformed(ActionEvent e) {
249     boolean react = false;
250     String texto = e.getActionCommand();
251     if ("Line".equals(texto)) { react = false;
252         inputline = ((RButton)(e.getSource())).getObject().toString();
253     } else if ("Nesting".equals(texto)) { react = false;
254         inputline = "Nesting = " + nesting(ta.getText(),ta.getCaretPosition());
255     } else if ("Word".equals(texto)) { react = true;
256         inputline = thisWord(ta.getText(),ta.getCaretPosition());
257     } else if ("Maximum".equals(texto)) { react = true;

```



```

258     inputline = maxExpression(ta.getText(),ta.getCaretPosition());
259 } else if ("Minimum".equals(texto)) { react = true;
260     inputline = minExpression(ta.getText(),ta.getCaretPosition());
261 } else if ("Next".equals(texto)) { react = true;
262     inputline = nextExpression(ta.getText(),ta.getCaretPosition());
263 } else if ("Previous".equals(texto)) { react = true;
264     inputline = preExpression(ta.getText(),ta.getCaretPosition());
265 } else { react = false;
266     System.err.println("ERROR: Action " + texto + " no implemented!");
267 }
268 statuslabel.setText(inputline);
269 if ( callingObject == null || !react ) {
270     System.out.println(oneLine(inputline));
271 } else {
272     endLine.setObject(oneLine(inputline));
273     endLine.doClick();
274 }
275 }
276
277
278 /**

```

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

```

@param args are the command line arguments
281 */
282 public static void main(String[] args) {
283     RKeyboard k = new RKeyboard(null, "RConsole");
284 }
285
286 }

```

## 2.15 File: [RClassTree.java](#)

```

1 /**

```

### 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

```

9 */
10 package RLisp;
11
12 import java.net.URL;
13
14 import java.lang.reflect.Array;
15 import java.lang.reflect.Field;
16 import java.lang.reflect.Constructor;
17 import java.lang.reflect.Method;
18
19 import java.io.File;

```

```
20 import java.util.Vector;
21 import java.util.Enumeration;
22
23 import java.util.jar.JarFile;
24 import java.util.jar.JarEntry;
25
26 import java.awt.Container;
27 import java.awt.Color;
28 import javax.swing.BoxLayout;
29 import javax.swing.JScrollPane;
30 import javax.swing.JFrame;
31 import javax.swing.WindowConstants;
32 import javax.swing.JButton;
33 import javax.swing.JToolBar;
34 import javax.swing.JTree;
35 import javax.swing.tree.DefaultMutableTreeNode;
36
37 import javax.swing.JFileChooser;
38
39 import java.awt.event.ActionListener;
40 import java.awt.event.ActionEvent;
41 import java.awt.event.WindowListener;
42 import java.awt.event.WindowEvent;
43 import javax.swing.event.TreeSelectionListener;
44 import javax.swing.event.TreeSelectionEvent;
45
46 public class RClassTree extends JFrame
47     implements ActionListener, TreeSelectionListener {
48
49     /**
```

Variable: `tree`

```
49 */
50     private JTree tree;
51
52     /**
```

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

```
54 */
55     private ActionListener callingObject;
56
57     /**
```

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

```
57 */
58     private Object ob;
59
60     /**
```

Method: `getSelectedObject()`

```
60 */
61     public Object getSelectedObject() { return(ob); }
62
63     /**
```

Variable: `loader` to load classes from any directory

```
63 */
64 private RClassLoader loader;
65
66 /**
```

Variable: `cd` is the selected directory or jar file

```
66 */
67 File cd;
68 /**
```

Variable: `cds` is cd path

```
68 */
69 String cds;
70 /**
```

Variable: `cdsl` is cds length

```
70 */
71 int cdsl;
72 /**
```

Variable: `fileSeparator`

```
72 */
73 char fileSeparator = System.getProperty("file.separator").charAt(0);
74
75
76 /**
```

Method: `setRoot(File)`

```
76 */
77 public int setRoot(File cd) {
78     this.cd = cd;
79     cds = cd.getAbsolutePath();
80     cdsl = cds.length();
81     return(cdsl);
82 }
83
84 /**
```

Constructor: `RClassTree(boolean, RClassLoader)`

