

Tutorial 11

Exploring Object-Based Programming



Objectives

- Explore nested functions
- Use the base object
- Create a custom object
- Define object properties and methods
- Explore associative arrays
- Define an object class
- Define an object from a customized class



Objectives (cont'd)

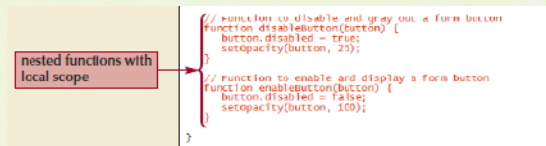
- Work with object prototypes
- Explore prototypal inheritance
- Work with the apply() and call() methods
- Explore the function object
- Test for valid function arguments
- Use the arguments variable

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Working with Nested Functions

- Use nested functions
 - To write code that controls the action and appearance of buttons
 - To limit a function to a local scope
 - Makes it invisible to any code outside of the containing function



*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Types of Executable Code Supported by JavaScript

Global code	<ul style="list-style-type: none">• Lies outside of a function• Automatically executed when encountered by the browser• Has global scope
Function code	<ul style="list-style-type: none">• Any code placed within a function• Must be called to be executed• Can be either local or global in scope, depending on whether the function is nested within another function
Eval code	<ul style="list-style-type: none">• Any code passed to the browser using the eval() function• Scope is limited within the eval() function itself• Syntax: eval(<i>string</i>)• Used when an application needs to create executable code during run time

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Introducing Custom Objects

- Three kinds of JavaScript objects:
 - **Native objects**
 - **Host objects**
 - **Custom objects (user-defined objects)**
- **Object constructor** or **constructor**: Function that defines every object in JavaScript
- **Object class**: The definition itself
- **Object instance** or **instance**: A specific case of an object class
- **Instantiating** an object: Creating an object from an object class

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Introducing Custom Objects

- All JavaScript objects are derived from a single fundamental **base object**
- Properties and methods of a base object:

Property or Method		Description
Property	<code>object.constructor</code>	Returns a reference to the constructor function of <i>object</i>
Method	<code>object.hasOwnProperty(prop)</code>	Returns a Boolean value indicating whether <i>object</i> supports the property <i>prop</i>
	<code>object.isPrototypeOf(object2)</code>	Returns a Boolean value indicating whether <i>object2</i> is an instance of <i>object</i>
	<code>object.propertyIsEnumerable(prop)</code>	Returns a Boolean value indicating whether the <i>prop</i> property of <i>object</i> is enumerable and can be used in a <i>for...in</i> loop
	<code>object.toString()</code>	Returns the type of <i>object</i> as the text string [<i>object Class</i>] where <i>Class</i> is the name of the object's constructor function
	<code>object.valueOf()</code>	Returns the value of <i>object</i> either as a text string, number, Boolean value, undefined, or null

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Introducing Custom Objects

- Defining an object property
 - To apply a property to an instance of a custom object:
object.property = value;
- Defining an object method
 - To create a custom method, associate the method with a function:
object.method = function

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Understanding Objects and Associative Arrays

- To access an object property or method:
 - Use the *object.property* and *object.method()* syntax,
- or -
 - Treat any object name as an array and the name of a property or method as a value within that array

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Understanding Objects and Associative Arrays

- **Associative array**
 - Contains a collection of keys, each associated with a value or set of values (vs. **index arrays**, in which array values are identified by their index number)
 - Provides compact way to define an object using an **object literal**
- **Encapsulation**
 - Ensures that functions defined for a custom object will not conflict with functions defined elsewhere in the application because scope of the function is local to object constructor

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Understanding Objects and Associative Arrays

- Associative arrays contain items that are not indexed; you cannot loop through contents of an array using a counter variable; instead, use the for ... in structure:

```
for (key in array) {  
  commands  
}
```

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Understanding Objects and Associative Arrays

- To define a custom object drawn from the base object:

```
var newObject = new Object() {  
  this.property = value;  
  this.method = function;  
  ...  
}
```

- To define a custom object as an object literal:

```
var newObject = {  
  property : value,  
  method : function,  
  ...  
}
```

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Creating an Object Class

- To define a class of objects, enter constructor function:

```
function object() {  
  this.prop1 = value1;  
  this.prop2 = value2;  
  ...  
  this.method1 = function1;  
  this.method2 = function2;  
  ...  
}
```
- To instantiate an object from an object class:

```
var newObject = new object();
```

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Working with Object Prototypes

- The **prototype** property stores an object that acts as a template for all new object instances created by the constructor
- To reference a prototype:

```
object.prototype
```
- To apply a property to an object prototype:

```
object.prototype.property = value;
```
- To apply a method to an object prototype:

```
object.prototype.method = function;
```

