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.)

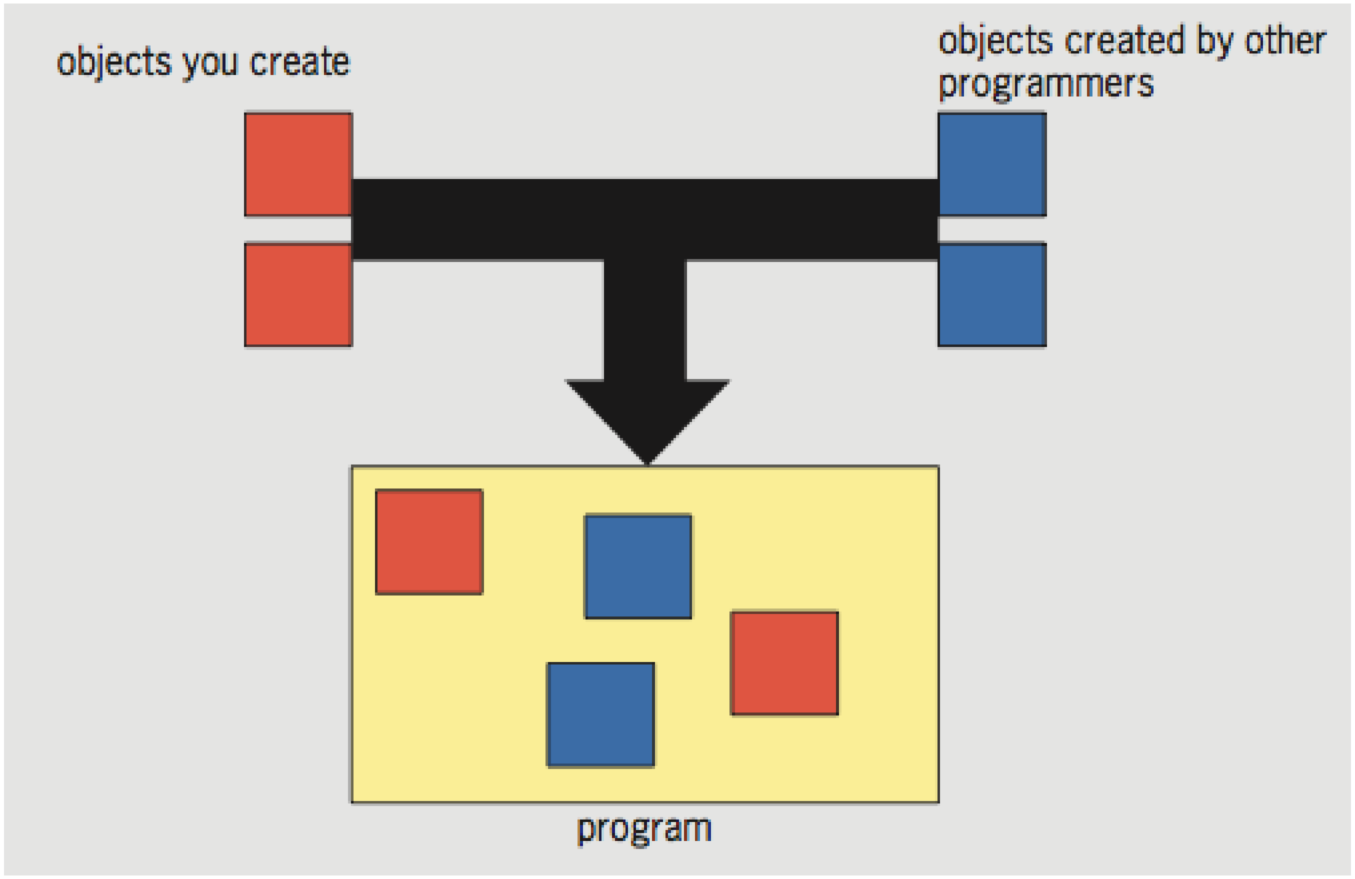


Figure 7-1 Programming with objects

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

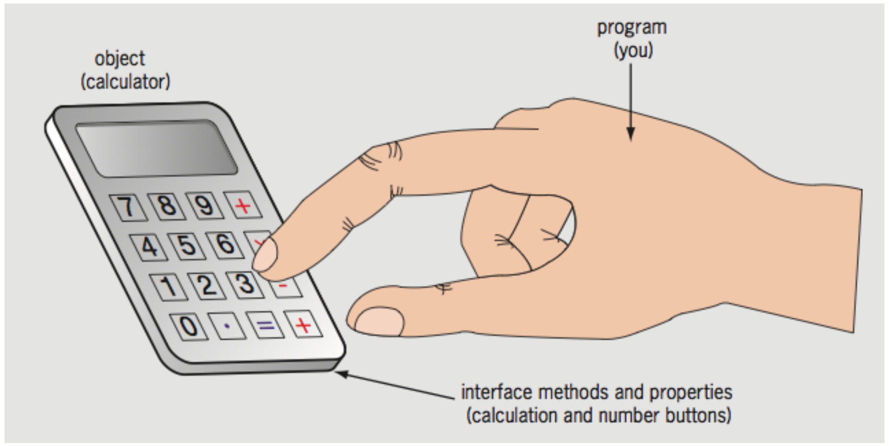

