```
@Cow::ISA = qw(Animal);
```

Or allow it as an implicitly named package variable:

```
package Cow;
use vars qw(@ISA);
@ISA = qw(Animal);
```

If you're bringing in the class from outside, via an object-oriented module, you change:

```
package Cow;
use Animal;
use vars qw(@ISA);
@ISA = qw(Animal);
```

into just:

```
package Cow;
use base qw(Animal);
```

And that's pretty darn compact.

8.1.8 Overriding the methods

Let's add a mouse, which can barely be heard:

```
# Animal package from before
{ package Mouse;
  @ISA = qw(Animal);
  sub sound { "squeak" }
  sub speak {
    my $class = shift;
    print "a $class goes ", $class->sound, "!\n";
    print "[but you can barely hear it!]\n";
  }
}

Mouse->speak;
```

which results in:

```
a Mouse goes squeak!
[but you can barely hear it!]
```

Here, `Mouse` has its own speaking routine, so `Mouse->speak` doesn't immediately invoke `Animal->speak`. This is known as "overriding". In fact, we didn't even need to say that a `Mouse` was an `Animal` at all, since all of the methods needed for `speak` are completely defined with `Mouse`.

But we've now duplicated some of the code from `Animal->speak`, and this can once again be a maintenance headache. So, can we avoid that? Can we say somehow that a `Mouse` does everything any other `Animal` does, but add in the extra comment? Sure!

First, we can invoke the `Animal::speak` method directly: