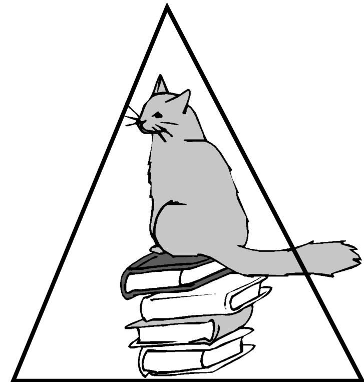
chapter

8

Objects



8.1 What Are Objects?

Objects are things we deal with every day. JavaScript deals with objects, as do most programming languages, and these languages are called object-oriented programming (OOP). Some people are apprehensive at the thought of tackling this kind of programming, and are perfectly happy to stick with top-down, procedural programs. But just as the everyday objects we use are not necessarily switchblades and chain saws, neither are programming objects. They are just a way of representing data. JavaScript is based on objects, so it's time to jump in.

When talking about JavaScript data types in Chapter 3, we discussed two types: primitive and composite. Objects are composite types. They provide a way to organize a collection of data into a single unit. Object-oriented languages, such as C++ and Java, bundle up data into a variable and call it an object. So does JavaScript.

When you learn about objects, they are usually compared to real-world things, like a cat, a book, or a triangle. Using the English language to describe an object, the object itself would be like a noun.

Nouns are described with adjectives. For the cat, it might be described as fat, furry, smart, or lazy. The book is old, with 400 pages, and contains poems. The triangle has three sides, three angles, and red lines. The adjectives that collectively describe these objects are called properties or attributes. The object is made up of a collection of these properties, or attributes.

In English, verbs are used to describe what the object can do or what can be done to it. The cat eats, sleeps, and meows. The book is read, its pages can be turned forward and backward, and it can be opened or closed by the reader. The triangle's sides and angles can be increased and decreased, it can be moved, and it can be colored. These verbs are called methods in object-oriented languages.

JavaScript supports several types of objects. They are as follows:

1. User-defined objects defined by the programmer
2. Core or built-in objects, such as Date, String, and Number (see Chapter 9)
3. Browser and Document objects (see Chapter 10)

8.1.1 Object Models and the Dot Syntax

An object model is a hierarchical tree-like structure used to describe all of the components of an object. When accessing an object in the tree, the object at the top of the tree is the root or parent of all parents. If there is an object below the parent it is called the child, and if the object is on the same level, it is a sibling. A child can also have children. A dot (.) is used to separate the objects when descending the tree; for example, a parent is separated from its child with a dot. In the following example, the *pet* object is subdivided into subordinate or child objects: a *cat* and a *dog*. The *cat* and the *dog* objects each have properties associated with them. In order to navigate down the tree to the cat's name, for example, you would stipulate *pet.cat.name,* and to get the dog's breed you would stipulate *pet.dog.breed.*

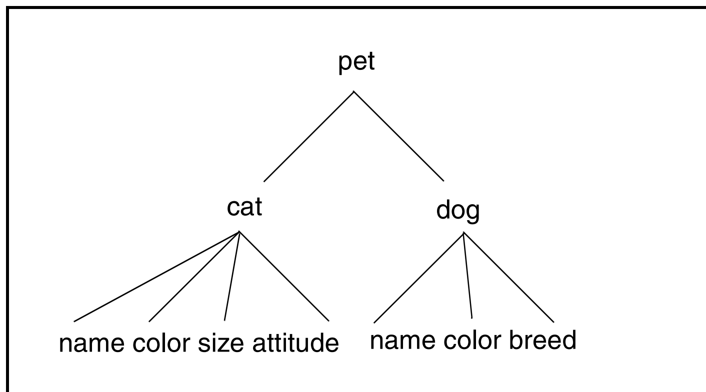


Figure 8.1 A hierarchical tree-like structure used to describe components of an object.

8.1.2 Creating an Object with a Constructor

JavaScript allows you to create an object in a number of ways, as discussed in detail in "User-Defined Objects" on page 131. One such way is with a constructor. A <u>constructor</u> is a special kind of method that creates an instance of an object. JavaScript comes with several built-in constructors. The *new* keyword precedes the name of the constructor that will be used to create the object.

```
var myNewObject = new Object(argument, argument, ...)
```

To create the *pet* object, for example, you could say:

```
var pet = new Object();
```

The *Object()* constructor, a special predefined constructor function, returns a reference to an object called *pet*, as shown in Example 8.1. The *pet* object has been instantiated and is ready to be assigned properties and methods.

