```
public function get_dog_application()
{
$xmlDoc = new DOMDocument();
if ( file_exists("dog_applications.xml") )
{
$xmlDoc->load( 'dog_applications.xml' );
$searchNode = $xmlDoc->getElementsByTagName( "type" );
```

The first part of the `get_dog_application` method should look familiar. This code opens the `dog_applications.xml` file (after making sure it exists). Then it loads the contents into `$xmlDoc`. The last line calls the PHP method `getElementsByTagName`. This method searches for all occurrences of "type" in `$xmlDoc` and places each occurrence in `$searchNode`.

```
foreach( $searchNode as $searchNode )
{
    $valueID = $searchNode->getAttribute('ID');
    if($valueID == $this->app)
    {
    $xmlLocation = $searchNode->getElementsByTagName( "location" );
    return $xmlLocation->item(0)->nodeValue;
    break;
    }
}
```

The `foreach` loop looks at each line contained in `$searchNode`. It uses the PHP method `getAttribute` to place the next line with an `ID` XML attribute into the property `$valueID`. Once it is placed in `$valueID`, the value in this property (`dog`, `selectbox`, or `breeds`) is compared to `$this->app`. (The property app was loaded with a value in the constructor.) If the `ID` is found in the XML file, `getElementsByTagName` will search for the next line that contains a `location` XML tag. The return line then takes the value in the location tag (the file name and its location) and returns it to the program that called this method. The `break` statement is used to break out of the loop early because you have already returned the value needed in the previous line of code. The `else` part of the method (shown in Example 4-11) will return `FALSE` if the XML file does not exist.

```
function create_object($properties_array)
```

The `create_object` method is used to create the `dog` and `get_breeds` objects. The `dog` object constructor was expecting four values (`dog_name, dog_weight, dog_breed,` and `dog_color`) in its constructor. The `get_breeds` app was not accepting any values in its constructor. In order to provide more efficient code in `dog_container`, you change the signature of the constructor (the top line of the function) of each class to accept an array, instead of individual values. This will allow you to pass any number of items into either constructor.

```
function __construct($properties_array)
{ // dog class constructor
if (method_exists('dog_container', 'create_object')) {
$name_error = $this->set_dog_name($properties_array[0]) == TRUE ? 'TRUE,' : 'FALSE,';
$breed_error = $this->set_dog_breed($properties_array[1]) == TRUE ? 'TRUE,' : 'FALSE,';
$color_error = $this->set_dog_color($properties_array[2]) == TRUE ? 'TRUE,' : 'FALSE,';
$weight_error= $this->set_dog_weight($properties_array[3]) == TRUE ? 'TRUE' : 'FALSE';
```