Example 5-4. The getbreeds.class with exception handling

```
<?php
class GetBreeds {
   function __construct($properties_array) {
        if (!(method_exists('dog_container', 'create_object')))
           { exit; }
   }
     private $result = "??";
  public function get_select($dog_app)
  {
            if (($dog_app != FALSE) && ( file_exists($dog_app)))
            {
                $breed_file = simplexml:load_file($dog_app);
                $xmlText = $breed_file->asXML();
                $this->result = "<select name='dog_breed' id='dog_breed'>";
                 $this->result = $this->result . "<option value='-1' selected>Select a dog
                 breed</option>";
                 foreach ($breed_file->children() as $name => $value)
                 {
                      $this->result = $this->result . "<option value='$value'>$value</option>";
                 }
                 $this->result = $this->result . "</select>";
                  return $this->result;
            }
            else
            {
                   throw new Exception("Breed xml file missing or corrupt");
            }
    }
}
?>
```

Comparing the previous `GetBreeds` class (in Example 4-11) with Example 5-4 shows only one change. It returns `'FALSE'` and throws a general exception indicating that the `breed.xml` file is missing or corrupt. Again, the `GetBreeds` class pushes any exceptions to the interface. The interface no longer has to determine if there are any exceptions. Even though missing file errors cannot be redirected to be handled as exceptions, the code uses `file_exists` to throw an exception if the file is missing.

Example 5-5. The dog_container.php file with exception handling

```
<?php
class dog_container
{
  private $app;
  private $dog_location;
  function __construct($value) {
      if (function_exists('clean_input')) {
          $this->app = $value;
       } else { exit; }
   }
```