EXAMPLE 8.1

```
    <html>
    <head><title>The Object() Constructor</title>
    <script language = "javascript">
1      var pet = new Object();
2      alert(pet);
    </script>
    </head>
    <body></body>
    </html>
```

EXPLANATION

1. The *Object()* constructor creates and returns a reference to a *pet* object. It is an empty object; i.e., it has no properties.
2. The returned value from the *Object()* constructor is a reference to an object, as shown in the Figure 8.2.

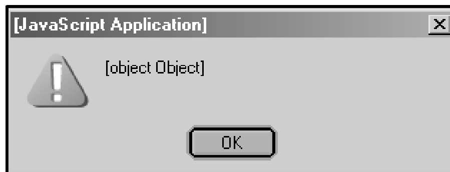


Figure 8.2 Output from Example 8.1.

The *pet* object could also be further subdivided as shown in Figure 8.1.

```
pet.cat = new Object();
pet.dog = new Object();
```

8.1.3 Properties of the Object

Properties describe the object and are connected to the object they describe with a dot. In Figure 8.1, the top object is the *pet* object. Although *cat* is an object in its own right, it is also considered a property of the *pet* object. In fact, any object subordinate to another object is also a property of that object. Both the *cat* and *dog* objects are properties of the *pet* object. The *cat* and the *dog* objects also have properties that describe them, such as *name*, *color*, *size*, and so forth.

To assign properties to the *cat* object, the syntax would be as follows:

```
pet.cat.name = "Sneaky";
pet.cat.color="yellow";
pet.cat.size="fat";
pet.cat.attitude = "stuck up";
```

EXAMPLE 8.2

```
    <html>
    <head><title>The Object() Constructor</title>
    <script language = "javascript">
        var pet = new Object();
1       pet.cat = new Object();
2       pet.cat.name = "Sneaky";
        pet.cat.color = "yellow";
        pet.cat.size = "fat";
        pet.cat.attitude = "stuck up";
    </script>
    </head>
    <body></body>
    </html>
```

EXPLANATION

1 New new object *cat* is created. It is subordinate to the *pet* object, so it is also a property of the *pet* object.

2 The *cat* object is assigned a *name* property with the value, *"Sneaky"*. It is also assigned color, size, and attitude properties.

In JavaScript you might see the syntax

```
window.document.bgColor = "lightblue";
```

The *window* is the top object in the hierarchy, the parent of all parents; the *document* is an object but, because it is subordinate to the *window*, it is also a property of the *window* object. Although the background color, *bgColor*, is a property of the document object, by itself it is not an object. (It is like an adjective because it describes the document.)

```
window
    document
        bgColor
```

8.1.4 Methods of the Object

Methods are special functions that object-oriented languages use to describe how the object behaves or acts. The cat purrs and the dog barks. Methods, like verbs, are action words that perform some operation on the object. For example, the *cat* object may have

a method called *sleep*() or *play*() and the dog object may have a method called *sit*() or *stay*(), and both of them could have a method called *eat*().

The dot syntax is used to call the methods just as it was used to separate objects from their properties. The method, unlike the property, is followed by a set of parentheses.

```
pet.cat.play();
```

Methods, like functions, can take arguments, or messages that will be sent to the object:

```
pet.dog.fetch("ball");
```

A JavaScript example:

```
window.close();
window.document.write("Hello\n");
```

8.2 User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called *Object*.

8.2.1 The *new* Operator

The *new* operator is used to create an instance of an object. To create an object, the *new* operator is followed by the constructor method. In the following example, the constructor methods are *Object*(), *Array*(), and *Date*(). These constructors are built-in JavaScript functions. A reference to the object is returned and assigned to a variable.

```
var car = new Object();
var friends = new Array("Tom", "Dick", "Harry");
var now= new Date("July 4, 2003");
```

8.2.2 The *Object()* Constructor

A constructor is a function (or method) that creates (constructs) and initializes an object. JavaScript provides a special constructor function called *Object*() to build the object. The return value of the *Object*() constructor is assigned to a variable. The variable contains a reference to the new object. The properties assigned to the object are <u>not</u> variables and are <u>not</u> defined with the *var* keyword. See Example 8.3.