```
84 */
85 public RClassTree(File dir, RClassLoader loader) {
86     this(null,dir,loader);
87 }
88
89 /**
```

Constructor: `RClassTree(ActionListener, RClassLoader)`

```
89 */
90 public RClassTree(ActionListener callingObject, RClassLoader loader) {
91     this(callingObject,null,loader);
92 }
93
94 /**
```

*Constructor: RClassTree(RClassLoader)*

```

94  */
95  public RClassTree(RClassLoader loader) { this(null,null,loader); }
96
97  /**

```

*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

```

107  */
108  public RClassTree(ActionListener callingObject, File dir, RClassLoader
    loader) {
109      super("Class Tree");
110      this.loader = loader;
111      Container contentPane = this.getContentPane();
112      contentPane.setLayout(new BorderLayout(contentPane,BorderLayout.Y_AXIS));
113
114      this.callingObject = callingObject;
115      if ( callingObject == null )
116          this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
117      else
118          this.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
119
120      JToolBar toolBar = new JToolBar();
121      JButton jbOK = new JButton("OK",this);
122      if (callingObject == null) jbOK.addActionListener(this);
123      else jbOK.addActionListener(callingObject);
124      toolBar.add(jbOK);
125      this.rootPane.setDefaultButton(jbOK);
126      JButton jbName = new JButton("Name");
127      if (callingObject == null) jbName.addActionListener(this);
128      else jbName.addActionListener(callingObject);
129      toolBar.add(jbName);
130      //contentPane.add(toolBar,BorderLayout.NORTH);
131      contentPane.add(toolBar);
132
133      DefaultMutableTreeNode top = new DefaultMutableTreeNode("Classes");
134      this.tree = new JTree(top);
135      tree.addTreeSelectionListener(this);
136      //tree.setRootVisible(false);
137
138      JScrollPane treeView = new JScrollPane(tree);
139      treeView.setPreferredSize(new java.awt.Dimension(200, 200));
140
141      try{
142          if (dir == null) setRoot( chooseRoot() );
143          else if ( dir.isDirectory() ) setRoot(dir);

```

```
144     if (cd == null) setRoot( new File(System.getProperty("user.dir")) );
145
146     loader.addURL( cd.toURL() );
147     populateTree(top, cd);
148
149     } catch (Throwable t) {
150         System.err.println(t);
151         if ( callingObject == null ) System.exit(0);
152         else { this.dispose(); return; }
153     }
154     contentPane.add(treeView);
155     //tree.setBackground(new Color(1.0F,1.0F,0F)); // yellow
156     this.setVisible(true);
157     this.pack();
158 }
159
160
161 /**
```

Method: [chooseRoot\(\)](#)

```
161 */
162 File chooseRoot() {
163     JFileChooser chooser = new JFileChooser();
164     chooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
165     chooser.setFileFilter(new RExtFilter(".jar"));
166     chooser.setSelectedFile(new File(System.getProperty("user.dir")));
167     int returnVal = chooser.showOpenDialog(null);
168     if( returnVal == JFileChooser.APPROVE_OPTION) {
169         return( chooser.getSelectedFile() );
170     }
171     return(null);
172 }
173
174 /**
```

Method: [populateTree\(DefaultMutableTreeNode, File\)](#)

```
174 */
175 private void populateTree(DefaultMutableTreeNode top, File cd) throws
    Throwable {
176     if ( cd.isFile() ) {
177         String fn = cd.getAbsolutePath();
178         int lp = fn.lastIndexOf('.');
179         if ( lp >= 0 ) {
180             String ext = fn.substring(lp);
181             int pr = cdsl; if ( fn.charAt(cdsl) == fileSeparator ) pr++;
182             String name = fn.substring(pr,lp).replace(fileSeparator, '.');
183             if ( ".class".equals(ext) ) addClassToTree(top, name);
184             if ( ".jar".equals(ext) ) addJarToTree(top, cd, name);
185         }
186     } else if ( cd.isDirectory() ) {
187         System.err.println( cd.toURL().toString() );
188         DefaultMutableTreeNode dir = new DefaultMutableTreeNode(cd);
189         top.add(dir);
190         File[] ls = cd.listFiles();
191         for(int i=0; i<ls.length; i++) populateTree(dir, ls[i]);

```

