contains the list of breeds). So you reset the app property (by calling `set_app`) in the container to "breeds". This tells the container program you are a `getBreeds` object, not a `dog` object. You then use the `get_dog_application` method of the container to find the location of the breeds XML file.

> *For more information on arrays, visit*
>
> *Examples: http://php.net/manual/en/function.array.php*
>
> *Videos: https://www.thenewboston.com/videos.php?cat=11&video=17024*

The PHP method `get_class_methods` is used to create an array of methods contained in `getBreeds`. Since the `get_select` method is the only method (besides the constructor), it is also the last method in the array. Its name is pulled from the array and then it is called using the property `$method_name ($result = $dog_data->$method_name($dog_app) )`. This allows the `getBreed` class to be completely independent of the `dog_interface`. The developer can change the name of the `get_select` method and everything would still work (as long as it's the last method in the class). This provides a complete split between the interface tier and the business rules tier.

The location of the XML file is passed into the `get_select` method of the `getBreeds` object (`$lab`), which uses the XML file for the data to create the select list box. The code for the select list box is dropped into `$result`. If the code did get dropped into `$result`, the code is displayed (`print $result`) back into the HTML form for the user to select a breed. If the file name was not valid, an error message (`print "System Error #3"`) will display instead of the select box.

The only other changes required are two slight changes to the `lab.html` and `get_breeds.js` files.

```
function AjaxRequest($value)
```

In the `get_breed.js` file, the function header for the `AjaxRequest` method has been changed to pass the actual file being called (this was not a requirement of this design, but it allows this file to be used for any program that is called via AJAX).

```
xmlHttp.open("GET", $value, true);
```

In addition, the `open` statement has been adjusted to use `$value` instead of a file name.

In the `lab.html` program, there are a couple of additional changes.

```
AjaxRequest('dog_interface.php');
```

The call to the JavaScript function now passes the file name, which has also been changed to the `dog_interface.php` file.

```
<form method="post" action="dog_interface.php" onSubmit="return validate_input(this)">
```

Finally, the action tag (in both form tag locations) for the HTML form has been changed to call the `dog_interface.php` program.

Notice that now, whenever you want to use the `Dog` application you call the interface first. Then the interface determines what you want to accomplish (get the breeds select box or process the properties for the dog you are creating).

You must communicate to all of your classes through the interface. The interface, in turn must create any required objects by using the container (except, of course, for the container itself). This follows the concepts of tier design.