```
$name_error = $this->set_dog_name($properties_array[0]) == 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';

$breed_error = $this->set_dog_breed($properties_array[1]) == TRUE ? 'TRUE,' : 'FALSE,';

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

$this->save_dog_data();

if(stristr($this->error_message, 'FALSE'))
{
        throw new setException($this->error_message);
}
} else
{
        exit;
}
```

The constructor of the dog class sets all the properties and throws an exception if there are problems. If no problems exist, the information is saved (via `save_dog_data`), and the program closes (`exit`).

In order to keep the data tier independent of the business rules tier, dependency injection will be used to discover the location and name of the `dog_data` class and to call the `processRecords` method from the class. You will borrow the logic from Chapter 4. Actually, you can use the `dog_container` from Example 4-10 without any changes. If you don't remember the details of this class, revisit Chapter 4.

The `dog_container` class includes the `get_dog_application` method, which uses the logic discussed several times to search the dog application XML file for the name of the file needed (`dog_data.php`). The `set_app` method allows you to pass the application type (`dogdata`) to search in `get_dog_application`. It also includes the `create_object` class that will determine the class name (`dog_data`), make an instance of the class, and pass the class (the address of the class in memory) back to the calling program. The class does require that a `clean_input` function exist in the calling program. You don't currently have one in the `Dog` class. However, you can create a shell (an empty function) in the class to meet this requirement.

To use the container, you can use the logic that was in the `dog_interface` program to make an instance of the container, find the location of `dog_data`, and make an instance of `dog_data` (without knowing the class name).

```
function clean_input() { }
private function save_dog_data()
{
if ( file_exists("e5dog_container.php")) {
        require_once("e5dog_container.php"); // use chapter 5 container w exception handling
} else {
        throw new Exception("Dog container file missing or corrupt");
}
        $container = new dog_container("dogdata"); // sets the tag name to look for in XML file
        $properties_array = array("dogdata"); // not used but must be passed into create_object
        $dog_data = $container->create_object($properties_array); // creates dog_data object
        $method_array = get_class_methods($dog_data);
        $last_position = count($method_array) - 1;
```