A data class should provide complete functionality for manipulating information. This includes the ability to read, insert, update, and delete information. Even if the current application does not require all these commands, logically, they should exist in the data class for future use.

A balance should be achieved between performance and the requirement to store information. While highly important information might require immediate storage, other information can be held in a data structure (list, array, and dataset) in the application until the user has completed any updates. Holding and making changes to information in the memory of the server, instead of the storage location, is much more efficient. Storing the information only after all changes have been completed will reduce several calls to the storage location down to two (initial retrieval of the information and saving of the updated information). Making changes to information in memory is always more efficient than making changes on a storage device (such as a hard drive).

Using a data class provides a logical ability to populate a data structure and to save information in a storage location automatically. Assuming that an instance of the data class will only be created when it is necessary to update information, the constructor of the class can be used to retrieve the information from storage and place it in the memory of the server. When the data object is no longer needed, logically, no more changes are required to the information. The destructor of the class can be used to return the information from memory to storage.

```
class dog_data
{
        function __construct()
        {
                $xmlfile = file_get_contents(get_dog_application("datastorage"));
                $xmlstring = simplexml:load_string($xmlfile);
                $array = (array)$xmlstring;
                print_r($array);
        }
}
```

This example constructor comes very close to providing useful information from an XML file. The PHP `file_get_contents` method opens a text file, drops the contents into a string, and closes the file. The constructor calls this method along with the `get_dog_application` method (same method that was used in `dog_container` in Example 5-5) to determine the file name and location of the XML data file. The contents of the file are then placed in `$xmlfile`. The PHP `simplexml:load_string` method then formats the data to allow the SimpleXML data model to traverse the information. At this point, the SimpleXML methods could be used to display and manipulate the data. However, the next line attempts to convert the XML data into an array. The (`array`) statement tries to use type casting. The `print_r` statement displays the results.

```
<?xml version="1.0" encoding="UTF-8"?>
<dogs>
<dog>
<dog_name>Woff</dog_name>
<dog_weight>12</dog_weight>
<dog_color>Yellow</dog_color>
<dog_breed>Lab</dog_breed>
</dog>
<dog>
<dog_name>Sam</dog_name>
<dog_weight>10</dog_weight>
<dog_color>Brown</dog_color>
<dog_breed>Lab</dog_breed>
</dog>
</dogs>
```