```

192     }
193   }
194
195   /**

```

Method: [addClassToTree\(DefaultMutableTreeNode, String\)](#)

```

195  */
196  void addClassToTree(DefaultMutableTreeNode top, String classname) {
197      DefaultMutableTreeNode mClass;
198      DefaultMutableTreeNode mArray;
199      DefaultMutableTreeNode mFields = null;
200      Field[] fields = null;
201      DefaultMutableTreeNode mNew = null;
202      Constructor[] constructors = null;
203      DefaultMutableTreeNode mMethods = null;
204      Method[] methods = null;
205      Class c = null;
206      try {
207          c = loader.loadClass(classname);
208          mClass = new DefaultMutableTreeNode(classname);
209          top.add(mClass);
210          mArray = new DefaultMutableTreeNode("Array");
211          mClass.add(mArray);
212          mArray.add( new DefaultMutableTreeNode( Array.newInstance(c,1) ) );
213          fields = c.getFields();
214          if ( fields.length > 0 ) {
215              mFields = new DefaultMutableTreeNode("Fields");
216              mClass.add(mFields);
217              for (int j=0; j<fields.length; j++) {
218                  mFields.add( new DefaultMutableTreeNode( fields[j] ) );
219              }
220          }
221          constructors = c.getConstructors();
222          if ( constructors.length > 0 ) {
223              mNew = new DefaultMutableTreeNode("Constructors");
224              mClass.add(mNew);
225              for (int j=0; j<constructors.length; j++) {
226                  mNew.add( new DefaultMutableTreeNode( constructors[j] ) );
227              }
228          }
229          methods = c.getMethods();
230          if ( methods.length > 0 ) {
231              mMethods = new DefaultMutableTreeNode("Methods");
232              mClass.add(mMethods);
233              for (int j=0; j<methods.length; j++) {
234                  mMethods.add( new DefaultMutableTreeNode( methods[j] ) );
235              }
236          }
237      } catch (java.lang.NoClassDefFoundError ncdfe) {
238          System.err.println("Class "+ ncdfe.getMessage() + " not loadable!");
239      } catch (Throwable t) { System.err.println(t); }
240  }
241
242  /**

```

Method: [addJarToTree\(DefaultMutableTreeNode, File, String\)](#)

```

242 */
243 void addJarToTree(DefaultMutableTreeNode top, File cd, String classname)
    throws Throwable {
244     DefaultMutableTreeNode jd = new DefaultMutableTreeNode(cd);
245     top.add(jd);
246     JarFile jar = new JarFile(cd);
247     Enumeration<JarEntry> jee = jar.entries();
248     JarEntry je = null;
249     String jen = null;
250     while ( jee.hasMoreElements() ) {
251         je = jee.nextElement();
252         jen = je.getName(); // System.out.println(jen);
253         if ( !je.isDirectory() && jen.lastIndexOf('.') >= 0 &&
254             ".class".equals(jen.substring(jen.lastIndexOf('.'))) )
255             addClassToTree(jd, jen.substring(0, jen.lastIndexOf('.')).replace('/', '.'))
    );
256     }
257 }
258
259
260 /**

```

#### 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

```

267 */
268 public void actionPerformed(ActionEvent e) {
269     String texto = e.getActionCommand();
270     if ("OK".equals(texto)) System.out.println(ob.toString());
271     else if ("Name".equals(texto)) System.out.println("Name");
272     else System.err.println("ERROR: Action " + texto + " no implemented!");
273 }
274
275
276 /**

```

#### Method: [valueChanged\(TreeSelectionEvent\)](#)

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

`@param e` the tree selection event

```

282 */
283 public void valueChanged(TreeSelectionEvent e){
284     DefaultMutableTreeNode node = (DefaultMutableTreeNode)
285         tree.getLastSelectedPathComponent();
286     if (node == null) return;
287     //Object nodeInfo = node.getUserObject();
288     //if (node.isLeaf()) { ob = nodeInfo; }
289     ob = node.getUserObject();
290 }
291
292
293 /**

```

**Method:** `main(String[])`

Just for testing this class.

