EXAMPLE 9-2. Javac's nonoptimal compiling

```
public int function()        public int function();
{                               0:   iconst_1
    boolean a = true;           1:   istore_1
    if (a)                      2:   iload_1
        return 1;               3:   ifeq    8
    else                        6:   iconst_1
        return 0;               7:   ireturn
}                               8:   iconst_0
                                9:   ireturn
```

This minimal optimization is easily justified because most of the optimization work is being left as an exercise for the latter stages of compilation. In JPC not only do we have this reasoning, but we also have the burden of extra time pressure:

- We want minimal overhead compiling in order to avoid stealing CPU cycles from the other emulation threads.
- We want minimal latency compiling so that the interpreted class is replaced as soon as possible. A high-latency compiler may find that the code is no longer needed by the time it has finished the compile.

Simple code generation

The compiling task in JPC is now a simple matter of translating the microcodes of a single interpreted basic block into a set of Java bytecodes. In the first approximation we will assume that the basic block cannot throw exceptions. This means that a basic block is a strictly defined basic block and has exactly one entry point and one exit point. Each variable modified by a given basic block can now be expressed as a function of the set of input registers and memory state. Internally in JPC, we collectively represent these functions as a single directed acyclic graph.

Graph source
: Sources represent input data in the form of register values or instruction immediates.

Graph sink
: Sinks represent output data in the form of register values and memory or I/O port writes. For each state variable that the block affects, there will be one sink.

Graph edge
: Edges represent variable value propagation within the graph.

Graph node
: Nodes represent operations performed on incoming edges whose results propagate along outgoing edges. In JPC, the operations are a single-state modification component of an interpreted microcode. Hence one microcode may map to multiple graph nodes if it affects multiple variables.