object
(calculator)
program
(you)
interface methods and properties
(calculation and number buttons)

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

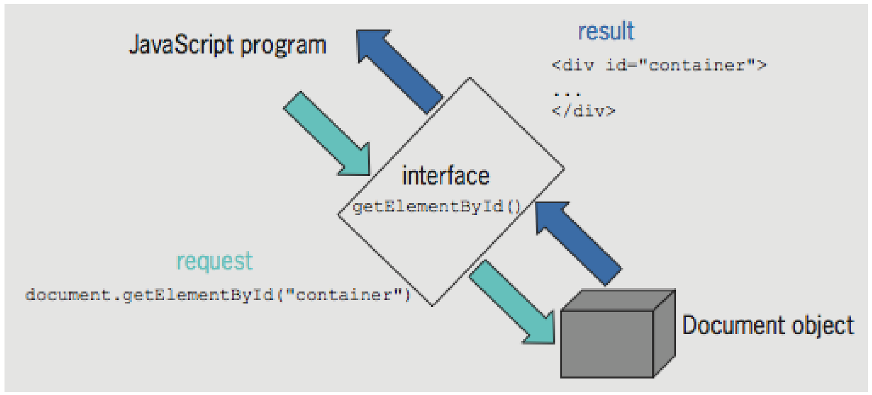
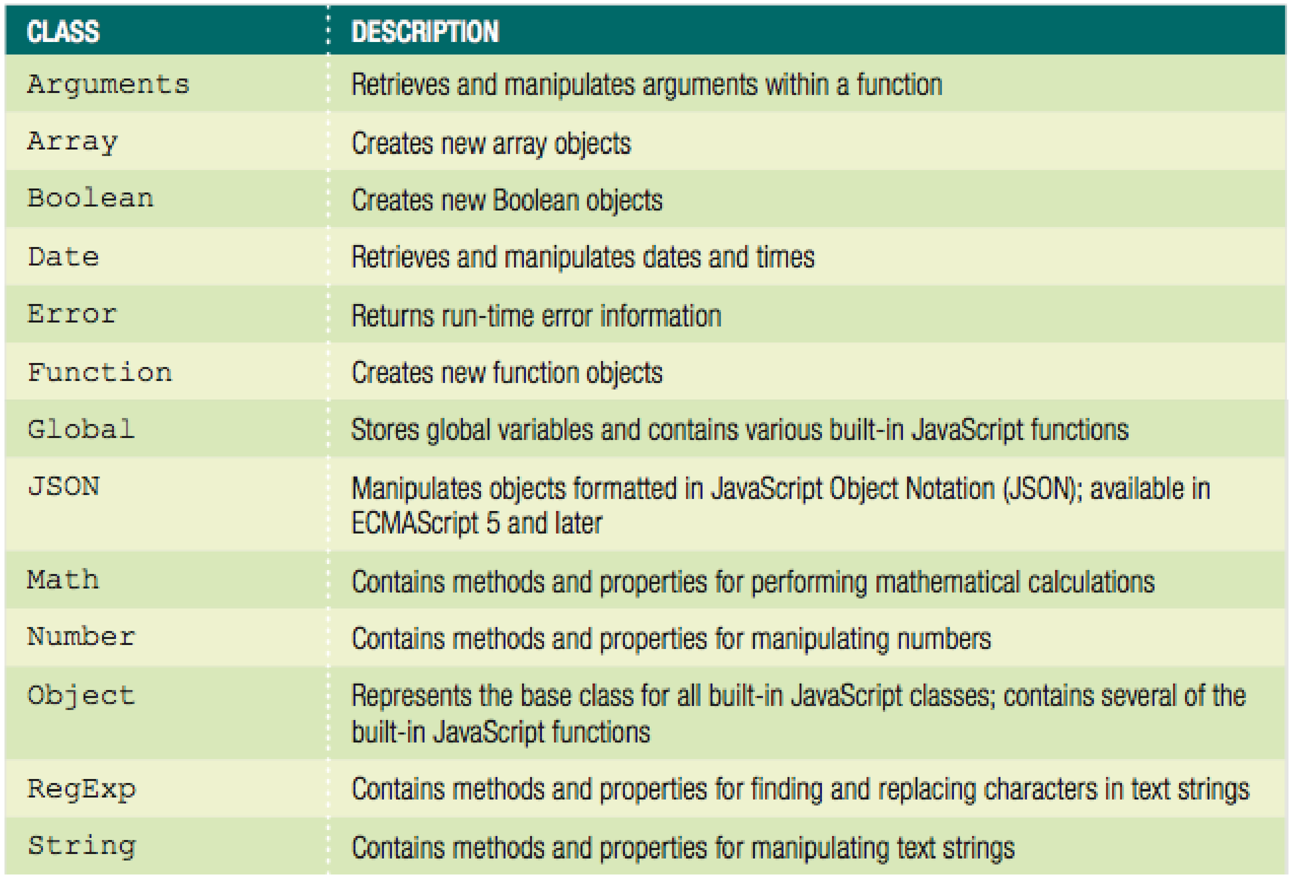


