# Exceptions and Meta-Level Programming in the Green Language

José de Oliveira Guimarães
Departamento de Computação - UFSCar
São Carlos - SP, Brazil
13565-905
jose@dc.ufscar.br

Reflective programs have a base and a meta level code. The base level is responsible for the program functionality, what is described in its specification. The meta level helps the base level do its job by providing services like concurrency control, transparent distribution, fault tolerance, and so on.

There are two types of reflection: introspective and behavioral. Introspective reflection allows the program to see its own structure. Behavioral reflection allows the program to change some parts of itself or its runtime system. At runtime, behavioral reflection is mainly implemented through metaobjects.

A language supporting *introspective reflection* supplies a library which allows one to discover, at runtime, the class of an object, the methods of a class, the superclass of a class, and so on. The Green Introspective Reflection Library allows us to examine the *stack of catch objects* at runtime, the methods of each object, their parameters, etc. Green also supports metaobjects. A metaobject is attached to an object to control the messages it receives. Every message sent to the object is redirected to a specific method of the metaobject. One can attach a metaobject to a *catch object* and then intercept the calling of a `throw` method, maybe changing the exception handling.

The interception (with metaobjects) and introspection of catch objects have obvious uses by software tools as debuggers. A debugger can easily show the *stack of catch objects* and the exceptions they can handle. It can attach metaobjects to *catch objects* to intercept exception signaling even if the source code that signals the exceptions is not available.

The code below shows an example of the interaction IRL/Exception Handling System. The program starts at method `run` of class `Program`. Method `print` prints in the standard output the stack of catch objects. For each object, it prints all of its `throw` methods. This program is one of the tests used for validating the Green Compiler which is available at
`http://www.dc.ufscar.br/~jose/green/green.htm`.

```
/*
  ok-sin24.g

  Tests if catch objects are removed from the stack of catch objets.
*/

class A subclassOf Exception
end

class C subclassOf Exception
```

```
end

class B
   public:
      proc throw( exc : A )
         begin
         Out.writeln("Catch class B: exception A thrown");
         end
      proc throw( exc : C )
         begin
         Out.writeln("Catch class B: exception C thrown");
         end
end

class D
   public:
      proc throw( exc : C )
         begin
         Out.writeln("Catch class D: exception C thrown");
         end
      proc throw( exc : A )
         begin
         Out.writeln("Catch class D: exception A thrown");
         end
end

class E
   public:
      proc throw( exc : C )
         begin
         Out.writeln("Catch class E: exception C thrown");
         end
end

object Program
   public:
      proc m()(exception : B)
         begin
         p();
         end
      proc p()(exception : B)
         begin
         exception.throw( A.new() );
         end
         // the execution of program starts at method run
      proc run()
         begin
         try(B.new())
```

```
            try(D.new())
            end
         end
         Out.writeln("Outside any try");
         print();
         Out.writeln("After print");
         try(B.new())
            try(D.new())
               try(E.new())
                  Out.writeln("Inside triple try");
                  print();
                  //m();
               end
            end
         end
         end

      proc print()
           // print prints the stack of catch objects. For each
           // catch object, it prints all of its throw methods.
            var i : integer;
         begin
         i = 0;
           // get the stack of catch objects
         var stack : Stack(Catch) = Runtime.getCatchObjectStack();
         var iter : Iter(Catch) = stack.getIter();
         while iter.more() do
            begin
              // get object by object form the stack of catch objects
            var aCatch : Catch = iter.next();
            Out.write(i);
            ++i;
              // write the name of the class of the catch object aCatch
            Out.writeln( aCatch.getClassInfo().getName() );
              // the for below writes the throw methods of catch object aCatch
            var pm : array(ClassMethodInfo)[] = aCatch.getClassInfo().getPublicMethods();
            for j : integer = 0 to pm.getSize() - 1 do
               if pm[j].getName().compareTo("throw") == 0
               then
                  Out.write("    throw(");
                  var ci : array(ClassInfo)[] = pm[j].getParameterTypes();
                  Out.write(ci[0].getName());
                  Out.writeln(")");
               endif
            end
         end
end
```