FORMAT

```
var myobj = new Object();
```

EXAMPLE 8.3

```
    <html>
    <head><title>User-defined objects</title>
1       <script language = "javascript">
2          var toy = new Object();    // Create the object
3          toy.name = "Lego";       // Assign properties to the object
           toy.color = "red";
           toy.shape = "rectangle";
4       </script>
    </head>
    <body bgcolor="lightblue">
5       <script language = "javascript">
6          document.write("<b>The toy is  a " + toy.name + ".");
7          document.write("<br>It is a " + toy.color + " "
                          + toy.shape+ ".");
8       </script>
    </body>
    </html>
```

EXPLANATION

1. JavaScript code startes here.
2. The *Object()* constructor is called with the *new* keyword to create an instance of an object called *toy*. A reference to the new object is assigned to the variable, *toy*.
3. The *toy* object's *name* property is assigned *"Lego"*. The properties describe the characteristics or attributes of the object. Properties are not variables. Do not use the *var* keyword.
4. This is the end of the JavaScript program.
5. A new JavaScript program starts here in the body of the page.
6. The global object called *toy* is available within the script. The value of the *toy* object's *name* property is displayed.
7. The values for the *color* and *shape* properties of the *toy* object are displayed.
8. This is the end of the JavaScript program. The output is shown in Figure 8.3.

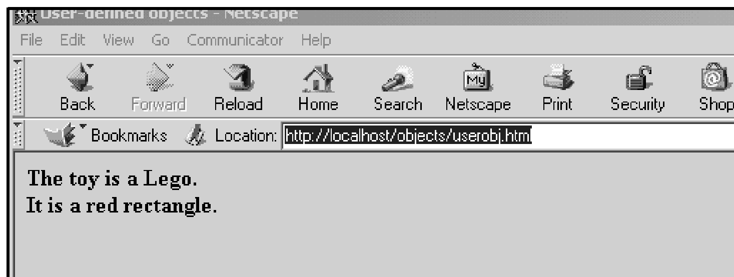


Figure 8.3 The *toy* object and its properties.

8.2.3 Creating the Object with a User-Defined Function

To create user-defined objects, you can create a function that specifies the object's name, properties, and methods. The function serves as a template or prototype of an object. When the function is called with the *new* keyword, it acts as a constructor and builds the new object, and then returns a reference to it.

The *this* keyword is used to refer to the object that has been passed to a function.

EXAMPLE 8.4

```
    <html>
    <head><title>User-defined objects</title></head>
    <script language = "javascript">

1       function book(title, author, publisher){
        // Defining properties
2           this.title = title;
3           this.author = author;
4           this.publisher = publisher;
5       }

    </script>
    <body bgcolor="lightblue"></body>
    <script language = "javascript">
6       var myBook = new book("JavaScript by Example",
                              "Ellie", "Prentice Hall");
7       document.writeln("<b>"  + myBook.title +
                           "<br>" + myBook.author +
                           "<br>" + myBook.publisher
            );
    </script>
    </body>
    </html>
```

EXPLANATION

1 This is a user-defined constructor function with three parameters.

2 The *this* keyword refers to the current object that is being created. The object is being assigned properties. The title of the book, *"JavaScript by Example"*, is being passed as the first parameter and assigned to the *title* property.

3 The author, *"Ellie"*, is assigned to the *author* property.

4 The publisher, *"Prentice Hall"*, is assigned to the *publisher* property.

5 This is the closing curly brace that terminates the function definition.

6 The variable, *myBook*, is assigned a reference to the newly created object.

7 The *title* property of the *myBook* object will be displayed. All of the properties of the *book* object are displayed in Figure 8.4.

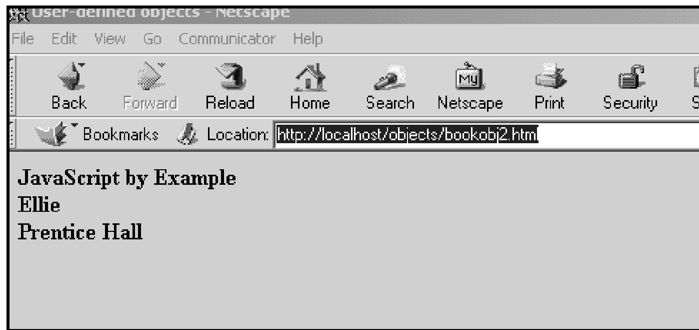


Figure 8.4 Output from Example 8.4.

8.2.4 Defining Methods for an Object

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it. The methods are functions that let the object do something or let something be done to it. There is little difference between a function (see Chapter 7, "Functions") and a method, except that a function is a standalone unit of statements and a method is attached to an object and can be referenced by the *this* keyword.