```
@param args command line options are ignored
298 */
299 public static void main(String[] args) {
300     boolean choosing = true;
301     if ( args.length > 0 && args[0].equals("false") ) choosing = false;
302     URL[] urls = new URL[1];
303     File ud = new File(System.getProperty("user.dir"));
304     try { urls[0] = ud.toURL(); }
305     catch (java.net.MalformedURLException mue) {} // always right
306     RClassLoader rcl = new RClassLoader(urls);
307     RClassTree frame = new RClassTree(rcl);
308 }
309
310 }
```

**2.16 File: [RExtFilter.java](#)**

```
1 /**
```

**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

```
11 */
12 package RLisp;
13
14 import java.io.File;
15 import java.io FilenameFilter;
16
17 public class RExtFilter extends javax.swing.filechooser.FileFilter
18 implements FilenameFilter, java.io.FileFilter {
19
20 /**
```

**Variable:** `ext` contains the extension

```
20 */
21 String ext;
22
23 /**
```

**Constructor: [RExtFilter\(String\)](#)**

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

```
26 */
27 public RExtFilter(String ext) { this.ext = ext; }
28
29
30 /**
```



*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

```
39 */
40 public boolean accept(File dir, String name) {
41     String ext = "";
42     if ( name.lastIndexOf('.') > 0 )
43         ext = name.substring( name.lastIndexOf('.') );
44     return ( this.ext.equals(ext) );
45 }
46
47 /**
```

*Method: [accept\(File\)](#)*

Tests if the specified file should be 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

```
55 */
56 public boolean accept(File file) {
57     if ( file.isFile() ) {
58         String name = file.getPath();
59         String ext = "";
60         if ( name.lastIndexOf('.') > 0 )
61             ext = name.substring( name.lastIndexOf('.') );
62         return ( this.ext.equals(ext) );
63     } else return(true);
64 }
65
66 /**
```

*Method: [getDescription\(\)](#)*

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

`@return` a readable description of the filter

```
71 */
72 public String getDescription() {
73     if ( ext.length() > 1 )
74         return( ext.substring(1) + " files" );
75     else return("ERROR: Extension not yet defined!");
76 }
77
78 }
```

## 2.17 File: [RClassLoader.java](#)

```
1 /**
```

### 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

```
9 */
10 package RLisp;
11
12 import java.net.URL;
13 import java.net.URLClassLoader;
14
15 public class RClassLoader extends URLClassLoader {
16
17     /**
```

### Constructor: [RClassLoader\(URL\[\]\)](#)

```
17 */
18     public RClassLoader(URL[] urls) { super(urls); }
19
20     /**
```

### 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

```
28 */
29     public void addURL(URL url) {
30         if ( url == null ) return;
31         URL[] urls = this.getURLs();
32         boolean isNew = true;
33         for (int i=0; i<urls.length; i++) isNew = isNew && !url.equals(urls[i]);
34         if ( isNew ) super.addURL(url);
35     }
36
37 }
```

## 2.18 File: [RLisp.log](#)

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

## 2.19 File: RLisp2jar.bat

