Constant-time operations

An operation or method takes constant time if the time it takes to carry it out does not depend on the size of its operands.

For example, an array element reference b[i] takes constant time, but printing out all elements of array b is not constant time but instead takes time proportional to the size of b. Also, this assignment takes constant time:

```
b[i]= b[i] + 2
```

But for a String variable s, the assignment

```
s= s + 'c'
```

does *not* take constant time in Java because a new String object has to be created and all the characters of s and 'c' have to be copied into this new object, Thus, this assignment takes time proportional to the length of s.

Also, testing whether two Strings are equal can take time proportional to their lengths.

If method contains only assignment statements that are constant time, if-statements, conditional expressions, and the like, chances are that it is a constant-time operation or method. If it contains a loop, chances are that it is not constant-time. But you have to be careful, as the following example shows.

Consider an array b and the following statements:

```
int sum= 0;
for (int k= 0; k < Math.min(50, b.length); k= k+1)
    sum= sum + b[k];
```

Since the loop iterates at most 50 times, and since each expression or assignment in it is a constant-time operation, the code itself is constant time. It has a maximum of 50 array-element additions, for example, no matter how big the array is. But the following loop with initialization is not constant time but takes time proportional to the size of b.

```
int sum= 0;
for (int k= 0; k < b.length; k= k+1)
    sum= sum + b[k];
```