**Operators**

This chapter covers in detail implementing your own *operators*—methods with the same syntax as the familiar mathematical operators. Operators are often used to build *domain-specific languages*—minilanguages embedded inside Scala. *Implicit conversions* (type conversion functions that are applied automatically) are another tool facilitating the creation of domain-specific languages. This chapter also discusses the special methods apply, update, and unapply. We end the chapter with a discussion of *dynamic invocations*—method calls that can be intercepted at runtime, so that arbitrary actions can occur depending on the method names and arguments.

-Identifiers contain either alphanumeric or operator characters.

Unary and binary operators are method calls.

• Operator precedence depends on the first character, associativity on the last.

• The apply and update methods are called when evaluating expr(args).

• Extractors extract tuples or sequences of values from an input .

• Types extending the Dynamic trait can inspect the names of methods and arguments at runtime.

**Identifiers**

The name of variables, functions, classes and so on are collectively called ***identifiers***. In Scala, you have more choices for forming identifiers than in other languages. Of course, you can follow the time-honored pattern: sequences of alphanumeric characters, starting with an alphabetic carácter ora n underscore, such as input1 or next\_token.

As in Java, Unicode characters are allowed. For example, quantité or ποσó are valid identifiers, In addition, you can use operator characters in identifiers:

-The ASCII characters ! # % & \* + - / : < = > ? @ \ ^ | ~ that are not letters, digits, underscore, the .,; punctuation marks, parentheses () [] {}, or quotation marks ' ` ".

For example, \*\* and √ are valid identifiers. With the definition

val √ = scala.math.sqrt \_

-You can write √(2) to compute a square root. This may be a good idea, provided one’s programming environment makes it easy to type the symbol.

The identifiers @ # : = \_ => <- <: <% >: ⇒ ← are reserved in the specification, and you cannot redefine them.

If the carácter ‘=’ is reserved as a operator, it shoud be used as =(), shoudn’t it?

Finally, you can include just about any sequence of characters in backquotes. For example, val `val` = 42

That example is silly, but backquotes can sometimes be an “escape hatch.” For example, in Scala, yield is a reserved word, but you may need to access a Java method of the same name. Backquotes to the rescue: Thread.`yield`().

**Infix operators**

You can write