```
1 cd ..
2 javac RLisp/RPair.java
3 javac RLisp/RFrame.java
4 javac RLisp/REnvironment.java
5 javac RLisp/RLisp.java
6 javac RLisp/RLispJava.java
7 javac RLisp/RLispInterpreter.java
8 javac RLisp/RLispConsole.java
9 javac RLisp/RButton.java
10 javac RLisp/RKeyboard.java
11 javac RLisp/RClassTree.java
12 javac RLisp/RExtFilter.java
13 javac RLisp/RClassLoader.java
14 jar cf RLisp/RLisp.jar RLisp/RPair.class RLisp/RPair.java
15 jar cf RLisp/RLisp.jar RLisp/RFrame.class RLisp/RFrame.java
16 jar uf RLisp/RLisp.jar RLisp/REnvironment.class RLisp/REnvironment.java
17 jar uf RLisp/RLisp.jar RLisp/RLisp.class RLisp/RLisp.java
18 jar uf RLisp/RLisp.jar RLisp/RLispJava.class RLisp/RLispJava.java
19 jar uf RLisp/RLisp.jar RLisp/RLispInterpreter.class RLisp/RLispInterpreter.java
20 jar uf RLisp/RLisp.jar RLisp/RLispConsole.class RLisp/RLispConsole.java
21 jar uf RLisp/RLisp.jar RLisp/RButton.class RLisp/RButton.java
22 jar uf RLisp/RLisp.jar RLisp/RKeyboard.class RLisp/RKeyboard.java
23 jar uf RLisp/RLisp.jar RLisp/RClassTree.class RLisp/RClassTree.java
24 jar uf RLisp/RLisp.jar RLisp/RExtFilter.class RLisp/RExtFilter.java
25 jar uf RLisp/RLisp.jar RLisp/RClassLoader.class RLisp/RClassLoader.java
26 echo jar uf RLisp/RLisp.jar RLisp/RLisp.tex
27 echo jar uf RLisp/RLisp.jar RLisp/RLisp.pdf
28 jar uf RLisp/RLisp.jar RLisp/RLisp.log
29 jar uf RLisp/RLisp.jar RLisp/RLisp.lisp
30 jar uf RLisp/RLisp.jar RLisp/RLispJava.lisp
31 jar uf RLisp/RLisp.jar RLisp/RLispArray.lisp
32 jar uf RLisp/RLisp.jar RLisp/RLispMaths.lisp
33 jar uf RLisp/RLisp.jar RLisp/Primes.lisp
34 jar uf RLisp/RLisp.jar RLisp/RLisp2jar.bat
35 echo Main-Class: RLisp/RLispConsole> RLisp.MF
36 echo Class-Path: .\ RLisp.jar>> RLisp.MF
37 jar umf RLisp.MF RLisp/RLisp.jar
38 del RLisp.MF
39 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 keyboard

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

This document, including all the code contained in it, is *copyright* by Ramón Casares.

All the code contained in this document is free, and it can be modified under the [Free Software Foundation](#) prescriptions, also known as “GNU General Public License”. This means that the programs produced from these ones are under the very same prescriptions, so they will also be free and modifiable in the same way.

## Java Index

`aboutAction()` (method in class  
`RLispConsole`): [§2.12 page 49](#)

`accept(File)` (method in class  
`RExtFilter`): [§2.16 page 65](#)

- accept(File, String) (method in class RExtFilter): §2.16 page 65
- actionPerformed(ActionEvent) (method in class RClassTree): §2.15 page 63
- actionPerformed(ActionEvent) (method in class RKeyboard): §2.14 page 56
- actionPerformed(ActionEvent) (method in class RLispConsole): §2.12 page 44
- addClassToTree(DefaultMutableTreeNode, String) (method in class RClassTree): §2.15 page 62
- addJarToTree(DefaultMutableTreeNode, File, String) (method in class RClassTree): §2.15 page 62
- addURL(URL) (method in class RClassLoader): §2.17 page 66
- applyCompound(RPair, RPair, REnvironment) (method in class RLisp): §2.4 page 18
- applyLambda(RPair, RPair, REnvironment) (method in class RLisp): §2.4 page 18
- applyPrimitive(Object, RPair) (method in class RLisp): §2.4 page 18
- applyRho(RPair, RPair, REnvironment) (method in class RLisp): §2.4 page 19
- apply(RPair, REnvironment) (method in class RLisp): §2.4 page 17
- arrayClass(Class, int) (method in class RLispJava): §2.5 page 27
- arrayToString(Object[]) (method in class RLispConsole): §2.12 page 49
- baseURL (variable in class RLispJava): §2.5 page 20
- bind(Object, Object) (method in class RFrame): §2.2 page 9
- callingObject (variable in class RClassTree): §2.15 page 58
- callingObject (variable in class RKeyboard): §2.14 page 52
- car() (method in class RPair): §2.1 page 6
- car (variable in class RPair): §2.1 page 2
- cd (variable in class RClassTree): §2.15 page 59
- cdr() (method in class RPair): §2.1 page 6
- Cdr() (method in class RPair): §2.1 page 6
- CDR() (method in class RPair): §2.1 page 6
- cdr (variable in class RPair): §2.1 page 2
- cds (variable in class RClassTree): §2.15 page 59
- cdsl (variable in class RClassTree): §2.15 page 59
- chooseRoot() (method in class RClassTree): §2.15 page 61
- closeAction() (method in class RLispConsole): §2.12 page 44
- cons(Object, RPair) (method in class RPair): §2.1 page 6
- counter (variable in class RLisp): §2.4 page 14
- counter(int) (method in class RLispInterpreter): §2.6 page 28
- CtoC(Class) (method in class RLispJava): §2.5 page 26
- cutString(String, int) (method in class RLispConsole): §2.12 page 49
- define(Object, Object) (method in class REnvironment): §2.3 page 11
- delims (variable in class RPair): §2.1 page 3
- endLine (variable in class RKeyboard): §2.14 page 52
- ENV (variable in class RLispInterpreter): §2.6 page 28
- equals(Object) (method in class RPair): §2.1 page 8
- evalCond(RPair, REnvironment) (method in class RLisp): §2.4 page 16
- evalDef(RPair, REnvironment) (method in class RLisp): §2.4 page 15
- evalEval(RPair, REnvironment) (method in class RLisp): §2.4 page 15

