```
## in Animal
sub name {
  my $either = shift;
  ref $either ?
    $either->{Name} :
    "an unnamed $either";
}
```

And of course `named` still builds a scalar sheep, so let's fix that as well:

```
## in Animal
sub named {
  my $class = shift;
  my $name = shift;
  my $self = { Name => $name, Color => $class->default_color };
  bless $self, $class;
}
```

What's this `default_color`? Well, if `named` has only the name, we still need to set a color, so we'll have a class-specific initial color. For a sheep, we might define it as white:

```
## in Sheep
sub default_color { "white" }
```

And then to keep from having to define one for each additional class, we'll define a "backstop" method that serves as the "default default", directly in `Animal`:

```
## in Animal
sub default_color { "brown" }
```

Now, because `name` and `named` were the only methods that referenced the "structure" of the object, the rest of the methods can remain the same, so `speak` still works as before.

8.1.20 A horse of a different color

But having all our horses be brown would be boring. So let's add a method or two to get and set the color.

```
## in Animal
sub color {
  $_[0]->{Color}
}
sub set_color {
  $_[0]->{Color} = $_[1];
}
```

Note the alternate way of accessing the arguments: `$_[0]` is used in-place, rather than with a `shift`. (This saves us a bit of time for something that may be invoked frequently.) And now we can fix that color for Mr. Ed:

```
my $talking = Horse->named("Mr. Ed");
$talking->set_color("black-and-white");
print $talking->name, " is colored ", $talking->color, "\n";
```

which results in:

```
Mr. Ed is colored black-and-white
```