The substrings (pieces of the string) can then be dropped into individual properties using a `list` object. For our needs, you can split the `$dog_properties` string as follows.

```
list($dog_weight, $dog_breed, $dog_color) = explode(',', $dog_properties);
```

> *For more information on the explode function, visit:*
>
> http://php.net/manual/en/function.explode.php

This will drop `no weight` into `$dog_weight`, `no breed` into `$dog_breed`, and `no color` into `$dog_color`. These three properties are also being created inside the `lab.php` program in this same line of code. I happen to give them the same names as their counterparts in the `Dog` class. However, remember if you had not created the `Dog` class, you would not know the original variable names. It would not matter, because you can call them anything you want and accomplish the same task.

Now that you have the variables containing the information, you can recreate the original `print` statement in the `lab.php` program instead of in the `dog.php` library.

```
print "Dog weight is $dog_weight. Dog breed is $dog_breed. Dog color is $dog_color.";
```

Notice that you did *not* include the `$this` pointer. You are not executing this statement within a class. You don't create instances of the `lab.php` program. There is only one instance of the program (because it is not a class and cannot have multiple instances). So the `$this` pointer is unnecessary.

The new `lab.php` program would now look like Example 3-7.

Example 3-7. The lab.php program with print statement

```
<?php
require_once("dog.php");
$lab = new Dog;
$dog_properties = $lab->get_properties();
list($dog_weight, $dog_breed, $dog_color) = explode(',', $dog_properties);
print "Dog weight is $dog_weight. Dog breed is $dog_breed. Dog color is $dog_color.";
?>
```

Assuming there are no errors in your program, the output will be the same as Figure 3-1, unchanged from the previous version of the program. However, the `Dog` class now meets one of the standards of the business rules tier by returning information to the program that calls it without attempting to format the output. The lab.php program now handles formatting the output.

Do It

1. Adjust the `speak` method in the `dog.php` file to return the `bark` string but not `print` it. Also adjust the call to the method in the `lab.php` file to display the output of the string. You can accept the string from the method and `print` the string in one line of code using syntax similar to the following:

   ```
   print $lab->speak();
   ```

2. Adjust the `$chow` object in the `lab.php` file to properly handle the return of the properties string and the `speak` string.

3. Adjust the `animal` class to return any strings instead of printing them. Adjust the program that makes an instance of the `animal` class to accept and display the strings that are returned.