COURSE TECHNOLOGY
CENGAGE Learning

JavaScript, Sixth Edition

Chapter 7
Using Object-Oriented JavaScript

Objectives

When you complete this chapter, you will be able to:

- Explain basic concepts related to object-oriented programming
- Use the `Date`, `Number`, and `Math` objects
- Define your own custom JavaScript objects

Introduction to Object-Oriented Programming

- Object-oriented programming
  - Allows reuse of code without having to copy or recreate it

Reusing Software Objects

- Object-oriented programming (OOP)
  - Creating reusable software objects
    - Easily incorporated into multiple programs
- Object
  - Programming code and data treated as an individual unit or component
  - Also called a component
- Data
  - Information contained within variables or other types of storage structures

Reusing Software Objects (cont'd.)

- Objects range from simple controls to entire programs
- Popular object-oriented programming languages
  - C++, Java, Visual Basic

Reusing Software Objects (cont'd.)

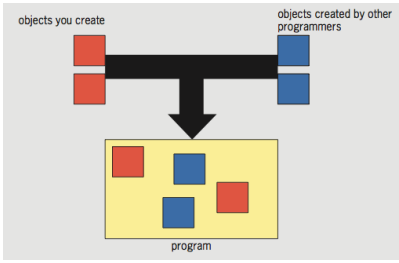

Figure 7-1 Programming with objects

What Is Encapsulation?

- Encapsulated objects
  - Code and data contained within the object itself
- Encapsulation places code inside a "black box"
- Interface
  - Elements required for program to communicate with an object
- Principle of information hiding
  - Any methods and properties other programmers do not need to access should be hidden

What Is Encapsulation? (cont'd.)

- Advantages of encapsulation
  - Reduces code complexity
  - Prevents accidental bugs and stealing of code
- Programming object and its interface
  - Compare to a handheld calculator

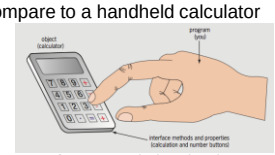

Figure 7-2 Calculator interface

What Is Encapsulation? (cont'd.)

- `Document` object is encapsulated (black box)
  - `getElementById()` method
    - Part of the interface JavaScript uses to communicate with the `Document` object
- Microsoft Word: example of an object and its interface

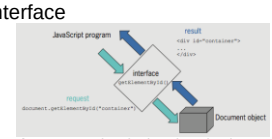

Figure 7-3 Using the interface for the `Document` object

Understanding Classes

- Classes
  - Grouping of code, methods, attributes, etc., making up an object
- Instance
  - Object created from an existing class
- Instantiate: create an object from an existing class
- Instance of an object inherits its methods and properties from a class
- Objects in the browser object model
  - Part of the web browser
  - No need to instantiate them to use them

Using Built-In JavaScript Classes

CLASS	DESCRIPTION
Arguments	Retrieves and manipulates arguments within a function
Array	Creates new array objects
Boolean	Creates new Boolean objects
Date	Retrieves and manipulates dates and times
Error	Returns run-time error information
Function	Creates new function objects
Global	Stores global variables and contains various built-in JavaScript functions
JSON	Manipulates objects formatted in JavaScript Object Notation (JSON); available in ECMAScript 5 and later
Math	Contains methods and properties for performing mathematical calculations
Number	Contains methods and properties for manipulating numbers
Object	Represents the base class for all built-in JavaScript classes; contains several of the built-in JavaScript functions
RegExp	Contains methods and properties for finding and replacing characters in text strings
String	Contains methods and properties for manipulating text strings

Table 7-1 Built-in JavaScript classes

Using Built-In JavaScript Classes (cont'd.)

- Instantiating an object
  - Some of the built-in JavaScript objects used directly in code
  - Some objects require programmer to instantiate a new object
  - Example: `Math` object's `PI (π)` property in a script

```
// calculate the area of a circle based on its radius
function calcCircleArea() {
  var r = document.getElementById("radius").value;
  var area = Math.PI * Math.pow(r, 2); // area is pi times ↵
  radius squared
  return area;
}
```

Using Built-In JavaScript Classes (cont'd.)

- Instantiating an object (cont'd.)
  - Can instantiate `Array` object using array literal
    - Example: `var deptHeads = [];`
  - Can instantiate empty generic object using object literal
    - Example: `var accountsPayable = {};`
    - Generic object literal uses curly braces around value
  - Can't use object literal for `Date` object
    - Must use constructor
    - Example: `var today = new Date();`

Using Built-In JavaScript Classes (cont'd.)