- evalJarray(RPair, REnvironment)  
(method in class RLispJava): §2.5  
page 22
- evalJava(RPair, REnvironment)  
(method in class RLispJava): §2.5  
page 21
- evalJload(RPair, REnvironment)  
(method in class RLispJava): §2.5  
page 25
- evalJnew(RPair, REnvironment)  
(method in class RLispJava): §2.5  
page 21
- evalJpath(RPair, REnvironment)  
(method in class RLispJava): §2.5  
page 25
- evalJrun(RPair, REnvironment)  
(method in class RLispJava): §2.5  
page 23
- evalJset(RPair, REnvironment)  
(method in class RLispJava): §2.5  
page 24
- evalLambda(RPair, REnvironment)  
(method in class RLisp): §2.4 page 16
- eval(Object, REnvironment) (method  
in class RLisp): §2.4 page 14
- evalRho(RPair, REnvironment)  
(method in class RLisp): §2.4 page 16
- evalRPair(RPair, REnvironment)  
(method in class RLisp): §2.4 page 17
- evalSequence(RPair, REnvironment)  
(method in class RLisp): §2.4 page 16
- evalSet(RPair, REnvironment)  
(method in class RLisp): §2.4 page 15
- evalSpecial(RPair, REnvironment)  
(method in class RLisp): §2.4 page 15
- evalSpecial(RPair, REnvironment)  
(method in class RLispJava): §2.5  
page 21
- Eval(String) (method in class  
RLispInterpreter): §2.6 page 28
- eval(String) (method in class  
RLispInterpreter): §2.6 page 29
- executeObject(Object) (method in  
class RLispConsole): §2.12 page 38
- ext (variable in class RExtFilter): §2.16  
page 64
- extend(RFrame) (method in class  
REnvironment): §2.3 page 11
- fileSeparator (variable in class  
RClassTree): §2.15 page 59
- firstFrame() (method in class  
REnvironment): §2.3 page 11
- getDescription() (method in class  
RExtFilter): §2.16 page 65
- getName(Class) (method in class  
RLispConsole): §2.12 page 43
- getObject() (method in class RButton):  
§2.13 page 51
- getSelectedObject() (method in class  
RClassTree): §2.15 page 58
- getValue(Class) (method in class  
RLispConsole): §2.12 page 41
- getValues(Class[]) (method in class  
RLispConsole): §2.12 page 42
- initLisp() (method in class  
RLispConsole): §2.12 page 36
- inputline (variable in class RKeyboard):  
§2.14 page 52
- isAtom(Object) (method in class  
RPair): §2.1 page 5
- isList(Object) (method in class  
RPair): §2.1 page 5
- isNil() (method in class RPair): §2.1  
page 3
- isNil(Object) (method in class RPair):  
§2.1 page 3
- isPrimitive(Object) (method in class  
RLisp): §2.4 page 18
- isRPair(Object) (method in class  
RPair): §2.1 page 4
- isSpecial(RPair) (method in class  
RLisp): §2.4 page 15
- isSpecial(RPair) (method in class  
RLispJava): §2.5 page 21
- isTrue(Object) (method in class  
RLisp): §2.4 page 14
- keyboardAction() (method in class  
RLispConsole): §2.12 page 45
- keys() (method in class REnvironment):  
§2.3 page 13
- keys() (method in class RFrame): §2.2  
page 9

