For more information about the __construct method, visit:

Examples: http://php.net/manual/en/language.oop5.decon.php

Videos: https://www.thenewboston.com/videos.php?cat=11&video=17181

First, let's discuss the use of the special method called `__toString` (note the two underscores) in Example 3-12. Constructor methods are not allowed to return information (by default). The `return` statement cannot be used within the constructor. In order to return error messages created in the constructor to the calling program (`lab.php`), you must trick the program. The `__toString` method allows the programmer to decide what will occur if an attempt is made to use the `print` (or `echo`) method with the object name (`print $lab;`). Normally an error message would occur claiming the object cannot be converted to a string (`print` and `echo` can only display strings). This can be overridden by including a `__toString` method with a statement that returns a string. You can overcome this problem of being able to return the error messages by allowing the value in the `$error_message` property to be returned if the `print $lab;` statement is executed.

For more information on the __toString method and other magic methods visit http://php.net/manual/en/language.oop5.magic.php.

The `TRUE` and `FALSE` constants that are returned by the `set` methods also cause a problem because they are constants and not strings. If you attempt to convert these constants to a string using a method (such as `strval(TRUE);`), the values that they represent (1 for `TRUE`, 0 for `FALSE`) would become a string instead of '`TRUE`' or '`FALSE`'. Therefore, they cannot be returned via the `__toString` method. To overcome this problem we create the following code in the constructor to do a conversion from `TRUE` to '`TRUE`' or `FALSE` to '`FALSE`'.

```
$name_error = $this->set_dog_name($value1) == TRUE ? 'TRUE,' : 'FALSE,';
```

The order of operations will cause the `set_dog_name` method to execute before any of part of this code. The `set_dog_name` method returns `TRUE` or `FALSE` (constants). Assuming that the method returns a `TRUE` after the execution, the code line would now be

```
$name_error = TRUE == TRUE ? 'TRUE,' : 'FALSE,';
```

The order of operations then requires that the comparison (`TRUE == TRUE`) be evaluated. Of course, this evaluates to `TRUE`. The statements between the `?` and the `:` are used.

```
$name_error = 'TRUE,';
```

Thus `$name_error` is set to the string "`TRUE,`", which is now a string, not a constant.

Also note that a ', has been added in preparation for the next '`TRUE`' or '`FALSE`' value. Each value passed (except the last value) must be separated by a ', to allows the string to be separated later.

The other three similar lines are evaluated and also place a '`TRUE`,' or '`FALSE`,' in the error message properties (the weight error evaluation does not include a comma at the end of the string since it is the last one evaluated).

The last line of code in the constructor is evaluated.

```
$this->error_message = $name_error . $breed_error . $color_error . $weight_error;
```