- Performing garbage collection
  - Garbage collection
    - Cleaning up, or reclaiming, memory reserved by a program
  - Declaring a variable or instantiating a new object
    - Reserves memory for the variable or object
  - JavaScript knows when a program no longer needs a variable or object
    - Automatically cleans up the memory

Using the `Date`, `Number`, and `Math` Classes

- Three of most commonly used JavaScript classes:
  - `Date`, `Number`, and `Math`

Manipulating the Date and Time with the `Date` Class

- `Date` class
  - Methods and properties for manipulating the date and time
  - Allows use of a specific date or time element in JavaScript programs

CONSTRUCTOR	DESCRIPTION
`Date()`	Creates a `Date` object that contains the current date and time provided by the device
`Date(milliseconds)`	Creates a `Date` object based on the number of milliseconds that have elapsed since midnight, January 1, 1970
`Date(date_string)`	Creates a `Date` object based on a string containing a date value
`Date(year, month[, day, hours, minutes, seconds, milliseconds])`	Creates a `Date` object with the date and time set according to the passed arguments; the `year` and `month` arguments are required

Table 7-2 `Date` class constructors

Manipulating the Date and Time with the `Date` Class (cont'd.)

- Example:
  - `var today = new Date();`
  - Month and year date representation in a `Date` object
  - Stored using numbers matching actual date and year
- Days of the week and months of the year
  - Stored using numeric representations
    - Starting with zero: similar to an array
- Example:
  - `var independenceDay = new Date(1776, 6, 4);`

Manipulating the Date and Time with the `Date` Class (cont'd.)

- After creating a new `Date` object
  - Manipulate date and time in the variable using the `Date` class methods
- Date and time in a `Date` object
  - Not updated over time like a clock
  - `Date` object contains the static (unchanging) date and time
    - Set at the moment the JavaScript code instantiates the object

Manipulating the Date and Time with the `Date` Class (cont'd.)

METHOD	DESCRIPTION
`getDate()`	Returns the date of a `Date` object
`getDay()`	Returns the day of a `Date` object
`getFullYear()`	Returns the year of a `Date` object in four-digit format
`getHours()`	Returns the hour of a `Date` object
`getMilliseconds()`	Returns the milliseconds of a `Date` object
`getMinutes()`	Returns the minutes of a `Date` object
`getMonth()`	Returns the month of a `Date` object
`getSeconds()`	Returns the seconds of a `Date` object
`getTime()`	Returns the time of a `Date` object
`now()`	Returns the current time as the number of milliseconds that have elapsed since midnight, January 1, 1970 (ECMAScript 5 and later only)

Table 7-3 Commonly used methods of the `Date` class (*continues*)

Manipulating the Date and Time with the `Date` Class (cont'd.)

METHOD	DESCRIPTION
`setDate(date)`	Sets the date (1–31) of a `Date` object
`setFullYear(year[, month, day])`	Sets the four-digit year of a `Date` object; optionally allows you to set the month and the day
`setHours(hours[, minutes, seconds, milliseconds])`	Sets the hours (0–23) of a `Date` object; optionally allows you to set the minutes (0–59), seconds (0–59), and milliseconds (0–999)
`setMilliseconds(milliseconds)`	Sets the milliseconds (0–999) of a `Date` object
`setMinutes(minutes[, seconds, milliseconds])`	Sets the minutes (0–59) of a `Date` object; optionally allows you to set seconds (0–59) and milliseconds (0–999)
`setMonth(month[, date])`	Sets the month (0–11) of a `Date` object; optionally allows you to set the date (1–31)
`setSeconds(seconds[, milliseconds])`	Sets the seconds (0–59) of a `Date` object; optionally allows you to set milliseconds (0–999)
`setTime()`	Sets the time as the number of milliseconds that have elapsed since midnight, January 1, 1970
`toLocaleString()`	Converts a `Date` object to a string, set to the current time zone
`toString()`	Converts a `Date` object to a string
`valueOf()`	Converts a `Date` object to a millisecond format

Table 7-3 Commonly used methods of the `Date` class

Manipulating the Date and Time with the `Date` Class (cont'd.)

