> The classics are books that exert a peculiar influence, both when they refuse to be eradicated from the mind and when they conceal themselves in the folds of memory, camouflaging themselves as the collective or individual unconscious.
>
> Every rereading of a classic is as much a voyage of discovery as the first reading.
>
> A classic is a book that has never finished saying what it has to say.
>
> The classics are the books that come down to us bearing the traces of readings previous to ours, and bringing in their wake the traces they themselves have left on the culture or cultures they have passed through (or, more simply, on language and customs).
>
> A classic does not necessarily teach us anything we did not know before. In a classic we sometimes discover something we have always known (or thought we knew), but without knowing that this author said it first, or at least is associated with it in a special way. And this, too, is a surprise that gives much pleasure, such as we always gain from the discovery of an origin, a relationship, an affinity.
>
> The classics are books which, upon reading, we find even fresher, more unexpected, and more marvelous than we had thought from hearing about them.
>
> A classic is a book that comes before other classics; but anyone who has read the others first, and then reads this one, instantly recognizes its place in the family tree.

Books are not computer languages, of course, and yet these definitions may also apply to our task.

Everything Is an Object

Today's popular object-oriented computer languages (C++, Java, and C#) are not purely object-oriented. Not everything is an object. Some types are *primitive*. Hence, for example, we cannot subclass an integer. Arithmetic is performed the usual way on pure numbers, not by invoking methods on objects. This brings performance benefits, and it may make things easier for people coming to object orientation from a procedural language background.

But if we do decide to have everything as an object, then the situation changes drastically. In Smalltalk, integers up to 31 bits long are instances of the `SmallInteger` class (in fact, there is an abstract `Integer` class and its subclasses `SmallInteger`, `LargePositiveInteger`, and `LargeNegativeInteger`, with conversions carried out automatically by the system as necessary). We can perform ordinary arithmetic on them, but we can also do more. The `SmallInteger` class offers no less than 670 methods (or *selectors*, in Smalltalk parlance), as we can easily find out with the following code snippet:

```
SmallInteger allSelectors size
```