Goto Label and Switch

Implementing Simula Goto Statement
in Java using Exception handling and
ByteCode Engineering.

Simula Standard says:

A goto-statement interrupts the normal sequence of operations, by defining its successor explicitly by the value of a designational expression (i.e. a program point). Thus the next statement to be executed is the one at this program point.

The program point referenced by the designational expression must be visible at the goto-statement

Jeg tror vi må diskutere hva "visible" egentlig bety i denne sammenhengen. Kommer tilbake til det.

Java does not support labels like Simula. The Java Virtual Machine (JVM), however, has labels. A JVM-label is simply a relative byte-address within the byte-code of a method.

This implementation will generate Java-code which is prepared for Byte Code Engineering.
See chapter .....

Suppose a Simula program containing labels and goto like this:

```
Begin
  L: ...
     goto L;
     ...
 End;
```

This will be coded as:

```
public final class adHoc00 extends BASICIO$ {
    public int prefixLevel() { return(0); }
    final LABQNT$ L=new LABQNT$(this,0,1); // Local Label #1=L
    public adHoc00(RTObject$ staticLink) {
        super(staticLink);
        BBLK();
        BPRG("adHoc00");
    } // End of Constructor

    // SimulaProgram Statements
    public RTObject$ STM() {
        adHoc00 THIS$=(adHoc00)CUR$;
        LOOP$:while(JTX$>=0) {
            try {
                JUMPTABLE$(JTX$); // For ByteCode Engineering
                // Statements ....
                LABEL$(1); // L
                // Statements ....
                throw(L); // GOTO EVALUATED LABEL
                // Statements ....
                break LOOP$;
            }
            catch(LABQNT$ q) {
                CUR$=THIS$;
                if(q.SL$!=CUR$ || q.prefixLevel!=0) {
                    CUR$.STATE$=OperationalState.terminated;
                    throw(q); // Re-throw exception for non-local Label
                }
                JTX$=q.index; continue LOOP$; // GOTO Local L
            }
        }
        EBLK();
        return(null);
    } // End of SimulaProgram Statements
}
```

This needs a lot of explanation ....

Label Quantities.

At source line 3. the label 'L' is declared like this:

```
final LABQNT$ L=new LABQNT$(this,0,1); // Local Label #1=L
```

Where LABQNT$ is defined by:

```
public final class LABQNT$ extends RuntimeException
{ public RTObject$ SL$; // Static link, block in which the label is defined.
  public int prefixLevel; // PrefixLevel for classes, zero otherwise.
  public int index; // Ordinal number of Label within its Scope(staticLink).

  public LABQNT$(RTObject$ SL$,int prefixLevel,int index)
  { this.SL$=SL$; this.prefixLevel=prefixLevel; this.index=index; }
}
```

A goto-statement is simply coded as:

```
throw(L); // GOTO EVALUATED LABEL
```

And this exception is catched and tested (lines 23 - 38) throughout the operating chain. If the label does not belong to this block instance the exception is re-thrown.

In the event of no matching block instances the exception is caught by an UncaughtExceptionHandler like this:

```
public void uncaughtException(Thread thread, Throwable e) {
    if(e instanceof LABQNT$) {
        // POSSIBLE GOTO OUT OF COMPONENT
        RTObject$ DL=obj.DL$;
        if(DL!=null && DL!=CTX$) {
            DL.PENDING_EXCEPTION$=(RuntimeException)e;
            DL.resumeThread();
        } else {
            ERROR("Illegal GOTO "+e);
            ...
        } else ...
}
```

Thus, when a QPS-component is left we raise the PENDING_EXCEPTION flag and resume next operating component. The resume-operations will re-trow the exception within its Thread.

For further details see the source code of RTObject.java

Byte Code Engineering.

The Apache Commons Byte Code Engineering Library (BCEL) is used to modify the byte code to allow very local goto statements.

Apache says:

The Byte Code Engineering Library (Apache Commons BCEL™) is intended to give users a convenient way to analyze, create, and manipulate (binary) Java class files (those ending with .class). Classes are represented by objects which contain all the symbolic information of the given class: methods, fields and byte code instructions, in particular.

Such objects can be read from an existing file, be transformed by a program (e.g. a class loader at run-time) and written to a file again. An even more interesting application is the creation of classes from scratch at run-time. The Byte Code Engineering Library (BCEL) may be also useful if you want to learn about the Java Virtual Machine (JVM) and the format of Java .class files.

More info at: http://commons.apache.org/proper/commons-bcel/

To easily modify the code, the compiler generates certain method call in the .java file:

```
- LABEL$(n); // Label #n
```

This method-call is used to signal the occurence of a label. The byte-code address is collected and some instruction are removed. The parameter 'n' is the label's ordinal number.

I.e. Try to locate the instruction sequence:

```
PREV-INSTRUCTION
ICONST n
INVOKESTATIC LABEL$
NEXT-INSTRUCTION
```

Pick up the number 'n', remember address and remove the two middle instruction.

```
- JUMPTABLE$(JTX$); // For ByteCode Engineering
```

This method-call is a placeholder for where to put in a Jump-Table.

Try to locate the instruction sequence:

```
PREV-INSTRUCTION
GETFIELD JTX$
INVOKESTATIC JUMPTABLE$
NEXT-INSTRUCTION
```

And replace it by the instruction sequence:

```
PREV-INSTRUCTION
GETFIELD JTX$
TABLESWITCH ... uses the addresses collected for labels.
NEXT-INSTRUCTION
```

For further details see the source code of ByteCodeEngineering.java

Goto virtual label

```
BEGIN

   class A; virtual: label L;
   begin
      goto L;
   end;

   A class B;
   begin
      L: outtext("OK");
   end;

   ref(B) x;
   x:-new B;

END;
```

Kompilern gir følgende warning:

```
LINE 3: WARNING: Goto Virtual label L is not fully implemented, may result in
Runtime ERROR
```

Og ved runtime får vi:

```
Thread:main[Simula adHoc01]: SIMULA RUNTIME ERROR: NOT IMPLEMENTED: Goto Virtual
Label
```

Kan vi leve med dette ?

Kan vi påstå at L i subklasse ikke er synlig ved goto ?