Let's look into these statements a little further. First of all, remember that any code you can put into a normal method you can put into `finalize()`. For example, in the `finalize()` method you could write code that passes a reference to the object in question back to another object, effectively *ineligible-izing* the object for garbage collection. If at some point later on this same object becomes eligible for garbage collection again, the garbage collector can still process the object and delete it. The garbage collector, however, will remember that, for this object, `finalize()` already ran, and it will not run `finalize()` again.

CERTIFICATION SUMMARY

This chapter covered a wide range of topics. Don't worry if you have to review some of these topics as you get into later chapters. This chapter includes a lot of foundational stuff that will come into play later.

We started the chapter by reviewing the stack and the heap; remember that local variables live on the stack and instance variables live with their objects on the heap.

We reviewed legal literals for primitives and `String`s, and then we discussed the basics of assigning values to primitives and reference variables, and the rules for casting primitives.

Next we discussed the concept of scope, or "How long will this variable live?" Remember the four basic scopes, in order of lessening life span: static, instance, local, and block.

We covered the implications of using uninitialized variables, and the importance of the fact that local variables MUST be assigned a value explicitly. We talked about some of the tricky aspects of assigning one reference variable to another and some of the finer points of passing variables into methods, including a discussion of "shadowing."

Finally, we dove into garbage collection, Java's automatic memory management feature. We learned that the heap is where objects live and where all the cool garbage collection activity takes place. We learned that in the end, the JVM will perform garbage collection whenever it wants to. You (the programmer) can request a garbage collection run, but you can't force it. We talked about garbage collection only applying to objects that are eligible, and that eligible means "inaccessible from any live thread." Finally, we discussed the rarely useful `finalize()` method and what you'll have to know about it for the exam. All in all, this was one fascinating chapter.

TWO-MINUTE DRILL

Here are some of the key points from this chapter.

Stack and Heap

- ❑ Local variables (method variables) live on the stack.
- ❑ Objects and their instance variables live on the heap.

Literals and Primitive Casting (OCA Objective 2.1)

- ❑ Integer literals can be binary, decimal, octal (such as `013`), or hexadecimal (such as `0x3d`).
- ❑ Literals for `long`s end in `L` or `l`.
- ❑ Float literals end in `F` or `f`, and `double` literals end in a digit or `D` or `d`.
- ❑ The `boolean` literals are `true` and `false`.
- ❑ Literals for `char`s are a single character inside single quotes: `'d'`.

Scope (OCA Objective 1.1)

- ❑ Scope refers to the lifetime of a variable.
- ❑ There are four basic scopes:
  - ❑ Static variables live basically as long as their class lives.
  - ❑ Instance variables live as long as their object lives.
  - ❑ Local variables live as long as their method is on the stack; however, if their method invokes another method, they are temporarily unavailable.
  - ❑ Block variables (for example, in a `for` or an `if`) live until the block completes.