- Each portion of a `Date` object can be retrieved and modified using the `Date` object methods
  - Examples:

```
var curDate = new Date();
curDate.getDate();
```

- Displaying the full text for days and months
  - Use a conditional statement to check the value returned by the `getDay()` or `getMonth()` method
  - Example:
    - `if/else` construct to print the full text for the day of the week returned by the `getDay()` method

Manipulating the Date and Time with the `Date` Class (cont'd.)

```
var today = new Date();
var curDay = today.getDay();
var weekday;
if (curDay === 0) {
   weekday = "Sunday";
} else if (curDay === 1) {
   weekday = "Monday";
} else if (curDay === 2) {
   weekday = "Tuesday";
} else if (curDay === 3) {
   weekday = "Wednesday";
} else if (curDay === 4) {
   weekday = "Thursday";
} else if (curDay === 5) {
   weekday = "Friday";
} else if (curDay === 6) {
   weekday = "Saturday";
}
```


Manipulating the Date and Time with the `Date` Class (cont'd.)

- Example: include an array named `months`
  - 12 elements assigned full text names of the months

```
var today = new Date();
var months = ["January","February","March",↵
        "April","May","June",↵
        "July","August","September",↵
        "October","November","December"];
var curMonth = months[today.getMonth()];
```


Manipulating Numbers with the `Number` Class

- `Number` class
  - Methods for manipulating numbers and properties containing static values
    - Representing some numeric limitations in the JavaScript language
  - Can append the name of any `Number` class method or property
    - To the name of an existing variable containing a numeric value

Manipulating Numbers with the `Number` Class (cont'd.)

- Using `Number` class methods

METHOD	DESCRIPTION
`toExponential(decimals)`	Converts a number to a string in exponential notation using the number of decimal places specified by *decimals*
`toFixed(decimals)`	Converts a number to a string using the number of decimal places specified by *decimals*
`toLocaleString()`	Converts a number to a string that is formatted with local numeric formatting style
`toPrecision(decimals)`	Converts a number to a string with the number of decimal places specified by *decimals*, in either exponential notation or in fixed notation
`toString(base)`	Converts a number to a string using the number system specified by *base*
`valueOf()`	Returns the numeric value of a `Number` object

Table 7-4 `Number` class methods

Manipulating Numbers with the `Number` Class (cont'd.)

- Using `Number` class methods (cont'd.)
  - Primary reason for using any of the "to" methods
    - To convert a number to a string value with a specific number of decimal places
  - `toFixed()` method
    - Most useful `Number` class method
  - `toLocaleString()` method
    - Converts a number to a string formatted with local numeric formatting conventions

Manipulating Numbers with the `Number` Class (cont'd.)

- Accessing `Number` class properties

PROPERTY	DESCRIPTION
`MAX_VALUE`	The largest positive number that can be used in JavaScript
`MIN_VALUE`	The smallest positive number that can be used in JavaScript
`NaN`	The value `NaN`, which stands for "not a number"
`NEGATIVE_INFINITY`	The value of negative infinity
`POSITIVE_INFINITY`	The value of positive infinity

Table 7-5 `Number` class properties

Performing Math Functions with the `Math` Class

- `Math` class
  - Methods and properties for mathematical calculations
- Cannot instantiate a Math object using a statement such as: `var mathCalc = new Math();`
  - Use the `Math` object and one of its methods or properties directly in the code
- Example:

```
var curNumber = 144;
var squareRoot = Math.sqrt(curNumber); // returns 12
```


Performing Math Functions with the `Math` Class (cont'd.)

METHOD	RETURNS
`abs(x)`	The absolute value of *x*
`acos(x)`	The arc cosine of *x*
`asin(x)`	The arc sine of *x*
`atan(x)`	The arc tangent of *x*
`atan2(x, y)`	The angle from the x-axis of the point represented by *x*, *y*
`ceil(x)`	The value of *x* rounded to the next highest integer
`cos(x)`	The cosine of *x*
`exp(x)`	The exponent of *x*
`floor(x)`	The value of *x* rounded to the next lowest integer
`log(x)`	The natural logarithm of *x*
`max(x, y)`	The larger of *x* or *y*
`min(x, y)`	The smaller of *x* or *y*
`pow(x, y)`	The value of *x* raised to the *y* power
`random()`	A random number
`round(x)`	The value of *x* rounded to the nearest integer
`sin(x)`	The sine of *x*
`sqrt(x)`	The square root of *x*
`tan(x)`	The tangent of *x*

Table 7-6 Math class methods

Performing Math Functions with the `Math` Class (cont'd.)

