```
sub display {
    my $self = shift;
    my @keys = @_ ? @_ : sort keys %$self;
    foreach $key (@keys) {
        print "\t$key => $self->{$key}\n";
    }
}
```

44.1.4 Method Invocation

For various historical and other reasons, Perl offers two equivalent ways to write a method call. The simpler and more common way is to use the arrow notation:

```
my $fred = Critter->find("Fred");
$fred->display("Height", "Weight");
```

You should already be familiar with the use of the -> operator with references. In fact, since `$fred` above is a reference to an object, you could think of the method call as just another form of dereferencing.
Whatever is on the left side of the arrow, whether a reference or a class name, is passed to the method subroutine as its first argument. So the above code is mostly equivalent to:

```
my $fred = Critter::find("Critter", "Fred");
Critter::display($fred, "Height", "Weight");
```

How does Perl know which package the subroutine is in? By looking at the left side of the arrow, which must be either a package name or a reference to an object, i.e. something that has been blessed to a package. Either way, that's the package where Perl starts looking. If that package has no subroutine with that name, Perl starts looking for it in any base classes of that package, and so on.
If you need to, you *can* force Perl to start looking in some other package:

```
my $barney = MyCritter->Critter::find("Barney");
$barney->Critter::display("Height", "Weight");
```

Here `MyCritter` is presumably a subclass of `Critter` that defines its own versions of find() and display(). We haven't specified what those methods do, but that doesn't matter above since we've forced Perl to start looking for the subroutines in `Critter`.
As a special case of the above, you may use the `SUPER` pseudo-class to tell Perl to start looking for the method in the packages named in the current class's `@ISA` list.

```
package MyCritter;
use base 'Critter';    # sets @MyCritter::ISA = ('Critter');

sub display {
    my ($self, @args) = @_;
    $self->SUPER::display("Name", @args);
}
```

It is important to note that `SUPER` refers to the superclass(es) of the *current package* and not to the superclass(es) of the object. Also, the `SUPER` pseudo-class can only currently be used as a modifier to a method name, but not in any of the other ways that class names are normally used, eg:

```
something->SUPER::method(...);      # OK
SUPER::method(...);                 # WRONG
SUPER->method(...);                 # WRONG
```

Instead of a class name or an object reference, you can also use any expression that returns either of those on the left side of the arrow. So the following statement is valid:

```
Critter->find("Fred")->display("Height", "Weight");
```

and so is the following:

```
my $fred = (reverse "rettirC")->find(reverse "derF");
```