EXAMPLE 8.5

```
    <html>
    <head><title>Simple Methods</title>
    <script language = "javascript">
1       function distance(r, t){  // Define the object
2           this.rate = r;        // Assign properties
            this.time = t;
        }
3       function calc_distance(){  // Define a function that will
                                   // be used as a method
4           return this.rate * this.time;
        }
    </script>
    </head>
    <body bgcolor="lightblue">
    <script language="javascript">
5       var speed=eval(prompt("What was your speed
                              (miles per hour)? ",""));
        var elapsed=eval(prompt("How long did the trip take?
                                (hours)?" ,""));
```

EXAMPLE 8.5 (CONTINUED)

```
6       var howfar=new distance(speed, elapsed);
                                          // Call the constructor
7       howfar.distance=calc_distance;    // Create a new property
8       var d = howfar.distance();        // Invoke method
9       alert("The distance is " + d + " miles.");
    </script>
    </body>
    </html>
```

EXPLANATION

1 This is the constructor function. It creates and returns a reference to an object called *distance*. It takes two parameters, *r* and *t*.

2 The object (referenced by the *this* keyword) is assigned properties.

3 The function *calc_distance()* will be used later as a method for the object.

4 The function returns the results of this calculation to the variable, *d*, on line 8.

5 The user is prompted for input in this statement and the next. (See Figure 8.5.) The string he enters is evaluated by the *eval()* method and assigned as a number to the variables *speed* and *elapsed*.

6 A new object called *howfar* is created with the *new* constructor. Two arguments are passed, the rate (in miles per hour) and the time (in hours).

7 A new property for the *howfar* object is created. It is assigned the name of the function, *calc_distance*, that will be used as a method. Note: only the name of the function is assigned without the parentheses. Putting them there would result in an error.

8 The method called *distance()* is invoked for the *howfar* object. The returned value is assigned to variable, *d*.

9 The alert box displays the distance traveled. (See Figure 8.6.)

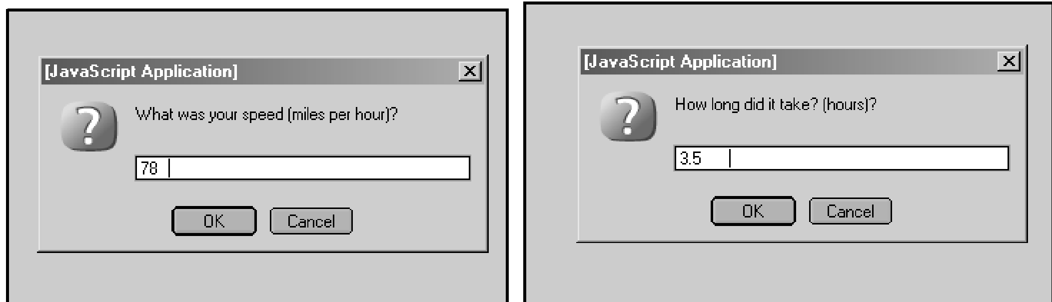


Figure 8.5 The user is prompted for input.

Figure 8.6 Final output displayed from Example 8.5.

A Method Defined in a Constructor. Methods can automatically be assigned to an object in the constructor function so that the method can be applied to multiple instances of an object.

EXAMPLE 8.6

```
    <html>
    <head><title>User-defined objects</title>
    <script language ="javascript">
1       function book(title, author, publisher){    // Receiving
                                                     // parameters
2           this.pagenumber=0;       // Properties
            this.title = title;
            this.author = author;
            this.publisher = publisher;
3           this.uppage = pageForward;    // Assign function name to
                                          // a property
4           this.backpage = pageBackward;
        }
5       function pageForward(){    // Functions to be used as methods
            this.pagenumber++;
            return this.pagenumber;
        }
6       function pageBackward(){
            this.pagenumber--;
            return this.pagenumber;
        }
    </script>
    </head>
    <body bgcolor="lightblue">
    <script language = "javascript">
```

EXAMPLE 8.6 (CONTINUED)

```
7       var myBook = new book("JavaScript by Example", "Ellie",
                             "Prentice Hall" );   // Create new object
8       myBook.pagenumber=5;
9       document.write( "<b>"+ myBook.title +
                        "<br>" + myBook.author +
                        "<br>" + myBook.publisher +
                        "<br>Current page is " + myBook.pagenumber );
        document.write("<br>Page forward: " );
10      for(i=0;i<3;i++){
11          document.write("<br>" + myBook.uppage());
            // Move forward a page
        }
        document.write("<br>Page backward: ");
        for(;i>0; i--){
12          document.write("<br>" + myBook.backpage());
            // Move back a page
        }
    </script>
    </body>
    </html>
```

EXPLANATION

1 This is the constructor function that is used to build the object by assigning it properties and methods. The parameter list contains the values for the properties *title, author*, and *publisher*.