- keys(Class, boolean) (method in class REnvironment): §2.3 page 13
- keyTyped(KeyEvent e) (method in class RKeyboard): §2.14 page 56
- lineAction(Object) (method in class RLispConsole): §2.12 page 44
- lisp (variable in class RLispInterpreter): §2.6 page 28
- list() (method in class REnvironment): §2.3 page 12
- listAction() (method in class RLispConsole): §2.12 page 45
- loadAction() (method in class RLispConsole): §2.12 page 47
- loader (variable in class RClassTree): §2.15 page 59
- loadFromAction() (method in class RLispConsole): §2.12 page 47
- loadLispAction() (method in class RLispConsole): §2.12 page 47
- lookup(Object) (method in class REnvironment): §2.3 page 11
- lookup(Object) (method in class RFrame): §2.2 page 9
- main(String[]) (method in class RClassTree): §2.15 page 64
- main(String[]) (method in class RKeyboard): §2.14 page 57
- main(String[]) (method in class RLispConsole): §2.12 page 50
- main(String[]) (method in class RLispInterpreter): §2.6 page 29
- nameAction() (method in class RLispConsole): §2.12 page 46
- nil (variable in class RPair): §2.1 page 2
- nth(int) (method in class RPair): §2.1 page 6
- nth(Object, int) (method in class RPair): §2.1 page 7
- ob (variable in class RClassTree): §2.15 page 58
- okTreeAction(Object) (method in class RLispConsole): §2.12 page 45
- pathAction() (method in class RLispConsole): §2.12 page 46
- pdfAction(String) (method in class RLispConsole): §2.12 page 48
- populateTree(DefaultMutableTreeNode, File) (method in class RClassTree): §2.15 page 61
- prettyPrint(Object) (method in class RLispConsole): §2.12 page 50
- Primes.lisp (file): §2.11 page 32
- rbo (variable in class RButton): §2.13 page 51
- RButton (class): §2.13 page 51
- RButton.java (file): §2.13 page 51
- RButton(String, Object) (constructor): §2.13 page 51
- rcl (variable in class RLispConsole): §2.12 page 34
- rcl (variable in class RLispJava): §2.5 page 20
- RClassLoader (class): §2.17 page 66
- RClassLoader.java (file): §2.17 page 66
- RClassLoader(URL[]) (constructor): §2.17 page 66
- RClassTree (class): §2.15 page 57
- RClassTree(ActionListener, File, RClassLoader) (constructor): §2.15 page 60
- RClassTree(ActionListener, RClassLoader) (constructor): §2.15 page 59
- RClassTree(boolean, RClassLoader) (constructor): §2.15 page 59
- RClassTree.java (file): §2.15 page 57
- RClassTree(RClassLoader) (constructor): §2.15 page 60
- readFile(String) (method in class RLispConsole): §2.12 page 37
- REnvironment (class): §2.3 page 10
- REnvironment() (constructor): §2.3 page 10
- REnvironment.java (file): §2.3 page 10
- REnvironment(RFrame) (constructor): §2.3 page 10
- REnvironment(RFrame, REnvironment) (constructor): §2.3 page 11
- restFrames() (method in class REnvironment): §2.3 page 11

