5.2 THE BASICS

Ruby has variables, assignments, the usual control structures, and even exceptions. As in Python, there are no primitives, only objects; you can find the unique identifier of any object x with x`.object_id`. Every object has a unique class, which can be found with x`.class`:

```
fail unless nil.class == NilClass
fail unless false.class == FalseClass
fail unless true.class == TrueClass
fail unless 3.class == Integer
fail unless (2**1000).class == Integer
fail unless 2.0.class == Float
fail unless :dog.class == Symbol
fail unless "dog".class == String
fail unless (1..5).class == Range
fail unless [1,2,3,4,5].class == Array
fail unless {x: 1, y: 2}.class == Hash
fail unless {'x' => 1, 'y' => 2}.class == Hash
```

Ruby features a large number of built-in classes. A class can have at most one **superclass**. Superclasses allow us to express `IS-A` relationships: each of the expressions `5.is_a? Integer`, `5.is_a? Numeric`, `5.is_a? Object`, and `5.is_a? BasicObject` evaluate to `true`. Figure 5.1 shows part of this hierarchy.

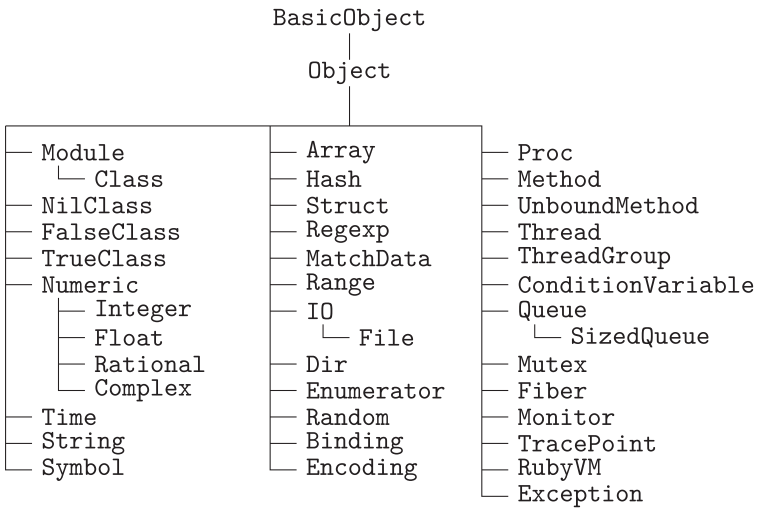

Figure 5.1 Part of the standard class hierarchy in Ruby

Like Python, Ruby classifies floating point numbers differently than integers, and allows integer values to grow as large as available memory will allow them to grow: computing `2 ** 100` happily produces 1267650600228229401496703205376. The language is "strongly typed" in the sense of allowing very few implicit coercions: you can't, for example, add strings and numbers. But everything *can* be coerced to a boolean. Like Lua, and very unlike Python, the *only* expressions that are falsy are `false` and `nil`.