which can be further reduced to

```
ldc 86400
```

Table 14.1 lists some other rules for a peephole optimizer. The variables `x` and `y` stand for any field or local variable. For the operations `aload` and `astore`, you may perform the same operations on the corresponding `float` and `int` instructions.

TABLE 14.1: *Poophole optimization rules*

Replace	With	Reason
`dup` `swap`	`dup`	The result of a `dup` is two of the same element; swapping produces an identical result.
`swap` `swap`		A double swap accomplishes nothing.
`iinc 1 1` `iinc 1 1`	`iinc 1 2`	Since it takes just as long to increment by 2 as to increment by 1 (on most systems), you might as well do the increment just once.
`nop`		A `nop` instruction has no effect, so you might as well eliminate it.
`getfield x` `getfield x`	`getfield x` `dup`	Two `getfield` instructions one immediately after the other leave no opportunity for the field value to change, so you can replace the second one with a `dup`. *Exception:* If the field is marked `volatile`, then its value may be changed by some other thread, and this optimization would be in error.
`aload x` `aload x`	`aload x` `dup`	Some systems have special fast operations for duplicating the top element of the stack.
`astore x` `aload x`	`dup` `astore x`	Some systems have special fast operations for duplicating the top element of the stack.
`aload x` `astore x`		These instructions cancel each other out.
`astore x` `astore x`	`astore x` `pop`	Storing into the same variable twice is unnecessary; only the second store needs to be performed.
`aload x` `pop`		These instructions cancel each other out.

`aload x`
`aload y`
`areturn`
	`aload y`
	`areturn`
		Since the value of *x* is left on the top of the stack at the time of the return, there is no reason to push it. This optimization applies to any instruction except a method invocation, branch instruction, or storing into a field.