- resultToString() (method in class RLispConsole): §2.12 page 50
- RExtFilter (class): §2.16 page 64
- RExtFilter.java (file): §2.16 page 64
- RExtFilter(String) (constructor): §2.16 page 64
- RFrame (class): §2.2 page 8
- RFrame() (constructor): §2.2 page 9
- RFrame.java (file): §2.2 page 8
- RFrame(Object, Object) (constructor): §2.2 page 9
- RKeyboard (class): §2.14 page 52
- RKeyboard(ActionListener, String) (constructor): §2.14 page 53
- RKeyboard.java (file): §2.14 page 52
- RLisp (class): §2.4 page 13
- RLisp2jar.bat (file): §2.19 page 67
- RLispArray.lisp (file): §2.9 page 31
- RLispConsole (class): §2.12 page 33
- RLispConsole.java (file): §2.12 page 33
- RLispConsole(String) (constructor): §2.12 page 34
- RLispInterpreter (class): §2.6 page 28
- RLispInterpreter.java (file): §2.6 page 28
- RLispInterpreter(RLisp) (constructor): §2.6 page 28
- RLispJava (class): §2.5 page 19
- RLisp.java (file): §2.4 page 13
- RLispJava.java (file): §2.5 page 19
- RLispJava.lisp (file): §2.8 page 31
- RLispJava(RClassLoader) (constructor): §2.5 page 20
- RLisp.lisp (file): §2.7 page 30
- RLisp.log (file): §2.18 page 66
- RLispMaths.lisp (file): §2.10 page 32
- RPair (class): §2.1 page 2
- RPair() (constructor): §2.1 page 2
- RPair.java (file): §2.1 page 2
- RPair(Object, Object) (constructor): §2.1 page 3
- RPair(String) (constructor): §2.1 page 3
- saveAction() (method in class RLispConsole): §2.12 page 48
- saveToAction() (method in class RLispConsole): §2.12 page 48
- session() (method in class RLispConsole): §2.12 page 37
- sessionAction() (method in class RLispConsole): §2.12 page 45
- setLogFile(String) (method in class RLispConsole): §2.12 page 37
- set(Object, Object) (method in class REnvironment): §2.3 page 11
- setObject(Object) (method in class RButton): §2.13 page 51
- setRoot(File) (method in class RClassTree): §2.15 page 59
- StoC(String) (method in class RLispJava): §2.5 page 26
- StoO(Class, String) (method in class RLispJava): §2.5 page 27
- substitute(Object, Object) (method in class RPair): §2.1 page 7
- t (variable in class RLisp): §2.4 page 13
- toArray() (method in class RPair): §2.1 page 5
- toArray(Object) (method in class RPair): §2.1 page 5
- toArray(String) (method in class RLispConsole): §2.12 page 49
- Tokenize(String) (method in class RPair): §2.1 page 3
- tokenize(String) (method in class RPair): §2.1 page 4
- toString() (method in class REnvironment): §2.3 page 12
- toString() (method in class RFrame): §2.2 page 10
- toString() (method in class RLisp): §2.4 page 19
- toString() (method in class RLispInterpreter): §2.6 page 29
- toString() (method in class RLispJava): §2.5 page 27
- toString() (method in class RPair): §2.1 page 7
- toString(boolean) (method in class REnvironment): §2.3 page 12



toString(boolean) (method in class RPair): §2.1 page 8	valueChanged(TreeSelectionEvent) (method in class RClassTree): §2.15 page 63
tree (variable in class RClassTree): §2.15 page 58	windowClosing(WindowEvent) (method in class RLispConsole): §2.12 page 43
treeAction() (method in class RLispConsole): §2.12 page 46	writeln() (method in class RLispConsole): §2.12 page 38
unnamedAction() (method in class RLispConsole): §2.12 page 45	writeln(String) (method in class RLispConsole): §2.12 page 38
	write(String) (method in class RLispConsole): §2.12 page 38

## Table of Contents

<b>RLisp: The Program</b> . . . . .	<b>1</b>
<b>1 Introduction</b> . . . . .	<b>1</b>
<b>2 Files</b> . . . . .	<b>2</b>
2.1 RPair.java . . . . .	2
2.2 RFrame.java . . . . .	8
2.3 REnvironment.java . . . . .	10
2.4 RLisp.java . . . . .	13
2.5 RLispJava.java . . . . .	19
2.6 RLispInterpreter.java . . . . .	28
2.7 RLisp.lisp . . . . .	30
2.8 RLispJava.lisp . . . . .	31
2.9 RLispArray.lisp . . . . .	31
2.10 RLispMaths.lisp . . . . .	32
2.11 Primes.lisp . . . . .	32
2.12 RLispConsole.java . . . . .	33
2.13 RButton.java . . . . .	51
2.14 RKeyboard.java . . . . .	52
2.15 RClassTree.java . . . . .	57
2.16 RExtFilter.java . . . . .	64
2.17 RClassLoader.java . . . . .	66
2.18 RLisp.log . . . . .	66
2.19 RLisp2jar.bat . . . . .	67
<b>3 To-Do List</b> . . . . .	<b>68</b>
3.1 To enhance the list command . . . . .	68
3.2 To enhance the session command . . . . .	68
3.3 To enhance the input of arrays . . . . .	68
3.4 To add edit buttons to the keyboard . . . . .	68
<b>4 Legalities</b> . . . . .	<b>68</b>
<b>Java Index</b> . . . . .	<b>68</b>
<b>Table of Contents</b> . . . . .	<b>73</b>