PROPERTY	DESCRIPTION
`E`	Euler's constant *e*, which is the base of a natural logarithm; this value is approximately 2.7182818284590452354
`LN10`	The natural logarithm of 10, which is approximately 2.302585092994046
`LN2`	The natural logarithm of 2, which is approximately 0.6931471805599453
`LOG10E`	The base-10 logarithm of *e*, the base of the natural logarithms; this value is approximately 0.4342944819032518
`LOG2E`	The base-2 logarithm of *e*, the base of the natural logarithms; this value is approximately 1.4426950408889634
`PI`	A constant representing the ratio of the circumference of a circle to its diameter, which is approximately 3.1415926535897932
`SQRT1_2`	The square root of 1/2, which is approximately 0.7071067811865476
`SQRT2`	The square root of 2, which is approximately 1.4142135623730951

Table 7-7 Math class properties

Performing Math Functions with the `Math` Class (cont'd.)

- Example:
  - Use the `PI` property to calculate the area of a circle based on its radius
    - Code uses the `pow()` method to raise the radius value to second power, and the `round()` method to round the value returned to the nearest whole number

```
var radius = 25;
var area = Math.PI * Math.pow(radius, 2);
var roundedArea = Math.round(area); // returns 1963
```


Defining Custom JavaScript Objects

- JavaScript: not a true object-oriented programming language
  - Cannot create classes in JavaScript
  - Instead, called an object-based language
- Can define custom objects
  - Not encapsulated
  - Useful to replicate the same functionality an unknown number of times in a script

Declaring Basic Custom Objects

- Use the `Object` object
  - `var objectName = new Object();`
  - `var objectName = {};`
- Can assign properties to the object
  - Append property name to the object name with a period

Declaring Basic Custom Objects (cont'd.)

- Add properties using dot syntax
  - Object name followed by dot followed by property name
  - Example:

```
InventoryList.inventoryDate = new Date(2017, 11, 31);
```


Declaring Basic Custom Objects (cont'd.)

- Can assign values to the properties of an object when object first instantiated
- Example:

```
var PerformanceTickets = {
  customerName: "Claudia Salomon",
  performanceName: "Swan Lake",
  ticketQuantity: 2,
  performanceDate: new Date(2017, 6, 18, 20)
};
```


Declaring Sub-Objects

- Value of a property can be another object
  - called a sub-object
  - Example–`order` object with `address` sub-object:

```
var order = {
  orderNumber: "F5987",
  address: {
    street: "1 Main St",
    city: "Farmington",
    state: "NY",
    zip: "14425"
  }
};
```

Referring to Object Properties as Associative Arrays

- Associative array
  - An array whose elements are referred to with an alphanumeric key instead of an index number
- Can also use associative array syntax to refer to the properties of an object
- With associative arrays
  - Can dynamically build property names at runtime

Referring to Object Properties as Associative Arrays (cont'd.)

- Can use associative array syntax to refer to the properties of an object
- Example:

```
var stopLightColors = {
  stop: "red",
  caution: "yellow",
  go: "green"
};
stopLightColors["caution"];
```


Referring to Object Properties as Associative Arrays (cont'd.)

- Can easily reference property names that contain numbers
  - Example:

```
var order = {
  item1: "KJ2435J",
  price1: 23.95,
  item2: "AW23454",
  price2: 44.99,
  item3: "2346J3B",
  price3: 9.95
};
```


Referring to Object Properties as Associative Arrays (cont'd.)

- Can easily reference property names that contain numbers (cont'd.)
  - To create order summary:

```
for (var i = 1; i < 4; i++) {
  document.getElementById("itemList").innerHTML +=↵
    "<p class='item'>" + order["item" + i] + "</p>";
  document.getElementById("itemList").innerHTML +=↵
    "<p class='price'>" + order["price" + i] + "</p>";
};
```


Referring to Object Properties as Associative Arrays (cont'd.)