2 The *this* keyword refers to the *book* object. The *book* object is given a *pagenumber* property initalized to *0*.

3 A method is defined by assigning the function to a property of the *book* object. *this.uppage* is assigned the name of the function, *pageForward*, that will serve as the object's method. Note that only the name of the method is assigned to a property. There are no parentheses following the name. This is important. If you put parentheses here, you will receive an error message. When the method is called you use parentheses.

4 The property *this.downpage* is assigned the name of the function, *pageBackward*, that will serve as the object's method.

5 The function *pageForward()* is defined. Its purpose is to increase the page number of the book by one, and return the new page number.

6 The function *pageBackward()* is defined. Its purpose is to decrease the page number by one and return the new page number.

7 A new object called *myBook* is created. The *new* operator invokes the *book()* function with three arguments: the title of the book, the author, and the publisher.

8 The *pagenumber* property is set to *5*.

9 The properties of the object are displayed in the browser window.

EXPLANATION (CONTINUED)

10 The *for* loop is entered. It will loop three times.

11 The *uppage()* method is called for the *myBook* object. It will increase the page number by 1 and display the new value, each time through the for loop.

12 The *backpage()* method is called for the *myBook* object. It will decrease the page number by 1 and display the new value, each time through the loop. The output is shown in Figure 8.7.

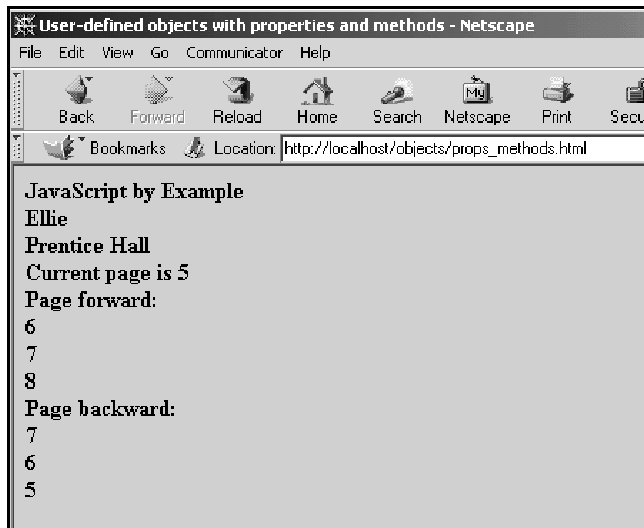


Figure 8.7 Calling user-defined methods. Output from Example 8.6.

Properties Can Be Objects. In "Properties of the Object" on page 129 we said that any object subordinate to another object is also a property of that object; thus, if a parent object has objects below it in the hierarchy, those child objects are properties of their parent and separated from their parent with a dot. So how would you create subordinate objects? You create a subordinate object just as you create any other object—with a constructor method. The one thing you must remember is that if the object being created is already a property of another object, you cannot use the *var* keyword preceding its name. For example, *var pet.cat = new Object()* will produce an error because *cat* is a property of the *pet* object and properties are never variables. (See Figure 8.1.) Weird, huh?

EXAMPLE 8.7

```
    <html>
    <head><title>Properties Can be Objects</title>
        <script language = "javascript">
1           var pet = new Object();  // pet is an object
2           pet.cat = new Object();  // cat is a property of the pet
                                     // object. cat is also an object
3           pet.cat.name="Sylvester";  // cat is assigned properties
            pet.cat.color="black";
4           pet.dog = new Object();
            pet.dog.breed = "Shepherd";
            pet.dog.name = "Lassie";
        </script>
    </head>
    <body bgcolor="lightblue">
        <script language = "javascript">
5           document.write("<b>The cat's name is " +
                            pet.cat.name + ".");
6           document.write("<br>The dog's name is " +
                           pet.dog.name + ".");
        </script>
    </body>
    </html>
```

Output:

```
5   The cat's name is Sylvester.
6   The dog's name is Lassie.
```

EXPLANATION

1 A new *pet* object is created with the *Object()* constructor.

2 The *Object()* constructor creates a *cat* object below the *pet* in the object hierarchy; that is, a *cat* object subordinate to the *pet* object and also a property of it. You cannot precede *pet.cat* with the keyword *var* because properties are never considered variables.

