this.refuel = function() {
    return "Refueling Vehicle with regular 87 octane gasoline";
}

this.mainTasks = function() {
    return "Driving to work, school, and the grocery store";
}

Note how the wheelCount and curbWeightInPounds properties are defined within the constructor using the var keyword, making the properties private. The properties are no longer public, and attempting to access the value of the wheelCount property via dot notation, like so:

```
var numberOfWheels = vehicle.wheelCount;
```

will return undefined instead of the actual value of wheelCount.

Since the properties are now private, you need to provide publicly available functions that can access these properties. The getWheelCount, setWheelCount, getCurbWeightInPounds, and setCurbWeightInPounds functions do just that. The Vehicle object now satisfies the concept of information hiding by allowing access to private properties only via publicly available functions.

Classical Inheritance in JavaScript

The prototype-based inheritance scheme of JavaScript works well enough, but it’s not a natural way of programming for those used to the class-based inheritance schemes in languages such as C++ and Java. For those who would rather eschew the prototype-based method of inheritance and use a more classically based approach, read on.

Bob Clary⁴ of Netscape proposed a method by which an object could inherit the properties and functions from another object using a single, generic script. The script simply copies the properties and functions of the “parent” object to the “child” object. For this purpose, we’ll show how to modify the script slightly so that only the properties and functions that don’t exist on the child object are copied to the child object; doing so allows functions on the child object to override functions on the parent. The generic function for creating an inheritance relationship between two objects looks like this:

```
function createInheritance(parent, child) {
    var property;
    for(property in parent) {
        if(!child[property]) {
            child[property] = parent[property];
        }
    }
}
```

The createInheritance function takes two arguments, the parent object and the child. The function simply iterates through all the members of the parent object (a member being

---