Figure 7-3 Using the interface for the `Document` object

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.)

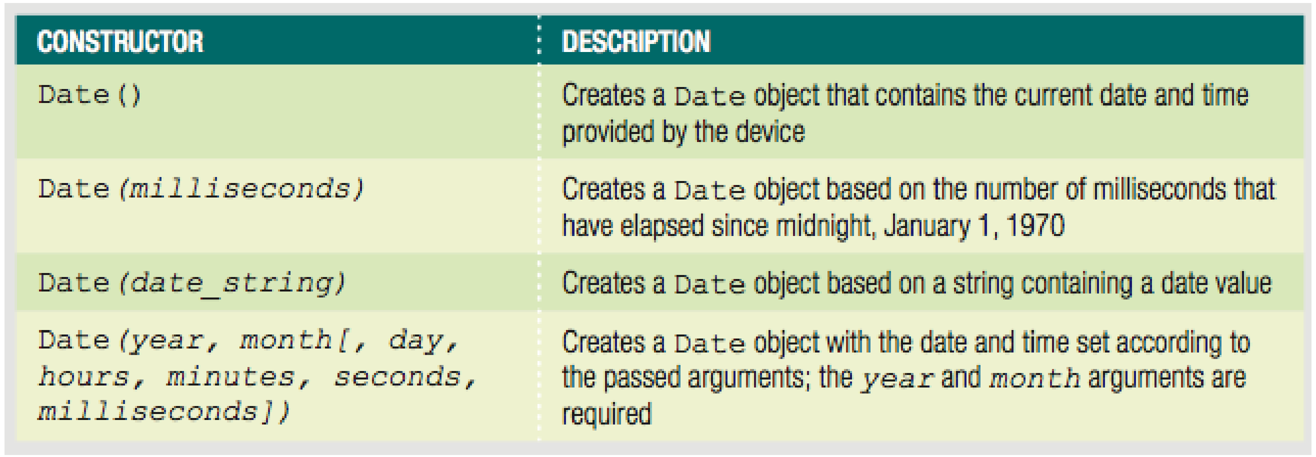
- Performing garbage collection
  - Garbage collection
    - Cleaning up, or reclaiming, memory reserved by a program
  - Declaring a variable or instantiating a new object
    - Reserves memory for the variable or object
  - JavaScript knows when a program no longer needs a variable or object
    -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