3 The new object also has a property called *name* which is assigned a value, *Sylvester.*

4 The *Object()* constructor creates an *dog* object below the *pet* in the object hierarchy; that is, a *dog* object subordinate to the *pet* object and also a property of it.

5 The *name* property for the *cat* object is displayed.

6 The *name* property for the *dog* object is displayed.

8.2.5 Object Literals

When an object is created by assigning it a comma-separated list of properties enclosed in curly braces, it is called an object literal. Each property consists of the property name followed by a colon and the property value. An object literal can be embedded directly in JavaScript code.

FORMAT

```
var object = { property1: value, property2: value };
```

Example:

```
var area = { length: 15, width: 5 };
```

EXAMPLE 8.8

```
    <html>
    <head><title>Object Literals</title>
    </head>
    <body bgcolor="yellow">
        <script language = "javascript">
1           var car = {
2           make: "Honda",
            year: 2002,
            price: "30,000",
            owner: "Henry Lee",
3           };
4           var details=car.make + "<br>";
            details += car.year + "<br>";
            details += car.price + "<br>";
            details += car.owner + "<br>";
            document.write(details);
        </script>
    </body>
    </html>
```

EXPLANATION

1 An object literal *car* is created and initialized.

2 The properties for the *car* object are assigned. Properties are separated from their corresponding values with a colon and each property/value pair is separated by a comma.

3 The object definition ends here.

4 The variable called *details* is assigned the properties of the *car* object for display. The output is shown in Figure 8.8.

Honda
2002
30,000
Henry Lee

Figure 8.8 Literal object properties. Output from Example 8.8.

8.3 Manipulating Objects

8.3.1 The *with* Keyword

The *with* keyword is used as a kind of shorthand for referencing an object's properties or methods.

The object specified as an argument to *with* becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object. (If a method is used, don't forget to include the parentheses after the method name.)

FORMAT

```
with (object){
    < properties used without the object name and dot>
}
```

Example:

```
with(employee){
    document.write(name, ssn, address);
}
```

EXAMPLE 8.9

```
    <html>
    <head><title>The with Keyword</title>
    <script language = "javascript">
1       function book(title, author, publisher){
2           this.title = title;   // Properties
            this.author = author;
            this.publisher = publisher;
3           this.show = display;  // Define a method
        }
```

EXAMPLE 8.9 (CONTINUED)

```
4       function display(anybook){
5          with(this){   // The with keyword
6             var info = "The title is " + title;
              info += "\nThe author is " +  author;
              info += "\nThe publisher is " + publisher;
7             alert(info);
           }
        }
     </script>
     </head>
     <body bgcolor="lightblue">
     <script language = "javascript">
8       var childbook = new book("A Child's Garden of Verses",
                                  "Robert Lewis Stevenson",
                                  "Little Brown");
9       var adultbook = new book("War and Peace",
                                  "Leo Tolstoy",
                                  "Penguin Books");
10      childbook.show(childbook);  // Call method for child's book
11      adultbook.show(adultbook);  // Call method for adult's book
     </script>
     </body>
     </html>
```

EXPLANATION

1. The *book* constructor function is defined with its properties.
2. The *book* object is described with three properties: *title*, *author*, and *publisher.*
3. The *book* object's property is assigned the name of a function. This property will serve as a method for the object.
4. A function called *display* is defined.
5. The *with* keyword will allow you to reference the properties of the object without using the name of the object or the *this* keyword. (See "The Math Object" on page 172 in Chapter 9.)
6. A variable called *info* is assigned the property values of a *book* object. The *with* keyword allows you to specify the property name without a reference to the object (and dot) preceding it.
7. The *alert* box displays the properties for a book object.
8. The constructor function is called and returns an instance of a new book object called *childbook.*
9. The constructor function is called and returns an instance of another book object called *adultbook.*
10. The *show()* method is called passing a reference to the *childbook* object.
11. The *show()* method is called passing a reference to the *adultbook* object.

Figure 8.9 The *childbook* object and its properties.

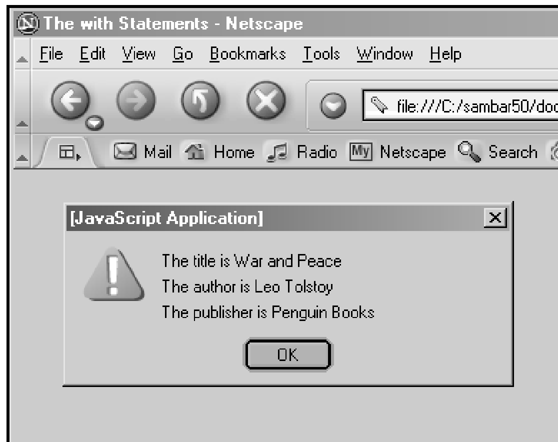


Figure 8.10 The *adultbook* object and its properties.