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Working with Object Prototypes

- The prototype property
 - Can be used with native JavaScript objects (Array, Date, and String)
 - Allows you to extend JavaScript objects by creating customized properties and methods for them

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Working with Object Prototypes

Public method	<ul style="list-style-type: none">• Publically available• Can be made at any time using object's prototype
Private method	<ul style="list-style-type: none">• Accessible only within object itself and not outside of that object• Can be made only within constructor function itself
Privileged method	<ul style="list-style-type: none">• Able to access private variables and methods, but is itself accessible to the public• Relies on the value returned by calling the private <code>getFilename()</code> function• Can be made only within constructor function itself

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Working with Object Prototypes

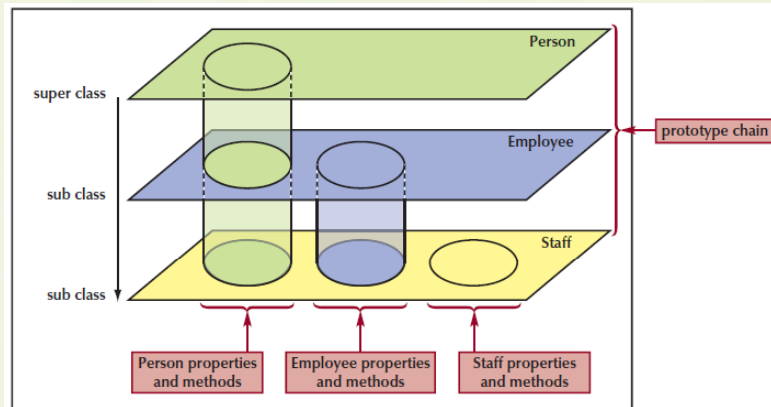
- Creating **prototypal inheritance**
 - All native JavaScript objects are derived from a single base object; any object can act as a base for new object classes through the use of prototypes
 - To create a **prototype chain**:
 - Specify each object prototype as an instance of the object above it in the class hierarchy
 - Define the relationship between the object classes (order is important; start at the top of the hierarchy and move down to the lower sub classes)

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Working with Object Prototypes

- Creating prototypal inheritance



*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Adding a Property to a Prototype

- A custom property can store a document element using the expression:

```
object.property = document.createElement(elem)
```

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



The Changing Context of the `this` Keyword

- Common source of error when working with custom methods and nested functions is failure to keep track of the changing context of the `this` keyword
- The `this` keyword always refers to the current object, usually the object that initiated the function or method
- Limit use of the `this` keyword for non-nested functions or for situations where its context is completely clear

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Developing More Custom Properties and Methods

- Applying and calling a function
 - To apply a function or method to an object, use the `apply()` method:
`function.apply(thisObj, argArray)`
 - To call a function or method for use with an object, run:
`function.call(thisObj, arg1, arg2, arg3, ...)`

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Exploring the Function Object

- Supports its own collection of properties and methods
- Use properties of the Function object to return information about constructors

Properties and Methods		Description
Property	<code>function.name</code>	Returns the name of the function, <i>function</i> (not currently supported by Internet Explorer or Opera)
	<code>function.caller</code>	Returns the function that called <i>function</i> (not currently supported by Opera)
	<code>function.length</code>	Returns the number of arguments expected by <i>function</i>
Method	<code>function.apply(thisObj, thisArray)</code>	Applies <i>function</i> to <i>thisObj</i> using argument values stored in the array, <i>thisArray</i>
	<code>function.call(thisObj, arg1, arg2, ...)</code>	Applies <i>function</i> to <i>thisObj</i> using arguments in the list <i>arg1, arg2, ...</i>
	<code>function.toString()</code>	Returns the code of <i>function</i> as a text string

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Exploring the Function Object

- Function object also includes an arguments variable that:
 - Returns detailed information about the parameter values passed to the function
 - Is similar to an array, although its contents cannot be modified or added to
 - Is only accessible within the function body, not outside of it
 - Properties of the arguments variable

Property	Description
arguments.length	Returns number of arguments passed into the function
arguments.callee	Returns a reference to the current function
arguments.caller	Returns a reference to the function that called the current function

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Exploring the Function Object

- Testing for errors
 - Use the arguments variable to verify that the correct number of parameter values have been passed to a function
 - If the number of values does not match the expected length, user can be alerted
 - Use the arguments variable to test the data type of each value passed to a function against a required data type
 - Data types can be tested in two ways: typeof() method and constructor property

*New Perspectives on
JavaScript and AJAX, 2nd Edition*



Exploring the Function Object

- To return the data type of a variable as a text string:
`typeof(variable)`
- To return the constructor of a variable:
`variable.constructor`