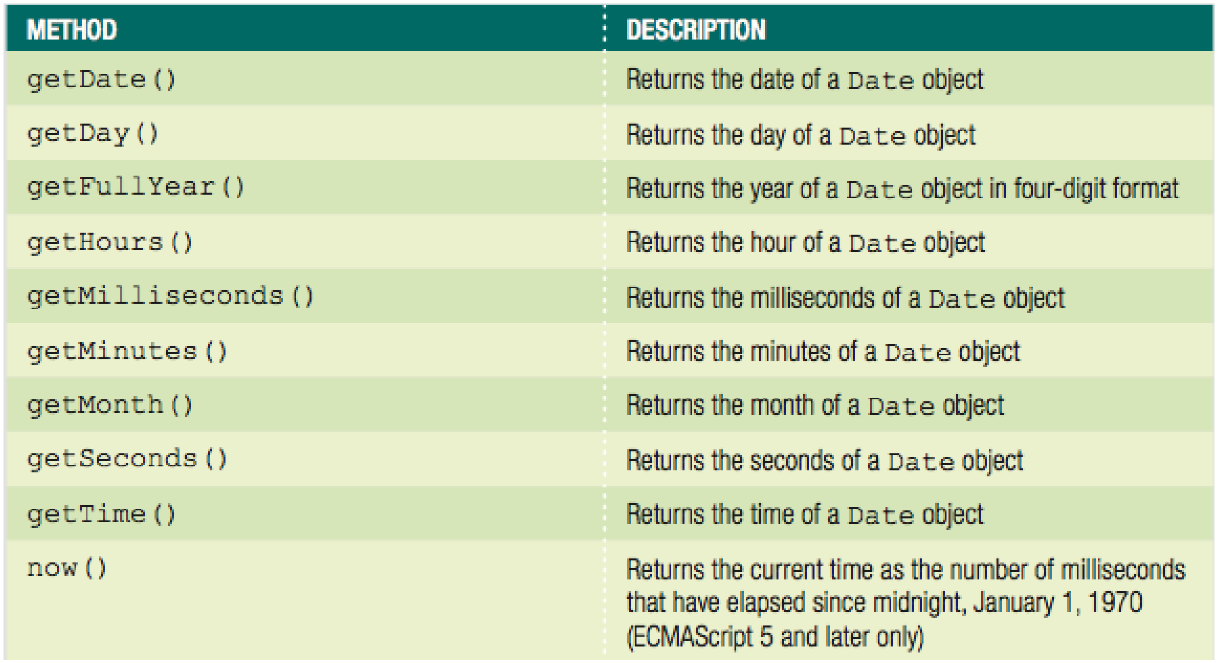
Manipulating the Date and Time with the `Date` Class (cont'd.)