8.3.2 The *for/in* Loop

JavaScript provides the *for/in* loop, which can be used to iterate through a list of object properties or array elements. The *for/in* loop reads: for each property in an object (or for each element in an array) get the name of each property (element), in turn, and for each of the properties (elements), execute the statements in the block that follows.

The *for/in* loop is a convenient mechanism for looping through the properties of an object.

FORMAT

```
for(var property_name in object){
    statements;
}
```

EXAMPLE 8.10

```
    <html>
    <head><title>User-defined objects</title>
        <script language = "javascript">
1           function book(title, author, publisher){
2               this.title = title;
                this.author = author;
                this.publisher = publisher;
3               this.show=showProps;   // Define a method for the object
            }
4           function showProps(obj, name){
            // Function to show the object's properties
                var result = "";
5               for (var prop in obj){
6                   result += name + "." + prop + " = " +
                              obj[prop] + "<br>";
                }
7               return result;
            }
        </script>
    </head>
    <body bgcolor="lightblue">
        <script language="javascript">
8           myBook = new book("JavaScript by Example", "Ellie",
                              "Prentice Hall");
9           document.write("<br><b>" + myBook.show(myBook, "myBook"));
        </script>
    </body>
    </html>
```

EXPLANATION

1. The function called *book* will define the properties and methods for a *book* object. The function is a template for the new object. An instance of a new *book* object will be created when this constructor is called.
2. This is the first property defined for the *book* object. The *this* keyword refers to the current object.
3. A function name called *showProps* is assigned to a property of the object, thus creating a method for the object.
4. The function called *showProps* is defined, tasked to display all the properties of the object.

EXPLANATION (CONTINUED)

5 The special *for/in* loop executes a set of statements for each property of the object.

6 The name and value of each property is concatenated and assigned to a variable called *result*. *obj[prop]* is used to key into each of the property values of the *book* object.

7 The value of the variable *result* is sent back to the caller. Each time through the loop, another property and value are displayed.

8 A *new book* object called *myBook* is created (instantiated).

9 The properties for the book object are shown in the browser window; see Figure 8.11. Notice how the method and its definition are displayed.

```
myBook.title = JavaScript by Example
myBook.author = Ellie
myBook.publisher = Prentice Hall
myBook.show = function showProps(obj, name) { var result = ""; for (i in obj) { result += name + "." + i + " = " + obj[i] + "
"; } return result; }
```

Figure 8.11 The *book* object's properties.

8.3.3 Extending Objects with Prototypes

Object-oriented languages support a feature called inheritance, where one object can inherit the properties of another. JavaScript implements inheritance with prototypes. As of Netscape Navigator 3.0, it is possible to add properties to objects after they have been created by using the *prototype* object.

JavaScript functions are automatically given an empty *prototype* object. If the function serves as the constructor for an object, then the *prototype* object can be used to implement inheritance. When the properties are assigned to a given object by a constructor function, the *prototype* object gets the same properties. Each time a new object of the same class is created, that object also inherits the *prototype* object and all the same properties. The good news is that even after an object has been created, it can be extended with new properties that will also become part of the *prototype*. Then any objects created after that will automatically inherit the new properties.

What Is a Class? In object-oriented languages, the object's data describes the properties. The object, along with its properties and methods, is bundled up into a container called a class, and one class can inherit from another, and so on. Even though JavaScript doesn't have a class mechanism per se, it mimics the class concept with the constructor and its *prototype* object.

Each JavaScript class has a *prototype* object and one set of properties. Any objects created in the class will inherit the *prototype* properties. Let's say we define a constructor function called *Employee()* with a set of properties. The *prototype* object has all the same properties. The *Employee()* constructor function represents a class. The constructor is

called and instantiates an object called *janitor*, and then the constructor is called again and instantiates another object called *manager*, and so on. Each instance of the *Employee()* class automatically inherits all the properties defined for the *Employee* through its *prototype*.

After an object has been created, new properties can be added with the *prototype* property. This is how JavaScript implements inheritance.