- Can also write generic code to add new object properties that incorporate numbers
  - Example—adding items to shopping cart:

```
totalItems += 1; // increment counter of items in order
currentItem = document.getElementById("itemName").innerHTML;
currentPrice = document.getElementById("itemPrice").innerHTML;
newItemPropertyName = "item" + totalItems; // "item4"
newPricePropertyName = "price" + totalItems; // "price4"
order.newItemPropertyName = currentItem; // order.item4 = (name)
order.newPricePropertyName = currentPrice;
// order.price4 = (price);
```

- Allows for as many items as user wants to purchase

Creating Methods

- Object method simply a function with a name within the object
- Two ways to add method to object
  - Provide code for method in object
  - Reference external function

Creating Methods (cont'd.)

- Specify method name with anonymous function as value
  - Example:

```
var order = {
  items: {},
  generateInvoice: function() {
    // function statements
  }
};
```


Creating Methods (cont'd.)

- Specify method name with existing function as value
  - Example:

```
function processOrder() {
  // function statements
}
var order = {
  items: {},
  generateInvoice: processOrder
};
```

  - Reference to existing function cannot have parentheses

Enumerating custom object properties

- Custom objects can contain dozens of properties
- To execute the same statement or command block for all the properties within a custom object
  - Use the `for/in` statement
  - Looping statement similar to the `for` statement
- Syntax

```
for (variable in object) {
  statement(s);
}
```


Enumerating custom object properties (cont'd.)

- `for/in` statement enumerates, or assigns an index to, each property in an object
- Typical use:
  - validate properties within an object

Enumerating custom object properties (cont'd.)

- Example—checking for empty values:

```
var item={
  itemNumber: "KJ2435J",
  itemPrice: 23.95,
  itemInstock: true,
  itemShipDate: new Date(2017, 6, 18),
};
for (prop in order) {
  if (order[prop] === "") {
    order.generateErrorMessage();
  }
}
```


Deleting Properties

- Use the `delete` operator
- Syntax

```
delete object.property
```

- Example:

```
delete order.itemInStock;
```

Defining Constructor Functions

- Constructor function
  - Used as the basis for a custom object
  - Also known as object definition
- JavaScript objects
  - Inherit all the variables and statements of the constructor function on which they are based
- All JavaScript functions
  - Can serve as a constructor

Defining Constructor Functions (cont'd.)

- Example:
  - Define a function that can serve as a constructor function

```
function Order(number, order, payment, ship) {
  this.customerNumber = number;
  this.orderDate = order;
  this.paymentMethod = payment;
  this.shippingDate = ship;
}
```


Adding Methods to a Constructor Function

- Can create a function to use as an object method
  - Refer to object properties with `this` reference
  - Example:

```
function displayOrderInfo() {
  var summaryDiv = document.getElementById("summarySection");
  summaryDiv.innerHTML += ("<p>Customer: " +↵
    this.customerNumber + "</p>");
  summaryDiv.innerHTML += ("<p>Order Date: " +↵
    this.orderDate.toLocaleString()+ "</p>");
  summaryDiv.innerHTML += ("<p>Payment: " +↵
    this.paymentMethod + "</p>");
  summaryDiv.innerHTML += ("<p>Ship Date: " +↵
    this.shippingDate.toLocaleString() + "</p>");
}
```


Using the `prototype` Property

- After instantiating a new object
  - Can assign additional object properties
    - Use a period
- New property only available to that specific object
- `prototype` property
  - Built-in property that specifies the constructor from which an object was instantiated
  - When used with the name of the constructor function
    - Any new properties you create will also be available to the constructor function

Using the `prototype` Property (cont'd.)

- Object definitions can use the `prototype` property to extend other object definitions
  - Can create a new object based on an existing object

Summary

- Object-oriented programming (or OOP)
  - The creation of reusable software objects
- Reusable software objects
  - Called components
- Object
  - Programming code and data treated as an individual unit or component
- Objects are encapsulated
- Interface represents elements required for a source program to communicate with an object

Summary (cont'd.)

- Principle of information hiding
- Code, methods, attributes, and other information that make up an object
  - Organized using classes
- Instance
  - Object created from an existing class
- An object inherits the characteristics of the class on which it is based
- Date class contains methods and properties for manipulating the date and time

Summary (cont'd.)

- `Number` class contains methods for manipulating numbers and properties
- `Math` class contains methods and properties for performing mathematical calculations
- Can define custom object
  - object literal
- Can create template for custom objects
  - constructor function
- `this` keyword refers to object that called function
- `prototype` property specifies object's constructor