- 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

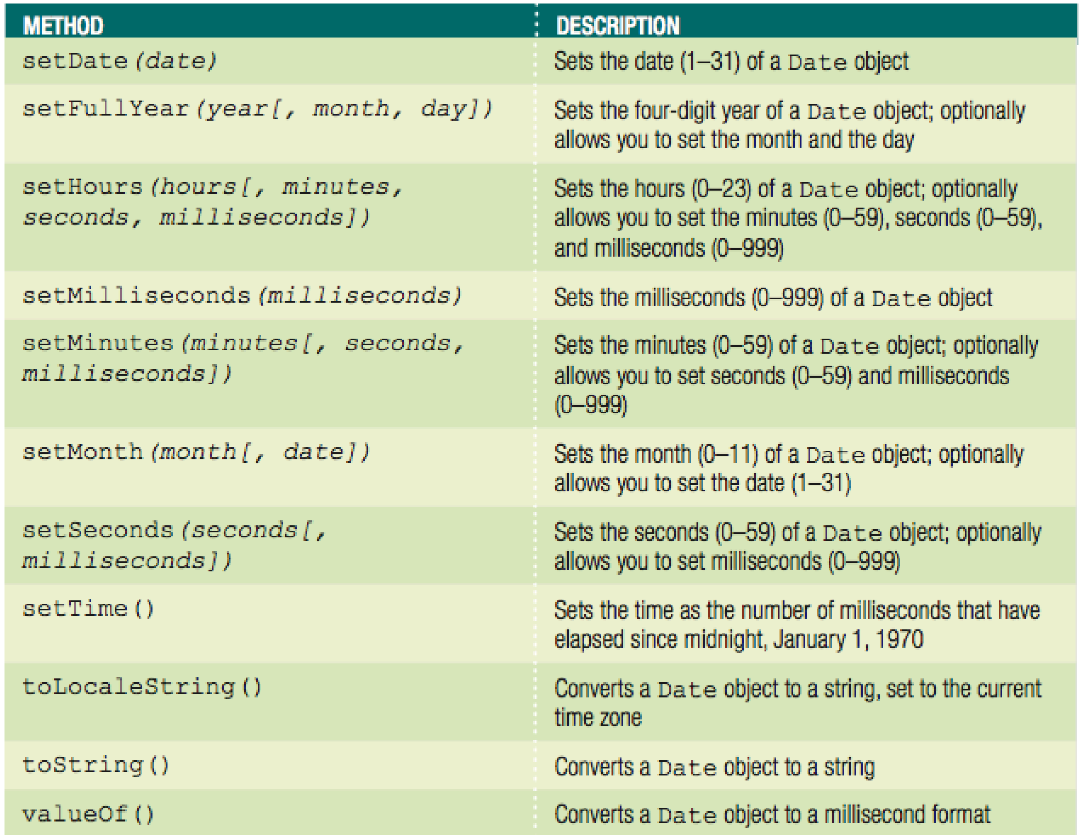
Manipulating the Date and Time with the `Date` Class (cont’d.)



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

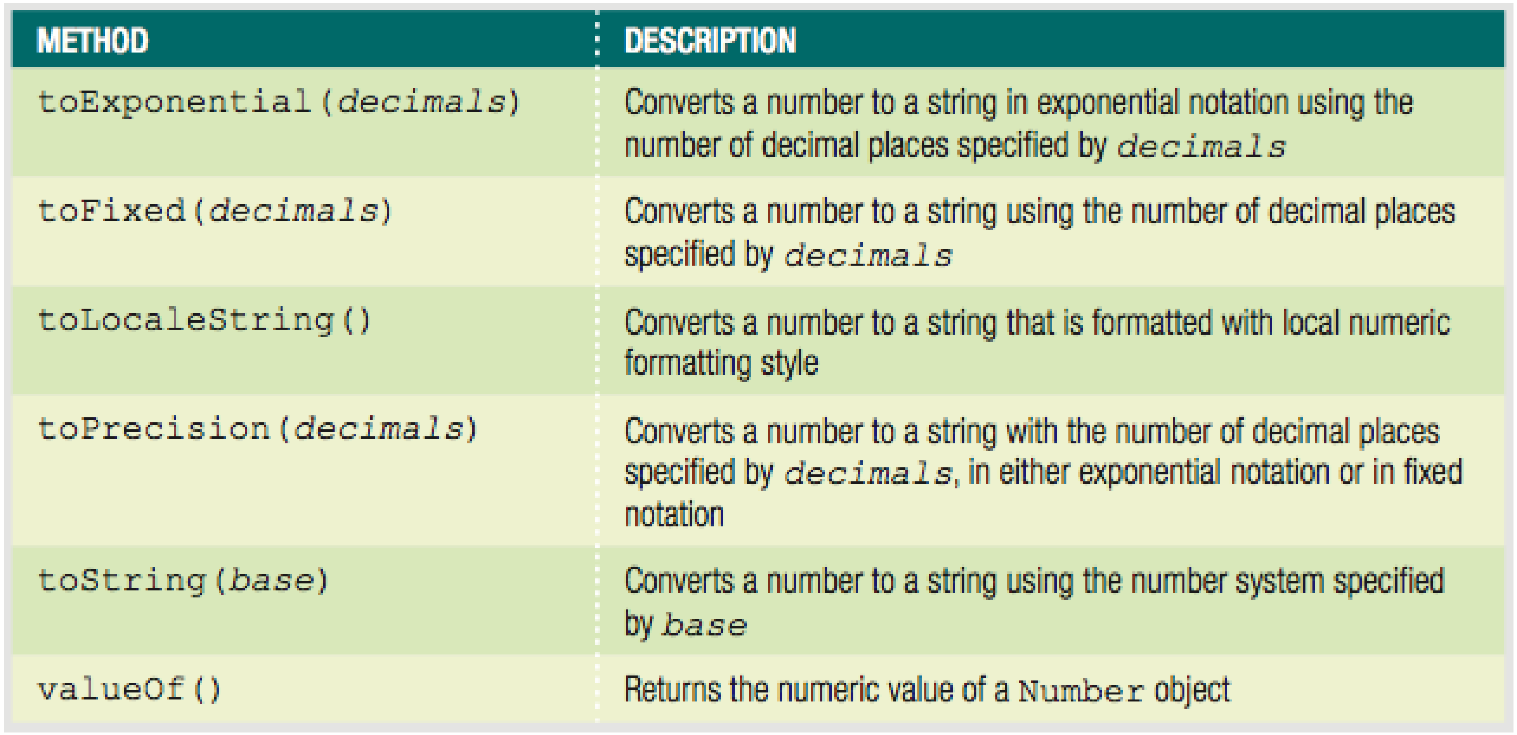
```
var today = new Date();
var months = ["January","February","March",↵
             "April","May","June",↵
             "July","August","September",↵
             "October","November","December"];
var curMonth = months[today.getMonth()];
```