EXAMPLE 8.11

```
    <html>
    <head><title>User-defined objects and Inheritance</title>
    <script language = "javascript">
1       function Book(title, author, publisher){    // The Book class
            this.title = title;
            this.author = author;
            this.publisher = publisher;
            this.show=showProps;
        }
2       function showProps(obj,name){
            var result = "";
            for (var i in obj){
               result += name + "." + i + " = " + obj[i] + "<br>";
            }
            return result;
        }
    </script>
    </head>
    <body bgcolor="lightblue">
    <script language="javascript">
    // Add a new function
3       function lastEdition(){
            this.latest=prompt("Enter the latest edition for
                                  "+this.title,"");
            return (this.latest);
        }
    // Add a new property with prototype
4   Book.prototype.edition=lastEdition;
5   var myBook=new Book("JavaScript by Example", "Ellie",
                        "Prentice Hall");
    // Define a new method
    document.write("<br><b>" + myBook.show(myBook,"myBook")+"<br>");
6   document.write("The latest edition is "+ myBook.edition()+"<br>");
    </script>
    </body>
    </html>
```

EXPLANATION

1 The function called *Book* defines the properties and methods for a *Book* object. *Book* is a JavaScript class. Each object has a prototype whose properties it inherits. An instance of a new *Book* object will inherit all of these properties.

2 A function called *showProps* is defined. It uses the special *for* loop to iterate through all the properties of an object. It will be used to create a method for the *Book* object, called *show()*.

3 A function called *lastEdition()* is defined. It returns the latest edition of the book.

4 A new property is given to the *Book* object using the *prototype* property, followed by the property name, *edition*. This property is assigned the name of a function called *lastEdition*, thus creating a new method for the *Book* class.

5 A new *Book* object, called *myBook*, is created. It has inherited all of the original properties of the *Book* class, plus the new property defined by the *prototype* property, called *edition*.

6 The new method is called for the *myBook* object.

Extending a JavaScript Object. Since all objects have the *prototype* object, it is possible to extend the properties of a JavaScript built-in object, just as we did for a user-defined object. (See Chapter 9, “JavaScript Core Objects.”)

EXAMPLE 8.12

```
    <html><head><title>Prototypes</title>
    <script language = "javascript">
    // Customize String Functions
1       function uc(){
2           var str=this.big();
3           return( str.toUpperCase());
        }
4       function lc(){
5           var str=this.small();
6           return( str.toLowerCase());
        }
7       String.prototype.bigUpper=uc;
8       String.prototype.smallLower=lc;

9       var string="This Is a Test STRING.";

10      string=string.bigUpper();
        document.write(string+"<br>");
11      document.write(string.bigUpper()+"<br>");
12      document.write(string.smallLower()+"<br>");
    </script>
    </head>
    <body bgcolor="lightblue"></body>
    </html>
```

EXPLANATION

1 A function called *uc* is defined. It will manipulate a *String* object.
2 The *big()* method is an HTML method that will increase the font size (one size larger than the current font) for the *String* object.
3 The string will be returned with a larger font and all letters in uppercase.
4 A function called *lc* is defined. It will also manipulate the *String* object.
5 The *small()* method is an HTML method that will decrease the font size (one size smaller than the current font) for the *String* object.
6 The string will be returned with a smaller font and all letters in lowercase.
7 The function *uc* is assigned to the *String.prototype.bigUpper* property, creating a new method for the *String* object.
8 The function *lc* is assigned to the *String.prototype.smallLower* property, creating another new method for the *String* object.
9 This is the *String* object that will be manipulated by the new methods created by the *prototype* property.
10 When the *string.bigUpper()* method is called, the string is converted to uppercase with all letters in a bigger font.
11 The *string.bigUpper()* method is called again, creating a larger string all in capital letters.
12 When the *string.smallLower()* method is called, the string is converted to lowercase with all letters in a smaller font. See output in Figure 8.12.

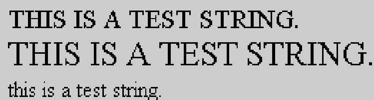


Figure 8.12 Extending properties to a built-in class. Output from Example 8.12.

EXERCISES

1. Create a *circle* object and a method that will calculate its circumference.

2. Write a function that will create a *clock* object.
   a. It will have three properties: *seconds*, *minutes*, and *hours*.
   b. Write two methods: *setTime()* to set the current time and *displayTime()* to display the time.
   c. The user will be prompted to select either a.m., p.m., or military time. The value he chooses will be passed as an argument to the *display()* method.
   d. The output will be either

   `14:10:26` or `2:10:26`

   depending on what argument was passed to the display() method.