Constant expressions

Essentially a *constant expression* is an expression of some primitive type or of type String that can be evaluated at compile-time. This is because:

(1) Its operands are either
   a. literals of a primitive type or String (e.g. 1, 5.0E20, "xyz") or
   b. variables whose values can be evaluated at compile-time and don't change, e.g. a variable declared like this: public static final int b= 2;
(2) Its operations are the usual ones (+, -, /, *, &&, ||, !, ==, <=, etc. but not ++ and --)
(3) Its evaluation does not cause an exception to be thrown.

The Java compiler can replace such expressions by their values, making the program more efficient. This process can also help catch errors at compile-time.

For example, the following method p will not compile because the return statement is unreachable because the while-loop condition evaluates to true. This is detected at compile-time. It is, essentially, a syntactic error. Delete keyword **final** from the declaration of h; the loop condition is no longer a constant expression, and method p can be compiled.

```
public static void p() {
   int k= 5;
   final int h= 6;
   while (true  &&  h < Integer.MAX_VALUE) { k= k+1; }
   return;
}
```

Processing constant String expressions

Suppose two or more constant String expressions yield the same value. Only *one* object of class String will be created and used as the value of all those expressions. This saves space. The Java specification says that method String.intern is used to do this; have a look at this method.

As an example, a call of method q below prints true:

```
public static void q() {
   System.out.println("xy"  ==  "x" + "y");
}
```

A caution with DrJava's interactions pane

The interactions pane in the IDE DrJava is special, and the above paragraph does not apply to it. Execute the following statement in the interactions pane and it prints *false*.

```
System.out.println("xy" == "x" + "y");
```