It is instructive to examine how this code works. `allSelectors` is a class selector that does what its name suggests. It returns all the selectors of a class in a `Set` (in fact, an `IdentitySet`, but this does not make any difference to us here). The `Set` is itself a first-class object with its own selectors; one of them, `size`, tells us the number of elements it contains.

Among the `SmallInteger` selectors we find the expected arithmetic operators. We also find trigonometric and logarithmic functions, functions to compute factorials, greatest common divisor, and least common multiple. There are functions for bit manipulation, and many others.

What we meet as integer primitives in other languages are actually `SmallInteger` instances in Smalltalk. This explains why:

```
2 raisedTo: 5
```

works, and more intriguingly, why the following works as well:

```
(7 + 3) timesRepeat: [ Transcript show: 'Hello, World'; cr ]
```

Selectors that take arguments have a colon (`:`) appended to them before each argument. Selectors standing for arithmetic and logical operators, such as `+` just shown, are exempt from that rule. `Transcript` is a class representing something similar to the system console. `cr` stands for carriage return, and a semicolon (`;`) cascades messages, so `cr` is sent to `Transcript`. We can evaluate this code in an interpreter window (usually called a *workspace* in Smalltalk) and see directly what happens.

Of course, not all 670 `SmallInteger` methods are defined in `SmallInteger`. `SmallInteger` is part of an inheritance hierarchy, shown in Figure 14-1, where we also see the number of selectors for each of the `SmallInteger` ancestors. Most of the selectors are inherited by `Object`, and understanding what `Object` offers explains a great deal of Smalltalk architecture (in Squeak, the actual root of the hierarchy is `ProtoObject`, but this is a minor detail).

Instances of `Object` have at their disposal comparison selectors (both equality, denoted by `=`, and identity, denoted by `==`); selectors for making copies (both deep, by invoking `deepCopy`, and shallow, by invoking `shallowCopy`); and selectors for printing on streams, error handling, debugging, message handing, and others. Only a few of the hundreds of `Object` methods are of use for everyday programming. Methods in Smalltalk are organized in groups called *protocols*, and checking the protocol descriptions makes finding methods easier.

Methods themselves are first-class objects in Smalltalk. To see what this means in terms of the overall architecture, take the code:

```
aRectangle intersects: anotherRectangle
```

where `aRectangle` and `anotherRectangle` are instances of class `Rectangle`. When `aRectangle`, the *receiver* (as objects that receive a message are called in Smalltalk) receives the `intersects:` message, the Smalltalk interpreter will do the following (Conroy and Pelegri-Llopart 1983):

Download at WoweBook.Com