▫ An object is composed of a bunch of instance variables and a link to a class.
オブジェクトはインスタンス変数とクラスへのリンクで構成されている。

▫ The methods of an object live in the object's class. (From the point of view of the class, they're called instance methods.)
オブジェクトのメソッドはオブジェクトのクラスに生息している。（クラスから見た場合、それはインスタンスメソッドと呼ばれる。）

▫ The class itself is just an object of class Class. The name of the class is just a constant.
クラスはClassクラスのオブジェクトである。クラスの名前は定数である。

▫ Class is a subclass of Module. A module is basically a package of methods. In addition to that, a class can also be instantiated (with new) or arranged in a hierarchy (through its superclass).
ClassはModuleのサブクラスである。モジュールは、基本的にメソッドがまとめられたものである。そのことに加えて、クラスは（newを使って）インスタンス化したり（superclassを使ってたどれる）階層構造を作ったりできる。

▫ Constants are arranged in a tree similar to a file system, where the names of modules and classes play the part of directories and regular constants play the part of files.
定数はファイルシステムのようにツリー状に構成されている。モジュールやクラスの名前がディレクトリの役割、（それら以外の通常の）定数がファイルの役割を担っている。

▫ Each class has an ancestors chain, beginning with the class itself and going up to BasicObject.
クラスには、クラスから始まりBasicObjectに至るまでの継承チェーンがある。

▫ When you call a method, Ruby goes right into the class of the receiver and then up the ancestors chain, until it either finds the method or reaches the end of the chain.
メソッドを呼び出すと、Rubyはレシーバのクラスに向かって右へ進み、継承チェーンを上へ進む。それは、そのメソッドが見つかるか、継承チェーンが終わるまで続く。

▫ When you include a module in a class, the module is inserted in the ancestors chain right above the class itself. When you prepend the module, it is inserted in the ancestors chain right below the class.
クラスにモジュールをインクルートすると、そのモジュールはクラスの継承チェーンの真上に挿入される。モジュールをプリペンドすると、そのモジュールはクラスの継承チェーンの真下に挿入される。

▫ When you call a method, the receiver takes the role of self.
メソッドを呼び出すと、そのレシーバはselfの役割を担う。

▫ When you're defining a module (or a class), the module takes the role of "self".
モジュール（やクラス）を定義すると、そのモジュールがselfの役割を担う。

▫ Instance variables are always assumed to be instance variables of "self".
インスタンス変数は常にselfのインスタンス変数と見なされる。

▫ Any method called without an explicit receiver is assumed to be a method of "self".
レシーバを明示していないメソッド呼び出しは、selfのメソッド呼び出しだと見なされる。

▫ There is only one kind of object—be it a regular object or a module.
オブジェクトは1種類しかない（それが通常のオブジェクトでもモジュールでも）。

▫ There is only one kind of module—be it a regular module, a class, or a singleton class.
モジュールは1種類しかない（それが通常のモジュールでもクラスでも特異クラスでも）。

▫ There is only one kind of method, and it lives in a module—most often in a class.
メソッドは1種類しかなく、モジュール（通常はクラス）に生息している。

▫ Every object, classes included, has its own “real class,” be it a regular class or a singleton class.
すべてのオブジェクトは（クラスも含めて）「本当のクラス」を持っている（それが通常のクラスでも特異クラスでも）。

▫ Every class, with the exception of BasicObject, has exactly one ancestor— either a superclass or a module. This means you have a single chain of ancestors from any class up to BasicObject.
（BasicObjectを除いた）すべてのクラスは、（スーパークラスまたはモジュールという）祖先を1つだけ持っている。つまり、あらゆるクラスからBasicObjectまでの1本の継承チェーンが存在するということだ。

▫ The superclass of the singleton class of an object is the object’s class. The superclass of the singleton class of a class is the singleton class of the class’s superclass. (Try repeating that three times, fast!)
オブジェクトの特異クラスのスーパークラスは、オブジェクトのクラスである。クラスの特異クラスのスーパークラスは、クラスのスーパークラスの特異クラスである（3回繰り返してみよう！もっと速く！）。

▫ When you call a method, Ruby goes “right” in the receiver’s real class and then “up” the ancestors chain. That’s all there is to know about the way Ruby finds methods.
メソッドを呼び出すと、Rubyはレシーバの本当のクラスに向かって「右へ」進み、継承チェーンを「上へ」進む。Rubyのメソッド探索について知るべきことは以上だ。

引用元: [Ruby Metaprogramming Tokyo](https://github.com/yasslab/ruby-metaprogramming-tokyo)
著者: Paolo Perrottaさん ([@nusco](https://twitter.com/nusco))
翻訳: 角 征典さん ([@kdmsnr](https://twitter.com/kdmsnr))