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

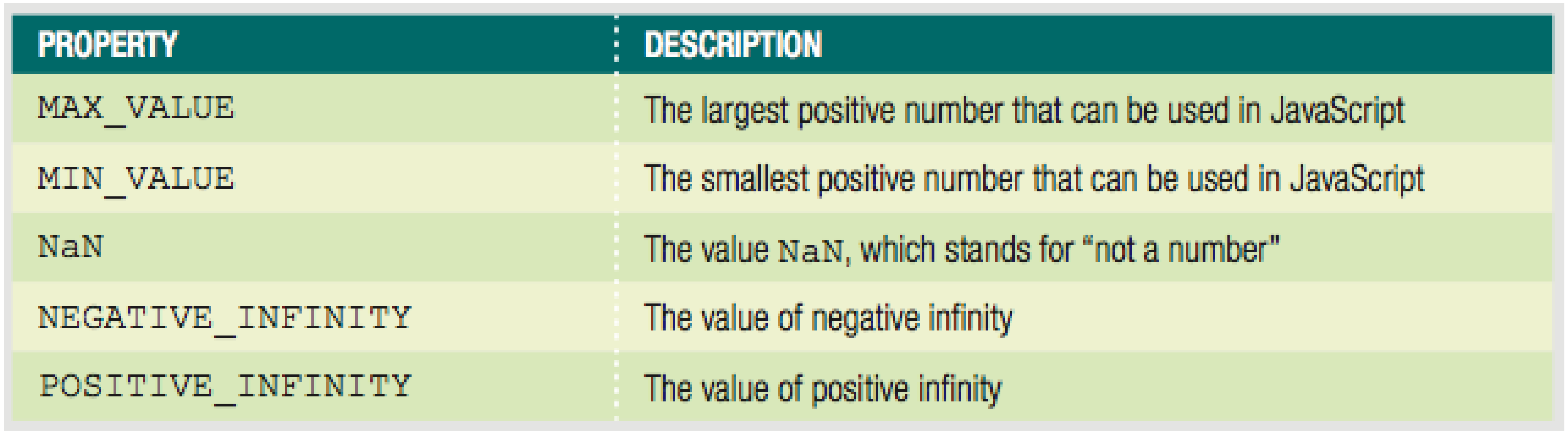
Table 7-4 `Number` class methods

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